Università degli Studi di Cagliari

DOTTORATO DI RICERCA

IN INGEGNERIA ELETTRONICA ED INFORMATICA

Ciclo XXVI

NOVEL LOAD IDENTIFICATION TECHNIQUES AND A STEADY STATE SELF-TUNING PROTOTYPE FOR SWITCHING MODE POWER SUPPLIES

Settore scientifico disciplinare di afferenza

ING-INF/01 (Elettronica)

Presentata da:	ANDREA CONGIU
Coordinatore Dottorato	PROF. FABIO ROLI
Tutor/Relatore	PROF. MASSIMO BARBARO

Esame finale anno accademico 2012 – 2013

University of Cagliari

PhD Course in

Electronic and Computer Engineering

Class XXVI

NOVEL LOAD IDENTIFICATION TECHNIQUES AND A STEADY STATE SELF-TUNING PROTOTYPE FOR SWITCHING MODE POWER SUPPLIES

Scientific field

ING-INF/01 (Elettronica)

Author:	ANDREA CONGIU
PhD Course Coordinator:	PROF. FABIO ROLI
Advisor:	PROF. MASSIMO BARBARO

Academic year of PhD defence 2012 – 2013

Abstract

Control of Switched Mode Power Supplies (SMPS) has been traditionally achieved through analog means with dedicated integrated circuits (ICs). However, as power systems are becoming increasingly complex, the classical concept of control has gradually evolved into the more general problem of power management, demanding functionalities that are hardly achievable in analog controllers. The high flexibility offered by digital controllers and their capability to implement sophisticated control strategies, together with the programmability of controller parameters, make digital control very attractive as an option for improving the features of dcdc converters. On the other side, digital controllers find their major weak point in the achievable dynamic performances of the closed loop system. Indeed, analog-to-digital conversion times, computational delays and sampling-related delays strongly limit the small signal closed loop bandwidth of a digitally controlled SMPS. Quantization effects set other severe constraints not known to analog solutions. For these reasons, intensive scientific research activity is addressing the problem of making digital compensator stronger competitors against their analog counterparts in terms of achievable performances.
In a wide range of applications, dcdc converters with high efficiency over the whole range of their load values are required. Integrated digital controllers for Switching Mode Power Supplies are gaining growing interest, since it has been shown the feasibility of digital controller ICs specifically developed for high frequency switching converters. One very interesting potential benefit is the use of autotuning of controller parameters (on-line controllers), so that the dynamic response can be set at the software level, independently of output capacitor filters, component variations and ageing. These kind of algorithms are able to identify the output filter configuration (system identification) and then automatically compute the best compensator gains to adjust system margins and bandwidth. In order to be an interesting solution, however, the self-tuning should satisfy two important requirements: it should not heavily affect converter operation under nominal condition and it should be based on a simple and robust algorithm whose complexity does not require a significant increase of the silicon area of the IC controller.
The first issue is avoided performing the system identification (SI) with the system open loop configuration, where perturbations can be induced in the system before the start up. Much more challenging is to satisfy this requirement during steady state operations, where perturbations on the output voltage are limited by the regular operations of the converter. The main advantage of steady state SI methods, is the detection of possible non-idealities occurring during the converter operations. In this way, the system dynamics can be consequently adjusted with the compensator parameters tuning. The resource saving issue, requires the development of äd-hocßelf-tuning techniques specifically tailored for integrated

digitally controlled converters.
Considering the flexibility of digital control, self-tuning algorithms can be studied and easily integrated at hardware level into closed loop SMPS reducing development time and R & D costs. The work of this dissertation finds its origin in this context. Smart power management is accomplished by tuning the controller parameters accordingly to the identified converter configuration.
The main difficult for self-tuning techniques is the identification of the converter output filter configuration. Two novel system identification techniques have been validated in this dissertation. The open loop SI method is based on the system step response, while dithering amplification effects are exploited for the steady state SI method. The open loop method can be used as autotunig approach during or before the system start up, a step evolving reference voltage has been used as system perturbation and to obtain the output filter information with the Power Spectral Density (PSD) computation of the system step response.
The use of $\Delta\Sigma$ modulator is largely increasing in digital control feedback. During the steady state, the finite resolution introduces quantization effects on the signal path causing low frequency contributes of the digital control word. Through oversampling-dithering capabilities of $\Delta\Sigma$ modulators, resolution improvements are obtained. The presented steady state identification techniques demonstrates that, amplifying the dithering effects on the signal path, the output filter information can be obtained on the digital side by processing with the PSD computation the perturbed output voltage. The amount of noise added on the output voltage does not affect the converter operations, mathematical considerations have been addressed and then justified both with a Matlab/Simulink fixed-point and a FPGA-based closed loop system.
The load output filter identification of both algorithms, refer to the frequency domain. When the respective perturbations occurs, the system response is observed on the digital side and processed with the PSD computation. The extracted parameters are the resonant frequency ans the possible ESR (Effective Series Resistance) contributes, which can be detected as maximum in the PSD output. The SI methods have been validated for different configurations of buck converters on a fixed-point closed loop model, however, they can be easily applied to further converter configurations. The steady state method has been successfully integrated into a FPGA-based prototype for digitally controlled buck converters, that integrates a PSD computer needed for the load parameters identification.
At this purpose, a novel VHDL-coded full-scalable hybrid processor for Constant Geometry FFT (CG-FFT) computation has been designed and integrated into the PSD computation system. The processor is based on a variation of the conventional algorithm used for FFT, which is the Constant-Geometry FFT (CG-FFT). Hybrid CORDIC-LUT scalable architectures, has been introduced as alternative approach for the twiddle factors (phase factors) computation needed during the FFT algorithms execution. The shared core architecture uses a single phase rotator to satisfy all TF requests. It can achieve improved logic saving by trading off with computational speed. The pipelined architecture is composed of a number of stages equal to the number of PEs and achieves the highest possible throughput, at the expense of more hardware usage.

Introduction

Consequently at the diffusion of embedded systems, requirements as area optimization, very high precision regulated voltage, power efficiency and minimization of external components become the main goal during the converter design. Dedicated complex analog solutions have been exploited for applications of portable instruments where the efficiency (e.g. very low ripple requirements) has to be kept as high as possible over a wide range of loads. The growing regulators complexity and the power management, have highlighted how digital solutions can be considered especially when resource saving become the main aspect of the design. The high programmability level and the computational power, both joined with the possibility to have complex control solutions have pushed the research on digitally controlled SMPS. An overview of recent developments in digital control strategies for dcdc switching power converters is presented in[54].
Since 2000 the research approached to digital solution for SMPS control [64, 118, 55, 93]. During the recent years the research community has been much more interested in power and area optimization for digital ICs dedicated to the SMPS control. Furthermore, most of research moved to algorithms integration for supporting the digital control and its quality (e.g. self-tuning techniques). Even if lots of improvements have been done, actually only about the one third of on market SMPS are digitally controlled. However, interest and know-how are both growing more and more day by day. During last fifteen years, digital solution for SMPS has been strongly studied. Most of studies has been focused on DPWM-based control improvements and digital quantization limits. In [64, 118, 55, 93, 119, 56, 82, 87, 109, 120, 116, 133, 19, 128, 95, 45, 84, 134, 111, 110, 127, 77, 15] digital control improvements for SMPS has been addressed. Limit Cycle Oscillations (LCOs) contributes, caused by interfacing analog-to-digital converters (ADC) and the digital pulse-width modulator (DPWM) has been addressed in [85] [86], where quantization effects has been modelled. In [63] an exact small signal discrete-time model for digitally controlled dcdc converters has been introduced. The model, which is based on well-known approaches to discrete-time modelling and the standard Z-transform, takes into account modulator effects and delays in the control loop. Quantization noise effect in DPWM-based regulators represents the main limit of this approach. Because dithering-oversampling-averaging capabilities, Delta-Sigma ($\Delta\Sigma$) devices are good candidate for resolution increasings [65, 61, 37, 51, 138, 76]. In [76] the impact of the Noise Transfer Function (NTF) was studied, a third order structure should be choose to avoid idle tones effects. A $\Delta\Sigma$-DPWM controller for switching converters was designed and implemented in [51] and the DPWM low resolution increasing was confirmed also in [138], where undesirable low-frequency LCOs are eliminated and tight output voltage regulation is obtained.

Feasibility of completely integrated digital controllers was demonstrated for the first time in [82][87][120], where innovative solutions for the main constituents of a digital controller, namely the compensator, the A/D converter and the digital pulse-width modulator, were presented. Based on a look-up table structure, the PID compensator employed in [82] presented reduced complexity. Delay-line and windowed ADCs were used in these works for fast conversion times and small area requirements. A ring oscillator-MUX DPWM was implemented in [87], while in [82] a hybrid counter/delay line architecture was considered as a suitable trade off between resolution, area and power consumption. Further works in the area exploited ring-oscillator ADCs [120][133].
Proportional Integrative Derivative (PID) compensator has been strongly studied and integrated in digital control loop for dcdc converters. A high frequency control solution with a PID regulator based on look-up tables was proposed in [92]. In [106], a model reference tuning for PID regulators of dcdc converters was created thanks to crossover frequency analysis, while A non-linear PID-like regulator has been proposed in [126].
Along with solutions aimed to an increasingly deeper integration, the research activity also focused on exporting control approaches widespread in the analog world into the digital domain, an example being the investigation and development of digital current-mode control techniques [19, 84]. In [19] a predictive model was introduced, a digital current-mode controller for dcdc converters FPGA-based prototype was designed in [84].
Further improvements for digital control are often based on digital signal processing algorithms, which can be easily designed and integrated. A continuous time digital signal processing is used in [140, 137] to achieve very fast transient response. The continuous time DSP performs also a charge-balance based algorithm to achieve voltage recovery through a single on-off action of the power switch. A Middlebrook's loop-gain measurement technique for on-line stability margin monitoring in digitally controlled SMPS has been exploited in [74].

The digital approach enables smart solutions for regulators, e.g. self-tuning or autotuning capabilities of a regulator permits to automatically adapt regulator gains over a specific power plant (on-line controllers). System frequency response can be tuned by adjusting PID gains [106]. One of the most intense and interesting research activities concerns the development and implementation of hardware-effective autotuning techniques for digital compensator [25, 102, 104, 135, 136, 139, 66, 67, 14, 13, 103, 101, 97, 9, 50, 48, 17, 4, 5, 72, 90, 16, 60, 59, 26, 105, 73, 47, 2].

Off-line digitally control feedback for SMPS, presents a static configuration of the compensator. The off-line controllers do not allow the best system bandwidth for a large set of loads (converter configurations), compensator parameters can be stored in a non-volatile memory and loaded in a programmable controller at system power-on. In this way, different sets of pre-calculated parameters can be run for many environmental conditions on the same control hardware. Joining the feasibility to identify and/or monitor the dcdc output filter (system identification), with the dynamic setting of the compensator gains, avoids the duty to optimize the compensator. More evolved tuning algorithms literally perform an automatic design of the compensator parameters through a number of *on-line* measurements and post-processing operations (on-line controllers).
The system identification (SI) generally falls into two main categories: parametric and non-parametric methods [57, 58]. Parametric methods return the parameters of the system model such as the coefficients of a system difference equation, transfer function, or state-space

model. Non-parametric methods return impulse response and/or frequency response data directly.
Parametric methods require, in addition to the selection of an appropriate input stimulus, an a priori selection of a parametrized model structure including system order and number of zeros, construction of a suitable prediction error equation and loss function, and methods to minimize the loss function [9, 50, 17, 4, 5, 72, 90, 16, 60, 59, 26, 105, 73, 47]. Non-parametric methods do not assume a system model and require only selection of an appropriate stimulus. The LCO-based self-tuning methods are an example of nonparametric approaches. System LCOs can be induced either by using a relay [25, 102, 104] or reducing the DPWM resolution [135, 136, 139]. However, this kind of approach allows to know the frequency loop response only at the stimulated frequency. Another non-parametric method is focused on obtaining an approximation of the system impulsive response, that is calculated exploiting the cross correlation method. For a white noise input the impulse response of the system is proportional to the cross correlation between input and output, while the correlation itself rejects any disturbances to the system as long as they are uncorrelated with the input [57, 58, 66, 67, 14, 13, 103, 101, 97]. In these approaches a Pseudo Random Binary Sequence(PRBS)is used to emulate a digital white noise source and perturb the closed loop system. Such identification method has been integrated into auto-regulated digital converters, with different control techniques.
Beside purely digital control solutions, mixed-signal controllers are worth to be mentioned [112, 96, 113, 98]. These approaches combine simple integrated analog blocks and digital provisions with the aim to achieve analog-like dynamic performances, but still retaining the advantages of digital systems like programmability, low passive component count and control robustness.
Self-Tuning algorithms are mainly composed by processing approaches based on frequency domain analysis [66, 67, 14, 13, 103, 101, 97]. The FFT and its hardware implementation has been largely exploited and wide range of custom FFT processor architectures can be found in literature. Pipelined structures have both high throughput and computational efficiency, but when area constraints are stronger than timing, non-pipelined counterparts are usually preferred. Scalable architectures in terms of number of processing elements (PEs), are flexible solutions. Astola and Akopian [8] introduced a family of hardware-oriented algorithms resulting in scalable constant geometry (CG) structures. In [107] a not-in-place architecture targeted for ASIC implementation is proposed. This solution uses data shuffling registers, thus it does not require a memory for intermediate results.
Classic approaches to FFT designs require a large amount of memory for storing precomputed twiddle factors. The CORDIC iterative algorithm presented in [115] allows computation of twiddle factors at runtime. To this purpose many architectures introduce this algorithm inside the PEs by substituting the complex multipliers with iterative phase rotations. With this approach, in [130] the number of iterations has been reduced by using optimized sequences and corresponding scale factors both stored in a LUT. In [122] a multi-bank RAM structure to reduce memory logic is presented. On the other hand, some systems replace the twiddle factor storage ROM with a CORDIC-based generation system. Nonethless, CORDIC hardware implementations can be very expensive in terms of area usage [11, 34], consequently hardly suitable for a scalable FFT approach.

The entire research has been focused on Innovation Project 2010, shared between the EOLAB group of the University of Cagliari and the Automotive Department of Infineon Tech-

nologies AG (Villach-Austria). Aim of this activity is to develop and execute an innovative self-tuning prototype (on-line controller) for future automation technologies to allow the integration of power supplies for different set of applications reducing development time and R & D costs.
Off-line controllers for Switching Mode Power Supplies (SMPS) need worst case considerations on stability margins and closed loop bandwidth. Load characteristics are unknown at the design stage and the converter is tested with a limited range of loads. On-line controllers for SMPS, are based on self-tuning algorithms and adjust the system dynamics resulting in optimal design over the full range of loads. The self-tuning techniques are often two-step evolving approach: first the SI identify the output filter structure of the converter, then, during the regulation step the compensator is set in order to satisfy both system margins and bandwidth. Non-parametric SI techniques, are based on appropriate stimulus to extract load information. These kind of approaches can be distinguished in two main categories:

- Steady state system identification technique.
- Open loop system identification technique.

Non-parametric steady state SI techniques, have the constraint to inject very little perturbations and permit to monitor load variation during the steady state. When non-idealities, like temperature variations occur during the steady state, the ESR contribution in the load capacitance increases. In this condition, the system can have poor stability margins due to zero contribution inserted by the ESR. Then, it is fundamental to study the converter control to output transfer function (identification process) and consequently adapt the PID gains in order to increase margins and bandwidth. A steady state PRBS-based approach has been presented in [101].
Non-parametric open loop identification techniques, can introduce a bigger perturbation in the system because they do not operates during nominal converter operations. They can be applied before the system start up. Once the algorithm is performed best PID gain can be computed before closed loop operations, where load parameter variation can be observed with the steady state identification technique.
In this dissertation two novel techniques of load identification has been introduced and validated, one refers to the steady state SI technique and the second to the open loop SI technique. Both techniques are focused on the load parameters extraction in the frequency domain. Essentially, they extract the output filter resonant frequency (f_0) and the zero frequency (f_z) due to possible the ESR contribution. Even if the load identification is completely different, the introduced SI techniques are both focused the same load parameters. The system response during the respective perturbations, can be recorded in the digital domain (ADC output) and processed with Power Spectral Density (PSD) analysis. The resonant frequency and the ESR contribution can be extracted, in both cases, from the PSD output by maximum searching analysis. The resonant frequency is detect as maximum in the PSD output, while the ESR contribution is obtained as second peak in the processing output. Both SI approaches have been patented.

The steady state SI technique has been performed through the dithering amplification. In this approach a $\Delta\Sigma$ modulator is used to oversample the digital control word and amplify the considered quantization noise. Consequently, faster variations of the digital control word are obtained and automatically averaged by the low-pass configuration of the output filter. The amplified quantization noise can be considered as independent addictive input

for the closed loop configuration, its effects are reflected on the output voltage through the digital control. Thanks to the further amount of noise added on the entire range of frequencies, the resonant frequency f_0 can be easily detected. For the f_z identification in the PSD, a particular structure of noise shaper has been used. Inserting a notch contribution at the resonant frequency, the amplified dithering effects are more concentrate on frequency next to f_0. In this way, the ESR contribution can be detect as a second peak when a change of slope occurs in the PSD output. Perturbations considered for this non-parametric method have been demonstrated through a mathematical model. For considered system configuration ($70MHz$ clock frequency and $449kHz$ switching frequency), about $100mV$ of average perturbation on the output voltage is introduced when dithering effects have been doubled. The open loop identification technique is based on the system step response. In this approach a step evolving reference voltage has been considered as system perturbation. The system response is observed and processed when steps occur.

In this dissertation, a digitally controlled buck converter has been prototyped (off-line controller). A fixed-point Matlab/Simulink closed loop system has been modelled to define the needed hardware resolution. Comparing fixed with floating point closed loop models, the hardware resolution has been tuned and used during the VHDL design of the control feedback. To validate the VHDL-coded control loop a mixed signal hardware-software (VHDL-Matlab) FPGA-based co-simulation model for PID-based SMPS has been realized. Co-simulation modelling with Xilinx System Generator is one of the actual State of Art approach used for mixed-signal models verification. The closed loop prototype has been obtained by interfacing the interfacing a Test-Chip (TC) provided by Infineon Technologies AG with a Virtex-6 FPGA where the VHDL-coded digital control loop has been coded.
The SI algorithms have been first mathematically described and then verified into the fixed-point model, for different buck converter configurations. The steady state identification algorithm has been also validated into the FPGA-TC prototype, where the dithering amplification has been verified with Matlab processing of the hardware ADC output. As last step, an on-line controller working during the steady state has been prototyped. The steady state SI has been used, to realise this self-tuning prototype for SMPS, in this closed loop implementation a VHDL-coded PSD computer has been integrated as processing unit.

As the aim is to obtain a self-tuning platform, the PSD computer has been designed considering scalability constraints.
The PSD computation algorithm in based on the FFT, in this dissertation the design and hardware implementation of a scalable processor for the computation of the Fast Fourier Transform (FFT) is discussed.
The processor is based on a variation of the conventional algorithm used for FFT, which is the Constant-Geometry FFT (CG-FFT). By performing a perfect shuffle permutation at the end of every computational stage, the aforementioned algorithm allows to implement a simple structure composed of parallel Processing Elements (PEs) and an interconnection network. The number of PEs can be tuned according to the requirements of computational speed and hardware usage. Each PE implements the basic operation of the FFT, which is called butterfly, and involves the complex multiplication for phase factors. These are called Twiddle Factors (TFs), and in conventional approaches they are retrieved from a ROM. Other solutions involve the usage of a system that computes TFs on-the-fly by micro-rotations, called CORDIC. In this dissertation, a novel algorithm for TF computation that takes scalability into consideration is derived. It either calculates each twiddle factor from a previous one, or it retrieves it from a set of $log_2 N-1$ basic TFs, where N is the length of the FFT. This is imple-

mented in two twiddle factors generator architectures, called shared core and pipelined [24]. These hardware structures compute twiddle factors both by retrieving them from a ROM and by phase rotations. Hence they are called hybrid CORDIC-LUT scalable archtiectures.
The shared core architecture uses a single phase rotator to satisfy all TF requests. It can achieve improved logic saving by trading off with computational speed. The pipelined architecture is composed of a number of stages equal to the number of PEs and achieves the highest possible throughput, at the expense of more hardware usage.
The architecture of the processor is fully scalable in terms of transform length, data bits and Processing Elements (PEs). Each PE implements the basic operation of the FFT, and all the PEs operate in parallel. A processor with many PEs will take less clock cycles to complete an FFT computation of a given length than a processor with few PEs, but will occupy more FPGA resources. This tradeoff makes the number of PEs a degree of freedom in the DSP system design.

This dissertation is organised as follows:

- **Chapter 1.** In this chapter a brief theoretical introduction to dcdc power supplies has been presented referring to buck converters. The ideal converter has been introduced and main characteristic waveforms has been shown. Distinction between CC and DC conduction modes have been addressed as well as the losses in the converter. Consideration about converter design specifications have been shown in order to be taken into account during the dissertation.

- **Chapter 2.** In this chapter, the ac characterisation of the buck converter in CCM has been presented. Main techniques for the ac small signal characterization are discussed, and the frequency behaviour of open loop converter is presented with consideration about ESR effects. A Matlab/Simulink open loop model has been presented and validate though simulations results.

- **Chapter 3.** In this chapter, the closed loop configuration for digitally controlled buck converters has been discussed. Firstly, delays effects introduced by the digital control loop have been characterised in the frequency domain, then design specifications have been addressed. Considerations about PID compensator design and their effect in the system bandwidth have been shown. Both AD and DA quantization effects are largely discussed and, the causes of LCOs during the steady state are highlighted. The last sections refers to $\Delta\Sigma$ description. The main characteristics of the modulator are highlighted considering the error feedback consideration.

- **Chapter 4.** In this chapter the entire discussion have been addressed in order to design a FPGA-based prototype of the digital control loop. First a fixed-point Matlab/Simulink model has been realised through comparison with the floating point solutions. Once resolution consideration about the digital control loop have been drawn, a VHDL design has been described. Closed loop hardware verifications have been shown with mixed signal hardware-software (VHDL-Matlab) FPGA-based co-simulations.

- **Chapter 5.** In this chapter a closed loop prototype, or off-line controller, for digitally controlled buck converters is realised exploiting the digital control loop prototyped in the previous section. The closed loop implementation has been realised interfacing the FPGA with a Test Chip (TC). I/O configuration for the FPGA is discussed together

with the external loop configuration for the TC. Closed loop results have been presented with oscilloscope-based measurements for the robust prototype.

- **Chapter 6.** In this chapter main load identification techniques for on-line controllers have been shown distinguishing between parametric and non-parametric approaches at the State of Art. Some non-parametric SI approaches have been modelled in the Matlab/Simulink closed loop floating point model.

- **Chapter 7.** In this chapter two novel non-parametric SI techniques have been presented. Mathematical considerations have been drawn. Load identification results have been shown through the PSD computation for different buck converter configurations.

- **Chapter 8.** Discussion in this chapter has been done with the aim of obtaining a self-tuning prototype or on-line controller for SMPS. The steady state SI technique has been first validated at hardware level, and then integrated into the off-line controller to realise a self tuning prototype. Identification results have been shown for different buck converter configurations through the Chipscope Pro tool interface, together with the regulation step.

- **Chapter 9.** In this chapter considerations have been addressed up the hardware characterisation of a full-scalable PSD computer. First a scalable FFT architecture has been designed and then a hybrid CORDIC-LUT algorithm for twiddle factors generation is introduced. A full scalable hybrid FFT processor is designed and integrated into the PSD computer.

Contents

Abstract i

1 A general overview of buck converters 1
- 1.1 Ideal Model . . . 1
- 1.2 Continuous Conduction Mode . . . 3
- 1.3 Discontinuous Conduction Mode . . . 5
- 1.4 Power Losses . . . 7
- 1.5 Output filter design considerations . . . 8

2 Buck Converter in Continuous Conduction Mode 9
- 2.1 AC small signal model . . . 9
  - 2.1.1 Transfer functions . . . 11
- 2.2 Matlab modelling . . . 15
- 2.3 Matlab/Simulink open loop model . . . 18

3 Digitally controlled buck converters in CCM 23
- 3.1 The digital control feedback . . . 23
  - 3.1.1 Delays in the digital loop . . . 25
  - 3.1.2 The compensator . . . 27
- 3.2 Design considerations . . . 29
  - 3.2.1 PID Compensator . . . 30
- 3.3 Digital quantization effects . . . 32
  - 3.3.1 ADC and DPWM resolution . . . 32
  - 3.3.2 Compensator resolution . . . 33
  - 3.3.3 Limit cycles oscillations . . . 33
- 3.4 Delta Sigma modulator . . . 34
  - 3.4.1 Error feedback configuration . . . 36
  - 3.4.2 Noise transfer function . . . 37

4 Digital control feedback prototype 39
- 4.1 Matlab/Simulink closed loop model . . . 39
  - 4.1.1 ADC modelling . . . 42
  - 4.1.2 PID compensator modelling . . . 44
  - 4.1.3 Delta Sigma modulator modelling . . . 46
  - 4.1.4 DPWM modelling . . . 48

4.1.5 PID tuning tool . . . 50
4.2 VHDL-coded digital control feedback . . . 51
4.2.1 PID Compensator design . . . 53
4.2.2 Delta Sigma modulator design . . . 54
4.2.3 DPWM design . . . 57
4.3 FPGA-based co-simulation closed loop system . . . 58

5 Digitally controlled buck converter prototype: Off-line controller 65
5.1 FPGA configuration . . . 69
5.1.1 Clock generator . . . 70
5.1.2 Observing FPGA internal signals . . . 71
5.1.3 FPGA I/O mapping . . . 72
5.2 Test Chip external loop configuration . . . 74

6 Modelling of system identification techniques at the State of Art 77
6.1 Non-parametric methods . . . 79
6.1.1 LCO-based methods . . . 79
6.1.2 Harmonic response analysis . . . 82
6.1.3 PRBS-based methods . . . 84
6.2 Parametric methods . . . 85

7 Novel non-parametric system identification methods 87
7.1 Open loop system identification technique . . . 88
7.1.1 Resonant frequency identification results . . . 90
7.1.2 ESR effects identification results . . . 92
7.2 Steady state system identification technique . . . 97
7.2.1 Resonant frequency and ESR identification algorithm . . . 98
7.2.2 Resonant frequency identification results . . . 109
7.2.3 ESR effects identification results . . . 113
7.3 Processing and extraction algorithm design specifications . . . 125

8 Steady state self-tuning prototype: On-line controller 129
8.1 Steady state system identification method validation . . . 129
8.1.1 Results . . . 132
8.2 Self-tuning prototype . . . 138
8.2.1 Hardware description . . . 140
8.2.2 Results . . . 145

9 Scalable FFT and autocorrelation-based HDL processor 159
9.1 Scalable FFT architecture . . . 161
9.1.1 Description of the implemented architecture . . . 161
9.1.2 RTL design . . . 166
9.2 TF module general structure and LUT-based architecture . . . 178
9.2.1 Inside *twiddle_factors_system* . . . 178
9.2.2 LUT-based TF generation . . . 178
9.3 Hybrid CORDIC-LUT twiddle factor generator . . . 181
9.3.1 Properties of the CG-FFT . . . 181

9.3.2 Overall architectural remarks . . . 185
9.3.3 Hybrid CORDIC-LUT architecture *shared-core* . . . 187
9.3.4 Hybrid CORDIC-LUT architecture *pipelined* . . . 193
9.4 PSD computer . . . 195
9.4.1 Autocorrelation-based DSP algorithm: PSD computation . . . 195
9.4.2 System architecture . . . 196
9.4.3 Frequency feature extraction . . . 208
9.5 Testing and simulation environment . . . 208
9.5.1 VHDL testbenches . . . 208
9.5.2 MATLAB simulation scripts . . . 210
9.6 Simulation and synthesis results . . . 212
9.6.1 Hybrid twiddle factors generators . . . 212
9.6.2 Scalable FFT processor . . . 224
9.6.3 PSD computer . . . 232

10 Conclusions **237**

Bibliography **243**

A Quick configuration guide **261**
A.1 Configuring the autocorrelation system . . . 261
A.1.1 Setting the parameters . . . 261
A.2 Selecting the architecture of the RAM . . . 261
A.3 Configuring the FFT processor . . . 262
A.3.1 Setting the parameters . . . 262
A.3.2 Selecting the twiddle factor generation architecture . . . 262
A.3.3 Initialising the ROM . . . 262

B Background mathematical concepts **265**
B.1 The Fast Fourier Transform . . . 265
B.2 The CORDIC algorithm . . . 269
B.3 Theorical remarks on the autocorrelation function . . . 272

C State of Art in FFT hardware architectures **277**
C.1 Pipelined structures . . . 277
C.2 Memory-based structures . . . 280
C.3 FFT-specific CORDIC . . . 290

List of Figures

1.1 Buck converter: ideal model. . . . 1
1.2 Buck converter: switching node voltage waveform. . . . 2
1.3 Buck converter: closed loop buck configuration. . . . 3
1.4 Buck converter: Open loop configuration. . . . 4
1.5 Buck converter: inductor voltage waveform. . . . 4
1.6 Buck converter: Inductor current waveform. . . . 4
1.7 Buck converter: asynchronous control configuration. . . . 6
1.8 Buck converter: waveforms in DCM mode. . . . 7

2.1 Linearisation approach for buck converters. . . . 10
2.2 Averaged method analysis: model. . . . 12
2.3 Averaged method analysis: buck converter circuit. . . . 12
2.4 Small signal circuit: ideal buck converter. . . . 13
2.5 Small signal circuit : buck converter with ESR contribution. . . . 13
2.6 Control to Output transfer function: ideal buck converters. . . . 15
2.7 Control to Output transfer function: buck converters with ESR contribution. . . . 16
2.8 Equivalent circuit: buck converter with non-idealities. . . . 16
2.9 Matlab function: buck converter with non-idealities. . . . 17
2.10 Matlab/Simulink model: open loop buck converter. . . . 18
2.11 Simulink user's defined block. . . . 19
2.12 Matlab function: gate driver. . . . 19
2.13 Simulink: Fixed-step simulations. . . . 20
2.14 Matlab/Simulink: open loop output voltage ($D = 0.5$, $V_{in} = 14V$). . . . 20
2.15 Matlab/Simulink: open loop zoomed output voltage ($D = 0.5$, $V_{in} = 14V$). . . . 21

3.1 Digitally controlled buck converter: model. . . . 24
3.2 Control to output transfer function: loop delay effects. . . . 26
3.3 DPWM waveforms. . . . 27
3.4 ADC output characteristic: zero error bin. . . . 28
3.5 Control to output transfer function: PID effects. . . . 31
3.6 PID model: parallel structure. . . . 32
3.7 Digitally controlled buck converter: model with $\Delta\Sigma$ modulator. . . . 35
3.8 $\Delta\Sigma$ modulator: error feedback configuration. . . . 35
3.9 $\Delta\Sigma$ modulator: 2nd order error feedback configuration. . . . 37
3.10 Control to output and noise transfer functions. . . . 38

4.1 Digitally controlled buck converter: Matlab/Simulink model. . . . 40
4.2 Fixed-point closed loop model: output voltage. . . . 41
4.3 Fixed-point closed loop model: zoomed output voltage. . . . 41
4.4 Simulink block: ADC model. . . . 42
4.5 ADC Matlab function: floating point (left side) and fixed-point (right side). . . . 43
4.6 Matlab/Simulink: fixed-point ADC output characteristic. . . . 44
4.7 PID Matlab function: floating point (left side) and fixed-point (right side). . . . 45
4.8 Matlab/Simulink: ΔΣ floating and fixed-point output comparison. . . . 46
4.9 Simulink ΔΣ blocks: floating and fixed-point comparison. . . . 47
4.10 ΔΣ Matlab function: fixed-point. . . . 48
4.11 DPWM waveforms. . . . 49
4.12 Matlab/Simulink floating and fixed-point comparison: model. . . . 49
4.13 Matlab/Simulink floating and fixed-point comparison: Digital resolution tuning. . . . 50
4.14 PID gains tuning tool. . . . 51
4.15 Digital control feedback: open loop hardware implementation. . . . 52
4.16 VHDL implementation: PID proportional part. . . . 54
4.17 VHDL implementation: ΔΣ dithering function. . . . 55
4.18 VHDL implementation: ΔΣ noise shaper. . . . 56
4.19 VHDL implementation: ΔΣ noise shaper with $2kHz$ notch. . . . 57
4.20 VHDL/Matlab co-simulation closed loop: model. . . . 59
4.21 VHDL/Matlab co-simulation closed loop: output. . . . 60
4.22 VHDL/Matlab co-simulation closed loop: output (zoomed). . . . 60
4.23 VHDL/Matlab co-simulation closed loop: load step output reaction. . . . 61
4.24 VHDL/Matlab co-simulation closed loop: load step reaction (zoomed). . . . 62
4.25 VHDL/Matlab FPGA-based closed loop buck converter. . . . 62

5.1 FPGA-TestChip: prototype model. . . . 66
5.2 FPGA-Test Chip: prototype 67
5.3 FPGA-TC: prototype output signals. . . . 68
5.4 FPGA output: ADC clock. . . . 68
5.5 FPGA output: DPWM square wave output. . . . 69
5.6 Co-simulation and FPGA-TC prototype: outputs comparison. . . . 69
5.7 FPGA-TC prototype: load step reaction. . . . 70
5.8 FPGA clocking wizard: $70MHz$ clock generator. . . . 71
5.9 ChipScope Pro Analyzer interface: input error. . . . 72
5.10 FPGA I/O prototype configuration: UCF file. . . . 73
5.11 Virtex6 FPGA: I/O mapping 73
5.12 Test Chip GUI interface: external loop configuration. . . . 75
5.13 Test Chip: I/O mapping. . . . 75

6.1 Closed loop configuration and self-tuning algorithm: on-line controller model. . . . 78
6.2 Matlab/Simulink model: LCO-based identification method. . . . 80
6.3 Matlab/Simulink model: LCO-based identification method output. . . . 81
6.4 Matlab/Simulink model: harmonic system response identification method. . . . 83
6.5 Matlab/Simulink model: harmonic system response identification method output 83
6.6 Pseudo Random Binary Sequence: Autocorrelation (2^{10}samples) 85

7.1 Non-idealities: ESR contribution effects on the steady state output waveforms. . . 87
7.2 Digitally controlled buck converter: steady state identification model. 89
7.3 Open loop identification method: step evolving reference voltage. 89
7.4 Open loop identification method: step response output voltage. 90
7.5 Open loop identification method: step evolving reference voltage (zoom). 91
7.6 Buck converter configurations: load identification (no ESR contribution). 91
7.7 Open loop identification method result: $f_{0,buck1}$, $ESR = 0\Omega$ 92
7.8 Open loop identification method result: $f_{0,buck2}$, $ESR = 0\Omega$. 93
7.9 Open loop identification method result: $f_{0,buck3}$, $ESR = 0\Omega$. 93
7.10 Buck converter configurations: load identification (ESR contribution). 94
7.11 Open loop identification method result: $f_{0,buck1}$, $ESR = 0.5\Omega$. 95
7.12 Open loop identification method result: $f_{0,buck2}$, $ESR = 1\Omega$. 95
7.13 Open loop identification method result: $f_{0,buck3}$, $ESR = 1\Omega$. 96
7.14 Open loop identification method result: $f_{0,buck3}$, $ESR = 1.8\Omega$. 96
7.15 Digitally controlled buck converter: Matlab/Simulink fixed-point model. 97
7.16 Steady state identification method: $\Delta\Sigma$ error feedback configuration. 98
7.17 Steady state identification method: dithering modelling. 98
7.18 Steady state identification method: dithering effects on the control to output transfer function (no ESR contribution). 99
7.19 Steady state identification method: dithering amplification effects on the control to output transfer function (no ESR contribution, $f_{0,buck1}$). 100
7.20 Control to output transfer function and noise shaper configurations. 101
7.21 Steady state identification method: dithering amplification effects on the steady state output voltage. 103
7.22 Steady state identification method: PSD of dithering amplification effects on the control to output transfer function (no ESR contribution). 103
7.23 Steady state identification method: dithering amplification effects on the control to output transfer function (ESR contribution). 104
7.24 Steady state identification method: dithering amplification effects on the control to output transfer function (ESR contribution and notch effects). 105
7.25 Steady state identification method: dithering effects on the output voltage. 107
7.26 Steady state identification method: dithering amplification effects on the output voltage ($\alpha = 2$). 107
7.27 Steady state identification method: dithering amplification effects on the output voltage ($\alpha = 3$). 108
7.28 Steady state identification method: dithering amplification effects on the output voltage (comparison). 108
7.29 Steady state identification method result: FFT and PSD comparison ($f_{0,buck1}$, $\alpha = 1$). 109
7.30 Steady state identification method PSD results: $f_{0,buck1}$, $ESR = 0\Omega$. 110
7.31 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$. 111
7.32 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$. 112
7.33 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 1). 113
7.34 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 2). 114
7.35 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 3). 114
7.36 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 4). 115
7.37 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 1). 115
7.38 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 2). 116

7.39 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 3). 116
7.40 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 4). 117
7.41 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$, $f_n = f_0$. 118
7.42 Steady state identification method PSD results: $f_{0,buck1}$, $ESR = 0.5\Omega$ (trial 1). 119
7.43 Steady state identification method PSD results: $f_{0,buck1}$, $ESR = 0.5\Omega$ (trial 2). 119
7.44 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 1\Omega$ (trial 1). 120
7.45 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 1\Omega$ (trial 2). 120
7.46 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 1\Omega$. 121
7.47 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 1.8\Omega$ (trial 1). 122
7.48 Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 1.8\Omega$ (trial 2). 122
7.49 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 1.2\Omega$ 123
7.50 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0.9\Omega$. 124
7.51 Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0.8\Omega$.. 124
7.52 Steady state identification method: Matlab/Simulink fixed-point model. 126
7.53 Steady state identification method PSD results: linear and log scale ($f_{0,buck1}$). . . . 127
7.54 Steady state identification method PSD results: linear scale ($f_{0,buck1}$). 127

8.1 Steady state identification prototype: model. 130
8.2 Steady state identification prototype: testing environment. 131
8.3 Hardware dithering amplification: $\Delta\Sigma$ modulator. 131
8.4 Steady state identification prototype: PSD output for $f_{0,buck1}$, $ESR = 0\Omega$. 132
8.5 Steady state identification prototype: PSD output for $f_{0,buck2}$, $ESR = 0\Omega$. 133
8.6 Steady state identification prototype: PSD output for $f_{0,buck3}$, $ESR = 0\Omega$. 134
8.7 Steady state identification prototype: multiple PSD output for $f_{0,buck1}$, $ESR = 0\Omega$. 135
8.8 Steady state identification prototype: multiple PSD output for $f_{0,buck2}$, $ESR = 0\Omega$. 136
8.9 Steady state identification prototype: multiple PSD output for $f_{0,buck3}$, $ESR = 0\Omega$. 137
8.10 Steady state identification prototype: statistics for $f_{0,buck1}$, $ESR = 0\Omega$. 137
8.11 Steady state identification prototype: statistics for $f_{0,buck2}$, $ESR = 0\Omega$. 138
8.12 Steady state identification prototype: statistics for $f_{0,buck3}$, $ESR = 0\Omega$. 138
8.13 Steady state identification prototype: PSD output for $f_{0,buck1}$, $ESR = 0.5\Omega$. 139
8.14 Steady state identification prototype: PSD output for $f_{0,buck2}$, $ESR = 1\Omega$. 140
8.15 Steady state identification prototype: PSD output for $f_{0,buck3}$, $ESR = 1\Omega$. 141
8.16 Steady state identification prototype: PSD output for $f_{0,buck3}$, $ESR = 1.8\Omega$. 142
8.17 Self-tuning prototype: hardware description. 144
8.18 Selt tunign algorithm: FSM (reset transitions omitted). 146
8.19 Self-tuning prototype: FPGA I/O configuration. 146
8.20 Self-tuning prototype: DEMO configuration[23]. 147
8.21 Self-tuning prototype: extracted load parameters ($f_{0,buck1}$,$ESR = 0\Omega$). 148
8.22 Self-tuning prototype: extracted load parameters ($f_{0,buck2}$,$ESR = 0\Omega$). 148
8.23 Self-tuning prototype: extracted load parameters ($f_{0,buck3}$,$ESR = 0\Omega$). 149
8.24 Self-tuning prototype: extracted load parameters ($f_{0,buck1}$,$ESR = 0.5\Omega$). 149
8.25 Self-tuning prototype: extracted load parameters ($f_{0,buck3}$,$ESR = 1\Omega$). 150
8.26 Self-tuning prototype: extracted load parameters ($f_{0,buck3}$,$ESR = 1.8\Omega$). 150
8.27 Self-tuning prototype: averaged extracted load parameters ($f_{0,buck2}$,$ESR = 1\Omega$). . 151
8.28 Self-tuning prototype: averaged extracted load parameters ($f_{0,buck3}$,$ESR = 1.8\Omega$). 152
8.29 Self-tuning prototype: PID gains regulation ($f_{0,buck2}$,$ESR = 1\Omega$). 153
8.30 Self-tuning prototype: load step response before the PID gains regulation. 153

8.31 Self-tuning prototype: load step response after the PID gains regulation. 154
8.32 Self-tuning prototype: dithering amplification effects on the output voltage. . . . 158

9.1 Data at the end of computation. Picture obtained with 18-bit resolution and scaling factor of 7. 160
9.2 Flowgraph of the CG-FFT. 161
9.3 Example of perfect shuffle. 162
9.4 Structure of the scalable architecture 164
9.5 Operation of FIFO blocks. 165
9.6 Diagram of architecture *pe_structural*. 171
9.7 Diagram of architecture *seu_behavioral*. 173
9.8 Diagram of architecture *datapath_structural*. 175
9.9 Dataflow diagram of the control of the processor. 177
9.10 Structural diagram of architecture *twiddle_factors_structural*. 179
9.11 Structural diagram of *tfc_datapath_structural* 180
9.12 Radix-2 DIF CG-FFT flowgraph with $\mathscr{I}_s$ sequences. 182
9.13 Distribution of twiddle factors in the Gauss'plane. 184
9.14 The *shared core* scalable CORDIC architecture 187
9.15 Structural diagram of module *cordic_core* 189
9.16 Structural diagram of the *shared core* architecture 191
9.17 Dataflow diagram of the *shared core* control 192
9.18 The *pipelined* scalable CORDIC architecture 193
9.19 Digital Signal Processing model. 194
9.20 Structural diagram of the datapath of the PSD computer. 199
9.21 Structural diagram of *autocorrelation_ram_structural* 204
9.22 Examples of addressing and BRAM selection fields. 205
9.23 Functional diagram of the control of the PSD computer. 206
9.24 Organisation of the TSE on the VHDL side. 209
9.25 Organisation of the TSE on the MATLAB side. 211
9.26 Error statistics on $\{W_N^{(k)}\}$ with rounding disabled. 214
9.27 Error statistics on $\{W_N^{(k)}\}$ with rounding enabled. 215
9.28 Shared core architecture speedup. 216
9.29 Computational data storage estimates for TF generators. 217
9.30 Synthesis results for the shared core architecture. 219
9.31 Maximum operating frequency of the shared core architecture. 220
9.32 Synthesis results for the pipelined architecture. 221
9.33 Maximum operating frequency of the pipelined architecture. 222
9.34 Synthesis results as a function of $B' - B$. 223
9.35 Maximum operating frequency as a function of $B' - B$. 224
9.36 Samples from the dataset. 225
9.37 Mean relative error as a function of B for the FFT processor. 225
9.38 Computation time of the FFT processor as a function of P. 226
9.39 Synthesis results of the FFT processor as a function of P. 228
9.40 Maximum operating frequency of the FFT processor as a function of P. 229
9.41 Performances of generated architectures. From [100]. 231
9.42 Data at the end of the first processing stage. 233
9.43 Square moduli at the second processing stage before scaling. 233

9.44 Data at the end of processing. . . . 234
9.45 Slice resource occupation of the PSD computer as a function of P. . . . 235
9.46 DSP blocks occupation as a function of P. . . . 236
9.47 PSD computer maximum frequency as a function of P. . . . 236

A.1 Functional diagrams of twiddle factor generators. . . . 263

B.1 The Gentleman-Sande butterfly. . . . 268
B.2 Length-16, Decimation-in-Frequency, In-order input, Radix-2 FFT. From [1]. . . . 270
B.3 Length-16, Decimation-in-Time, In-order output, Radix-2 FFT. From [1]. . . . 271
B.4 Sequence involved in computing an aperiodic discrete convolution. From [78]. . . . 273
B.5 Procedure for the periodic convolution of two periodic sequences. From [78]. . . . 274

C.1 Classic types of FFT pipelined architectures. From [39]. . . . 278
C.2 Flowgraph of the radix-2^2 DIF FFT algorithm. $N = 16$. From [39]. . . . 278
C.3 Pease's algorithm flowgraph, $N = 8$. From [75]. . . . 279
C.4 SPIRAL project architectures. From [70]. . . . 280
C.5 zhang-chen's structure of parallel FFT with CORDIC. From [132]. . . . 281
C.6 Vite-Frias' architecture Memory-based radix-4 design scheme. From [114]. . . . 282
C.7 Sung's architecture 2^{13}-point CORDIC-based split-radix design scheme. From [108]. 283
C.8 Al Sallab's memory-optimized FFT architecture. From [3]. . . . 284
C.9 Xiao's memory reduced CORDIC FFT. From [121]. . . . 285
C.10 O'sullivan's in-place FFT architecture. From [79]. . . . 286
C.11 Comparison of available architectures for Xilinx LogiCORE IP. From [123]. . . . 286
C.12 Xilinx LogiCORE *pipelined, streaming I/O* architecture. From [123]. . . . 287
C.13 Available Xilinx LogiCORE in-place *burst I/O* FFT architectures. From [123]. . . . 288
C.14 Available Altera MegaCore in-place FFT architectures. From [6]. . . . 289
C.15 Yu's CORDIC architecture. From [129]. . . . 290
C.16 Garrido's FFT-oriented CORDIC. From [33]. . . . 291

List of Tables

1.1 Buck converter: converter specifications. 8

4.1 ADC error encoding. 53
4.2 Digital control prototype: Virtex6 resource usage. 63
4.3 Digital control prototype: resource usage. 64

5.1 Test Chip: LSB_{ADC} possible settings 67

7.1 Steady state identification method results: Resonant frequency ($\alpha = 2$). 112
7.2 Steady state identification method results: Resonant frequency ($\alpha = 3$) 112
7.3 Steady state identification method results: ESR identification ($\alpha = 2$ $f_n = f_0$). . . . 125

8.1 Steady state identification prototype: f_0 identification results ($ESR = 0\Omega$). 135
8.2 Steady state identification prototype: ESR identification results. 136
8.3 tab:Self-tuning prototype: extracted, averaged load parameters and desired values comparison. 152
8.4 Self-tuning prototype: resource usage 154
8.5 Self-tuning prototype: Virtex6 resource usage. 158

9.1 Naming convention adopted in the design. 167
9.2 Considered values of $B - B'$. 213
9.3 Comparison between the proposed architecture and [79] 227
9.4 Synthesis results for the scalable archtiecture with $N = 256$. 229
9.5 Comparison between the proposed architecture and [125] 230
9.6 Comparison between the scalable hybrid and other architectures. 230

A.1 Matching of *fft_global_settings.vhd* and *autocorrelation_settings.vhd*. 262
A.2 Selection of the architecture for the internal ROM. 263

Chapter 1

A general overview of buck converters

In this chapter a brief theoretical introduction to dcdc power supplies is presented. The main distinction among dcdc converters, is usually done between step up (boost converters) and step down (buck converters) configurations. Step up converters output a dc voltage V_{out} bigger then the dc input value V_{in}, while, the dc output value V_{out} is lower then V_{in} for step down configurations. The entire dissertation refers to step down dcdc buck converters, converters overview in the following sections has been addressed just referring to buck converters. Considerations about efficiency in real converters have been drawn in terms of power losses.
Once that the theoretical overview has been discussed, considerations about design specifications are addressed referring to the output filter.

1.1 Ideal Model

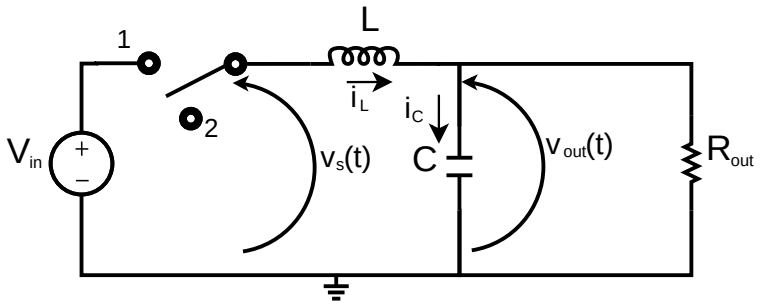

Figure 1.1: Buck converter: ideal model.

The ideal buck converter configuration is presented in Fig.1.1. The time variant signal $v_s(t)$, is equal to the dc input voltage V_{in} when the switch S is in position 1 and, is null when it is in position 2. Considering the switch driven by a square wave signal with duty cycle

$D = \frac{t_{on}}{T_s}$ and period $T_s = \frac{1}{f_s}$, the signal $v_s(t)$ is consequently a square wave having frequency f_s (Fig.1.2).

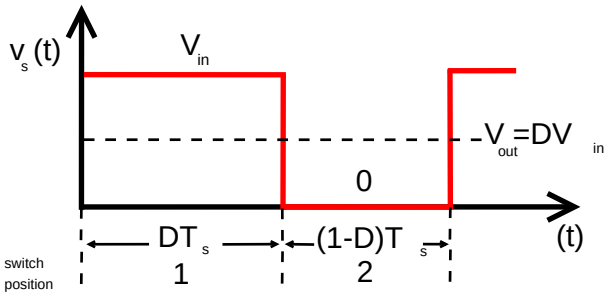

Figure 1.2: Buck converter: switching node voltage waveform.

Switch duty cycle variations define the dc value of the output voltage and, from Fourier analysis, it can be assumed that the dc value of a square wave is equal to its averaged value in the considered switching period (Fig.1.2). Hence, during a steady state condition the dc output voltage for a buck converter is equal to the input voltage times the duty cycle of the square wave which drives the switch S:

$$V_{out} = DV_{in}. \tag{1.1}$$

Considering the Eq.1.1 the buck converter presents a linear control characteristic, and, because $0 \geq D \leq 1$ the output voltage is less than or equal to the input voltage.
The system efficiency of the ideal model in Fig.1.1 is close to 100%. The power dissipated by the switch is ideally null, when the switch is in the open position the current is equal to zero and contrariwise the voltage is null during the close position. The second order LC output filter is required in order to avoid the switching frequency harmonics on the output voltage. However, a control block is always integrated into a power system. In a buck converter the output voltage is function of the duty cycle and the feedback control system modulates the duty cycle in order to obtain the desired dc output level. A closed loop buck converter is modelled in Fig.1.3, non-idealities as the inductor series resistance (R_L) and the Effective Series Resistance (ESR) on the output capacitor are as well represented. The equivalent of the ideal switch S (Fig.1.1 is represented with two power-MOS (M_{HS} and MLS) driven by the square wave generated from the control feedback. The regulation signal outputs from the controller is computed in order to minimize the difference $V_{out} - V_{ref}$, where V_{ref} is the desired dc output voltage.

During the steady state $V_{out} \approx V_{ref}$ and the duty cycle is modulated sinusoidally by the controller, the output switch voltage contains low frequency components which are reflected on the output voltage. The LC filter corner frequency is selected both in order to pass the desired low frequency components of $v_s(t)$ and to attenuate the high-frequency switching harmonics. Considering a real filter, some harmonics generated by the switch give rise to a ripple voltage v_{ripple} over the dc value V_{out}:

$$v_{out}(t) = V_{out} + v_{ripple}. \tag{1.2}$$

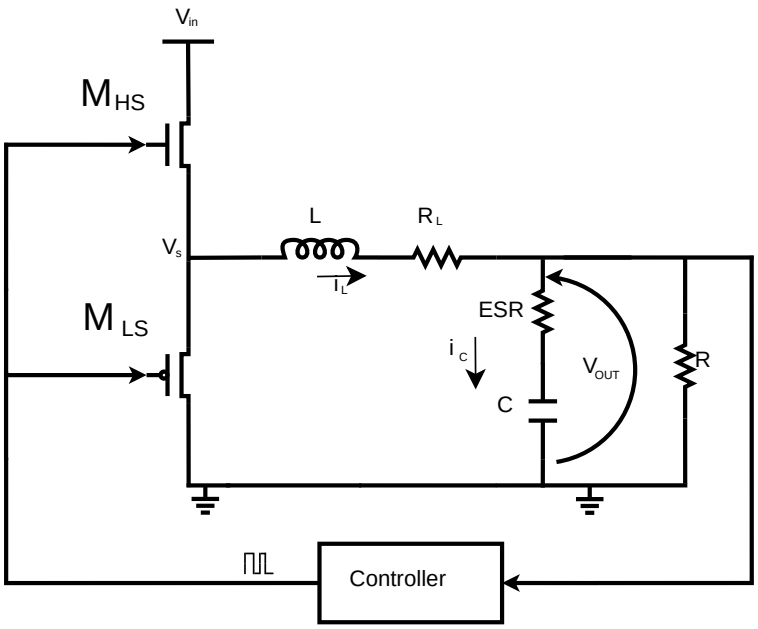

Figure 1.3: Buck converter: closed loop buck configuration.

The undesired ripple voltage is then consequent of the incomplete suppression of the switching harmonics by the low-pass filter, this ac components have to be very little in a well designed converter.

1.2 Continuous Conduction Mode

Continuous conduction mode (CCM) is the most used operative condition for a dcdc converter. Referring for instance to an open loop buck converter (Fig.1.4) the CCM is obtained when the current (i_L) flows continuously every switching step through the inductor (L).

If the ripple voltage on the output dc value can be neglected, the inductor voltage v_L value can be referred only to the V_{out}(Fig.1.5). Referring to Fig.1.1, $v_L = V_{in} - V_{out}$ when the switch is in position 1 and $v_L = -V_{out}$ in position 0. As consequence of the inductor characteristic equation, is simple to get the inductor current slope $i_L(t)$ during the switching period (Fig.1.6). The inductor current begins at some initial value $i(0)$. During the first subinterval, with the switch in position 1, the inductor current increases with the slope $\frac{V_{in}-V_{out}}{L}$. At time $t = DT_s$ the switch changes to position 2. The current then decreases with the constant slope given by $\frac{-V_{out}}{L}$. At time $t = Ts$ the switch changes back to position 1, and the process repeats. The inductor value can be chosen in order to reduce ripple effects in the inductor current. Taking into account that for $t = DT_s$ the inductor current is equal to two times the ripple current, it is easy to evaluate the ripple effects:

$$\Delta i_l = \frac{V_{in} - V_{out}}{2L}(DT_s). \tag{1.3}$$

The Eq.1.3 is usually used to dimension the inductor L during the converter design. From the inductor current point of view, the steady state condition is reached when the total current

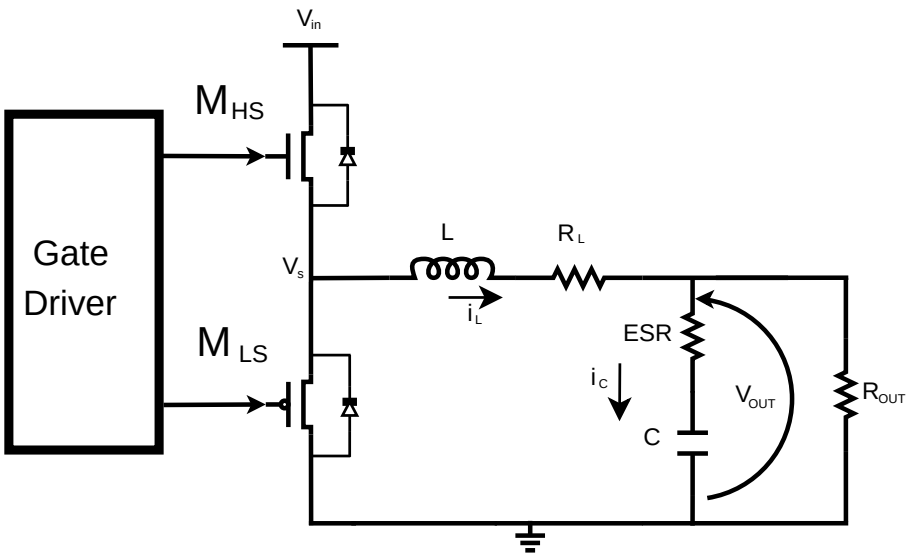

Figure 1.4: Buck converter: Open loop configuration.

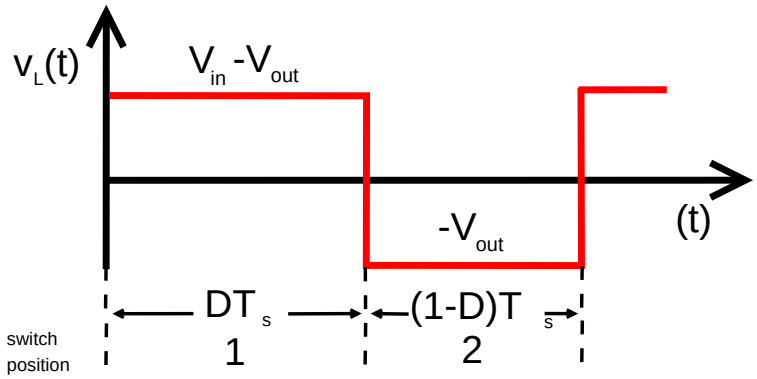

Figure 1.5: Buck converter: inductor voltage waveform.

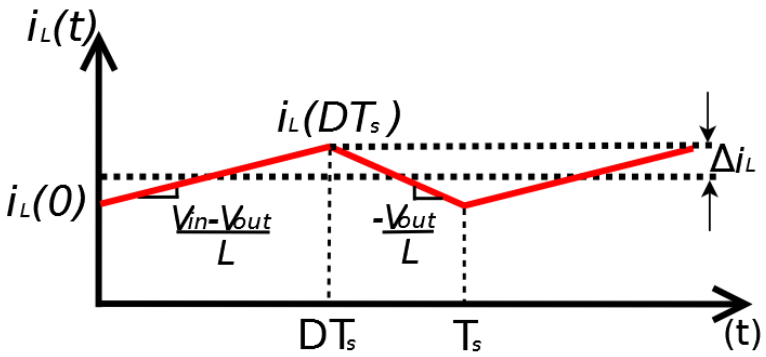

Figure 1.6: Buck converter: Inductor current waveform.

per switching period is the same even if the duty cycle D changes. Consequently, the current $i_L(0)$ in $t = 0$ is equal to the current in $t = T_s$(Fig.1.6):

$$\frac{i_L(T_s) - i_L(0)}{T_s} = \frac{1}{LT_s}\int_0^{T_s} v_L(t)\,dt = 0. \tag{1.4}$$

By dividing per T_s the Eq.1.4, the averaged value $\langle v_L \rangle$ in the switching period is null. Considering Fig.1.5, the averaged inductor voltage can be also written as:

$$\langle v_L \rangle = 0 = D(V_i n - V_o ut) + (1 - D)(-Vout). \tag{1.5}$$

The steady state condition in Eq.1.1 can be easily extracted from Eq.1.5. This confirm that, the duty cycle D will be the main parameter to be considered in order to have a well regulated converter.

So far ripple effects v_{ripple} on the output voltage has been neglected. Two main factors contribute to this undesired effect, the first is related to the output capacitor C and the second by the ESR (Fig.1.4). Considering a null ESR, the ripple voltage Δv_{out} can be easily related to the ripple current Δi_L:

$$\Delta v = \frac{\Delta i_L T_s}{8C}. \tag{1.6}$$

Considerations on the output capacitance can be drawn from the Eq.1.6, its value is usually defined fixing the maximum ripple accepted. The ripple on the voltage output can be expressed by substituting the Eq.1.3 in the Eq.1.6:

$$\Delta v = \frac{1}{8} V_{out}(1 - D)\frac{T_s^2}{LC}. \tag{1.7}$$

Taking into account both that the switching frequency is $f_s = \frac{1}{T_s}$ and the resonant frequency is $f_0 = \frac{1}{\sqrt{2\pi LC}}$, the relative voltage can be expressed in terms of frequency:

$$\frac{\Delta v}{V_{out}} = \frac{\pi^2}{2}(1 - D)\frac{f_0^2}{f_s^2}. \tag{1.8}$$

From 1.8 can be deduced how choosing a resonant frequency much lower than the switching frequency, ripple effects can be reduced. Furthermore, because ESR increases ripple effects on the output voltage, this assumption becomes more significant when the series resistance in the output capacitor can't be neglected. Another important parameter which is usually considered during the converter design is the relationship between the ripple voltage and the time constant $\tau_= = R_{out}C$. For lower time constants values a smaller output voltage ripple can be achieved, this assumption comes out from assuming a ripple current piecewise linear.

1.3 Discontinuous Conduction Mode

The Discontinuous Conduction Mode (DCM) is obtained when switches part of the buck converter are implemented as unidirectional devices. Let us refer to the model in Fig.1.7. The DCM arises when the inductor current ripple is large enough to cause the polarity reversing of the applied switch current, hence the current unidirectional assumptions are violated. The DCM typically occurs with large inductor current ripple in a converter operating

at light load and containing current-unidirectional switches. Since it is usually required that converters operate with their loads removed, DCM is frequently encountered. Indeed, some converters are purposely designed to operate in DCM for all loads.
When the output current $Iout$ is higher than $i_L/2$ the inductor current flows continuously and the converter still works in CCM. When $I_{out} < i_L/2$ the diode conducts and the inductor current decrease, the inductor discharge its previous stored energy to the output capacitor until the current drops to zero. Moreover, the diode blocks the reverse current and the inductor current remains zero until the next switching cycle 1.8. Because the inductor currents is zero for some intervals, this kind of operative mode is called discontinuous.

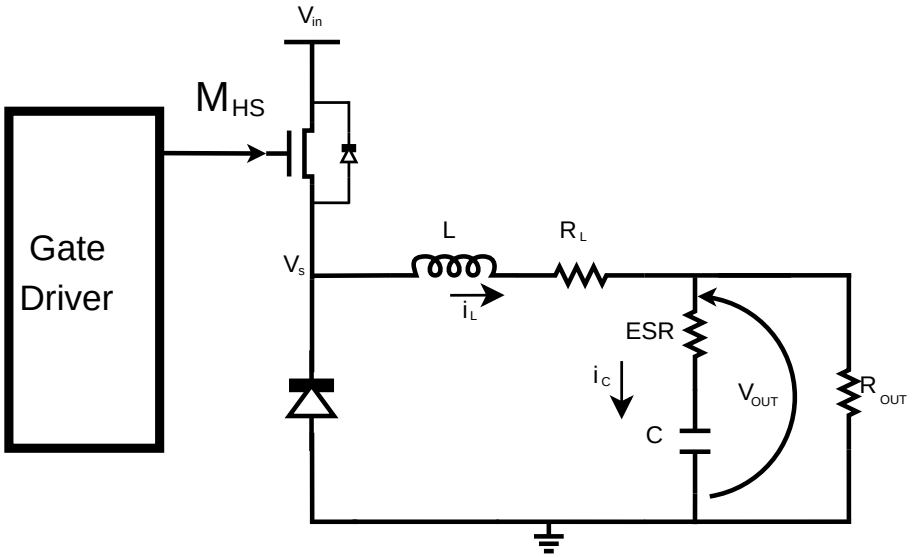

Figure 1.7: Buck converter: asynchronous control configuration.

In DCM the properties of the converter change significantly and the steady state condition previously described in Eq.1.1 is not valid. As described the DCM arises when $I_{out} < i_L/2$, it is immediate to correlate this behaviour with the load resistance R_{out}. The critical boundary between CCM and DCM is usually described as function of the duty cycle D and in terms of critical load resistance R_{crit}.

$$R_{crit}(D) = \frac{2L}{(1-D)T_s}. \tag{1.9}$$

The critical resistance introduced in Eq.1.9, permits to evaluate the converter tendency to switch from continuous to discontinuous conduction mod. For $R_{out} < R_{crit}$ the converter operates in CCM and for $R_{out} > R_{crit}$ the DCM is triggered.
Because a null inductor current subinterval is obtained, the conversion factor D differs from CCM and is function of the critical output current ($I_{out_critic} = R_{crit} V_{out}$):

$$D = \frac{V_{out}}{V_{in}} \sqrt{\frac{I_{out}/I_{out_crit}}{1 - V_{out}/V_{in}}}. \tag{1.10}$$

The inductor current ripple Δi_L is equal to the current peak (Fig.1.10) and is equal to:

$$\Delta i_L = \frac{V_{in} - Vout}{L} D T_s. \tag{1.11}$$

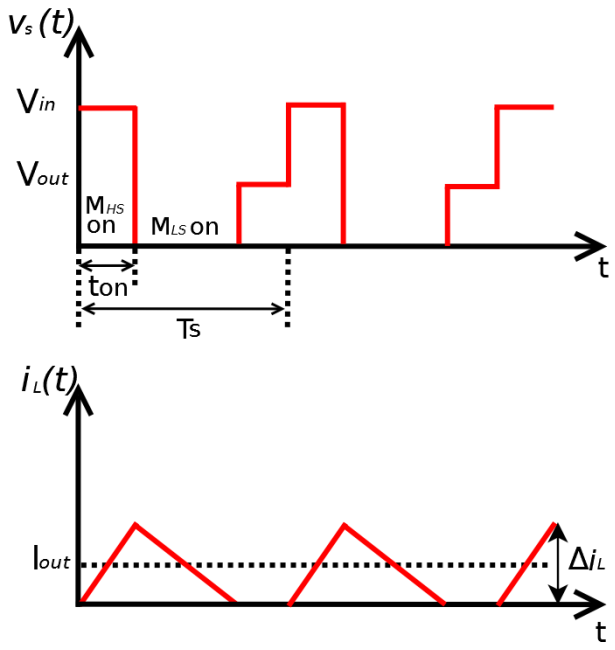

Figure 1.8: Buck converter: waveforms in DCM mode.

Considering both Eq.1.1 and Eq.1.10, the averaged duty cycle is lower during the DCM operative mode. Averaged considerations can be done about the duty cycle D, it results lower in DCM mode and the output voltage ripple is not considered a constraint during the converter design.

1.4 Power Losses

Power losses in conventional dcdc converters are typically summarized in three categories [28]:

$$P_{loss} = P_{cond}(I_{load}) + P_s + P_{fixed}. \tag{1.12}$$

In Eq.1.12 conduction loss P_{cond}, switching loss P_s and fixed loss (P_{fixed}) can be respectively distinguished.
The conduction losses directly depend on the load current I_{load}. The finite on-resistance of the power transistor, the diode forward voltage drop and both inductor (R_L) and capacitor (ESR) series resistance are the main contributions.
The switching losses are related to energy dissipated (E_s) for the switching activity. Power transistors output and gate capacitance, diode capacitance, diode stored minority charge, inductor and transformer core loss, snubber loss and date driver losses are the all contributions which affect switching losses.
The fixed losses comprise loss factor which are not part neither of load current nor switching frequency. Fixed contributions are caused by the controller standby current and all the leakage currents in the power transistors, diode and capacitor.
In order to reduce the losses, variable frequency converters has been introduced in [28] and [99]. By reducing the switching frequency as the load current decreases, the converter standby current is as well decreased by one to two orders of magnitude. At high values of

Symbols	Constraints
I_{out_max}	Maximum load current
I_{peak_limit}	Switch peak current limit
V_{in}	Input voltage
V_{out}	Output voltage
$\Delta Vout/Vout$	DC output voltage precision
$\Delta Vout$	Output voltage ripple

Table 1.1: Buck converter: converter specifications.

load current the P_{cond} is dominant and high efficiency is usually achieved replacing the power diode with a power transistor [27, 68, 69]. The voltage drop across the on-resistance of the transistor results to be smaller than the forward voltage of the diode thus reducing the conduction loss. As the output current decreases the inductor current can assume negative values for a portion of the switching period leading the capacitor C to discharge through the inductor L and then consuming additional power. For this reason, the converter is usually allowed to work in DCM. Losses mathematical formulas are well known for converters working in CCM [29]. At the State of the Art, most of approaches tend to approximate the CCM loss characterization model for describing losses in DCM [49]. The derivation of comprehensive formulas for estimating both switching and conduction losses in buck converters operating in DCM-only scenarios is presented in [31].

1.5 Output filter design considerations

During previous sections, indirect considerations regarding the load output filter have been done. In a dcdc buck converter the output is composed by a second order LC filter. In Equations 1.6, 1.7 and 1.8, has been shown that the output voltage ripple mainly depends on the output capacitance and the related ESR contributions. Both assuming a situation where ripple effects are mainly dependent on the load capacitance and having constraints on the output voltage ripple, it is possible to extract the resonant frequency f_0 from Eq.1.8 and consequently the product LC. In this approach the switching frequency has been imposed to respect output voltage ripple constraint, however the extracted resonant frequency has to be at least ten times lower than the switching frequency. To ensure that switching frequency harmonics are well filtered out, the switching frequency is often chosen twenty times bigger. So far a way to find the product LC has been described. Usually the the inductance L is defined by imposing the inductor current ripple constrain in the Eq.1.3, then considering LC/L the capacitance value can be easily obtained. The output capacitance can be chosen to reduce the output voltage ripple, for lower time constant values a smaller output voltage ripple can be achieved. Ceramic capacitors are usually preferred because their smaller time constant, smaller capacitor physical profile and higher reliability. However electrolytic capacitors are largely used in power supply filter circuits at low frequencies, indeed they provide high capacitance in a small volume.
Specifications on both current and voltage commonly given during a dcdc converter design, have been summarised in Tab.1.1.

Chapter 2

Buck Converter in Continuous Conduction Mode

Aim of this chapter is to introduce the buck converter and to describe its properties during the Continuous Conduction Mode (CCM). Theoretical steady state results, obtained in the previous chapter, will be proven with a parametric Matlab/Simulink modelled buck converter. The first section summarise various approaches used to obtain the ac model of a buck converter. However, all methods are based on averaging, perturbation and linearisation of the model. The circuit averaging method has been first presented, then the ac small signal circuit is deduced with this technique. A further characterization of the converter in the frequency domain, has been done trough the *Line to Output* and *Control to Output* transfer functions. Matlab plots of the control to output transfer function is presented for three *LC* output load conditions and also considering Effective Series Resistance (ESR) contributions on the output capacitor of one buck converter configuration.
Furthermore, a Matlab embedded function has been realised to emulate a buck converter with parasitic modelling. Firstly, a circuit characterisation of the converter has been introduced, hence, a Matlab embedded function has been coded after the classical circuit analysis method. The realised function has been integrate into the Simulink model of the open loop buck converter. The steady state behaviour in CCM operative mode has been then confirmed with Simulink simulations. The introduced Matlab/Simulilnk modelling technique represents an alternative user-friendly approach for converters modelling.

2.1 AC small signal model

Aim of this section is the ac modelling of a CCM buck converter, a small signal reference model has always to be known when a regulation closed loop system has to be constructed. The modelling process of a system is always a step by step process. Firstly, the main behaviour is studied then non-idealities are added. Different methods can be used to obtain a small signal model of a CCM converter, these are mainly current injected approach, circuit averaging, and the state-space averaging method. The end results of all these methods are nearly equivalent, however the averaging and small signal linearisation are both key steps for modelling PWM converters.

A classical modelling method is based on averaging the equations over the switching period, then when system perturbation occurs (every converter signal is constituted by a steady state or dc term plus a perturbation one) these equations are linearised around an operating point. This approach is mainly based on the integration of low frequency components in the inductor and capacitor waveforms:

$$L\frac{d\langle i_L(t)\rangle_{T_s}}{dt} = \langle v_L(t)\rangle_{T_s}, \tag{2.1}$$

$$C\frac{d\langle v_C(t)\rangle_{T_s}}{dt} = \langle i_C(t)\rangle_{T_s}, \tag{2.2}$$

Where in 2.1 the expression $\langle i_L(t)\rangle$ is the average of $i_L(t)$ over an interval T_s:

$$\langle i_L(t)\rangle_{T_s} = \frac{1}{T_s}\int_t^{t+T_s} i_L(\tau)\,d\tau. \tag{2.3}$$

As in Chap.1, during the steady state both the inductor volt-second and capacitor charge balance (Eq.2.1 and Eq.2.2) are zero. These are non-linear equations, which can use to describe how both the inductor currents and capacitor voltages change when non-zero averaged inductor voltage and capacitor current are respectively considered over a switching period. To obtain a linear model that is easier to analyse, it is usual to construct a small signal model linearised around an operating point. Considering for instance the dependence of the steady state output voltage V_{out} on the duty cycle D, the steady state equation (Eq.1.1) for the ideal buck converter is not valid. In Fig.2.1 is shown the linearisation approach for a buck converter operating with a 50% duty cycle. With a non-linear control to output characteristic, variations on the duty cycle will be reflected as variations on the output voltage. If control variations are small enough the corresponding output voltage variations can be computed by the linearisation mechanism.

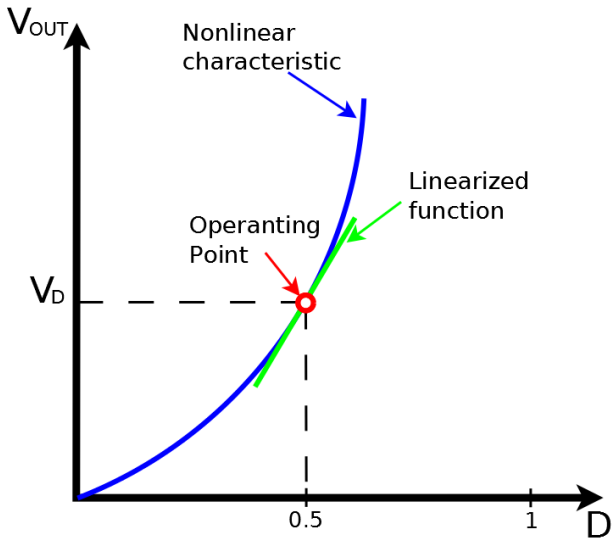

Figure 2.1: Linearisation approach for buck converters.

Two well-known approaches for the ac modelling method are respectively the state-space averaging and the circuit averaging. Because of their switching operation, power electronic converters are periodic time-variant systems. The generalized state-space averaging method is a way to model them as time independent systems, defined by a unified set of differential equations able to represent circuit waveforms. Therefore, it can be a convenient approach for designing controllers dedicated to switch converters.
Circuit averaging is another technique for derivation of converter equivalent circuits[117]. In this approach, the averaging process is applied directly on the converter waveforms directly. Since circuit averaging involves averaging and small signal linearisation, it is equivalent to state-space averaging. The key step in circuit averaging is to replace the converter switches with voltage and current sources to obtain a time-invariant circuit topology. The waveforms of the voltage and current generators are defined to be identical to the switch waveforms of the original converter 2.2. Once a time-invariant circuit network is obtained, the converter waveforms can be averaged over one switching period in order to remove the switching harmonics. Then, any non-linear element in the averaged circuit model can be perturbed and linearised leading to the small signal ac model.
In Fig.2.2 is presented the circuit studied during circuit averaging method, while the equivalent model for an ideal buck converter has been shown in Fig.2.3.
The *switch network* is a two-port network, a classic approach to analyse this block is to choose two terminals as independent inputs. Referring to the circuit in Fig.2.3, the independent inputs are respectively $v_1(t) = v_{in}(t)$ and $i_2(t) = i_L(t)$. The circuit equations are firstly averaged, hence the system is perturbed and then linearised. The resulting model gas been shown in Fig.2.4, where every capital letter indicates the dc value of the signal and every letter with an hat indicates its small signal ac variation.

The small-sgnal ac equivalent model of the converter is characterized by three equations:

$$L\frac{d\hat{i}_{out}(t)}{dt} = \hat{d}(t)V_{in} + D\hat{v}_{in}(t), \tag{2.4}$$

$$C\frac{d\hat{v}_{out}(t)}{dt} = \hat{i}_{out}(t) - \frac{\hat{v}_{out}(t)}{R}, \tag{2.5}$$

$$\hat{i}_{in}(t) = \hat{d}(t)I_{out} + D\hat{i}_{out}(t). \tag{2.6}$$

Equations 2.4, 2.2 and 2.6 are respectively the *inductor voltage*, the *capacitor current* and the *input current* relationships, the system in Fig.2.4 is obtained from these equations. The function of the averaged switch model (Fig.2.4) is, the transformation of dc and small ac voltage and current levels according to the 1:D conversion and, the introduction of ac voltage and current variations into the converter circuit driven by the control input *d(t)*.

2.1.1 Transfer functions

Starting from the small signal ac model introduced in the previous section, it is possible to deduce the transfer functions of the system. A dcdc converter has usually two external inputs, its frequency behaviour is then described through two distinct transfer functions. In addition to the classical line to output transfer function, the control to output transfer function describes the relationship between the input control (duty cycle) and the output voltage. Most of cases the control to output transfer relationship has larger interest, if the input voltage is supposed to be approximately stable, the output voltage in CCM for a buck

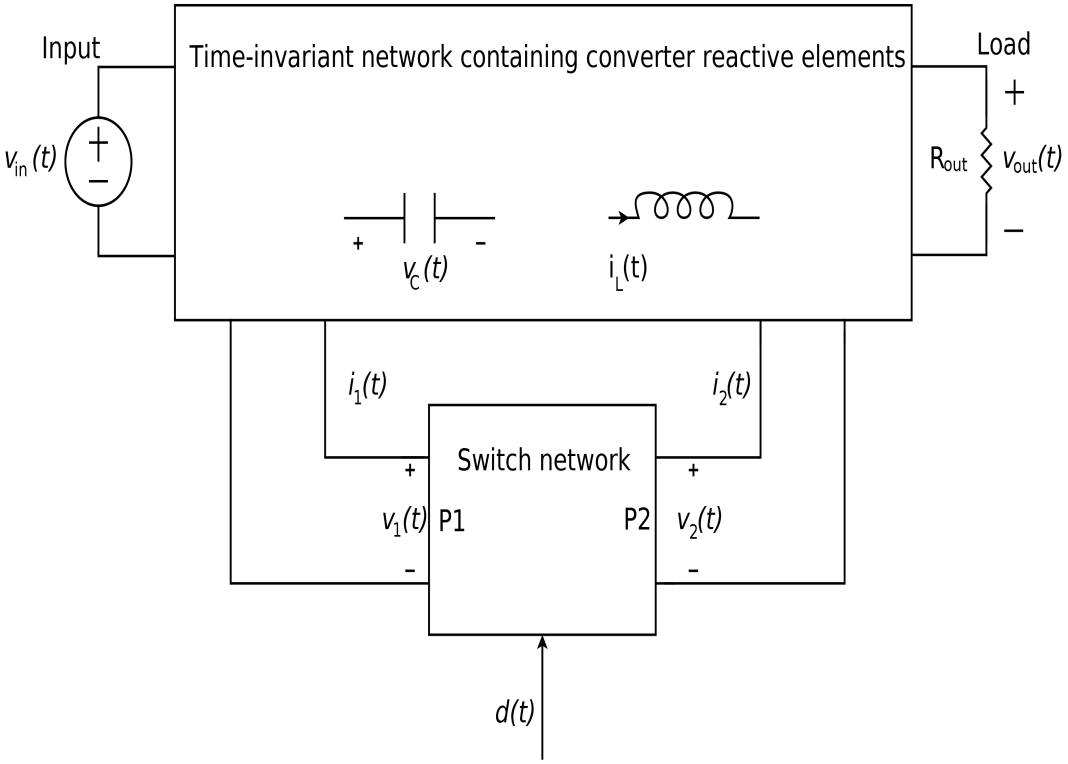

Figure 2.2: Averaged method analysis: model.

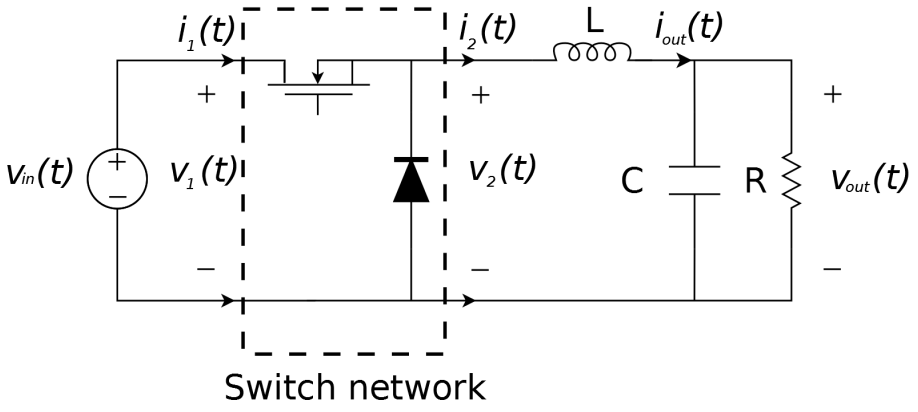

Figure 2.3: Averaged method analysis: buck converter circuit.

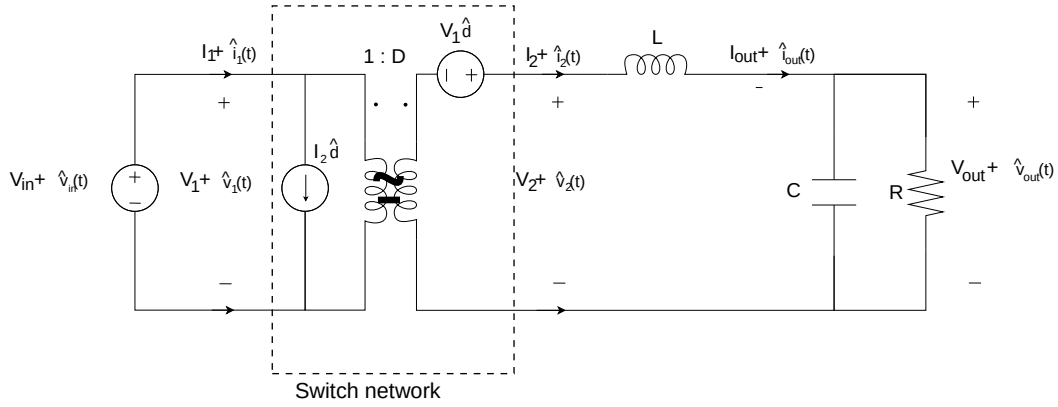

Figure 2.4: Small signal circuit: ideal buck converter.

converter is directly related to the duty cycle D (Eq.1.1). In Sec.2.1 a small signal ac model has been obtained for an ideal buck converter, in case of non-idealities both the circuit in Fig.2.4 and the transfer function change. Further non-idealities can be, for instance, the ESR and/or the on-resistance of the Power-MOS where the input control is applied. In Fig.2.5 is presented the small signal ac model with the ESR contribution.

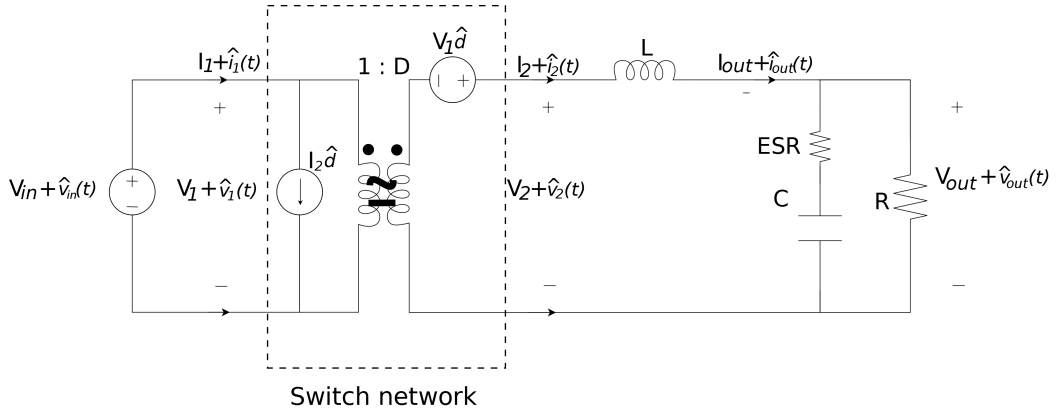

Figure 2.5: Small signal circuit : buck converter with ESR contribution.

The line to output transfer function is found by setting the duty cycle variations $\hat{d}(S)$ to zero, and then solving the small signal ac model for the transfer function from $\hat{v}_{in}(s)$ to $\hat{v}_{out}(s)$:

$$G_{v_{in}}(s) = \left.\frac{\hat{v}_{out}(s)}{\hat{v}_{in}(s)}\right|_{\hat{d}(s)=0}. \tag{2.7}$$

The transfer function in Eq.2.7 tells how disturbance in the input voltage are reflected on the output voltage. For instance if the converter input voltage contains some harmonics, this transfer function is used to define their effect on the output voltage. On the counterpart the control to output transfer function is found considering null input voltage variations

($\hat{v}_{in}(s) = 0$) and analysing the equivalent ac model for $\hat{v}_{out}(s)$ as a function of $\hat{d}(s)$:

$$G_{v_d}(s) = \left.\frac{\hat{v}_{out}(s)}{\hat{d}_{in}(s)}\right|_{\hat{v}_{in}(s)=0}. \tag{2.8}$$

The control to output function in Eq.2.8, clarify how duty cycle variations are reflected on the output voltage. The $G_{vd}(s)$ is very important for the regulator performance, indeed it is the main component of the loop gain in closed loop regulators.
A further important parameter is the *Converter Output Impedance*, which describes how variations in load current affect the output voltage:

$$Z_{out}(s) = \left.\frac{\hat{v}_{out}(s)}{-\hat{i}_{load}(s)}\right|_{\hat{v}_{in}(s),\hat{d}(s)=0}. \tag{2.9}$$

Referring to a ideal small signal ac model for the buck converter (Fig.2.4), both obtained transfer functions have the same second order structure:

$$G(s) = G_{d0}\frac{1}{(\frac{s}{\omega_0})^2 + \frac{2}{Q\omega_0} + 1}. \tag{2.10}$$

Using conventional circuit analysis, resulting ideal transfer functions are:

$$G_{v_{in}}(s) = \left.\frac{\hat{v}_{out}(s)}{\hat{v}_{in}(s)}\right|_{\hat{d}(s)=0} = D\frac{1}{s^2LC + s\frac{L}{R_{out}} + 1}, \tag{2.11}$$

$$G_{v_d}(s) = \left.\frac{\hat{v}_{out}(s)}{\hat{d}_{in}(s)}\right|_{\hat{v}_{in}(s)=0} = \frac{V_{out}}{D}\frac{1}{s^2LC + s\frac{L}{R_{out}} + 1}, \tag{2.12}$$

$$Z_{out}(s) = \left.\frac{\hat{v}_{out}(s)}{-\hat{i}_{load}(s)}\right|_{\hat{v}_{in}(s),\hat{d}(s)=0} = \frac{sL}{s^2LC + s\frac{L}{R_{out}} + 1}. \tag{2.13}$$

The obtained transfer functions refer to an ideal buck converter structure and its small signal ac model, an important role is played by the ESR contribution. Including ESR effect in circuit averaged model (Fig.2.5), the transfer functions change their structure. Because results in this dissertation refer to the control to output transfer function, the ESR effects in the frequency response are highlighted only for this kind of transfer function:

$$G_{v_d}(s) = \frac{V_{out}}{D}\frac{sC * ESR + 1}{s^2LC + s\frac{L}{R_{out}} + 1}, \tag{2.14}$$

Compared with the ideal buck converter, the ESR contribution causes a zero insertion in the transfer function. Hence the control to output transfer function is composed by two complex conjugated poles and possibly one zero due to the ESR.

In Fig.2.6 is presented the frequency behaviour of the ideal buck converter. In this figure the control to output transfer function has been modelled in Matlab, the relationship 2.12 has been characterized in terms of V_{in}, L, C, R_{out} and then plotted in a Bode digram. Three different buck configurations has been distinguished, every case is characterized by an inductance $L = 47\mu H$, an input voltage $V_{in} = 14V$ and an output resistance $R_{out} = 50\Omega$ considered. The load capacitance are $22\mu F$ (blue curve), $10\mu F$ (red curve) and $4.7\mu F$ (green

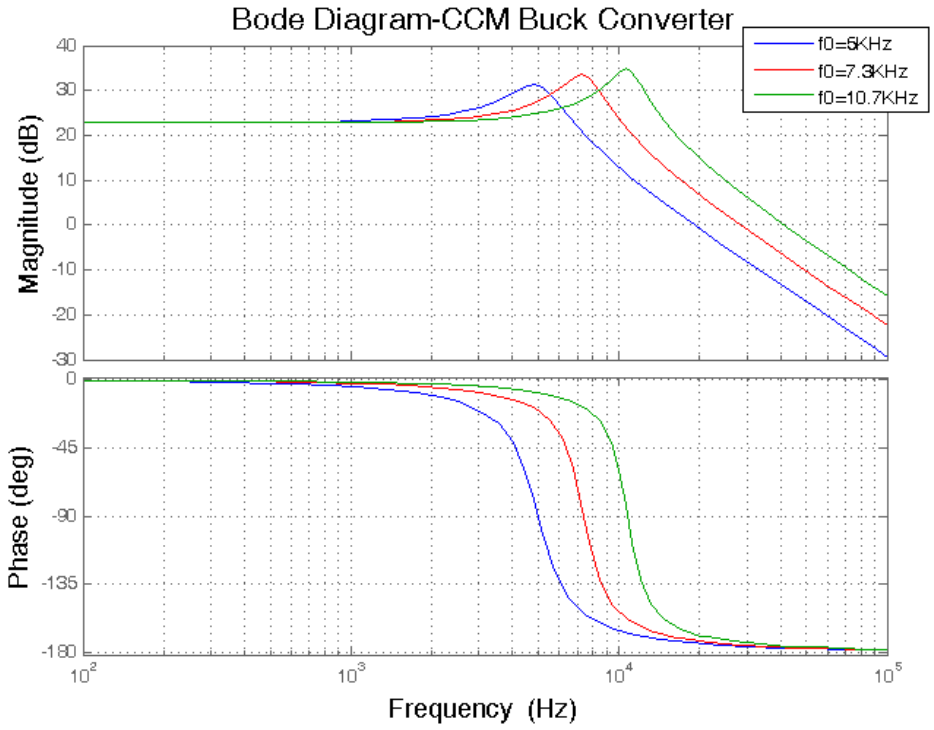

Figure 2.6: Control to Output transfer function: ideal buck converters.

curve), through the relationship $f_0 = \frac{1}{\sqrt{2\pi LC}}$ the resulting resonant frequencies are respectively $5kHz$, $7.3kHz$ and $10.7kHz$. As can be observe for the three curves presented in Fig.2.6, after the resonant frequency f_0 the magnitude goes down with a slope of $-40dB/dec$ and the phase quickly rolls down at $-180°$. For this kind of systems, having a good control loop can be very important in order to increase margins and bandwidth.
In Fig.2.7 the $ESR = 0.5\Omega$ contribution has been introduced in the configuration with $C = 22\mu F$. The zero contribution at $f_z = \frac{1}{2\pi ESR*C} = 14.5kHz$ is visible in the red curve both in terms of magnitude and phase, a larger bandwidth is obtained and the magnitude goes down with $-20dB/dec$ of slope.

2.2 Matlab modelling

In the previous section, Matlab-based results of the control to output transfer function has been presented. The aim of this section is to built a Simulink block including the mathematical behaviour of the buck converter, this model is needed to test the power stage in open loop mode (constant input duty cycle).
Starting from a converter reference model, it is possible to solve the differential equations and extract the state variables which characterise the system. These solutions can be exploited to describe a Matlab function and then, integrated into a Simulink block. This kind of approach is very useful to emulate the ac behaviour of the converter. The solution can be computed in cases where non-idealities, losses and parasitics elements can be consid-

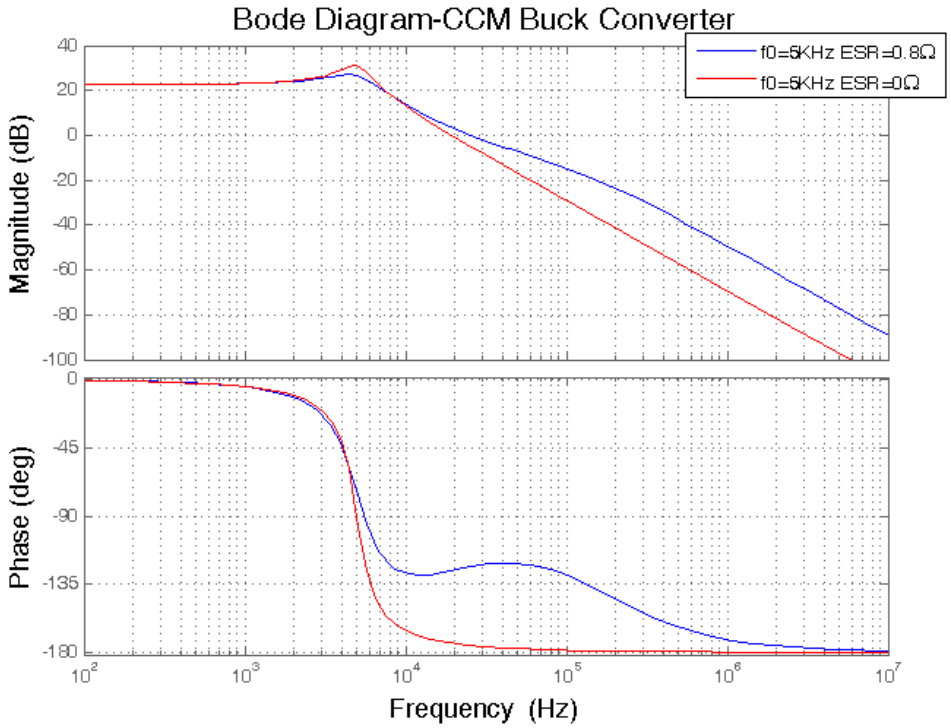

Figure 2.7: Control to Output transfer function: buck converters with ESR contribution.

ered, for instance the equations can be computed considering finite on/off resistance of the power-MOS and many others effects.

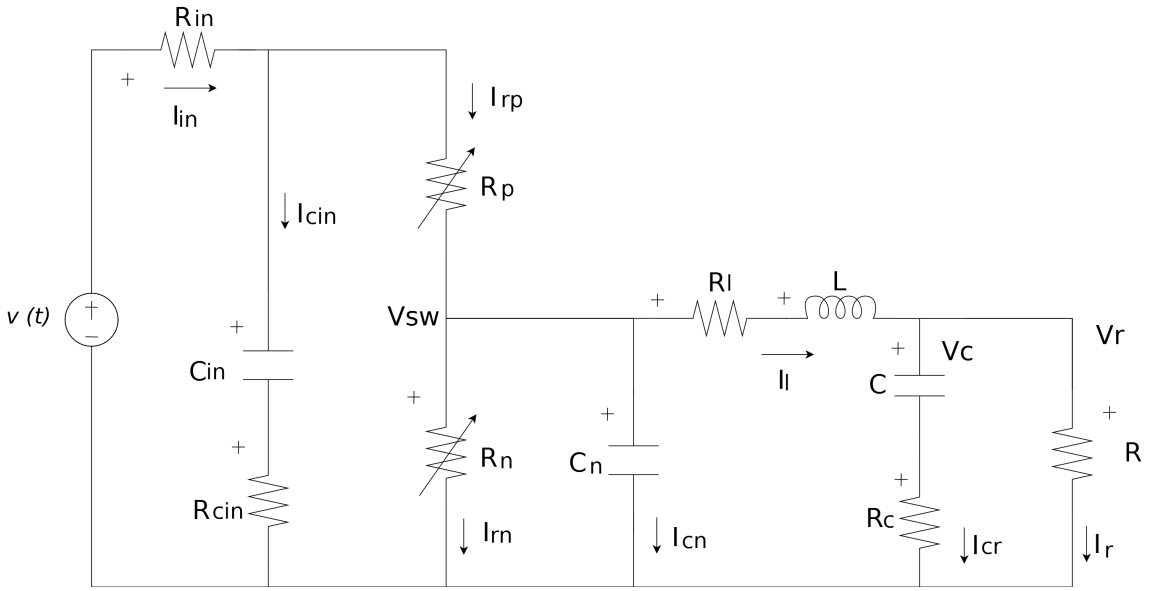

Figure 2.8: Equivalent circuit: buck converter with non-idealities.

Let's consider to model a buck converter with the circuit in Fig.2.8. This circuit takes into account parasitics elements and series resistor both for load inductor and load capacitor. The power-MOS (high-side and low-side) are modelled like variable resistors (respectively R_p and R_n) with a parasitic capacitor (Cn). This circuit is described with nineteen equations in nineteen variables, the presence of four memory elements define just as many state equations. This system of equations can be solved, for instance, by using the MuPAD toolbox just writing al the relationship simply deduced with the classical circuitry analysis technique. This approach is very useful when many equations has to be solved, very complicated solutions are easy to meet. In case of the buck converter circuit in Fig.2.8, the MuPAD resulting solutions can be used to create a Matlab Function which models the converter.

```
%#eml
function [Vr , Il, Vs]   = buck_CCM(Vin, R, pmos_on, nmos_on, parameters)
persistent Vco Vcno Vcpo Ilo  L Rc Rl Cn Cp C T %persistent it's used to declare global variables
%isempty is a function used for the initialisation
if isempty(Vco)
    T = 1/ parameters.Fdig / parameters.analog_OS;
    % capacitors state variable initial condition
    Vco =  parameters.Buck.Vc_initial;
    Vcno =10e-3;
    Vcpo= 10e-3;
    %inductor state variable intial condition
    Ilo =  parameters.Buck.Il_initial;
    %Load
    L = parameters.Buck.L/T ;
    Rl = parameters.Buck.Rl;
    C = parameters.Buck.C/T;
    Rc = parameters.Buck.Rc;
    %Power MOS parassitic capacitor
    Cn = parameters.Buck.Cn/T;
    Cp = parameters.Buck.Cp/T;
end
%Emulating the switching control input
Rp = parameters.Buck.Rp_off;
Rn = parameters.Buck.Rn_off;
if pmos_on >=0.5
    Rp = parameters.Buck.Rp_on;
elseif nmos_on >= 0.5
    Rn = parameters.Buck.Rn_on;
end
% equations from solver
Vcp=(Rp*(Rn*(Ilo*L*(1 + C*(Lc + R + Rc)) - (L + R + Rl)*(Cn*Vcno - Cp*Vcpo) + ...
    + R*(1 + Cn*Rn + C*(Lc + Rc + Rl + Rn + Cn*(Lc + Rc + Rl)*Rn)))* Vin));
Vcn=Rn*Rp*-(Ilo*L*(1 + C*(Lc + R + Rc))) + (L + R + Rl)*(Cn*Vcno - Cp*Vcpo) + ...
    (R + C*R*(Lc + Rc) + L*(1 + C*(Lc + R + Rc)) + Rl + C*(Lc + R + Rc)*Rl)*Rn*(1 + Cp*Rp)*Vin;
Vc = Ilo*L*R*(Rn + Rp + (Cn + Cp)*Rn*Rp) + Ilco*Lc*((L + R + Rl)*Rn + (L + R + Rl + Rn + (Cn + Cp)*(L + R + Rl)*Rn)*Rp)...
    + Cn*R*Rn*Rp*Vcno +Vco + R*Rn*(1 + Cp*Rp)*Vin;
Il = (Ilo*L*(1 + C*(Lc + R + Rc))*(Rn + Rp + (Cn + Cp)*Rn*Rp) + ...
    Cp*R*Rn*Rp*Vcpo - Cp*Rc*Rn*Rp*Vcpo + (Lc + R + Rc)*Rn*(1 + Cp*Rp)*Vin) + Rn*(Cn*Rp*Vcno + Vin + Cp*Rp*(-Vcpo + Vin));
Vr = (R*(Ilo*L*(1 + C*(Lc + Rc))*(Rn + Rp + (Cn + Cp)*Rn*Rp) + ...
    (Lc + Rc)*Rn*(1 + Cp*Rp)*Vin) + Rn*(Cn*Rp*Vcno + Vin + Cp*Rp*(-Vcpo + Vin)));
%output switchin node
Vs = Vcn;
% update state variables
Vcno = Vcn;
Vcpo = Vcp;
Vco = Vc;
Ilo = Il;
```

Figure 2.9: Matlab function: buck converter with non-idealities.

In Fig.2.9 is shown the Matlab implementation of the model in Fig.2.8, equations in this figure has been cut for plotting reasons. The function *buck_CCM* receives as input the input voltage V_in, the load resistor value R, the pointer *parameters* and both the driver signals *pmos_on* and *nmos_on* which emulate the input square wave of duty cycle D. The pointer *parameters* permits the access to the structure *Buck* which contains the constant parameters of the system, for instance the inductance value L which can be easily accessed

with the entry *parameters.Buck.L*. The global variables are declared with the key word *persistent*, among these there are load components comprising the series resistance (L,Rl,C,Rc), the power-MOS with non-idealities representation (Rp,Rn,Cn,Cp) and the state variables (Vco,$Vcno$,$Vcpo$,Vco). The variables initialisation is done with the function *isempty()* and thanks to the parameters file. Among them variables is the time step T, which is step of the computations used during the simulations. In this case T is coincident with the switching frequency which drives the power-MOS. Still referring to Fig.2.9, after the initialisation there is the power-MOS representation. The variable resistances Rp and Rn can be equal either to their on or off value, indeed when the input signal *pmos_on* is high the Rp is equal to its on value and Rn to its off value and vice versa. Both on and off resistance values are defined in the *paramters* structure. Then, the equations related to the circuit in 2.8 solved through MuPAD are just copied and pasted in the Matlab description after the converter parameters. As visible in the block **equations from solver** in Fig2.9, every time step the function computes the state variables related to memory elements and the output voltage (Vr). The output computed from the *buck_CCM* function are, the output voltage, the inductor current and the switching voltage Vs which correspond to switching node Vsw in the circuit model.

2.3 Matlab/Simulink open loop model

Once the converter has been modelled through a Matlab function (Fig.2.9), the open loop system (as in Fig.1.7) can be modelled in a Simulink-based behavioural system. The embedded function described in Fig.2.9, can be integrated in a Simulink block for testing and characterizing the power plant in CCM.

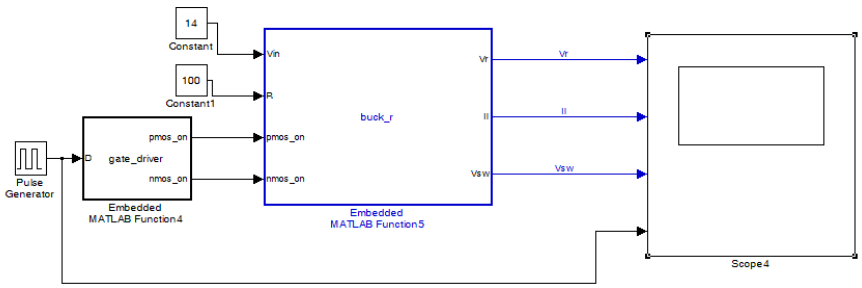

Figure 2.10: Matlab/Simulink model: open loop buck converter.

In Fig.2.10 is shown the Simulink model used for testing the buck converter. The block **buck_r** is the Simulink representation of the embedded function described in the previous chapter (2.10). In Fig.2.11 is shown the **buck_r** block implementation, it contains only the function needed to recall the Matlab modelled converter (Fig.2.9). As in the modelled function, this block receives the load resistance value R, the input voltage Vin and the signals *pmos_on* and *nmos_on* to emulate the input control square wave. Moreover the outputs are the output voltage Vr, the inductor current Il and the switching node voltage Vsw.

A further block and its related embedded function is needed to generate the gate driver function (Fig.2.10), the Simulink **gate_driver** block contains directly the implementation de-

```
function [Vr , Il, Vsw]    = buck_r(Vin, R, pmos_on, nmos_on, parameters)
%#eml
% This block supports the Embedded MATLAB subset.
% See the help menu for details.
[Vr , Il, Vsw]    = buck_r(Vin, R, pmos_on, nmos_on, parameters);
```

Figure 2.11: Simulink user's defined block.

scribed in Fig.2.12. This function receives the square wave D, generated with a Simulink built-in **Pulse_Generator**, and outputs both signals to drive the high side and low side power-MOS (Fig.1.7).
Starting from the circuit representation and passing through the Matlab embedded function realization, has been described how is possible to arrive to a Simulink modelling of an open loop buck converter. Next to the realization is the model test parameter choice, for the buck converter has been chosen $R = 100\Omega$, $Vin = 14V$. Signals *pmos_on* and *nmos_on* can be alternatively either 1 or 0, depending on the **Pulse_Generator**. The generated square wave has a fifty percent duty cycle and a switching frequency f_s one hundred times slower than the *clock* frequency ($f_s = f_{clk}/100$), it means that a fifty percent duty cycle square wave is generated every one hundred *clock* cycles. The term *clock* used in this context refers to the interval between two computations done by the Simulink simulation solver (fixed-step simulation in Fig.2.13).

```
function [pmos_on, nmos_on] = gate_driver(D)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.


if D == 1
    pmos_on =1;
    nmos_on =0;
elseif D == 0
    pmos_on =0;
    nmos_on =1;
else
    pmos_on =0;
    nmos_on =0;
end
```

Figure 2.12: Matlab function: gate driver.

In Fig.2.13 has been shown two windows related to timing configuration of the simulation. Let's call the *clock* as $Fdig$ and, because the simulation frequency is a global parameter, its value is initialized through the *parameters* structure. A clock of $Fdig = 70MHz$ has been chosen for the simulation. The right window in Fig.2.13 is related to the Simulink solver, the simulation is a fixed-step type with a sample time time equal to $1/70MHz = 14.3nsec$. Moreover, the left window in the same figure is related to the **Pulse_Generator** configuration, the configuration define a fifty percent duty cycle square wave with a *switching* time one hundred times bigger than the *clock* time.

Simulation results has been summarised both in Fig.2.14 and 2.15, where signals connected to the block **Scope** of Fig.2.10 has been shown. In Fig.2.14 is shown the output voltage Vr. Can be concluded that the main steady state relationship (Eq.1.1) for buck converters working in CCM has been confirmed. The output voltage is equal to the input voltage times

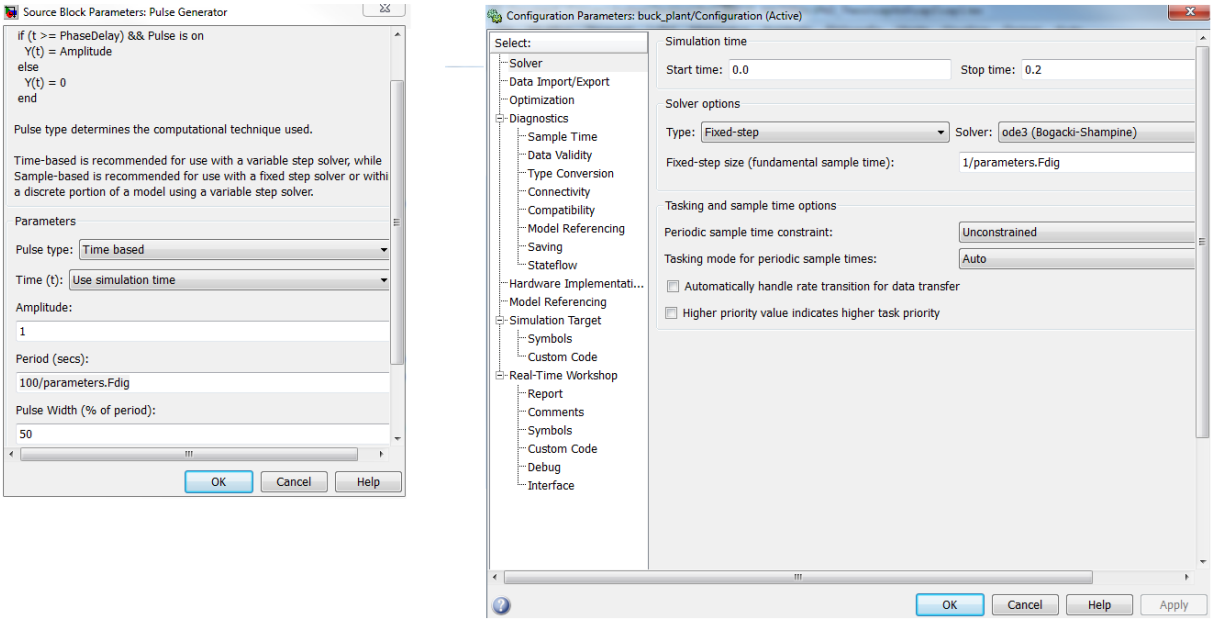

Figure 2.13: Simulink: Fixed-step simulations.

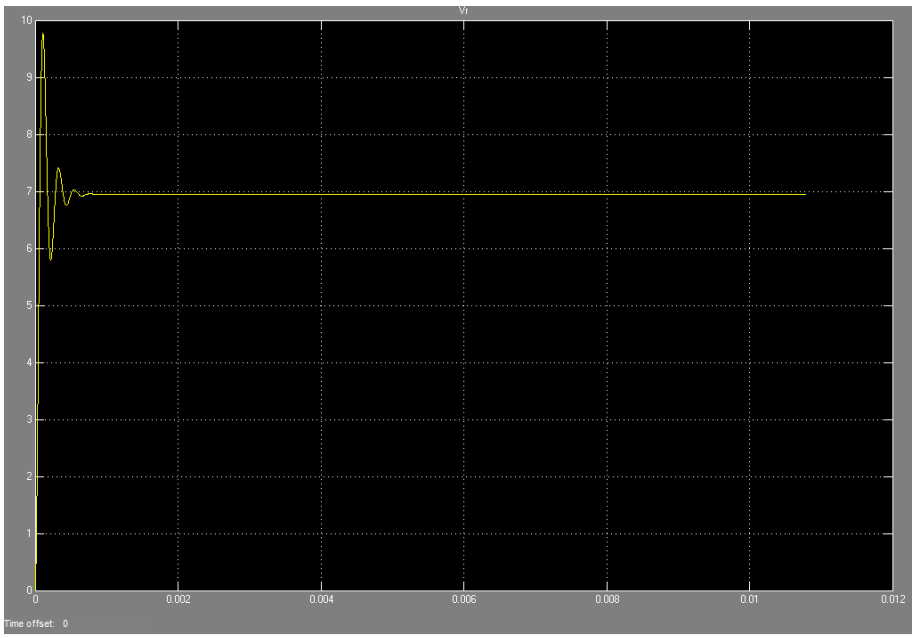

Figure 2.14: Matlab/Simulink: open loop output voltage ($D = 0.5$, $V_{in} = 14V$).

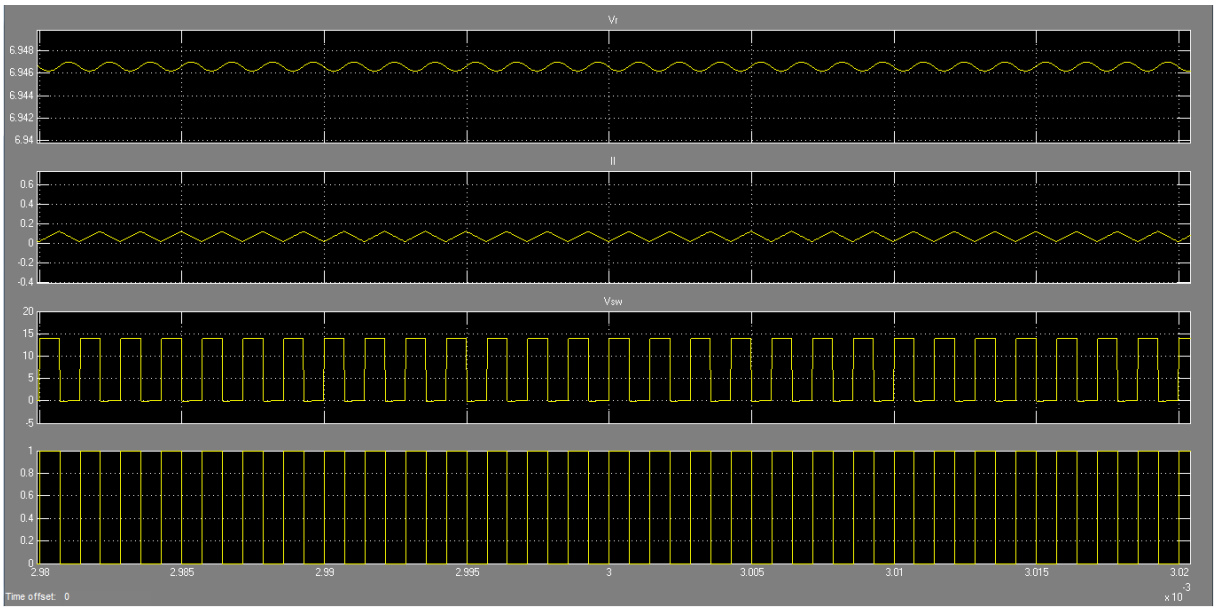

Figure 2.15: Matlab/Simulink: open loop zoomed output voltage ($D = 0.5$, $V_{in} = 14V$).

the duty cycle of the square wave applied at the **gate driver** block. Considering $Vin = 14V$ and $D = 0.5$ the expected output voltage is of $7V$, the steady state condition is proven in Fig.2.14. In Fig.2.15 has been shown a steady state zoom of the, output voltage (Vr), inductor current Il, the switching voltage Vsw and the fifty percent duty cycle input square wave. From this picture some observations can be drawn. Firstly the output voltage is very close the expected value and ripple effects at the *switching* frequency ($f_s = Fdig/100 = 700kHz$) can be observed. The inductor current il has the expected steady state behaviour for a CCM buck converter, as discussed in the previous chapter and showed in figure 1.6 it results null every switching period and max in the on/off transition of the square wave. The switching node (Vsw) as well confirms the expected behaviour. It is a square wave having the same duty cycle of the input square wave and, is equal to the input voltage when the high side power-MOS on (and the low-side is off) or equal to zero when the low side is off (and high side is on).

Chapter 3

Digitally controlled buck converters in CCM

Reducing ripple effects on the output voltage, more precision requests on the dc output level and avoiding low frequency noise, are all factors that make necessary to move from an open to a closed loop system to compute the control duty cycle every switching period T_s. In this chapter theoretical considerations about the digital control loop are introduced referring to buck converters. The control loop computes the duty cycle needed to reach the desired dc output voltage V_{ref} and, at the same time, it has to improve the system dynamic. Considerations in the frequency domain, are shown through Matlab obtained Bode diagrams of the transfer functions related to control loop blocks in Fig.3.1. Moreover, theoretical design basics of the Proportional Integrative Derivative (PID) compensator are introduced and, improvements in terms of both margins and bandwidth are considered.
Consideration about the effects of the finite loop resolution are drawn together with loop design considerations. The Loop resolution improvements through the dithering effects are considered. The $\Delta\Sigma$ modulator in its Error Feedback configuration is characterized and different noise shaper structures are compared.

3.1 The digital control feedback

A digitally controlled buck converter in presented in Fig.3.1, the control structure is mainly the digital representation of a classical analog control structure for power converters. The digital compensator represents the main block for a regulation loop, most of cases the best choice for CCM buck converters control is the Proportion Integrative Derivative (PID) compensator. Proportional Derivative compensator (PD) leads to better rejection of high frequency disturbances, indeed is used both to improve the phase margin and extend the bandwidth of the feedback loop. Proportional Integrative (PI) compensator leads both to better rejection of low frequency disturbances and very small steady state error, because the low frequency loop gain is increased. The PID compensator or regulator, combines these properties in order two achieve the advantages of both approaches.

In Cha.1, the steady state relationship which drives the buck converter is $V_{out} = DV_{in}$ has

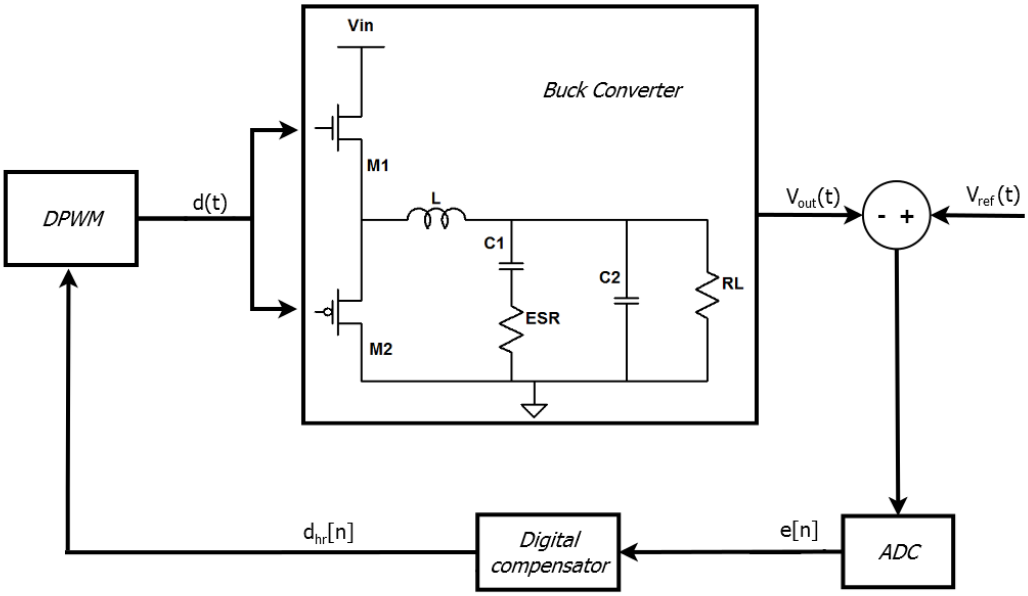

Figure 3.1: Digitally controlled buck converter: model.

been introduced. The regulation loop has to be designed to compute the duty cycle control word. The desired output voltage or the reference voltage is indicated with V_{ref}, every switching step the error between the actual V_{out} and V_{ref} is computed. The error $e[n]$ is then digitalized thanks to the ADC, it depends on the ADC resolution and represents the amount that the regulator have to compensate. The PID receives the error and computes the new high resolution duty cycle $d_{hr}[n]$ to compensate the error. The DPWM block considers the MSB (Most Significant Bits) of $d_{hr}[n]$ to generate the related square wave $d(t)$ to drive both high and low side power-MOS (M_{HS} and MLS). This general overview of the digital control feedback loop, highlights that in this kind of approach the main constraint is the resolution and the quantization effects [93]. Both the ADC and the DPWM introduce quantizations effects which are important to be modelled and taken into account. This kind of problems have been addressed in [85, 86], the most important considerations during a digital loop design refer to the relationship between quantizations effects introduced by the ADC and the DPWM, these blocks represent the real interface with the buck converter and are essentially non-linear quantizer which could introduce *limit cycle oscillation* (LCO) in the output voltage (Sec.3.3.3). If this oscillation occurs, the converter stability could be affected and cannot be predicted by the stability theory developed for linear time invariant systems. Therefore, additional conditions have to be considered during the design of the digital controller in order to avoid LCO.
Furthermore, delays introduced by DPWM and ADC in a control loop have to be considered during the design phase. In digitally controlled dcdc converters all the converter variables are sampled at a sampling period T. The output voltage is monitored at instants $t = nT$ where n is an integer number in the range $[0, +\infty]$. The sampling of the converter variables,

together with the delay introduced by each control block, will introduce a delay in the feedback. The sampling period of the ADC converter has to be chosen properly to avoid aliasing in the spectrum of the sampled error voltage. Because of the periodic behaviour, the most intuitive choice is to set the sampling frequency equal to the switching frequency of the converter ($T = T_s$). In this case, all signals of the digital controller will be updated once every switching cycle and the power consumption can be limited. An exact small signal discrete time model for digitally controlled pulse-width modulated (PWM) dcdc converters operating in constant frequency continuous conduction mode (CCM) with a single effective AD sampling instant per switching period, has been introduce in [63]. The model takes into account sampling, modulator effects and delays in the control loop.

3.1.1 Delays in the digital loop

During the control loop design ,the delay introduced by each block involved in the regulation have to be considered. Let's call t_d the total delay in the loop:

$$t_d = t_c + t_{d1} + t_{dpwm}, \tag{3.1}$$

where t_c is the ADC conversion time, t_{d1} is the delay introduced by the digital compensator, t_{dpwm} is the $DPWM$ delay and t_g is the gate driver propagation delay. The main contribution to this delay is represented by delay introduced both to ADC and DPWM quantizers. Referring to the control to output transfer function in Eq.2.8 and considering that a delay in the S-domain has the expression $\exp -st_d$, the resulting transfer function becomes:

$$G_{vd}(s)|_{td} = G_{vd}(s)e^{-st_d}. \tag{3.2}$$

This contribution introduces high frequency poles in the transfer function, without having consequences for the magnitude. In Fig.3.2 delay contribution has been considered in the Bode plot, for a buck converter with resonant frequency $f_0 = 5kHz$ and switching frequency $f_s = 449kHz$. In the example a total time delay close to the switching period has been chosen, in a real implementation this values are very close because the new control world computation takes t_d and have to be done before the switching period $t_s = 1/f_s$.

Hence the first constraint which has to be considered during the digital control system is the compromise between delays introduced by the feedback loop and the switching period. The new duty cycle has to be computed before every switching period, the sampling time T of the new error plus the computation time t_d has to be less then T_s.
The continuous square wave $d(t)$ is output from the DPWM, it translates the digital control word in a time continuous square wave and can be seen as a digital to analog converter (DAC). The combination of DAC and ADC blocks might originate a delay in the feedback loop that is approximately equal to half of the sampling period T, where $T = Ts$. The modulator delay depends on its structure, single and double edge are the most used. Moreover, hybrid DPWM structures has been presented in [127, 111].

This section is focused on single edge DPWM, the duty cycle $d_{hr}[n]$ is sampled at the beginning of each switching cycle and its value is constant until the next cycle 3.3. The DPWM can be modelled as a ramp, it samples the duty cycle at switching time and at the same time activates the on period of the square wave $d(t)$. When the ramp reaches the sampled duty

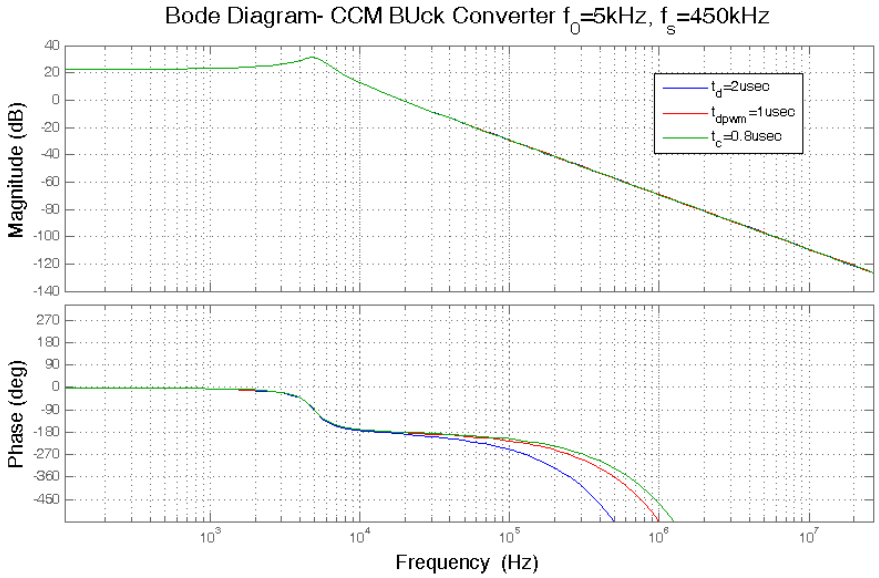

Figure 3.2: Control to output transfer function: loop delay effects.

cycle the off period of $d(t)$ start. This behaviour is summarised in Fig.3.3. There is no delay from the sampling instant of the duty cycle and the turn on instant of the signal output by the DPWM, but there is a delay between the sampling instant of the duty cycle and the turn off instant of the output signal. The modulator delay, in this case, can be easily determined to be $t_{dpwm} = DT_s$, where D is the average duty cycle value.
Form the digital point of it results natural to think at the DPWM as a counter. Because this counter is activate every switching step to count at the system clock frequency (f_{clk}), its minimum resolution n_{DPWM} depends on the relationship between clock and switching frequency:

$$n_{DPWM} = \left| \log_2 \left(\frac{f_{clk}}{f_s} \right) \right|. \tag{3.3}$$

For instance, for $f_{clk} = 70MHz$ and a switching frequency $f_s = 449kHz$ a 8 bits DPWM is needed. The main advantages of this approach are that, counter-based modulators are able to reduce or avoid LCOs and only one counter is used in terms of resources. On the counterpart, the main disadvantages is need of high clock frequency [93, 127, 110].
According to what said, the DPWM transfer function can be easily deduced:

$$G_{DPWM}(s) = K_{DPWM} \exp(-st_{DPWM}) = \frac{1}{2^{n_{DPWM}} - 1}. \tag{3.4}$$

Further quantization effects in a control loop are introduced from the non-linear ADC structure. The ADC transfer function is defined as:

$$G_{ADC}(s) = \frac{K_{ADC} \exp(-st_c)}{1 + \frac{s}{\omega_{ADC}}}. \tag{3.5}$$

In Fig.3.4 is shown the ADC characteristic related to the model used in the entire thesis. A zero error bin can be introduced to model the ripple voltages. During the steady state

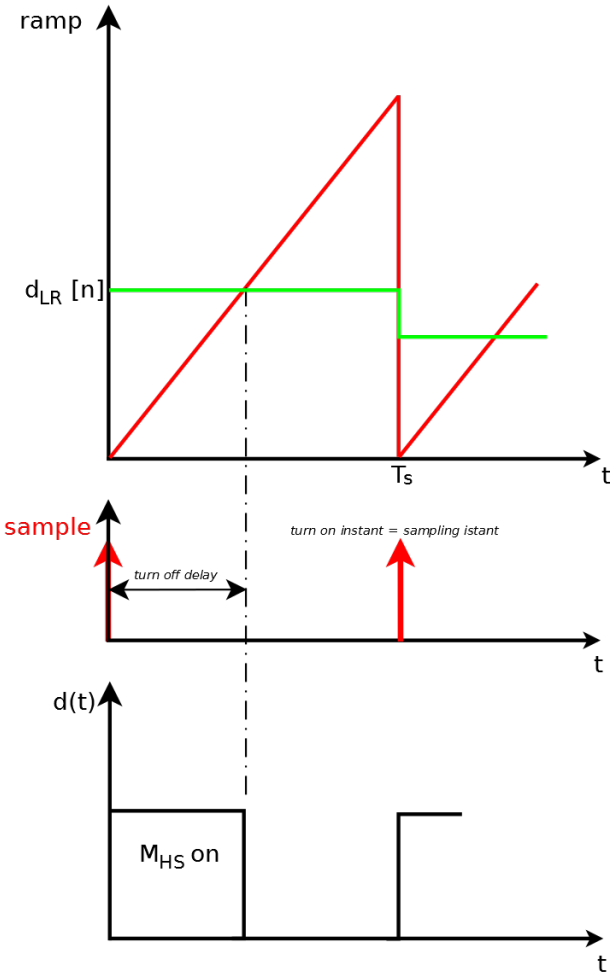

Figure 3.3: DPWM waveforms.

the error should be ideally null ($V_{out} = V_{ref}$), but ripple effects are introduced by both the switching frequency harmonics and the quantization noise. This kind of ADC permits to achieves effectively higher resolution in digitally controlled dcdc converters ans, a non-zero scheme for output voltage error coding, increases the LCO frequency beyond the resonant frequency f_0[134].

3.1.2 The compensator

The compensator design approach usually starts from the continuous time domain and then moves to the digital domain. To design the digital compensator both the control to output transfer function $G_{vd}(s)$ and the total delay in the digital control loop has to be known (Sec.3.2). Two main approaches can be adopted to study the compensator transfer function:

- The **Direct design** approach works directly in the Z-domain and the transfer function $G_c(z)$ can be directly obtained without passing for the S-domain.
- The **Emulation** approach take the advantage of the knowledge about compensator design in the continuous domain.

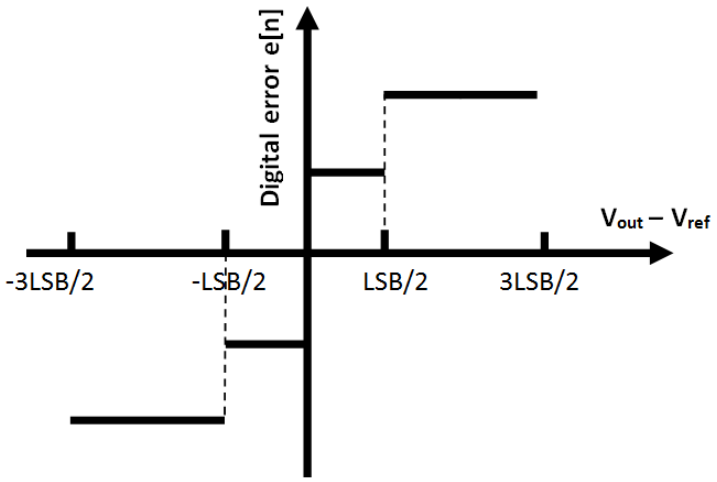

Figure 3.4: ADC output characteristic: zero error bin.

The *numerical integration* and the *pole-zero mapping* are numerical techniques exploited by the emulation approach. The numerical integration technique approximates a system described by a set of linear differential equations, while the pole-zero mapping find the equivalent discrete expression of a continuous transfer function just substituting a pole locate in $s = s_0$ with the equivalent $z = e^{s_0 T s}$ in the Z-domain.
The numerical integration technique writes the transfer function in terms of system differential equations, then the derivative of the state variables are approximate with *Forward rectangular*,*Backward rectangular* or *Trapezoid* rule. These three approaches bring to an equivalent between s and z, it means that starting from a transfer function in the S-domain the equivalent in the Z-domain is obtained substituting s with its equivalent z:

- Forward rectangular rule:

$$s \rightarrow \frac{z-1}{T_s}, \tag{3.6}$$

- Backward rectangular rule:

$$s \rightarrow \frac{z-1}{zT_s}, \tag{3.7}$$

- Trapezoid rule or bilinear transformation:

$$s \rightarrow \frac{2}{T_s}\frac{z-1}{z+1}. \tag{3.8}$$

Some considerations about the stability of the system during the mapping in the Z-domain can be done. The discrete time system obtained with the trapezoid method, is stable if and

only if the continuous time system is stable. When the backward method is used, the condition is necessary but not sufficient.
Another mapping approach is the *bilinear with prewarping*, it is consequence of the trapezoid method and is obtained imposing that the discrete time transfer function must have exactly the same gain and phase of its continuous equivalent. In this case the relationship between $s = j\omega_0$ and z is the trapezoid rule or bilinear transformation:

$$s \rightarrow \frac{\omega_0}{\tan(\omega_0 T_s/2)} \frac{z-1}{z+1}. \tag{3.9}$$

Proportional Integrative Derivative (PID) compensator is usually used for buck converters regulation, it combines both PD and PI properties. The PD leads to better rejection of high frequency disturbances and is used both to improve the phase margin and extend the bandwidth of the feedback loop. Proportional Integrative (PI) compensator leads both to better rejection of low frequency disturbances and very small steady state error, because the low frequency loop gain is increased. The PID compensator or regulator, combines these properties in order two achieve the advantages of both approaches.
Let's call $G_{PID}(s)$ the PID compensator transfer function, it's composed by two zeros and one pole at s=0:

$$G_{PID}(s) = K_{PID}\left(1 + \frac{s}{2\pi f_{z1}}\right)\left(1 + \frac{s}{2\pi f_{z2}}\right). \tag{3.10}$$

The PID gain (K_{PID}) and the related zero positions can be chosen by considerations on the control to output structure of the converter, indeed the compensator is usually exploited to modify the dc gain, margins and bandwidth in order to obtain the best system dynamic. Increasing the dc gain, means to obtain a more precise output voltage, moreover bandwidth and margins of the closed loop system can be adjusted in order to improve the system dynamic.

3.2 Design considerations

Once that both the control to output transfer function of the buck converter and the control loop delay are modelled, the design of a closed loop converter can be addressed. As mentioned in the previous chapter the control to output transfer function in the continuous domain can be obtained from the small signal model after the circuit averaging analysis.
During the steady state the ADC works mainly around the zero error bin and the LSB_{ADC} is mainly related to the ripple constraints. Taking into account the Eq.1.8, for an output voltage of $5V$, a resonant frequency of $5kHz$ and a switching frequency of $449kHz$, the expected ripple voltage magnitude is in the range of few tens of mV
Just considering the ratio between the clock and the switching frequency, the resolution in counter based DPWM is automatically chosen by the Eq.3.3.
It is well know, that DA converters are affected by the The Zero Order Hold (ZOH). It is a mathematical model of the practical signal reconstruction done by a conventional digital to analog converter (DAC) and, describes the effect of converting a discrete time signal to a continuous time one by holding each sample value for one sample interval. This effect only occurs in real systems where the samples are converted back to the analog world through a filtering operation, hence this is not critical in dcdc converters. The DPWM is not performing any filtering action, but the some related problems could appear if the duty cycle changes

before the next duty cycle. In counter based DPWM this effects is not present, but has to be considered if any small signal variation is present.
The design of the feedback loop is mainly focused on the regulator structure. The compensator design is usually done with frequency considerations, it has to adjust the closed loop system dynamics. From the implementation point of view, it computes the digital control world for the DPWM. Once that the resolution of the DPWM has been decided, the MSB of the digital control world must be respected form the digital implementation of the converter.

3.2.1 PID Compensator

The compensator is usually designed in the continuous domain and then the equivalent in the Z-domain is computed (Sec.3.1.2). Its design is done considering the delayed control to output transfer function (Fig.3.2). The open loop considered transfer function, is composed by the product among the converter control to output, the ADC and the DPWM transfer functions:

$$G_c(s) = G_{vd}(s)G_{DPWM}(s)G_{ADC}(s)e^{t_d}, \tag{3.11}$$

where $G_{vd}(s)$, $G_{DPWM}(s)$ and G_{ADC} respectively refers to Eq.2.14,3.4 and 3.5.
To properly design the digital compensator an accurate model of the analog plant is required. The control to output transfer function $G_{vd}(s)$ is derived by averaging the converter signals over the switching period and, therefore, it is an approximate model of the converter. The model, thus, does not take into account of high frequency effects which cannot be neglected when designing digital dcdc converters with bandwidth higher than $f_s/20$. In order to maximize the convert bandwidth a standard approach is to fix the desired bandwidth $f_c = f_s/20$. The $G_{vd}(s)$ structure is mainly composed by two complex conjugated poles (Fig.2.6), the PID structure presented in Eq.3.10 can be set in order to increase the bandwidth up to $f_s/20$ and then adjust the system dynamic.
Referring to $G_{PID}(s)$ in Eq.3.10, there are two main approaches to chose the zeros position in order to increase the bandwidth of $G_c(s)$. Both f_{z1} and f_{z2} can compensate the conjugated pole pair of G_{vd}, otherwise can be placed close to the resonant frequency f_0 of the converter. Once that the zeros of the regulator are placed, the required bandwidth can be obtained adjusting the gain K_{PID}.

Delays on the control to output transfer function has been considered in Fig.3.5, for a system having $f_0 = 5kHz$ and a bandwidth of $f_c = 19.7kHz$. Considering a switching frequency $f_s = 449kHz$, a good compromise is to obtain a bandwidth close to $f_C = \frac{f_{sw}}{20} = 22.5kHz$. The green curve is the PID shape, in this case the two zeros are respectively at $0.7f_0$ and $0.9f_0$ and the bandwidth of $G_{vd}(s)G_{PID}$ is adjusted considering a PID gain $K_{PID} = 6000$. The obtained system bandwidth is of $fc = 25.1kHz$, while $14.3dB$ and 59.4Âř are respectively gain and phase margins. It can be observed as starting from an unstable system, the PID brings the system into a stability condition by adjusting both bandwidth and margins.
Once the controller has been studied the equivalent in the Z-domain can be easily computed in Matlab by considering the bilinear with prewarping numerical integration approach. Starting from:

$$G_{PID}(s) = 6000\left(1 + \frac{s}{2\pi 0.7}\right)\left(1 + \frac{s}{2\pi 0.9}\right), \tag{3.12}$$

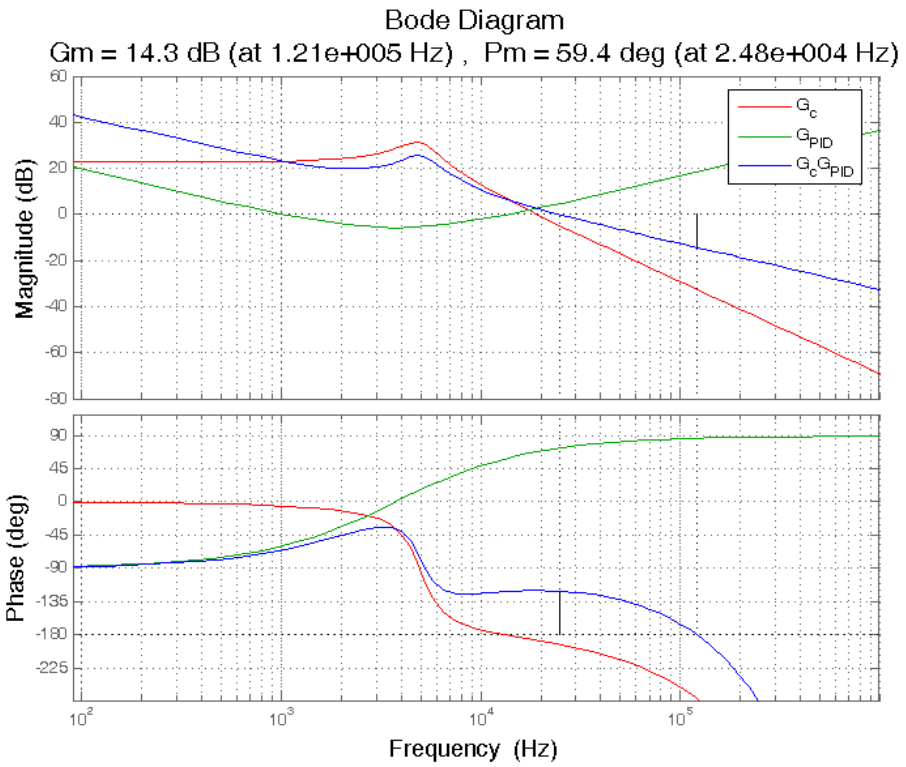

Figure 3.5: Control to output transfer function: PID effects.

its equivalent in the Z-domain with $f_s = 449kHz$ is:

$$G_{PID}(z) = 10^9 \left(\frac{1.31z^2 + 2.63z + 1.31}{z^2 - 1} \right). \tag{3.13}$$

From the implementation point of view the compensator has to compute every switching T_s the new high resolution control word $d_{hr}[n]$ starting from the ADC output error $e[n]$. Several implementation of $G_{PID}[z]$ can be considered, for a real implementation the best choice is always dealing between resource saving and computation time. The most used implementation come directly out from the continuous implementation of the PID regulator:

$$d(t) = K \left(e(t) + \frac{1}{T_I} \int_0^t e(\tau)\, d\tau + T_D \frac{de(t)}{dt} \right), \tag{3.14}$$

with a related transfer function in the S-domain:

$$G_{PID}(s) = K_P + \frac{K_I}{s} + K_D s, \tag{3.15}$$

Where the gains $K = K_P$, $K_I = \frac{K}{T_I}$ and $K_D = KT_D$ are respectively proportional, integrative and derivative coefficients. The expression of the converter in the Z-domain can be obtained from Eq.3.15 or more easily from the discrete expression of Eq.3.14:

$$d[n] = d[n-1] + k_p e[n] + K_i e[n] + k_d(e[n] - e[n-1]) = k_p e[n] + K_i \sum_{r=0}^{n} e[r] + k_d(e[n] - e[n-1]). \tag{3.16}$$

The transfer function in the Z-domain can be directly obtained from its discrete equivalent (Eq.3.16) whit the advantage of having the same gains:

$$G_{PID}(z) = K_p + K_i \frac{z}{z-1} + K_d \frac{z-1}{z}. \tag{3.17}$$

In this way, the well known parallel structure for the PID has been obtained in Fig.3.6.

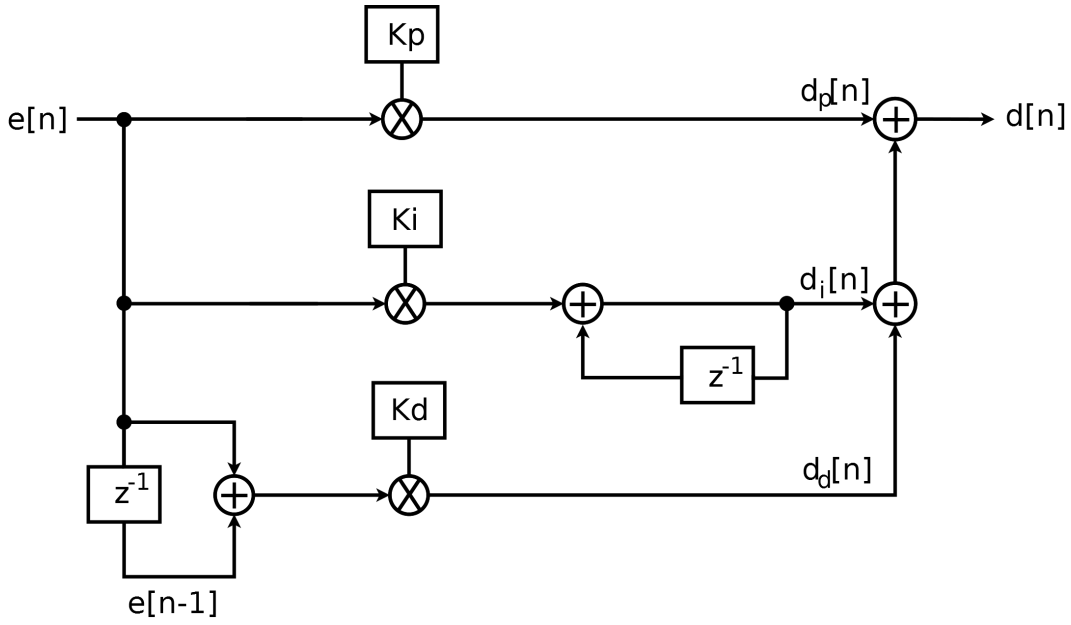

Figure 3.6: PID model: parallel structure.

The computation time is optimized in parallel structures because no delay is present on the signal path, but the request of hardware resources could be high due to multipliers use. A solution used to avoid the use of multipliers, is to exploit Look Up Tables (LUTs) where the pre-computed product between gain and error is stored.

3.3 Digital quantization effects

Digital system are always affected by quantizations effects. Considering the digital control loop introduced in Fig.3.1, the ADC, the DPWM and the digital compensator are sources of quantization noise. Truncation effects due to the finite resolution, can be considered the main difference between digital and continuous control. The problem of modelling quantization effects in digitally controlled buck converters has been addressed in [85, 86]. Digital control structure for SMPS are usually designed with fixed-point considerations, hence uniform quantizers are used because the resolution of control world is constant during the entire computation.

3.3.1 ADC and DPWM resolution

The AD converter introduces the first quantization step in the regulation loop, it outputs a finite resolution error $e[n]$. If the converter resolution is not enough the entire loop resolution loop can be compromise. Let's define n_{adc} as the digital resolution of the error and FSR

as the full scale range of the converter, the quantization step q_{adc} on the voltage error signal can be expressed as:

$$q_{adc} = \frac{FSR}{2^{n_{adc}}}. \tag{3.18}$$

During the steady state $e[n]$ is close to zero and variations less than $1LSB$ cannot be converted. To increase the resolution and avoid strong limit cycle effects, an ADC with zero error bin mapping can improve the resolution even if the same quantization step is used. Using zero error bin mapping characteristic (Fig.3.4), it means to don't map the zero as digital output world. During the steady state an error comprises between $\pm\frac{1}{2}LSB_{ADC}$ is always read by the compensator which always react. The dc value of the output voltage is related to $D = \frac{V_{out}}{V_{in}}$ which usually is not a natural number. During the steady state oscillations occur due to the finite resolution and are then considered in the zero error bin of the ADC and filtered out by the buck converter. In this way the compensator could react and adjust the duty cycle with a sinusoidal trend. These oscillation are averaged by the low pass structure of the buck converter and the resulting averaged dc value of duty cycle is then more precise even if the same resolution is kept. The LSB_{ADC} is usually chosen with considerations on the output voltage ripple in Eq.1.8, small ripple effects are obtained when the switching frequency is much more high then the resonant frequency.

The resolution of the DPWM is simple the resolution of a one step counter which has to be reset every switching step (see Sec.4.1.4). This reason the minimum number of bits is defined by the Eq.3.3 and the maximum is limited by hardware resources constraints. The high resolution control world $d_{hr}[n]$ is then truncated in a low resolution one $d_{lr}[n]$ to output $d(t)$.
Increasing the DPWM resolution more bits of the compensated duty cycle can be considered (bigger MSB) and consequently more resolution is reflected on d(t)(Fig.3.3).

3.3.2 Compensator resolution

Once that the discrete expression of the comparator has been obtained it has to be designed in the digital world with a finite resolution. If the number of employed bits is not enough the system could be unstable because zeros/poles position in the frequency domain has not been respected. Zeros, high and low frequency poles all require a high number otherwise they cannot be distinguished each other. Moreover, a large number of bits is needed especially when hardware multiplication exists in the implementation.
However the number of bits should be large enough to ensure the system stability, then both zeros and poles have to be distinguished. Assuming the system stability, is like to consider the compensator resolution so high that quantization effects introduced in the digital loop are only depending on the ADC and the DPWM. Indeed, the DPWM is mainly a counter, only n_{DPWM} bits (MSB) of the PID output are considered. Moreover, the considered input error is output from the ADC and if it results compromised the entire regulation will be compromised as well.

3.3.3 Limit cycles oscillations

Limit cycle oscillations (LCOs) are periodic oscillations of V_{out}, which occurs during the steady state and are not due the DPWM switching activity[85, 86]. These persistent oscilla-

tions present a frequency much lower than the switching frequency and are hard to predict. Non-linearities introduced by the ADC and DPWM quantizers could be source of LCOs.
There are two main kind of steady state LCOs, *static* and *dynamic*. Static conditions are mainly related to the loop resolution, when a dc solution for the output voltage does not exist the zero error bin is never reached. On the counterpart, dynamic conditions refer to the non-linear quantizers transfer functions (describing functions).

The static conditions are obtained assuming that quantization effects for the digital compensator can be neglected, in this case a dc value for the output voltage could or could not exists. Steady state cycling is then related to both ADC and DPWM quantization effects. The first condition can be intuitively understood. If there is no DPWM quantization level that maps the output voltage in the ADC zero error bin, then the system will exhibit limit cycle oscillations. It results that to avoid LCOs the $DPWM$ resolution has to be larger than the ADC one:

$$n_{dpwm} > n_{adc}. \tag{3.19}$$

The just mentioned condition (Eq.3.19) is not sufficient to avoid steady state cycling on the output voltage, further considerations have to be done regarding the integral gain of the PID compensator. If during a perturbation of the output voltage, the gain of the integrator K_i is not low enough to map the output voltage inside the zero error bin limit cycle oscillations will occur. The condition can be summarised in:

$$0 < G_{vd}K_i = V_{in}K_i > 0.5 \tag{3.20}$$

Dynamic conditions are less intuitive to find and involve complex mathematical analysis, them are mainly related to non-linear system stability theory and are here omitted. The LCOs are often exploited during the identification phase of the autoregulation algorithms (Cha:6). Static conditions are exploited in [135, 136, 139]and dynamic conditions in [25, 102, 104].

3.4 Delta Sigma modulator

Delta Sigma ($\Delta\Sigma$) modulation reminds the basic idea used in modern data converters. It exploits the feedback of the quantization error in order to improve the output resolution. This method usually encode high resolution signals into lower resolution ones. Referring to $\Delta\Sigma$-based AD, an high resolution analog signal is encoded in a digital signal using the error feedback (dithering). The digital low resolution signal changes faster than the higher resolution one and is averaged through low pass filtering.
This principle could be used not only during a data conversion, but also in a digital loop wherever it is needed to convert the high resolution word in a lower one. The considered digital control loop is showed in 3.7, the $\Delta\Sigma$ modulator can be inserted between the digital compensator and the DPWM. As mentioned, the DPWM introduces quantization effects on the high resolution word output from the compensator. In counter-based DPWM (Sec.4.1.4) the high resolution PID output $d_{hr}[n]$ (n_h bits) is truncated in a lower resolution one $d_{lr}[n]$ (n_l bits). For this reason, a $\Delta\Sigma$ modulator block is inserted between the digital compensator and the DPWM, in order to consider the quantization noise of $n_h - n_l$ bits. Its configuration for digital control loop is called *error feedback* (Fig.7.16), the quantization error introduced by the DPWM is considered during next switching cycles and added to high resolution control word.

The $\Delta\Sigma$ modulator works exploiting three main basics:

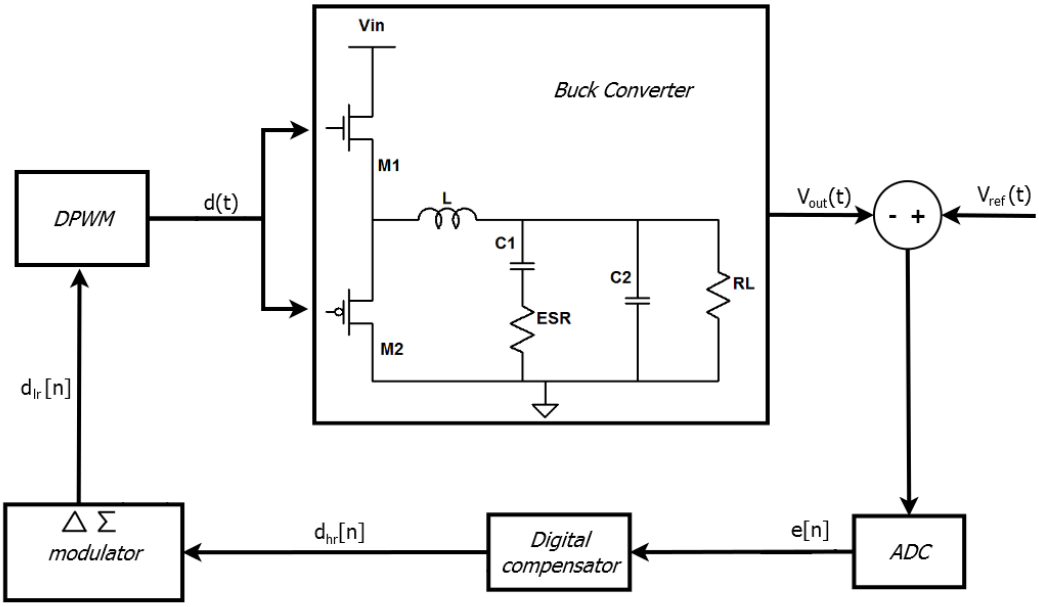

Figure 3.7: Digitally controlled buck converter: model with ΔΣ modulator.

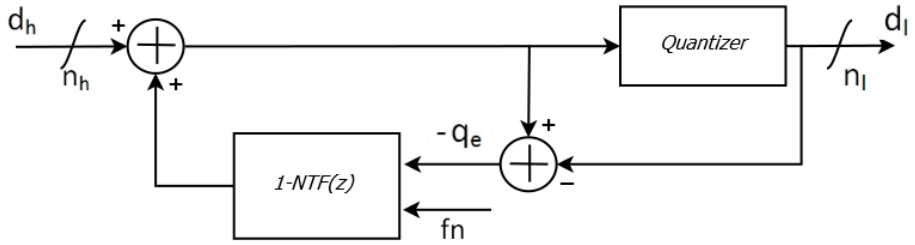

Figure 3.8: ΔΣ modulator: error feedback configuration.

- **Dithering**. The shaped quantization noise is added on the signal path.
- **Oversampling**. During the steady state sinusoidal duty cycle variations are obtained due to the quantization effects. The quantization noise can be considered an independent additive white noise uniformly distributed inside $[\pm\frac{LSB_{DPWM}}{2}]$. If f is the frequency of this sinusoidal contribution and, the sampling frequency f_s is much higher then the Nyquist rate $(2f)$, then signal at frequency f results to be oversampled. This assumption permits to reduce the noise at the frequency of interest. The same quantization noise, is distributed in the range $[0:2f]$ for Nyquist converters while, is distributed over a larger range $[0:f_s]$ for ΔΣ modulators.
  The oversampling is implicit in digitally controlled buck converters, indeed, the switching frequency f_s is much higher then the input signal frequencies that are limited to the output filter bandwidth.

- **Averaging**. Due to dithering and oversampling, the lower resolution signal changes faster then the higher resolution input. The low pass characteristic of a buck converter automatically permits to average faster duty cycle variations.

3.4.1 Error feedback configuration

The ΔΣ modulation for digital loops is used to improve the resolution of the digital word each time we move from a high resolution word to a lower one. The error feedback configuration is shown in Fig.7.16, this configuration is mainly composed by two adders, one quantizer and one high pass filter or noise shaper (NTF(z)). Three main reasons make very attractive the ΔΣ:

- unitary signal transfer function, indeed none delay is added on the signal path and no phase contributes are added to the control to output converter transfer function.
- resolution improving of the DPWM.
- resource saving oriented device. From the digital design point of view it is mainly composed by a high pass filter.

A digital design of the error configuration modulators has been presented in [51] and has been integrated in digitally controlled SMPS in[65, 61, 37]. The DPWM resolution improving due to the modulator and ADC zero error bin has been presented in[138] for low resolution controllers.
The order of the simulator is defined by the noise transfer function (NTF(z)) described in 3.4.2. A digital implementation of a second order modulator is presented in 3.9. This error feedback configuration is ideal for the digital implementation, it exploits adders which are difficult to implement in the analog world. Because quantizers are free in a digital implementation, the hardware costs are mainly depending on the noise shaper order and the considered resolution (n_h and n_l bits). Both the noise transfer function (NTF(z)) and signal transfer function (STF(z)) can be obtained from the model shown in Fig.3.9:

$$Y(z) = X(z)+(1+H(z))*E_2(z) = X(z)+(1-2z^{-1}+z^{-2})*E_2(z) = X(z)+(1-z^{-1})*E_2(z). \quad (3.21)$$

The resulting signal transfer function from Eq.3.21 is unitary, therefore none delay on the signal path is added:

$$STF(z) = \frac{Y(z)}{X(z)} = 1, \quad (3.22)$$

while the second order noise shaper is:

$$NTF(z) = \frac{Y(z)}{E_2(z)} = (1-z^{-1})^2. \quad (3.23)$$

The discrete time representation of the model shown in Fig.7.16 is:

$$d_l = d_h + NTF(z) * q_e. \quad (3.24)$$

The DPWM resolution improving is demonstrated in [65, 61, 37, 138, 76] and can be easily intuitively justified. Let's refer to Fig.3.9, the high resolution digital representation is an n_h bits length word, it is composed by both most and least significant bits which are respectively

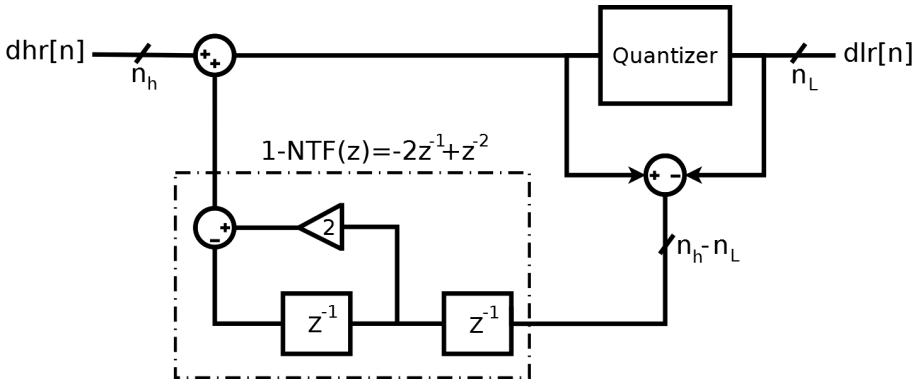

Figure 3.9: $\Delta\Sigma$ modulator: 2nd order error feedback configuration.

n_l and $n_h - n_l$ bits length words. The quantized low resolution word is composed only by the MSB and the introduce quantization error is the LSB. Let's consider a non natural number for the high resolution duty cycle during the steady state:

- Without modulator, the quantization error is neglected and the low resolution word presents a sinusoidal behaviour inside one LSB_{DPWM}.
- Introducing a $\Delta\Sigma$ modulator the quantization noise contribution inside the LSB_{DPWM} is considered (dithering) and results distributed in the range of frequencies up to the switching frequencies (oversampling). In this way the duty cycle change faster with different frequency contributes, while the noise can be high pass filtered (next section) and added on the signal path (dithering).

In order to obtain a duty cycle value closer to the desired one, this fast low resolution duty cycle need to be averaged. In digitally controlled SMPS the averaging function is automatically done through the low pass characteristic of the output filter. During the steady state the output voltage have to stay inside the zero error bin of the ADC, this means that the resolution improvements n_x in function of n_h can be defined in terms of maximum voltage error[76]:

$$n_x = \max_{n_h \geq n_l+1} \left[\min \left((n_h - n_l), log_2 \left(\frac{V_i n / 2^{n_l}}{|V_{err}|_{max}} \right) \right) \right] \tag{3.25}$$

3.4.2 Noise transfer function

The noise transfer function NTF(z) defines the order of the modulator. The modulator feedback is a discrete time filter $1 - NTF(z)$(Fig.3.9), it operates at the same rate f_s of the DPWM and processes $n_h - n_l$ bits (quantization error). The Noise Transfer Function is usually a high pass filter used to shape the quantization noise at higher frequencies. In order to evaluate both the best solution and the effect of the modulator loop insertion, different noise shaper structures has been analysed in[76].
Different noise shaping structures has been shown in Fig.7.20, together with a buck converter control to output transfer function . Considered noise transfer functions are first order ($NTF_1(z) = 1 - z^{-1}$), second order ($NTF_2(z) = (1 - z^{-1})^2$), third order ($NTF_3(z) = (1 - z^{-1})^3$)

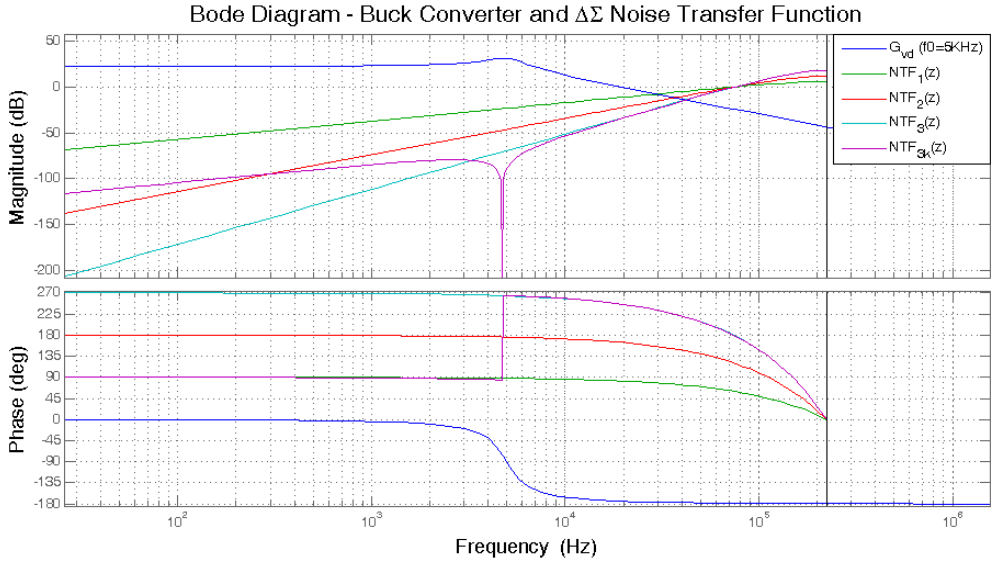

Figure 3.10: Control to output and noise transfer functions.

and third order modified. The third order modified transfer function is based on a third order structure:

$$NTF_{3k}(z) = (1 - z^{-1})(1 - kz^{-1} + z^{-2}), \tag{3.26}$$

where $K = 2cos(2\pi\frac{f_n}{f_{sw}})$ permits to insert a notch effect at $f = f_n$. For each order of the $NTF(z)$, the noise is suppressed at low frequencies. For any structure the power noise is the same, but how the order increases the low frequency contribute is lower the high frequency one increases. In the third order modified modulator the zero position could be adjusted attenuating the noise around at the resonance frequency of the output filter. The first and second order noise shaper, introduces higher noise level at the resonance frequency of the output filter without killing high frequencies noise. The third order NTF is the best compromise, furthermore, it prevents non-linearities due to the interaction between ΔΣ modulators and the DPWM quantizer. This interaction can produce limit cycles contributes, which result in significant spectral spikes (idle tones). When the quantizer output is periodic, idle tones with frequency $f = \frac{f_{sw}}{2^{n_h - n_l}}$ are generated, the third order modulators avoid this issue[76].

Chapter 4

Digital control feedback prototype

A Matlab/Simulink based modelling technique, has been introduced in Cha.2 to characterize the open loop configuration of the buck converter. Moreover, this approach has been to verify the introduced theoretical considerations. The same approach is used in this chapter to model and verify the digitally controlled closed loop power supply.
Theoretical considerations for digitally controlled buck converters have been drawn in Cha.3 considering, the introduced loop delay, the quantization effects and the limit cycles issues. Main blocks as ADC, PID, and $\Delta\Sigma$ have been presented with design considerations here exploited during the close loop system realization. In Matlab-based approaches, digital control modules are described as functions which are integrated into Simulink blocks. The entire feedback modelling flow, moves from a floating point implementation to a fixed-point model which can be used as reference model during the HDL (Hardware Design Language) encoding of the digital control. The FPGA prototyping of the control feedback is described and the hardware-software (VDHL-Matlab) FPGA-based co-simulation model is presented and characterised.
Results in this chapter have been considered in order to validate the all presented models up to the digital control VHDL design. Float and fixed-point Matlab/Simulink models, are compared in order to define the digital control loop resolution. Hardware/software co-simulation results are presented to validate the VHDL-coded control feedback. With this kind of approach, the closed loop configuration can be obtained exploiting Matlab/Simulink models for the analog blocks (ADC and buck converter) while the digital part is directly mapped on a Virtex6 FPGA. In this way the hardware testing is verified avoiding the use of user coded testbenches. From syntheses, results a maximum frequency of $124.748 MHz$. The hardware resource usage is summarised in Tab.4.3 and Tab.4.2, which respectively refer to the digital control resources usage and the FPGA resources usage.

4.1 Matlab/Simulink closed loop model

A user defined floating point Matlab/Simulink closed loop system is presented in Fig.7.15. Buck converter and gate driver blocks introduced in Cha.2 are controlled through the digital feedback. The entire implementation is based on the theory presented in the previous chapter. A zero error bin mapping ADC has been used, both third order $\Delta\Sigma$ and DPWM

counter-based are used to compute the low resolution duty cycle $d_{lr}[n]$. A PID-based compensator is used to compute the high resolution control word starting from the error $e[n]$. Parameters configuration, is the same in Sec.2.3 to model the open loop buck converter ($f_0 = 5kHz$ and $f_s = 449kHz$). The switching frequency is ensured by setting a DPWM counter from 0 to $\frac{f_{clk}}{f_s} = 156$ for clock frequency of $70MHz$. The clock frequency has been emulated through the fixed-step Matlab solver configuration. A reference voltage $V_ref = 3.3V$ and input voltage $V_{in} = 14V$ are used in this implementation. In Fig.4.2 is presented the out-

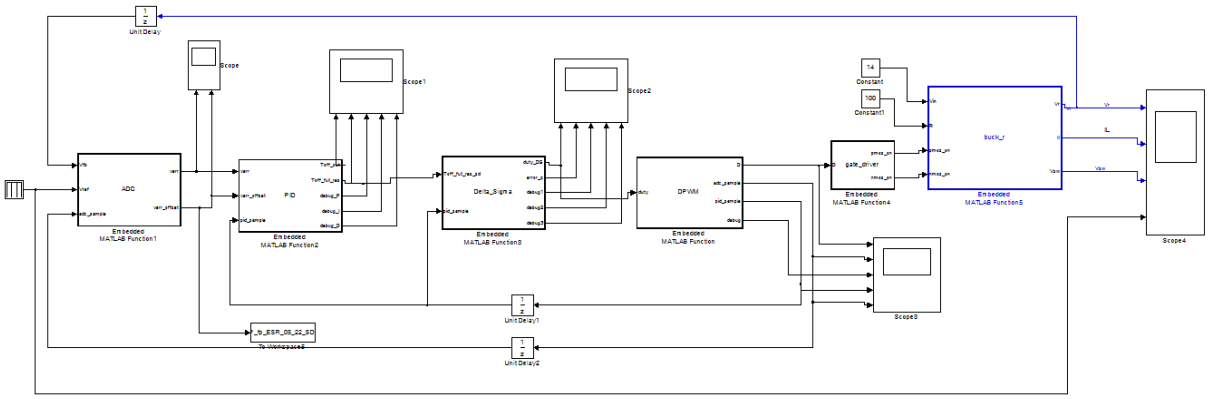

Figure 4.1: Digitally controlled buck converter: Matlab/Simulink model.

put voltage (called V_r as in model in Fig.2.10) and the reference voltage, for the fixed-point model implementation. As desired, the output voltage is equal to the reference one during the steady state. Moreover, a soft start-up is obtained thanks to the linear evolution of the reference voltage V_{ref}. In Fig.4.3 is shown a zoom taken during the steady state. It is visible how the output voltage is very close to desired value. Frequency contributes during steady state are far from a single tone contribute (LCO), this due to the ADC zero ero bin, the $\Delta\Sigma$ modulator and the high resolution PID. As mentioned in the previous chapter, all these contributes brings to be close to a desired output voltage even if an ideal duty cycle of $D = V_{ref}/V_{in} = 0.2357$.

Having a fixed-point reference model can be very useful when it is desired to move to a digital hardware implementation of the control law. In the next section the model presented in Fig.7.15 is described comparing block by block the floating with the fixed-point implementations.

Moving to a fixed-point implementation starting to a floating, is usually done by a dedicated Matlab toolbox. An alternative way to implement the fixed-point behaviour is to insert in the floating point implementation some devices:

- If we want to work with a 9 bit signed number we must consider an integer within ± 256. Operations like additions, subtractions and multiplications are simple to handle and possible overflows have to be evaluated by floor functions.
- In a digital implementation shifting operations are fundamentals, the Matlab equivalent can be obtained by handling these operations:
- left shit by K of y is $y = 2^k$

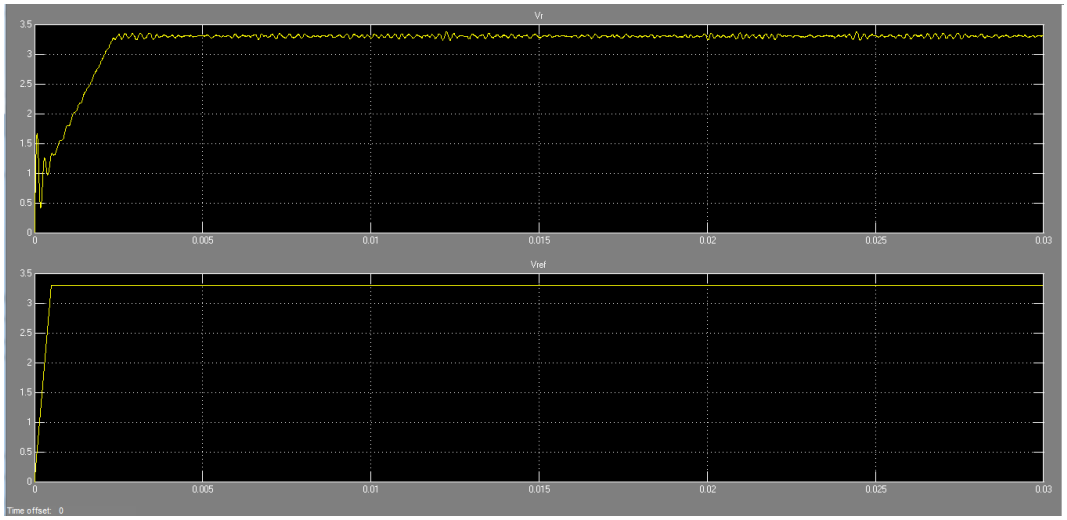

Figure 4.2: Fixed-point closed loop model: output voltage.

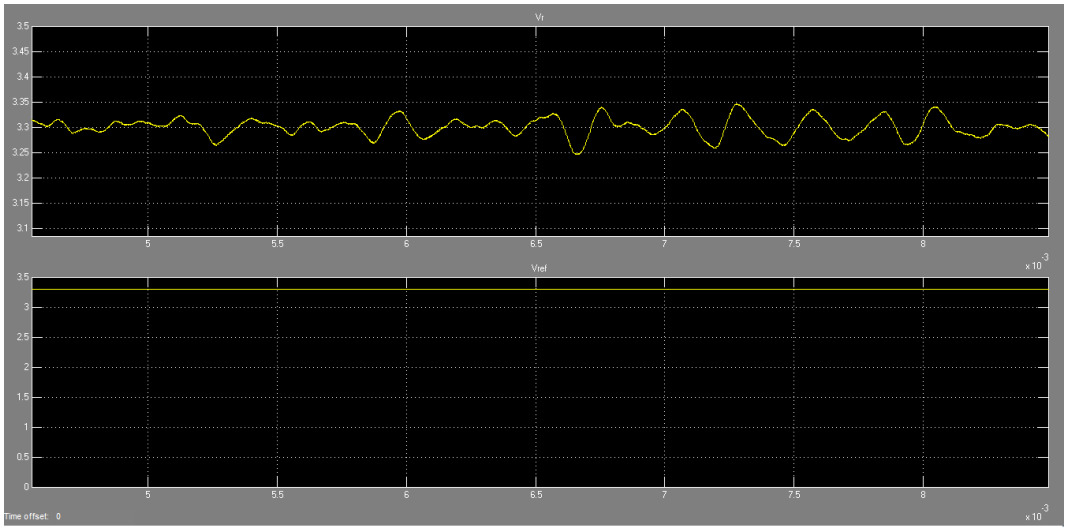

Figure 4.3: Fixed-point closed loop model: zoomed output voltage.

- right shit by K of y is $y = 2^{-k}$
- Overflows handling is related to the number which has to be checked:
  - For N bits unsigned number **x**, the modulus operation (mod) can be used: **x**= mod(**x**, 2^N)
  - For N bit signed number a function have to be introduced:

```
function y = wrap_around_signed(x,N);
      while x > (2^(N-1)-1)
              x = x - 2^N;
      end
      while x < (-2^(N-1))
            x = x + 2^N;
      end
      y = x;
    end
```

    The function *wrap_around_signed* takes as input the number **x** and outputs **x** only if this is less of the clamping values $\pm 2^{N-1}$.

- To extract 5 bits **REGISTER**(7 downto 3) from a register **REGISTER**(7 downto 0):

```
x = floor(REGISTER/$2^3$);
x = mod(\textbf{x},$2^5$);
```

Normal Matlab instructions are enough for building a fixed-point model if these properties are exploited. During algorithms implementation, for every computation both overflows checking and integer part extraction (using the *floor()* Matlab function) permit to obtain a fixed-point model.

4.1.1 ADC modelling

To consider a zero error bin ADC permits to achieve higher resolution for digitally controlled dcdc converters and, furthermore, a non-zero scheme for output voltage error coding increases the LCOs frequency beyond the resonant frequency f_0[134]. For this reason the modelled ADC presents the same theoretical approach considered in Cha3, where the zero error bin configuration is preferred. The Simulink block is presented in Fig.4.4, its inputs are the buck converter output voltage or feedback voltage V_{fb}, the reference voltage and the start of conversion (*adc_sample*) which comes from the DPWM. The implemented Matlab func-

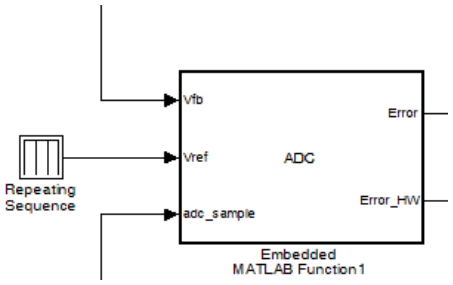

Figure 4.4: Simulink block: ADC model.

```
%FLOATING POINT ADC
if (adc_sample == 1)
    if p.Vq ==0
        samples(1) = Vfb-Vref;
    else
        samples(1) = floor((Vfb-Vref)/p.Vq)*p.Vq;
            if samples(1)>=0
                samples(1) = samples(1) + 1*p.Vq*(1/8);
            else
                samples(1) = samples(1) + 1*p.Vq*(1-1/8);
            end
    end
end

%clamp the error
if samples(1) > 7* p.Vq
    verr= 8* p.Vq;
elseif  samples(1) < -8* p.Vq
     verr= -8* p.Vq;
else
     verr=samples(1);
end
```

```
%FIXED-POINT ADC
if adc_sample == 1
        % SAMPLE
        error_temp = Vfb - Vref;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%% Flash ADC 4 bits
        %%%%%%%%%%%%%%%%%%%%%%%%%%%
        error_temp = floor(error_temp / p.Vq);
        if error_temp > 7
            error_temp = 7;
        end

        if error_temp < -8
            error_temp = -8;
        end


        error_temp = error_temp*8;  % append 3 LSB to 0

%         % stabilize the 0 error bin ....
        if error_temp >=0
            error_temp = error_temp +1;
        else
            error_temp = error_temp +7;
        end
end
```

Figure 4.5: ADC Matlab function: floating point (left side) and fixed-point (right side).

tion for a 4-bit ADC is presented in Fig.4.5 both for floating (left hand side) and fixed-point (right hand side) version. The quantization step Vq is defined as a converter parameter and is equal to $15mV$. The ADC is designed to convert a small voltage range (window) and his zero error bin is centred around the target output voltage (window ADC) during the steady state.

Floating point

When the start of conversion is high (*adc_sample*), the voltage error *verr* is computed simply mapping the error $V_{fb} - V_{ref}$ in one among the sixteen levels comprises between $[7 * 15mV, -8 * 15mV]$. If the value exceeds this voltage levels a clamping function is coded. The mapping of the error voltage in the ADC scale is simply computed through calculating the related level and multiplying per V_q the integer value (raw six of floating point point implementation). The zero error bin considered LSB is $\pm V_q/8$, the entire ADC scale presents an offset due to the zero error bin equal to $1/8$ and the null voltage error corresponding code is not contemplated.

Fixed-point

The fixed-point implementation is obtained removing the V_q voltage from the floating point mapping function. In raw 9 of fixed-point code, the error is an integer value in the interval $[7, -8]$ and a clamping function is here considered in order to do not exceed this interval. When the start of conversion is high, the error $e[n]$ (indicated as *error_temp*) is computed and every time multiplied for 2^3 in order to gain 3 bits in terms of resolution. This means that the error input to the PID will be an integer in the interval $[56, -64]$ and that $LSB_{ADC} = 8$. The zero error bin is considered, the digital code do not map $e[n] = 0$ and only considers levels ± 1 around the zero.
It can be noticed that in both configurations the zero error bin is equal to $LSB_{ADC}/8$, indeed

in the fixed point this assumption is still valid thanks to the added 3 bits. The ADC behaviour for a linear evolving input is shown in Fig.4.6. Just removing the quantization step and multiplying the level for 2^3, the same result is obtained for the floating point model.

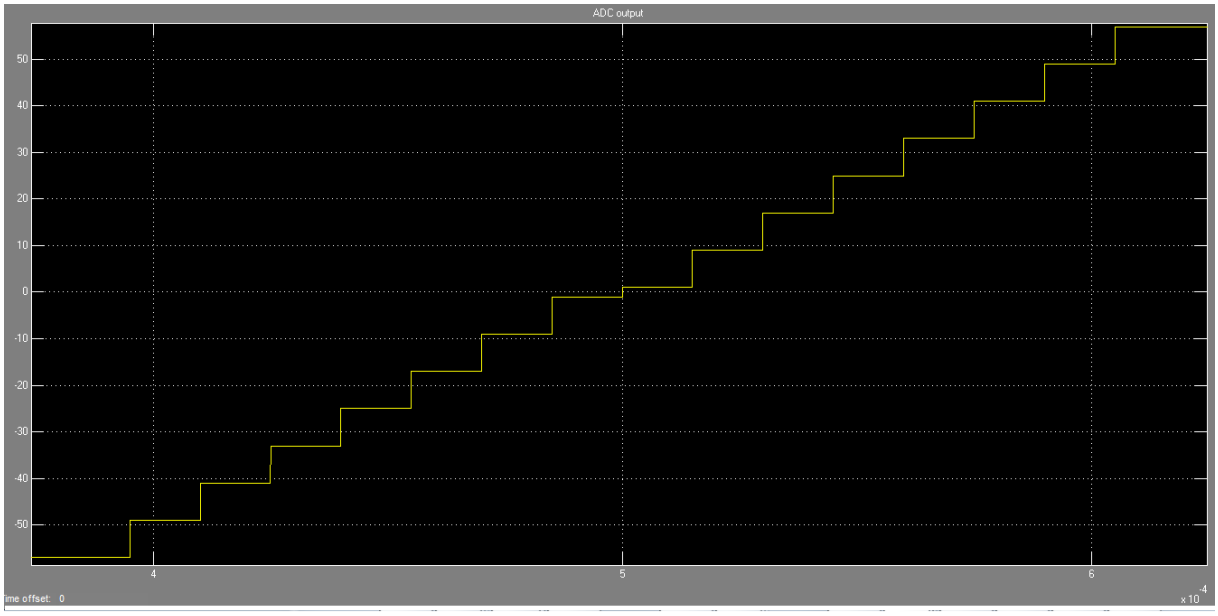

Figure 4.6: Matlab/Simulink: fixed-point ADC output characteristic.

4.1.2 PID compensator modelling

The considered PID parallel structure has been described in Sec.3.2.1 of the previous chapter. This approach comes from the Eq.3.16, the main advantage is that starting from the discrete PID expression its equivalent in the Z-domain can be obtained without gains factor computation. Moreover, the equivalent in the S-domain can be obtained by using the backward relationship (Eq.3.9).
The digital compensator computation is activated through the signal *pid_sample* generated by the DPWM every switching step. In the implemented model the ratio, $f_0/f_s = 156$ is called *OS_factor* and is defined as system parameter. Compensator gains are considered as parameters of the system and read out from the *parameters* structure. The PID Matlab modelling is shown in Fig.4.7, where both floating (left hand side) and fixed-point (right hand side) models can be distinguished.

Floating point

When the computation is activated the PID parallel algorithms is applied to the ADC output error voltage V_{err} (raw 3). After every new duty cycle computation three steps are always repeated. Firstly the new duty cycle has to be clamped to prevent overflows because it cannot exceed the maximum value (*OS_factor*). Last two steps are respectively related to both to integral part and error updating for the derivative part. The integral part updating is necessary because, the computation involves all previous computed integral contributes (Eq.3.16). The

```
%FLOATING POINT PID
if (pid_sample == 1)
    duty_f = p.Kgain* (integrator + p.KP * Vadc + p.KD * (Verr -Verr_z1)) * parameters.OS_factor;
 %CLAMP duty
    if duty_f <= 4
        duty_f = 4;
    end
    if duty_f >= parameters.OS_factor-4
        duty_f = parameters.OS_factor-4;
    end
    duty_v = duty_f;
    %integrator
    integrator = integrator + p.KI * Verr;
    Verr_z1 = Verr;
end
duty=duty_v;
```

```
%FIXED-POINT PID
if pid_sample ==1
      % Proportional mantissa
      Proportional = (parameters.CTRL.KP_m * Error) ;
      Proportional = wrap_around_signed(Proportional, 32);
      % Differential mantissa
      D = (Error - Error_z1);
      Differential = (parameters.CTRL.KD_m * D) ;
      Differential = wrap_around_signed(Differential, 32);
      % Integral mantissa
      Toff_full_res = Integral;
      Toff_full_res = wrap_around_signed(Toff_full_res, 32);
      % Proportional exponent
      Toff_full_res = Toff_full_res + Proportional * 2^(11-parameters.CTRL.KP_e);
      Toff_full_res = wrap_around_signed(Toff_full_res, 32);
      % Differential exponent
      Toff_full_res = Toff_full_res + Differential * 2^(11-parameters.CTRL.KD_e);
      Toff_full_res = wrap_around_signed(Toff_full_res, 32);
      % update integral part
      Integral = Integral + (parameters.CTRL.KI_m * Error) * 2^(11-parameters.CTRL.KI_e);
      Integral = wrap_around_signed(Integral, 32);
      % CLAMPING integral part to avoid LCO
      if Integral < 0
          Integral =0;
      end
      if Integral > parameters.OS_factor* 2^13
          Integral =parameters.OS_factor* 2^13;
      end
      % CLAMPING duty cycle PWM modulator uses only the MSBs
      Toff = floor (Toff_full_res / 2^11);
      if Toff < 4
          Toff = 4;
          Toff_full_res = 4*2^13;
          Qnoise =0;
      end
      if Toff > parameters.OS_factor -4
          Toff = parameters.OS_factor -4;
          Toff_full_res = (parameters.OS_factor -4)*2^13;
          Qnoise =0;
      end
      % update register for differentiator
      Error_z1 = Error;
end
Toff_dpwm = Toff;
Toff_full_res_sd = Toff_full_res;
```

Figure 4.7: PID Matlab function: floating point (left side) and fixed-point (right side).

error updating is related to the derivative contribute, indeed every step the differential part is computed considering both the actual error and the previous input error. Compensator gains are stored in the *parameters* structure and in this implementation are natural numbers in the interval [1,16].

Fixed-point

This implementation refers to a down-counter DPWM (Fig.4.11), for this reason the computed low resolution duty cycle is called *Toff*. The model is organized as the floating point and the duty cycle computation is followed by the clamping and the updating phase. The high resolution duty cycle ($d_{hr}[n]$) is called *Toff_full_res*, the model output $d_{hr}[n]$ to the $\Delta\Sigma$ modulator and also the low resolution duty cycle when only the DPWM is used. During the entire computation typical fixed-point function are used. In this example a full resolution of 32 bits is used, but considering 4 bits respectively for mantissa and exponent of the PID gains a resolution of 28 bits is resulted enough to approximate the results of the floating point model. Because fixed-point compensator gains are organised in mantissa and exponent, the entire computation of the full resolution have to be divided in two phases. Firstly mantissa

contributes are computed and then the exponent parts are integrated in the computation. The full resolution word is composed with 12 bits LSB, indeed the low resolution duty cycle is obtained in raw 30 where a right shift of 12 bits is computed. Clamping is done both on high and low resolution duty cycle, the integral part has been clamped as well to avoid LCOs. Hence, a digital compensator with 28 bits MSB and 12 bits LSB has been modelled. Compensator gains are organised with 4 bits both for mantissa and exponent, the model can output d_{lr} and d_{hr} which corresponds to the off time for down-counter DPWM.

4.1.3 Delta Sigma modulator modelling

The modulator modelling, refers to the third order error feedback configuration which avoid idle tones. The main function of this block is the *dithering*, the output duty cycle is computed considering also previous quantization effects. Because differences between floating and fixed-point $\Delta\Sigma$ are just related to overflows checks and floor functions, the model description is addressed only for the fixed-point implementation in Fig.4.10. The main different between fixed and floating point representations is related to the source of quantization noise. Because the DPWM is a one-step counter, it introduces always a quantization effects on the high resolution duty cycle. The PID output in the floating point implementation is a real number, the introduced quantization noise is related to the *floor()* function applied to computed the low resolution duty cycle. On the counterpart, the quantization noise in the fixed-point representation is equal to 12 bits LSB considered during the high resolution compensator computation. The output comparison between fixed and floating point implementations is shown n Fig.4.8, the considered resolution (28 bits MSB and 12 bits LSB) ensures a very good match. This result is obtained for a constant input applied to the Matlab/Simulink model presented in Fig.4.9.

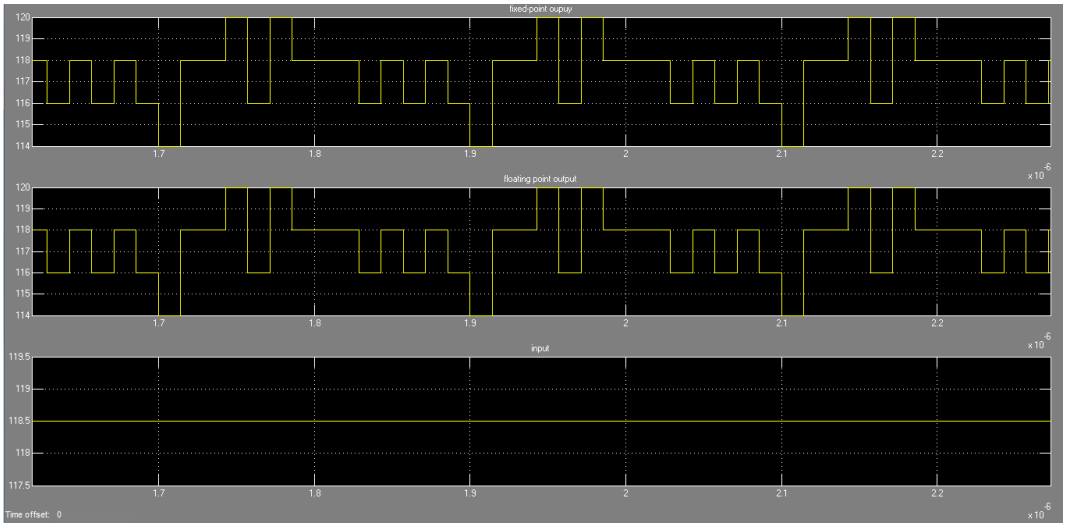

Figure 4.8: Matlab/Simulink: $\Delta\Sigma$ floating and fixed-point output comparison.

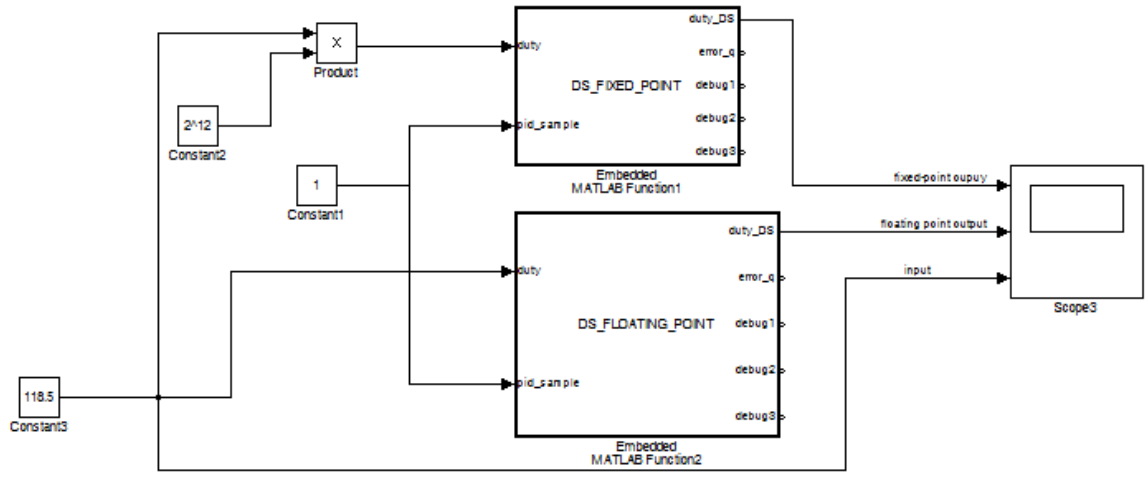

Figure 4.9: Simulink ΔΣ blocks: floating and fixed-point comparison.

Fixed-point

Every switching step the computation is enabled by the DPWM through the signal *pid_sample*. The modulator does not add any delay on the signal path, for this reason the enable computation signal can be the same used for the digital compensator. In this model the overflow is checked over a parametric number of bits through the parameter *sd_res*, the considered number of bits is the same of the compensator resolution (28 bits).
The first computation step is the dithering consideration, the full resolution duty cycle (*duty_st*) is computed taking into account the shaped quantization noise of previous step (*shaped_error_Z1*). Once that *duty_st* has been computed, the low resolution duty cycle for the DPWM is obtained just eliminating the LSB (*q_error*). This quantization error is saved in the variable *q_error_Z1*, it is the component z^{-1} of the next noise shaping computation (*shaped_error*).

At every step the noise shaping computation involves three previous quantization errors (Eq.4.1)and is saved in the variable *shaped_error_Z1* in order to be ready for the dithering during the next cycle.

$$1 - NTF(z) = 3z^{-1} - 3z^{-2} + z^{-3}. \tag{4.1}$$

The *shaped_error* computation is based on a three entries vector (*error_vect*), which is updated after every noise shaping computation in order to prepare z^{-1},z^{-2} and z^{-3} quantization error elements for the next computation. The duty cycle is computed considering the previous noise shaper output (*shaped_error_Z1*), at the same time the next noise shaper output (*shaped_error*) can be already computed because it is based on previous quantization errors.

```
%FIXED-POINT Delta Sigma
if pid_sample == 1
    %dithering
    duty_st = duty2q-(shaped_error_Z1);
    duty_st = floor(duty_st);
    duty_st = wrap_around_signed(duty_st,p.sd_res);
    % oquantized output for the DPWM
    Toff_low_res_sd = floor(duty_st/2^11);
    %recordign the quantization noise (input to 1-NTF(z) block)
    q_error = duty_st - Toff_low_res_sd*2^11;
    q_error = floor(q_error);
    q_error = wrap_around_signed(q_error,27);
    %third order NTF(z)
         shaped_error = (3*error_vect(1)-3*error_vect(2)+error_vect(3));
         shaped_error=floor(shaped_error);
         shaped_error = wrap_around_signed(shaped_error,p.sd_res);
    %error updating
    q_error_Z1 = q_error;
    error_vect_1 = error_vect(1);
    error_vect_2 = error_vect(2);
    % Error vector
    error_vect = [q_error_Z1 error_vect_1 error_vect_2];
end
shaped_error_Z1 = shaped_error;
error_q = q_error;
duty_DS = (Toff_low_res_sd);
```

Figure 4.10: ΔΣ Matlab function: fixed-point.

4.1.4 DPWM modelling

The DPWM is a one-step counter, then there is not distinction between floating and fixed-point implementation. If a down-counter from *OS_factor* to 0 is considered, the computed duty cycle represents the off time. Let's refer to Fig.4.11, for *OS_factor*$-d_{lr}[n]$ clock cycles the square wave is high (on-period) and for duty cycle clock periods the is equal to 0. The DPWM outputs also signals needed to enable control word computations every switching cycle, this counter can be seen as the main control of the system. The entire computation latency has to be then the switching time (Fig.4.11). The fixed-point computation does not take into account the real hardware delay, indeed when the computation is enabled just one fixed-step (T_{clk}) is needed. This reason only two clock cycles are need for the entire fixed-step computation, the *pid_sample* activates the compensator one time per cycle after that the ADC start of conversion (*adc_sample*) is high and $e[n]$ is computed. In Fig.4.12 is presented the model used to compare fixed and floating points models in order to very the digital resolution. The open loop structure of the controller comprises the ADC, the PID and ΔΣ, the model has been realised to emulate a steady state condition and to validate the chosen loop resolution. Both ADC models receive as input a constant $3.3V$ reference voltage and a feedback voltage V_{fb} which a sinusoid with unitary magnitude and dc value equal to the reference voltage. As described the compensator compute the high resolution duty cycle, the modulator adds quantization noise contributes and generates the low resolution duty cycle ($Toff$) for the DPWM. The result of the simulation is shown in Fig.4.13. The resolution for the fixed point implementation can be considered sufficient, both models present the same behaviour and

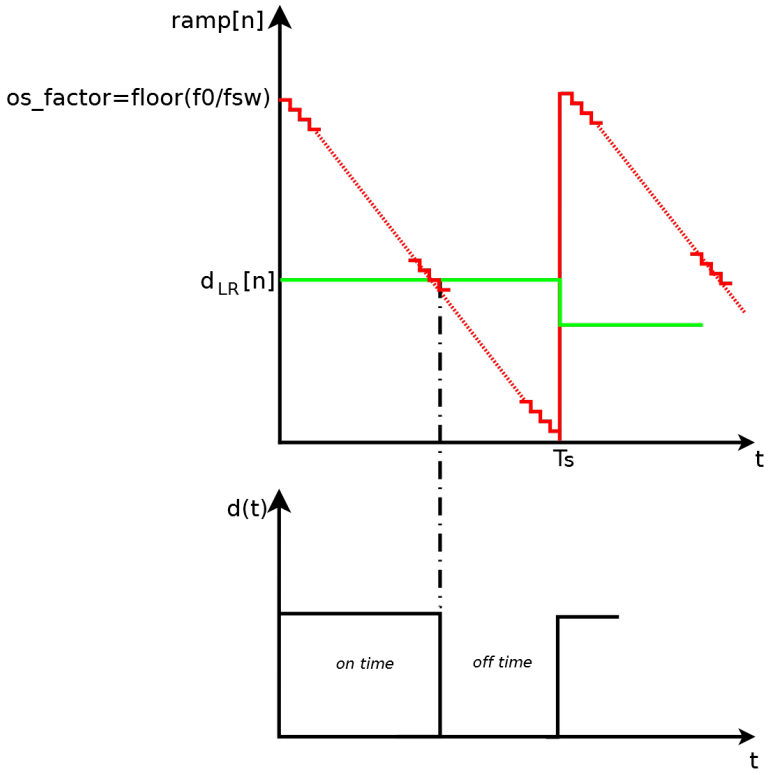

Figure 4.11: DPWM waveforms.

the fixed-point implementation can be used as reference model for the hardware implementation. The closed loop fixed-point model has been presented in Fig.7.15, the number of bits

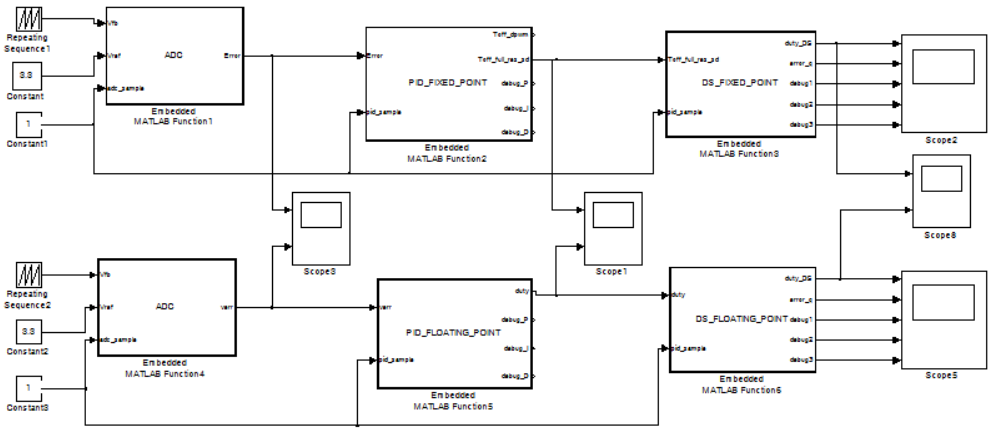

Figure 4.12: Matlab/Simulink floating and fixed-point comparison: model.

using for the regulation loop has been fixed through comparisons with the floating point model. Once comparable results has been obtained (Fig.4.13), the hardware realisation of the

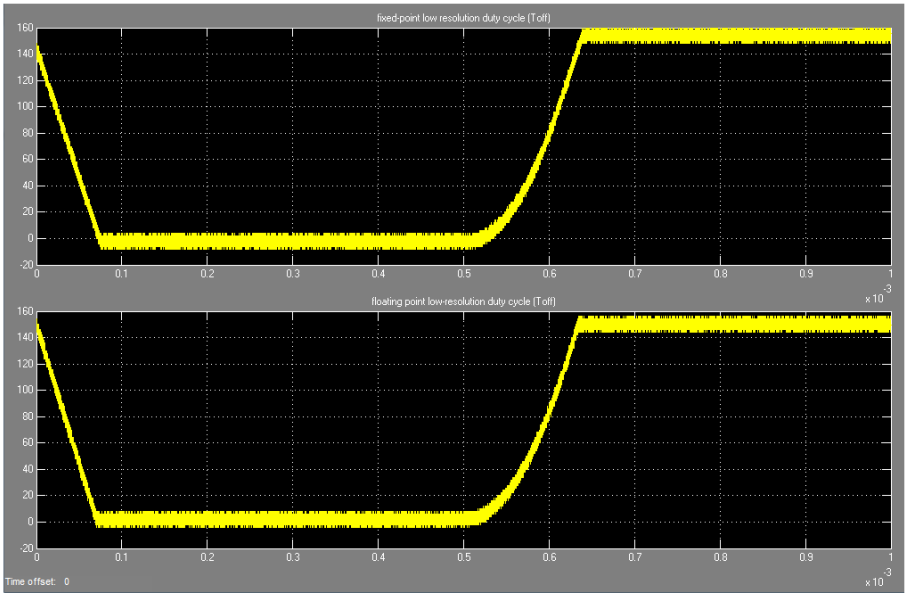

Figure 4.13: Matlab/Simulink floating and fixed-point comparison: Digital resolution tuning.

digital regulation loop can be designed referring to the fixed-point closed loop model.

4.1.5 PID tuning tool

All theoretical consideration addressed in the previous chapter and the fixed-point PID model can be used to realize a GUI (Graphical User Interface) to quickly chose the compensator gains in order to adjust system bandwidth. In Fig.4.14 is shown the *gui_buck* user interface, where it is possible to choose al parameters related to a buck converter and the compensator gains. Once parameters have been chosen, pushing on the buttom *Calculate Bode Diagram* the system automatically runs the related relationships and plot the plant both in S-domain and Z-domain, the compensator structure and the open loop transfer function ($G_{PID}(s)G_{vd}(s)$). The buck converter transfer function is calculated with non-idealities, the on resistance of the power-MOS and the ESR effects can be included. Moreover a dual capacitor load can be evaluated, with the possibility to model the ESR contribution for both capacitors. The global parameters include the input voltage and desired output voltage, while the switching frequency is calculated automatically when the clock frequency and the division factor (*OS_factor*) have been selected. Compensator gains are related to the fixed-point implementation, 4 bits numbers respectively for mantissa and exponent can be set. The open loop transfer function $G_c(z)G_{vd}(z)$ has 58°and 5dB respectively for phase and gain margin. This approach can be very useful when compensator gains have to set for a new buck converter or to observer system changing when global parameters change as well.

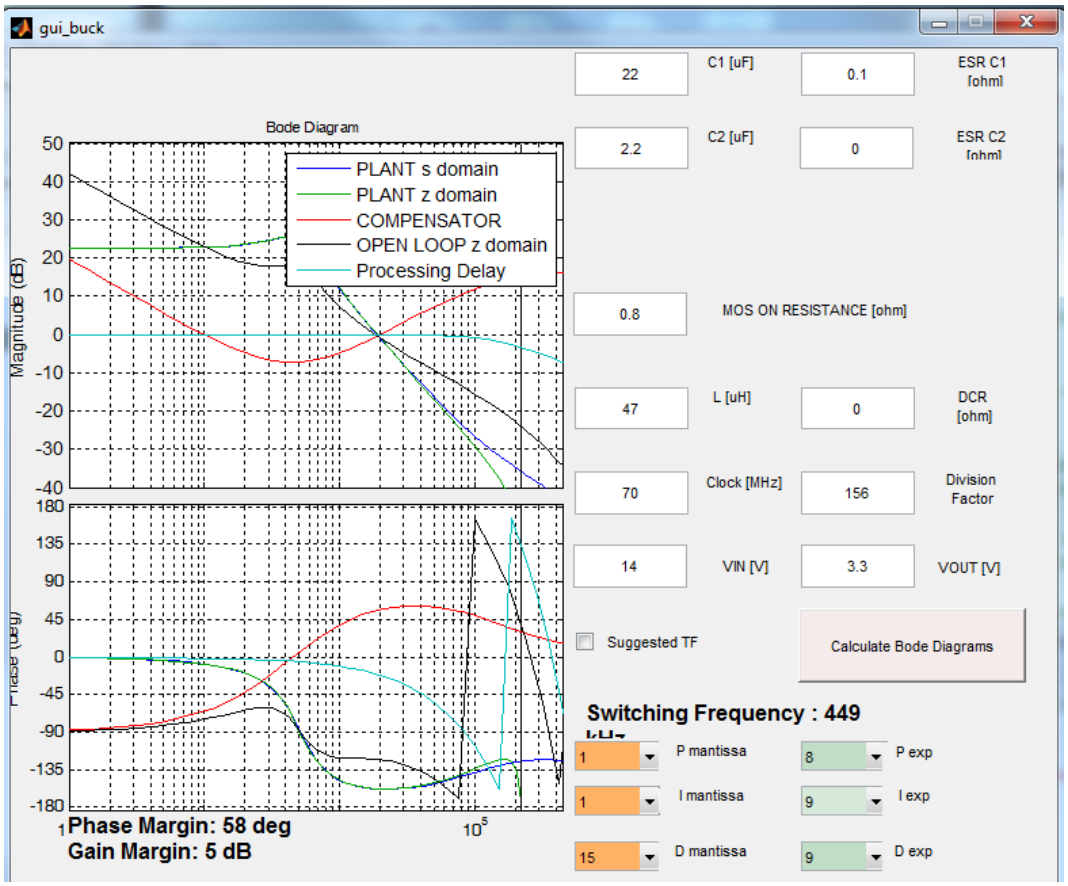

Figure 4.14: PID gains tuning tool.

4.2 VHDL-coded digital control feedback

Once that fixed-point model has been realised and verified, the hardware representation can be coded with same resolution considerations. The Digital compensator, the modulator and the DPWM has been VHDL coded and mapped on Virtex6 FPGA. The hardware description of the digital control loop is shown in Fig.4.15. Because the ADC scale is centred around the zero error bin, the **error_calc** block handles the error from the ADC to generate both two's complement and signed magnitude representation for the PID compensator. This operation does not add nay delay, indeed the **error_calc** is asynchronous. In Tab.4.1 is summarised the signed magnitude representation generated form the ADC output. This encoding technique considers the first bit of the ADC output digital code as the sign (0 is for positive and 1 for negative), while the magnitude is the ADC output or its complement respectively for positive and negative numbers. The sign and magnitude values (respectively *sign_error_i* and *abs_error_s*) are input to the compensator, they respectively represent the first bit and left over four bits of the signed magnitude column in Tab.4.1.

The **start_fsm** is necessary for closed loop configuration. Only when the start up phase ends, this state machine activates the PID and $\Delta\Sigma$ computation in order to close the loop during the steady state phase. The start up is done in an open loop configuration where the low resolution duty cycle (Toff) is generated inside the DPWM.

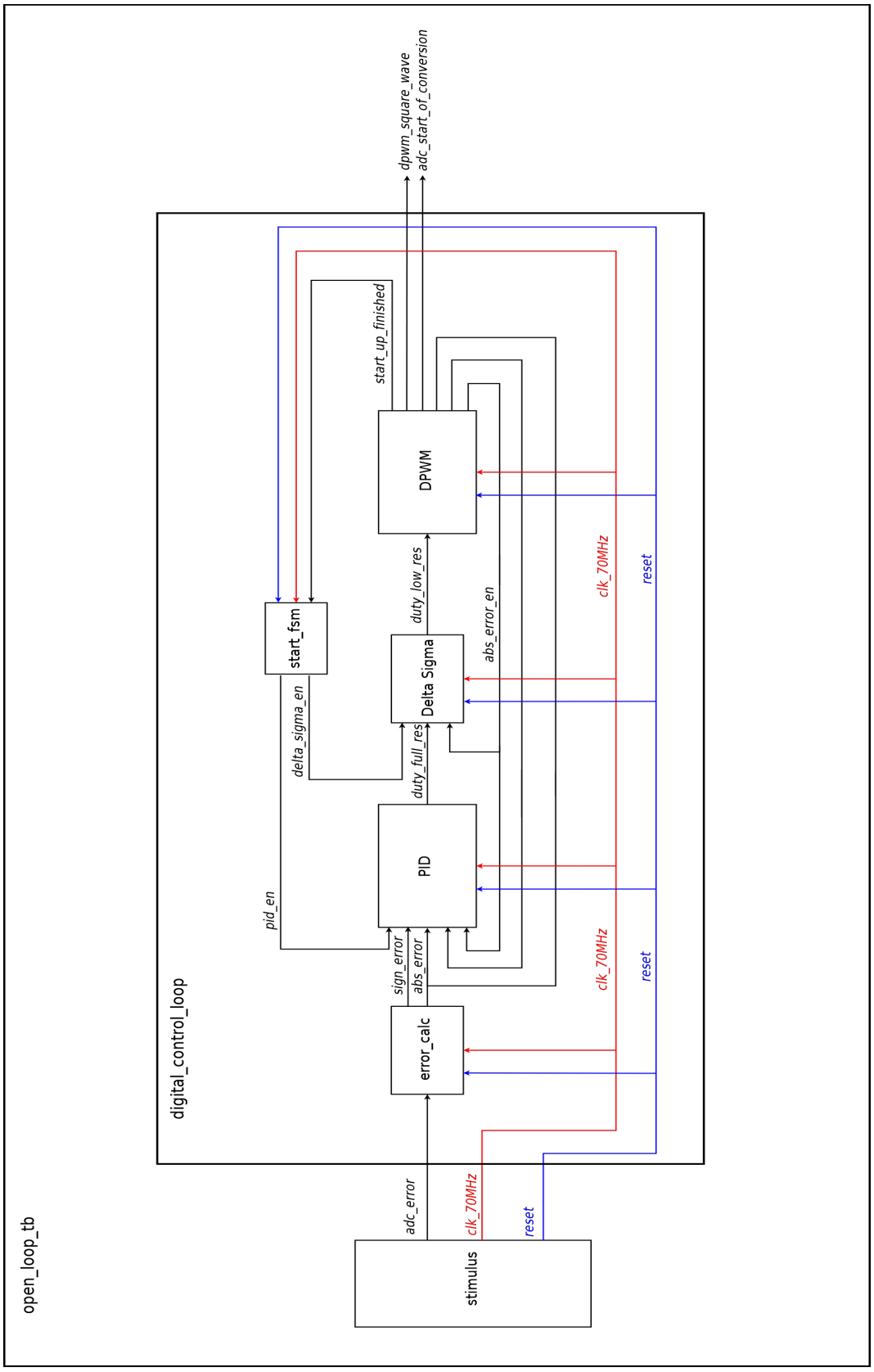

Figure 4.15: Digital control feedback: open loop hardware implementation.

ADC error [LSB_{adc}]	4bits ADC output	Signed Magnitude
8	0111	00111
7	0110	00110
6	0101	00101
5	0100	00100
4	0011	00011
3	0010	00010
2	0001	00001
1	0000	00000
-1	1111	10000
-2	1110	10001
-3	1101	10010
-4	1100	10011
-5	1011	10100
-6	1010	10101
-7	1001	10110
-8	1000	10111

Table 4.1: ADC error encoding.

4.2.1 PID Compensator design

The PID hardware implementation shown in Fig.4.15, receives the error from the asynchronous block **error_calc** that simply translates the ADC output error. The computation of the new control word is done every switching step and is activated with the signal *abs_error_en* generated by the DPWM. The PID computation is activated only during the steady state (through the signal *pwm_en*) generated by the **start_fsm**).
The compensator implementation has the same resolution discussed during the fixed-point modelling. Gains factors are represented with 4 bits respectively for the mantissa and for the exponent, and can be chosen by using the GUI presented in Sec.4.1.5. The computed duty cycle *duty_full_res* is a 28 bits word 12 bits LSB.
The entire PID computation, is done considering a signed magnitude representation as in Tab.4.1 for a window ADC designed to convert a small voltage range around the target output voltage. During the steady state, only the zero error bin is involved (ADC error ± 1), therefore, the sign changes and the magnitude is always equals to zero. As for the fixed-point implementation, errors related to the zero error bin are considered equal to one, while all the others are eight times bigger. The implementation of the proportional part (integral and differential are computed with the same approach) has been shown in Fig.4.16. When the absolute value of the error is null the error sign could be ± 1, the proportional mantissa is equal to the positive or negative proportional gain respectively if the sign is positive or negative. When the absolute value is not null the error magnitude is multiplied by 2^3 (as in the fixed-point implementation) and the computed mantissa is equal to the product between the shifted error and the gain factor. The sign is considered inside the PID gain, for negative input errors the gain *Kp_m_n_i* is used and *Kp_m_n_i* is used for positive signs.
Proportional, Integrative and Derivative calculations are done in parallel, the computations are respectively activated in the same way just described and the same time. This reason, it can be concluded that the designed PID introduces a latency of one clock cycle for the par-

allel computation plus one clock cycle to compute the sum of the three contributes (output has been registered to reduce the critic path between PID and modulator). After two clock cycles the full resolution duty cycle is ready.

```
1  -- Calculate proportional part
2        if pwm_en_i = '1' and abs_error_en_i = '1' then -- calculate when abs_error is ready and after the start up
3              -- Proportional Mantissa
4              if to_integer(abs_error_s) = 0 then      -- abs_error is equal to zero -> zero_error_bin implementation
5                 if sign_error_i = '0' then -- positive
6                     proportional_v := resize( kp_m_i   ,   28);
7                 else                       -- negative
8                     proportional_v := resize( kp_m_n_i , 28);
9                 end if;
10             else
11                if sign_error_i = '0' then -- positive
12                   proportional_v := shift_left(resize( kp_m_i   * abs_error_s, 28), 3);
13                else                       -- negative
14                   proportional_v := shift_left(resize(kp_m_n_i  * abs_error_s, 28), 3);
15                end if;
16             end if;
17
18             -- Proportional Exponent
19             for i in 0 to 15 loop
20                if i = kp_e_i then
21                   proportional_s <= shift_left(proportional_v, (11 - i));
22                end if;
23             end loop;
24       end if;
25 end if;
26 proportional_o     <= std_ulogic_vector(proportional_s); -- output here ==> same as MATLAB
```

Figure 4.16: VHDL implementation: PID proportional part.

4.2.2 Delta Sigma modulator design

The ΔΣ hardware block used in the model shown in Fig.4.15, have to be designed considering that no further delay cannot be added on the signal path. The dithering effect has to be introduced when the high resolution duty cycle is ready. To respect this timing constraint, it is necessary that both the noise shaping and the PID output are ready together. The modulator adds noise shaped contributes to PID computed high resolution word, these contributes are computed considering quantization noise from previous steps (Eq.4.1). The compensator computation latency is of two clock cycles, the noise shaping computation can be enabled with the same signal of the PID start of computation (abs_error_en_i), in this way the the high resolution computation and shaped noise are ready at the same time and can be added together in asynchronously (asynchronous dithering). The modulator works with a full resolution of 28 bits and 12 bits LSB, as presented for the fixed point. Like the compensator and the DPWM, the ΔΣ computation is enabled only when the steady state is reached. Moreover, through the signal *delta_sigma_en_i* generated by the **start_fsm**, is possible to exclude this block form the control loop. When the modulator is not used, the low resolution duty cycle is obtained just discharging 12 bits LSB form the high resolution input duty cycle (no dithering).

```
3   --Dithering
4   duty_full_process_p : process (clk_i, rst_n_i)
5    begin  -- process dpwm_en_p
6      if rst_n_i = '0' then                  -- asynchronous reset (active low)
7        duty_first_full_res_s <= (others =>'0');
8      elsif clk_i'event and clk_i = '1'  then  -- rising clock edge
9        duty_first_full_res_s <= signed(duty_full_res_i) + duty_ntf_error_s;
10     end if;
11   end process duty_full_process_p;
12
13  --Low resolution duty cycle and quntization error computation
14  duty_low_res_p : process (clk_i, rst_n_i)
15   begin  -- process dpwm_en_p
16     if rst_n_i = '0' then                  -- asynchronous reset (active low)
17       duty_low_res_s <= (others =>'0');
18       duty_quant_err_s <= (others =>'0');
19     elsif clk_i'event and clk_i = '1'  then  -- rising clock edge
20       if delta_sigma_en_i = '1' then
21             -- modultator enabled
22             duty_low_res_s <= std_ulogic_vector(duty_first_full_res_s(22 downto 11));
23             duty_quant_err_s <= -("0000000000000000" & duty_first_full_res_s(10 downto 0));
24       else
25             -- modultator disabled
26             duty_low_res_s   <= duty_low_res_i;
27             duty_quant_err_s <= (others =>'0');
28       end if;
29     end if;
```

Figure 4.17: VHDL implementation: ΔΣ dithering function.

Consideration addressed up to now, are shown in Fig.4.17 where both dithering and low resolution VHDL implementation are described. The full resolution duty cycle (*duty_first_full_res_s*)is calculated at every clock cycle and is the sum of the input full resolution duty cycle (*duty_full_res_i*) with the noise shaped error (*duty_ntf_error_s*). Low resolution duty cycle computation generates both signals *duty_low_res_s* and *duty_quant_err_s* respectively considering the MSB and the LSB of *duty_first_full_res_s*. The signal quantization error *duty_quant_err_s* is the actual error and is used as z^{-1} contribute during the next computation.

The noise shping function design is shown in Fig.4.18. Every clock cycle the computation on previous quantization errors is enabled through the signal *delta_sigma_en_calc_i*. This signal is simply equal to the PID start of computation emphabs_error_en_i, indeed the noise shaped error *duty_ntf_error_s* computation is activated in the same instant of the compensator so that these two computations can be ready together. The $1-NTF_3(z)$ (rows 57, 58 and 59 of Fig.(4.18) is coded by shifting previous quantization errors and its output is stored in the register *duty_ntf_error_s*. The signal *ntf_mod_en_s* is used to distinguish between normal noise shaper and modified noise shaper. The normal third noise shaper ($NTF_3(z)$) refers to Eq.4.1, while the modified noise shaper refers to Eq.3.26. When a zero insertion is considered in the noise shaping function, the signal *duty_ntf_error_s* is computed considering the digital implementation of:

$$1-NTF(z) = (k+1)z^{-1} - (k+1)z^{-2} + z^{-3}, \tag{4.2}$$

where $K = 2cos(2\pi\frac{f_n}{f_{sw}})$ permits to insert the notch at f_n. From the digital point of view, a solution have to be used for the cosine computation. When none zero is inserted $f_n = 0$ and $k = 2$, the classical third order NTF is considered. Let's introduce $H = k+1$ and consider a

```
52      elsif clk_i'event and clk_i = '1' then
53        if delta_sigma_en_i = '1' then
54          if delta_sigma_en_calc_i = '1' and pwm_en_i = '1' then
55            if ntf_mod_en_i = '0' then
56            -- NTF -- third order
57               noise_shaped_v   := shift_left(q_error_Z1_s,1)+ q_error_Z1_s; -- 3*e[n-1]
58               noise_shaped_1_v := -noise_shaped_v + (shift_left(q_error_Z2_s,1)+ q_error_Z2_s); -- -3*e[n-1]+ 3*e[n-2]
59               noise_shaped_2_v := noise_shaped_1_v - q_error_Z3_s; -- 3*e[n-1]+ 3*e[n-2] - e[n-3]
60            else
61            -- NTF-zero in 8.95KHz fsw=449KHz --
62            noise_shaped_v := shift_left(q_error_Z1_s,11)+ shift_left(q_error_Z1_s,10) - shift_left(q_error_Z1_s,4) ;
63            noise_shaped_1_v := noise_shaped_v - (shift_left(q_error_Z2_s,11)+ shift_left(q_error_Z2_s,10) - shift_left(q_error_Z2_s,4));
64            noise_shaped_2_v :=  shift_right(noise_shaped_1_v,10) +  q_error_Z3_s;
65            end if; --NTF MOD
66            -- NTF output
67            duty_ntf_error_s  <= noise_shaped_2_v;
68            -- updating error
69            q_error_z1_temp_v := q_error_z1_s;      -- saving old e[n-1] which will be e[n-2]
70            q_error_z2_temp_v := q_error_z2_s;      -- saving old e[n-2] which will be e[n-3]
71            q_error_z1_s      <= duty_quant_err_s;  -- saving actual error which for the next NTF-calculation will be e[n-1]
72            q_error_z2_s      <= q_error_z1_temp_v;
73            q_error_z3_s      <= q_error_z2_temp_v;
74          end if;
75        else
76            duty_ntf_error_s <= (others => '0');
77            q_error_z1_s     <= (others => '0');
78            q_error_z2_s     <= (others => '0');
79            q_error_z3_s     <= (others => '0');
80        end if;
81       --end if; -- added for cosimulation
82      end if;
83    end process noise_shaping_p;
84
85  duty_cycle_o          <= duty_low_res_s;
86  duty_ntf_error_o      <= std_ulogic_vector(duty_ntf_error_s);
87  duty_first_full_res_o  <= std_ulogic_vector(duty_first_full_res_s);
88  end rtl;
```

Figure 4.18: VHDL implementation: ΔΣ noise shaper.

desired zero introduced in $f_n = 2kHz$, then:

$$H = 1 + 2cos\left(2\pi \frac{f_n}{f_{sw}}\right) = 2.99922. \tag{4.3}$$

The noise shaper expression which has to be designed becomes:

$$1 - NTF(z) = (H)z^{-1} - (H)z^{-2} + z^{-3}. \tag{4.4}$$

A way to obtain a digital representation of H, is based on adding resolution and rounding to obtain an integer number. Let,s considering 10 bits shift left on the H value:

$$H * 2^{10} = 3071.2. \tag{4.5}$$

A little amount of error is introduced in the zero representation if the value of H is rounded to 3071, indeed considering $3071/2^{10}$ the zero position results in $f_n = 2.24kHz$. Assuming to accept this error, the expression of the noise shaper becomes:

$$1 - NTF(z) = (\frac{3071}{2^10})z^{-1} - (\frac{3071}{2^{10}})z^{-2} + z^{-3}. \tag{4.6}$$

From the digital encoding point of view, is easy to represent 3071 as a power of two ($2^{11} + 2^{10} - 1$), consequently the polynomial expression in Eq.4.6 can be easily coded:

- $noise_shaped_v = (2^{11} + 2^{10} - 1) * quantization_error[n-1]$.
- $noise_shaped_1_v = noise_shaped_v + (2^{11} + 2^{10} - 1) * quantization_error[n-2]$.
- $noise_shaped_2_v = noise_shaped_1_v/2^{10} + quantization_error[n-3]$.

The just mentioned approach is summarised in Fig.4.19 for $f_n = 2kHz$ and can be observed in rows 62, 63 and 64 of Fig.4.18, where the case $f_n = 8kHz$ has been integrated in the noise shaper design to generate the signal *duty_ntf_error_s*.

```
-- NTF-zero in 2.2382KHz fsw=450KHz -- 3071/1024    MOD
noise_shaped_v := shift_left(q_error_Z1_s,11)+ shift_left(q_error_Z1_s,10) - q_error_Z1_s; -- 3071/1024*e[n-1]
noise_shaped_1_v := noise_shaped_v - (shift_left(q_error_Z2_s,11)+ shift_left(q_error_Z2_s,10) - q_error_Z2_s);
noise_shaped_2_v :=  shift_right(noise_shaped_1_v,10) +  q_error_Z3_s;
```

Figure 4.19: VHDL implementation: ΔΣ noise shaper with $2kHz$ notch.

4.2.3 DPWM design

The DPWM module shown in Fig.4.15, receives the MSB of the computed high resolution duty cycle (with or without dithering). In a down-counter configuration the low resolution duty cycle is the off time of the generated square wave. From the VHDL encoding point of view, this block is mainly a counter that generates a square having an off time equal to the low resolution duty cycle 4.11. Furthermore, the DPWM works as control for the system, handling the start up phase and to generate enabling signals for each block.
During the start up both PID and ΔΣ modulator are disabled, the system is in an open loop configuration where only the ADC and the DPWM are used. In this phase the $Toff$ is not an input, its initial value is set close to the maximum (for instance $OS_factor - 4$) when the DPWM is initialised, hence, a minimum duty cycle square wave is generated. During the start up phase the DPWM generates also the start of conversion signal for the ADC and reads the error magnitude from the **error_calc** (Fig.4.15). Step by step (in order to have a soft start up, every step could be for instance $3T_s$) the DPWM decrements the off time and reads the magnitude of the error *abs_error_s*. Consequently, the duty cycle increases until when a magnitude error equal to zero is reached and the desired dc output voltage value is obtained. The steady state condition is triggered, and the DPWM output *start_up_finished* is high. This signal is read from the **start_fsm** which enables both PID and ΔΣ modulator, and only in this moment the DPWM outputs the obtained off time to initialize the compensator which will compute next high resolution $Toff$ ($d_{hr}[n]$) considering the sampled ADC output error.
In order to update the duty cycle every T_s, the closed loop computation have to compute the new $d_{lr}[n]$, needed from the DPWM to generate the square to drive the power-MOS (Fig.4.11). During steady state computations, the timing of the system is managed by the DPWM. Every switching cycle, it activates the ADC, the PID and the noise shaper computation respectively generating the signals *adc_start_of_conversion* and *abs_error_en* (Fig.4.15). To manage the timing, the DPWM block analyses firstly the low resolution $Toff$ ($d_{lr}[n]$) and then generates the respective start signals preparing the system for the next computation, with this scheme:

- the *adc_start_of_conversion* is generate at $Toff/2$.
- One clock cycle later the ADC start of conversion, the signed magnitude error is ready (*error_calc* is an asynchronous block). The PID computation can be enabled and the signal *abs_error_en* is high.
- Because dithering effects on high resolution $Toff$, the noise shaping computation is activated together with the PID.

Considering an ASIC implementation, PID output and dithering effects could be computed with asynchronous mode. The global latency needed to compute the new $Toff$ is basically of only two clock cycles (one for the ADC conversion e one for the PID computation). Just presented block designs are though for a FPGA prototyping. In order to increase the maximum frequency of the digital control (system speed-up), both PID and $\Delta\Sigma$ outputs are registered signals. This way, the critical path between the compensator (which uses multipliers) and the modulator is avoided, therefore, two more clock cycles have to be added. the global latency for the computation is of four clock cycles. This latency is automatically a constraint on the minimum $Toff$, the computed low resolution duty cycle cannot be less than four. Consideration about the max frequency of the regulation loop has been addressed in the next section. The hardware interfacing in an open loop configuration of modules just described has been shown in Fig.4.15, where also the testing environment has been integrated.

4.3 FPGA-based co-simulation closed loop system

First verifications at hardware level have been addressed in an open loop configuration presented in Fig.4.15. The VHDL-coded modules (**error_calc**, **start_fsm**, PID, $\Delta\Sigma$ and DPWM) have been integrated into a VHDL top module called **digital_control_loop**. This module has been used as DUT in a VHDL testbench (**open_loop_tb**), the stimuli to generate for the open loop configuration are the ADC error, the clock system and the reset signal. In order to emulate both start up and steady state conditions, the error sequence has been recorded from Matlab fixed-point model simulations (using the model 7.15) and used as input for the DUT. The **digital_control_loop** top module has been synthesized in Virtex6 FPGA and a maximum frequency of $124.748MHz$ can be reached, the open loop simulation can be clocked at $70MHz$.

Once that the debug of the open loop system has been done, the aim is to obtain a close loop system integrating the ADC and the buck converter in the VHDL open loop configuration just described. Both ADC and buck converter are analog blocks, hence a mixed-signal approach is needed to realize the complete system and verify the VHDL implementation without generating any stimulus.

Xilinx System Generator permits to integrate the VHDL-coded structures in the Matlab/Simulink environment and the mixed-signal co-simulation can be directly managed with Simulink. VHDL-Matlab co-simulations integrates both the ADC fixed-point point model and the Matlab/Simulink modelled buck converter within the **digital_control_loop**. This approach is very useful from the debug point of view especially in mixed signal designs, all signals can be monitored with Simulink scopes and VHDL outputs can be debugged directly during closed loop simulations. The closed loop system during co-simulations has

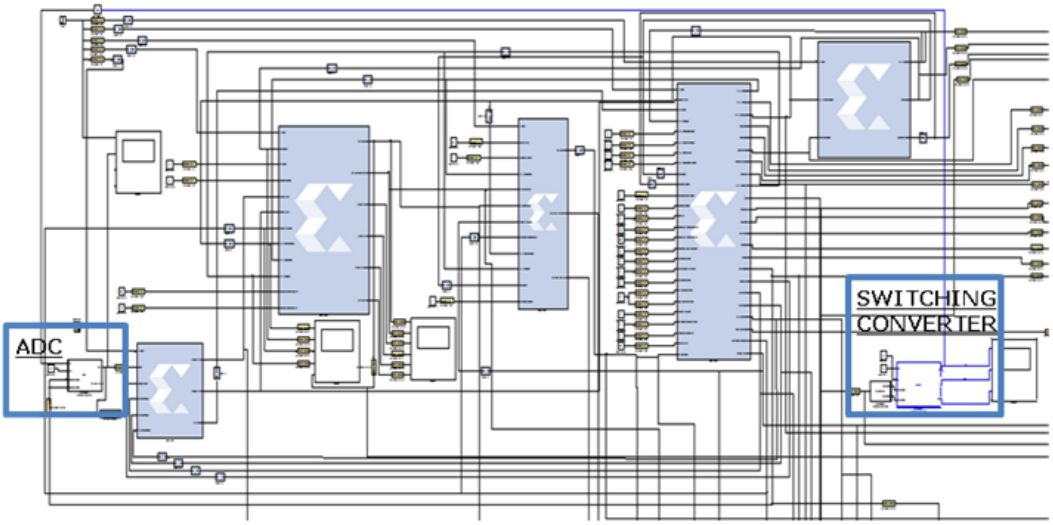

Figure 4.20: VHDL/Matlab co-simulation closed loop: model.

been shown in Fig.4.20. Grey coloured blocks with the Xilinx symbol (X) represent the **digital_control_loop** (Fig.4.15), the ADC and the Power plant are described as the Matlab/Simulink functions shown in Fig.7.15. The interface between VHDL blocks and Simulink blocks has done with three main blocks:

- **Assert** blocks are needed to the software simulator to understand the starting point of the computation. They work for hardware loops as like as z^{-1} works for Simulink loops between blocks. Among the options for this block the sample rate have to be set to define clock frequency or the computation step ($T_{clk} = 1/70MHz$).
- **Gateway-In** block are used as inputs to a Xilinxs blocks.
- **Gateway-Out** are used as outputs from a Xilinx blocks.

Hardware/Software co-simulations needs clock gating considerations to synchronize both types of computations. In every VHDL module a clock gating logic has to be inserted in order to synchronize the hardware simulator (Xilix iSim) with the software one (Simulink for ADC and buck converter). The VHDL-coded blocks cannot be simulated as Simulink blocks and with a double click on Xilinx blocks is necessary to set the hardware simulator. A default choice is the Xilinx's built-in simulator. Communications between these simulators are possible thanks both the clock gating logic in the VHDL and gateway blocks in the Simulink model.

The buck converter output voltage Vr and the inductor current during the co-simulation are shown in Fig.4.21, a reference voltage of $3.3V$ has been considered. The result is obtained with same settings of previous simulations $f_{clk} = 70MHZ$ and $f_{sw} = 449kHz(os_factor = 156)$. Compensator, modulator, DPWM, **error_calc** and **start_fsm** do not defer form the hardware description in the previous section, as well as both the fixed point ADC and the buck converter from the previously defined models. The considered buck converter presents a resonant frequency $f_0 = 5kHz$ ($C = 22\mu F, L = 47\mu H, ESR = 0\Omega$), an output resistance $R =$

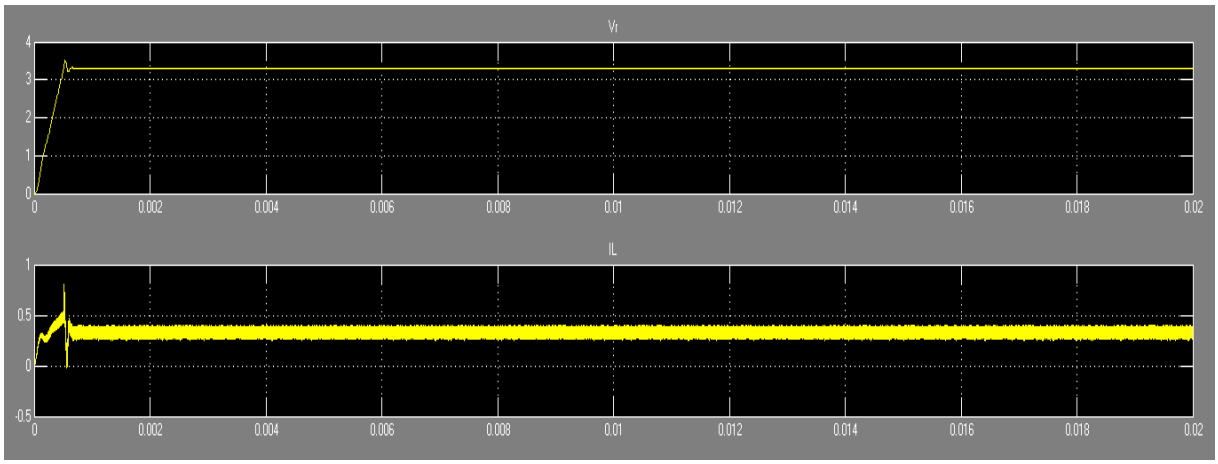

Figure 4.21: VHDL/Matlab co-simulation closed loop: output.

10Ω and input voltage $V_{in} = 10V$. In Fig.4.22 a zoom on the output voltage is shown, the switching voltage and the inductor current shows some ripple. The zoom confirms that the loop resolution set with fixed/floating point comparison (Fig.4.12), is sufficient to obtain a regulated output voltage very close to the reference value.

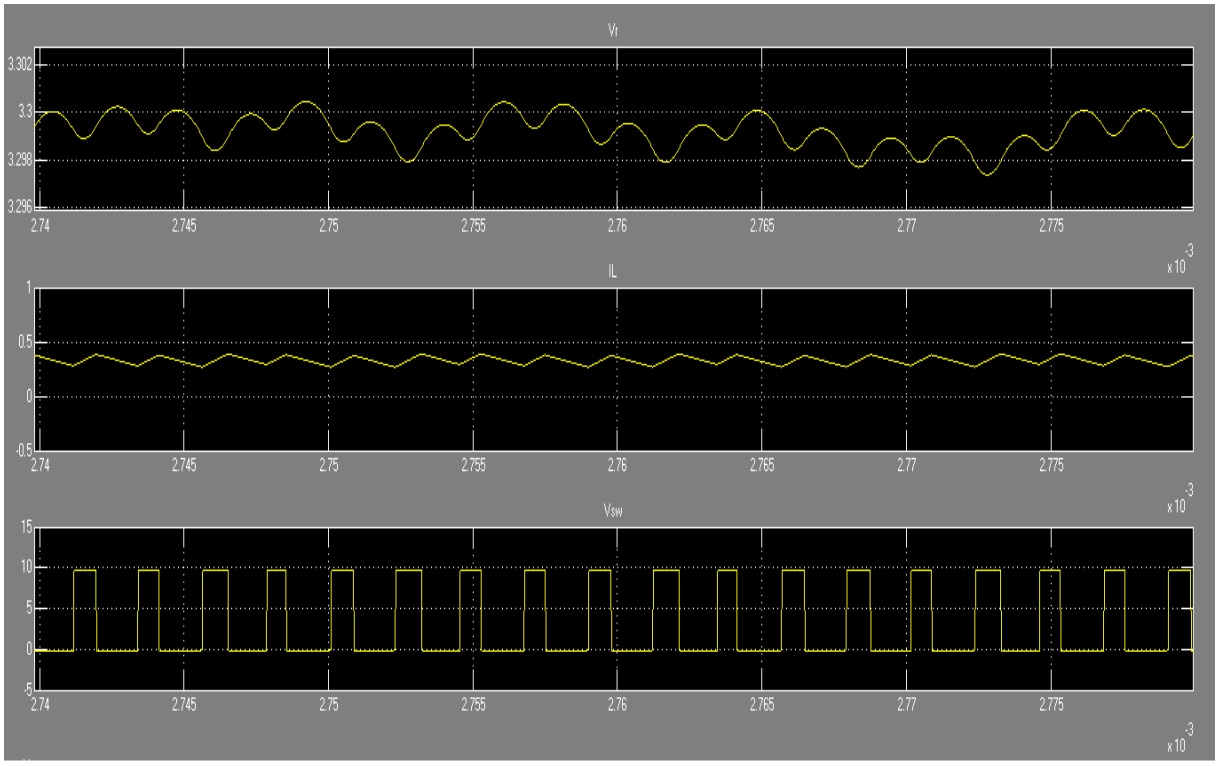

Figure 4.22: VHDL/Matlab co-simulation closed loop: output (zoomed).

To validate the designed regulation loop, the closed loop system can be tested during a load jump of the output resistance. The system is considered robust if is able to react and

compensate sharp output voltage variations, either during a positive or negative load jump. In Fig.4.23 can be observed that, even if a sharp load jump occurs, the system soon reacts driving the output voltage close to the desired dc level. Moreover, in Fig.4.24 the system response has been zoomed. The regulation loop is able to react to the load variations in both senses.

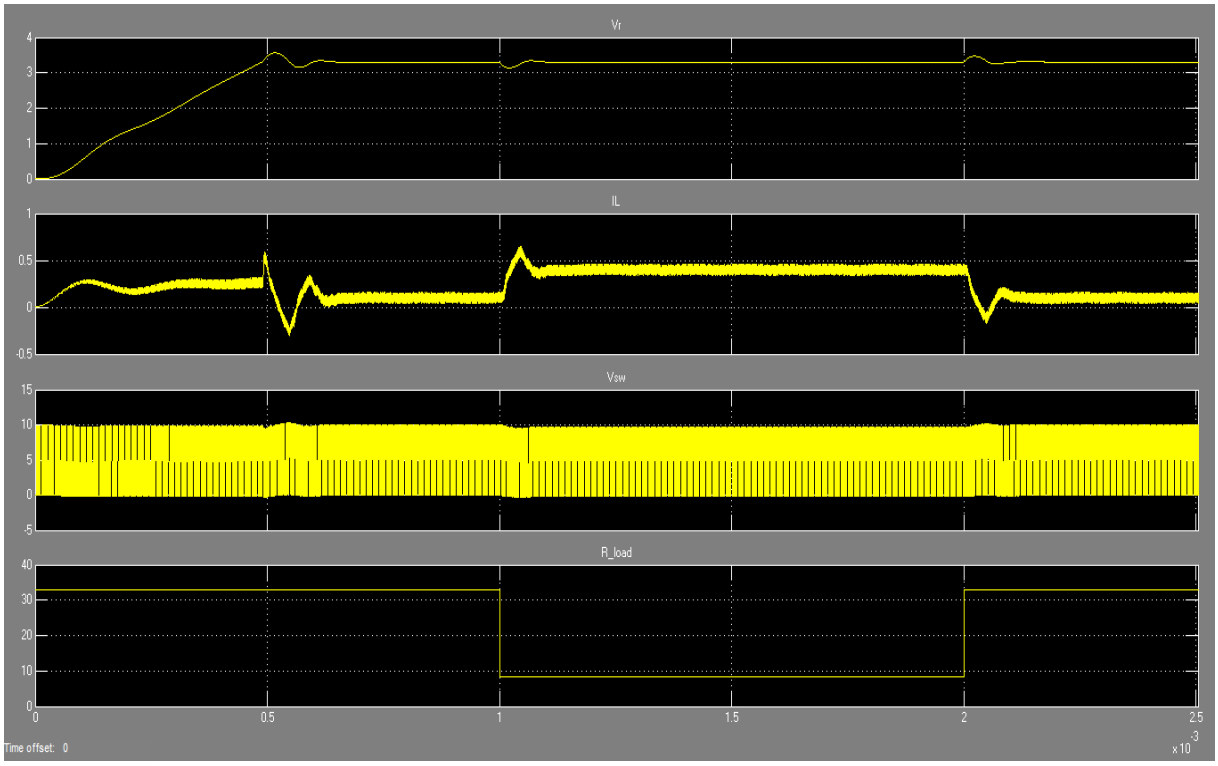

Figure 4.23: VHDL/Matlab co-simulation closed loop: load step output reaction.

Another approach to consider hardware/software mixed-signal co-simulations, is based on post-synthesis simulations. In the model considered in Fig.4.20 the analog blocks are still simulated in the same way (Simulink), while VHDL-coded blocks can be mapped on the FPGA to avoid any hardware simulator use. This way, the post-synthesis simulation output of digital blocks can be observed directly on Simulink Scopes. When this approach is used, the System Generator runs the synthesis of the top module **digital_control_loop** (Fig.4.15). The post-synthesis closed loop system has been shown in Fig.4.25. As in the open loop configuration the **digital_control_loop** reads the ADC output error and outputs both the ADC start of conversion (*adc_sample_s*) and the square wave (*pwm_o*) to drive the buck converter. The digital control design has been synthesize considering a Virtex6 FPGA, and the clock system is now provided by the FPGA built-in ring oscillator. Post-synthesis simulation results are equal to previously obtained, the regulation loop is confirmed to be robust and well designed.

In the model shown in Fig.4.25, the **digital_control_loop** presents some additional signals if compared with the same top module introduced in Fig.4.15 for the open loop simulations. The signal *pwm_num_ext_dc_i* is a test input used when the digital control and the external duty cycle value are given directly as input to the DPWM (as modelled in Fig.1.4).

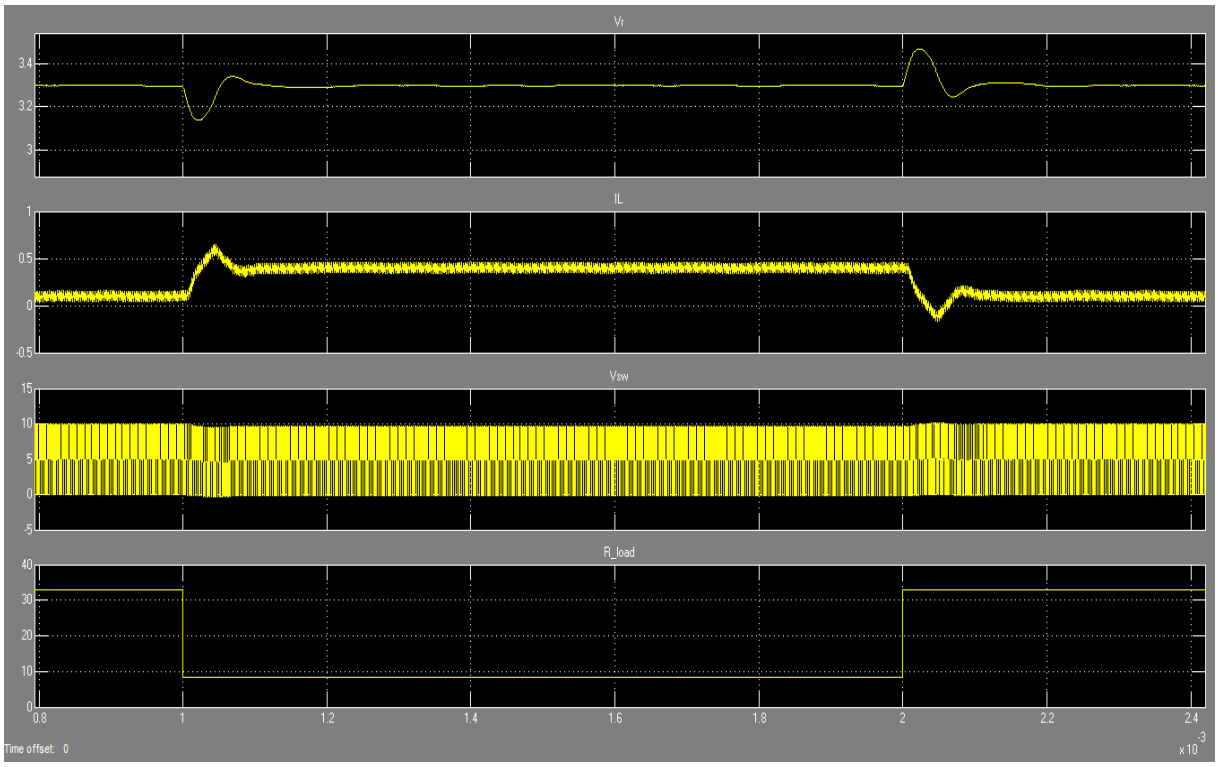

Figure 4.24: VHDL/Matlab co-simulation closed loop: load step reaction (zoomed).

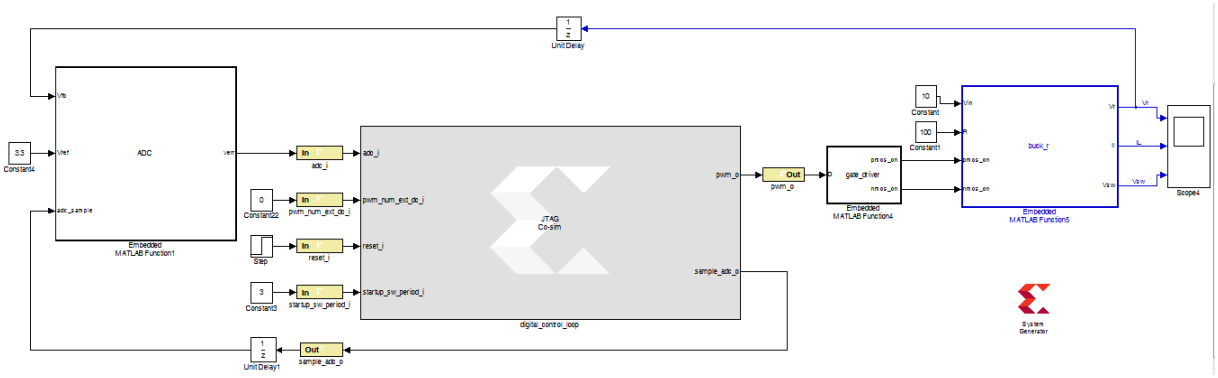

Figure 4.25: VHDL/Matlab FPGA-based closed loop buck converter.

The input *startup_sw_period_i* represents the number of switching cycles that, during the start up, the DPWM waits before reducing the $Toff$. In the closed loop system in Fig.4.25, the DPWM waits three switching cycles before changing the off time of the output square wave (soft start up).
The **digital_control_loop** design has been synthesized on a Virtex6 FPGA, a maximum frequency of 124.748MHz can be reached. The Virtex6 resources usage is presented in Tab.4.2, while, the involved hardware resources summarised in the HDL synthesis report has been shown in Tab.4.3), while

Virtex6 FPGA resources	
Slice Logic Utilization	
Number of Slice Registers	402 out of 301440 0%
Number of Slice LUTs	1123 out of 150720 0%
Number used as Logic	1123 out of 150720 0%
Slice Logic Distribution:	
Number of LUT Flip Flop pairs used	1203
Number with an unused Flip Flop	801 out of 1203 66%
Number with an unused LUT	80 out of 1203 6%
Number of fully used LUT-FF pairs	322 out of 1203 26%
Number of unique control sets	38
IO Utilization:	
Number of IOs	81
Number of bonded IOBs	77 out of 600 12%
Specific Feature Utilization:	
Number of BUFG/BUFGCTRLs	1 out of 32 3%

Table 4.2: Digital control prototype: Virtex6 resource usage.

Prototype resources usage	
# Multipliers	4
7x5-bit multiplier	4
# Adders/Subtractors	31
12-bit adder	3
12-bit subtractor	3
13-bit adder	3
14-bit subtractor	1
27-bit adder	2
28-bit adder	4
28-bit addsub	1
28-bit subtractor	8
29-bit adder	1
5-bit adder	2
6-bit subtractor	1
7-bit subtractor	1
8-bit subtractor	1
# Registers	57
1-bit register	28
12-bit register	9
28-bit register	13
5-bit register	3
6-bit register	2
7-bit register	2
# Comparators	15
11-bit comparator lessequal	1
12-bit comparator equal	2
12-bit comparator greater	4
13-bit comparator greater	2
28-bit comparator greater	1
29-bit comparator greater	2
5-bit comparator lessequal	1
6-bit comparator lessequal	1
7-bit comparator lessequal	1
# Multiplexers	96
1-bit 2-to-1 multiplexer	15
12-bit 2-to-1 multiplexer	20
28-bit 16-to-1 multiplexer	2
28-bit 2-to-1 multiplexer	32
29-bit 2-to-1 multiplexer	17
5-bit 2-to-1 multiplexer	4
5-bit 4-to-1 multiplexer	1
6-bit 2-to-1 multiplexer	3
7-bit 2-to-1 multiplexer	2
# FSMs	1

Table 4.3: Digital control prototype: resource usage.

Chapter 5

Digitally controlled buck converter prototype: Off-line controller

The hardware realisation and verification of the digital control loop has been presented in the previous chapter. Starting from floating point point realisation and moving to a fixed-point Matlab/Simulink modelling, a robust VHDL-coded control loop has been designed and tested through hardware/software closed loop co-simulations. The system has been synthesized on a Virtex6 FPGA, presenting a maximum frequency beyond $120MHz$. Fixed-point models has been used as reference during the hardware design to fix the resolution.
Prototyping of the digital control law has been done through post-synthesis co-simulations, the synthesised **digital_control_loop** (Fig.4.15) has been interfaced with Matlab/Simulink models of the ADC and buck converter, avoiding in this way any testbench for stimuli generation (Fig.4.20).
Aim of this chapter, is to introduce a tool-independent FPGA-based closed loop prototype of digitally controlled buck converters. In order to have a full-custom off-line controller, it is necessary to design the analog blocks that have been modelled up to now and, then, study the interface between these blocks and the digital control loop mapped on the FPGA.
The analog blocks have been designed by the company Infineon Technologies AG as part of a custom Test Chip (TC). This TC can be configured in different operative modes, the *external loop* configuration mode is one among those. With this operative mode, the ADC start of conversion or ADC clock sampling (***adc_sample***) and the DPWM square wave ($d(t)$), are both inputs to the TC and can be output from **digital_control_loop** mapped on the FPGA. The prototyping idea has been shown in Fig.5.1 where, the TC is in its external loop configuration and the FPGA contains the **digital_control_loop** configured to be interfaced with the TC. The ADC realised in the TC is totally equivalent to the one modelled up to now. It is designed to convert a small voltage range (window) around the target output voltage (window ADC). The resolution of the ADC is 4 bits and the nominal quantization step in the case of a Buck converter is around $15mV$. In this model the LSB size can be programmed (Sec.5.2) in 5 steps according to Tab.5.1.

The closed loop prototype realisation together with the measurement systems, have been shown in Fig.5.2. the TC and FPGA settings needed to configure the closed loop system are presented in the next sections. All the analog signal in the TC can be monitored directly using an oscilloscope, while, signals of the digital control can be monitored on the FPGA.

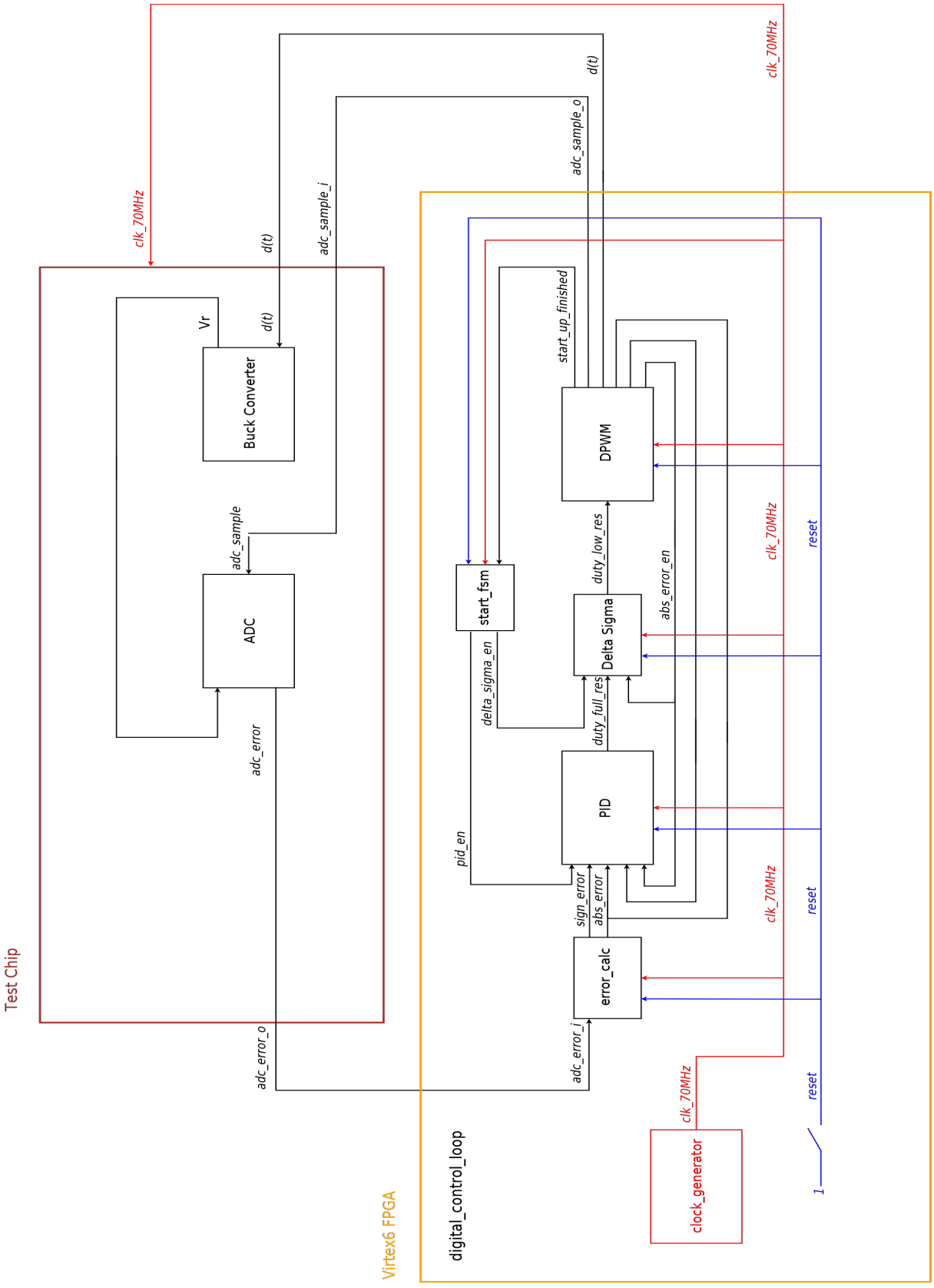

Figure 5.1: FPGA-TestChip: prototype model.

Configuration	LSB size [mV]
00000	10.00
00001	15.75
00010	20.25
00011	24.75
00100	30.00
Other	Not defined

Table 5.1: Test Chip: LSB_{ADC} possible settings

ChipScope Pro is a Xilinx tool for real time monitoring of designs mapped on the FPGA.

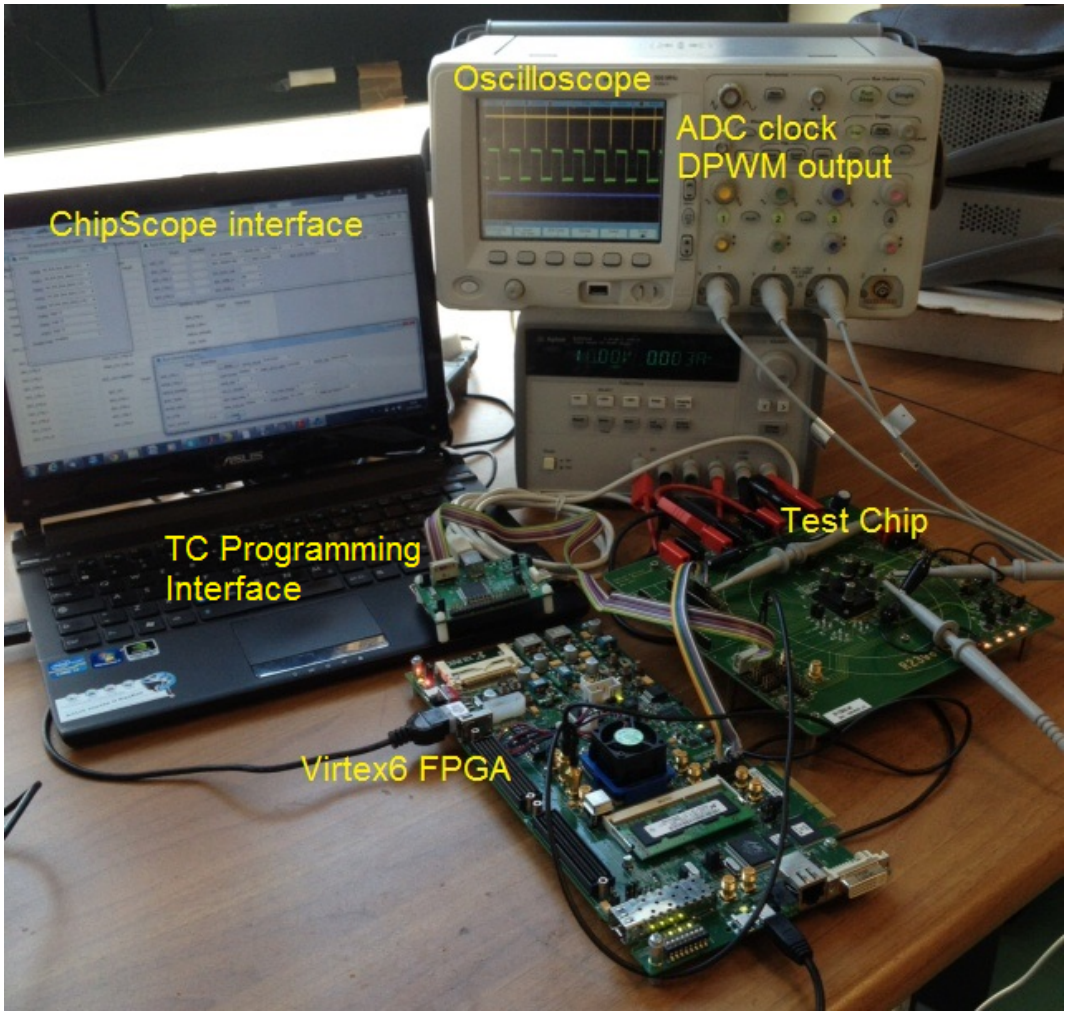

Figure 5.2: FPGA-Test Chip: prototype

In Fig.5.3 is shown the oscilloscope imagine of the buck converter output voltage (yellow curve), inductor current (violet curve) and the DPWM output (blue curve). It is visible how the prototype works, the closed loop prototype evolves from start up to steady state condition. In this example, the output voltage is regulated considering a $5V$ reference voltage and $10V$ input voltage. The clock frequency is of $70MHz$ and the switching frequency

is of $449kHz$. As for previous results, the buck converter is configured to have a resonant frequency $f_0 = 5kHz$. In Fig.5.4 and Fig5.5, the ADC clock (*adc_sample*) and the DPWM output ($d(t)$) has been shown. The ADC start of conversion is generated one time per switching cycle, indeed the frequency measured from the oscilloscope is closed to the switching frequency. As expected, the DPWM output during steady state is a 50% duty cycle square wave ($D = V_{ref}/V_{in} = 0.5$).

Figure 5.3: FPGA-TC: prototype output signals.

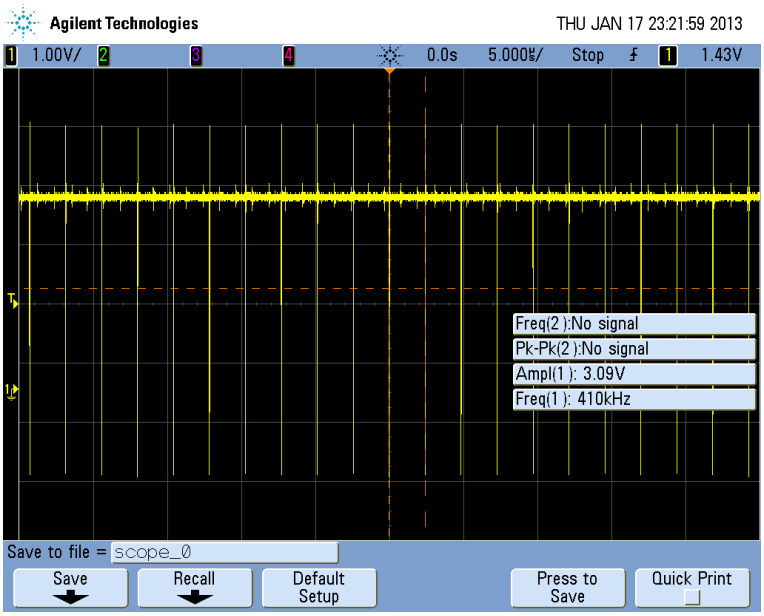

Figure 5.4: FPGA output: ADC clock.

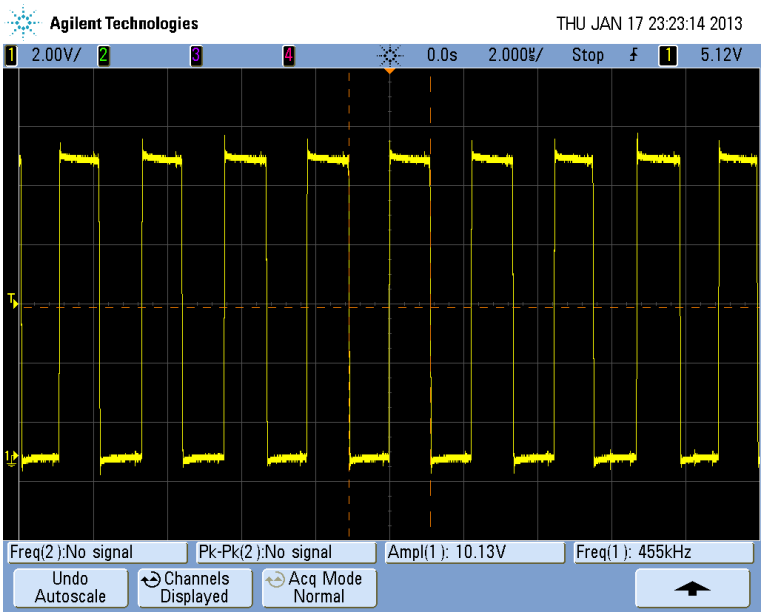

Figure 5.5: FPGA output: DPWM square wave output.

In order to validate the prototype structure, the comparison between the co-simulation and the prototype TC has been shown in Fig.5.6. The output voltage evolution of these models is the same and the same start up latency is obtained. The load step test on the prototype model has been shown in Fig.5.7, a robust digital control system has been then realised and verified.

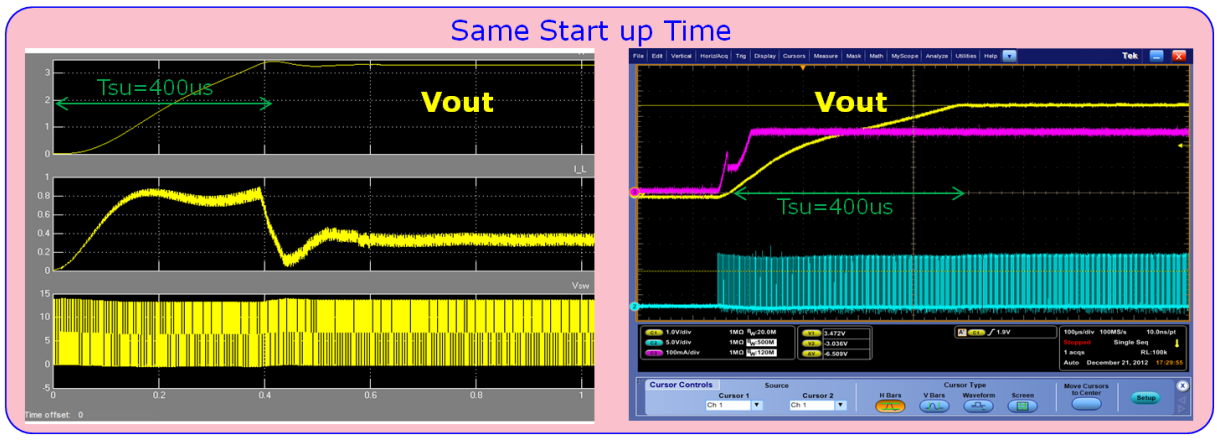

Figure 5.6: Co-simulation and FPGA-TC prototype: outputs comparison.

5.1 FPGA configuration

To configure the FPGA for the closed loop system in Fig.5.2, some features have to be added to the **digital_control_loop** used in the previous chapter for the co-simulation. Three main

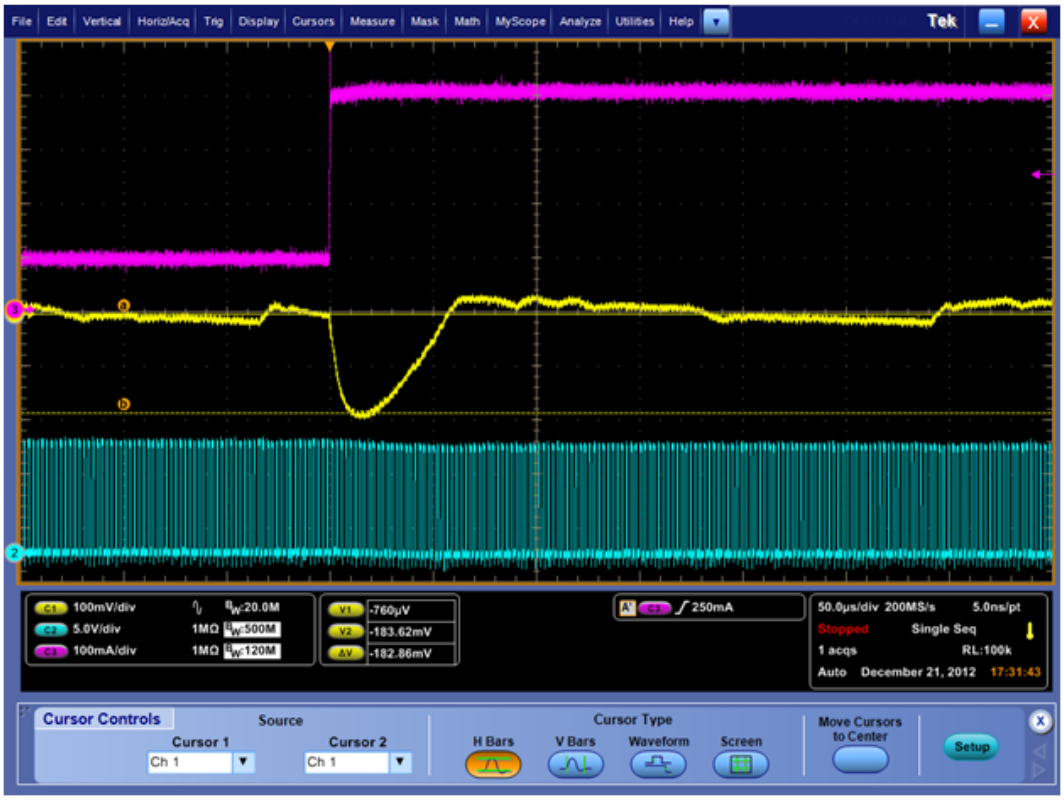

Figure 5.7: FPGA-TC prototype: load step reaction.

aspects have to be considered:

- VHDL design of *clock_generator* block (Fig.5.1).
- Observing post-synthesis internal signals.
- I/O mapping to manage the communication between FPGA and TC.

When these aspects have been addressed the final system is ready for the digital design flow. After the classical approach (Synthesys, Translate, Map and Place& Route) the bitstream file is generated, the Xilinx tool **iMPACT** permits to map the entire design (**digital_control_loop** in Fig.5.1) on the FPGA.

5.1.1 Clock generator

During hardware/software co-simulation the clock of the system was managed through clock gating logic inserted in the VHDL. To have a tool independent system, it is necessary to insert in the system a block to generate the desired clock system.
Let's refer to Fig.5.1, the **digital_control_loop** used during co-simulations (Fig.4.20)has been integrated with the **clock_generator** block. This further block generates the desired clock system (for instance $f_{clk} = 70MHz$) starting from the reference one in the FPGA.
With the Xilinx ISE Design Suite is possible to generate automatically the VHDL code related to the *clock_generator* block, in order to insert it inside the top level module to feed the

computational blocks (Fig.5.1). For this purpose, in Fig.5.8 is the clock wizard that permits to set the **clock_generator** input and output clocks. In the example, a differential input clock of $200MHz$ and a single ended output clock of $70MHz$ has been chosen. The ml605 evaluation board (Virtex6 FPGA) has an internal reference clock of $200MHz$, the output of the wizard is the VHDL implementation of a **clock_generator** able to generate $70MHz$ from $200MHz$ input differential clock.

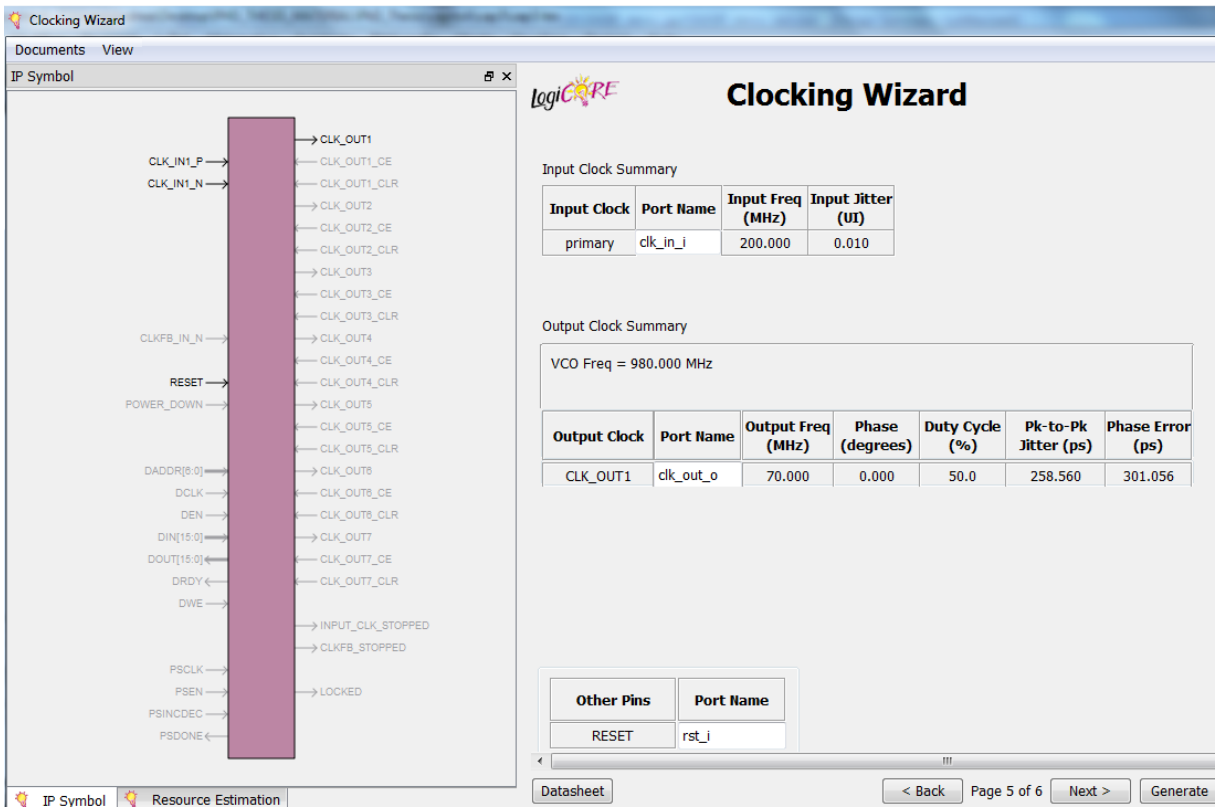

Figure 5.8: FPGA clocking wizard: $70MHz$ clock generator.

5.1.2 Observing FPGA internal signals

One of the advantages of the co-simulation is the easy debugging feature, digital signals (from simulation or from the FPGA) can be observed directly with Simulink Scopes. Once the design moves to a full-custom prototyping, ChipScope Pro tool allows to view any internal signal on the FPGA. Signals are captured in the system at the clock frequency and brought out through a programming interface.
ChipScope Pro tool is mainly composed of two tools, Chipscope Pro Core Inserter and ChipScope Pro Analyzer. The first permits to generate a hardware module, that contains signals to capture inside the FPGA. This module is inserted in the hierarchy inside the top level ((**digital_control_loop**)). Captured signals are displayed and analysed using the GUI of ChipScope Pro Analyzer tool, this software works as a real oscilloscope and permits to observe signal and buses without using any FPGA I/O resource. The interface of the ChipScope

Pro Analyzer tool, when Chipscope Pro Core Inserter have been set to monitor the ADC error (4 bits) output from the TC, have been shown in Fig.5.9 there is.

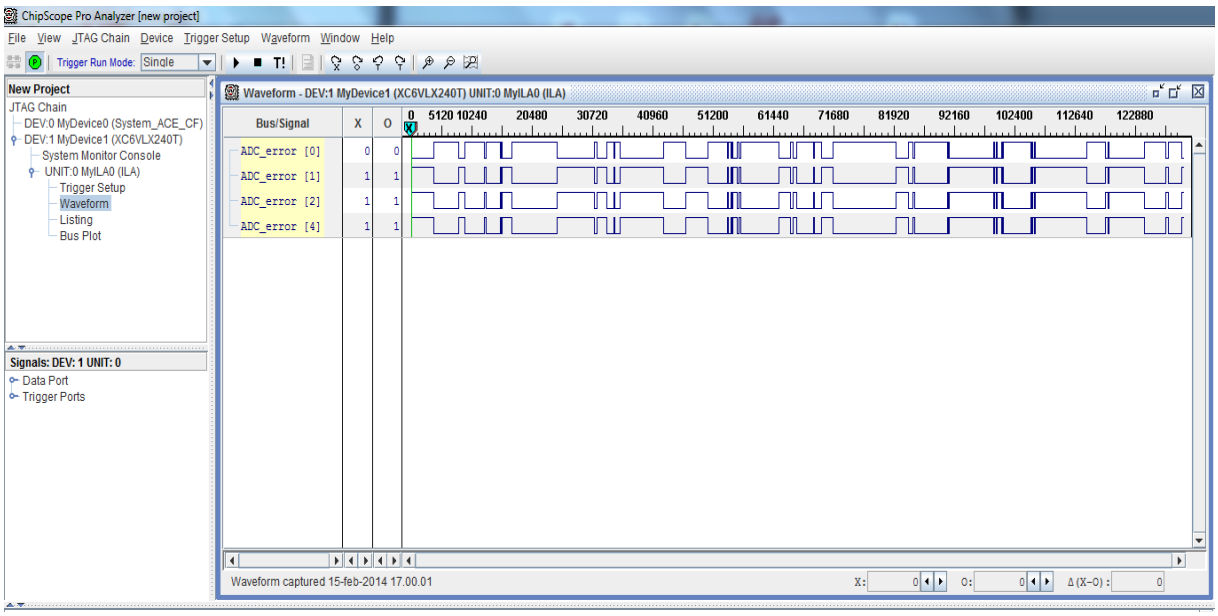

Figure 5.9: ChipScope Pro Analyzer interface: input error.

5.1.3 FPGA I/O mapping

To manage I/O on the FPGA is necessary add the UCF file into the design hierarchy. This file contains a list of implementation constraints given to the FPGA implementation tools to direct the mapping, placement, timing or other guidelines for the implementation tools to follow while processing an FPGA design. Implementation constraints are generally placed in the UCF file, but may exist in the HDL code, or in a synthesis constraints file.

Examples of implementation constraints are both LOC (placement) and PERIOD (timing) constraints. The UCF in Fig.5.10 is related to the system **digital_control_loop**, it compromises constraints to configure the clock generator, the reset signal and the I/O related to the FPGA-TC interface. The *clock_generator* is a block which generates a $70MHz$ clock from a differential input clock of $200MHz$, the constraint is needed to drive the differential clock to this block. In the ml605 evaluation board the position of the $200MHz$ ring oscillator is J9 and H9, for this reason the LOC constraints refers to these positions (raw 3 and 4). Moreover, the timing constraint of $200MHz$ has been used. The constraint on the **clock_generator** output clock ($70MHz$) is LOC=W34, this is needed to drive the clock system to an SMA connector of the Virtex6. A further LOC constraint is related to reset of the system, inside the top module this signal is connected to each block (Fig.5.1) of the system and with the UCF is connected to a push-button in position H10.
The 4 bits ADC error coming from the TC are read respectively in position AE12, AK11, AK12 and T28. The DPWM output (*pwm_o* or $d(t)$) and the ADC clock (*sample_adc*) have been respectively located on V34 and M22 (SMA connectors).

```
# System clock settings (clock_generator)
#Input  clock
Net "clk_in_i_P" LOC=J9 |  IOSTANDARD=LVDS_25  |  DIFF_TERM = TRUE;
Net "clk_in_i_N" LOC=H9 |  IOSTANDARD=LVDS_25  |  DIFF_TERM = TRUE;
Net "clk_in_i_P" TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 200000 kHz;
#Output clock
NET "clk_o"                          LOC = "W34";    ## 1   on J56 SMA    -- 70MHz
NET "clk_o" TNM_NET = "CLK_OUT1";
TIMESPEC "TS_CLK_OUT1" = PERIOD "CLK_OUT1" 70.002 MHz;

#Reset
Net "rst_i" PULLUP;
Net "rst_i" LOC= H10 | IOSTANDARD=SSTL15;
PIN "rst_i" TIG;
NET "rst_i" CLOCK_DEDICATED_ROUTE = FALSE;

# OUTPUTS to the Test Chip
NET "sample_adc_o"                   LOC = "V34";    ## 1   on J57 SMA
NET "pwm_o"                          LOC = "M22";    ## 1   on J55 SMA

# INPUTs from the Test Chip
NET "adc_i<0>"  LOC = "AE12";   ## 1   on J41
NET "adc_i<1>"  LOC = "AK11";   ## 3   on J41
NET "adc_i<2>"   LOC = "AK12";   ## 9   on J41
NET "adc_i<3>"  LOC = "T28";    ## 11  on J41
```

Figure 5.10: FPGA I/O prototype configuration: UCF file.

The I/O mapping have been summarised in Fig.5.11, where three SMA connectors have been used to bring the ADC clock, the DPWM output and the clock system to the TC. Furthermore, the reset on a push-button and the connector for the incoming 4 bits (ADC error) have been highlighted.

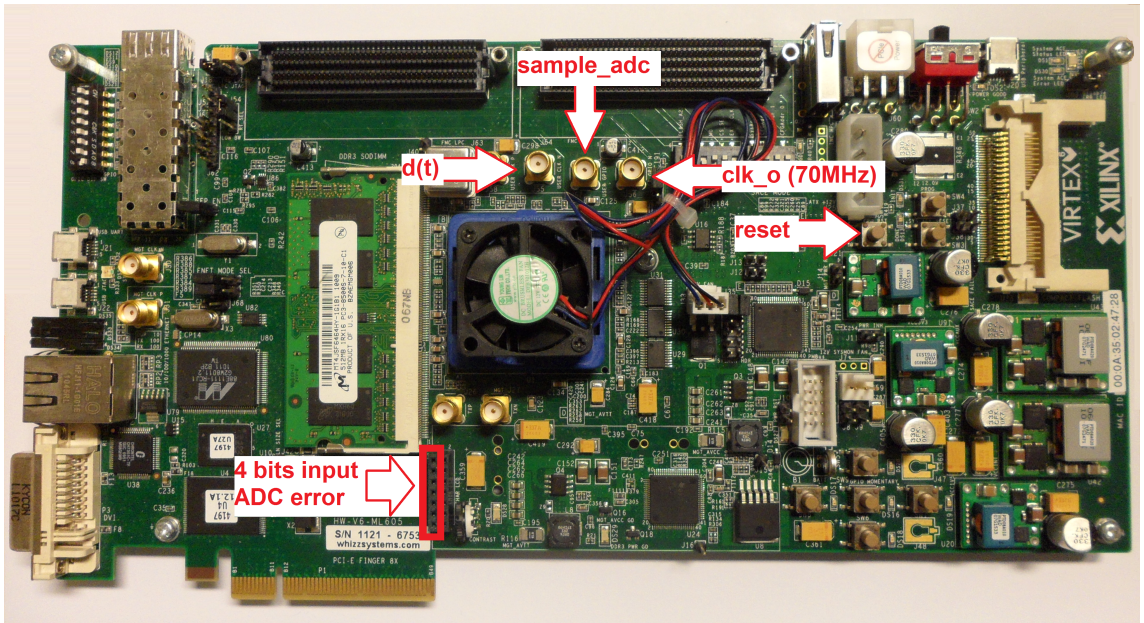

Figure 5.11: Virtex6 FPGA: I/O mapping

Through the constraints file and the clock wizard, the FPGA have been set for the prototyping realisation (Fig.5.1). Moreover, using the Chipscope Pro tool, signals can be monitored directly on the FPGA. With these setting, the FPGA is ready to be connected with the TC to realise the system in Fig.5.2

5.2 Test Chip external loop configuration

The Test Chip configuration for the prototyping is modelled in Fig.5.1. The external control loop configuration, permits to use both the buck converter and the 4 bits ADC to obtain the model in Fig.5.2. In this configuration the analog block are not emulated (as for the co-simulation) and the TC can be interfaced with the **digital_control_loop**. The external loop TC setting is obtained via-SPI programming interface (Fig.5.2). In Fig.5.12 is the graphical interface used to set the TC for the communication with the FPGA, this configuration is written on the TC registers thanks to the SPI interface. This TC6 Control Software is used to set the POM and both the ADC and buck converter registers. The ADC_CTRL1 register through its filed ADC_CLK_EXT, permits to configure the AD converter in order to be externally clocked. The start of conversion signal is then received from the FPGA through the SMA interface (Fig.5.13). Optionally, in the the ADC_CTRL2 the DIV_IDAC_LSB register can be set to move the the ADC LSB size according with Tab.5.1.
The register MODE_CTRL1 among Buck General Registers comprises the path EXT_LC. This setting is the external loop configuration for the buck converter, the square wave to drive the power-MOS on the TC have to come from the FPGA through the SMA interface (Fig.5.13). The POM is an eight bit configurable I/O register, the graphical interface permits to chose each signal can be read/write from the 8 pin port in Fig.5.13. In Fig.5.12, the POM is set to write the 4 bits ADC output respectively on 4 pins (a fifth one can be optionally used as debug function, to write out the ADC clock received from the FPGA). The POM I/O interface on the TC is highlighted on the TC in Fig.5.13, the ADC output is read through a 4 pins connection on the FPGA (5.11).

The TC I/O mapping have been summarised in Fig.5.13, where three SMA connectors have been used to receive the ADC clock, the DPWM output and the clock system to from the FPGA. Furthermore, the POM with 4 bits ADC output error have been highlighted.

Once that the FPGA and the TC have been configured with the respective prototyping settings, the interfacing (modelled in Fig.5.1) has been realised with three SMA connectors (to connect the clock system, DPWM square wave and the ADC clock sample) and four pins (to connect the ADC bits).
When the TC has been configured in external loop, the **digital_control_loop** as in Fig.5.1 can be mapped on the Virtex6 in order to realize the closed loop prototype. The setting of the entire system proposed in Fig.5.1 has been realised and shown in Fig.5.2. Digital control signals can be monitored on the FPGA (Fig.5.9) and signals on the TC through the oscilloscope (Fig.5.3). The closed loop configuration have been tested, in Fig.5.5 a 50% duty cycle is obtained for $V_{ref} = 5V$ and $V_{in} = 10V$. In Fig.5.3 an output voltage of $5V$ is obtained through FPGA-based regulation.

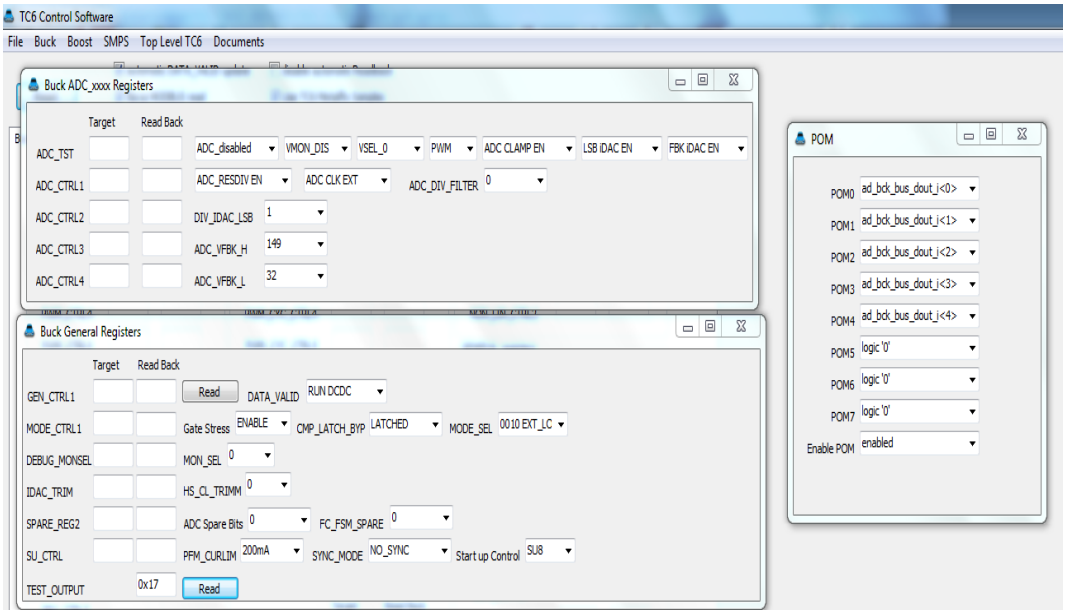

Figure 5.12: Test Chip GUI interface: external loop configuration.

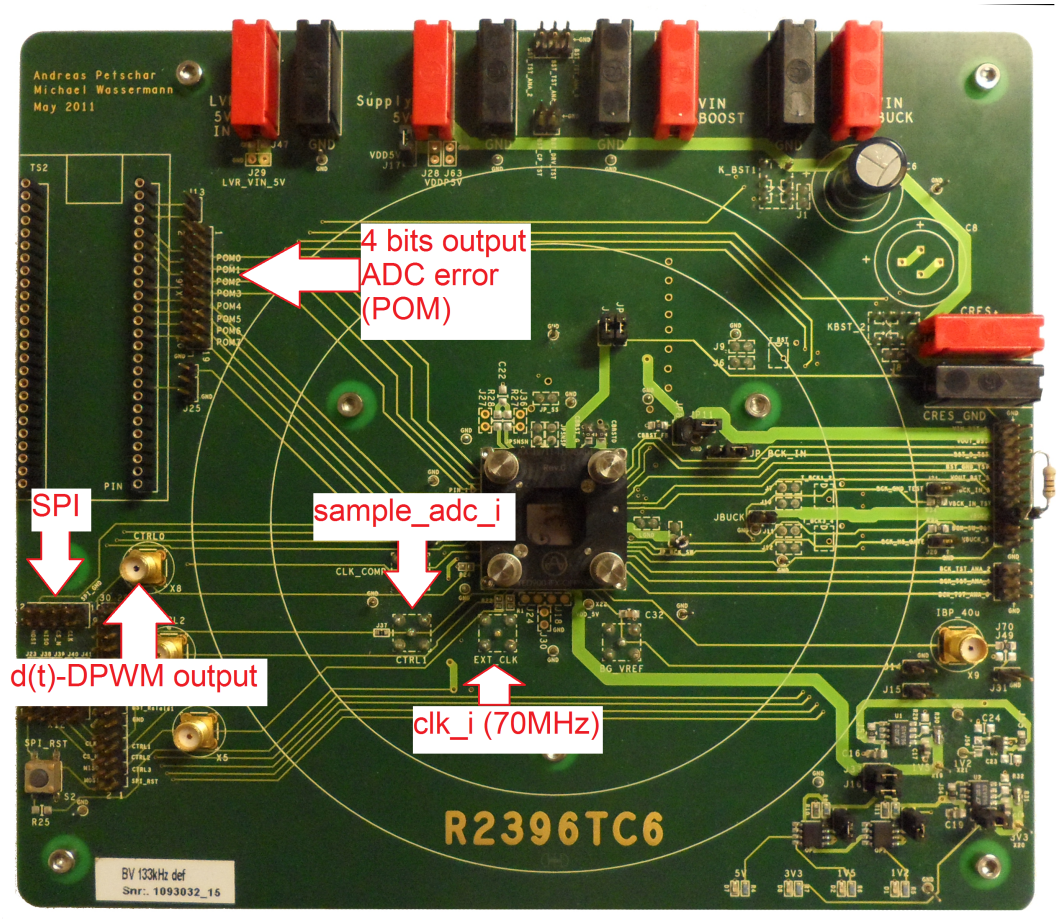

Figure 5.13: Test Chip: I/O mapping.

Chapter 6

Modelling of system identification techniques at the State of Art

The high programmability level and the computational power, both joined with the possibility to have complex control solutions have pushed the research on digitally controlled SMPS. The digital approach enables smart solutions for regulators, e.g. self-tuning or autotuning capabilities of a regulator. The main goal of an autotuning algorithm is the optimization of the control law for the closed loop system, compensator gains can be automatically set after that load (dcdc output filter) considerations have been drawn. The dynamic adjustment of PID coefficients to improve margins and bandwidth of the system, permits to use the same control loop to integrate power supplies for different set of applications, hence, reducing development time and R & D costs. The static configuration of a compensator does not allow the best system bandwidth for a large set of loads (off-line controllers). Joining the feasibility to identify and/or monitor the dcdc output filter, with the dynamic setting of the compensator gains, avoid the duty to study the compensator (on-line controllers). The digital PID compensator can be automatically set after the load identification, furthermore, the coefficient dynamic adjustment can be done during the steady state through the load monitoring. If non-idealities like the ESR contribution occurs, the load identification have be performed during the steady state. Monitoring joined with the consequent PID coefficient change overcome the problem to have situations of non-optimal regulation. In Fig.6.1 has been modelled the on-line controller approach, where the classical control feedback loop is combined with the self-tuning algorithm.
The algorithm is able to identify the load and automatically set the controller configuration. The same digital control loop can be automatically optimized for each load over a wide range.

.

The system identification (SI) generally falls into two main categories: parametric and non-parametric methods [57, 58]. Parametric methods return the parameters of the system model such as the coefficients of a system difference equation, transfer function, or state-space model. Non-parametric methods return impulse response and/or frequency response data directly.
Parametric methods require, the selection of an appropriate input stimulus, a priori selection of a parametrized model structure including system order and number of zeros, the

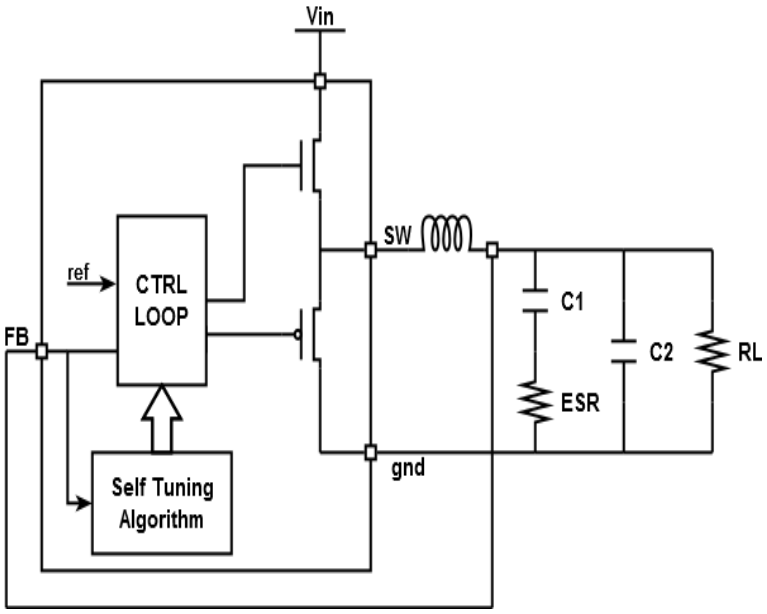

Figure 6.1: Closed loop configuration and self-tuning algorithm: on-line controller model.

construction of a suitable prediction error equation and loss function, and methods to minimize the loss function [17, 4, 5, 72, 90, 16, 60, 59, 26, 105, 73, 47]. Non-parametric methods do not assume a system model and require only selection of an appropriate stimulus. The LCO-based self-tuning methods are an example of non-parametric approaches.
System LCOs can be induced either by using a relay [25, 102, 104] or reducing the DPWM resolution [135, 136, 139]. However, this kind of approach allows for the frequency loop response to be known only at the stimulated frequency. Another non-parametric method, obtains an approximation of the system impulsive response exploiting the cross-correlation method. For a white noise input the impulse response of the system is proportional to the cross-correlation between input and output, while the correlation itself rejects any disturbances to the system as long as they are uncorrelated with the input [57, 58, 66, 67, 14, 13, 103, 101, 97]. In these approaches a Pseudo Random Binary Sequence(PRBS)is used to emulate a digital white noise source and perturb the closed loop system. Such identification method has been integrated into autoregulated digital converters, with different control techniques. In this chapter, main non-parametric methods are presented with their modelling in the floating point models presented in Cha.4.
Nowadays, the distinction between parametric and non-parametric methods is not so defined and fully shared by researchers in this field. This chapter is mainly focused on non-parametric models, the interest is to highlight the main approaches to obtain the system impulse response using only stimulus injection and avoiding mathematical models. In non-parametric algorithms the autoregulation is usually based on two steps. Firstly a signal is injected for the load identification step, then the best compensator gains are computed (regulation). Some parametric approaches are based on mathematical models that minimize the error by changing the compensator gains, hence, the regulation step is part of the identification.

Among non-parametric SI, a further distinction can be done between steady state SI

and open loop SI technique. The steady state SI adds a little perturbation during nominal converter operations and is able to monitor possible non-idealities occurring during steady state operations. For the on-line controllers based on this approach, the PID tuning can be repeated during the steady state and possible ESR contributions (due to temperature variations) can be compensated. On the counterpart, open loop SI techniques can be performed before the system start up in order to set the best PID configuration in relationship with the identified load. However this kind of approach does not permit to update the PID configuration during the converter operations, indeed they cannot be performed during closed loop operations. The LCO-based approaches are mainly open loop SI techniques, while, PRBS-based approach can works during the steady state.

6.1 Non-parametric methods

Non-parametric methods are based on the injection of signal to identify the load. In this approach the identification of the load is the heart of the algorithm and, for instance, compensation gains can be pre-computed and stored in a LUT where can be retrieved to increase the bandwidth of identified output filter. The main non-parametric methods are based on:

- causing Limit cycle oscillations (LCOs) [25, 102, 104, 135, 136, 139]
- studying the harmonic response [48]
- injecting pseudo random binary sequence (PBRS) [66, 67, 14, 13, 103, 101, 101, 97]

The main disadvantage of the LCOs and harmonic response techniques is that the load identification is strongly reflected on the output voltage and the closed loop system cannot be used during the load identification. These techniques cannot be used to monitor load variations monitoring during the steady state.

6.1.1 LCO-based methods

As mentioned in Sec.3.3.3, LCOs are unwanted oscillations of the output voltage. Some load identification approaches exploit the LCOs to extract load information as, for instance, the output filter resonant frequency. The LCO-based methods induce oscillation by causing the absence of one these conditions:

- the DPWM resolution must be smaller the LSB_{ADC}. (static condition)
- the Nyquist criteria must be respected. (dynamic condition)

Methods based on the lack of the static condition have been presented in [135, 136, 139], while the dynamic condition has been exploited in [25, 102, 104].
The dynamic condition method is basically related to the Nyquist criteria:

$$N(a)G(j\omega_x) = -1, \tag{6.1}$$

where $N(a)$ is constant with frequency and is the relay describing function presented in Eq.6.2. On the basis of Eq.6.1, the oscillations at ω_x will occur when the phase of the system $G(j\omega)$ is $-180°$.

$$N(a) = \frac{4D_r}{\pi\sqrt{a^2-\epsilon^2}+j\epsilon}. \tag{6.2}$$

In Eq.6.2 D_r is the magnitude of the relay output square waveform, while ϵ is the width of the hysteresis needed to avoid multiple zero crossing due to the noise in the sensed variable. Because imaginary contribution in the describing function can be neglected, the function $\frac{1}{N(a)}$ can be considered on the real axis in the complex plane and the system continue to oscillate when the phase of $G(s)$ is very closed to $-180\,°$.
The identification technique is interested on the identification of the resonant frequency $f0$. The buck converter control to output transfer function ($G_{vd}(s)$) is mainly composed of two complex conjugated poles, then, the phase displacement at $f0$ is of $-90°$. Adding an integrator in the control loop, a further contribution of $-90°$is inserted over all the frequencies and, a phase contribution of $-180°$will result at the resonant frequency $\omega_0 = 2\pi f_0$. Inserting then a relay in the control loop, the system will oscillate at the resonant frequency.
This approach has been verified in the floating point model introduced in Cha.4. Let's consider the closed loop system is shown in Fig.6.2 ($G(s) = G_{vd}(s)$), the compensator (PID) is removed and both the relay block and the pole ($1/s$) are inserted. The further $-90°$phase displacement inserted by the integrator, joined with the relay induce oscillation in the system at the output filter resonant frequency ($f_0 = \frac{1}{2\pi\sqrt{LC}}$). In Fig.6.3 are shown the out-

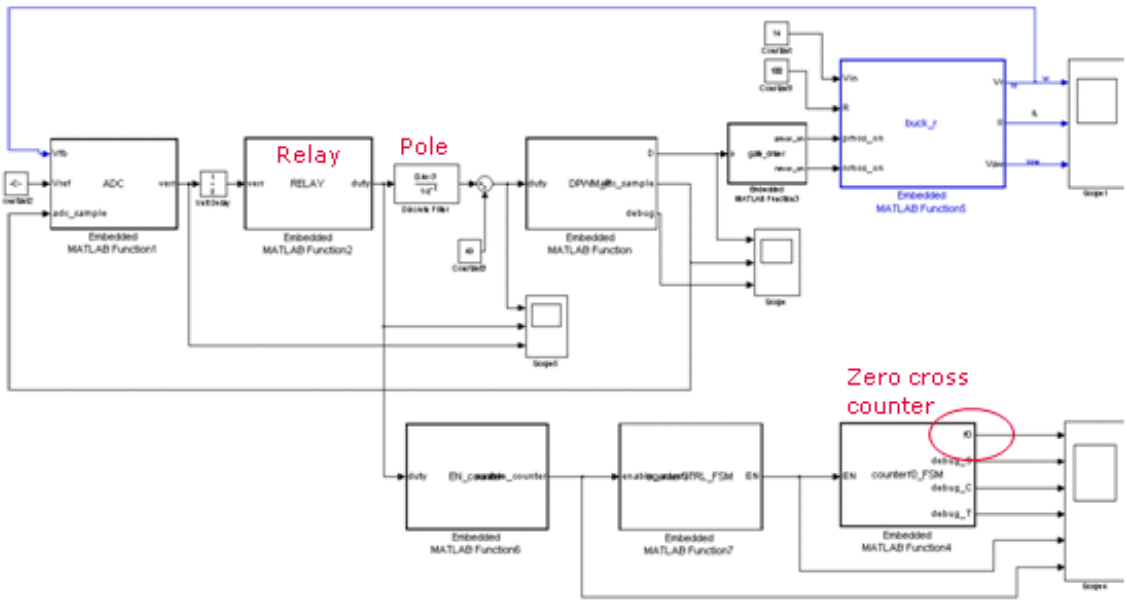

Figure 6.2: Matlab/Simulink model: LCO-based identification method.

put voltage, the inductor current and the relay output related to the system presented in Fig.6.2. The LCOs frequencies at the resonant frequency can be observed on the output voltage, moreover, both the error voltage and the duty cycle have the same oscillation frequency. A square waveform results much easier to be processed, just counting zero crossings is easy to extract the resonant frequency from the relay output waveform. In the Matlab/Simulink model shown in Fig.6.3, three block are dedicate to extract the resonant frequency from the oscillations, two blocks are used to eliminate the firsts stabilizing oscillations (non-constant period) and the **zero cross counter** to calculate the cut-off frequency $f0$. The counter is enabled for two periods after the first zero crossing. In this example the ideal

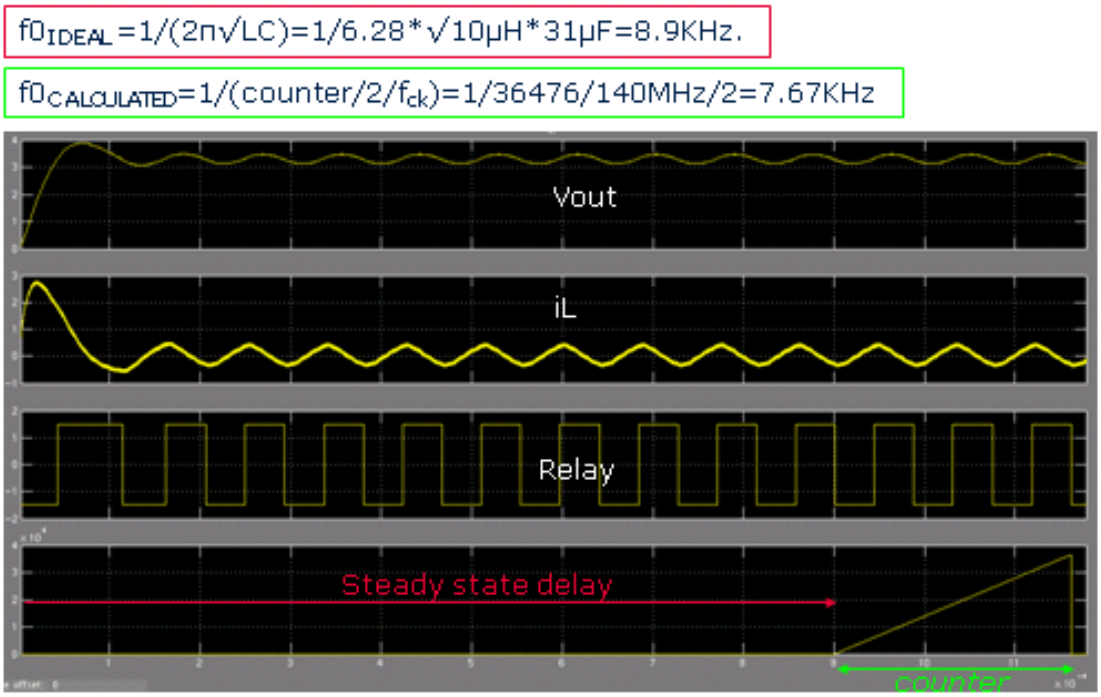

Figure 6.3: Matlab/Simulink model: LCO-based identification method output.

resonant frequency is $f_0 = \frac{1}{2\pi\sqrt{LC}} = 8.9kHz$ ($L = 10\mu H C = 31\mu F$), the extracted frequency is $f_{0,extracted} = \frac{2f_{clk}}{counter} = \frac{2*140MHz}{36476} = 7.67kHz$, about $1kHz$ lower than the ideal value. The difference can be attributed mainly to the imaginary contribution introduced by the hysteresis. This approach is concentrated on obtaining the frequency response of the system at one frequency, non-idealities like the ESR contributions are neglected and could be considered only through the PID coefficients tuning. An iterative procedure is done to calibrate the PID gains, the first zero is fixed at the extracted resonant frequency, while, both second zero and the gain are set with an iterative procedure to respectively satisfy phase margins and the required bandwidth [25]. The same LCO-based procedure is used for the identification purposes in[135], the main difference is related on the tuning of the PID gain. The PID gain is adjusted by considering the phase displacement between the duty cycle and an introduced sinusoidal perturbation, having frequency and phase respectively equal to the desired bandwidth and margin. This method to monitor both phase margin and bandwidth, comes directly from the Middlebrook's method ([29]), it is well known in the analog world and exploited in [74] to design an on-line stability margin monitor for a digitally controlled SMPS.

Different LCO-based approaches exploits the static condition. A controller temporally reduces the DPWM resolution and, when oscillations occur, the extraction of the cut-off frequency and load information are read from the duty-cycle behaviour. The amplitude and the period of the LCO are both influenced by the input voltage and the compensator gains and parameters extraction is however complicated from a mathematical point of view.

6.1.2 Harmonic response analysis

This kind of approach has been exploited in[48]. The harmonic response of a system $y(t)$ is defined for a sinusoidal input $x(t)$:

$$x(t) = Asin(\omega_x t) \rightarrow X(f) = A\left(\sum_0^\infty x(t)e^{j\omega t}\right). \tag{6.3}$$

$$y(t) = A\left|G(j\omega_x)\right| sin(\omega_x t + arg(G(j\omega_x))) \rightarrow Y(f) = A\left|G(j\omega_x)\right|\left(\sum_0^\infty x(t)e^{j\omega t}\right)e^{j\omega[arg(G(j\omega_x))]}. \tag{6.4}$$

On the basis of Eq.6.3 and Eq.6.4, for a sinusoidal input at frequency ω_x the system response in the frequency domain ($Y(f)$) have an amplitude equal to the product between its amplitude and the gain of the system transfer function ($G(jw)$) at the same frequency. Moreover, its phase is shifted respect to the input signal by the phase of the system response at ω_x. Making the ratio between output and input in the frequency domain, the frequency response at ω_x is obtained in terms of module and phase. Let's refer these assumptions to the control to output transfer function ($G_{vd}(s)$) of a digitally controlled SMPS:

$$\frac{V_{out}(f)}{D(f)} = \frac{Y(f)}{X(f)} = \left|G_{vd}(j\omega_x)\right| e^{j\omega[arg(G(j\omega_x))]}. \tag{6.5}$$

for each considered ω_x, the control to output transfer function can be obtained in terms of magnitude and phase through the ratio Eq.6.5. In Fig.6.4 is shown the floating point digital control loop and, in the lower part, blocks needed for the harmonic response identification technique (Eq.6.5). An oscillator is needed to generate *sine* and *cosine* signals at the desired frequency ω_x, while four correlator blocks are needed to evaluate real and imaginary part both for input and output signals. The implemented correlator function summarised in Eq.6.6 for a generic function $f(t)$, is applied in the model shown Fig.6.4 both for the input sinusoidal duty cycle and the output voltage.

$$\int_0^\infty f(t)e^{j\omega t}\,dt = \sum_0^\infty f(t)e^{j\omega t} = \sum_0^\infty f(t)(cos(\omega t) + jsin(\omega t)). \tag{6.6}$$

Input sinusoidal duty cycle and the converter output voltage are respectively represented as complex numbers at the correlator output and, then, expressed in terms of magnitude and phase to compute the Eq.6.5. Computing the ratio between output and input modules and the subtraction between output and input phases, respectively magnitude and phase of $G(i\omega)|_{\omega_x}$ are obtained.
This technique is a point-wise calculation of the bode diagram related to the control to output transfer function of the converter, for each chosen ω_x the computation have to be repeated and can be expansive in term of latency of the identification process. Results for the harmonic technique applied at the resonant frequency ($\omega_x = \omega_0$) can be observed in Fig.6.5. The dc value of the control to output transfer function is related to the input voltage, a magnitude value of $13V/V$ is obtained for an input voltage of $14V$. The expected phase contribution of $G_{vd}(j\omega)$ at the resonant frequency is of $-90°$, the obtained one in our model is of $-91°$.
This approach could be used for the PID parameters tuning, a sinusoidal signal is inserted at the desired bandwidth and compensator gains adjusted according with the system response.

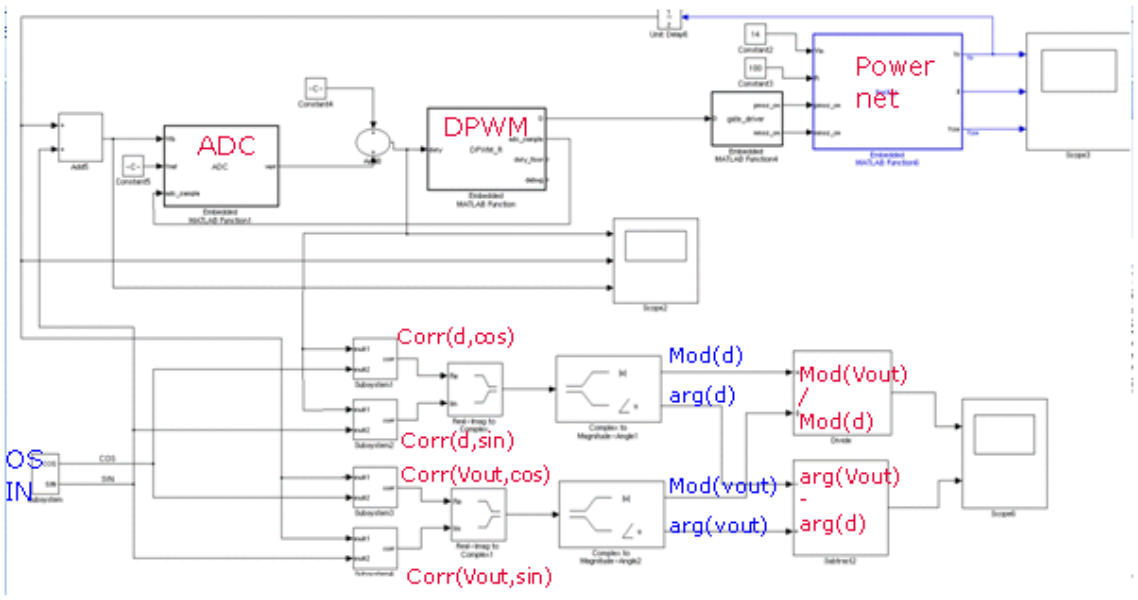

Figure 6.4: Matlab/Simulink model: harmonic system response identification method.

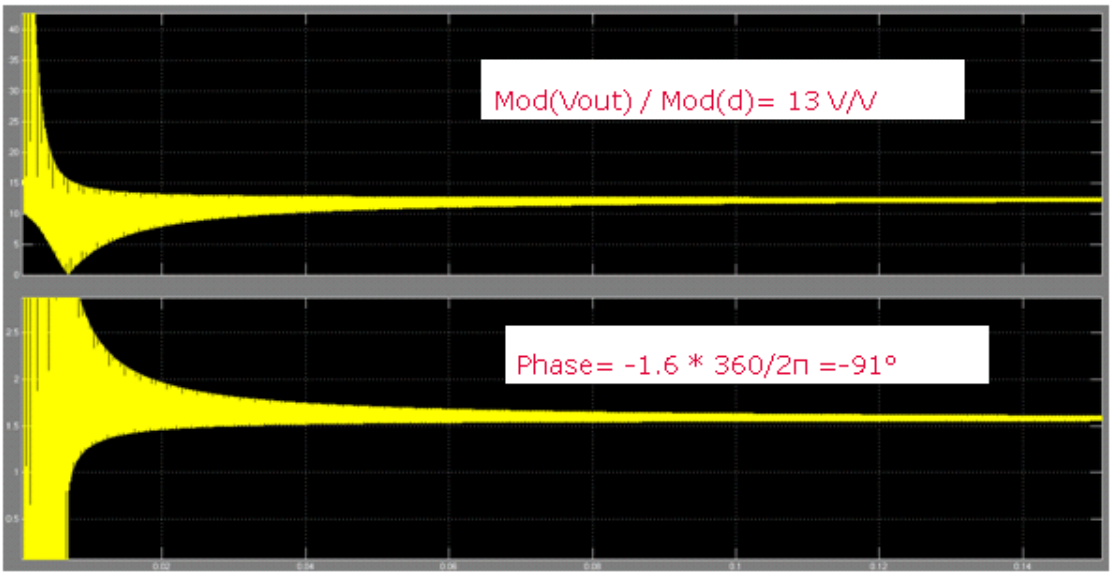

Figure 6.5: Matlab/Simulink model: harmonic system response identification method output

6.1.3 PRBS-based methods

This technique is mainly based on the properties of the cross-correlation for a input white noise signal. Considering small signal disturbance, a power converter during steady state is assumed as linear time-invariant discrete time system. The sampled system can be described by the Eq.6.7, where $y(n)$ is the sampled output, $h(k)$ the discrete time system impulse response, $u(k)$ the input digital control signal and $v(n)$ the disturbances (for instance both switching and quantization noise).

$$y(n) = \sum_{K=1}^{\infty} h(k)u(n-k) + v(n). \tag{6.7}$$

The cross-correlation $R_{uy}(m)$ between the input control signal $u(n)$ and output signal $y(n)$ is obtained substituting the Eq.6.7:

$$R_{uy}(m) = \sum_{n=1}^{\infty} u(n)y(n+m) = \sum_{n=1}^{\infty} h(k)R_{uu}(m-n) + R_{uv}(m), \tag{6.8}$$

where $R_{uu}(m)$ is the autocorrelation of the input signal and $R_{uv}(m)$ is the cross-correlation between input and noise.
Let's consider autocorrelation properties when the input control $u(k)$ is an ideal white noise signal:

$$\begin{cases} R_{uu}(m) = \delta(m) \\ R_{uu}(m) = 0, \end{cases} \tag{6.9}$$

the autocorrelation of the white noise is an ideal delta function, and the cross-correlation between the white noise and the disturbances $v(n)$ is ideally zero. Assuming these considerations, the cross-correlation for a white noise input is the discrete time impulse response:

$$R_{uy}(m) = h(m). \tag{6.10}$$

The control to output transfer function in the frequency domain can be derived by applying the discrete Fourier transform (DFT) to the cross-correlation output Eq.6.10:

$$R_{uy}(m) \rightarrow G_{vd}(j\omega) \tag{6.11}$$

All this assumptions are true if the input control signal is an ideal white noise. Practical digital implementations emulate the white noise considering a finite number of bits. The Pseudo Random Binary Sequence (PRBS) based on the Maximum Length Sequence (MLS), is used to approximate the white noise. A PRBS is a deterministic periodic signal that can be easily generated in hardware using a shift register and feedback taps[62]. For a p-bit shift register, the location of the taps can be selected so that the period of the resulting sequence of 0's and 1's has the maximal length of $N = 2^p - 1$. If an approximation of the white noise is obtained, the autocorrelation of a finite length PRBS is not a perfect impulse and some noisy components are present around $\delta(m)$ (Fig.6.6). This undesired effect is reflected on the obtained control to output transfer function, indeed, a big amount of noise appears at the high frequencies $G_{vd}(j\omega)$) introducing some resolution problems on $G_{vd}(j\omega)$. A first approach to reduce the high frequency noise is to use a maximum length PRBS repeated L times, so the result is an L-period PRBS. Then, the cross-correlation is averaged over multiple periods to get the system impulse response. Whit this approach the noise is reduced but not eliminated yet. Many PRBS-based identification methods at the State of the Art, try to reduce the noise at medium-high frequencies. There are mainly two further approaches:

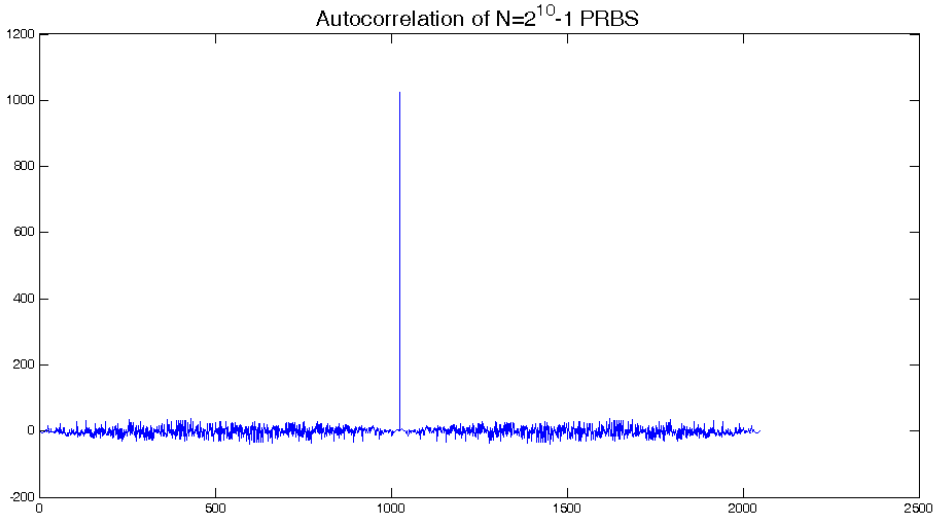

Figure 6.6: Pseudo Random Binary Sequence: Autocorrelation (2^{10}samples) .

- delaying the sampling of the output voltage by half of the test sequence clock period, the $G_{vd}(j\omega)$ dynamics are better captured. The control sequence During each clock cycle has only a single value of perturbation and the output changes continuously within the same interval.
- Knowing the approximation of the true impulse response, a windowing effects can improve the high frequency estimation [103].

6.2 Parametric methods

Parametric methods require the selection of an appropriate input stimulus and a priori selection of a parametrised model structure, that includes system order and number of zeros. The construction of a suitable prediction error equation and methods to minimize the loss function are both necessary as well. The least square method has been introduced in [29][57]. This approach is an off-line parameter estimation method based on a set of measurements like:

$$\bar{y}(k) = \bar{f}(k,p) + \bar{e}(k), \tag{6.12}$$

where k is the number of measurements, $\bar{y}(k)$ is the measurements vector, $\bar{f}(k,p)$ is the known vector function and $\bar{e}(k)$ is the unknown error vector. The fundamental notion of least square estimation, is to evaluate the unknown parameters that minimizes the sums of squared differences between the known model and the measurements. It tries to minimize the error function[9, 50, 4, 5, 90, 47].
knowing the approximation of the control to output transfer function the least squares method could be used for the model fitting procedure. The PRBS-based method could be used to approximate the control to output transfer function (noise contributions are present at medium-high frequencies), then the least squares parametric method can be applied to

fit the approximated plant with a parametric model[9]. The parametric model Eq.6.14, is a rational function where N and M are respectively numerator and denominator order. Numerator and denominator coefficients are grouped respectively in two vectors, $\bar{A}$ and $\bar{B}$ respectively for the numerator and denominator coefficients.

$$G_{candidate}(s) = \frac{A_N s^N + A_{N-1} s^{N-1} + ... + A_0}{B_N s^N + B_{N-1} s^{N-1} + ... + B_0}. \tag{6.13}$$

The fitting problem is to find the coefficients of the candidate plant, in order to minimize the error model described by the fitting error $\epsilon_{fit} = (j\omega_k)$:

$$\epsilon_{fit} = (j\omega_k) = \left| G_{candidate}(s) - G_{uy}(s) \right|_{s=j\omega_k}. \tag{6.14}$$

To evaluate the quality of the fitting, the method least squares uses the cost function J_{WLS} defined as the weighted sum of the ϵ_{fit} over the frequency indices of interest:

$$J_{WLS} = \sum_k W(j\omega_k) * \left(G_{candidate}(s)|_{j\omega_k} - G_{uy}(s) \right)^2. \tag{6.15}$$

The coefficients of the candidate plant are managed by iterative numerical methods (such as Gaussian-Newton algorithm) to minimize the cost function. The choice of the weighting function can be done taking care about the model to fit. It has been observed that, the control to output transfer function obtained whit the non-parametric correlation method is usually noisy in the upper decade of the interested frequency range, then, the weighting function has to equally prioritize the fitting across the frequency range of interest and the importance of fitting can be logarithmically de-emphasized with increasing frequency[9].

Chapter 7

Novel non-parametric system identification methods

Two novel system identification (SI) technique for self-tuning algorithms are proposed and analysed in this chapter. Compensator parameters are usually static and set to work on a wide range of loads (off-line controller), knowing the load configuration is possible to set the best PID coefficients for each buck converter configuration in order to optimize bandwidth and margins (on-line controllers). The self-tuning method is most of times a two step approach, after the load identification (buck converter output filter) the best compensator is set to improve the system bandwidth.
As introduced in the previous chapter, non-parametric identification techniques are based on system perturbation. Most of them do not work during the steady state. This kind of approach cannot be used for monitoring possible load variations during the steady state, where some non-idealities (e.g. the ESR contribution) can change the system dynamic. On the counterpart, the identification techniques that work during the steady state (steady state SI), are able to identify these non-idealities and consequently adapt the PID coefficients. When non-idealities like temperature variations occur, the ESR contribution and the related

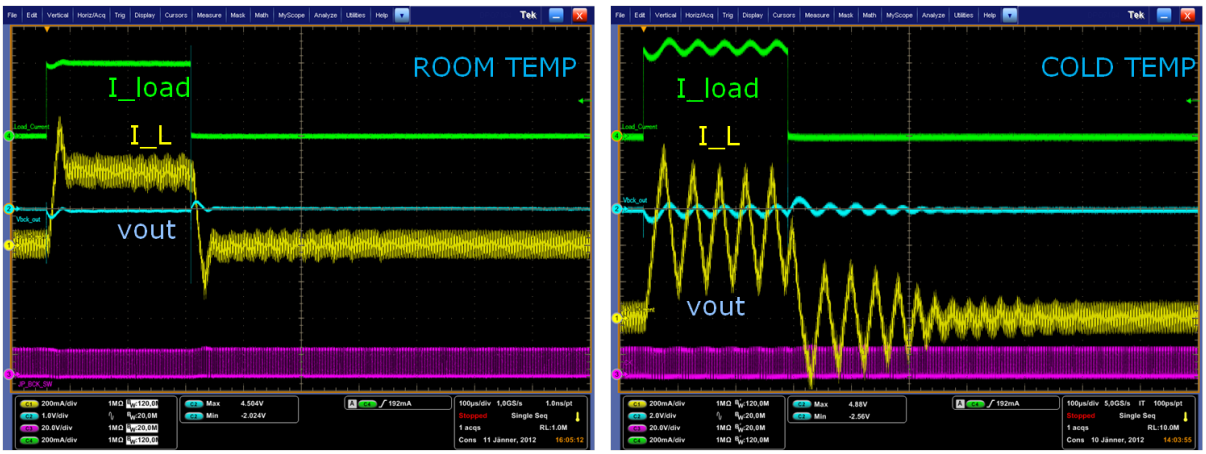

Figure 7.1: Non-idealities: ESR contribution effects on the steady state output waveforms.

zero contribution can bring the system to have poor stability margins. The load step system response for the same system configuration (same PID) has been shown in Fig.7.1, both when the ESR contribution is null (at room temperature) and its contribution increases (cold temperature). When the ESR contribution is considered, the system recovery time is much longer and LCOs are induced in the output voltage. Having poor stability margins, it brings the system to be slower in terms of recovery time of the output voltage and, moreover, bigger ripple effects are visible on the output voltage. In this case, is fundamental to study the buck converter control to output transfer function (identification process) and consequently adapt the PID gains in order to increase margins and bandwidth.

The SI algorithms presented in this chapter, are both focused on extracting the output filter resonant frequency (f_0) and the ESR contribution (zero contribution f_z).
The open loop SI technique, exploits the relationship between the step and impulsive response of the system in order to extract the control to output transfer function of the buck converter. It can be applied during the system start up in order to identify the load and consequently tune the PID parameters before that the steady state is reached.
The steady state SI extracts the load parameters through the amplification of dithering effect introduced by the $\Delta\Sigma$ modulator. The dithering amplification is reflected on the output voltage during the steady state, consequently its effects can be observed in the ADC output error $e[n]$. Observing the product between $\Delta\Sigma$ noise transfer function ($NTF(z)$) and the control to output transfer function ($G_{vd}(s)$) of the buck converter, information about the output filter in the frequency domain can be obtained. The basic idea, is to amplify the quantization noise in order identity the output filter parameters. During the steady state, perturbation effects can be observed inside the ADC zero error.
Results presented in this chapter, refer to the fixed-point model described in Cha.5 and modelled in fig.7.2. For the steady state identification method two further inputs have to be considered for the $\Delta\Sigma$ modulator, α is used to amplify the dithering and f_n to introduce a notch in the noise transfer function during the ESR identification.

7.1 Open loop system identification technique

The system start up is related to the reference voltage behaviour. In real implementations a soft start up is preferred, for instance, it can be obtained with a linear evolution from 0 to the desired dc output value V_{ref} of the reference voltage.
Let's considered the start up open loop configuration presented in Sec.4.2.3. During the start up both PID and $\Delta\Sigma$ modulator are excluded and the system evolution is handled only by the DPWM. The DPWM increases the duty cycle step by step and detect the steady state condition when the difference between the reference and the output voltage is $e[n] = 0$. A step evolving reference voltage could cause intermediate open loop steady states conditions. In Fig.7.3 is shown the example of a reference voltage that evolves with three steps before reaching $V_{ref} = 3.3V$. The output voltage evolution has been shown in Fig.7.4, some intermediate steady states containing the buck converter step response can be induced in the output voltage. In Fig.7.5 is shown the output voltage consequent to a V_{ref} step.
Because reference voltage steps occur in a open loop configuration, the system in this case is mainly represented by the buck converter control to output transfer function $G_{vd}(s)$. The system step response contains information about poles and zeros of the $G_{vd}(s)$ and it is strictly related to the impulsive response through a derivative connection. The output volt-

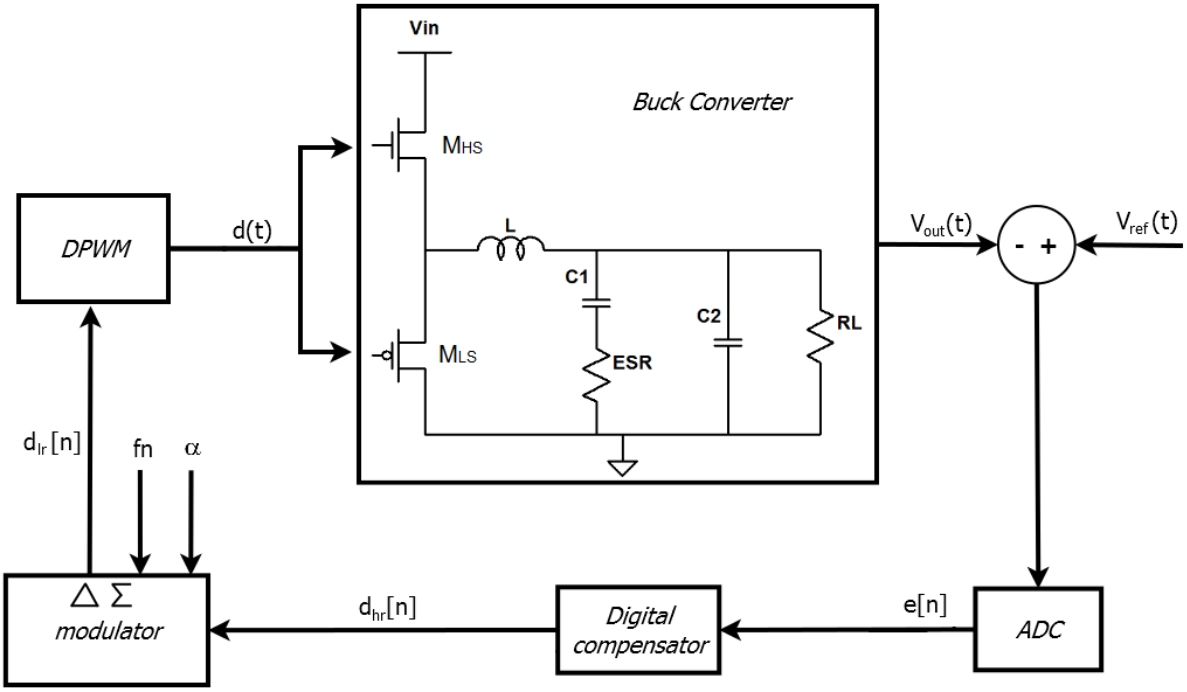

Figure 7.2: Digitally controlled buck converter: steady state identification model.

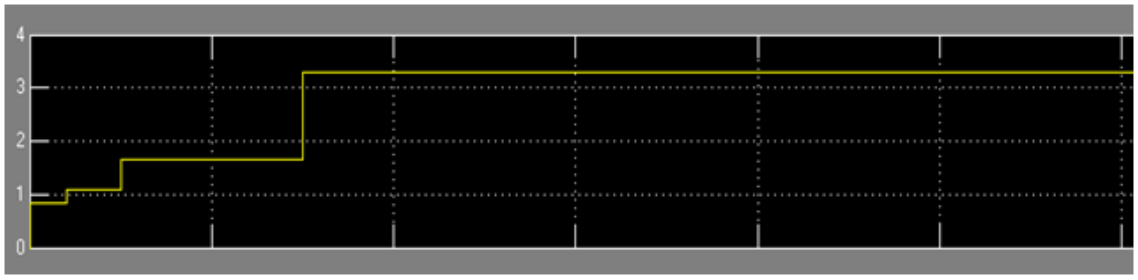

Figure 7.3: Open loop identification method: step evolving reference voltage.

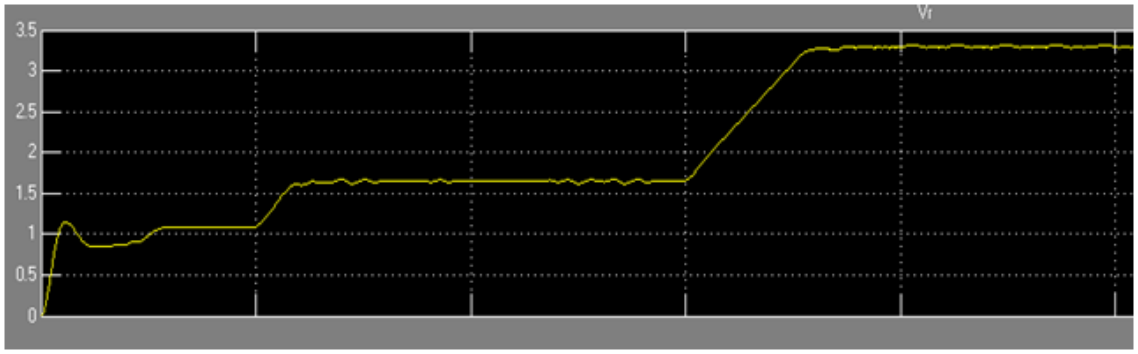

Figure 7.4: Open loop identification method: step response output voltage.

age response can be observed at the ADC output, the error $e[n]$ during the intermediates open loop steady state contains the digital representation of the step response. The step response $y(t)$ for a buck converter $G_{vd}(s)$ is the Laplace anti-transform of:

$$y(t) = \mathscr{L}^{-1}\frac{G_{vd}(s)}{s} = 1 - \frac{e^{-\delta\omega_n t}}{\sqrt{1-\delta^2}} sen(\omega t + \phi), \tag{7.1}$$

where $\omega := \omega_n\sqrt{1-\delta^2}$,$\sigma := -\delta\omega_n$ and $\phi := \arctan\frac{\sqrt{1-\delta^2}}{\delta^2}$. Both the damping factor δ and the natural frequency ω_n of the system influence the position of the poles $p_{1,2} = \sigma \pm j\omega$ in $G_{vd}(s)$:

$$G_{vd}(s) = \frac{1}{1 + 2\delta\frac{s}{\omega_n} + \frac{s^2}{{\omega_n}^2}}, \tag{7.2}$$

The natural frequency related to a second order LC filter is the resonant frequency $\omega_0 = \frac{1}{\sqrt{LC}}$. Then impulsive response and the step response in the frequency domain are strictly related (Eq.7.3) and information about buck converter load parameters can be obtained from the step response in the frequency domain.

$$Y(s) = \mathscr{L}\frac{G_{vd(s)}}{s}. \tag{7.3}$$

7.1.1 Resonant frequency identification results

The ADC output during the step response is recorded at the simulation clock frequency ($f_{clk} = 70MHz$), but the ADC clock is generated once per switching cycle from the DPWM. The error is updated at the switching frequency $f_s = 449kHz$, a downsampling factor of $f_{clk}/f_s = 156$ have to be considered before processing the recorded ADC output vector. Let's consider the system step response for three different buck converter configurations $f_{0,buck1} = 4.9kHz$ ($L = 47\mu H, C = 22\mu F$), $f_{0,buck2} = 7.3kHz$ ($L = 47\mu H, C = 10\mu F$) and $f_{0,buck3} = 10.7kHz$ ($L = 47\mu H, C = 4.7\mu F$), summarised in Fig:7.6. The PSD computation of the system step response for these configurations has been shown respectively in Fig.7.7, Fig.7.8 and Fig.7.9, the maximum of the PSD computation is very close to the resonant frequency. The PSD has been computed just after that the V_{ref} step occurs, the consequent error $e[n]$ has been

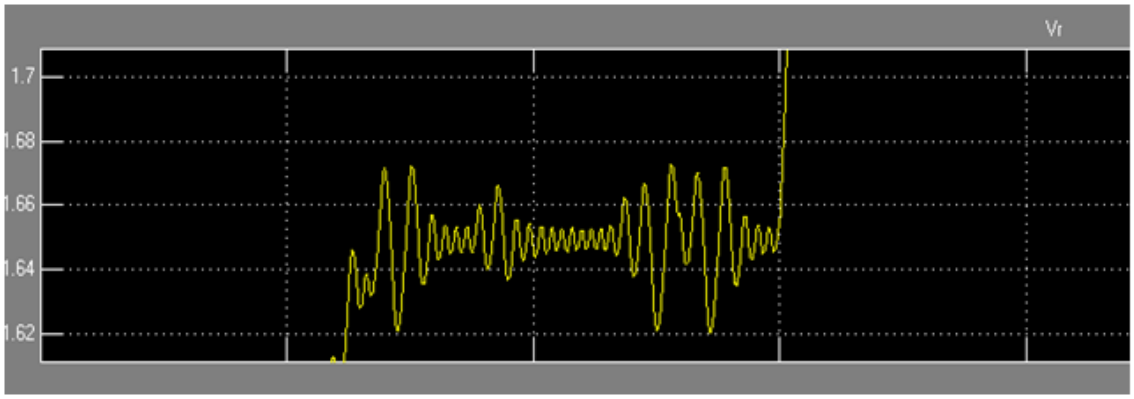

Figure 7.5: Open loop identification method: step evolving reference voltage (zoom).

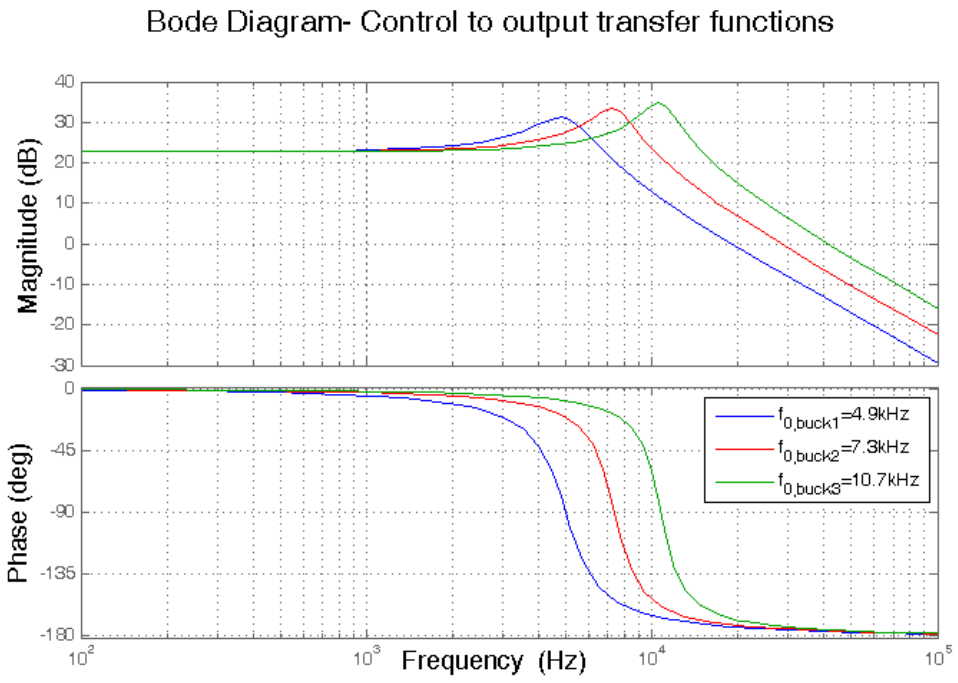

Figure 7.6: Buck converter configurations: load identification (no *ESR* contribution).

recorder for $NT_s = 0.285msec$ ($N = 128$ samples and $T_s = 1/449kHz$). Fixing both the number of points N considered to compute the PSD and the switching frequency f_s, the resolution Δf between frequencies which can be distinguished in the PSD is:

$$\Delta f = \frac{f_s}{2N} = 1.754kHz. \tag{7.4}$$

With a resolution $\Delta f = \frac{f_s}{2N} = 1.754kHz$ the resonant frequencies can be obtained as maximum of the PSD. As visible in Fig.7.7, Fig.7.8 and Fig.7.9, the obtained maximums are $f_{01} = 4.2kHz$, $f_{02} = 6.35kHz$ and $f_{03} = 10.6kHz$ respectively for configuration $f_{0,buck1}$, $f_{0,buck2}$ and $f_{0,buck3}$. These values are not far from the resonant frequencies of the buck converter configurations presented in Fig.7.6. Then a frequency response trend can be observed and a maximum value related to the resonant frequency can be detect in the PSD computation. For different load conditions, it results that the maximum extracted from PSD analysis is very close to the expected resonant frequency.

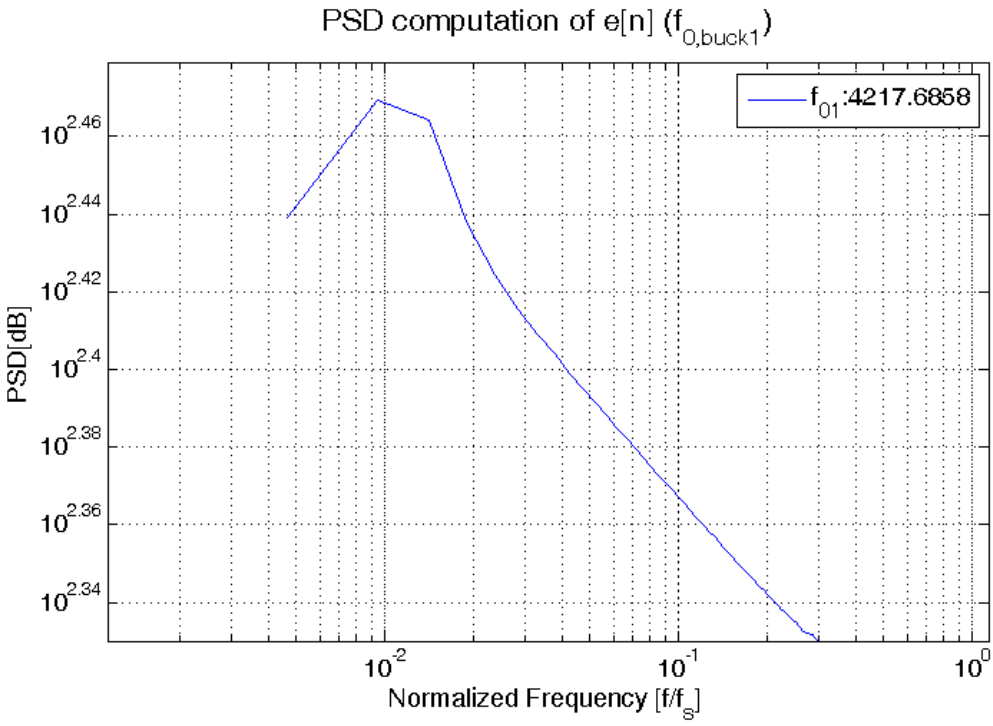

Figure 7.7: Open loop identification method result: $f_{0,buck1}$, $ESR = 0\Omega$.

7.1.2 ESR effects identification results

In the buck converter configurations shown in Fig.7.10, the ESR contribution is considered for three buck converter with respectively $f_{0,buck1} = 4.9kHz$ ($L = 47\mu H, C = 22\mu F$, $ESR = 0.5\Omega$), $f_{0,buck2} = 7.3kHz$ ($L = 47\mu H, C = 10\mu F$, $ESR = 1\Omega$) and $f_{0.buck3} = 10.7kHz$ ($L = 47\mu H, C = 4.7\mu F$, $ESR = 1\Omega$). The zero contribution ($f_z = 1/(2\pi ESR * C)$) has been introduce in $f_{z,buck1} = 14.5kHz$ for $f_{0,buck1}$, $f_{z,buck2} = 15.9$ for $f_{0,buck2}$ and $f_{z,buck3} = 33.8$ for

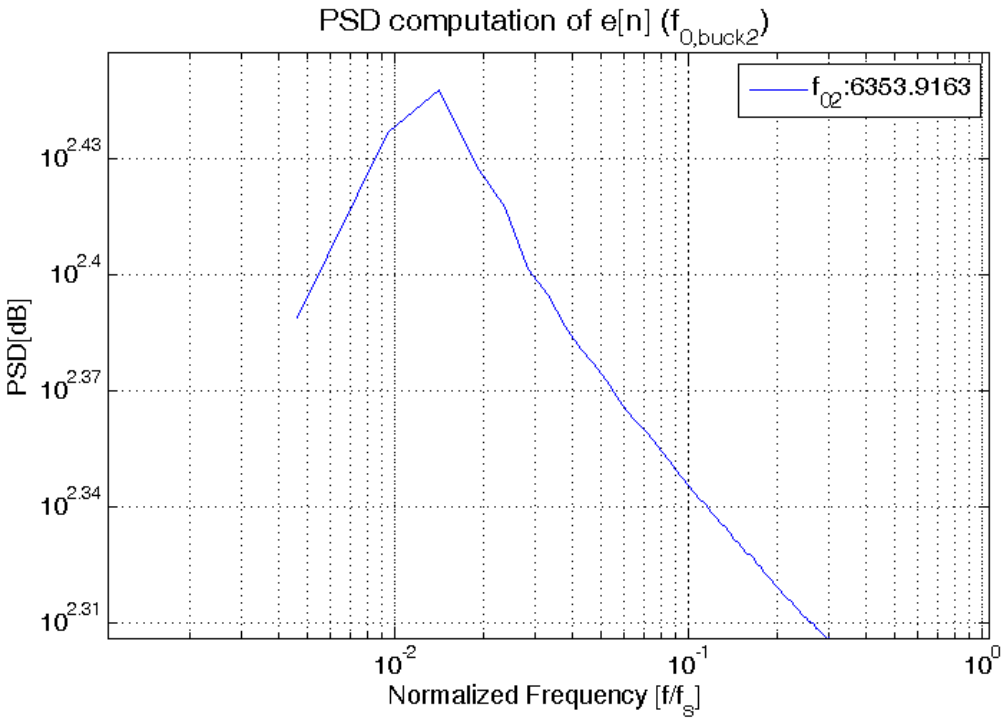

Figure 7.8: Open loop identification method result: $f_{0,buck2}$, $ESR = 0\Omega$.

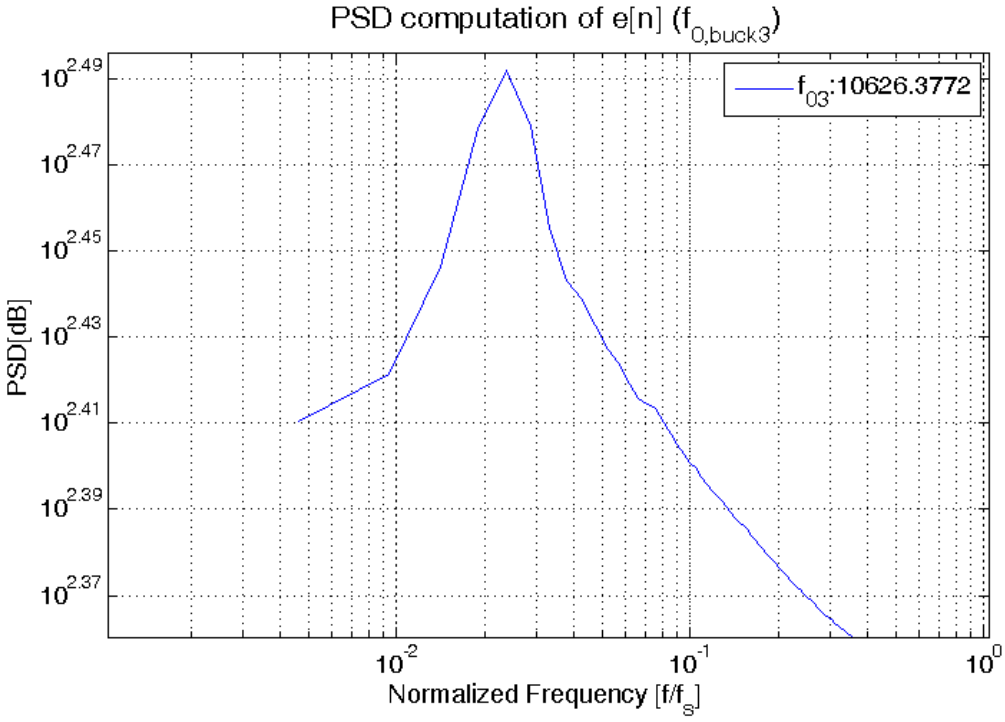

Figure 7.9: Open loop identification method result: $f_{0,buck3}$, $ESR = 0\Omega$.

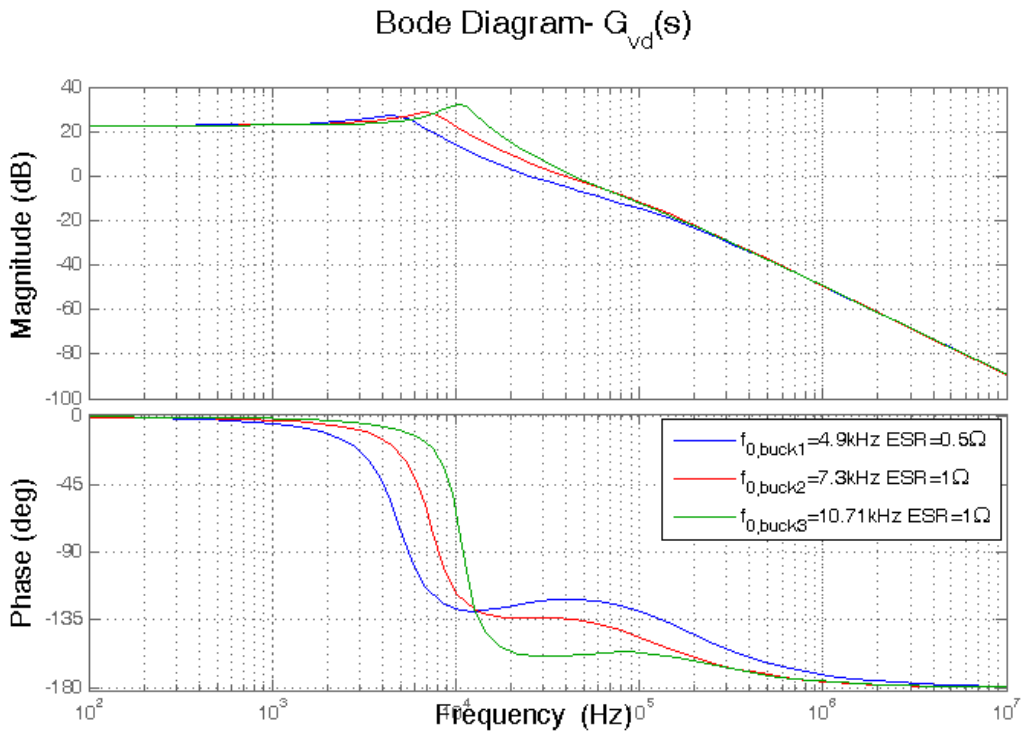

Figure 7.10: Buck converter configurations: load identification (ESR contribution).

$f_{0,buck3}$. In Fig.7.11 is shown the PSD computation when the ESR contribution is considered in the configuration $f_{0,buck1}$, the obtained resonant frequency can be still observed and the zero contribution can be detect at $f_{z1} = 0.03796 f_s = 17.1kHz$. However, is confirmed that the output filter structure can be observed through the PSD computation of the system step response. In the result shown in Fig.7.12, the ESR contribution for $f_{0,buck2}$ is at $f_{z2} = 0.02844 f_s = 12.8kHz$ and the obtained resonant frequency is $f_{02} = 8.49kHz$. In this case the obtained values are not far from the ideal values but a little error, due also to the finite resolution introduced by the processing, is introduced. The same analysis has been presented in Fig.7.13, and also in this case a non-dramatic overestimation is introduced on the obtained resonant frequency. The ESR contribution is difficult to observe, a change of slope can be detect at $f_{z3} = 0.05701 f_s = 25.6kHz$. In the configuration $f_{0,buck3}$ the ESR contribution introduces a zero far from the resonant frequency ($f_{z,buck3} = 33.8$) and, then, out from the bandwidth of the buck converter(Fig.7.10). For this reason its contribution is little in terms of magnitude and it is difficult to observe. Let's consider a bigger ESR contribution for the configuration $f_0, buck3$ ($L = 47\mu H, C = 4.7\mu F$, $ESR = 1\Omega$). For $ESR = 1.8\Omega$ the expected zero is at $f_{z,buck3} = 1/(2\pi 1.8 * 4.7\mu F) = 18.8kHz$. In Fig.7.14 is shown the PSD computation which refers to this configuration of buck converter. For a bigger contribution of ESR, a second peak can be easily detect at $f_{z3} = 0.04272 f_s = 19.2kHz$ that is very close to the expected zero position. Moreover, the resonant frequency information related to the maximum in the PSD is also in this case very precise. With the introduced open loop SI method, is then possible to obtain information about the buck converter configuration through the PSD computation of the system step response. The system step response approach, cannot

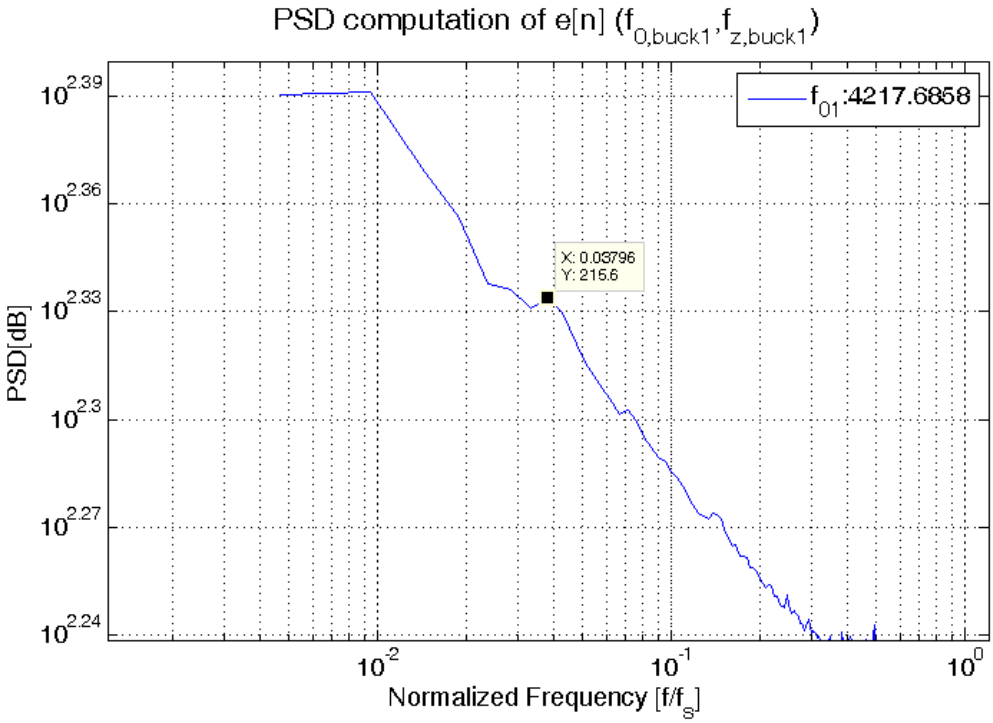

Figure 7.11: Open loop identification method result: $f_{0,buck1}$, $ESR = 0.5\Omega$.

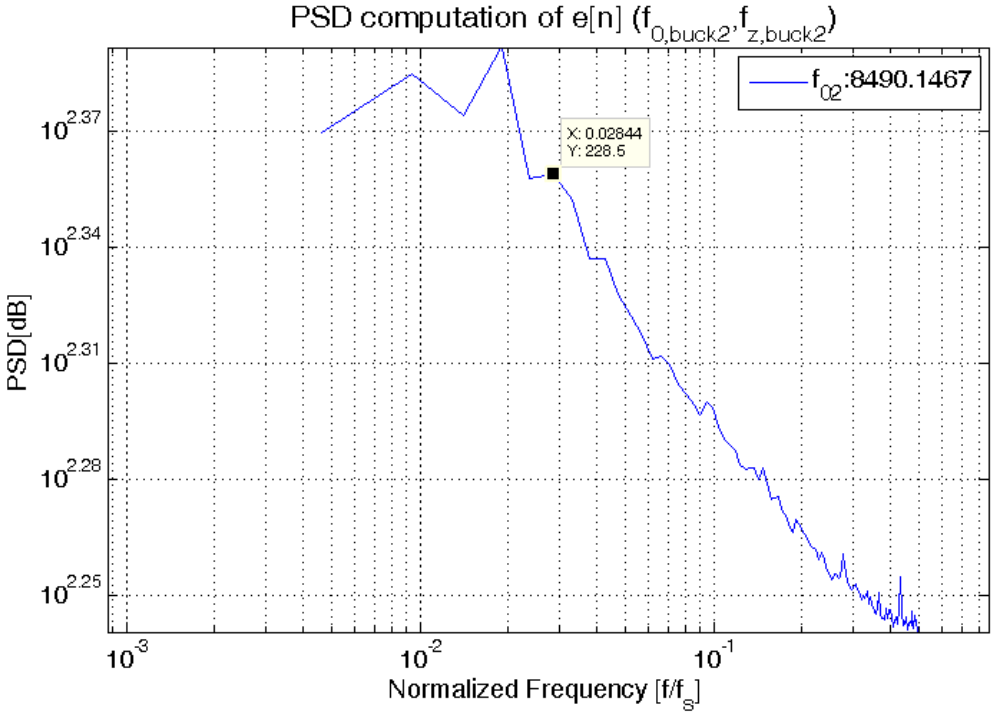

Figure 7.12: Open loop identification method result: $f_{0,buck2}$, $ESR = 1\Omega$.

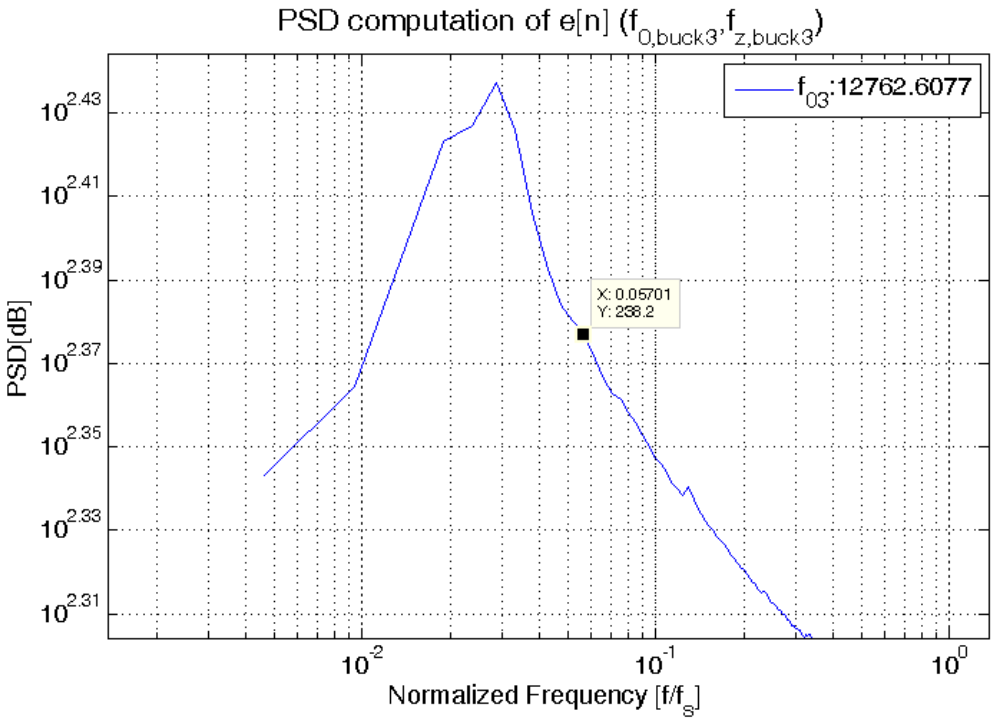

Figure 7.13: Open loop identification method result: $f_{0,buck3}$, $ESR = 1\Omega$.

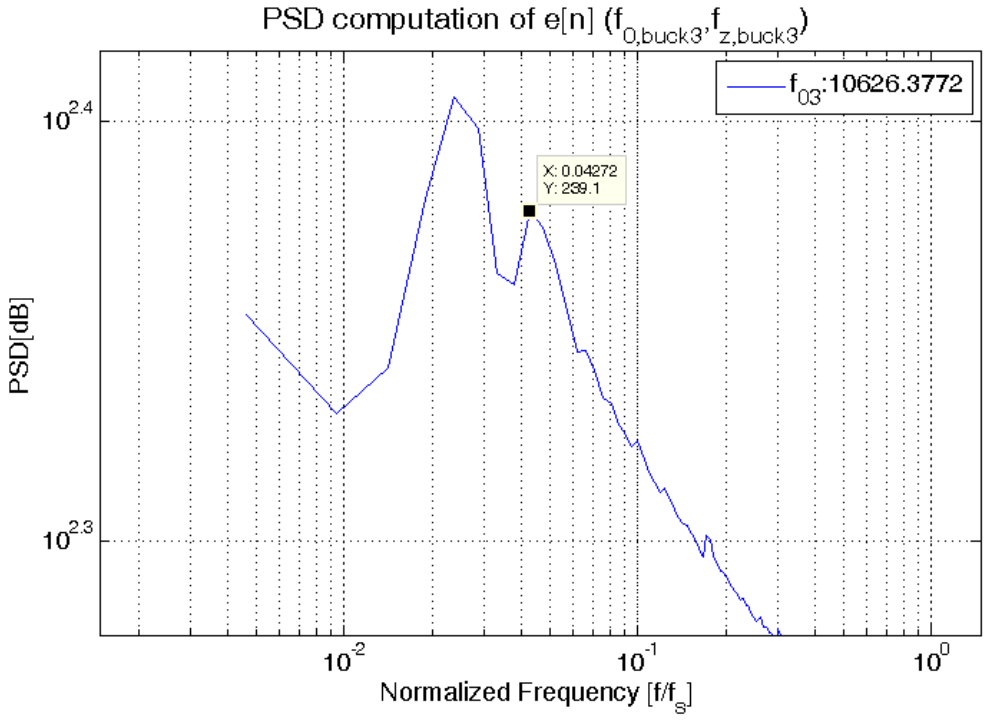

Figure 7.14: Open loop identification method result: $f_{0,buck3}$, $ESR = 1.8\Omega$.

be adopted to monitor load variations during the steady state voltage regulation, this kind of perturbation can be dangerous for loads supplied by the dcdc converter. Anyway it can be adopted during the start up of the system to compute the first PID configuration. From PSD analysis results that, this kind of approach is able to discern close resonant frequencies for different buck converter configurations and, moreover, it can be distinguished also when ESR contribution is present.

7.2 Steady state system identification technique

Steady state non-parametric identification methods are designed to be exploited by the closed loop configuration (Fig.7.2), with the aim to identify the buck convert in terms of resonant frequency (f_0) and ESR contribution during the steady state phase. Let's consider the closed loop Matlab/Simulink fixed point model shown in Fig.7.15 and the $\Delta\Sigma$ converter shown in Fig.7.16.

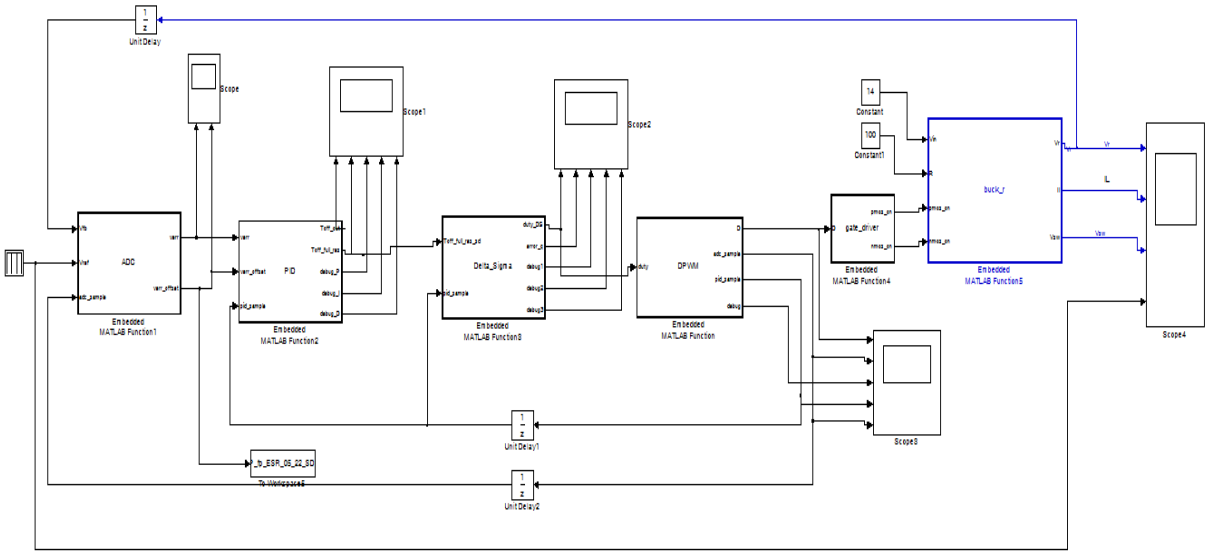

Figure 7.15: Digitally controlled buck converter: Matlab/Simulink fixed-point model.

The $\Delta\Sigma$ modulator has been introduced in Sec.3.4, where the DPWM resolution increasing demonstrated in[76] has been justified. During the steady state quantization effects are introduced by the DPWM. If this quantization noise is not added on the signal path (no dithering), sinusoidal variation at a certain frequency f are obtained for the low resolution duty cycle. Introducing a $\Delta\Sigma$ modulator the quantization noise contribution inside the LSB_{DPWM} is considered (dithering) and, due to the oversampling ($f_s >> f$), it results distributed in the range of frequencies up to the switching frequencies. In this way the duty cycle change faster with different frequency contributes.
Every switching step the quantization noise is high pass filtered and added to the high resolution digital control word. It is an independent additive white noise uniformly distributed inside $[\pm\frac{LSB_{DPWM}}{2}]$. Then, modelling of dithering contribution can be done with a further input for the system the system (Fig.7.17). Considering this model, dithering effects on $G_{vd}(s)$

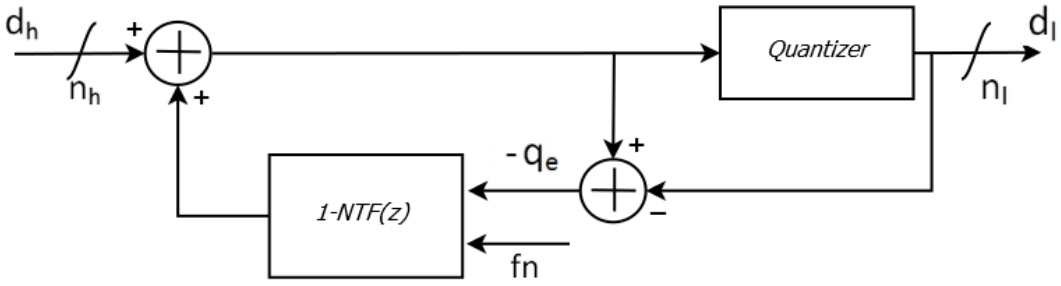

Figure 7.16: Steady state identification method: ΔΣ error feedback configuration.

can be observed at the ADC output by observing the product $G_{vd}(s)NTF(z)$. Let's consider

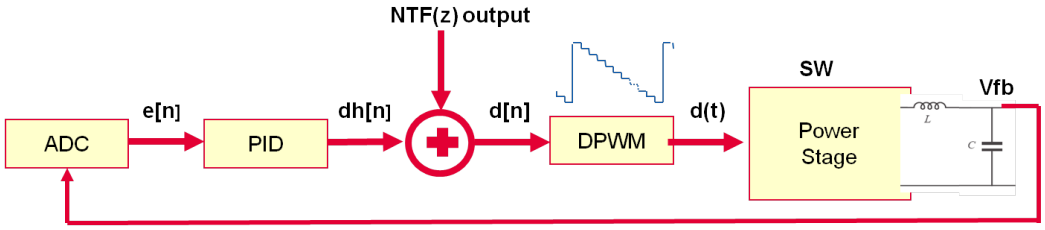

Figure 7.17: Steady state identification method: dithering modelling.

the three different buck converter configurations introduced in Fig.7.6. For these cases, the product between the control to output transfer function and the third order $NTF_3(z)$ has been considered in Fig.7.18. The output filter resonant frequency con be observed for all considered converter configurations. This confirms that output filter informations can be observed at the ADC output when the dithering effect is present.

7.2.1 Resonant frequency and ESR identification algorithm

Let's consider the factor $\alpha \in \mathbb{N}$ as the amplification factor ($\alpha > 1$) of the quantization noise in input to the $NTF(z)$. It can be introduced in the ΔΣ:

$$d'_{lr}[n] = d_{hr}[n] + NTF(z) * \alpha q_e. \tag{7.5}$$

In Fig.7.19 has been shown the effect on $G_{vd}(s)NTF_3(z)$ for $f_{0,buck1} = 4.9kHz$, when dithering contributions are amplified. For $\alpha > 1$, the magnitude of $G_{vd}(s)NTF_3(z)$ at the resonant frequency is amplified, then further amount of white noise is injected on the signal path over the entire range of frequencies.
The amplification of quantization noise can be modelled as a lowering of the DPWM resolution, it has always the same range of frequencies $[0 : f_s]$ but, is uniformly distributed inside a bigger LSB_{DPWM}. The ΔΣ reduces its resolution of a factor α to amplify the quantization error, hence, the DPWM resolution is forced to be $N'_{DPWM} = \frac{V_{in}}{2^{n_{DPWM}}/\alpha} = \frac{V_{in}}{2^{n'_{DPWM}}}$. The resonant frequency identification idea is related to the dithering amplification:

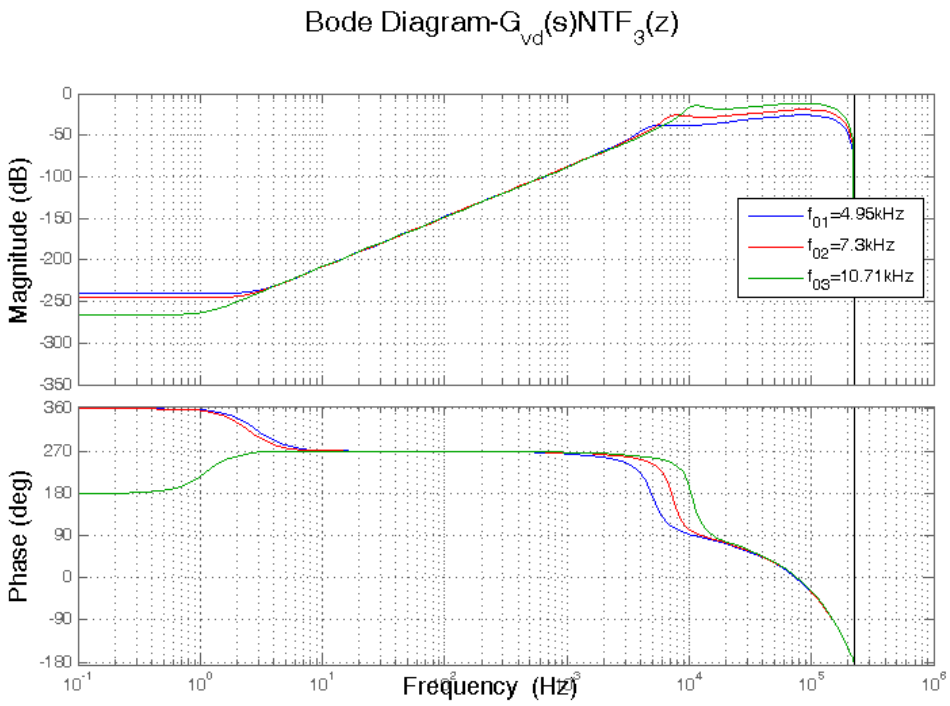

Figure 7.18: Steady state identification method: dithering effects on the control to output transfer function (no ESR contribution).

- When the steady state identification is on, the factor α is more than 1 and a noisy low resolution control word $d'_{lr}[n]$ is output from the DPWM. In this way a bigger amount of noise is injected on the signal path in the range $[0 : f_s]$ of frequencies. Duty cycle fast variations, due to the dithering, are amplified with α increasing and information about the resonant frequency can be observed inside the ADC zero error bin $(G_{vd}(s)NTF_3(z))$.

- When the steady state identification is off, the factor α is 1 and the normal $\Delta\Sigma$ implementation is obtained:

$$d_{lr}[n] = d_{hr}[n] + NTF(z) * q_e. \tag{7.6}$$

  In this case the same hardware used during the identification can be used to increase the control loop resolution[76].

For a DPWM down-counter $d'_{lr}[n]$ is the off-time, hence the duty cycle becomes function of α:

$$d(\alpha q_e)[n] = \frac{T_{on}}{T_{on} + T_{off}} = \frac{2^{n'_{DPWM}} - d'_{lr}[n]}{T_{sw}}, \tag{7.7}$$

during the steady state $V_{out} = d(t)V_{in}$ and any perturbation is reflected on the ADC zero error bin through $d'_{lr}[n]$.

In order to observe all the components of $G_{vd}(s)NTF_3(z)$ and the resonant frequency, the factor α have to amplify the noise effects on the output voltage in the range of interested

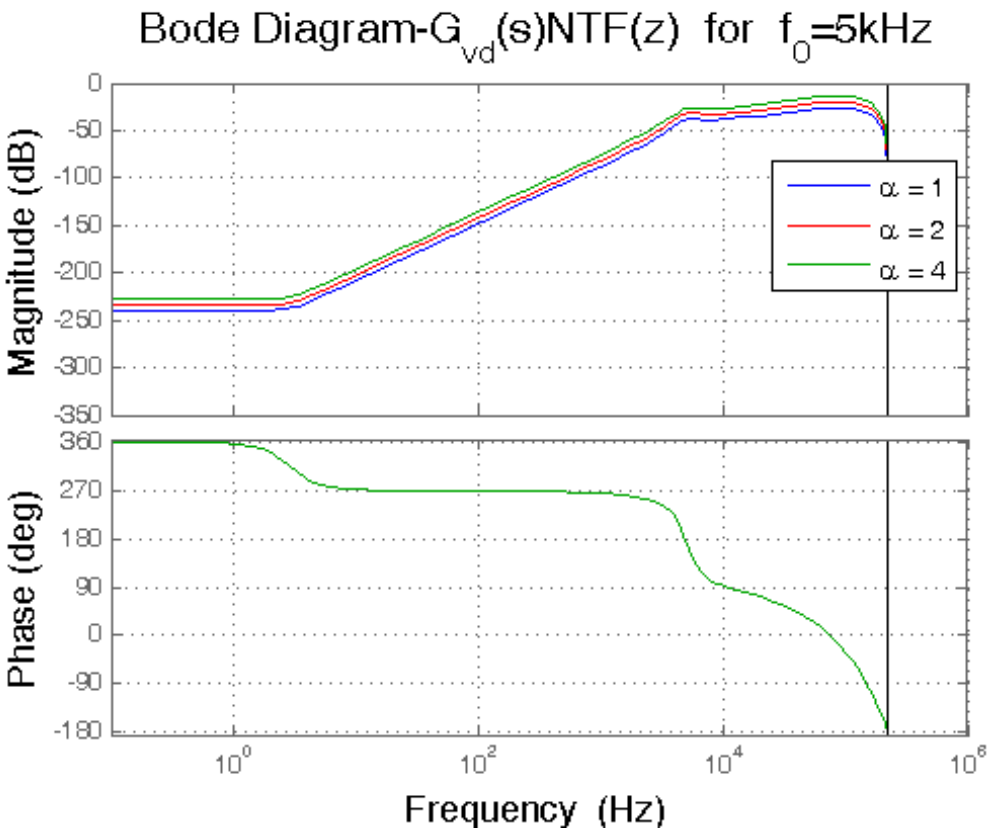

Figure 7.19: Steady state identification method: dithering amplification effects on the control to output transfer function (no ESR contribution, $f_{0,buck1}$).

frequencies. If information at a specified frequency have to be detected, the noise inserted have to be bigger then the ADC zero error bin:

$$\Delta v_{out}(j\omega) > LSB_{ADC}. \tag{7.8}$$

Let's define $\Delta d(f)$ as the perturbation $\alpha q_e(f)$ in the frequency domain. Considering the dithering effect as an additional input in the signal path (Fig.7.17), the expression of Eq.7.8 in function of $\delta d(f)$ is:

$$\Delta v_{out}(s) = \Delta d(s)\,|DPWM|\,|G_{vd}(s)|, \tag{7.9}$$

where $|DPWM| = 1/2^{n_{DPWM}}$. To detected the resonant frequency, the Eq.7.9 have to be satisfied for $s = j\omega_0$:

$$\Delta v_{out}(j\omega)\big|_{j\omega=j\omega_0} = \Delta d(j\omega)\big|_{j\omega=j\omega_0} \frac{1}{2^{n_{DPWM}}} \big|G_{vd}(j\omega)\big|_{j\omega=j\omega_0} > LSB_{ADC}. \tag{7.10}$$

For a buck converter, the control to output magnitude at the resonant frequency is greater than the input voltage V_{in} (depending on the damping factor), considering $\big|G_{vd}(j\omega_0)\big| \approx V_{in}$ approximates the magnitude of the output voltage oscillations:

$$\Delta v_{out}(f)\big|_{f=f_0} \approx \Delta d(f)\big|_{f=f_0} \frac{V_{in}}{2^{n_{DPWM}}} > LSB_{ADC}. \tag{7.11}$$

In order to detect output voltage variations due to the noise injection, the factor $\Delta d(f)$ could be used to amplify the injected quantization noise. Considering for instance $2^{n_{DPWM}} = 100$ and $V_{in} = 10V$:

$$\Delta v_{out}(f)\big|_{f=f_0} \approx \Delta d(f)\big|_{f=f_0} \frac{10}{100} = \Delta f 100mV = \alpha q_e(f) 100mV. \tag{7.12}$$

The quantization noise $q_e(f)$ is an additive white noise present in the entire range of frequencies. For instance, observing the $|NTF_3(z)|$ (Fig.7.20) in the range of frequency of f_0 and f_z, a shaped quantization error of about $q_e = -40dB$ can be considered. Then the noise injected in the range of interested frequencies in the analysed case is:

$$\Delta v_{out}(f) \approx \alpha 10^{-2} 100mV = \alpha mV. \tag{7.13}$$

Considering the Eq.7.13, the perturbation on the output voltage is then strictly related to the noise amplification factor α. This assumption has been done assuming that $\left|G_{vd}(j\omega)\right|_{j\omega=j\omega_0} = V_{in}$. Considering buck converters control to output transfer functions (Fig.7.6), the magnitude at the resonant frequency is always more then V_{in} because the damping factor of complex conjugated poles. Let's consider a worst case condition of $20dB$ more than the input voltage:

$$\Delta v_{out} \approx \alpha 10mV, \tag{7.14}$$

an amplification noise factor of $\alpha = 2$ can introduce tens of millivolts on the output voltage (Eq.7.14). Referring to the model introduced in Cha.4, a noise amplification factor $\alpha = 2$ could be considered enough to detect the perturbations response with $LSB_{ADC} = 15mV$.

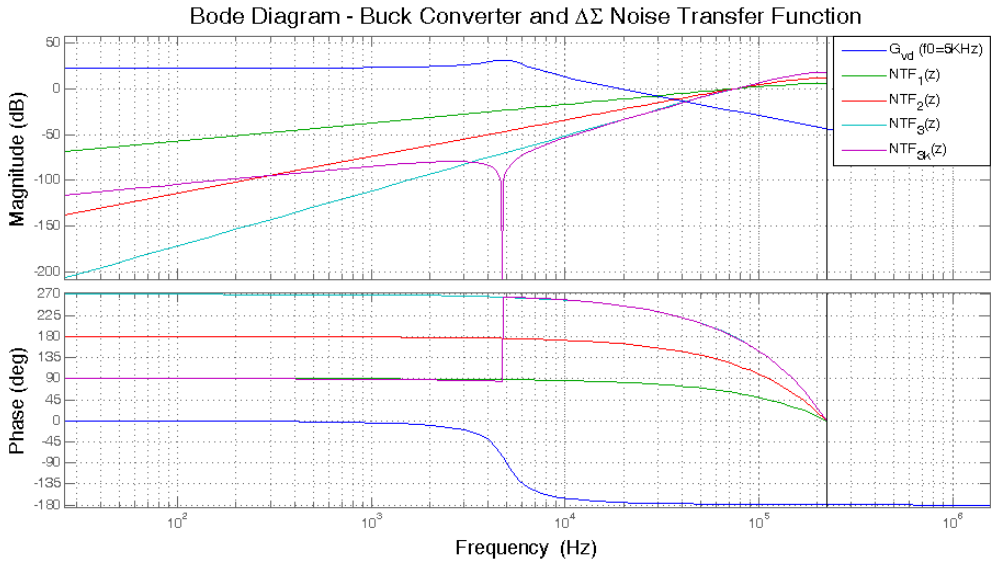

Figure 7.20: Control to output transfer function and noise shaper configurations.

Further consideration have to be done in terms of the resolution in the frequency range. Let's consider the ΔΣ modulator shown in Fig.7.16, the noise shaper process $n_h - n$ bits related to the quantization error. The minimum value of this resolution, defines the minimum

frequency f which is stimulated through the noise:

$$f = \frac{f_s}{2^{n_h - n} - 1}, \tag{7.15}$$

where f_s is the system switching frequency. In case of load identification, the minimum frequency of interest is related to the resonant frequency f_0. Considering as $f = 1kHz$ the minimum frequency to stimulate and $f_s = 449kHz$, a minimum resolution of $n_h - n = 9$ bits is needed.

The Power Spectral Density (PSD) of $G_{vd}(s)NTF_3(z)$ can be analysed in order to chose α. As discussed, a further amount of noise can be injected to detected the output filter frequency inside the LSB_{ADC} and is important to understand how this amount of noise is distributed over the range of frequencies. The quantization noise is a white noise and is constant over the entire range of frequencies, while, when a noise shaper is used the power of the noise is more concentrated at medium high frequencies. Considering α variations in the PSD of $G_{vd}(s)NTF_3(z)$, is important to understand both how the further amount of noise is shaped in frequency and if, enough noise is concentrated around the resonant frequency. In 7.22 α variations has been considered for the PSD of $G_{vd}(s)NTF_3(z)$. It can be deduced that, the noise is observable in the range of interested frequencies and. Comparing $\alpha = 1$ with $\alpha = 2$ is visible how more power is concentrated around the resonant frequency, the noise amplification is reflected on the overall range of interested frequencies and its effects can be detected inside the LSB_{ADC} during the steady state. The output variations can be observed on the ADC output error and processed to extract the output filter frequency information, the resolution of the q_e have to ensure that output filter range frequencies for a fixed value of switching frequency are stimulated. In Fig.7.21 is shown the dithering effects on the output voltage, when $\alpha = 2$ the noise amplification is reflected on the magnitude of the oscillations on the output voltage and steady state behaviour persists. In this example a switching frequency $f_s = 449kHz$ is considered for $f_{clk} = 70MHz$ and $V_{ref} = 3.3V$, the buck converter is configured with $f_{0,buck1} = 4.9kHz$.

As deduced up to now, a further amount of shaped noise can permits to observe output filter information at the ADC output. The discussion has been addressed focusing on the resonant frequency f_0, some considerations can be done to the ESR identification. In the buck converter configurations shown in Fig.7.10, the ESR contribution is considered for three buck converter with respectively $f_{0,buck1} = 4.9kHz$ ($L = 47\mu H, C = 22\mu F$, $ESR = 0.5\Omega$), $f_{0,buck2} = 7.3kHz$ ($L = 47\mu H, C = 10\mu F$, $ESR = 1\Omega$) and $f_{0,buck3} = 10.7kHz$ ($L = 47\mu H, C = 4.7\mu F$, $ESR = 1\Omega$). The zero contribution has been introduce in $f_{z,buck1} = 14.5kHz$ for $f_{0,buck1}$, $f_{z,buck2} = 15.9$ for $f_{0,buck2}$ and $f_{z,buck3} = 33.8$ for $f_{0,buck3}$. In Fig.7.23 has been considered $\alpha = 2$ in $G_{vd}(s)NTF_3(z)$, for the three cases presented in Fig.7.10. Information about the buck convert output filter can be observed in the frequency domain, even if a large ESR in the output capacitor. Moreover, both the resonant frequency and the change of slope due to the zero presence can be distinguished. The identification idea is to amplify the white noise overall the entire range of frequencies, in order to detect both the resonant frequency and the zero presence at the ADC output. Results shown in Fig.7.23 are useful only to observe the information related to the resonant frequency, the ESR effects can be very difficult to observe. To highlight the ESR contribution respect to the resonant frequency, could be helpful to use the modify third order NTF introduced introduce in Cha.3.4.2. The third order

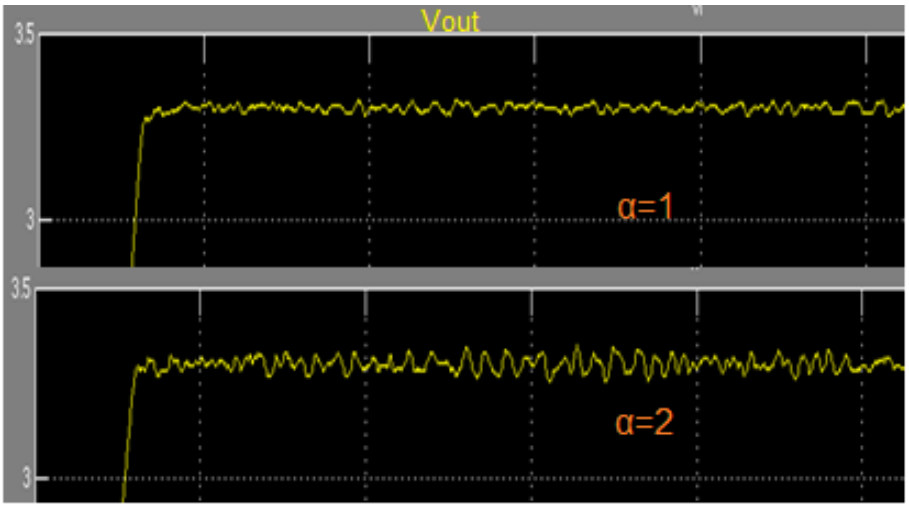

Figure 7.21: Steady state identification method: dithering amplification effects on the steady state output voltage.

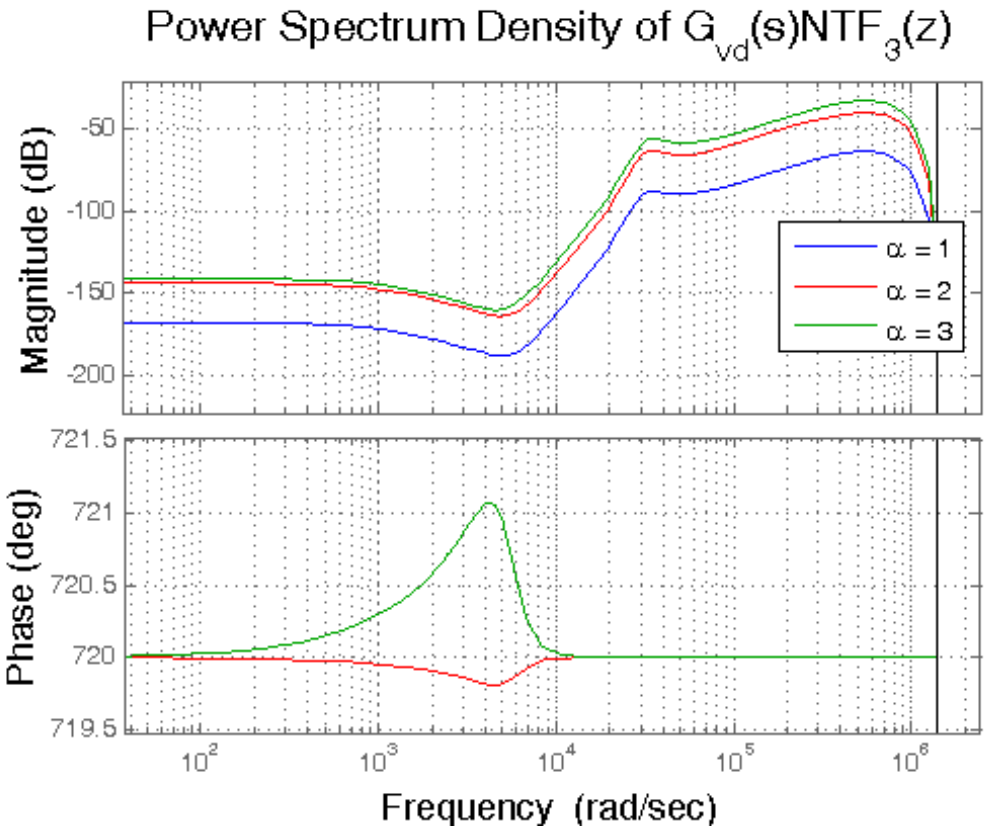

Figure 7.22: Steady state identification method: PSD of dithering amplification effects on the control to output transfer function (no *ESR* contribution).

modified NTF with a zero at the resonant frequency is:

$$NTF_{3k}(z) = (1 - z^{-1})(1 - kz^{-1} + z^{-2}), \tag{7.16}$$

where $K = 2cos(2\pi\frac{f_n}{f_s})$ permits to insert a notch at $f_n = f0$. In Fig.7.24 is shown the $G_{vd}(s)NTF_{3k}(z)$ with $\alpha = 2$ for the three buck converter configurations presented in Fig.7.10. The zero contribution at the resonant frequency permits to inject less noise at the resonant frequency, in this way the peak of $G_{vd}(s)NTF_{3k}(z)$ related to the resonant frequency can be attenuated and, on the counter part, the ESR contribution is highlighted.

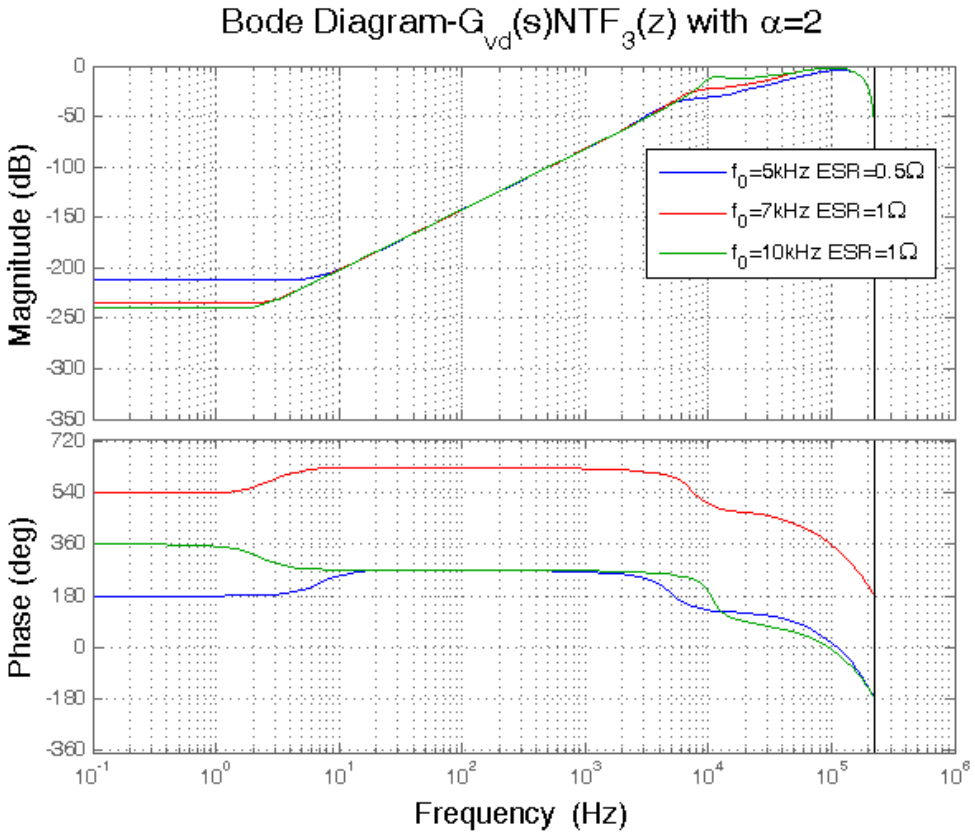

Figure 7.23: Steady state identification method: dithering amplification effects on the control to output transfer function (ESR contribution).

The noise amplification on the overall range of frequencies is reflected on the LSB_{ADC} and can ensures that oscillation can be detected on the digital side. This effect, can be seen as a resolution improvement for the load information detection. The further amount of noise is shaped at medium high frequencies and from PSD analysis results that the noise contribution is concentrated on the range of interested frequencies. Furthermore, to distinguish resonant frequency information from the zero contribution due to the ESR, a modified NTF can be used to shape the noise. The zero in the NTF can be placed close to the resonant frequency to attenuate the noise injection at this frequencies.
The mathematical equations of the described approach are mainly two. Firstly, every unitary amplification of the quantization noise add a perturbation $\Delta v(f)$ on the output voltage.

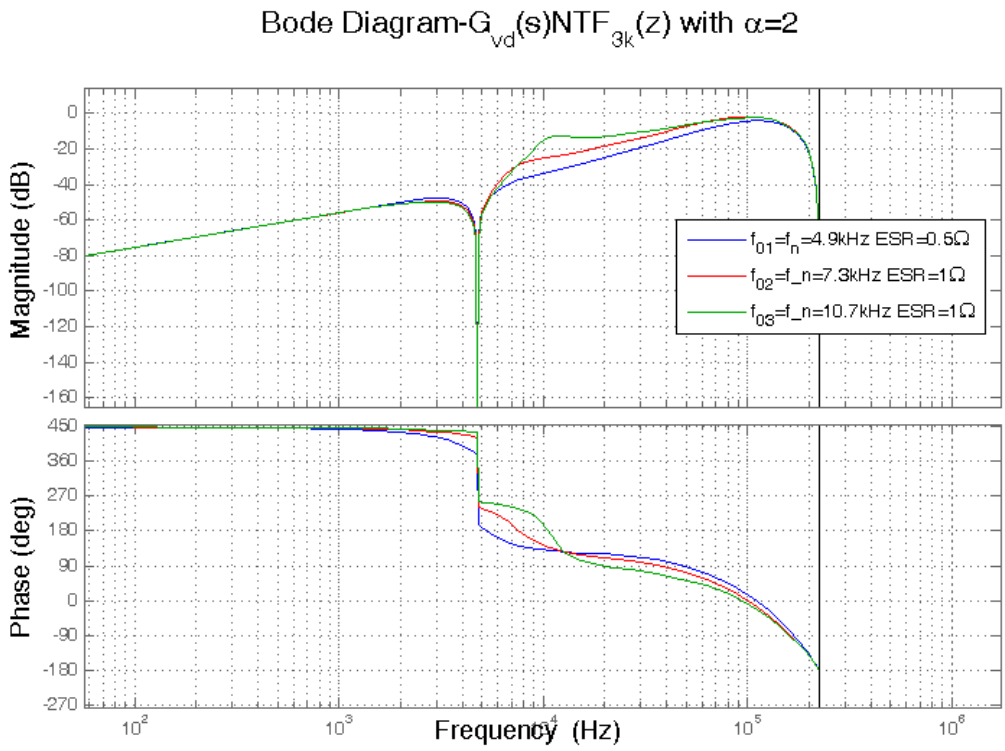

Figure 7.24: Steady state identification method: dithering amplification effects on the control to output transfer function (ESR contribution and notch effects).

Moreover, the noise increasing can be seen as DPWM resolution decreasing of the same factor α. The effective DPWM resolution N'_{DPWM}during the noise injection, has to be however bigger than the ADC to avoid LCOs [86]. Choosing the factor α the lost of resolution have to be taken into account as well.

Mathematical relationships for the steady state identification model can be summarised as follow:

- $d'_{lr}[n] = d_{hr}[n] + NTF(z) * \alpha q_e$ where $\alpha > 1$. A further amount of quantization noise can be added through the dithering amplification. high frequency variations are obtained in the digital control world, thanks to dithering and oversampling. Considering $G_{vd}(s)NTF(z)$, information about the buck converter resonant frequency f_0 can be distinguished, while the ESR contribution is exalted by considering a notch in the noise shaper.

- $f = \frac{f_s}{2^{n_h - n} - 1}$. The minimum quantization error resolution has to be calculate with respect to the switching frequency and the minimum stimulated frequency f. Computing the resolution in this way, permits to be sure that the noise can have contributions into the desired range of converters configurations.

- $\Delta v_{out}(j\omega) = \Delta d(j\omega) \frac{1}{2^{n_{DPWM}}} \left|G_{vd}(j\omega)\right| > LSB_{ADC}$. The noise amplification factor α has to be computed considering the magnitude of the output voltage perturbation. Approximations can be done to consider $\left|G_{vd}(j\omega)\right|_{\omega_0}$. A factor $\alpha = 2$ can be considered

enough with $f_s = 449kHz$, $V_{in} = 10V$ and $LSB_{adc} = 15mV$. This consideration permits to be sure that, dithering amplification effects can be detect at the ADC output.

- $N'_{DPWM} = \frac{V_{in}}{2^{n_{DPWM}/\alpha}} = \frac{V_{in}}{2^{n'_{DPWM}}}$. The factor α automatically reduces the DPWM resolution.
- $2^{n'_{DPWM}} > 2^{n_{adc}}$. To avoid LCOs during steady state the static condition introduce in Sec.3.3.3 have to be respected. This represent the upper limit to chose α.

In the next two sections, results are presented for a defined case of study having the same characteristics of systems presented in Cha.4. In these configurations, typical automotive parameters has been considered: $f_{clk} = 70MHz$ and $f_s = 449kHz$ defines a $n_{DPWM} = 8$ bits DPWM down-counter to count from 0 to $f_{clk}/f_s = 156$. The buck converter configurations are the same of Fig.7.6 ($f_{0,buck1}, f_{0,buck2}, f_{0,buck3}$) and Fig.7.10 where the ESR contributions ($f_{z,buck1}, f_{z,buck2}, f_{z,buck3}$) have been considered. The DPWM resolution is reduced to $n'_{DPWM} = 7$ bits (when $\alpha = 2$) and LCOs are avoided considering a 4 bits ADC with $LSB_{adc} = 15mV$. Moreover, results are presented comparing $\alpha = 1$, $\alpha = 2$ and $\alpha = 3$ which still avoid LCOs. In order to evaluate the $G_{vd}(s)NTF(z)$ results are presented considering the noise injection effects at ADC output, the error $e[n]$ is then processed in terms of PSD computation. The PSD computation has been preferred to simply FFT-based analysis both to exalt frequency information and simplify the detection of both f_0 and ESR load parameters.
The considered closed loop model has been shown in Fig.7.2, where the $\Delta\Sigma$ model refers to the one in Fig.7.16. A third order noise shaper has been considered taking into account observation drawn during analysis in the previous section. The entire closed loop implementation refers to models described in Cha.4, where the design has been described up to the VHDL implementation with Matlab-VHDL co-simulation of the digital control loop.
The output voltage behaviour due to the amplification of dithering effects for $f_{0,buck1} = 5kHz$ ($ESR = 0\Omega$) has been shown in Fig.7.25, Fig.7.26, Fig.7.27 respectively for $\alpha = 1$, $\alpha = 2$ and $\alpha = 3$. With the increasing value of α, the quantization noise is reflected on the output voltage. This amplification does not bring the system to instability even if $\alpha = 3$ is considered, the identification technique . In Fig.7.28 is shown the output voltage considering the difference between $\alpha = 2$ and $\alpha = 1$. Doubling the quantization noise the maximum magnitude of the noise amplification is of $150mV$.

The PID configuration during the steady state identification method is set with a priori consideration on the buck converter range of loads. Because the algorithm has to work both during the first load identification (when the output filter is unknown) and for monitoring load variations, the PID configuration is static and set in order to have a system stable in the considered range of possible loads that can be identified. The range of loads is automatically chosen by fixing the switching frequency, the resonant frequency of the output filter have to be lower than $f_s/20$ because ripple reasons on the output voltage. Once that the load is identify or monitored, the self-tuning algorithm have to change the generic PID into the one optimized (in terms of margins and bandwidth) for the obtained output filter. Results presented in the next subsections, are obtained with an open loop system bandwidth ($PID(z)G_{vd}(s)$) lower than the range of considered buck converters (Fig.7.6 and Fig.7.10) and positive phase margin around 10Âř. This kind of PID configuration can be called *slow*, indeed if a load step occurs the time to recover the steady state dc value can be longer

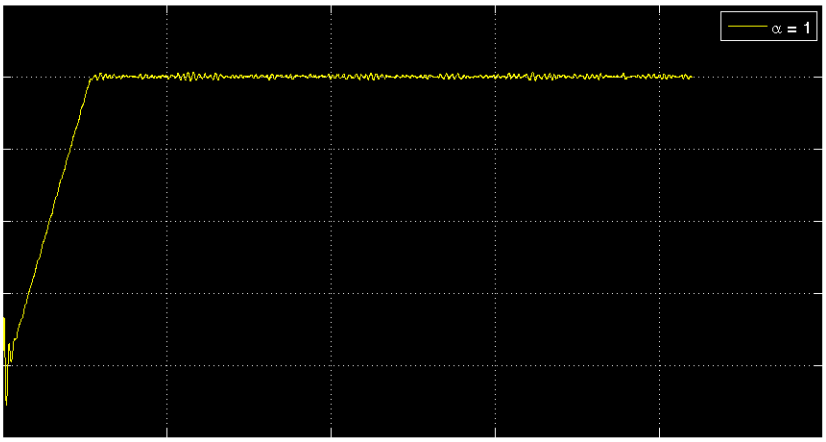

Figure 7.25: Steady state identification method: dithering effects on the output voltage.

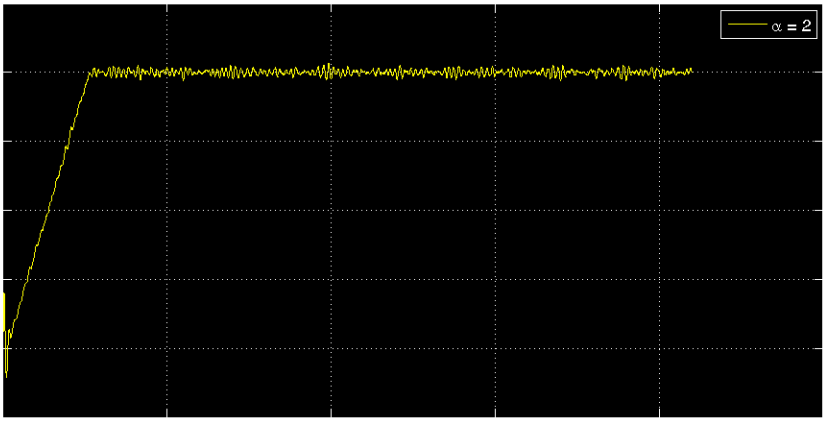

Figure 7.26: Steady state identification method: dithering amplification effects on the output voltage ($\alpha = 2$).

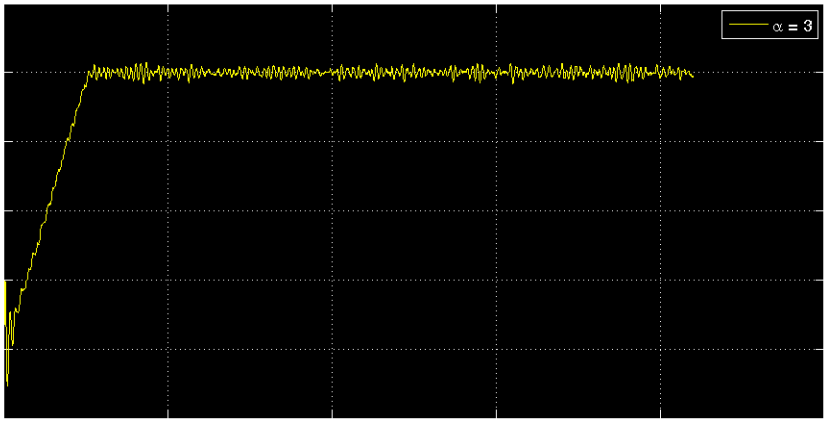

Figure 7.27: Steady state identification method: dithering amplification effects on the output voltage ($\alpha = 3$).

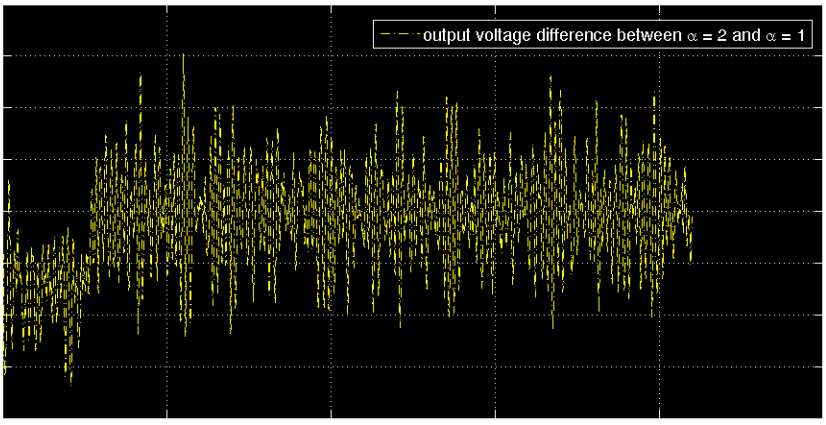

Figure 7.28: Steady state identification method: dithering amplification effects on the output voltage (comparison).

To extract the buck converter information from the $NTF_3(z)G_{vd}(s)$, the ADC output error is processed with the PSD computation. The Power Spectral Density computation is an autocorrelation-based algorithm (Sec.9.4.1) and the autocorrelation is often used with filtering purposes to remove huge amounts of noise. The difference between the FFT and the PSD computation of the ADC output error $e[n]$, can be observed in Fig.7.29 for an ADC output window recorded during the steady for $\alpha = 1$ and $f_{0,buck1}$. The PSD computation output, is more clean than the FFT and it is easier to detect information about the output filter. However, the shape of both considered processing related to $e[n]$ can be attributed to an impulsive response shape and, moreover, the maximum of the PSD is at $f_{01} = 2.65 * 10^4/2\pi = 4.22kHz$ which is very close to $f_{0,buck1}$ (Fig.7.6). All processing results of this section are ob-

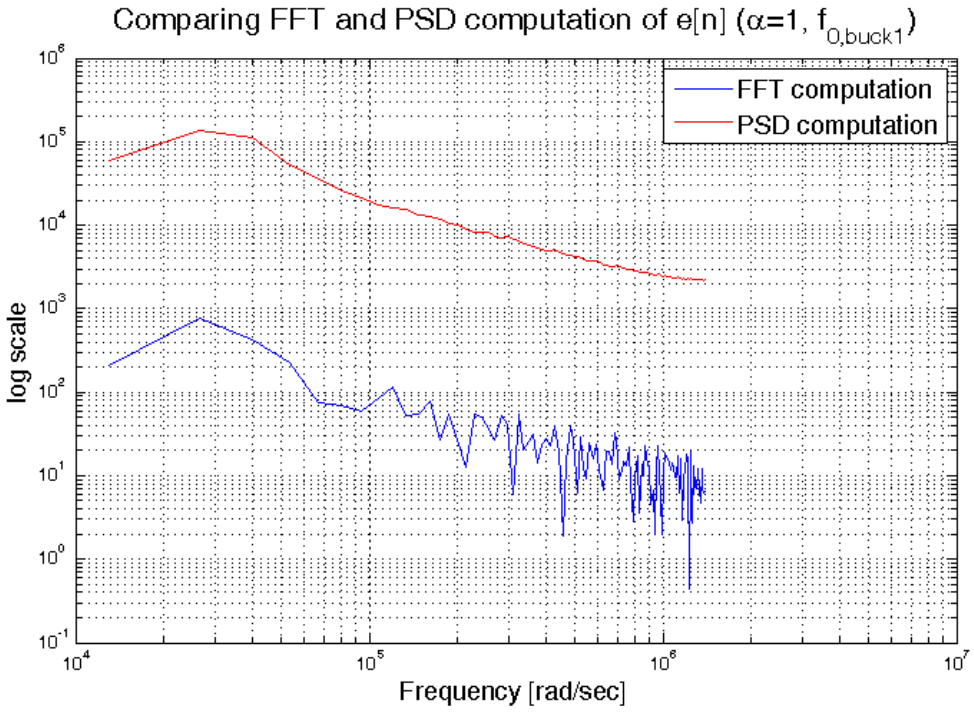

Figure 7.29: Steady state identification method result: FFT and PSD comparison ($f_{0,buck1}$, $\alpha = 1$).

tained considering $N = 2^7 = 128$ samples of $e[n]$, recorded at the switching frequency during the steady state. From the practical implementation point of view, the error is updated every switching frequency. On the counterpart, a finite resolution in the processing results is introduced and frequency information about the output filter are limited to the resolution Δf(Eq.7.4).

7.2.2 Resonant frequency identification results

The resonant frequency identification for the steady state SI method, is related to the quantization noise amplification α. To limit the magnitude on the oscillations on the output voltage the dithering effect is doubled (Fig.7.26) and tripled (Fig.7.27). From previous section, results that for the chosen system configuration parameters this amplification values can

be used for the steady state SI. Moreover, the information about f_0 have been obtained as maximum in the PSD of the ADC output error (Fig.7.29). Let's consider the α effects on configurations $f_{0,buck1} = 4.9kHz$, $f_{0,buck2} = 7.3kHz$ and $f_{0,buck1} = 10.7kHz$ (Fig.7.6). In Fig.7.30 is shown the PSD computed on e[n] for the α factor increasing, in this case the resonant frequency obtained as maximum of the PSD is $f_{01} = 4.22kHz$. However, can be observed that increasing the quantization noise the shape of the PSD is more clear, for $\alpha = 2$ and $\alpha = 3$ the sharp maximum can be distinguished while with $\alpha = 1$ information around the resonant frequency are less selective. Considerations about amplification dithering effects related to

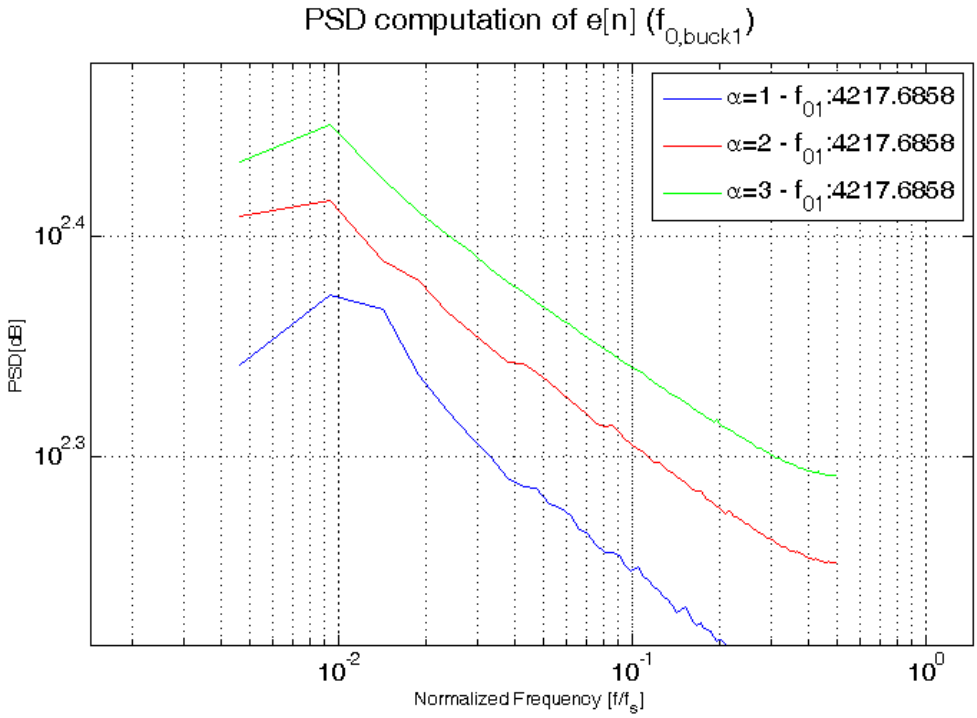

Figure 7.30: Steady state identification method PSD results: $f_{0,buck1}$, $ESR = 0\Omega$.

the resolution increasing on the resonant identification, are presented for $f_{0,buck2}$ in Fig.7.31. Increasing the resonant frequency makes necessary a further amount of noise to observe at the ADC output a maximum related to f_0. With $\alpha = 1$ two peaks can be distinguished in the range of the resonant frequency, then uncertainty on the resonant frequency identification is introduced. Increasing the dithering effect with $\alpha = 2$ a resonant frequency $f_{02} = 6.35kHz$ can be obtained as maximum on the PSD. On the counterpart, with $\alpha = 3$ the injected noise can move the obtained resonant frequency up to $8.5kHz$, more precision is obtained with $\alpha = 2$.

From this analysis can be deduced that, the amplification of the quantization noise on the signal path stimulates the output voltage and the more accurate information about the output filter can be obtained. The third case of the steady state SI refers to $f_{0,buck3}$. The PSD of $e[n]$ in function of α has been shown in Fig.7.32. In this case is visible as for $\alpha = 1$ the obtained maximum resonant frequency is not precise, while increasing the injected noise ($\alpha = 2$ and $\alpha = 3$) the identification results to be more precise. In particular, for $\alpha = 2$ the obtained resonant frequency (f_{02}) is equal to the desired one ($f_{0,buck2}$) and is not far when

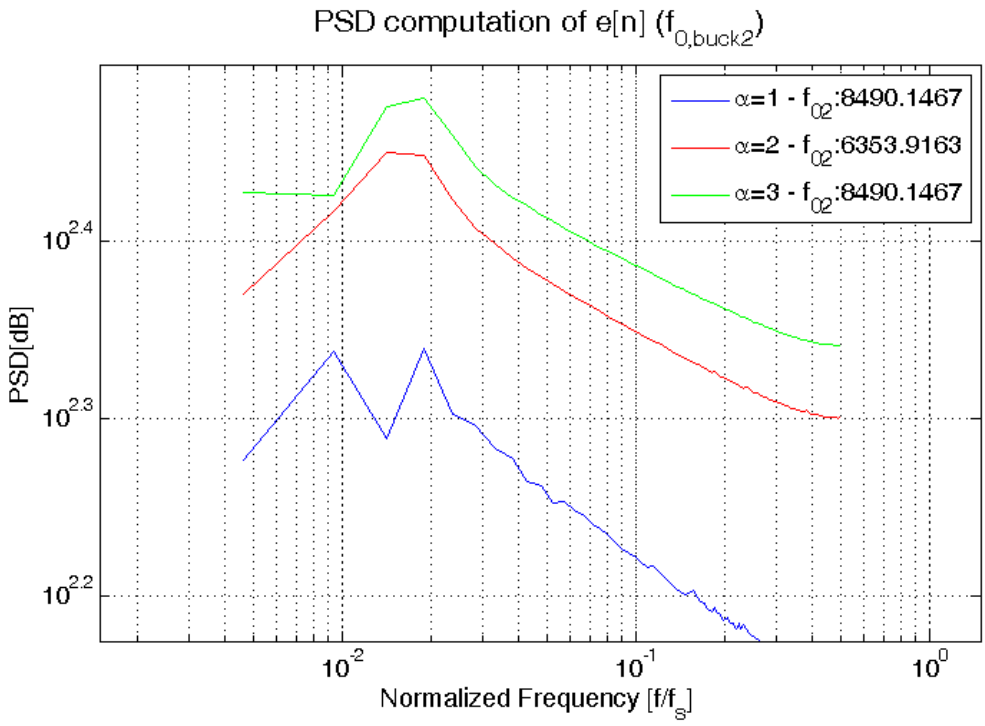

Figure 7.31: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$.

$\alpha = 3$ is considered.
From these analysis can be deduced as theoretical observations have been confirmed. The further amount of quantization noise due to the dithering permits to stimulate the buck converter output filter, even if the system still remains in the steady state. Moreover, output filter information can be obtained at the ADC output and the resonant frequency extracted as maximum of the PSD. In this range of considered loads and switching frequency, accurate identification results can be obtained with $\alpha = 2$, while, with $\alpha = 1$ results are not reliable and $\alpha = 3$ is less precise. Increasing the resonant frequency, the further amount of injected noise permits to identify the resonant frequency and for $\alpha = 2$ obtained values are very precise. Considerations just addressed, refer to samples $e[n]$ recorded during the same steady state window of $NT_s = 0.285msec$ ($N = 128$ and $T_s = 1/449kHz$). Fixing both the number of points N considered to compute the PSD and the switching frequency f_s, a finite resolution Δf (Eq.7.4) between frequencies which can be distinguished is obtained. The PSD processing results are then affected by the error between the obtained resonant frequencies and desired values values, because only values proportional to Δf can be obtained.
Because both $\alpha = 2$ and $\alpha = 3$ increase the precision on the identified resonant frequency and for $\alpha = 2$ results are very precise, a verification over more different acquisition NT_s can be done to confirm that $\alpha = 2$ ensures enough injection of white noise. The comparison between $\alpha = 2$ and $\alpha = 3$ is done considering four more different trials obtained for different acquisition windows NT_s. The trend is verified both for $f_{0,buck2}$ (Fig:7.33,7.34,7.35,7.36) and $f_{0,buck3}$ (Fig:7.37,7.38,7.39,7.40) because higher resonant frequencies are resulted much more affected from α variations. Results are summarised in Tab.7.1 and Tab.7.2, respectively for $\alpha = 2$ and $\alpha = 3$. Comparing averaged results in Tab.7.1 and Tab.7.2, can be observed

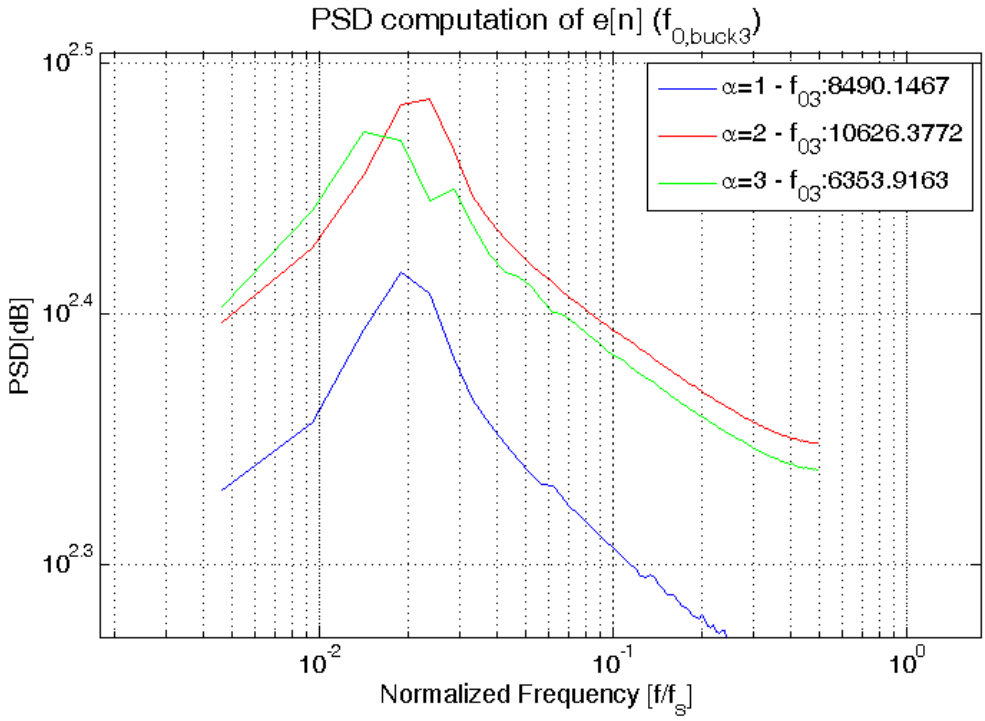

Figure 7.32: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$.

	$f_{02}[kHz]$	$f_{03}[kHz]$
window 1	8.5	10.6
window 2	6.4	8.5
window 3	6.4	10.6
window 4	8.5	10.6
average	7.45	10.075

Table 7.1: Steady state identification method results: Resonant frequency ($\alpha = 2$).

	$f_{02}[kHz]$	$f_{03}[kHz]$
window 1	6.4	8.5
window 2	6.4	10.6
window 3	4.2	8.5
window 4	8.5	10.6
average	6.38	9.55

Table 7.2: Steady state identification method results: Resonant frequency ($\alpha = 3$)

that for $\alpha = 2$ the obtained resonant frequency is always very close to the desired ones and for $\alpha = 3$ bigger oscillation in the results are obtained. Results obtained with $\alpha = 3$ can be compensated with averaging among the identifications, however $\alpha = 2$ ensures more precise results and lower perturbation on the output voltage.
Most of times the same best compensator can be use for buck converter configurations with close resonant frequency, indeed the magnitude of the LC-structure quickly rolls down to the resonant frequency with a slope $-40db/dec$. Consequently, a little error in the resonant frequency is not a dramatic from the regulation point of view. However, even if a little error could occur in some identifications, it can be compensated with averaging over the set of identification outputs or, considering the maximum inside of a set of extractions. The steady state methods cannot insert a big perturbation on the output voltage and a non-dramatic error on some identification can be accepted and easily compensated just with results averaging.

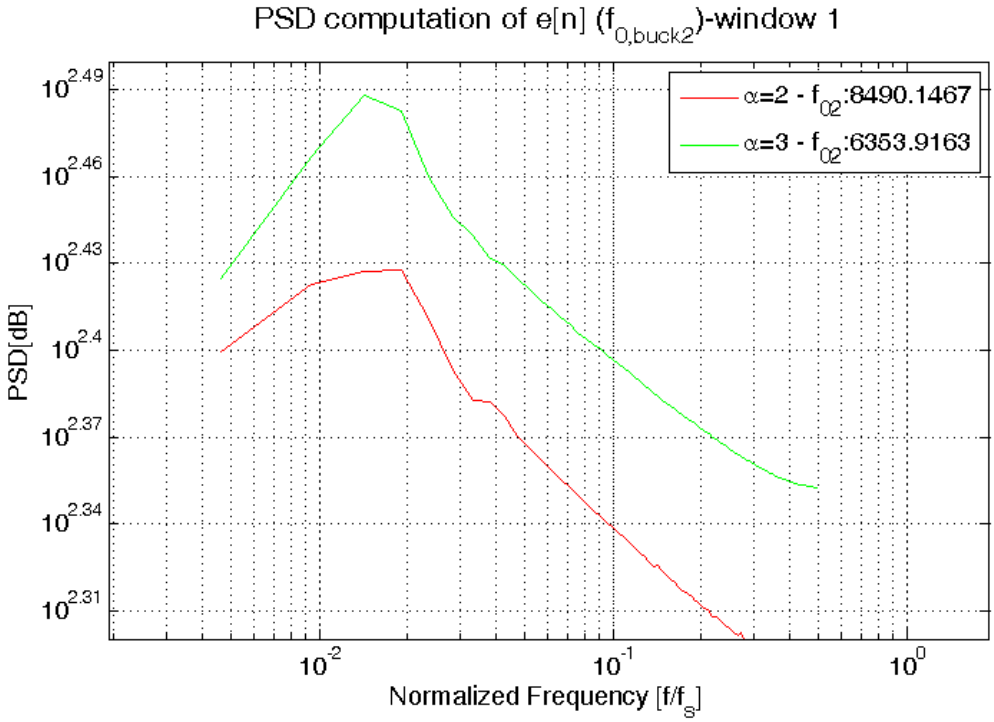

Figure 7.33: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 1).

7.2.3 ESR effects identification results

Non-idealities can modify the frequency behaviour of the control to output transfer function (Fig:7.1). The ESR contribution is strictly related to temperature increasing, the resulting zero at f_z can modify both bandwidth and margins of the system. Because the aim of the self-tuning algorithm is to design the best compensation law for the system and obtain both good margins and bandwidth, the compensator design have to take into account a possible zero close to system bandwidth. For this reason is important to have a steady state identification method which permits to monitor load configuration changing during the time life of

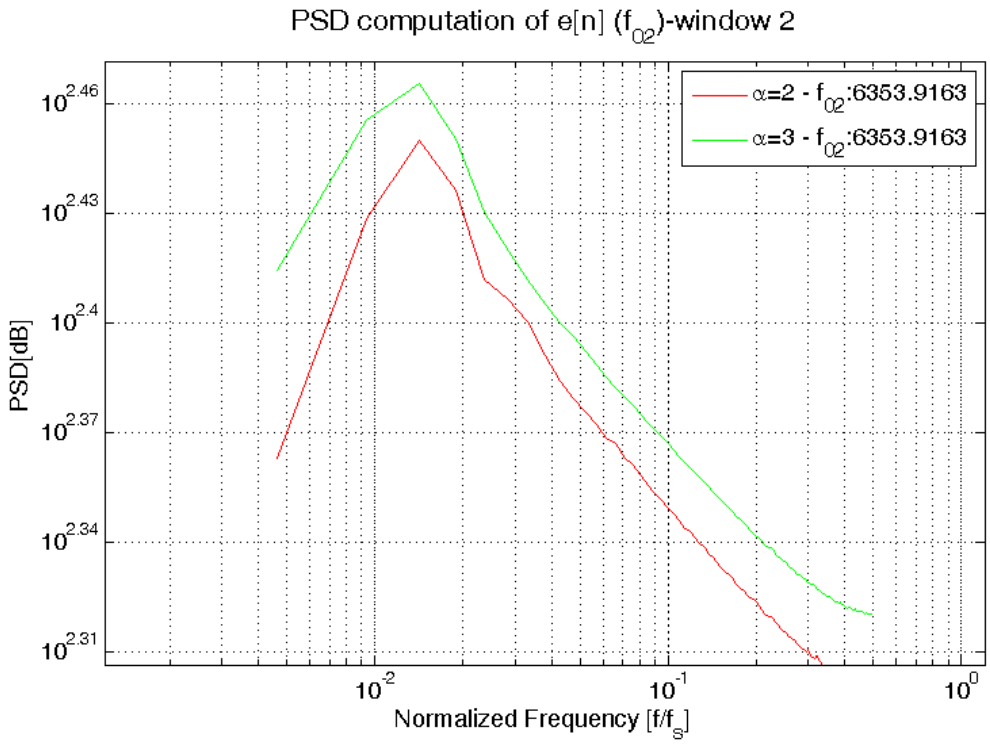

Figure 7.34: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 2).

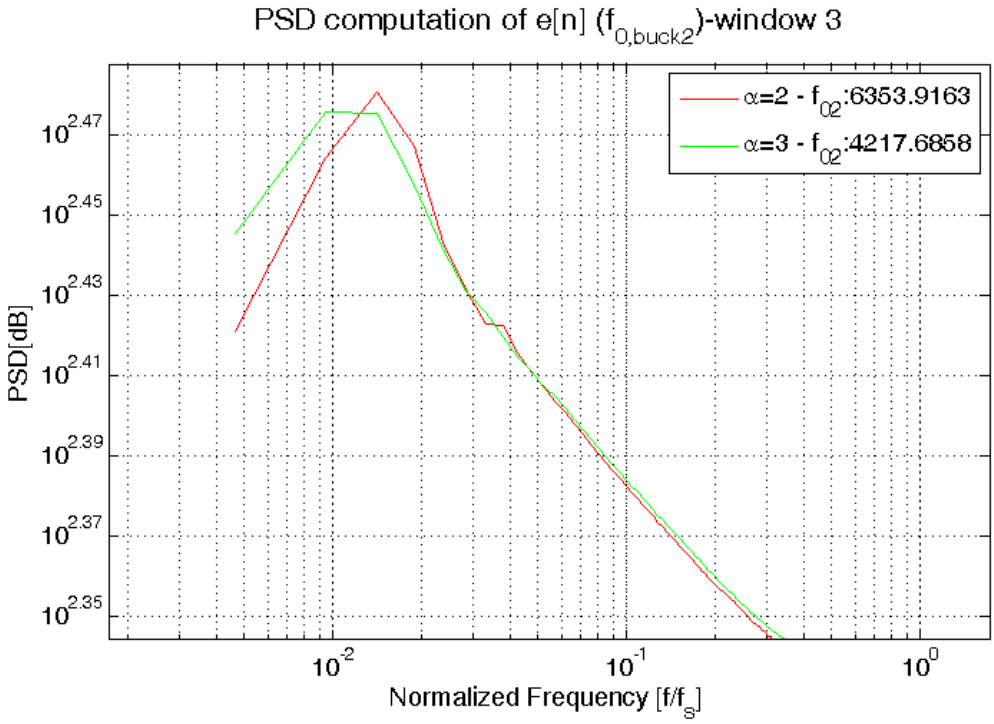

Figure 7.35: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 3).

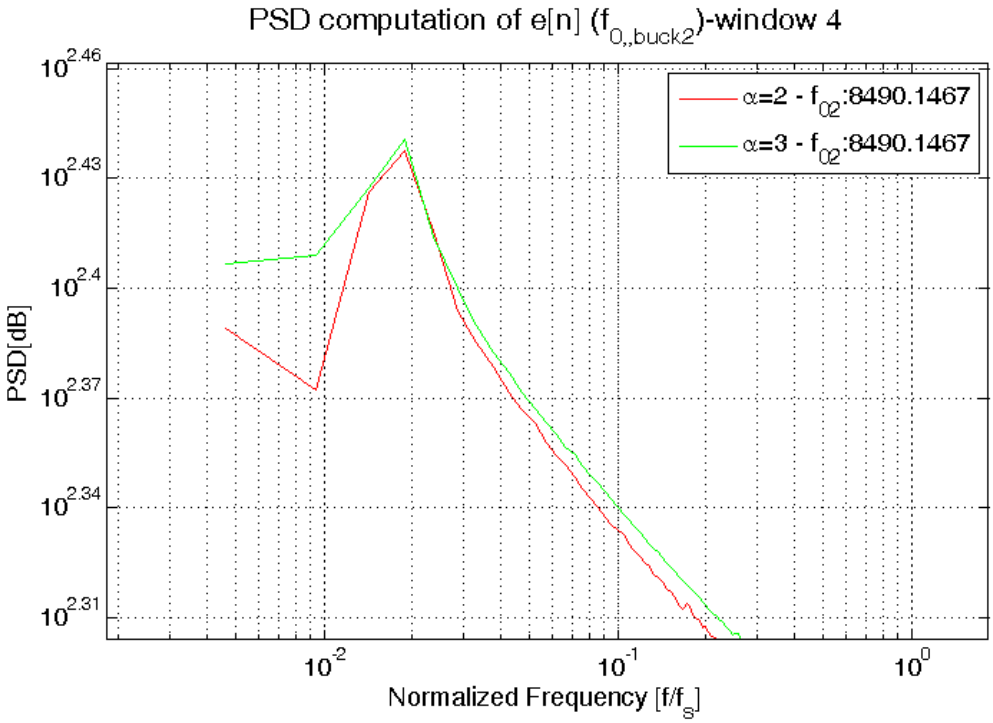

Figure 7.36: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0\Omega$ (trial 4).

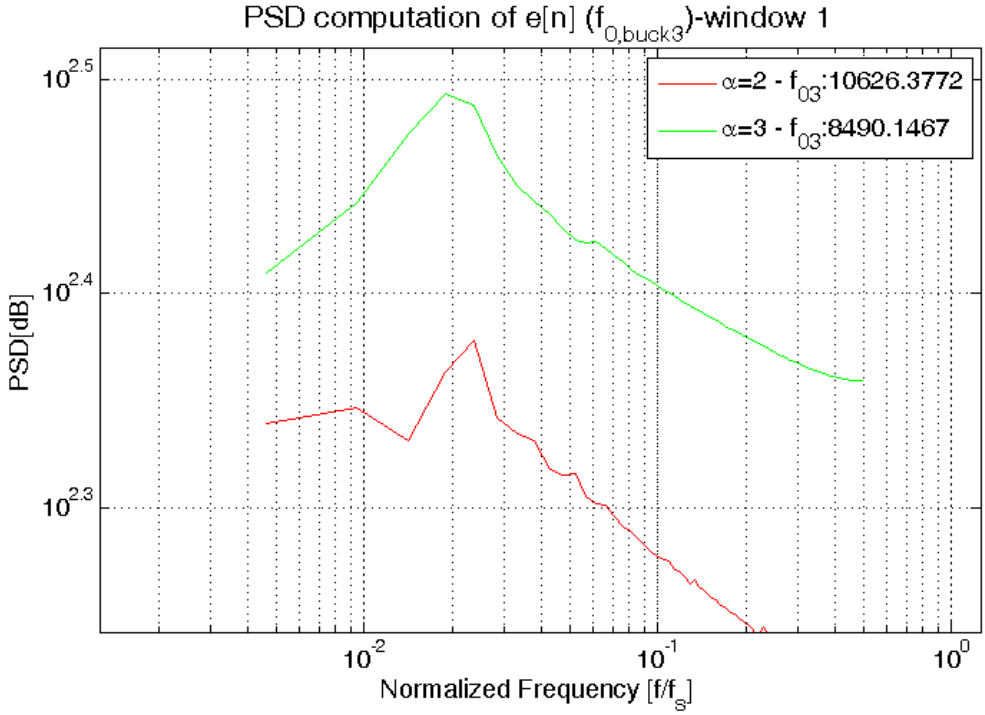

Figure 7.37: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 1).

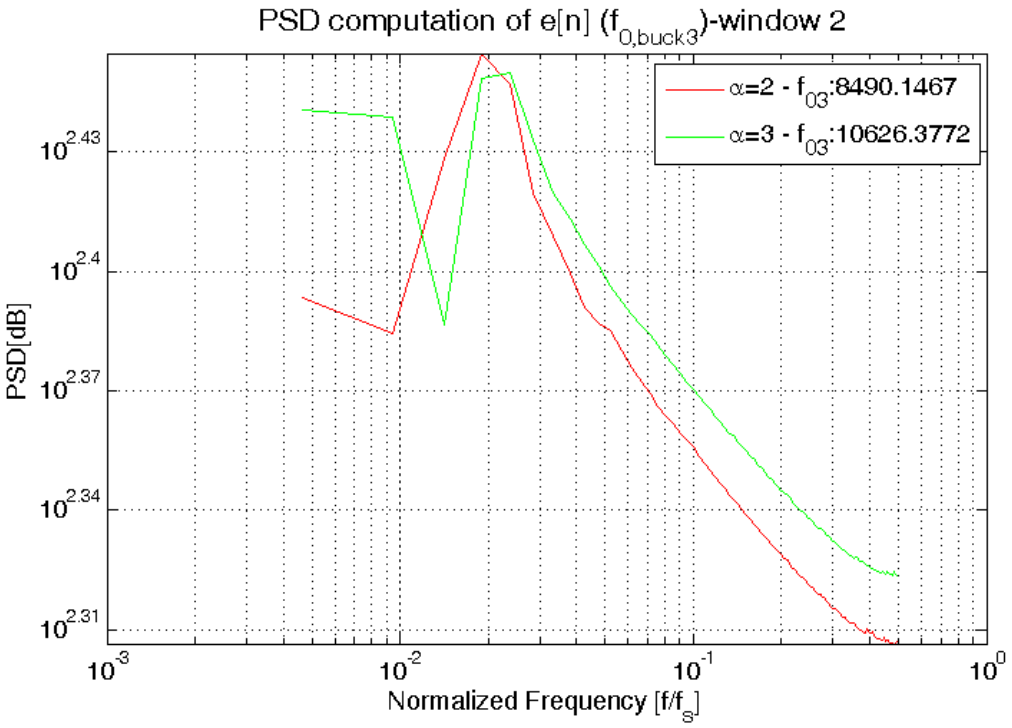

Figure 7.38: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 2).

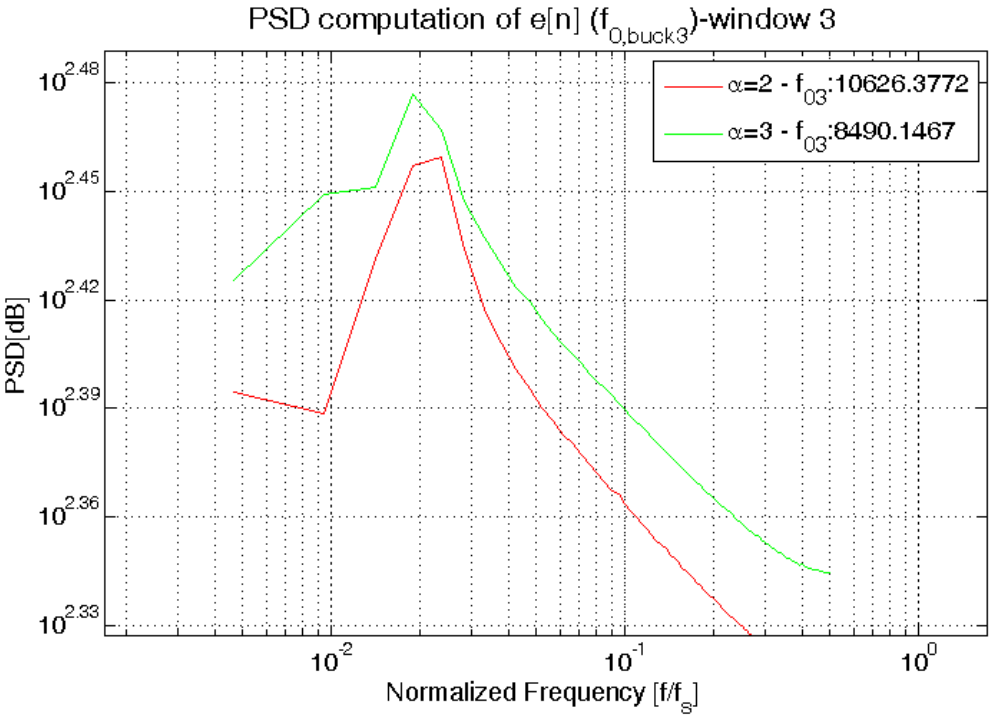

Figure 7.39: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 3).

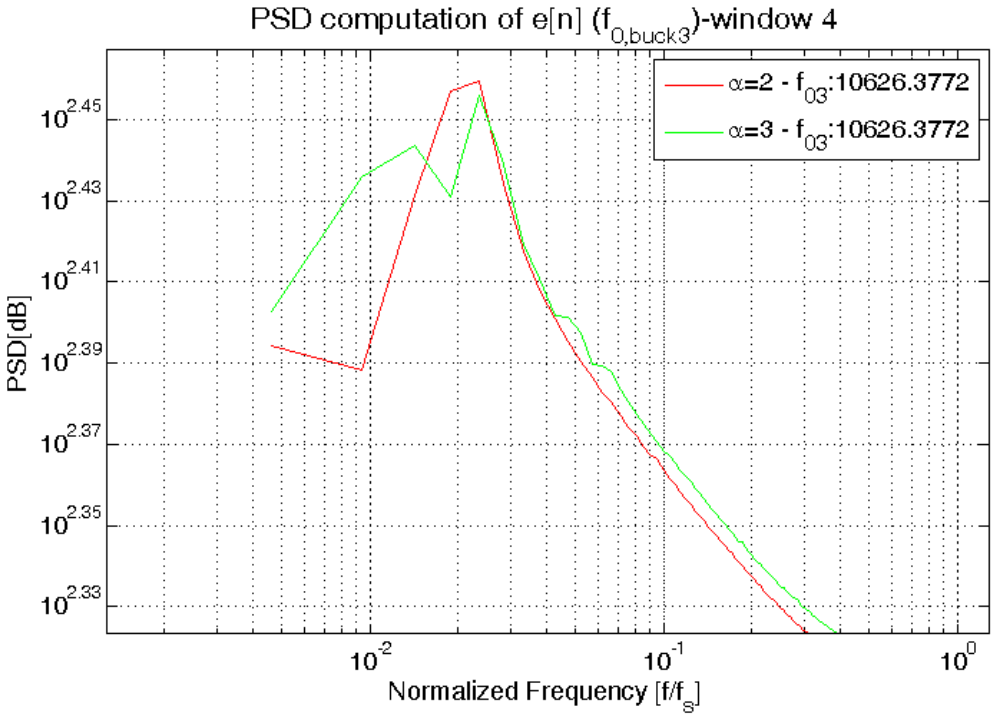

Figure 7.40: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$ (trial 4).

the converter.
Analysis in this section refers to the buck converter configurations shown in Fig.7.10 for $\alpha = 2$, the zero contribution have been considered respectively at $f_{z,buck1} = 14.5kHz$ ($ESR = 0.5\Omega$) for $f_{0,buck1} = 4.9kHz$, $f_{z,buck2} = 15.9$ ($ESR = 1\Omega$) for $f_{0,buck2} = 7.3kHz$ and $f_{z,buck3} = 33.8kHz$ ($ESR = 1\Omega$) for $f_{0,buck3} = 10.7kHz$. Results presented in this section refers to these three cases, the f_z identification is discussed referring to the comparison between $NTF_3(z)$ ($f_n = 0$) and the $NTF_{3k}(z)$ ($f_n \neq 0$), where the notch effect f_n is inserted at the resonant frequency obtained from the PSD computation with $NTF_3(z)$. The injected noise ($\alpha = 2$) can be shaped through the $NTF_{3k}(z)$ structure in order to concentrate a larger amount of noise on the range of interested frequencies. Because the zero contribution is close to the resonant frequency, the noise shaper structure can be modified ($NTF_{3k}(z)$) inserting a zero at the resonant frequency ($f_n = f_0$). In this way a further amount of noise can be concentrated on the range of the f_z and, moreover, the ESR contribution can be exalted while the maximum at the resonant frequency is reduced.
In order to verify that the zero in the noise shaper can be used to reduce the noise injection into a defined frequency, the notch in $NTF_{3k}(z)$ can be introduced at $f_n = f_{0,buck3}$ when no ERS contribution is considered. The dithering effect for $\alpha = 2$ in the configuration $f_{0,buck3} = 10.7kHz$ and $f_z = 0\Omega$, has been considered in in Fig.7.41. If none ESR contribution is considered, the zero inserted at $f_n = f_{0,buck3}$ in the noise shaper attenuates the peak at the resonant frequency while it exalts the information on the following range of frequencies. This is result proofs both that the further amount of noise is reflected on the output and that the noise shaper can be used to extrapolate output filter information through the modelling of the quantization noise shaper. Results shown in Fig.7.42 and Fig.7.43, refer to $ESR = 0.5\Omega$

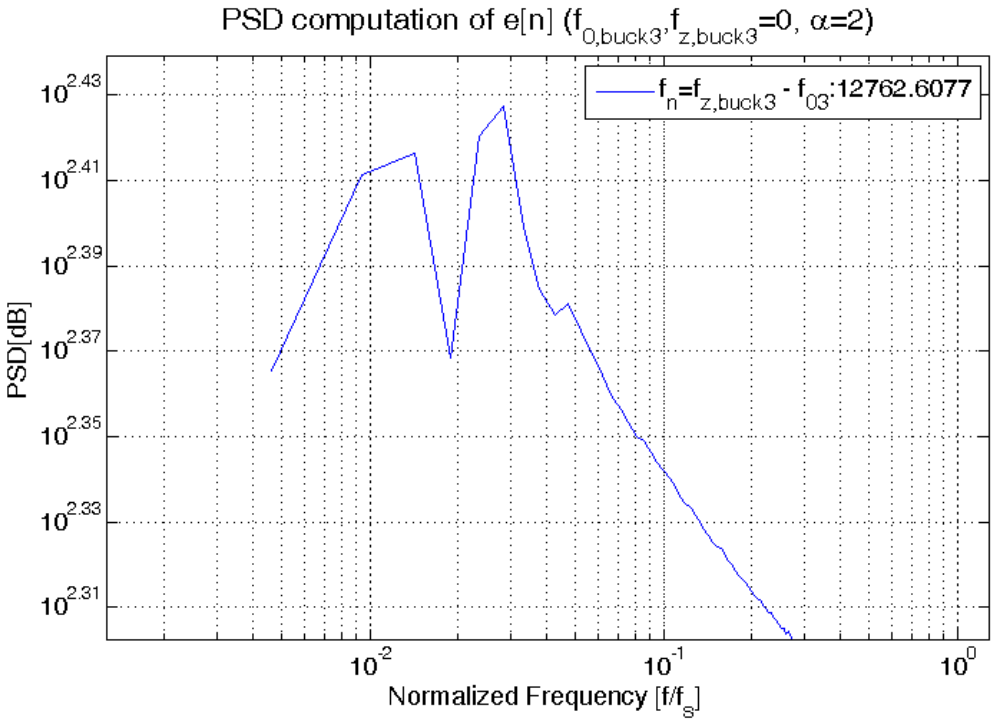

Figure 7.41: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 0\Omega$, $f_n = f_0$.

($f_{z,buck1} = 14.5kHz$) contribution for the configuration $f_{0,buck1} = 4.9kHz$. In Fig.7.42 both curves shown the maximum very close to the resonant frequency ($f_{02} = 4.2kHz$), in the blue curve (no notch) a third order $NTF_3(z)$ has been considered while in the red plot a zero at $f_n = f_{02} = 4.2kHz$ has been inserted in the noise shaper. Considering a third order modified $NTF_{3k}(z)$ a second peak can be detected at $f_z = 0.0332 f_s = 14.9kHz$, while with the third order noise shaper just a little change of slope can be observe in the frequency related to the ESR contribution. The same situation is repeated for the second example shown in Fig.7.43. The zero contribution due to the ESR is exalted concentrating the dithering in the frequencies following the resonant frequency. In these cases the blue curve does not show clear information about f_z, modifying the noise transfer function a clear peak related to the ESR contribution can be detect.

Results in Fig.7.44 and Fig.7.45 refer to $ESR = 1\Omega$ ($f_{z,buck2} = 15.9kHz$) contribution for the configuration $f_{0,buck2} = 7.3kHz$. In Fig.7.44 the main consequences to the zero insertion can be observed. In the blue curve ($f_n = 0$) the dithering amplification permits to identify the resonant frequency at $f_{02} = 8.5kHz$ and, inserting a zero at this obtained frequency in the noise shaper ($f_n = f_{02}$), the ESR contribution can be highlighted. Considering the $NTF_3(z)$, the resonant frequency can be detected as maximum of the PSD even if the presence of the ESR, while the ESR contribution cannot be clearly distinguished. Considering $NTF_{3k}(k)$ in the red curve, a second peak related to the zero contribution is presents at $f_z = 0.0332 f_s = 14.9kHz$. Almost the same considerations can be done referring to Fig.7.45. In this case none error is inserted in both detected resonant frequencies f_{02}, however the ESR contribution is highlighted at $f_z = 0.03796 f_s = 17.1kHz$ in the red curve with $f_n = f_{02}$.

In results shown in Fig.7.46, the $ESR = 1\Omega$ ($f_{z,buck3} = 33.8kHz$) contribution is con-

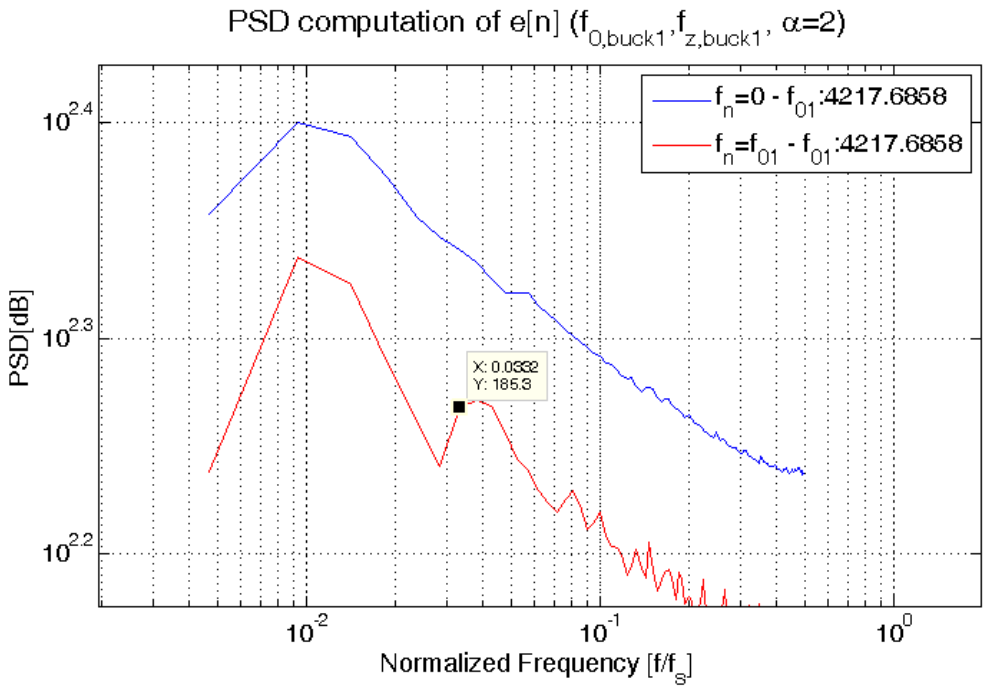

Figure 7.42: Steady state identification method PSD results: $f_{0,buck1}$, $ESR = 0.5\Omega$ (trial 1).

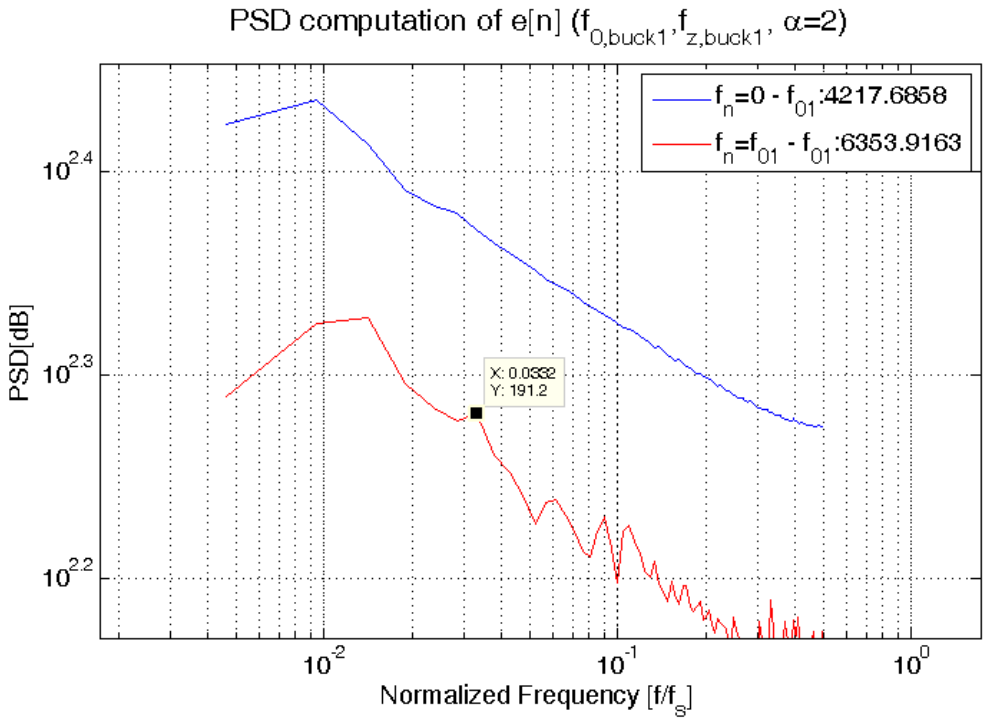

Figure 7.43: Steady state identification method PSD results: $f_{0,buck1}$, $ESR = 0.5\Omega$ (trial 2).

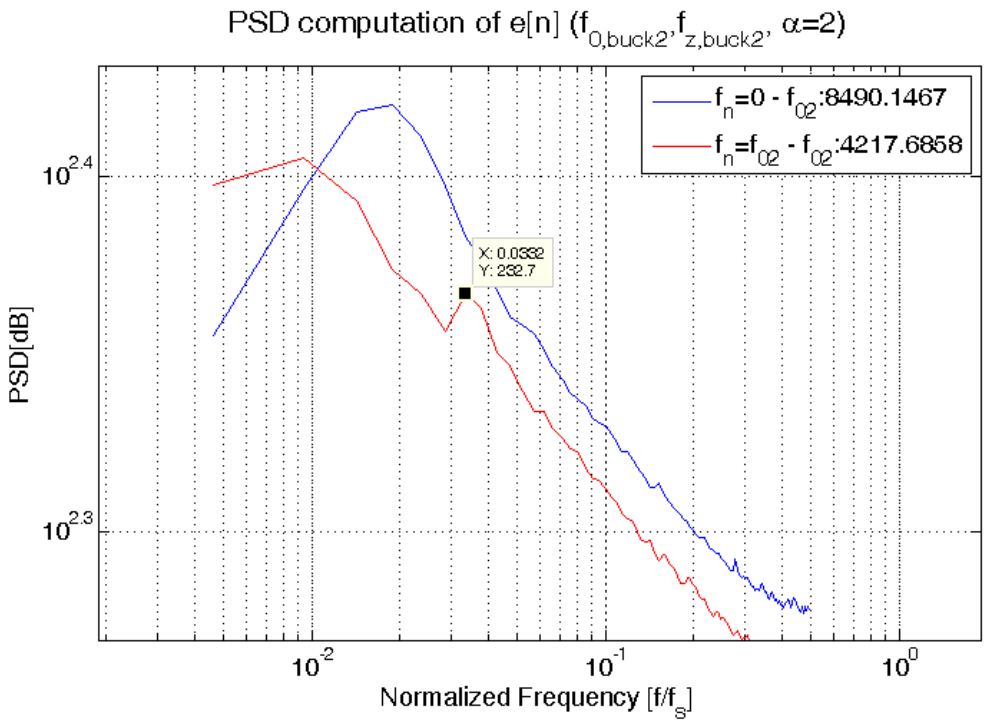

Figure 7.44: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 1\Omega$ (trial 1).

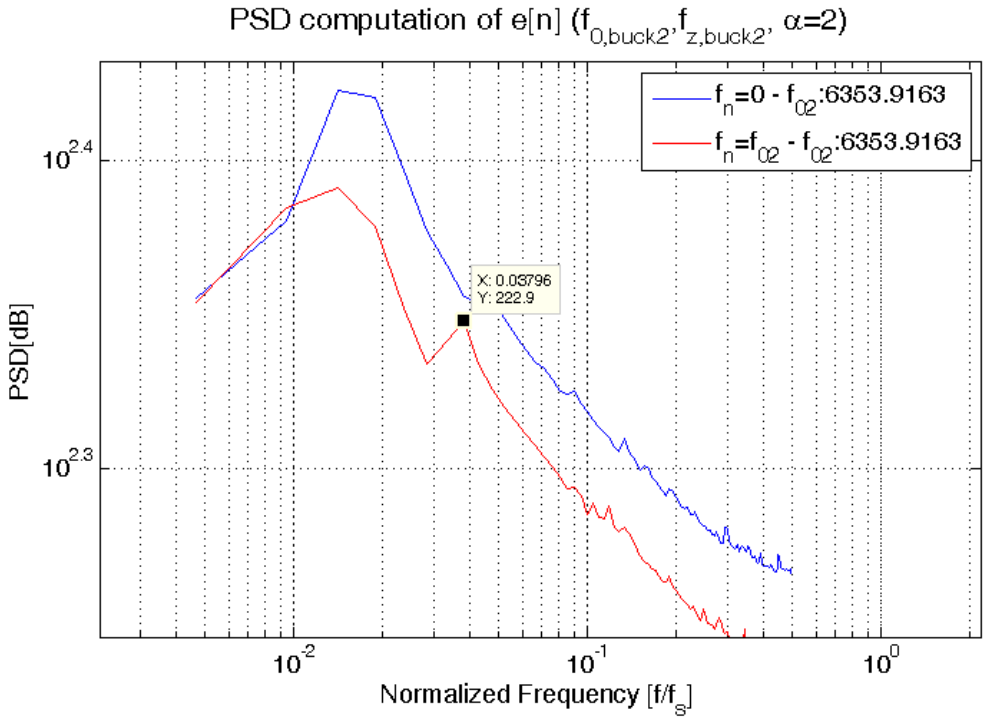

Figure 7.45: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 1\Omega$ (trial 2).

sidered for the configuration $f_{0,buck3} = 10.7kHz$. As said during the Open loop SI considerations for this converter configuration, the ESR contribution in this case is very far from the corner frequency that is close to the resonant frequency of the output filter for a second order LC filter. Indeed, the consideration drawn in this case can be the same done for the results shown in Fig.7.41, where the notch effect in the noise shaper is visible when none ESR is presents. In the results shown in Fig.7.46, the notch presence is visible at $f = 0.02368 f_s = 10.656kHz$ in the red plot, indeed, the zero in the noise shaper has been inserted at the resonant frequency $f_n = f_{03} = 10.6kHz$ derived from the blue curve. When the notch is detectable, the ESR contribution have to be observe at higher frequencies. When the notch effect noise shaper ends its contribution ($f = 0.03796 f_s = 17.1kHz$ in the red curve) the following frequencies are good candidates for the f_z detection. The first peak after $17.1kHz$ is at $f = 0.06177 f_s = 27.8kHz$ which is close to $f_{z,buck3}$.

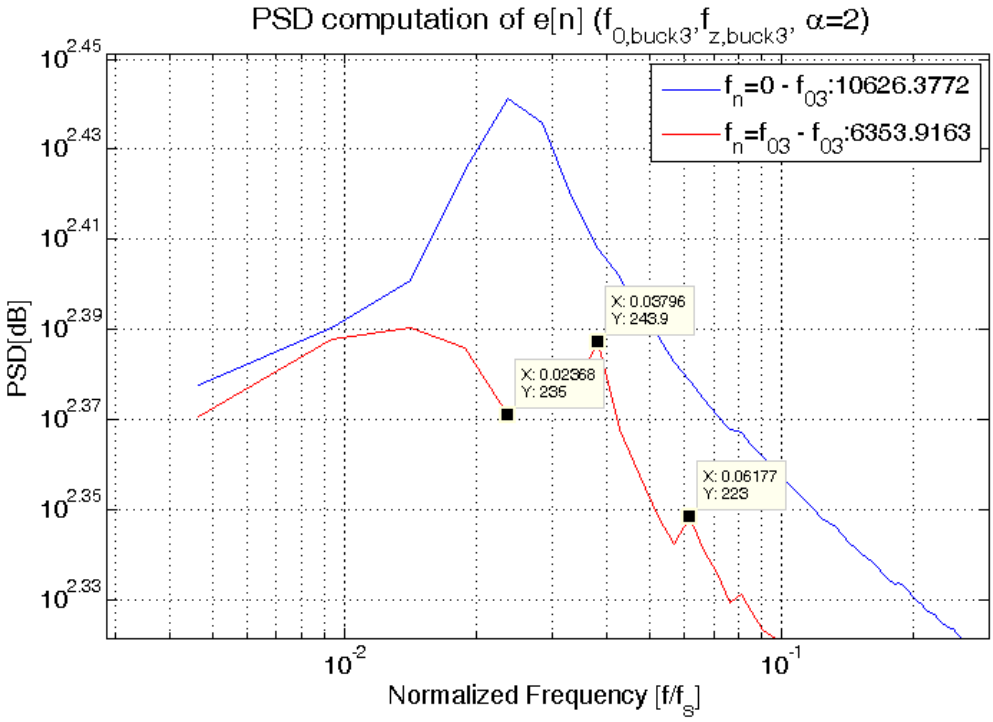

Figure 7.46: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 1\Omega$.

In the results shown in Fig.7.47 and Fig.7.48 the $ESR = 1.8\Omega$ contribution ($f_{z,buck3} = 18.8kHz$) has been evaluated for the configuration $f_{0,buck3} = 10.7kHz$. In these examples the trend described for $f_{0,buck2}$ and $f_{0,buck1}$ is confirmed. Even if the ESR presence, a classic third order $NTF(z)$ permits to identify the resonant frequency as maximum in the PSD. Inserting a zero in the noise shaper at this frequency a second peak related to the ESR contribution can be observed. The red curves obtained for $NTF_{3k}(z)$ present a second peak at $f_z = 0.04272 f_s = 19.2kHz$ very close to $f_{z,buck3} = 18.8kHz$, while the resonant frequency is still close to the calculated one but less precise then the blue curve where none zero in the noise shaper is inserted. To validate the algorithm three different ESR values has been introduced in the configuration $f_{0,buck2}$, related results have been presented in Fig.7.49, Fig.7.50 and Fig.7.51 respectively for $ESR = 1.2\Omega (f_{z,buck2} = 13.3kHz)$, $ESR = 0.9\Omega (f_{z,buck2} =$

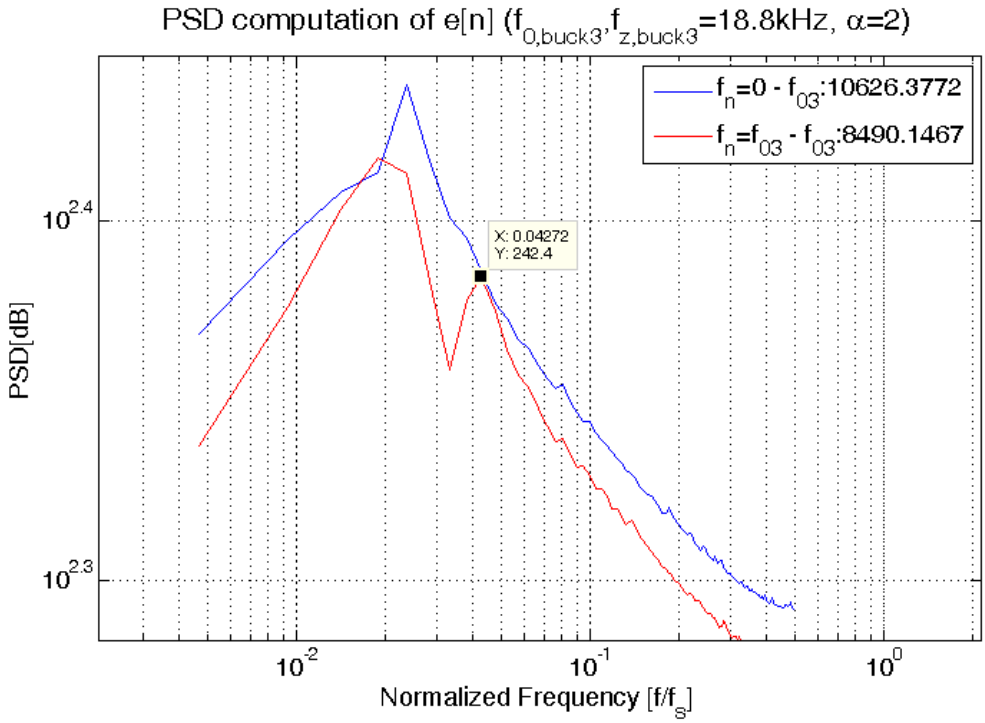

Figure 7.47: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 1.8\Omega$ (trial 1).

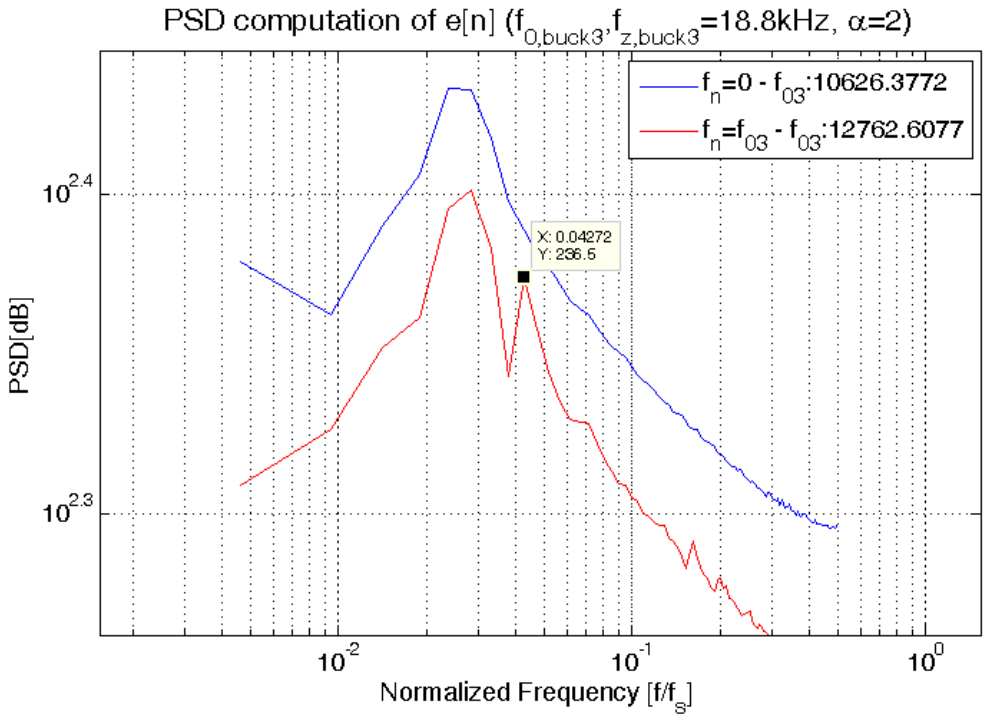

Figure 7.48: Steady state identification method PSD results: $f_{0,buck3}$, $ESR = 1.8\Omega$ (trial 2).

$17.6kHz$) and $ESR = 0.8\Omega(f_{z,buck2} = 19.9kHz)$. For this buck converter configuration four different ESR contributions are evaluated in order to verify the precision of the detected f_z. For $ESR = 1.2\Omega$ (Fig.7.49), the resonant frequency can be obtained as maximum in the PSD with $NTF_3(z)$ ($f_n = 0$) while information related to the ESR contribution can be obtained from the red curve where $f_n = f_{03}$. A second peak is present at $f_z = 0.02844 f_s = 12.8kHz$ and is very close to the expected zero contribution $f_{z,buck2} = 13.3$. For $ESR = 0.9\Omega$ (Fig.7.50) the zero contribution detected in the red curve through the $NTF_{3k}(z)$, is at $f_z = 0.03796 f_s = 17.1kHz$ very close to $f_{z,buck2} = 17.6kHz$. In Fig.7.51 a lower ESR contribution corresponding to $f_{z,buck2} = 19.9kHz$ is considered and, the insertion of notch effect in the noise shaper highlights a second peak at $f_z = 0.04272 f_s = 19.2kHz$. A third order noise transfer function,

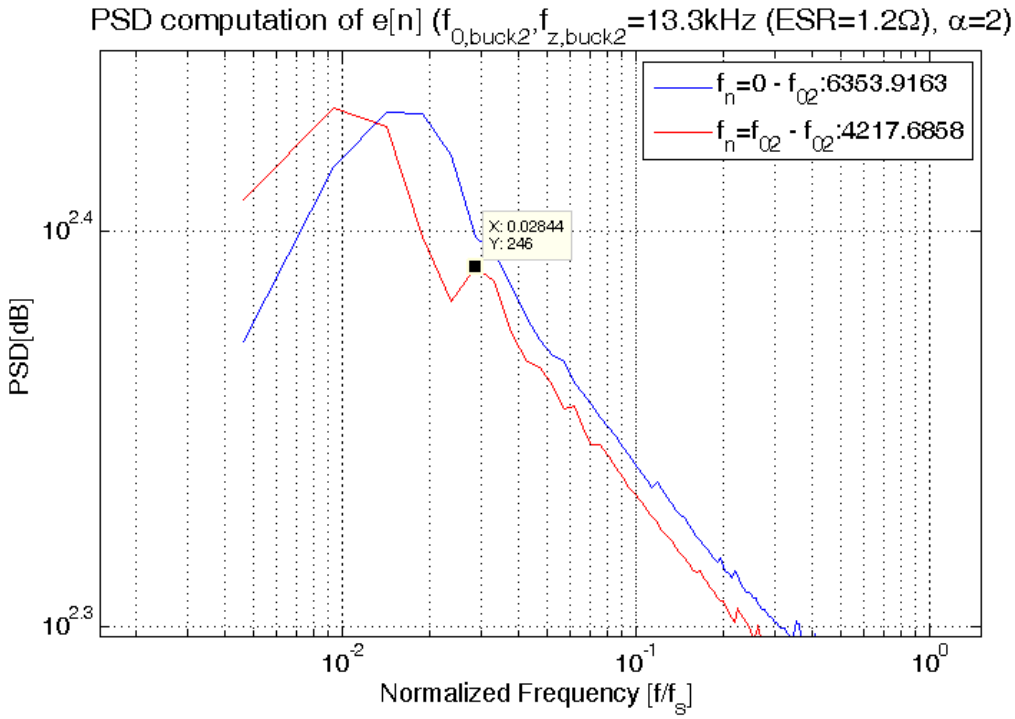

Figure 7.49: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 1.2\Omega$

combined with the amplification of the dithering effect ($\alpha = 2$) permits to detect the resonant frequency even if an ESR contribution is considered. From PSD analysis on $e[n]$, the resonant frequency can be detected as the maximum of processing, while, the zero introduced by the ESR contribution can be detect as a second peak in the PSD only if the noise shaper is modified. Through the notch effect inserted at the resonant frequency, a third order modified $NTF_{3k}(z)$ ensures to address the amplified noise in the range of frequencies related to the ESR contributions. In this way the zero contribution can be stimulated and detected as well with peak searchings.
The f_z identification results are summarised in Tab.7.3, where the obtained ESR contributions (f_{z1}, f_{z2} and f_{z3}) for the related buck converter configurations ($f_{0,buck1}$, $f_{0,buck2}$ and $f_{0,buck3}$) are compared with the expected values $f_{z,buck1}$, $f_{z,buck2}$ and $f_{z,buck3}$ defined for each considered ESR contribution. When the ESR contributions are close to the resonant frequency their contribution is detect with good precision, while a error occurs when the

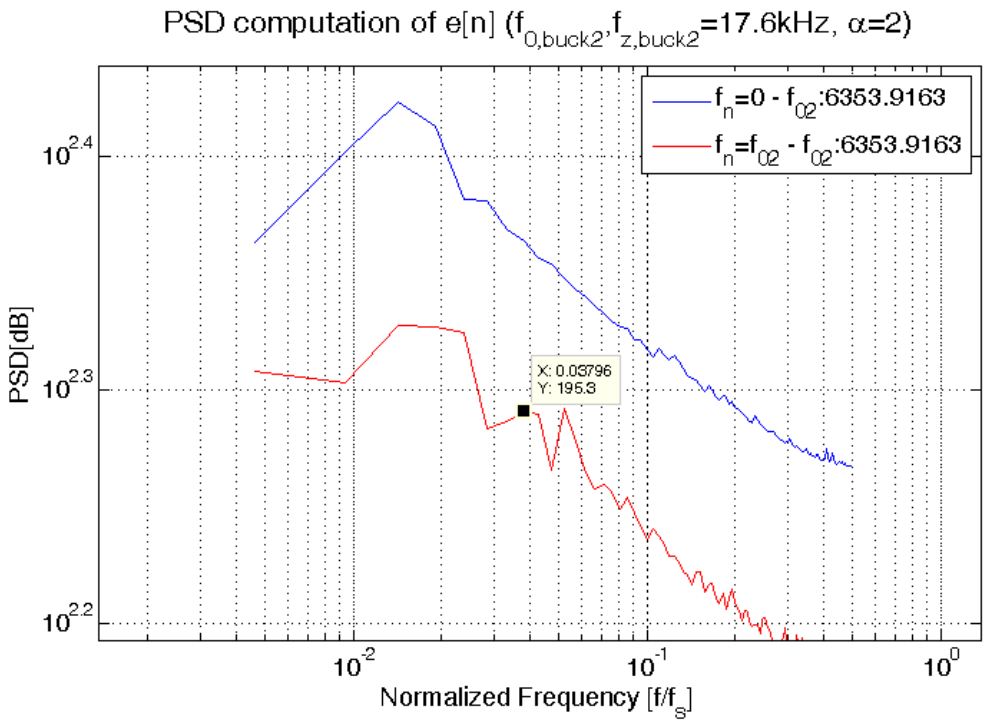

Figure 7.50: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0.9\Omega$.

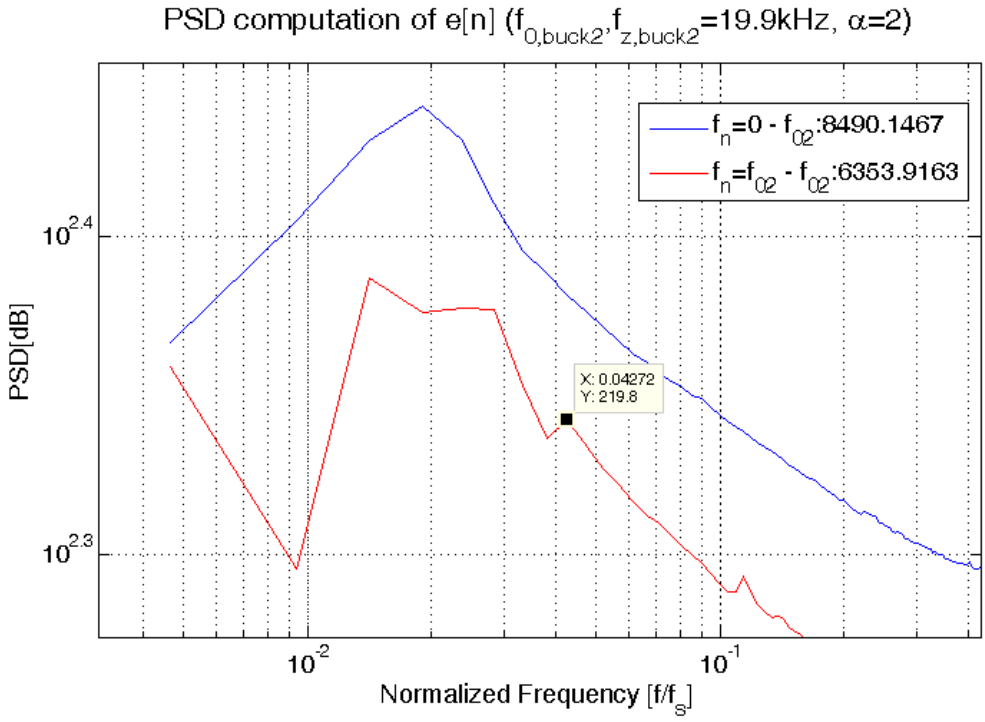

Figure 7.51: Steady state identification method PSD results: $f_{0,buck2}$, $ESR = 0.8\Omega$..

$ESR[\Omega]$	$f_{z1}[kHz]$	$f_{z,buck1}[kHz]$	$f_{z2}[kHz]$	$f_{z,buck2}[kHz]$	$f_{z3}[kHz]$	$f_{z,buck3}[kHz]$
0.5	14.9	14.5	-	-	-	-
0.8	-	-	19.2	19.9	-	-
0.9	-	-	17.1	17.6	-	-
1	-	-	14.9	15.9	27.8	33.8
1.2	-	-	12.8	13.3	-	-
1.8	-	-	-	-	18.8	18.8

Table 7.3: Steady state identification method results: ESR identification ($\alpha = 2$ $f_n = f_0$).

contribution is far from f_0. When considerable ESR contribution are very close, the algorithm permits to distinguish the related f_z. Indeed, for $f_{0,buck}$ the f_z contribution can be distinguished between $ESR = 0.8\Omega$ and $ESR = 0.9\Omega$.

7.3 Processing and extraction algorithm design specifications

The extraction algorithm is based on the PSD computation results and is aim is to extract the resonant frequency f_0 and the f_z related to the ESR contribution. The model presented in Fig.7.52, represents the approach used in the previous sections to show the results of the steady state identification method. The output voltage from the buck converter is read in digital domain at the ADC output, the error is recorded during the steady state for a window length of NT_s and processed through the PSD computation. All results have been shown in logarithmic scale, while the extraction algorithm have to be designed on the linear scale of the PSD output. The algorithm used to compute the PSD is based on the FFT of the autocorrelated N-length window of errors $e[n]$:

$$corr(e[n], N) = \mathscr{F}^{-1}\{\mathscr{F}\{e[n]\}\mathscr{F}\{e[n]\}^*\}. \tag{7.17}$$

The Eq.7.17 computes through the FFT algorithm the autocorrelation of N-length $e[n]$ which is long four times N. After the zero padding the FFT of Eq.7.17 have to be computed in order to obtain the PSD. The comparison between logarithmic and linear PSD output is presented in Fig.7.53 for the buck converter $f_{0,buck1}$. Blue curves are related to $ESR = 0\Omega$, while in the red ones $ESR = 0.5\Omega$ has been considered. A second peak related to $f_{z1} = 14.1kHz$ can be observed (in both red curves), next to the maximum related to the resonant frequency.

In Fig.7.54 is shown the PSD output in linear scale computed for $N = 2^9$ samples of $e[n]$, the results is a $4N$ length vector. It can be observed as, the processing output is a four time mirrored vector, and just one fourth can be studied to extract the load information. The extraction algorithm can easily define as a maximum searching function, that that have to distinguish two peaks on one fourth of the PSD output.

A VHDL-coded PSD computer has been designed and tested during a M.S. thesis work (Sec.9.4). The hardware implemented processing algorithm, refers to considerations just drawn and is based on these main steps:

- autocorrelation computed as in Eq.7.17. This imply the FFT, the complex conjugate and the computation of a hardware multiplication.

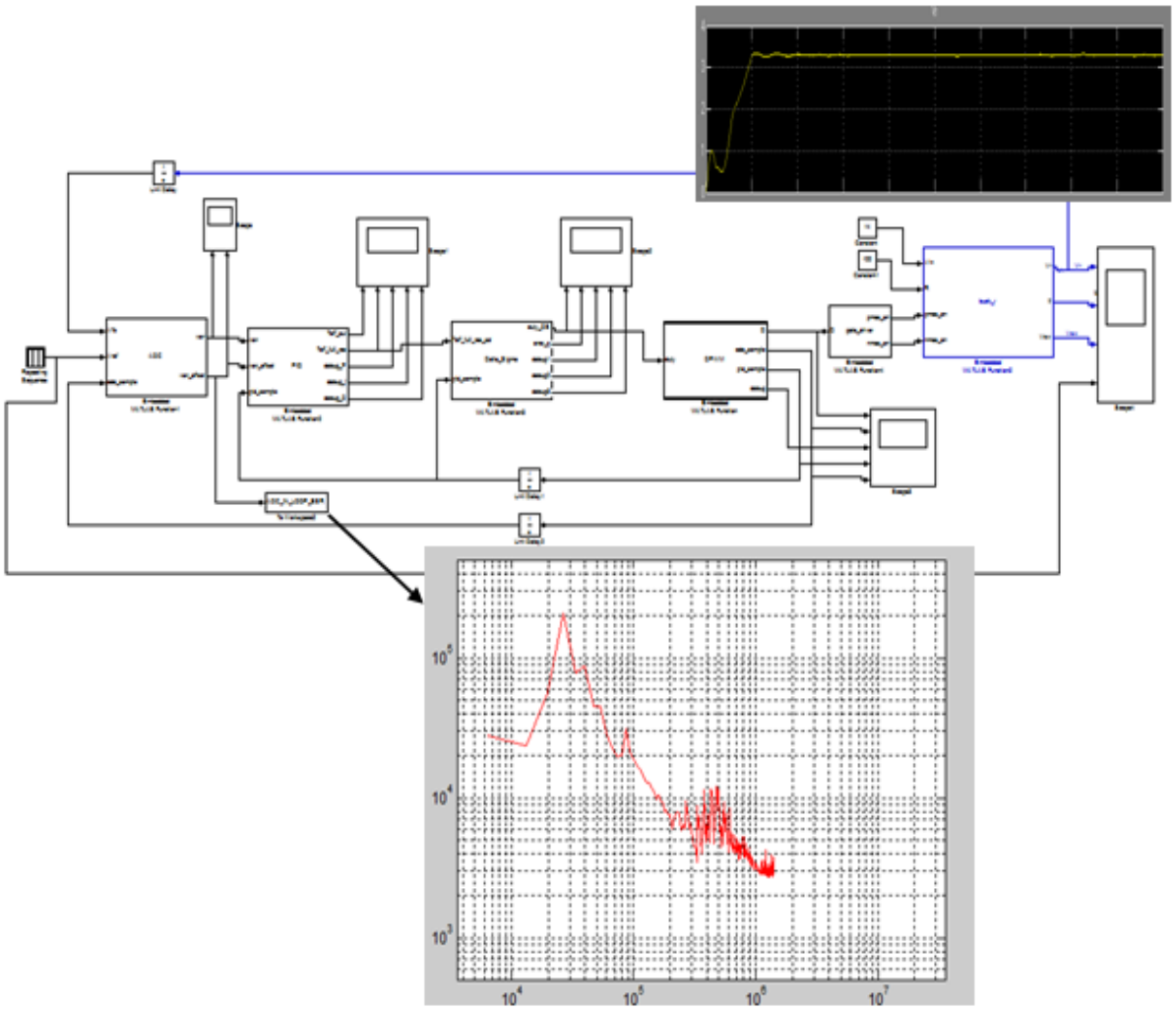

Figure 7.52: Steady state identification method: Matlab/Simulink fixed-point model.

- Inverse FFT computation and zero padding.
- FFT transform.

The hardware PSD computation involves the computation of three FFT, one inverse FFT and one hardware multiplication. Fast Fourier Transform and its hardware implementation are largely exploited, a wide range of custom FFT processor architectures can be found in literature. Pipelined structures have both high throughput and computational efficiency, but when area constraints are stronger than timing, non-pipelined counterparts are usually preferred. Scalable architectures in terms of number of processing elements (PEs), are flexible solutions. Astola and Akopian [8] introduced a family of hardware-oriented algorithms resulting in scalable constant geometry (CG) structures. In [107] a not-in-place architecture targeted for ASIC implementation is proposed. This solution uses data shuffling registers, thus it does not require a memory for intermediate results. Classic approaches to FFT designs require a large amount of memory for storing precomputed twiddle factors. The

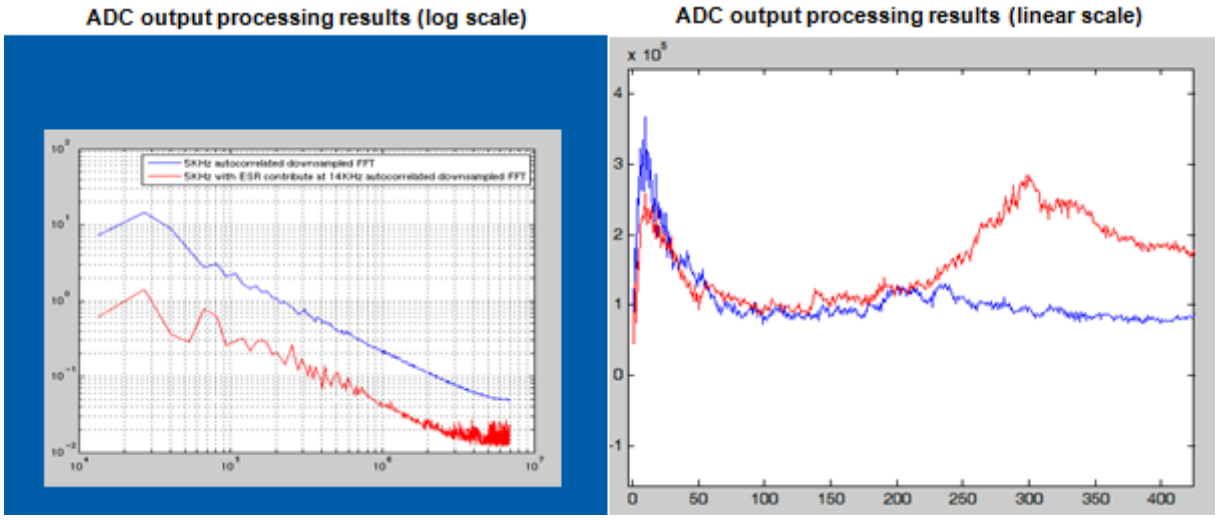

Figure 7.53: Steady state identification method PSD results: linear and log scale ($f_{0,buck1}$).

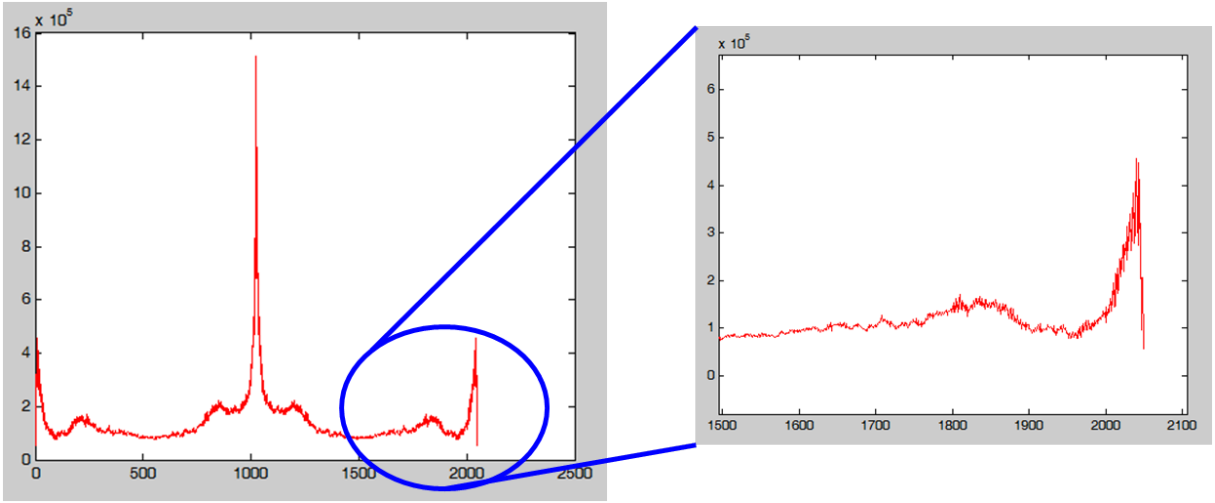

Figure 7.54: Steady state identification method PSD results: linear scale ($f_{0,buck1}$).

CORDIC iterative algorithm presented in [115] allows computation of twiddle factors at runtime. To this purpose many architectures introduce this algorithm inside the PEs by substituting the complex multipliers with iterative phase rotations. With this approach, in [130] the number of iterations has been reduced by using optimized sequences and corresponding scale factors both stored in a LUT. In [122] a multi-bank RAM structure to reduce memory logic is presented. On the other hand, some systems replace the twiddle factor storage ROM with a CORDIC-based generation system. Nonethless, CORDIC hardware implementations can be very expensive in terms of area usage [11, 34], consequently hardly suitable for a scalable FFT approach.
Observing the twiddle factor dataflow during the FFT computation (App.B), a novel hybrid algorithm for twiddle factor generation has been presented in [24] as core function of the FFT processor implemented in Sec.9.1. Only a reduced set of phase factors have to be stored in a LUT, missing twiddle factors can be generated from these with a simple CORDIC multiplication(Sec.9.3). Considering this observation, in [24] a novel scalable CORDIC-LUT approach related to the twiddle factor computation has been introduced in a standard CG-FFT computer. This way a full-custom scalable hybrid CORDIC-LUT processor for Constant-Geometry FFT computations has been implemented during the thesis work. The new HDL-coded architecture for Digital Signal Processing has been designed and integrated in the FPGA, providing 100% flexibility for concept customization thanks to full scalability both in terms of resource saving and computational latency.
A complete description of the atucorrelation-based DSP has been presented in Cha.9, together with the implemented scalable hybrid FFT architecture. The PSD hardware computation has been optimized considering its mirrored output (Fig.7.54), at the end of the computation just samples from $2N$ to $4N$ could be stored in a memory. Referring to Fig.7.54 where the PSD is computed for $N = 2^9$ samples, the HDL coded extraction algorithm have to work on samples from 1024 to 2048 which have the mirrored information which contain samples from 0 to 1023.
The information about the resonant frequency is obtained as maximum searching within the range of samples from $3N$ to $4N$. This range has been zoomed in Fig.7.54 for $N = 512$, as shown in Fig.7.53 the maximum in this range corresponds to the resonant frequency. Regarding the f_z extraction, it results easier to search the peak that occurs after the firs change of slope met scanning the range of samples from $2N$ to $3N$.

Chapter 8

Steady state self-tuning prototype: On-line controller

Aim of this chapter, is the hardware validation of the steady state identification algorithm and the realisation of a self-tuning FPGA-TC based on-line controller prototype (Fig.6.1). The hardware validation of the steady state SI, has been done considering the FPGA-TC prototype described in Cha.5. The external loop configuration presented in Sec.5.2 is used to observe the ADC output $e[n]$ directly on the FPGA. The validation model has been shown in Fig.8.2. In this approach, processing results are computed with Matlab fixed-point emulation model of the VHDL-coded PSD computer described in Sec.9.4 and, introduced in Sec.7.3. The PSD fixed-point model described in Sec.9.5, has been coded to debug the hardware implementation of the PSD computer.

Once the SI algorithm and the PSD computer have been verified, the steady state self-tuning on-line controller has been prototyped. This model is presented in Fig.8.1 and represents the on-line controller approach shown in Fig.6.1). The digital control loop (Fig.5.1) designed for the off-line controller, has been integrated together with the hardware needed for the identification algorithm (Extraction-regulation and PSD computation block). The PSD computation is based on the VHDL-coded system presented in Sec.9.4. The implemented load parameters identification method, is performed in two steps. A first step is used for the resonant frequency identification and a second one is used for the f_z identification. Once that the identification and the extraction algorithm (introduced in Sec.7.3 and implemented in Sec.9.4.3) end their computation, the regulation PID gains are consequently changed. To distinguish among these phases a Finite State Machine (**FSM_proc**) is introduced in the self-tuning model. The self-tuning prototype has been presented in[23].

8.1 Steady state system identification method validation

Identification results presented in this section, are obtained considering the FPGA-TC prototype (Cha.5). The ChipScope interface (Fig.5.9) permits to record and save the ADC output, this dataset can be saved and processed with the Matlab fixed-point emulation model of the full scalable PSD computer. The model used for the validation algorithm has been shown

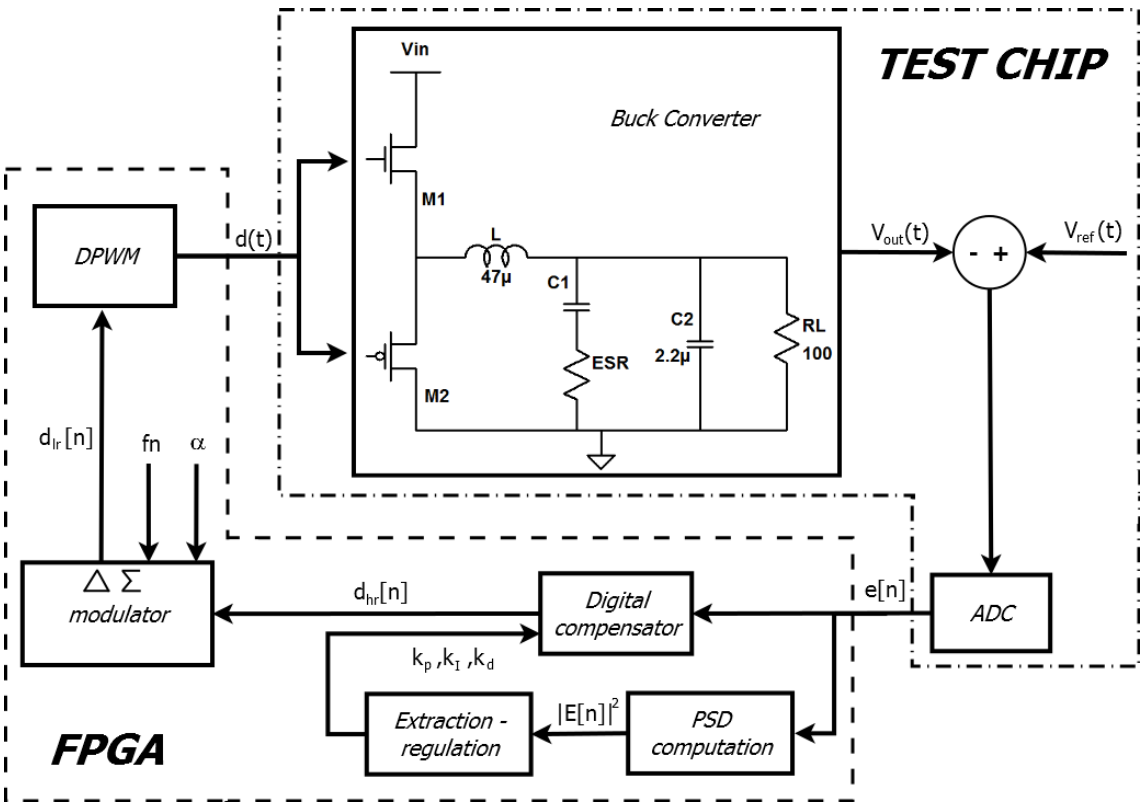

Figure 8.1: Steady state identification prototype: model.

in Fig.8.2. The aim of this approach is to validate the steady state SI technique, to study the extraction algorithm and debug the PSD computer.
The steady state identification results have been presented in Sec.7.2.3 and Sec.7.2.2, respectively for the resonant frequency and the ESR contribution identification. It has been demonstrated that dithering doubling joined with the modified noise shaper, permits to observe the buck converter frequency parameters (f_0 and f_z) through the PSD computation of $e[n]$. Referring to ΔΣ modulator presented in Fig.7.16, the inputs α and f_n respectively permit to amplify the dithering effect and to insert a zero at f_n in the noise shaper. The hardware solution related to the notch insertion has been presented in Sec.4.2.2. The noise shaper process $n_h - n$ bits when $\alpha = 1$. During the identification, $\alpha = 2$ and the quantization noise is doubled, hence, one more bit have to be considered in the noise contribute. In Fig.8.3 is shown the ΔΣ hardware encoding used to compute the duty cycle (*duty_low_res_s*). As described in Sec.4.2.2, when the modulator is used (*delta_sigma_en_i*=1) the dithering effect is considered on the signal path otherwise the incoming high resolution duty cycle is directly output (*duty_low_res_s* <= *duty_low_res_i*). When dithering effects are considered, the signal *duty_first_full_res_s* is computed by summing the high resolution input duty cycle with the quantization noise computed by the noise shaper during the previous switching cycle. The amplification of dithering effect is selected through the signal *res_quant_ds_i* which represents $\alpha - 1$. The dithering effect is related to αq_e, when *res_quant_ds_i=00* the quantization noise is not amplified ($\alpha = 1$, no identification). When the identification method is activated, the signal *res_quant_ds_i=01* and the quantization noise in input to the noise

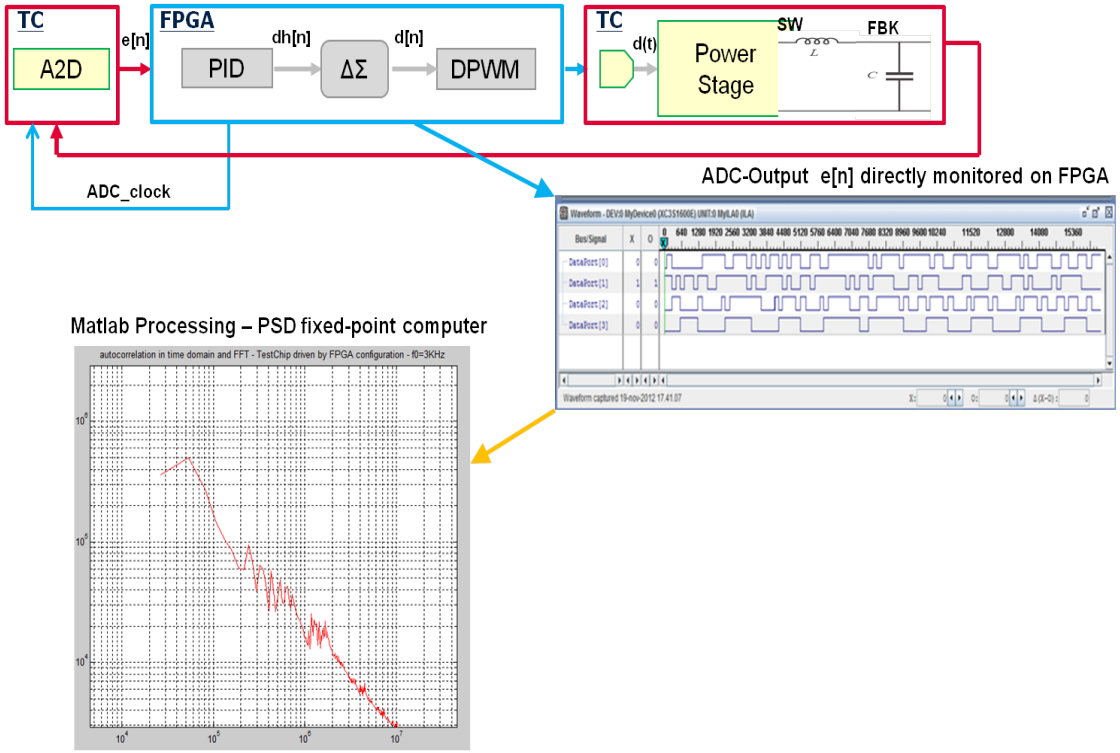

Figure 8.2: Steady state identification prototype: testing environment.

shaper *duty_quant_err_s* is doubled ($alpha = 2$).

```
duty_low_res_p : process (clk_i, rst_n_i)
 begin  -- process
   if rst_n_i = '0' then                    -- asynchronous reset (active low)
     duty_low_res_s <= (others =>'0');
     duty_quant_err_s <= (others =>'0');
   elsif clk_i'event and clk_i = '1'  then  -- rising clock edge
     if delta_sigma_en_i = '1' then --using delta sigma
        if res_quant_ds_i = "00" then  -- alpha = 1
           duty_low_res_s <= std_logic_vector(duty_first_full_res_s(22 downto 11));
           duty_quant_err_s <= -("00000000000000000" & duty_first_full_res_s(10 downto 0));
        elsif res_quant_ds_i = "01" then  -- alpha = 2
           duty_low_res_s <= std_logic_vector(duty_first_full_res_s(22 downto 12)&'0');
           duty_quant_err_s <= -("0000000000000000"  & duty_first_full_res_s(11 downto 0));
        end if;
     else
        duty_low_res_s <= duty_low_res_i;
        duty_quant_err_s <= (others =>'0'); --no dithering
     end if;
   end if;
  end if;--
 end process duty_low_res_p;
```

Figure 8.3: Hardware dithering amplification: ΔΣ modulator.

8.1.1 Results

On the basics of the previous chapter, doubling the quantization noise during the identification permits to identify the output filter resonant frequency as maximum of the PSD computed on the error $e[n]$. The ESR contribution is then detected, by inserting a notch at the extracted resonant frequency ($f_n = f_0$). In this section results related to the steady state identification method are then presented for $\alpha = 2$. A $70MHz$ clock system and a switching frequency of $449kHz$ are considered, the buck converter configurations are the same used during the previous section: $f_{0,buck1} = 4.9kHz$ ($L = 47\mu H, C = 22\mu F$), $f_{0,buck2} = 7.3kHz$ ($L = 47\mu H, C = 10\mu F$), $f_{0,buck3} = 10.7kHz$ ($L = 47\mu H, C = 4.7\mu F$) and possible zero contributions are $f_{z,buck1} = 14.5kHz$ ($ESR = 0.5\Omega$ for $f_{0,buck1}$), $f_{z,buck2} = 15.9$ ($ESR = 1\Omega$ for $f_{0,buck2}$) and $f_{z,buck3} = 18.8$ ($ESR = 1.8\Omega$ for $f_{0,buck3}$).
The load parameters extraction used to present all the next results has been introduced in Sec.7.3, refers to a Matlab implementation used as reference model for the PSD hardware algorithm encoding presented in Sec.9.4.3. It works on $N = 128$ samples, and considered analysis are obtained on the mirrored second half of the PSD output as described in Sec.7.3. Because the PSD output is mirrored, the relationship $\Delta f * (\mathbf{X} - N)$ have to be considered to convert a sample $\mathbf{X}$ among the $N = 128$ samples. As defined in Eq.7.4, the resolution of detectable frequencies through the PSD for $N = 128$ is $\Delta f = 1.754kHz$.

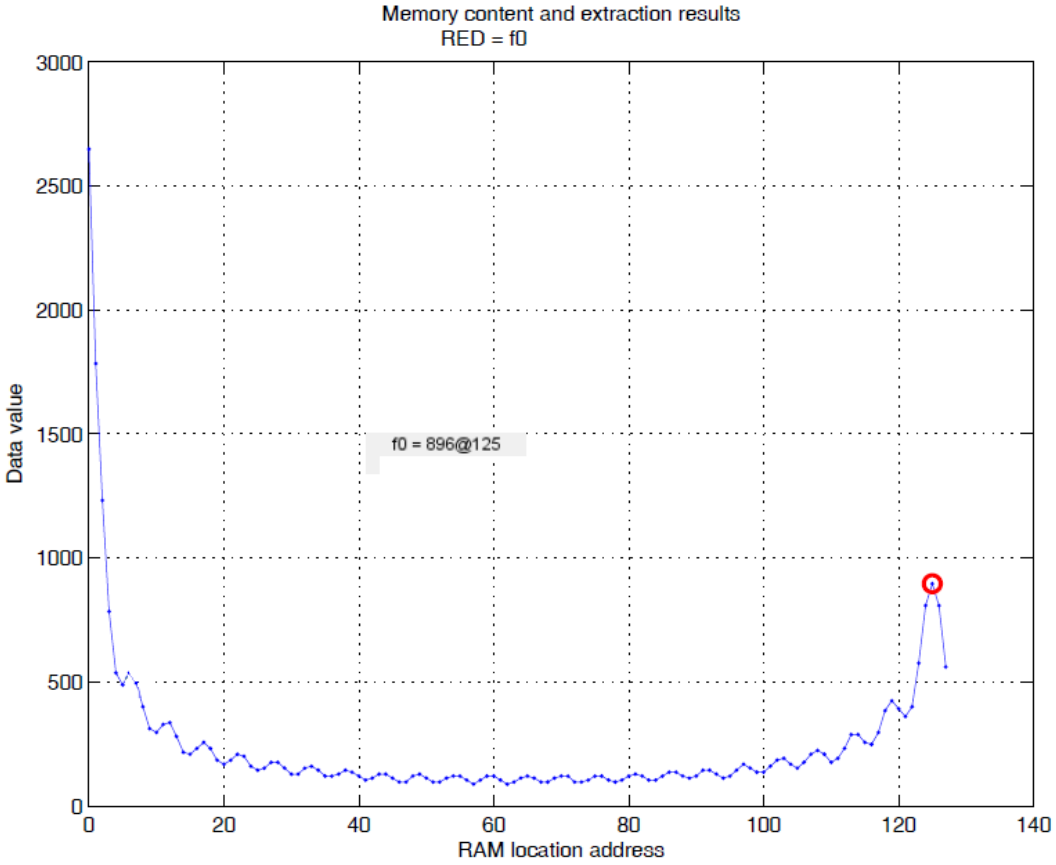

Figure 8.4: Steady state identification prototype: PSD output for $f_{0,buck1}$, $ESR = 0\Omega$.

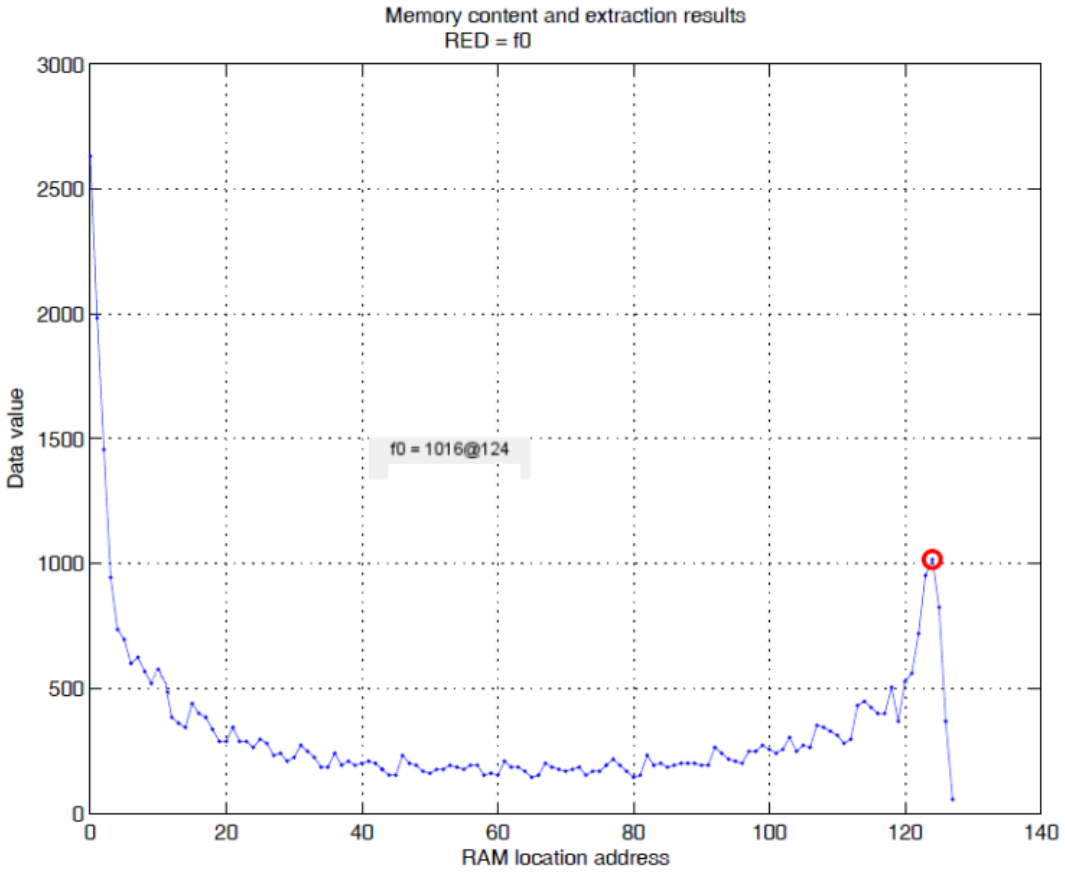

Figure 8.5: Steady state identification prototype: PSD output for $f_{0,buck2}$, $ESR = 0\Omega$.

Steady state algorithm results obtained for $\alpha = 2$ and $ESR = 0\Omega$ have been shown in Fig.8.4, Fig.8.4 and fig.8.6. Results related to the buck converter configuration $f_{0,buck1}$ are presented in Fig.8.4, the obtained resonant frequency as maximum of the PSD is $f_{01} = \Delta f * (128 - 125) = 5.26kHz$. In the results shown in Fig.8.4, the extracted resonant frequency is $f_{02} = \Delta f * (128 - 124) = 7.02kHz$, essentially equal to $f_{0,buck2}$. The result related to the third configuration $f_{0,buck3}$ has been shown in Fig.8.6, the extracted resonant is $f_{03} = \Delta f * (128 - 122) = 10.52kHz$ and it is practically equal to the desired value.
To validate the trend, a set of ten trials has been considered for the respective configurations with $ESR = 0\Omega$. Results in Fig.8.7, Fig.8.8 and Fig.8.9, have been shown multiple PSD respectively for configurations $f_{0,buck1}$, $f_{0,buck2}$ and $f_{0,buck3}$. The concentration of extracted resonant frequencies has not big variations, a trend of all the converter configurations can be distinguished. Results related to these multiple acquisitions are summarised in Fig.8.10, Fig.8.11 and Fig.8.12, where the number of occurrence and the average among the extracted resonance frequencies has been shown respectively for the configurations $f_{0,buck1}$, $f_{0,buck2}$ and $f_{0,buck3}$. The average value obtained for $f_{0,buck1}$ is $f_{01} = \Delta f * (128 - 125.5) = 4.38kHz$ (Fig.8.10), this value is very close to the desired one and even if some extraction contributions are at $f = \Delta f * (128 - 126) = 3.5kHz$ (which is just one kHz lower the desired value) the average permits to compensate oscillations around the resonant frequency and the finite resolution Δf. The trend is confirmed for multiple acquisitions related to the other two

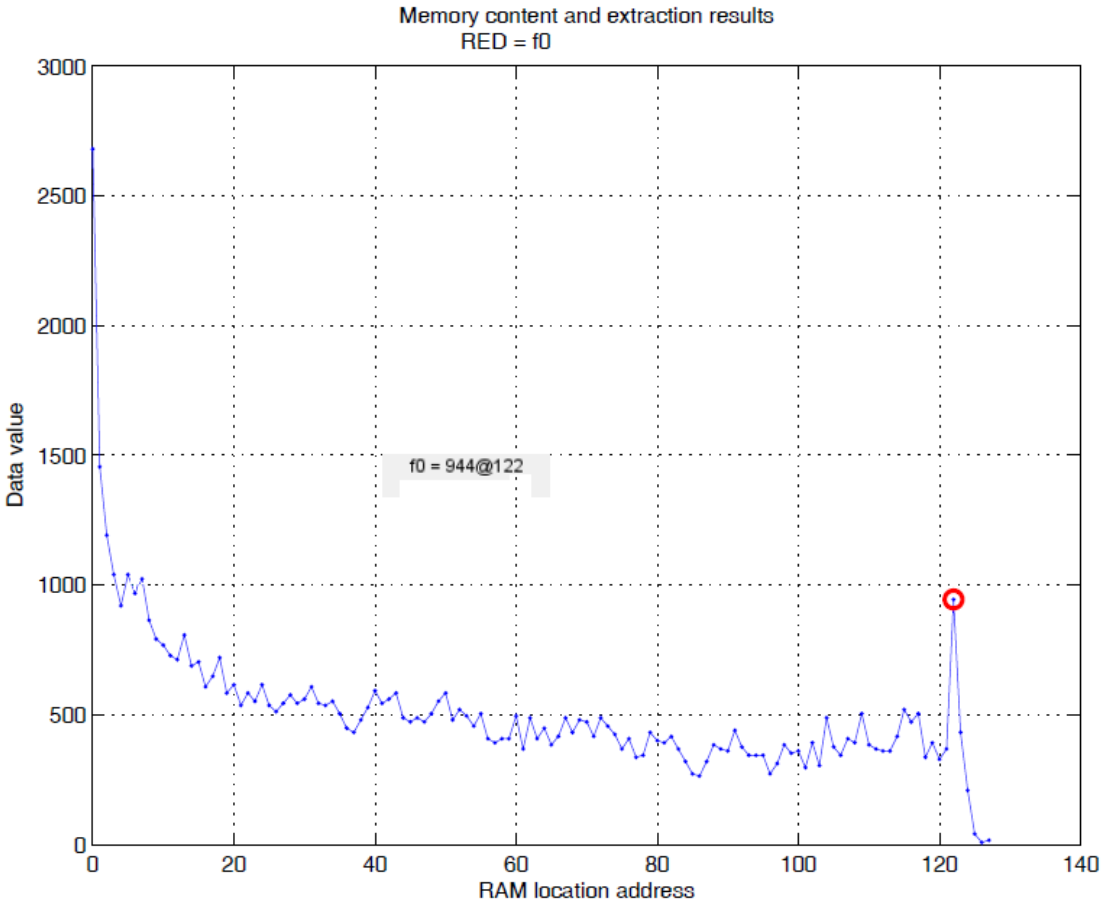

Figure 8.6: Steady state identification prototype: PSD output for $f_{0,buck3}$, $ESR = 0\Omega$.

configurations, the average value obtained for $f_{0,buck2}$ is $f_{02} = \Delta f * (128 - 124.4) = 6.32kHz$ (Fig.8.11) while for $f_{0,buck3}$ is $f_{03} = \Delta f * (128 - 122.4) = 9.82kHz$ (Fig.8.12). All the obtained values result to be very close to the desired ones and, moreover, the averaging among eight acquisitions permits to compensate the few wrong occurrence among the extracted resonant frequencies. Moreover, it can be noticed that the few wrong occurrence are however not very far from the desired values.
The obtained results for the resonant frequency identification have been presented in Tab.8.1, where also statistics among ten acquisitions have been integrated for the extracted values f_{01}, f_{02} and f_{03}.

The steady state identification method for the ESR contribution identification has been introduced in Sec.7.2. A noise shaper with a notch at the resonant frequency ($f_n = f_0$) is used to manipulate the dithering and exalt the zero at f_z in the PSD of $e[n]$. Also during the ESR identification, the quantization noise is doubled ($\alpha = 2$). In the results in Fig.8.13, the configuration $f_{0,buck1}$ and $f_{z,buck1} = 14.5kHz$ ($ESR = 0.5\Omega$) has been considered. A second peak is obtained and is detect in the left-hand side of the PSD output (Sec.7.3), the extracted contribution is at $f_z = 10\Delta f = 17.5kHz$. Similar results have been obtained in Fig.8.14 for the configuration $f_{0,buck2}$ and $f_{z,buck2} = 15.9kHz$ ($ESR = 1\Omega$), the detected zero

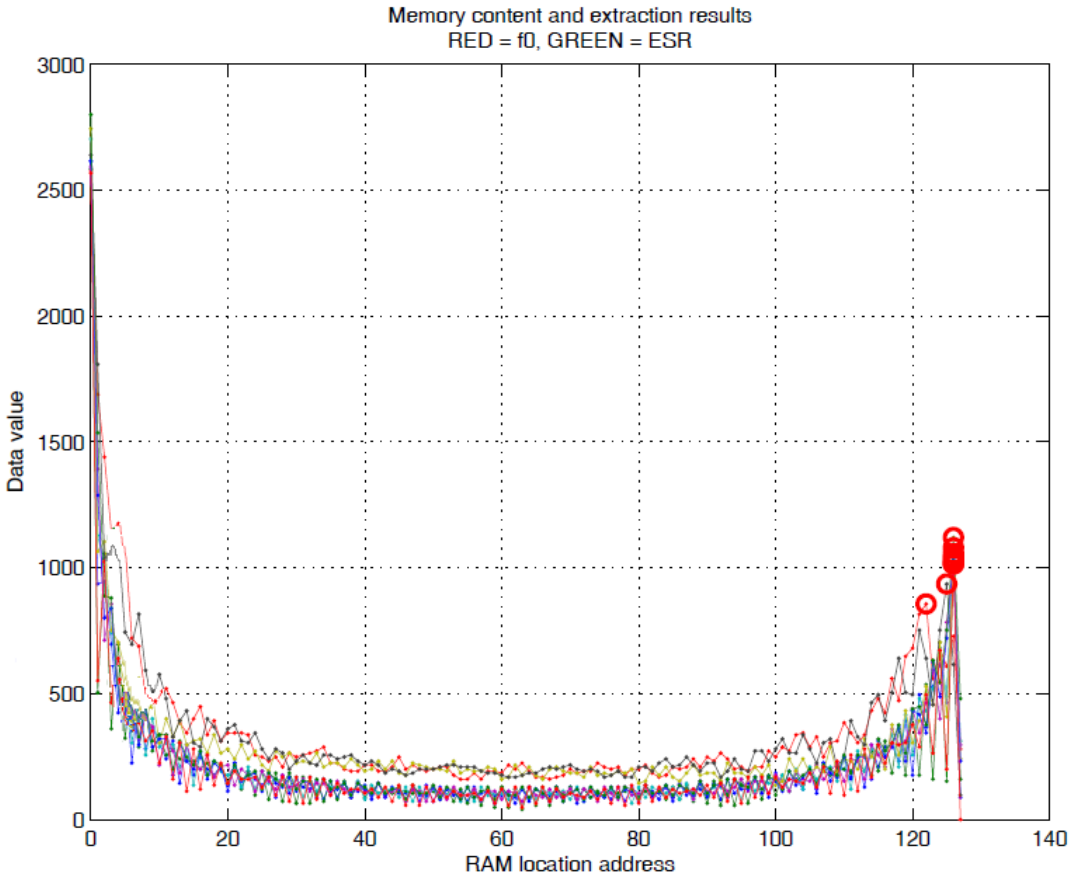

Figure 8.7: Steady state identification prototype: multiple PSD output for $f_{0,buck1}$, $ESR = 0\Omega$.

	$f_{01}[kHz]$	$f_{0,buck1}[kHz]$	$f_{02}[kHz]$	$f_{0,buck2}[kHz]$	$f_{03}[kHz]$	$f_{0,buck3}[kHz]$
f_0	5.26	4.9	7.02	7.3	10.52	10.7
statistics	4.38	-	6.32	-	9.82	-

Table 8.1: Steady state identification prototype: f_0 identification results ($ESR = 0\Omega$).

is at $f_{z2} = 9\Delta f = 15.8kHz$. As discussed in the previous section, considering $ESR = 1\Omega$ ($f_{z,buck3} = 33.8kHz$) in the buck converter $f_{0,buck3}$ inserts a zero contribution out of the bandwidth which is not easy to detect. In the results shown in Fig.8.14, this trend has been confirmed. When a zero at the resonant frequency has been inserted the ESR contribution is been detected at $f_{z3} = 15\Delta f = 26.3kHz$. A bigger ESR contribution for the third configuration has been considered in the results shown in Fig.8.16. For $ESR = 1.8\Omega$ the zero contribution is inserted at $f_{0,buck3} = 18.8kHz$ and the related second peak is extracted at $f_{z3} = 10\Delta f = 17.5kHz$. When the contribution becomes bigger and closer to the resonant frequency, a second peak is easier to be detected. Extraction results related to f_z have been summarised in Tab.8.2.

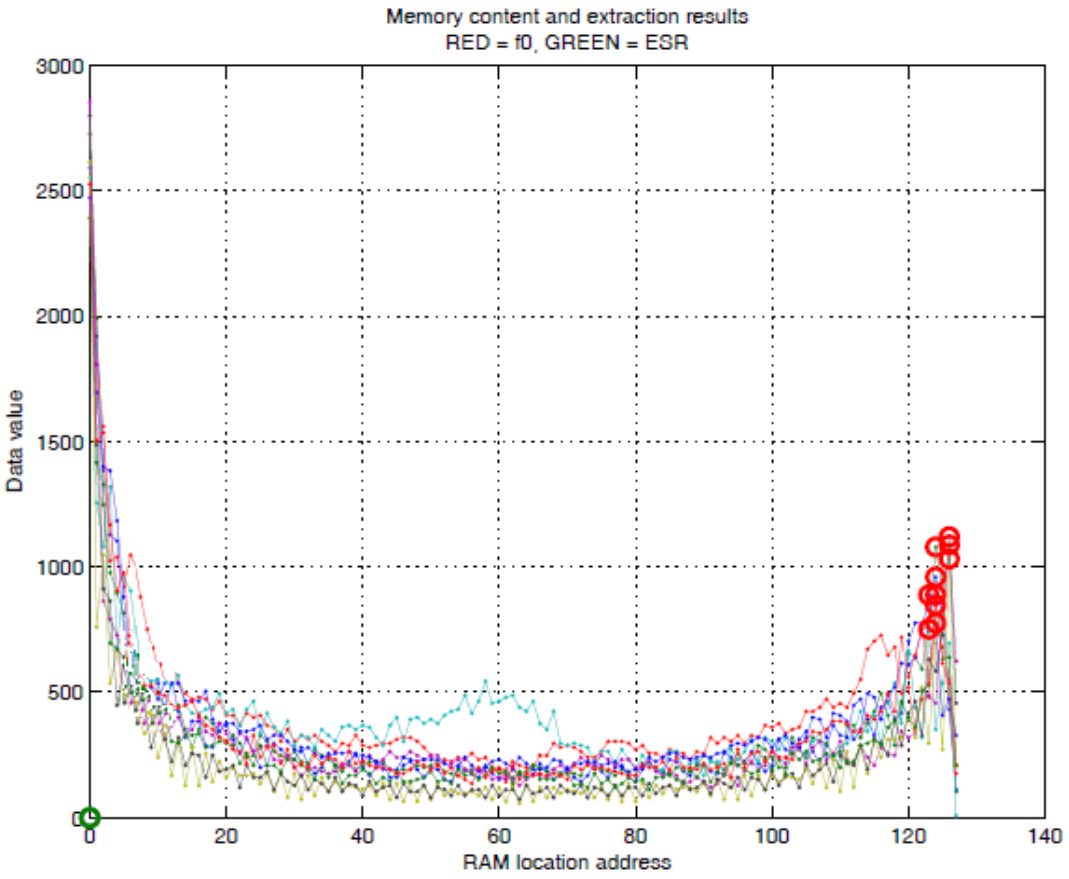

Figure 8.8: Steady state identification prototype: multiple PSD output for $f_{0,buck2}$, $ESR = 0\Omega$.

$ESR[\Omega]$	$f_{z1}[kHz]$	$f_{z,buck1}[kHz]$	$f_{z2}[kHz]$	$f_{z,buck2}[kHz]$	$f_{z3}[kHz]$	$f_{z,buck3}[kHz]$
0.5	17.5	14.5	-	-	-	-
1	-	-	15.8	15.9	26.3	33.8
1.8	-	-	-	-	17.5	18.8

Table 8.2: Steady state identification prototype: ESR identification results.

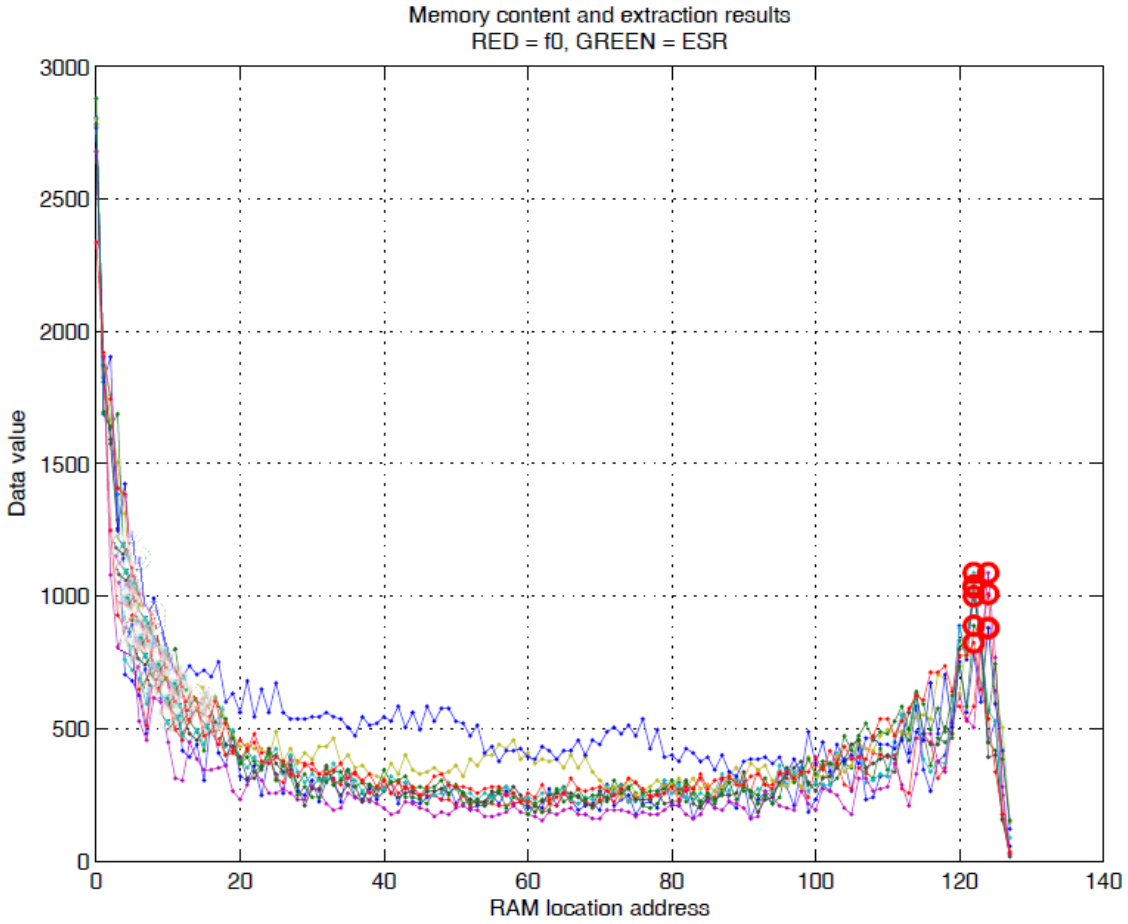

Figure 8.9: Steady state identification prototype: multiple PSD output for $f_{0,buck3}$, $ESR = 0\Omega$.

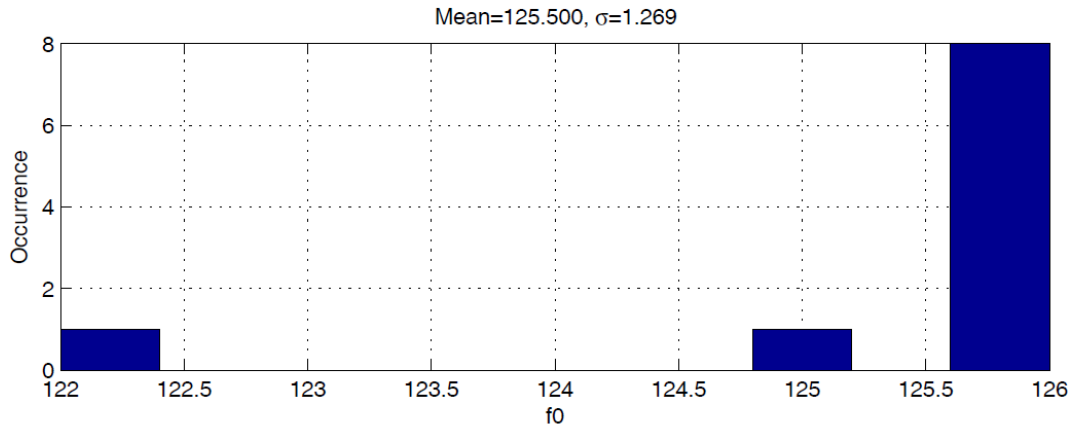

Figure 8.10: Steady state identification prototype: statistics for $f_{0,buck1}$, $ESR = 0\Omega$.

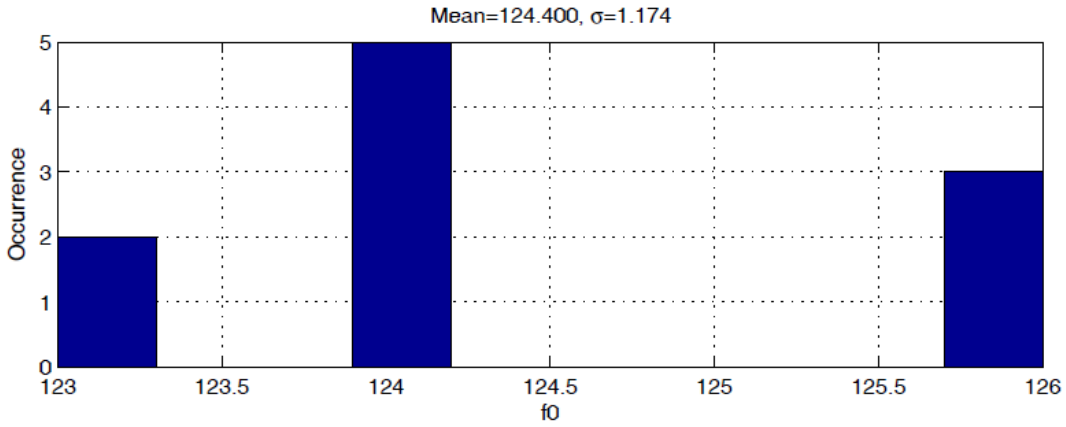

Figure 8.11: Steady state identification prototype: statistics for $f_{0,buck2}$, $ESR = 0\Omega$.

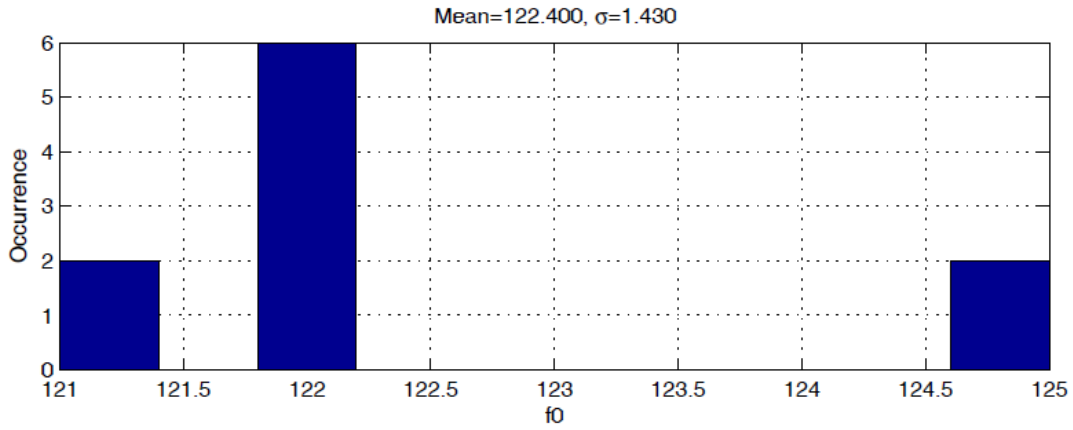

Figure 8.12: Steady state identification prototype: statistics for $f_{0,buck3}$, $ESR = 0\Omega$.

8.2 Self-tuning prototype

The Self-tuning prototype has been presented in [23], its implementation is based on the identification obtained in the previous chapter and validated in the previous section. The FPGA-TC prototype presented in Cha.5, has been integrated both with the PSD computer (Sec.9.4) and the extraction-regulation block needed for the self-tuning technique (Fig.8.1). The PSD computer implements a full-scalable VHDL-coded PSD computer, based on a novel technique for the FFT computation [24]. The extraction block has been realised as described in Sec.7.3. In the previous section results for the identification of f_0 and f_z have been presented in terms of number of sample related to the RAM address where the PSD output is store. The hardware implementation of extraction block, is the hardware representation of the algorithm used in the previous chapter to extract the parameters from the PSD output computed with the fixed-point emulation of the PSD computer. In the the PSD processing

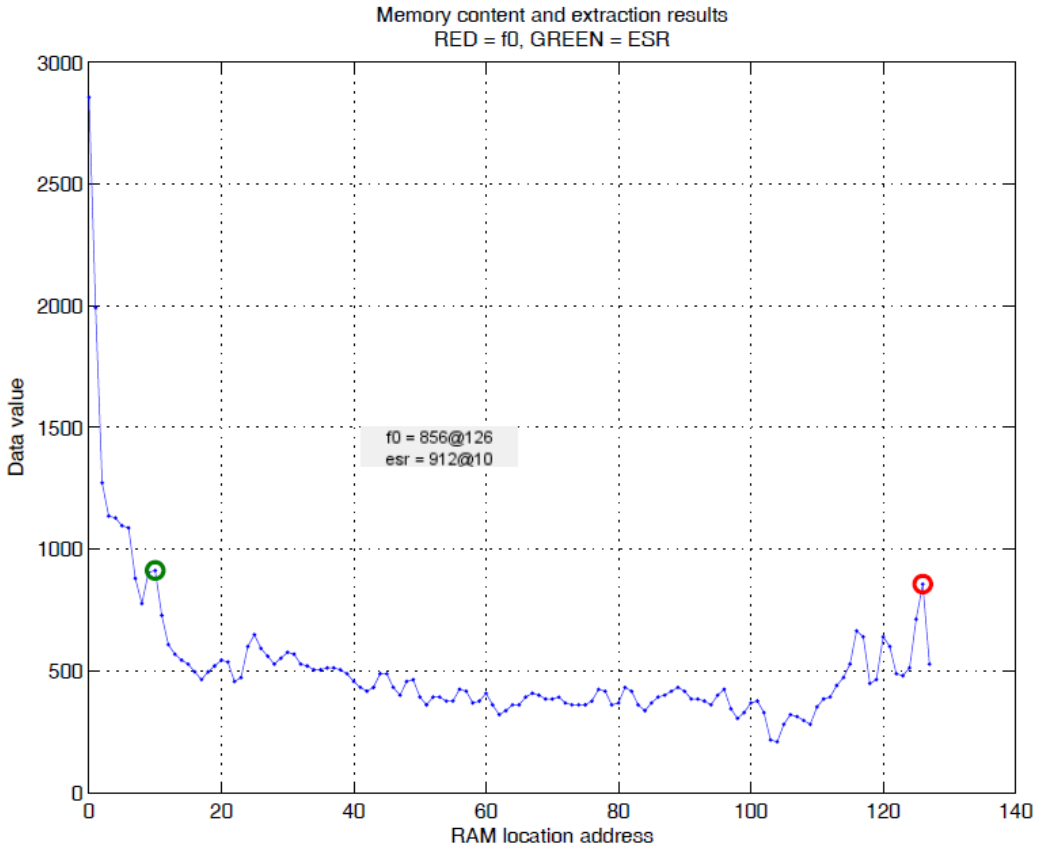

Figure 8.13: Steady state identification prototype: PSD output for $f_{0,buck1}$, $ESR = 0.5\Omega$.

results in the previous section, the extraction algorithm locates the resonant frequency on the right-hand side of the PSD output (red circle), while the ESR contribution is detected on the left-hand side of the processing (green circle). The hardware extraction simply scan a RAM where the PSD output is stored, for $N = 128$ the resonant frequency is detected as the maximum contained in the addresses interval $[N/2 : N]$ and the ESR contribution as a peak in $[0 : N/2]$. Because the output of the fixed-point computer is equal to the hardware implementation (Cha.9), in the previous section the algorithm has been validate with extractions essentially equals to the hardware PSD output.

The regulation algorithm in this prototype has been realised with a LUT (Look Up Table), the PID coefficients have to be changed according with the identification algorithm results in order to obtain the best system bandwidth and margins. In the LUT are stored the PID gains K_p,K_i and K_d, for each combination f_0 and f_z which can be obtained from the identification algorithm. The pair f_0 and f_z can be limited. Once that the switching frequency f_s is defined the resonant frequencies have to lower than $f_s/20$ while.

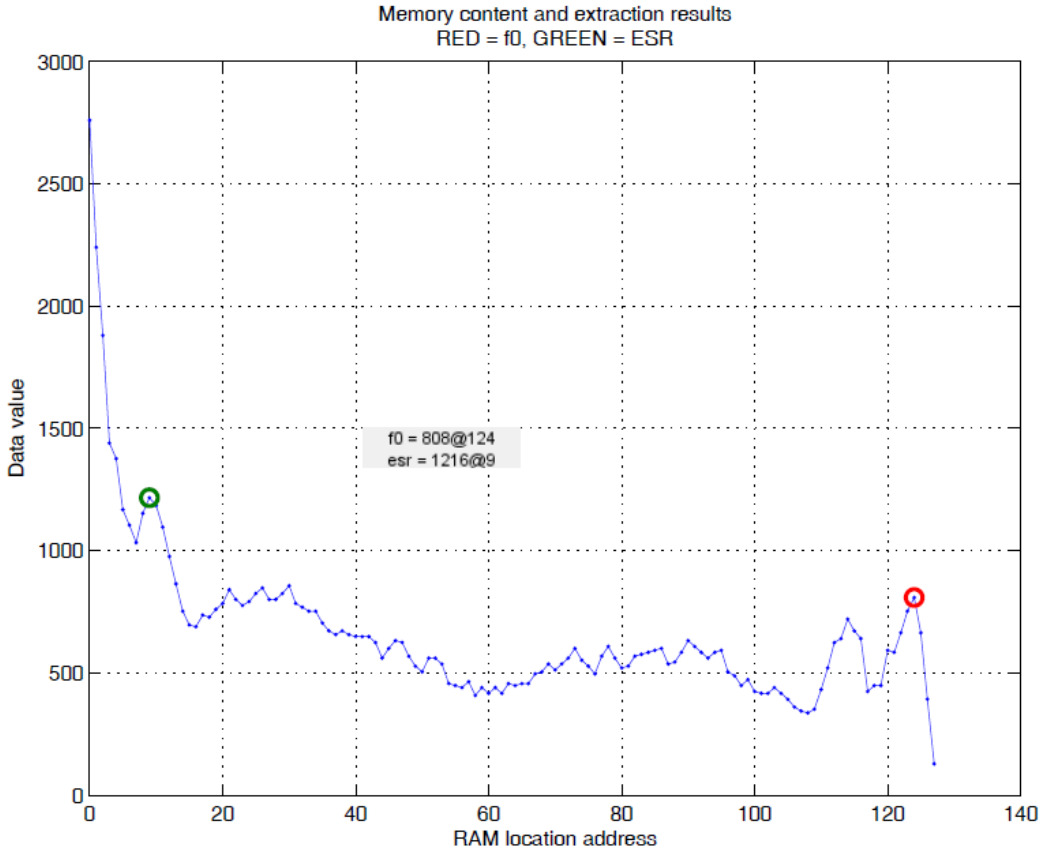

Figure 8.14: Steady state identification prototype: PSD output for $f_{0,buck2}$, $ESR = 1\Omega$.

8.2.1 Hardware description

The hardware description of the FPGA-TC based prototype of the model shown in Fig.8.1 has been presented in Fig.8.17, the digital control prototype described in Cha.5, and used in the previous section, has been integrated in the FPGA with the PSD computation, the RAM, the Extraction and the Regulation block. The external loop I/O configuration for the closed loop communication between the TC and the FPGA is the same presented in Cha.5.
A finite state machine (**FSM_proc**) and a counter has been integrated in the FPGA to define the evolution of the self-tuning model. Input signals to the **FSM_proc** are:

- *proc_en_i*. This signal is driven by the user through a push button (see UCF files in Sec.5.1.3) and is used to activate the algorithm sequence.
- *activate_compensation_i*. This signal is driven by the user through a push button, it activates the PID gains changing after that the load parameters identification have been completed.
- *esr_2be_detected_i*. Used to distinguish the extraction algorithm which has to be activate, with this signal the extraction of the resonant frequency can be distinguished from the extraction of f_z.

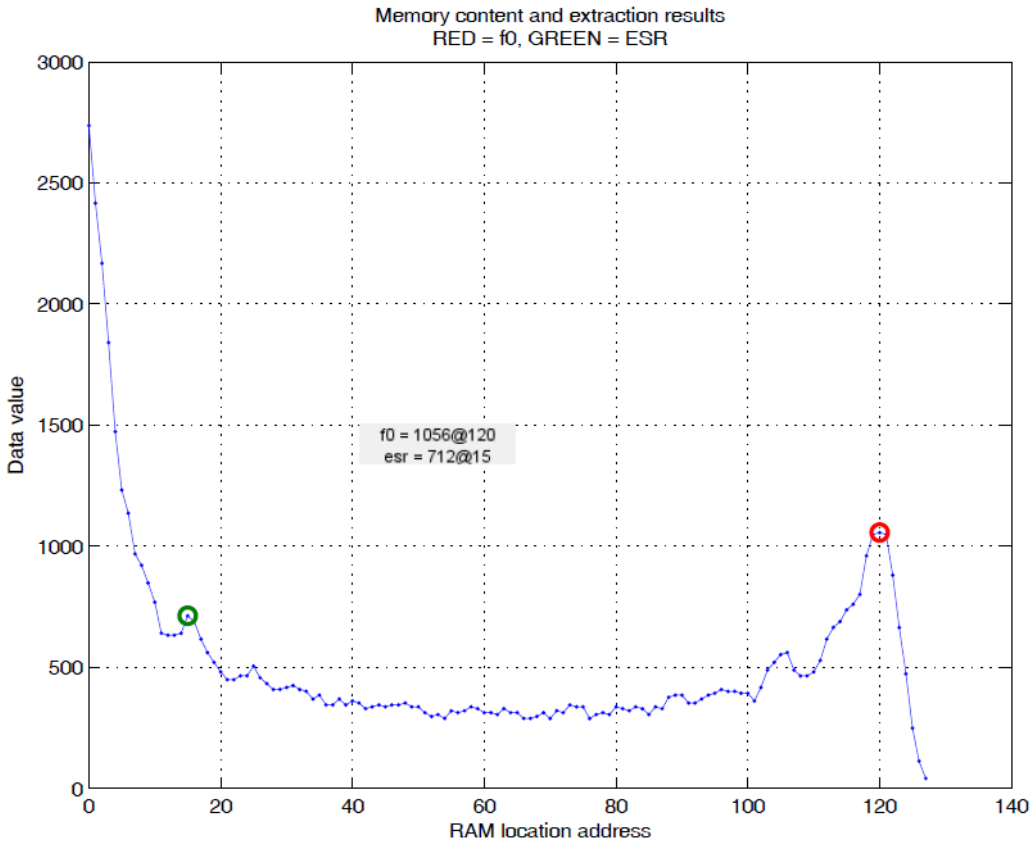

Figure 8.15: Steady state identification prototype: PSD output for $f_{0,buck3}$, $ESR = 1\Omega$.

- *adc_sample_i.* Is the ADC clock sample and it is used to give to synchronize the PSD computer with the ADC sample.
- *startup_finished_i.* This signal is used to activate the FSM, because the SI identification occurs during the steady state the algorithm is activated only when this condition is reached.
- *over_i* and *nready_i.* These are signals used to understand when the PSD computer ends the processing.
- *counter_i.* Is the output of the counter block and it used at the beginning of the processing. When the algorithm is activated the dithering amplification is enable for some switching cycles ($10T_s$ for instance) before starting the acquisition of samples to process. During this period the quantization noise is amplified.
- *abs_error_i.* The FSM receives the ADC output error in terms of absolute values, when the algorithm is enabled this 4 bit signal is output to the PSD computer block.
- *pid_coeff_ready_i.* This signal is high when the new PID gains are ready to be output from the regulation block to the PID.

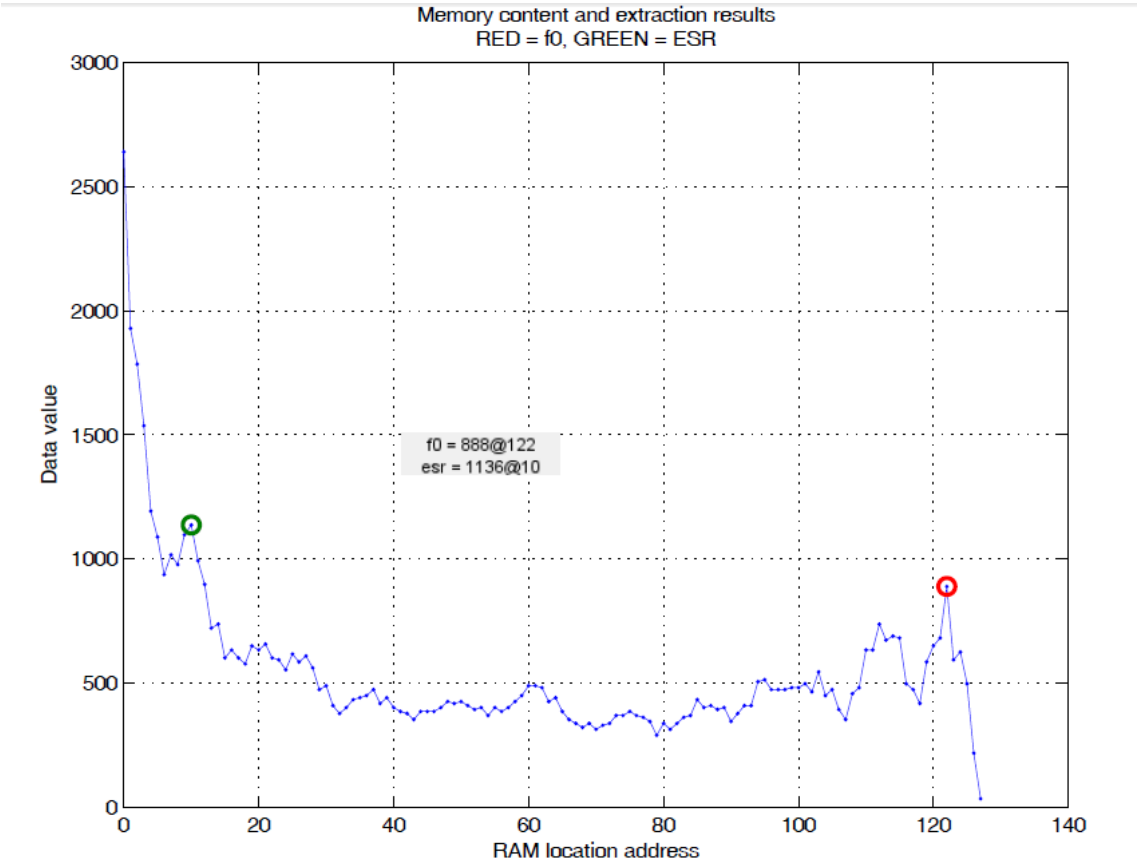

Figure 8.16: Steady state identification prototype: PSD output for $f_{0,buck3}$, $ESR = 1.8\Omega$.

- *extracted_data_ready_i.* When the extraction algorithm ends, this signal is high and the FSM is designed to decide either to activate the regulation or to wait for a new identification.

Output signals to the **FSM_proc** are:

- *res_quant_ds_o.* As described in the previous section, this signal is used to amplify the $\Delta\Sigma$ quantization noise when the identification algorithm is activated.
- *ntf_mod_enable_o.* When the ESR contribution exists, this signal enables the notch insertion at the extracted resonant frequency value.
- *processing_en_o.* Is used to activate the PSD computation.
- *processing_en_counter_o.* This signal enables the counter block at the beginning of the algorithm.
- *abs_error_o.* Is used to outputs the absolute value of $e[n]$ to the PSD computer when the processing is enabled.
- *regulation_en_o.* It enables the regulation block for the best PID gains computation.

- *avg_en_o*. Is used to enable the averaging over eight successive extractions. Averaging is done both on the extracted f_0 and f_z.
- *extraction_en_o*. When the PSD computation ends, this signal enables the extraction algorithm for the resonant frequency identification.
- *extraction_esr_en_o*. This signal is high when the identification algorithm have to extract both the ESR contribution and the resonant frequency.

The finite state machine shown in Fig.8.18 represents the control used to implement the SI self-tuning algorithm. The steady state identification conclusions addressed in the previous section and in the previous chapter had brought to a two step process, when also the ESR contribution have to be considered. Because the SI techniques work during steady state, the FSM moves from the **IDLE** state when the signal *start_up_finished_i* generated from the DPWM is high. The next state is the state **WAIT PROC**. In this state the FSM is waiting for starting the algorithm, all the outputs are equal to zero as for the **IDLE** state. When the signal *proc_en_i* is high the FSM enables the counter block (processing_en_counter_o is high) and amplifies the dithering introduced through the modulator (*res_quant_ds_o* assumes the value $\alpha-1$, to have $\alpha=2$ is *res_quant_ds_o*=01 as described in Fig.8.3). The signal *proc_en_i* is driven by the user with a push button on the FPGA, through the UCF file (see Sec.5.1.3) is possible to map the signal on a push button (LOC = G17 on SW7 pushbutton). Once that the counter is enabled, one clock cycle later the FSM is in the state **INSERT NOISE** where the dithering amplification continues to be inserted. The FSM remains in this state for some switching cycles ($10T_s$ in this example) before starting the processing. This state has been inserted to delay the start of the PSD computation respect to the noise amplification. The outputs in this state are the same of the state **WAIT PROC** (processing_en_counter_o is high and *res_quant_ds_o*=01). After ten switching cycles the FSM start the **PROCESSING**, in this state the counter is disabled (processing_en_counter_o=0), the dithering amplification is enabled for the entire processing and a classical third order noise shaper is used (*ntf_mod_enable_o*=0). The FSM receives the ADC output error (*abs_error_i*) and the clock of the ADC (*adc_sample_i*), during the **PROCESSING** the error $e[n]$ is output to the PSD computer. The FSM moves from the **PROCESSING** state when the PSD has been computed, the latency related to this state depends mainly on the number of point N considered for the computation and the number of processing elements (PEs) used for the FFT computation (cha:9.4). When the PSD computer ends the processing both signals *over_i* and *nready_i* go high, the FSM moves to the state **EXTRAC** where amplification of the dithering is disabled (*res_quant_ds_o*=00). In this state the extraction algorithm is enabled (*extraction_en_o* is high) to extract the resonant frequency (f_0) as maximum in the PSD output. When the extraction is enabled the algorithm scan the RAM through the signal *read_address_o* and read its content, the incoming data (*PSD_data_o*) are saved in the extraction block and only the maximum is saved at the end of the algorithm (Fig.8.17). In this phase only the resonant frequency is searched and only RAM addresses from $N/2$ to N are considered. When the extraction ends (*extracted_data_ready_i* is high) the FSM moves to **AVG ENABLE** or **NTF MOD** state receptively if the ESR has not to be or vice versa has to be detected. The distinction between this two signals is done considering the signal *esr_2be_detected_i*, which is set at the beginning from the user through a switch in the FPGA (LOC = K21 on SW1 DIP switch) mapped through the UCF file. If the signal *esr_2be_detected_i* is high the ESR contribution have to be detected and the next state is **NTF MOD**. In this state the zero in noise

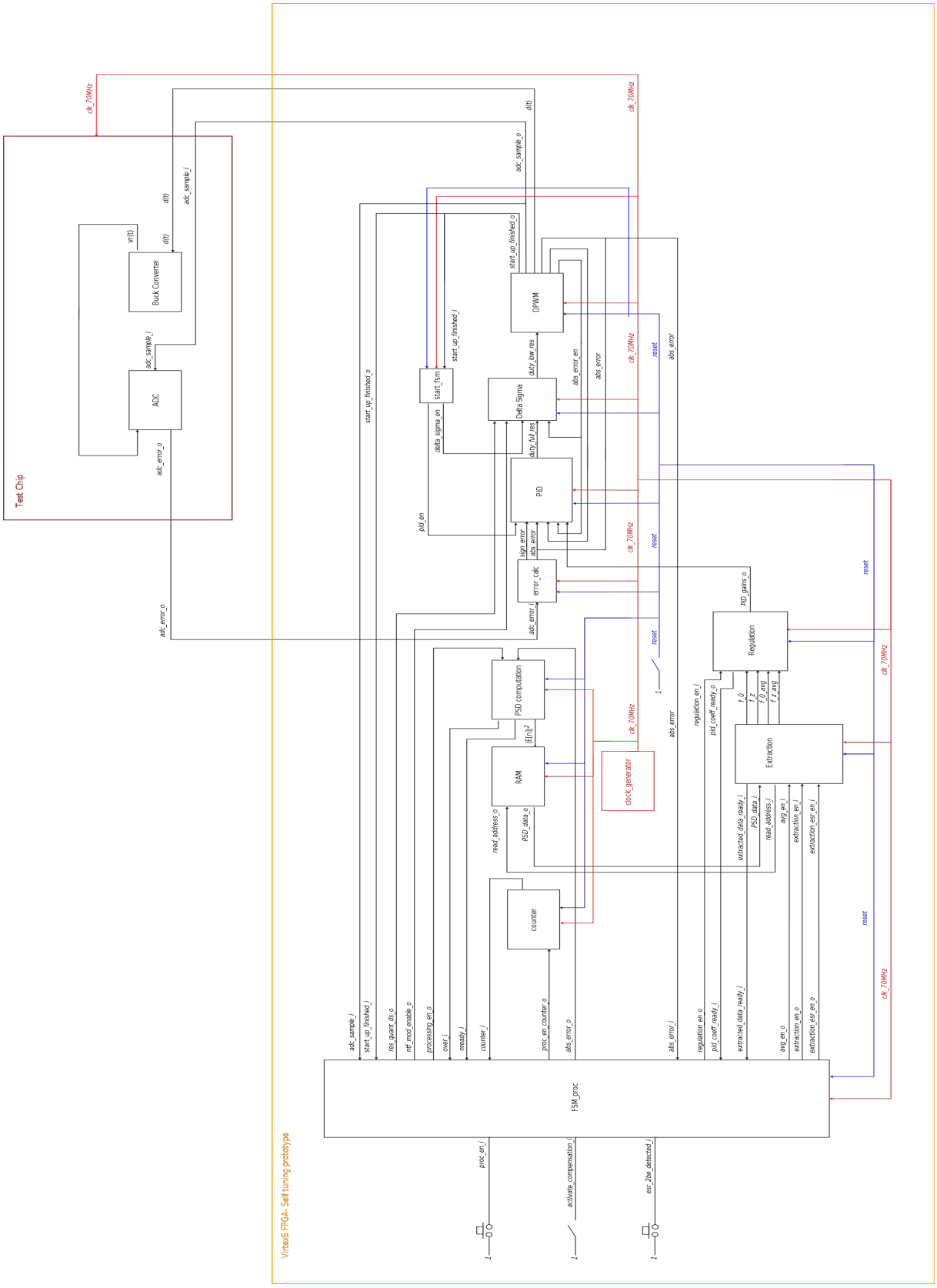

Figure 8.17: Self-tuning prototype: hardware description.

shaper ($f_n = f_0$) is inserted at the resonant frequency extracted when the FSM was in **EXTRAC**, hence, the output *ntf_mod_enable_o* is high and the dithering amplification is activated (*res_quant_ds_o*=01). Once that the NTF has been modified the FSM moves to the state **PROC ESR**, which is equal to state **PROC** but with the difference that the entire processing have to be computed considering a modified noise shaper. Once that the PSD computation ends (*over_i* and *nready_i* are both high, the next state is **EXTRAC ESR** and the dithering amplification is disabled (*res_quant_ds_o*=00). In this state the extraction procedure is enabled through *extraction_esr_en_o*=1, the extraction block read the RAM content from 0 to $N/2$ to search the ESR contribution and from $N/2$ to N to detect the resonant frequency. However, when the extraction block ends, the signal *extracted_data_ready_i* is high and the FSM moves in the state **AVG ENABLE**. In this state all the outputs except *avg_en_o* are null and the averaging function is enabled. This function simply counts and saves the last eight extractions to compute the average among them. As for the results presented in the previous section, the averaging permits to compensate some wrong extraction. One clock cycle later the FSM in the **WAIT** state where the user can decide for a new identification (*proc_en_i*=1) or to change the PID coefficients (*activate_compensation_i*=1) according with the extracted f_0 and/or f_z. The signal *activate_compensation_i* has been mapped trough the UCF file on a push button (LOC = A18 on SW6 push button). If the processing is activated from the user the next state is **COUNT ENABLE** and the algorithm repeats, otherwise, if the user press on *activate_compensation_i* the next state is **REGULATION**. In this state the signal *regulation_en_o* is high and new PID coefficients are output from the PID compensator. In this way the system reaches the best compensation in terms of bandwidth and margins, a robust system is obtained when moving from a compensator studied for a wide range of loads to the best one defined for the identified load. When the regulation block ends the computation of the new coefficients (in this prototype has been done simply with a LUT), the block outputs the signal *pid_coeff_ready_i*=1 and the FSM moves to the state **WAIT PROC** to repeat the sequence when the user likes.
The **IDLE** state can be reached from every state thorough the reset condition (reset signal on LOC= H10 push button), this transitions are omitted in Fig.8.18.

8.2.2 Results

In Fig.8.19 is shown the Virtex6 FPGA with the considered IO mapping done for the self-tuning prototype through the UCF file, the external loop FPGA-TC prototype configuration presented in Fig.5.11 has been integrated with signal to drive the FSM needed for the self-tuning algorithm. The Test Chip external loop configuration is the same as presented in Sec.5.2, the prototype global setting is shown in Fig.8.20.

Let's consider the system configuration as in the entire dissertation. A $70MHz$ clock system and a switching frequency of $449kHz$ are considered, while, the buck converter configurations are the same used during the steady state method discussion: $f_{0,buck1} = 4.9kHz$ ($L = 47\mu H, C = 22\mu F$), $f_{0,buck2} = 7.3kHz$ ($L = 47\mu H, C = 10\mu F$, $ESR = 1\Omega$), $f_{0,buck3} = 10.7kHz$ ($L = 47\mu H, C = 4.7\mu F$) and possible zero contributions are $f_{z,buck1} = 14.5kHz$ for $f_{0,buck1}$ ($ESR = 0.5\Omega$), $f_{z,buck2} = 15.9kHz$ for $f_{0,buck2}$ ($ESR = 1\Omega$), $f_{z,buck3} = 18.8kHz$ and $f_{z,buck3} = 33.8kHz$ respectively with $ESR = 1.8\Omega$ and $ESR = 1\Omega$ for $f_{0,buck3}$. The self-tuning prototype results are presented through a ChipScope interface, where it is possible to read the extracted parameters (f_0 and f_z) both in terms of extracted sample in the PSD output and in terms of frequency. Moreover, the PID gains have been shown. When the regulation is activated from

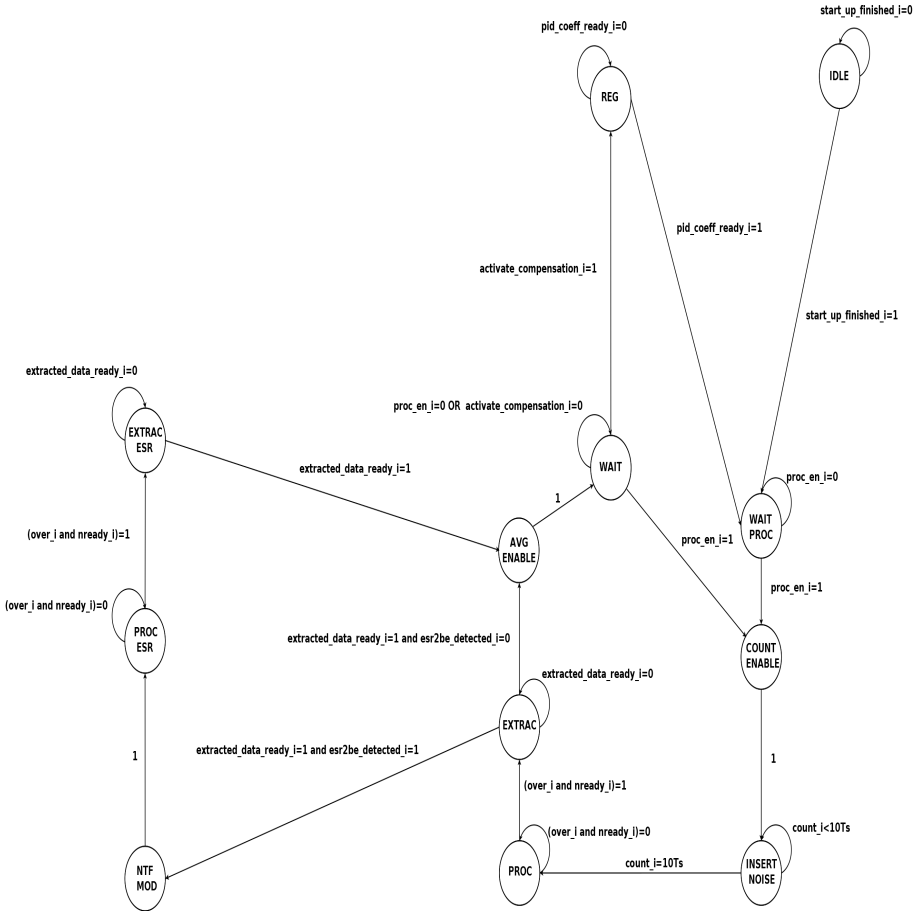

Figure 8.18: Selt tunign algorithm: FSM (reset transitions omitted).

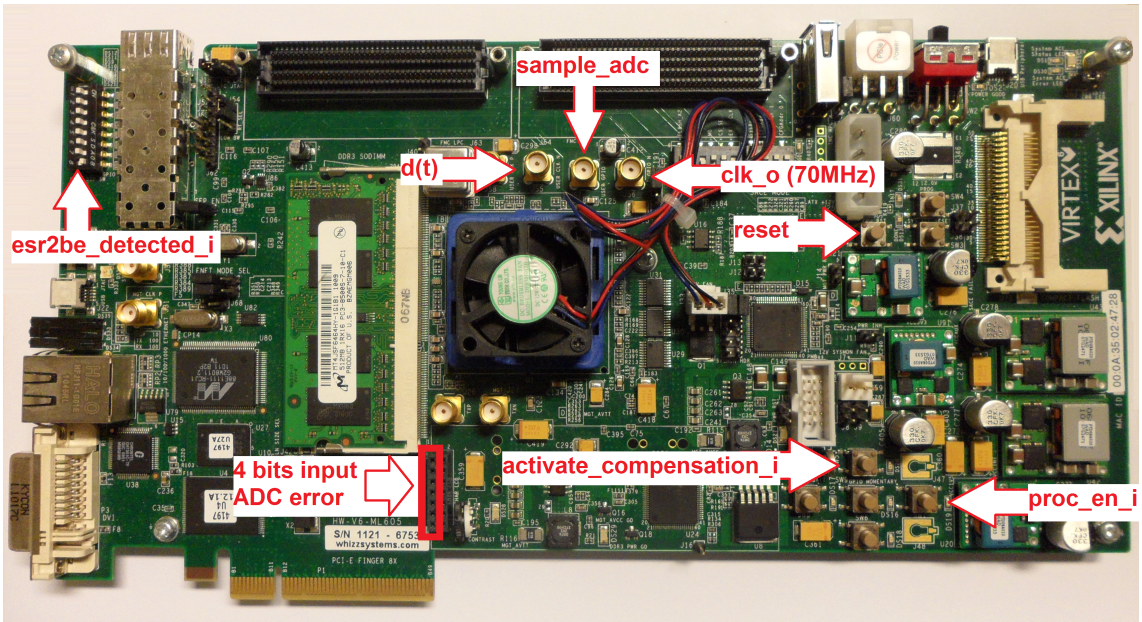

Figure 8.19: Self-tuning prototype: FPGA I/O configuration.

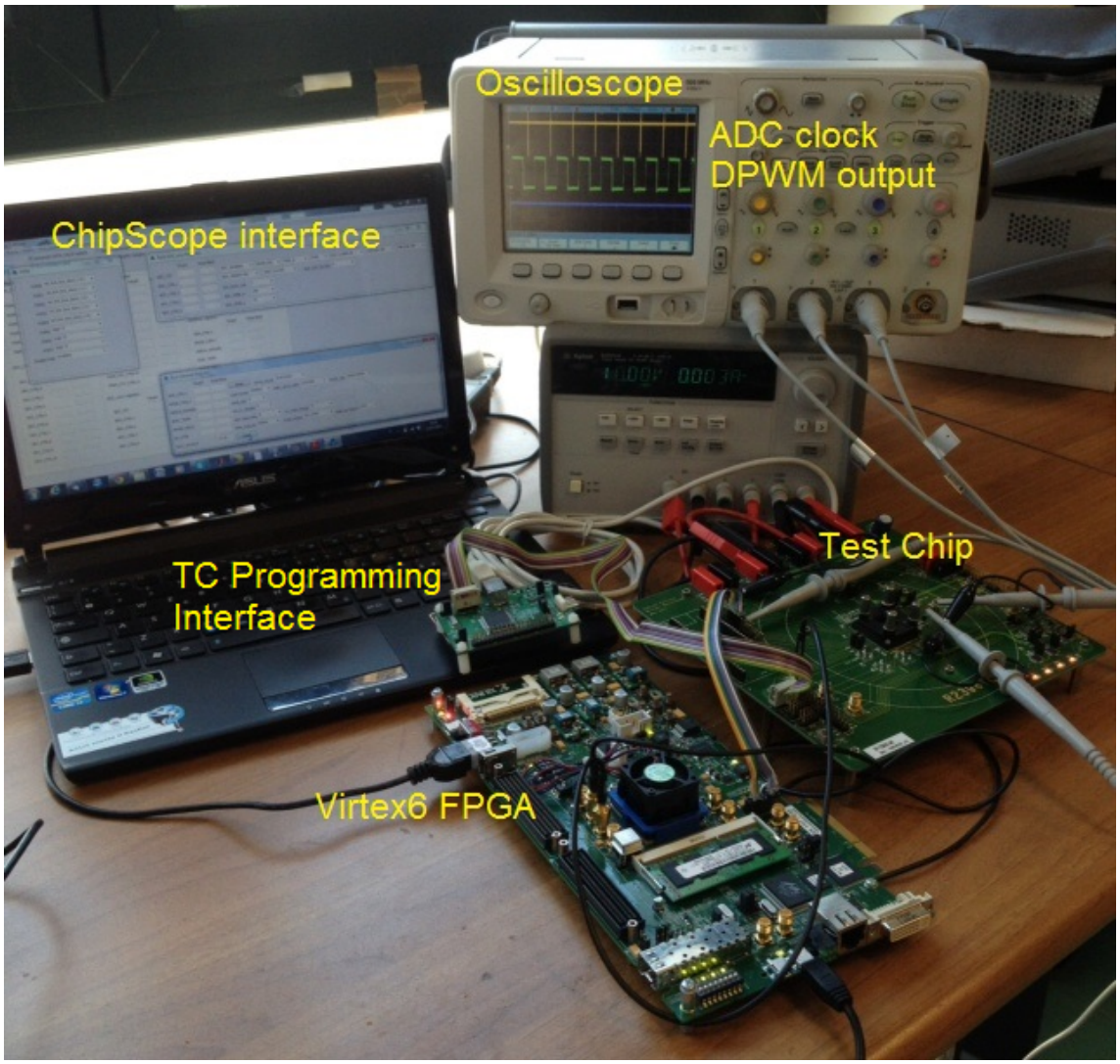

Figure 8.20: Self-tuning prototype: DEMO configuration[23].

the user, these values are computed considering the extracted parameters in order to set the best configuration for the identified load. The PID hardware design, as the entire digital control loop, is the same described in Cha.4 for the off-line controller. Only the $\Delta\Sigma$ has been modified (dithering amplification and notch insertion in the noise shaper) to implement the identification algorithm.

The prototype identification results, for the considered buck converter configurations when $ESR = 0\Omega$ have been presented in Fig.8.21, Fig.8.22 and Fig.8.23. Results shown in Fig.8.21 refer to the buck converter configuration $f_{0,buck1}$. The last identified resonant frequency is of $3.5kHz$ and the averaged value over eight acquisitions is $f_{01,avg} = 5.3kHz$ very close to the desired value $f_{0,buck1}$. The configuration $f_{0,buck2}$ has been considered to show results in Fig.8.22, in this case the averaged resonant frequency results to be at $f_{02,avg} = 7kHz$ essentially equal to the desired value. Results referring to the third case $f_{0,buck3}$ has been shown in Fig.8.23, in this case the averaged value results to be $f_{03,avg} = 12.3kHz$ not far from $f_{0,buck3}$.

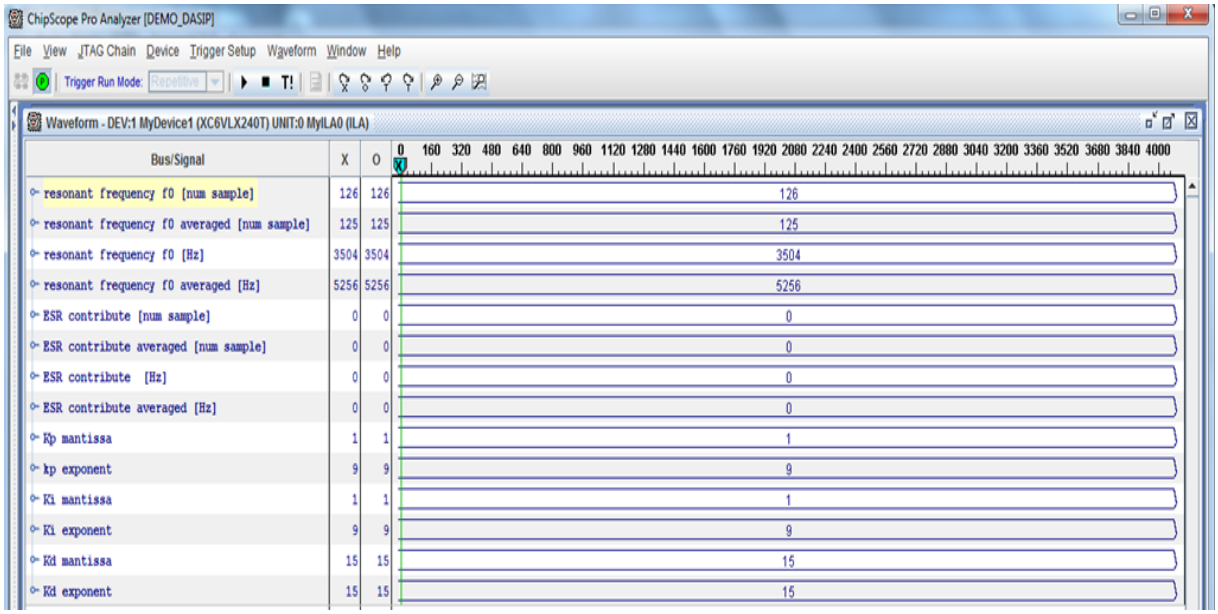

Figure 8.21: Self-tuning prototype: extracted load parameters ($f_{0,buck1}$, $ESR = 0\Omega$).

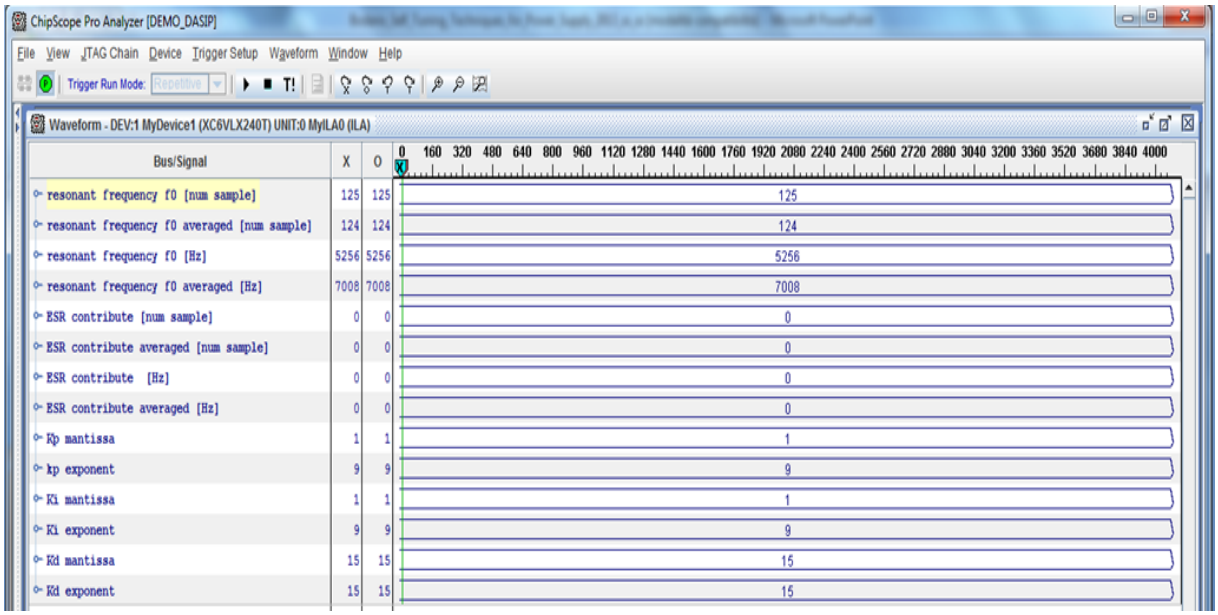

Figure 8.22: Self-tuning prototype: extracted load parameters ($f_{0,buck2}$, $ESR = 0\Omega$).

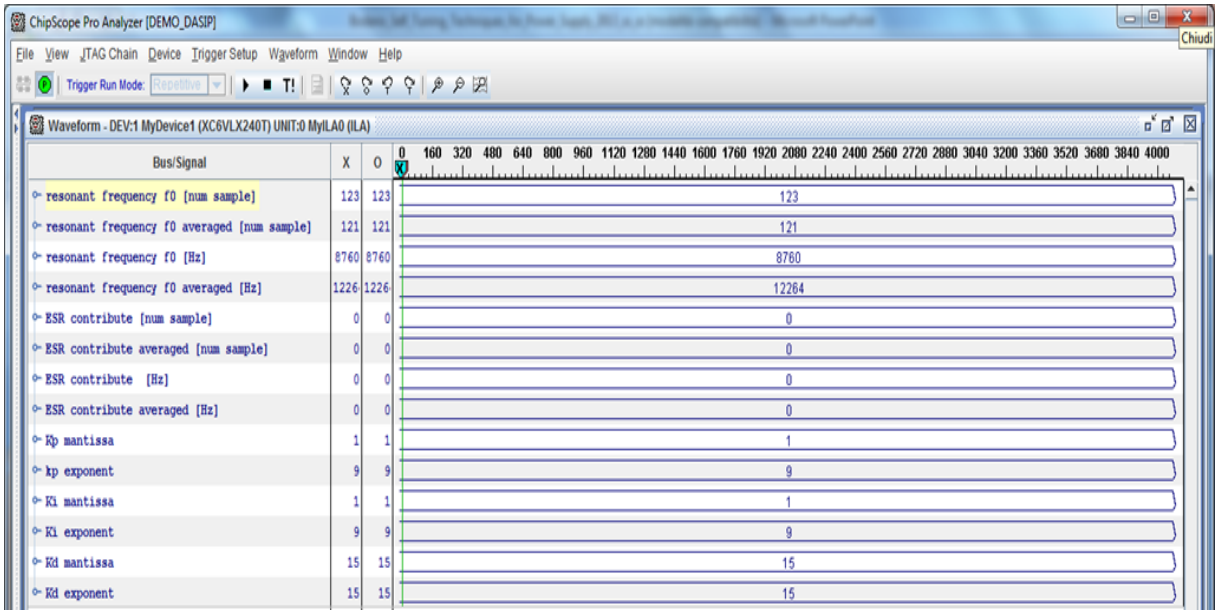

Figure 8.23: Self-tuning prototype: extracted load parameters ($f_{0,buck3}$,$ESR = 0\Omega$).

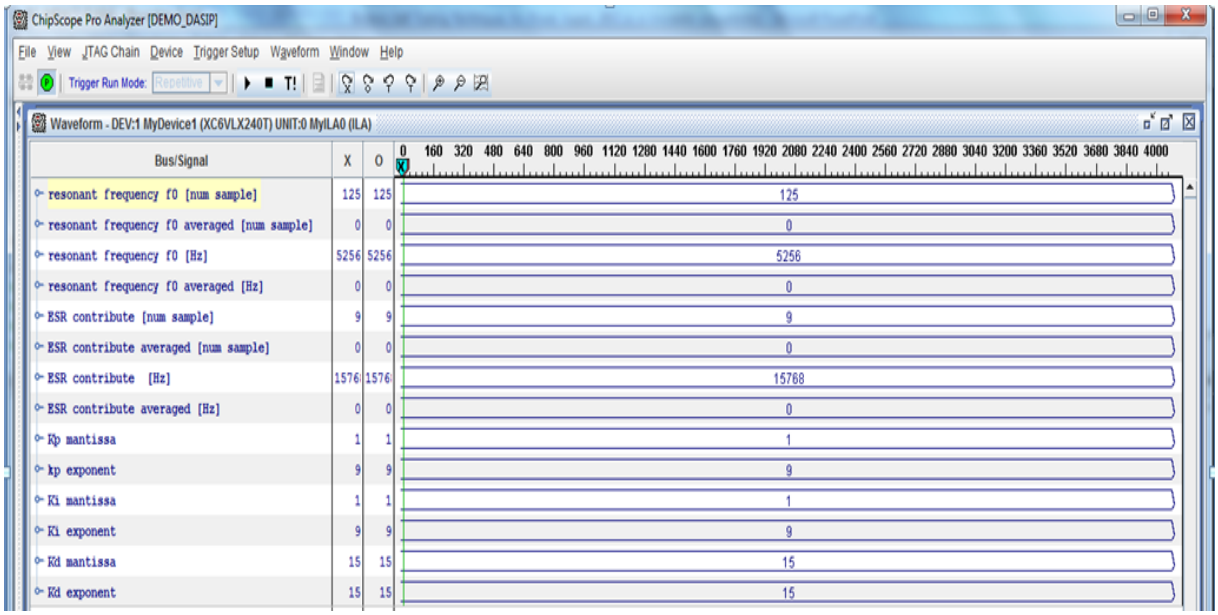

Figure 8.24: Self-tuning prototype: extracted load parameters ($f_{0,buck1}$,$ESR = 0.5\Omega$).

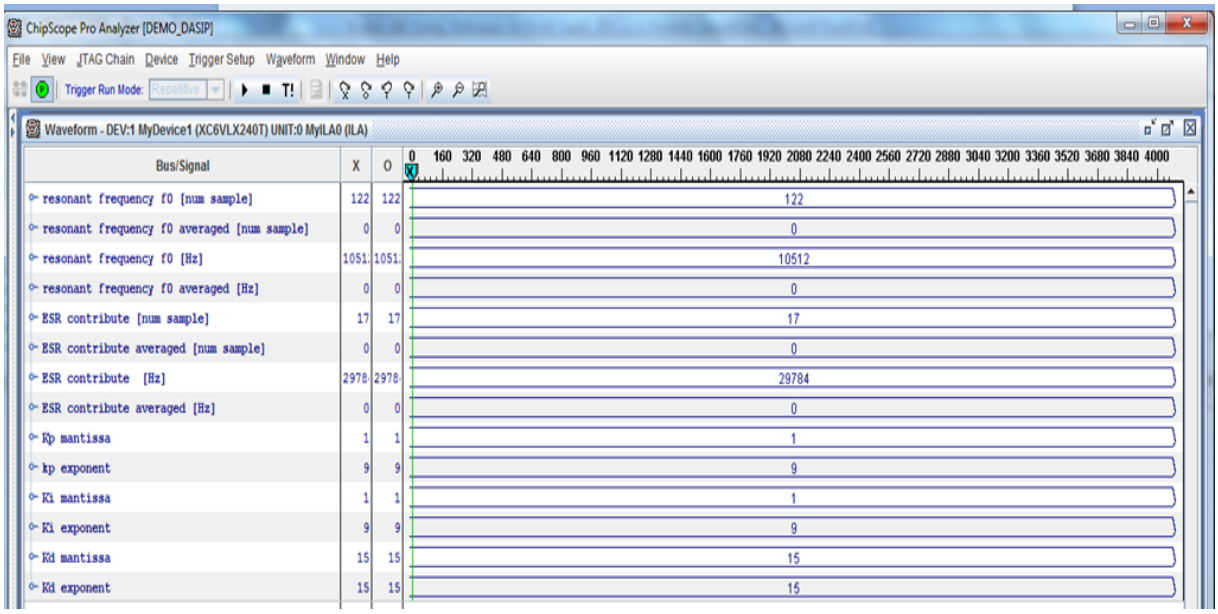

Figure 8.25: Self-tuning prototype: extracted load parameters ($f_{0,buck3}$,$ESR = 1\Omega$).

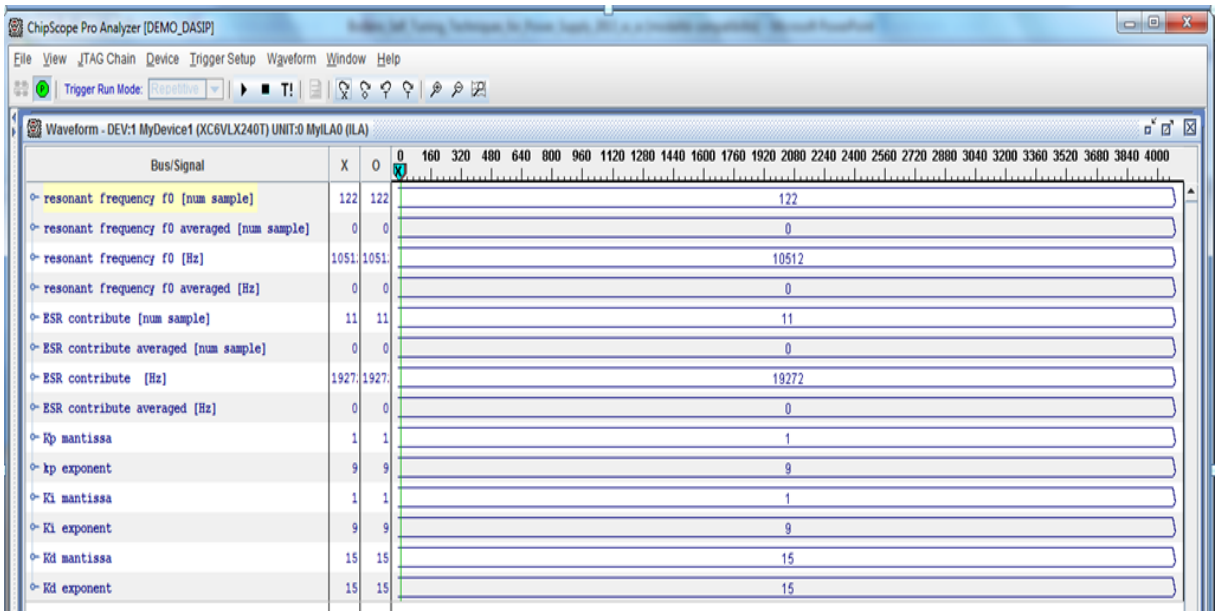

Figure 8.26: Self-tuning prototype: extracted load parameters ($f_{0,buck3}$,$ESR = 1.8\Omega$).

The ESR contribution identification has been considered in results shown in Fig.8.24, Fig.8.25 and Fig.8.26, respectively for the configurations $f_{z,buck1} = 14.5kHz$, $f_{z,buck3} = 33.8kHz$ ($ESR = 1\Omega$ for $f_{0,buck3}$) and $f_{z,buck3} = 18.8kHz$ ($ESR = 1.8\Omega$ for $f_{0,buck3}$). These results have been obtained, considering a notch at the extracted resonant frequency ($f_n = f_0$) in the third order modulator during the dithering amplification (see the processing FSM in Fig.8.18). In results shown in Fig.8.24, the extracted resonant frequency is $f_{01} = 5.3kHz$ and the ESR contribution is $f_{z1} = 15.8kHz$. Considering $ESR = 1\Omega$ in $f_{0,buck3}$, the identified zero is at $f_{z3} = 29.8kHz$ when a notch at $f_n = f_0 = 10.5kHz$ is inserted in the third order noise shaper. When the ESR contribution is approaching the bandwidth is confirmed that its contribution is detectable. In results presented in Fig.8.26, a larger contribution is considered in the configuration $f_{z,buck3} = 18.8kHz$ ($ESR = 1.8\Omega$), with a notch effect at $f_n = f_{03} = 10.5kHz$ the extracted zero contribution $f_{z3} = 19.3kHz$ is very precise.

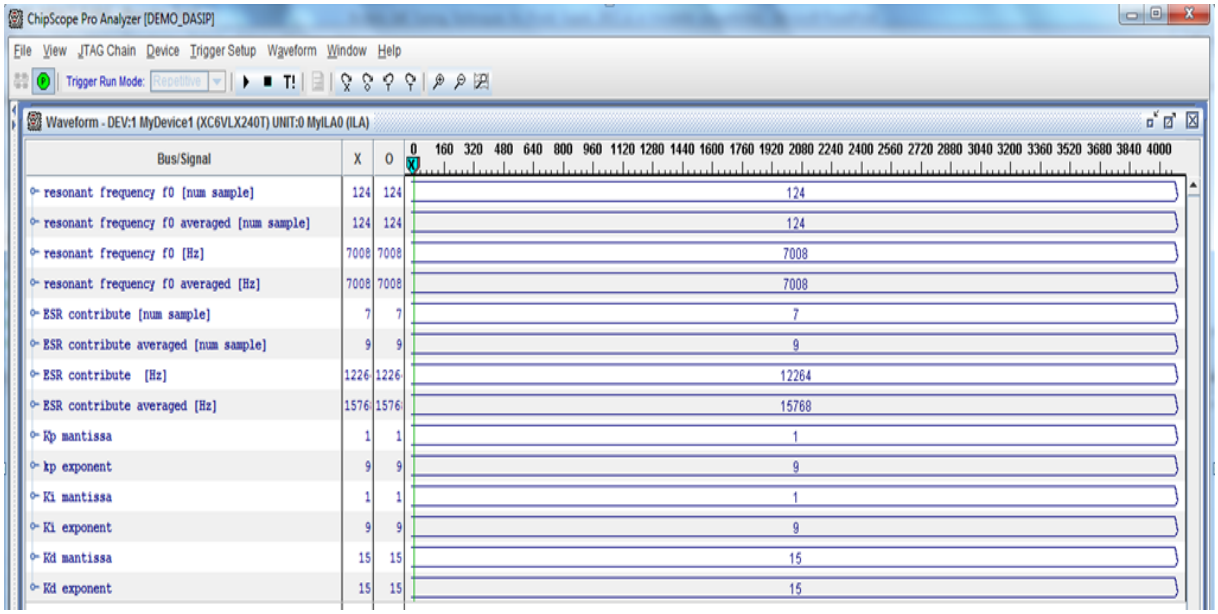

Figure 8.27: Self-tuning prototype: averaged extracted load parameters ($f_{0,buck2}$,$ESR = 1\Omega$).

The averaged results after eight acquisitions, have been shown in Fig.8.27 and Fig.8.28, respectively for $f_{z,buck2} = 15.9kHz$ ($f_{0,buck2} = 7.3kHz$ and $ESR = 1\Omega$) and $f_{z,buck3} = 18.8kHz$ ($f_{0,buck3} = 10.7kHz$ and $ESR = 1.8\Omega$). For the configuration $f_{0,buck2}$ the averaged values are $f_{02,avg} = 7kHz$ and $f_{z2,avg} = 15.8kHz$, both obtained values results to be very close to the desired values (Fig.8.27). The averaged ESR contribution for the configuration $f_{0,buck3}$ is $f_{z3,avg} = 17.5kHz$ (Fig.8.28).
The just presented results have been summarised in Tab.8.2.2, where averaged and extracted values have been compared with desired ones. Obtained values are very close with ones obtained both with the fixed-point model (previous chapter) and the hardware model with the PSD computed with the Matlab fixed-point emulation model (previous section). In this table the averaged and obtained values are compared with desired ones.

The ChipScope view when the regulation is activated is shown in Fig.8.29. After the identification presented with previous results, the PID coefficients change. As presented in Sec.4.2.1 for the PID hardware implementation, the compensator gains have been expressed

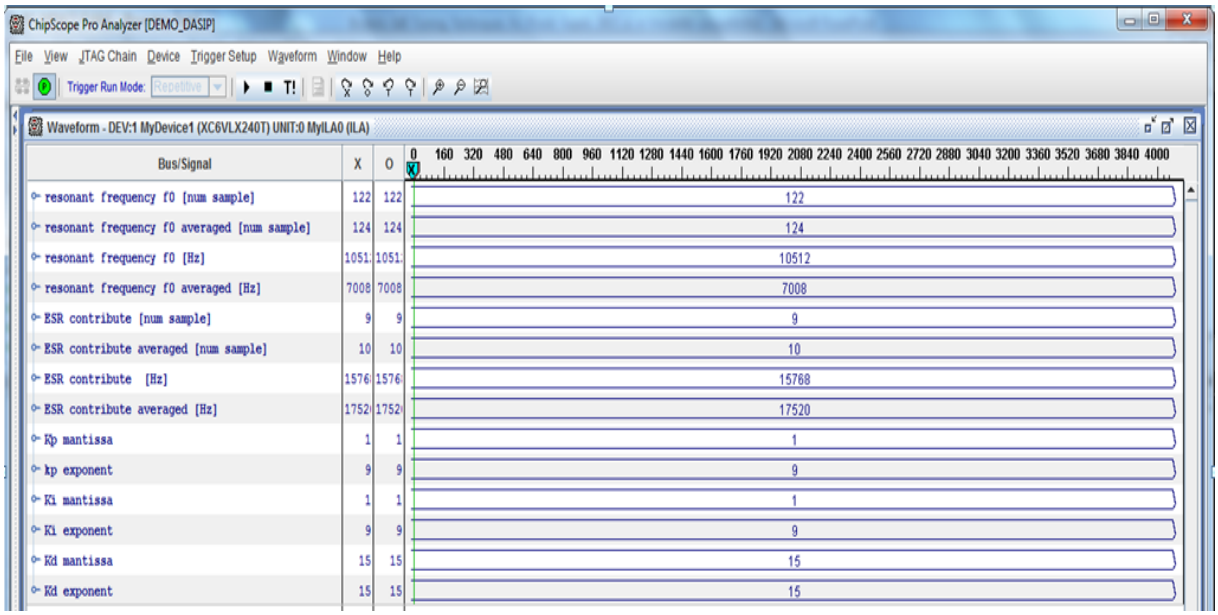

Figure 8.28: Self-tuning prototype: averaged extracted load parameters ($f_{0,buck3}$,ESR = 1.8Ω).

ESR Ω	$f_{z1,avg}$ $[kHz]$	$f_{z,buck1}$ $[kHz]$	$f_{01,avg}$ $[kHz]$	$f_{0,buck1}$ $[kHz]$	$f_{z2,avg}$ $[kHz]$	$f_{z,buck2}$ $[kHz]$	$f_{02,avg}$ $[kHz]$	$f_{0,buck2}$ $[kHz]$	$f_{z3,avg}$ $[kHz]$	$f_{z,buck3}$ $[kHz]$	$f_{03,avg}$ $[kHz]$	$f_{0,buck3}$ $[kHz]$
0.5	15.8	14.5	5.3	4.9	-	-	-	-	-	-	-	-
1	-	-	-	-	15.8	15.9	7	7.3	29.8	33.8	10.5	10.7
1.8	-	-	-	-	-	-			19.3	18.8	10.5	10.7
0	-	-	5.3	4.9	-	-	7	7.3			12.3	10.7

Table 8.3: tab:Self-tuning prototype: extracted, averaged load parameters and desired values comparison.

in terms of mantissa and exponent. When the regulation has been activated new parameters have been output from the regulation block to the PID. New gains are retrieved from a LUT. The LUT has been configured considering the range of possible buck converter configurations ($f_0 < f_s/20$), its entries are related to the possible extracted values which are limited to the finite resolution ($\Delta f = f_s/2N$) imposed by choosing $N = 128$ and the switching frequency. Moreover the PID coefficients for these possible entries can be easily defined by using the automated PID tuning tool presented in Sec.4.1.5. After the load identification, is possible to obtain the best PID configuration (in terms of bandwidth and margins) for the related buck converter configuration. In Fig.8.30 is shown the system reaction to a load step of $100mA$. In this case, the PID configuration is the one used during the identification ($kp_m = 1, ki_m = 1, kd_m = 15, kp_e = 9, ki_e = 9, kd_e = 15$). This configuration is chosen to be stable on the interested range of frequencies. Once the load has been identified and the regulation activated Fig.8.29, the PID gains change according with the identification. The load step system reaction after the regulation has been shown in Fig.8.31. The time needed to recover the output voltage after the step before changing the PID configuration (Fig.8.30) is about $1.6mses$ while, when the PID is changed after the identification, the recovery time is $770\mu sec$. Adjusting both margins and bandwidth of the system, in order to achieve the best

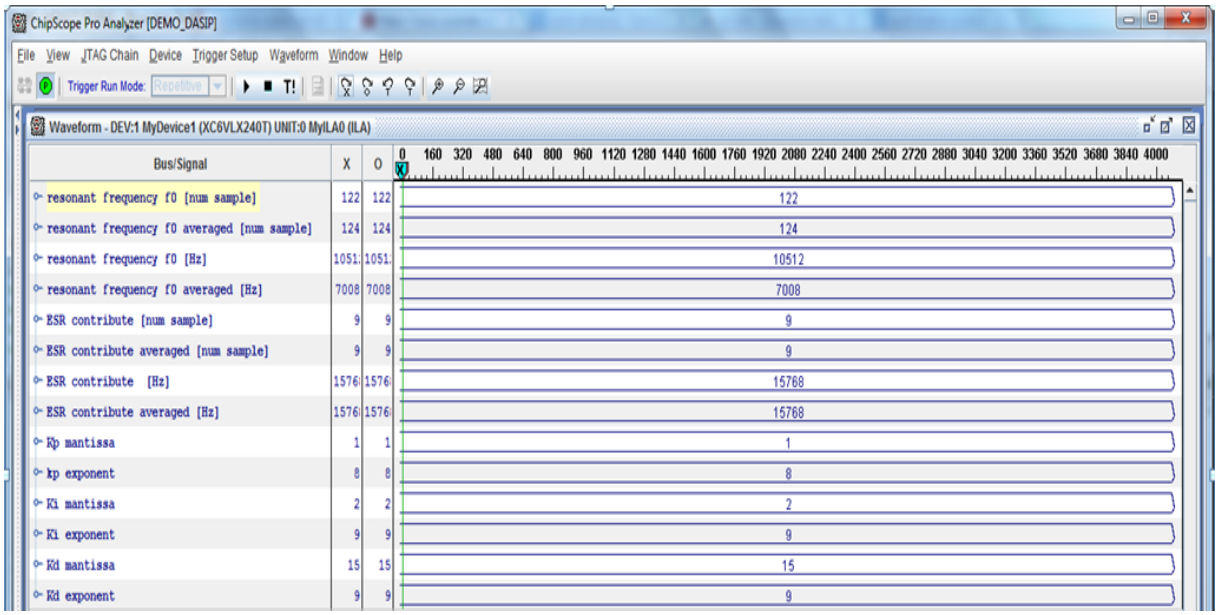

Figure 8.29: Self-tuning prototype: PID gains regulation ($f_{0,buck2}$,$ESR = 1\Omega$).

regulator settings related to the identified buck converter, permits to decrease the system recovery time. Then, if a load step occurs when the best compensation is obtained the system is able to faster recover the output voltage. The dithering amplification effects on the out-

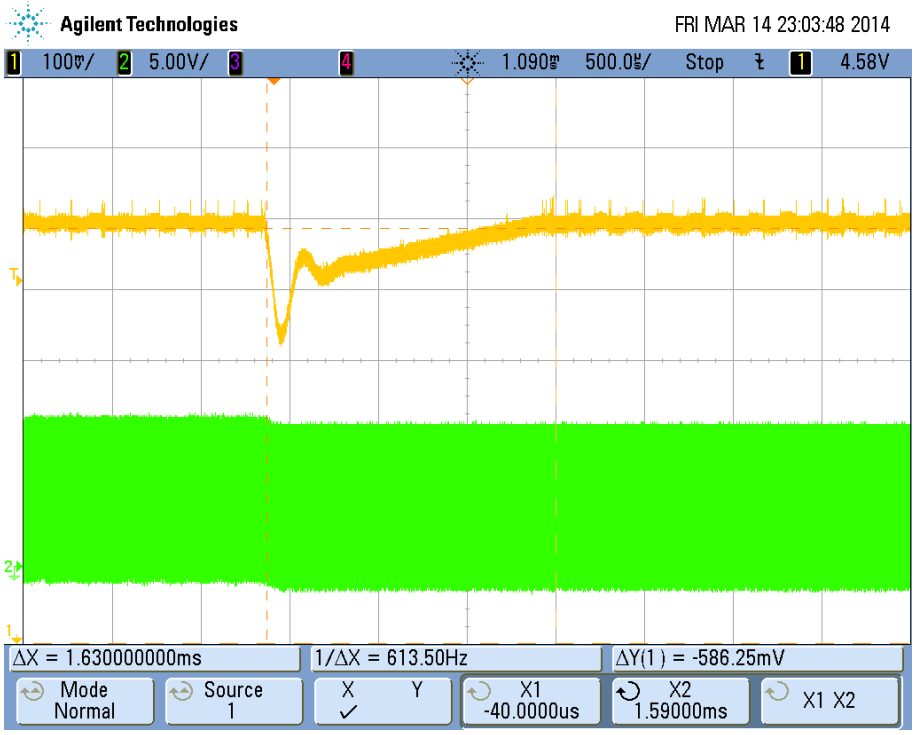

Figure 8.30: Self-tuning prototype: load step response before the PID gains regulation.

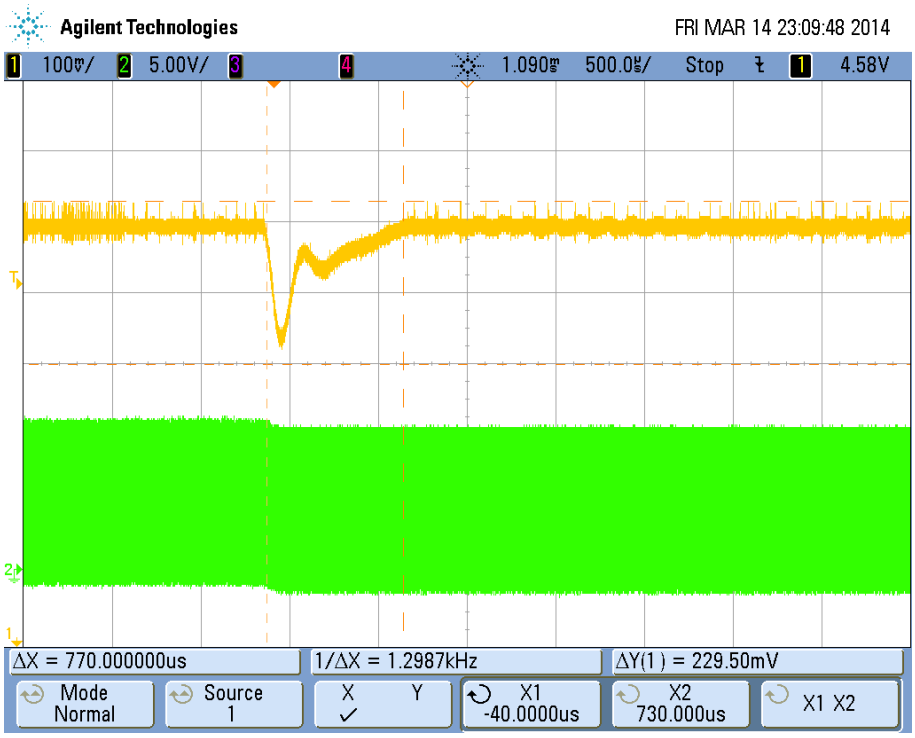

Figure 8.31: Self-tuning prototype: load step response after the PID gains regulation.

put voltage, has been shown in the previous chapter for the fixed-point model and for $\alpha = 2$ (Fig.7.28). To validate this result and the hardware VHDL-coded model, in Fig.8.32 has been shown the output voltage (V_OUT in yellow) when, the dithering effect is doubled and the identification is enabled. The violet signal $PROC_E$ in this figure, is triggered with the signal *proc_en_i* (which enable the self-tuning algorithm) and stay high for the entire self-tuning algorithm (when the FSM comes back to the state **WAIT PROC**). When $PROC_E$ is zero, the output voltage corresponds to the normal dithering condition ($\alpha = 1$) for the third order modulator. When the signal $PROC_E$ is high (*proc_en_i* is high) the dithering is doubled (*res_quant_ds_o = 01*) and its effects is reflected on the output voltage. An averaged increasing of about 150mV can be observed in the oscilloscope output (Fig.8.32). The magnitude of the perturbation can be assumed equal as expected and the theoretical model introduced in Sec.7.2 has been validated.

The digital system in Fig.8.17, have been synthesized considering a Virtex6 FPGA for a PSD computer working on $N = 128$ samples. The obtained maximum frequency is of $91.861 MHz$ [23]. and the hardware resource usage is summarised in Tab.8.4, while in Tab.8.5 is the resource usage related to the Virtex6 FPGA.

Table 8.4: Self-tuning prototype: resource usage

Prototype resources use	#
# RAMs	2
64x32-bit single-port distributed Read Only RAM	2

Table 8.4: continues into the next page

Table 8.4: continues from the previous page

Prototype resources use	#
# MACs	10
16x16-to-33-bit MAC	4
17x16-to-33-bit Mult with pre-adder	2
17x16-to-34-bit MAC with pre-adder	4
# Multipliers	8
16x16-bit multiplier	4
6x5-bit multiplier	4
# Adders/Subtractors	470
12-bit adder	3
12-bit subtractor	3
13-bit adder	3
13-bit subtractor	1
17-bit adder	12
17-bit subtractor	10
2-bit adder	1
20-bit adder	10
21-bit adder	76
22-bit adder	152
27-bit adder	2
28-bit adder	4
28-bit addsub	1
28-bit subtractor	3
29-bit adder	1
4-bit adder	3
5-bit adder	2
5-bit subtractor	2
6-bit adder	1
6-bit subtractor	1
7-bit adder	3
7-bit addsub	1
7-bit subtractor	2
8-bit subtractor	10
81-bit adder	80
81-bit subtractor	80
9-bit subtractor	3
# Adder Trees	3
20-bit / 6-inputs adder tree	2
28-bit / 4-inputs adder tree	1
# Counters	3
2-bit up counter	1
28-bit up counter	1
4-bit up counter	1

Table 8.4: continues into the next page

Table 8.4: continues from the previous page

Prototype resources use	#
# Accumulators	3
10-bit up loadable accumulator	2
4-bit up loadable accumulator	1
# Registers	15549
Flip-Flops	15549
# Comparators	38
1-bit comparator greater	1
11-bit comparator lessequal	1
12-bit comparator equal	2
12-bit comparator greater	4
13-bit comparator greater	2
16-bit comparator greater	3
28-bit comparator greater	1
29-bit comparator greater	2
3-bit comparator greater	2
4-bit comparator greater	1
5-bit comparator greater	1
5-bit comparator lessequal	1
6-bit comparator greater	2
6-bit comparator lessequal	1
7-bit comparator greater	8
7-bit comparator lessequal	2
9-bit comparator lessequal	4
# Multiplexers	1908
1-bit 2-to-1 multiplexer	843
1-bit 4-to-1 multiplexer	5
12-bit 2-to-1 multiplexer	19
16-bit 128-to-1 multiplexer	4
16-bit 2-to-1 multiplexer	171
16-bit 3-to-1 multiplexer	4
16-bit 4-to-1 multiplexer	4
20-bit 2-to-1 multiplexer	154
28-bit 16-to-1 multiplexer	2
28-bit 2-to-1 multiplexer	31
29-bit 2-to-1 multiplexer	17
3-bit 2-to-1 multiplexer	12
4-bit 2-to-1 multiplexer	14
5-bit 2-to-1 multiplexer	5
6-bit 2-to-1 multiplexer	7
7-bit 2-to-1 multiplexer	46
8-bit 6-to-1 multiplexer	4
80-bit 2-to-1 multiplexer	560

Table 8.4: continues into the next page

Table 8.4: continues from the previous page

Prototype resources use	#
9-bit 2-to-1 multiplexer	6
# Logic shifters	4
7-bit shifter logical left	2
7-bit shifter logical right	2
# FSMs	6
# Xors	8
1-bit xor2	8

Table 8.4: ends from the previous page

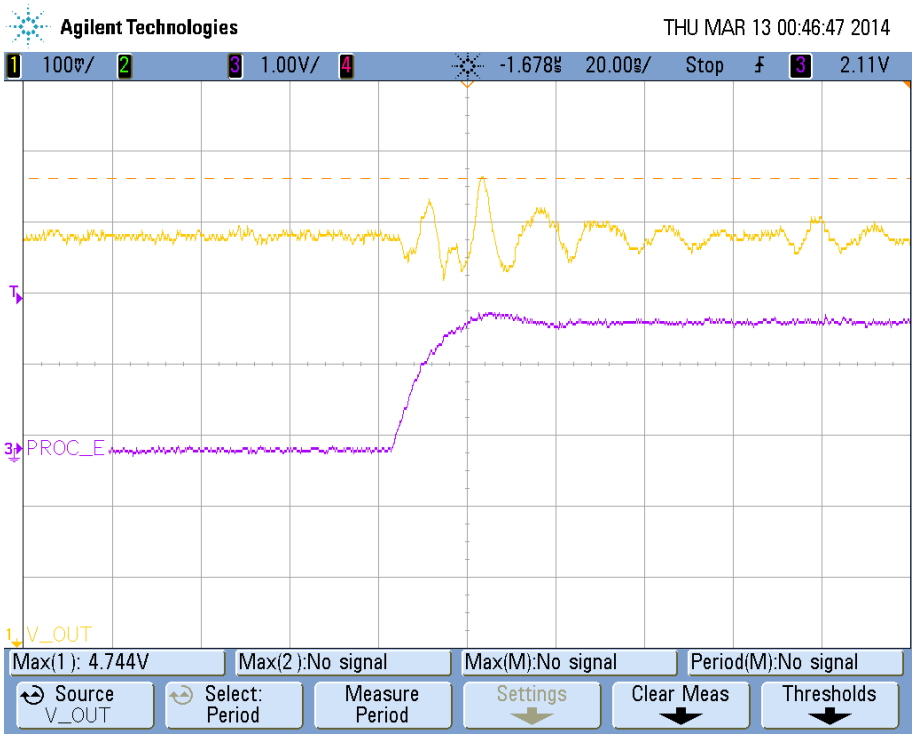

Figure 8.32: Self-tuning prototype: dithering amplification effects on the output voltage.

Virtex6 FPGA resources	
Slice Logic Utilization	
Number of Slice Registers	7491 out of 301440 2%
Number of Slice LUTs	21583 out of 150720 14%
Number used as Logic	21135 out of 150720 14%
Number used as Memory	448 out of 58400 0%
Number used as SRL	448
Slice Logic Distribution:	
Number of LUT Flip Flop pairs used	23190
Number with an unused Flip Flop	15699 out of 23190 67%
Number with an unused LUT	1607 out of 23190 6%
Number of fully used LUT-FF pairs	5884 out of 23190 25%
Number of unique control sets	311
IO Utilization:	
Number of IOs	196
Number of bonded IOBs	194 out of 600 32%
Specific Feature Utilization:	
Number of BUFG/BUFGCTRLs	2 out of 32 6%
Number of DSP48E1s	14 out of 768 1%

Table 8.5: Self-tuning prototype: Virtex6 resource usage.

Chapter 9

Scalable FFT and autocorrelation-based HDL processor

In this Chapter, a complete description of the autocorrelation-based DSP block (PSD computer) is given. The original base design of the system, which is discussed in [89, 24], consists in an autocorrelation-based processor whose central part is a scalable FFT module. The here described DSP block is the PSD computer exploited in the previous chapter for the on-line controller prototyping. This is composed of P processing elements that operate in parallel. The bigger P, the faster the computation but also the larger the requested chip area. P is a degree of freedom of the designer and must be chosen accordingly to the previous considerations. In the SMPS system case, resource usage requirements are tight and the clock frequency must be at least $70MHz$ (as for results presented in the entire dissertation).

The implemented algorithm is a variation of the FFT called Constant Geometry Structure FFT (CG-FFT). This algorithm allows data shuffling from one computational stage to another to keep the same structure. Thus, the data flow follows a constant scheme with benefits in terms of control complexity.

The architecture of the processor is fully scalable in terms of transform length, data bits and Processing Elements (PEs). Each PE implements the basic operation of the FFT, and all the PEs operate in parallel. A processor with many PEs will take less clock cycles to complete an FFT computation of a given length than a processor with few PEs, but will occupy more FPGA resources. This tradeoff makes the number of PEs a degree of freedom in the PSD computer design. The structure of the design was chosen considering resource usage requirements, while allowing improving of timing performances if necessary, thanks to scalability.

Computation of FFTs is performed by using sums, differences and multiplications by unitary module complex values called twiddle factors (TF). In conventional approaches twiddle factors are either retrieved from a Look-Up Table (LUT), or they are computed at runtime by exploiting a CORDIC pipeline-based system. For this final year project, a novel algorithm for twiddle factors generation exploiting the properties of CG-FFT was ideated, with the goal of reducing resource requirements by achieving an improved PE-scalability. Two hy-

brid CORDIC-LUT hardware architectures implementing the algorithm were designed. The first, called *shared core*, uses a single module to perform phase rotations of TFs while the second, called *pipelined*, achieves a higher throughput than the former but uses more hardware resources. Both architectures prove to be more efficient in terms of resource usage if compared to other architectures in the State of the Art. Furthermore, an analysis based on results comparing with MATLAB built-in functions *fft* and *xcorr*, showed that, accuracy being equal, the designed models always use less resources than the traditional approach.

The whole project was developed by using VHDL RTL, while the test environment was based on MATLAB. A fixed-point library implementing the rounding algorithms of the standard IEEE *fixed_pkg* VHDL package was written. A set of functions models every block of the processor and high-level language scripts generate stimuli and test vectors for the VHDL testbenches. This allows us to test the overall system and all the single blocks for every configuration of the parameters.

A PSD computer based on autocorrelation embedding the FFT scalable processor was implemented. The whole system was synthesized on Xilinx Virtex-5 FPGA showing complete fullfillment of design requirements and good accuracy performance, as seen in Fig. 9.1.

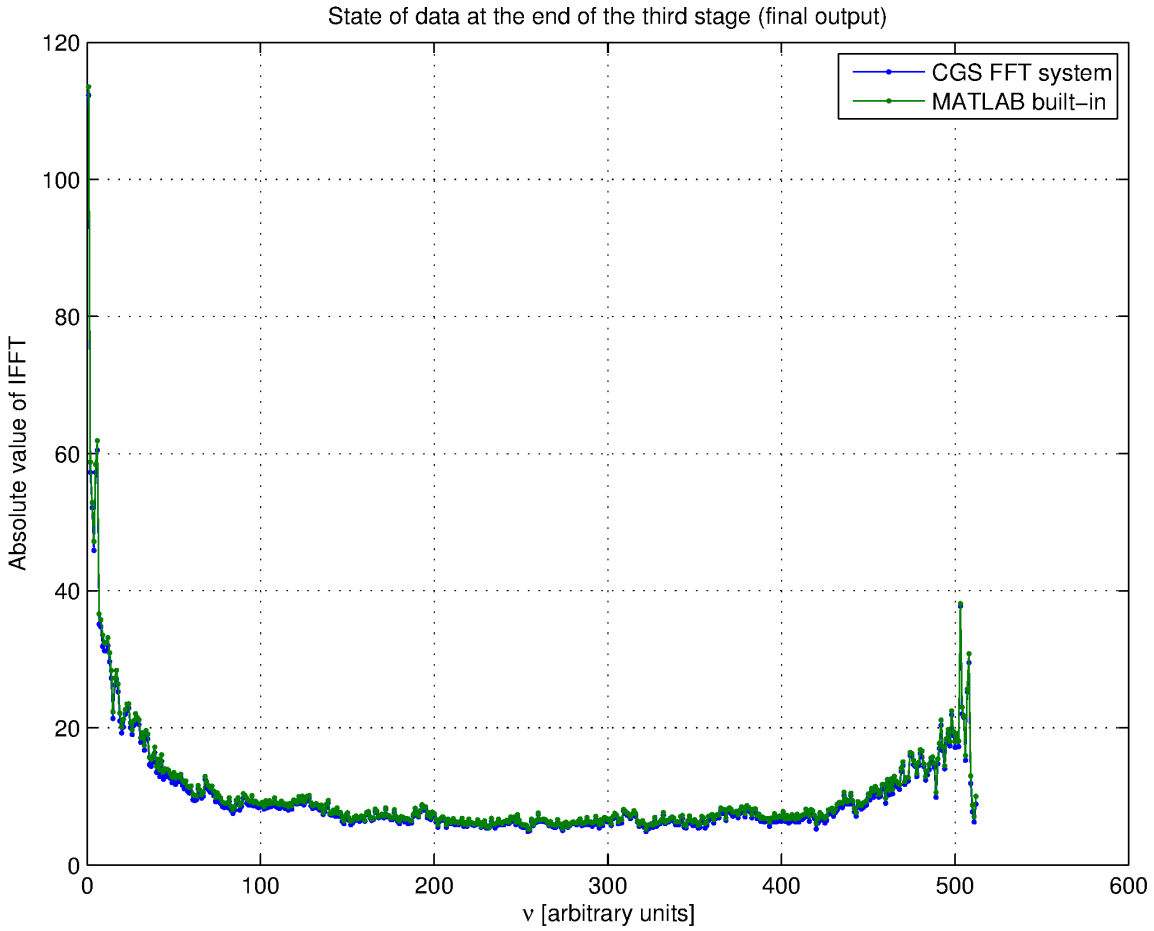

Figure 9.1: Data at the end of computation. Picture obtained with 18-bit resolution and scaling factor of 7.

9.1 Scalable FFT architecture

Constant geometry FFTs are algorithms that perform the same data shuffle at every stage. The first CG-FFT algorithm as a modified version of the radix-2 FFT was introduced in [83]. Since then, other authors have contributed with novel variations [7][7]. The implemented architecture takes advantage of the constant structure obtained by both approaches to achieve PE-scalability. In this chapter the family of architectures introduced in [107] and based on [7] is described. Furthermore, its VHDL RTL implementation is covered, with particular attention to the hierarchical structure of modules and custom data types used.

9.1.1 Description of the implemented architecture

A constant geometry algorithm for different classes of trigonometric transforms, like the FFT, the Discrete Harley Transform (DHT), the Discrete Sine and Cosine Transform (DST and DCT) was introduced in [7]. In the case of FFT, the regularity of the algorithm is due to the same ordering between the stages as in Peases's FFT [83], but with a generalisation to any radix. It is possible to observe that in the radix-2 FFT case the algorithms have the same structure, thus for the rest of the dissertation CG-FFT will be used to denote the result of both approaches without ambiguity.

An example dataflow of CG-FFT is shown in Fig. 9.2. The data shuffle performed between each stage is called *perfect shuffle* or *Faro shuffle*. This is represented in Pease's algorithm, whose formulation is in Eq. C.1, with $L^N_{\frac{N}{2}}$. The concept can be made clearer by using an example with a deck of cards. A perfect shuffle is performed when the deck is perfectly split into two and a card of one of the obtained decks is alternated with one from the other, as shown in Fig. 9.3. Shuffling the deck k times results in a perfect shuffle of *order* k. Moreover, a perfect shuffle of order $k = \log_2 N$, when N is a power of two, does not affect the original order of elements. The idea behind the architecture in [107], is to decompose this permutation in order to obtain scalable structures. Considering data at the beginning of each step

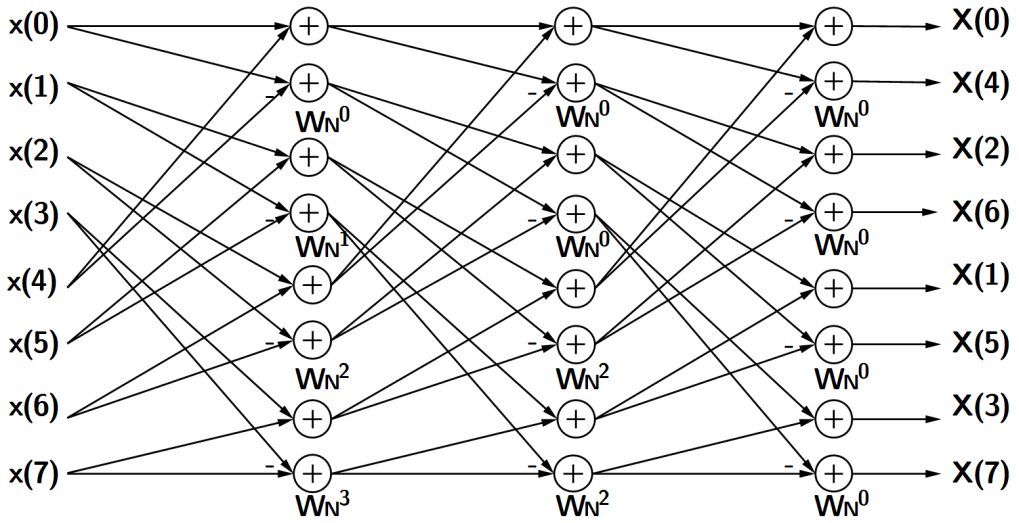

Figure 9.2: Flowgraph of the radix-2 DIF CG-FFT with $N = 8$.

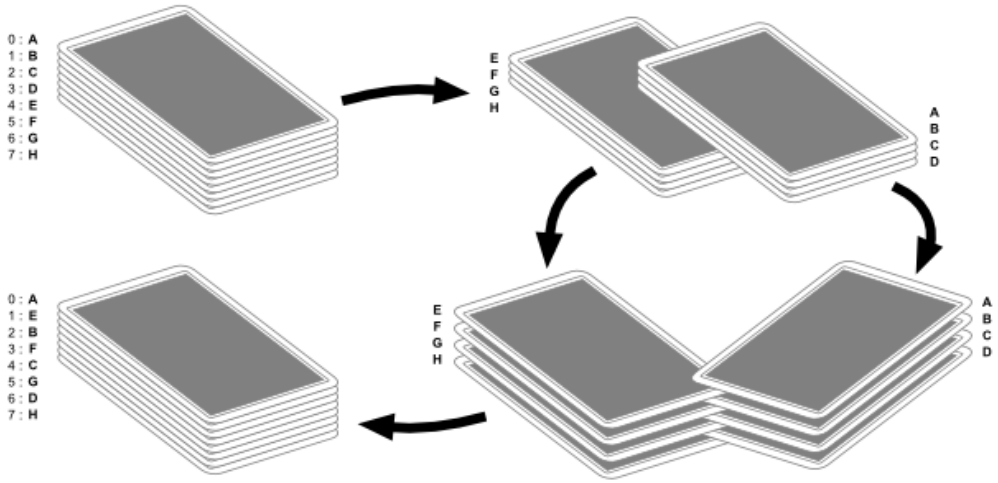

Figure 9.3: Example of perfect shuffle with a deck of cards. From [46].

of the CG-FFT as a vector, it is possible to associate an *address* to every element [107]. The address of element a_i from vector $\boldsymbol{a}$ is the binary representation of its index i written as

$$\text{bin}(i) = [x, y, z] = [\underbrace{x_u, \ldots, x_1}_{x}, \underbrace{y_v, \ldots, y_1}_{y}, \underbrace{z_w, \ldots, z_1}_{z}], \tag{9.1}$$

where x_k, y_k, z_k can be 0 or 1. The derivation of the architecture is based on the interpretation of binary addresses and x, y, z which are called *address fields*.

- w is equal to the radix of the algorithm, and z is the address field referring to the *path*. The corresponding decimal value indicates an input of the basic butterfly.
- If $P \in \mathbb{N}$, equal to a power of two, is the number of PEs in the architecture, then $v = \log_2 P$. Field y is said to be address field referring to the PE, which is associated to a natural number p ranging from 0 to $P-1$.
- The remaining address field x refers to the cycle at which data is available for computation. Because at every stage s, where $s = 0, \ldots, \log_2 N$, N data have to be processed, the number of cycles per stage is $C = \frac{N}{2P}$. Cycles are indicated in this text with c, thus $c = 0, \ldots, C-1$.

For example considering data element a_{13}, if $N = 16$ and $P = 2$, it results

$$\text{bin}(13) = [\underbrace{1,1}_{x}, \underbrace{0}_{y}, \underbrace{1}_{z}].$$

This means that element a_{13} must be fed to the first input of PE 0 at cycle 4. Note that $u + v + w = n$ with $n = \log_2 N$. Moreover, being $C = 4$, the overall computation will take 16 clock cycles.

A permutation R of the elements of vector $\boldsymbol{a}$ can be seen as the application of an *operator* ρ to their addresses. Two operators ρ_1 and ρ_2 may be applied in sequence

$$\rho_2\big(\rho_1[a_{n-1} \ldots a_0]\big) = \rho_2[b_{n-1} \ldots b_0],$$

where $[a_{n-1}\dots a_0]$ is the address of a the generic element of vector $\boldsymbol{v}$ and $[b_{n-1}\dots b_0]$ is the address of the same element after the application of ρ_1. The operator associated to the perfect shuffle permutation $L^N_{\frac{N}{2}(k)}$ of order k is $\sigma_{(k)}$, which is defined as follows:

$$\begin{aligned}\sigma_{(k)}[x,y,z] = \big[&[x_{u-k},\dots,x_1,y_v,\dots,y_{v-k+1}],\\ &[y_{v-k},\dots,x_1,z_w,\dots,z_{w-k+1}],\\ &[z_{w-k},\dots,y_1,x_u,\dots,x_{u-k+1}]\big].\end{aligned} \tag{9.2}$$

This corresponds to a circular shift of the address bits from right to left of k positions. The circular shift of two of the address fields, f_1 and f_2, can be indicated as $\sigma^{f_1 f_2}_{(k)}$. It is possible to prove [131] that

$$\sigma_{(k)} = \sigma^{xz}_{(k)}\sigma^{yz}_{(k)}. \tag{9.3}$$

In Suleiman's architecture operator $\sigma^{yz}_{(k)}$ is implemented by a wire interconnection called *Perfect Shuffle Network* (PSN), and operator $\sigma^{xz}_{(k)}$ by a special network calld *Sequential Perfect Shuffle and/or Un-shuffle Network* (SPSN).

Architectural structure

A block diagram illustrating the architecture is shown in Fig. 9.1.1. Both new samples and data from previous computations feed the PE, implementing a butterfly operation. After the execution of the PE, the PSN performs the first shuffling phase, and the *FIFO* blocks fulfil permutation $L^N_{\frac{N}{2}(1)}$ implementing the SPSN.

Fig. 9.1.1 represents the structure of a PE realising the radix-2 DIT butterfly. The *feeder* block selects which data input to consider, while another input comes from a ROM, or a TF generator, to feed a complex multiplier.

The structure of FIFO blocks is illustrated in Fig. 9.1.1: every block is composed of $\log_2 \frac{N}{P} - 1$ levels of Shift-Exchange Units (SEUs) that properly delay computational data. Each SEU is composed of two delay element arrays with a size equal to two power the level of the SEU, and a criss-cross switch. These blocks are cascaded and ordered inside the FIFO from the lowest level, which is 0, to the highest. When two data x_{in} and y_{in} feed the SEU, x_{in} is sent to the second delay array or directly to the output y_{out} depending on the state of the criss-cross switch, while y_{in} is directly inserted into the first delay element array. Each clock cycle, data stored in delay arrays move of one position, until they are output to the criss-cross switch (first array) or to x_{out} (second array). The state of the criss-cross switch changes according to its level: if this is i, transition will occur every 2^i clock cycles. Overall operation of FIFO modules is illustrated in Fig. 9.5, for a single PE and $N = 8$. In the given example only one FFT stage is considered, with input vector $[0,1,2,3,4,5,6,7]^\intercal$, then data feeding the block is assumed to be constant and equal to zero to allow a clearer representation. In the particular case $P = 1$, the PSN results in a simple non-crossed connection, consequently the output of SEU modules in the diagram is expected to be the perfect shuffle of the input vector. This is validated in the second stage, as observed in the right column of Fig. 9.5.

The permutation chain composed of PSN and FIFO constitutes an interruptible data exchange and ordering approach that differs from most of the solutions in the state of the art. Among pipelined architectures, the most similar is the iterative structure from the SPIRAL project which is shown in Fig. C.1. This is because of the coincidence between Pease and Astola-Akopian algorithms in the current case of study. A structure similar to FIFOs and

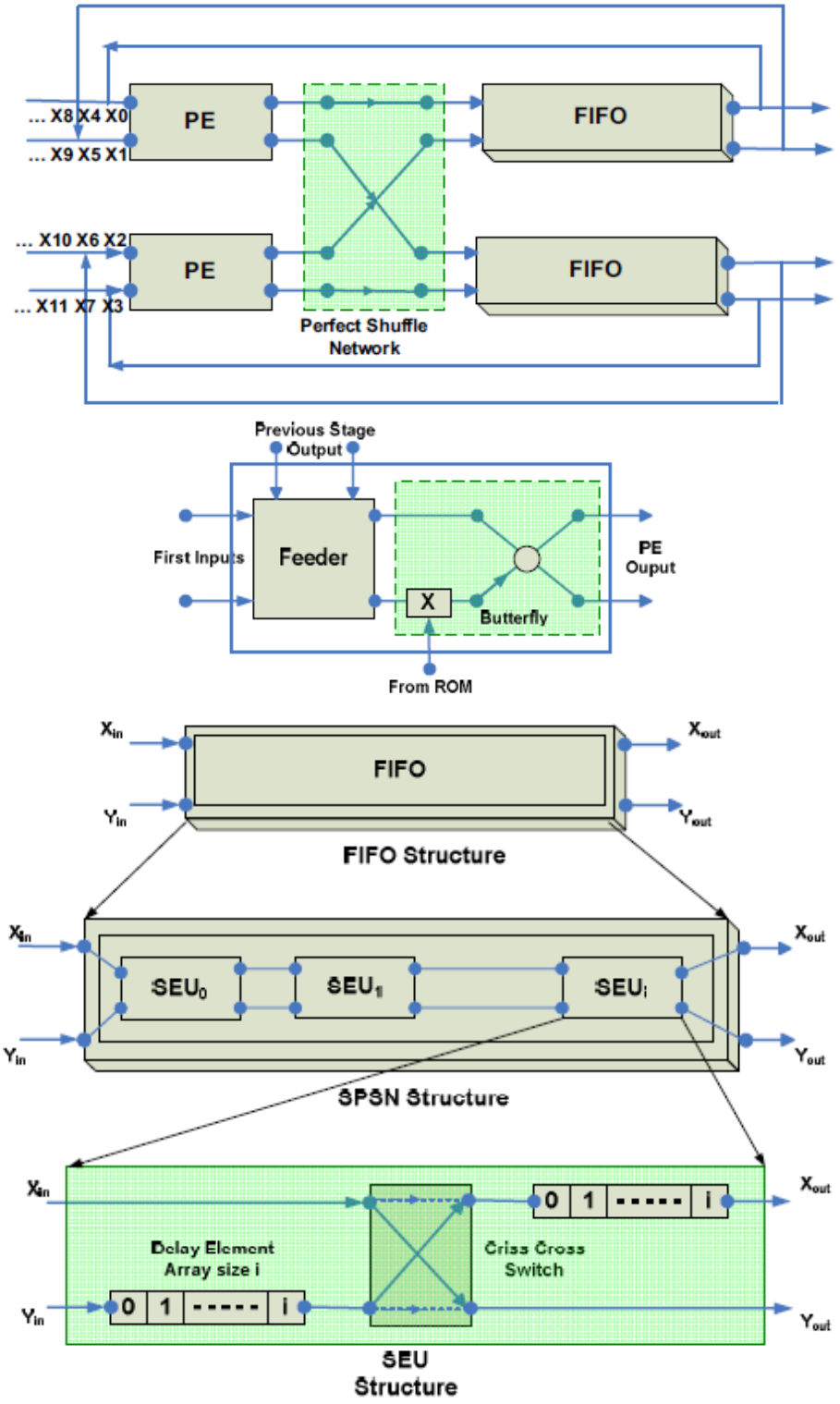

Figure 9.4: Structure of the scalable architecture. From [107].

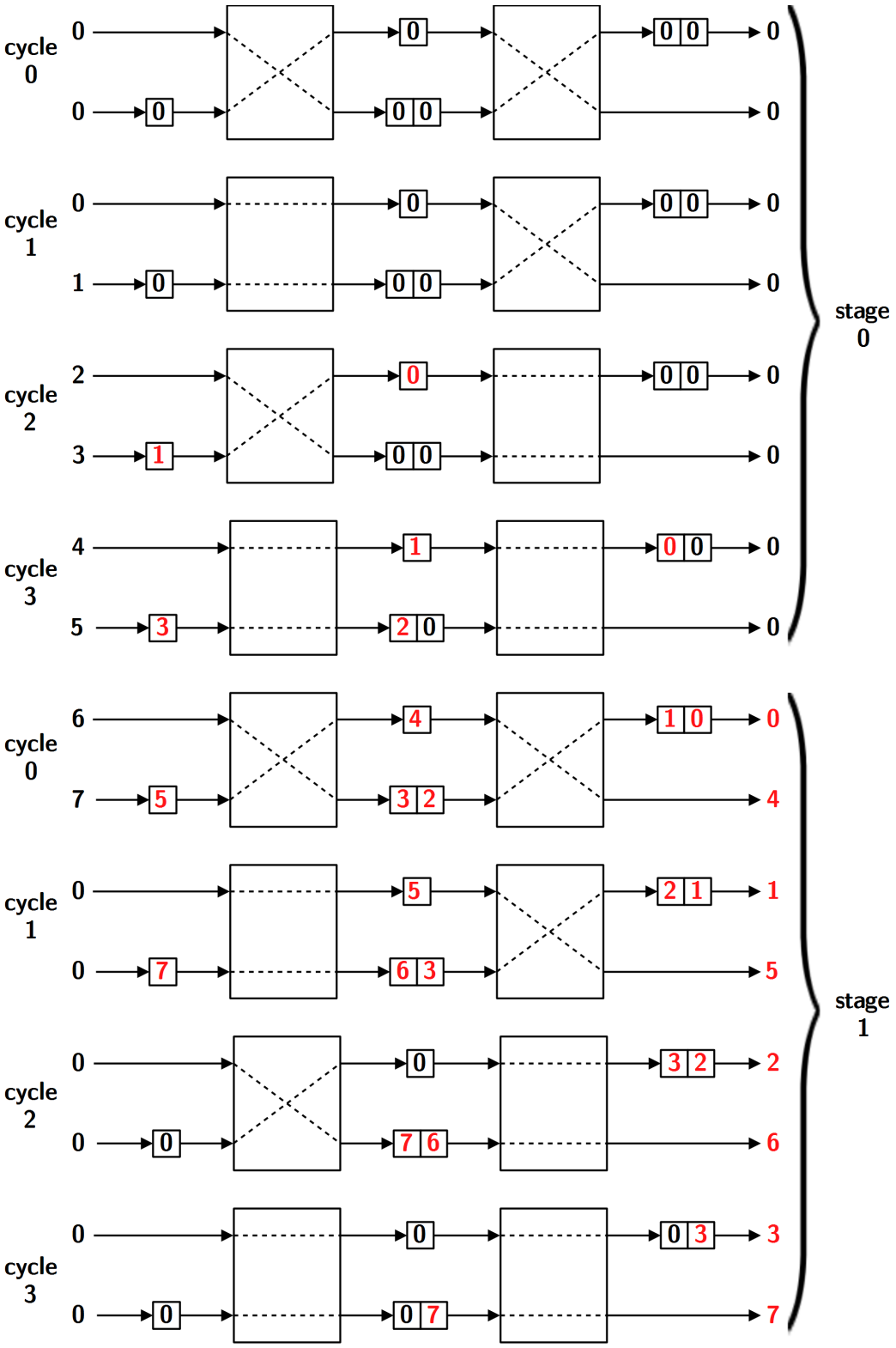

Figure 9.5: Operation of FIFO blocks. $N = 8$, $P = 1$.

SEUs is also discussed in the SPIRAL-related paper of 2005 [75]. The difference, though, is in the implementation of the PE, which is pipelined in [75] and single-clock in [107]. While possibly reducing the operating frequency of the design, the last solution can be more hardware-efficient, so more suitable to the requirements of the commissioned design. Moreover, the architectural family in [107] hardly supports pipelined PEs, because data shuffling and butterfly computations are supposed to have the same timing. Other pipeline structures such as conventional MDC and SDF have the number of PEs fixed and depending on N. With Suleiman's approach, this becomes a tunable parameter which is expected to allow improved flexibility.

The concept of single-clock PE is also present in [3] (Fig. C.8), but in this paper a RAM is used to perform reading and writing, thus an address generator must be used. On the contrary, in [107] no address management is necessary because all shuffling is performed clock by clock in the PSN and FIFO system. A similar structure with shuffling registers and analysed in section C.2 [132], but this structure (Fig. C.2) needs to manage RAM addresses for the output radix-2 stages. This observation can be extended in general to in-place architecture by noticing that in Suleiman's architecture, data are not written in the same register they are taken from. Consequently in [107] the structure is classified as *not-in-place*.

9.1.2 RTL design

The forementioned architecture has been implemented in VHDL RTL. The chosen standard of the language is VHDL93, which allows more advanced constructs if compared with the previous VHDL83 version, while mantaining a high compatibility with synthesis tools.

The whole project takes advantage of the IEEE *fixed_pkg* package [12]. This standard package is built-in in every VHDL200X tool, like Xilinx ISE, but needs a compatibility version to work with VHDL93. The package defines the types *ufixed* (unsigned fixed-point), and *sfixed* (signed fixed-point) with their unresolved matching types *u_ufixed* and *u_sfixed*. *Fixed_pkg* also introduces arithmetic and conversion functions. One of the features of this package is the possibility to select between truncation and rounding and between saturation or wrap when resizing a binary word. This is extensively used in the design both as a default or as an optional selectable capability, increasing the flexibility of the processor.

Throughout the design the naming convention in Tab. 9.1 was adopted for the coding of entities, architectures and packages. The name of architectures is obtained by adding *_behavioral* or *_structural* at the end of the name of the matching entity. When more than one architecture are available for a specified entity, then the suffix is descriptive of their features.

Global VHDL packages

The realised implementation is centered on a hierarchy of VHDL packages of constants. Depending on its context, each module imports some of these packages. In *fft_global_settings* the following editable constants are defined:

global_points_c is the number of points of the FFT, equal to N. It must be a power of two and the default value is 1024.

global_pes_c is the number of PEs of the processor. It must be equal to a power of two, and the default value is 4. It corresponds to P.

Table 9.1: Naming convention adopted in the design.

Suffix	Usage
_i	Input signals.
_o	Output signals.
_ty	Data types.
_g	Generics.
_c	Constants.
_s	Internal signals.
_v	Variables.
_p	Processes.

global_accuracy_int_c is the number of bits of the integer part in the signed fixed-point representation of data. One bit is necessarily a sign bit. The default value is 8.

global_accuracy_frac_c is the number of bits of the fractional part in the fixed-point representation of data. The default value is 10.

global_tf_accuracy_int_c is the number of bits of the integer part in the signed fixed-point representation of twiddle factors. The default value is 1, which means that only the sign bit is used.

global_tf_accuracy_frac_c is the number of bits of the integer part in the signed fixed-point representation of twiddle factors. The default value is 15.

global_pe_multipliers_rounding_c is the rounding type performed inside the PE. The default value is *fixed_truncate*.

All these constants are global: every module imports this package so that editing any of these values consequently causes a change in the structure of the architecture without needing modifications to further VHDL code. *fft_global_settings* also defines:

global_accuracy_tot_c equal to B, is the total number of bits of data words. It is obtained as:

$$global_accuracy_int_c + global_accuracy_frac_c.$$

global_stages_c equal to $\log_2 N$ is the number of stages of the FFT.

global_cycles_per_stage_c is the number of cycles that the computation of every stage takes. It is equal to

$$\frac{global_points_c}{2 \cdot global_pes_c}.$$

global_seu_levels_c is the number of SEU levels of every FIFO structure. It is equal to

$$\log_2 \frac{global_points_c}{global_pes_c} - 1.$$

Package *fft_types* defines the global types used throughout the whole design. These types are used both inside the modules and in their interfaces. The choice of using custom data types for interfaces instead of the more common *std_logic_vector* default type allows easier debugging for strucures with many PEs.

complex_part_ty is a signed number in fixed-point representation. The number of bits of its integer part is *global_accuracy_int_c* and of its fractional part is *global_accuracy_frac_c*. A *complex_part_ty* type signal or variable represents either the real or the imaginary part of computational data. It is defined with the following VHDL code:

```
subtype complex_part_ty is
  u_sfixed( global_accuracy_int_c-1 downto
          -global_accuracy_frac_c);
```

complex_part_vector_ty is an array of *complex_part_ty* elements.

complex_ty is a record of two *complex_part_ty* defined as follows:

```
type complex_ty is
    record
      real_p: complex_part_ty;
      imag_p: complex_part_ty;
    end record;
```

It represents an input or an output of a PE. An array of *complex_ty* is a *complex_vector_ty*.

twiddle_factor_part_ty is a signed number in fixed-point representation. It is composed of *global_tf_accuracy_int_c* for the integer part and *global_tf_accuracy_frac_c* for the fractional part. Its VHDL definition is

```
subtype twiddle_factor_part_ty is
  u_sfixed( global_tf_accuracy_int_c-1 downto
          -global_tf_accuracy_frac_c);
```

twiddle_factor_ty represents a twiddle factor in the architecture of the processor. It is defined as the record

```
type twiddle_factor_ty is
  record
    real_p: twiddle_factor_part_ty;
    imag_p: twiddle_factor_part_ty;
  end record;
```

Like previously described types, the *twiddle_factor_ty* data type has its own array counterpart *twiddle_factor_vector_ty*.

Package *math_utils* defines mathematical functions that are either to be implemented on hardware or used at compilation time. Almost every module and package, including *fft_global_settings* uses its features to implement the full scalability of the processor in terms of PEs, bit width of data words and number of elements to process.

Datapath

VHDL entity *pe* is implemented in architecture *pe_structural*, whose diagram is shown in Fig. 9.6. The first *mode_mux* block allows to choose what data to compute between *datafbk*, that are complex values output by the FIFO, and *dataext*, or rather data coming from an external source. Selection is driven with signal *sel_i*. Module *complex_ty_adder* performs both

complex addition and subtraction, saturating their values in case of overflow. This is possible by using to the *resize* function from packge *fixed_pkg*, which allows to choose both saturation and truncation type. *Complex_value_multiplier* implements complex multiplication with three real multipliers by exploiting Gauss'Alg. [21]. Its VHDL fully-parametric interface is in Lst. 9.1. Considering the two complex numbers $a+jb$ and $c+jd$, their product λ is obtained applying:

$$\begin{aligned} m_1 &= (a+b)\cdot c \\ m_2 &= (d+c)\cdot b \\ m_3 &= (d-c)\cdot a \\ \Re\{\lambda\} &= m_1 - m_2 \\ \Im\{\lambda\} &= m_1 + m_3. \end{aligned} \tag{9.4}$$

Module *complex_value_multiplier* too takes advantage of *fixed_pkg*: the rounding type is selectable by editing constant *global_pe_multipliers_rounding _c* in *fft_global_settings* that matches generic *rounding_g* of the entity interface when instantiated. Enabling rounding results in increased accuracy but also in usage of more logic [12], thus for default this is disabled. The complete implementation of Eq. (9.4) in architecture *complex_ty_ multiplier_behavioral* is shown in Lst. 9.2. In [107] Suleiman implements the proposed architecture by using four real multipliers for each PE. By choosing the full-parametric Gauss-based complex multiplier in the given listing, we expect to achieve a saving in terms of FPGA resources. Nonethless, Eq. (9.4) defines a procedure that requires three additional real additions or subtractions if compared to the traditional approach [21]. This could increase the critical path in the design. We consider this a minor drawback, given the tight resource usage requirements from the customer. In Fig. 9.6 signal *sum_s* realizes $y_l = x_l + x_{l+\frac{N}{2}}$ while signal

Listing 9.1: Interface of entity *complex_value_multiplier*.

```
library ieee;
use ieee.std_logic_1164.all;
library ieee_proposed;
use ieee_proposed.fixed_float_types.all;
use ieee_proposed.fixed_pkg.all;
use work.fft_types.all;

entity complex_value_multiplier is
 generic( a_int_g  : integer := 8;
    a_frac_g   : integer := 10;
    b_int_g    : integer := 8;
    b_frac_g   : integer := 10;
    out_int_g  : integer := 8;
    out_frac_g : integer := 10;
    rounding_g : fixed_round_style_type := fixed_round);
 port( datain_a_real_i : in u_sfixed(a_int_g-1 downto -a_frac_g);
   datain_a_imag_i : in u_sfixed(a_int_g-1 downto -a_frac_g);
   datain_b_real_i : in u_sfixed(b_int_g-1 downto -b_frac_g);
   datain_b_imag_i : in u_sfixed(b_int_g-1 downto -b_frac_g);
   dataout_real_o  : out u_sfixed(out_int_g-1 downto -out_frac_g);
   dataout_imag_o  : out u_sfixed(out_int_g-1 downto -out_frac_g));
end entity complex_value_multiplier;
```

product_s realizes $z_l = (x_l - x_{l+\frac{N}{2}})W_N^l$, thus implementing the Gentleman-Sande butterfly. Choosing radix-2 DIF instead of DIT like in [107], brings an advantage in terms of control

Listing 9.2: Behavioral architecture of module *complex_value_multiplier*.

```
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use ieee_proposed.fixed_float_types.all;
use work.fft_types.all;
use work.fft_global_settings.all;

architecture  complex_value_multiplier_behavioral of
          complex_value_multiplier is
begin
  compute_p: process(   datain_a_real_i,datain_a_imag_i,
                 datain_b_real_i,datain_b_imag_i)
    -- Temporary results are such that they can contain
    -- the result of a multiplication and a sum
    constant upper_bound_c  : integer :=
      sfixed_high(datain_a_real_i,'*',datain_b_real_i)+1;
    constant lower_bound_c  : integer :=
      sfixed_low(datain_a_real_i,'*',datain_b_real_i);
    variable x_v  : sfixed(upper_bound_c downto lower_bound_c);
    variable y_v  : sfixed(upper_bound_c downto lower_bound_c);
    variable z_v  : sfixed(upper_bound_c downto lower_bound_c);
  begin

    -- First step of Gauss' algorithm
    x_v := datain_b_real_i*(datain_a_real_i+datain_a_imag_i);
    y_v := datain_a_imag_i*(datain_b_real_i+datain_b_imag_i);
    z_v := datain_a_real_i*(datain_b_imag_i-datain_b_real_i);

    -- Assigning output and possible rounding
    dataout_real_o <= resize(   x_v-y_v,
                        out_int_g-1,
                        -out_frac_g,
                        fixed_saturate,
                        rounding_g);
    dataout_imag_o <= resize(   x_v+z_v,
                        out_int_g-1,
                        -out_frac_g,
                        fixed_saturate,
                        rounding_g);

  end process compute_p;
end architecture complex_value_multiplier_behavioral;
```

that will be explained in the following section 9.1.2. Being PE operations necessarily iterated throughout the execution of the FFT, performing additions can lead to a saturation of all data. Consequently, a one-place arithmetic right shift of both signals is necessary. This can be considered a division by N of the whole FFT, as required in Eq. (B.8). The second *mode_mux* multiplexer is driven by signal *mode_i* and selects between uncomputed data from the first multiplexer and *post_shift* signals. Its output feeds the *out_reg* output registers.

The PSN is implemented with a simple behavioral architecture, which is in Lst. 9.3. A variation of the number of PEs results in a different PSN structure, and this is managed with the global constants package *fft_global_settings*. Moreover this light solution shows the advantages of Suleiman's architecture. Permutation of data between PEs is achieved in few lines, without needing any memory nor control for information coherence and concurrent access. Constant *complex_zero_c* is a *complex_ty* value with all fields equal to zero. It is defined in *fft_types*.

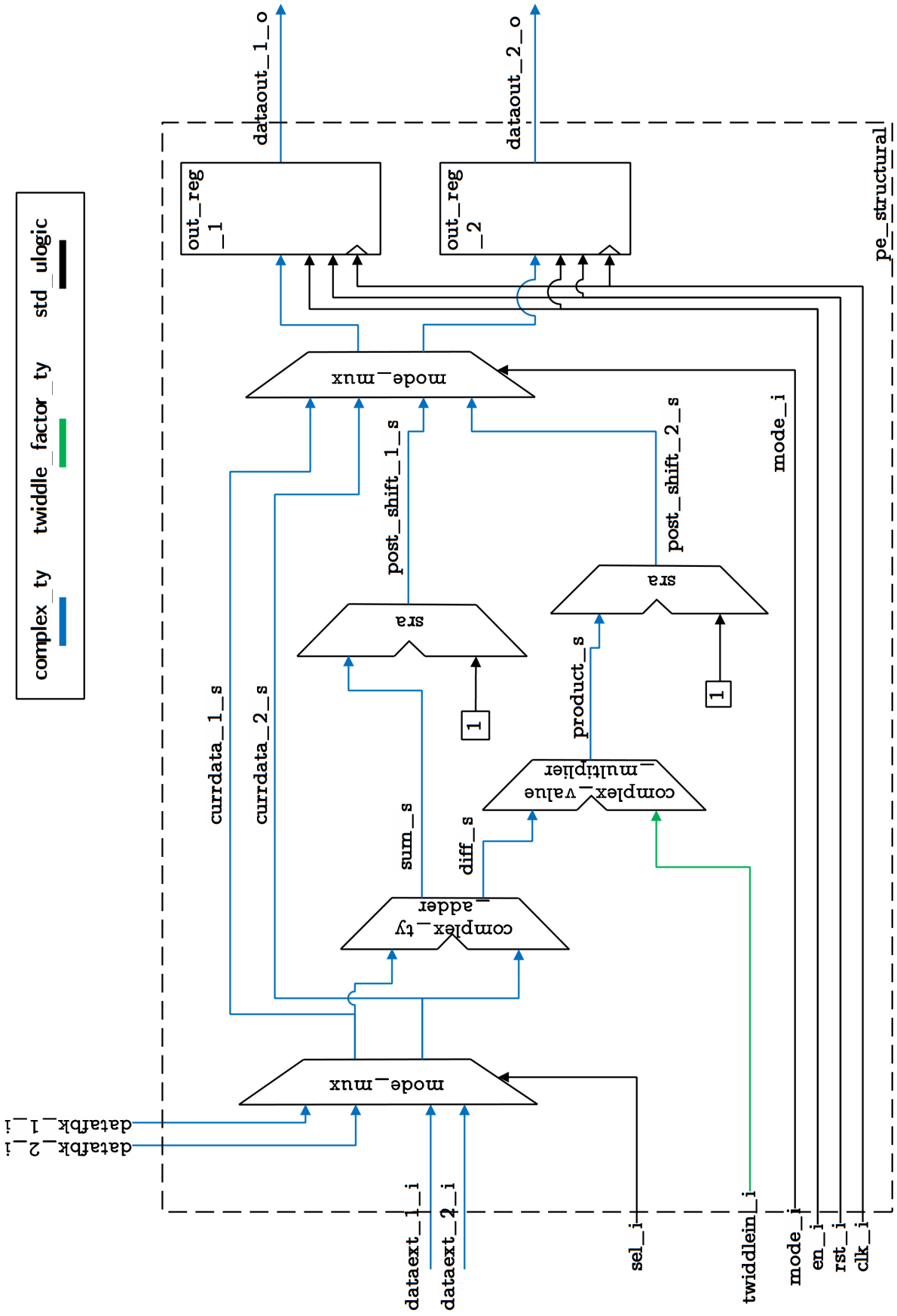

Figure 9.6: Diagram of architecture *pe_structural*.

Listing 9.3: Behavioral architecture of module *psn*.

```
use work.fft_types.all;
use work.fft_global_settings.all;

architecture psn_behavioral of psn is
begin
  process(datain_i)
  begin
    for i_v in 0 to 2*global_pes_c-1 loop
      -- first half of datain_is
      if i_v<=global_pes_c-1 then
        dataout_o(2*i_v)<=datain_i(i_v);
      -- second half of datain_is
      elsif i_v>=global_pes_c then
        dataout_o(2*(i_v-global_pes_c)+1)<=datain_i(i_v);
      else
        dataout_o(i_v)<=complex_zero_c;
      end if;
    end loop;
  end process;
end architecture psn_behavioral;
```

In order to analyse the implementation of FIFO blocks, it is necessary to explore the architecture of module *seu*. Delay arrays are implemented by the *chain* signals as a chain of registers. This structural choice avoids an overhead in the usage of FPGA built-in RAM modules that could be too many to be instantiated. Management of *chain* signals requires three processes:

- *criss_cross_p* implements the criss-cross switch of the SEU, and is driven by signal *state_ty*.
- *chain_comb_p* is a combinatorial process that computes the next state of register chains. When it is enabled with *en_i*, it performs a single-position shift of data in the chain and the insertion of a new element at the top. The behaviour is comparable to the classic FIFO queue.
- *chain_upd_p* is a sequential process that implements the actual registers forming the queue. Every clock cycle, it updates the queue according to the output of the previous *chain_comb_p* block.

A diagram depicting the behavioral architecture of a SEU is shown in Fig. 9.7. The output of the SEU module is taken from the criss-cross switch or from signal *chain_2_s* as its last element, taking advantage of its VHDL attributes with expression *chain_2_s'high*.

SEU modules are instantiated inside *fifo* structures by using the *generate* VHDL construct and generics. The structural scheme obtained is illustrated in Fig. 9.1.1.

Another fundamental module, whose architecture will be covered in the following chapter of this dissertation, is *twiddle_factors_system*. This provides the correct twiddle factors to the PE at each clock cycle. The module is driven by an enable signal. Internal counters keep trace of stage and cycle of the TFs that are being computed, and more sets of twiddle factors are in involved in the generation process at the same time. A single-bit signal specifies the transform type, if it is direct or inverse. Timing is managed by a ack-nack protocol: the *twiddle_factors_system* module outputs a high *valid_o* signal whenever the computation of

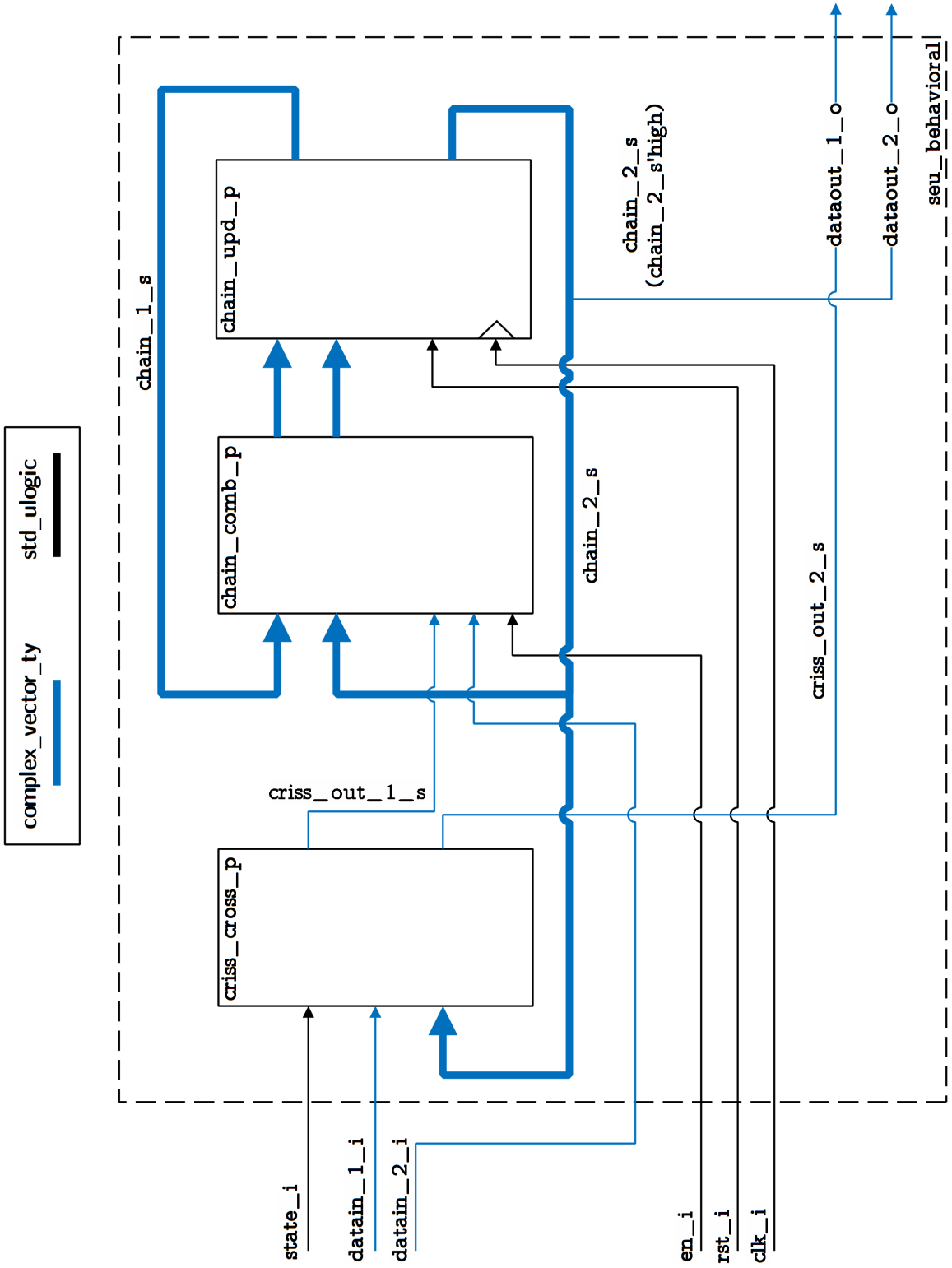

Figure 9.7: Diagram of architecture *seu_behavioral*.

a set of P twiddle factor is complete, and keeps generating TFs unless the *en_twiddle_calc_i* signal is driven to a low level by the control of the processor.

The complete dataflow of the processor is illustrated in Fig. 9.8. A *std_ulogic* register stores the requested transform type when enabled. Its output feeds the *twiddle_factors_system* module. This block receives an enable signal (*en_twiddle_calc_i*) and a synchronous clear as further inputs, and its *twiddle_valid* output is directly taken to the output of the datapath. A *counter* is used both to manage the state of *fifo* modules and to signal the reaching of the last cycle or of the last stage. *dataout_o* is an array of *complex_ty* values with P elements. These are taken at the output of the PEs, following the CG-FFT flowgraph shown in Fig.9.2.

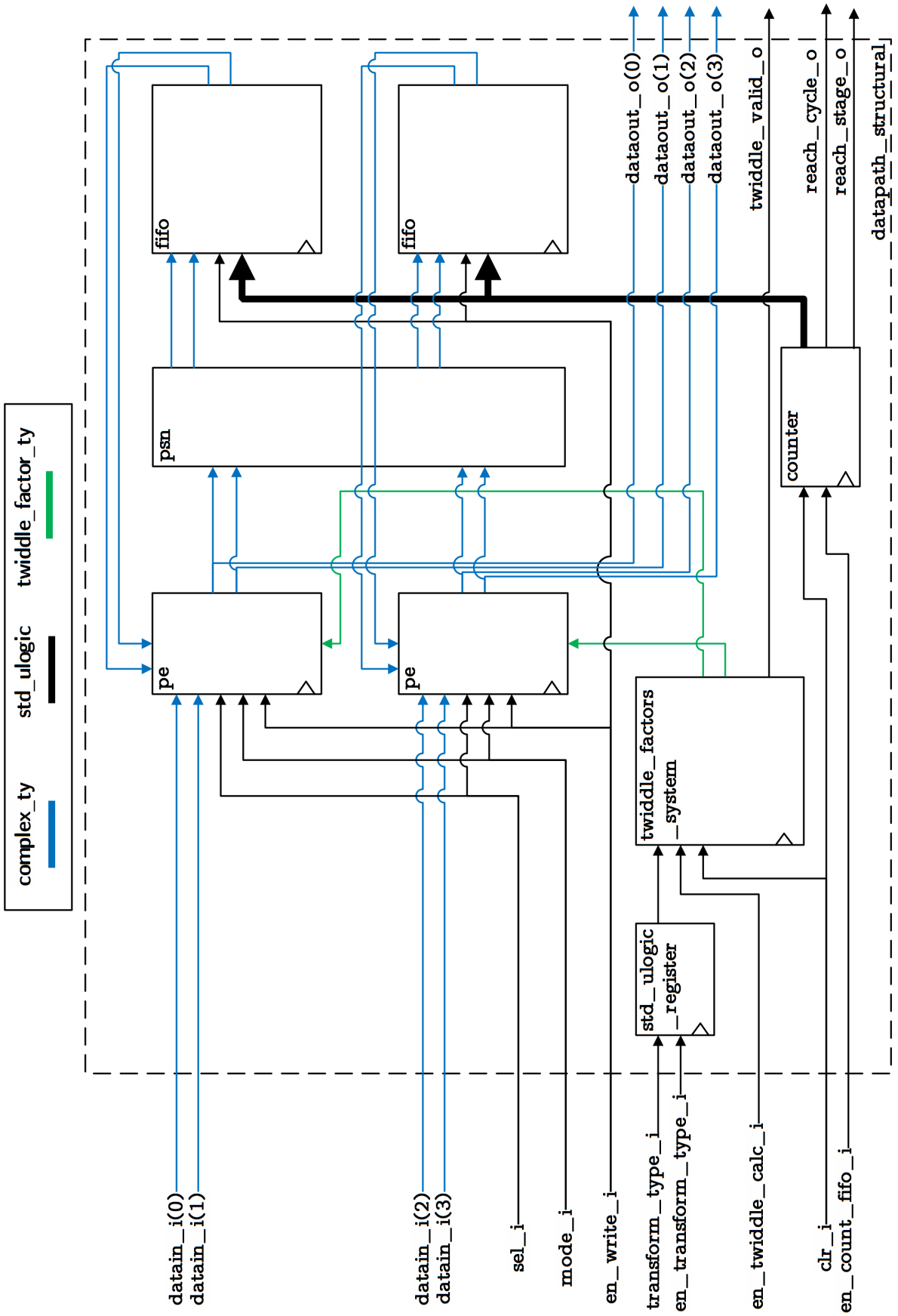

Figure 9.8: Diagram of architecture *datapath_structural*. $P = 2$. Clock and asynchronous reset signals are omitted for simplicity.

Control

The control of the processor is realized with a Moore state machine, whose dataflow diagram is shown in Fig. 9.9. The first two states are used to initialise the processor. State *wait_transf_type* indicates by using the signal *nready_o* if the system is ready or not to acquire the transform type. The system will be in this state until the signal *req_i* driven by an external module or by the user will be high.

When *req_i* is set to one, the state machine moves to *acq_transf_type*, driving *en_transform_type_o* to one. The *std_logic_register* in the datapath samples the bit specifying the transform type, and in the following clock cycle the state becomes *wait_data*. Between states *wait_data* and *acq_data*, the behaviour of the processor is similar to the states *wait_transf_type* and *acq_transf_type*. The difference is that in the first case data is sampled in the internal registers of the PEs and subsequently shifted through the FIFOs, while in the first only one bit is stored in the *std_ulogic_register* in Fig. 9.8. When in these states, signal *sel_o* is be set to one so that the input of each PE is not selected from the feedback connection and *mode_o* is be zero, allowing input data to be sampled without performing any computation on them (Fig. 9.6). When *reach_cycle_i* becomes high, all FIFO locations have been filled, thus calculation can start.

The first computational stage *comp_tf* is needed to require the generation of the first twiddle factors. The processor waits until *twiddle_factors_system* signals that TFs are available by driving *twiddle_valid_i*. The Moore machine commutes to state *comp*, unless the TF generator sets *twiddle_valid_i* to zero, or counting is completed. A loop between *comp* and *comp_tf* ensures a correct synchronization of both the *twiddle_factors_system* and data inside the PEs.

When both *reach_stage_i* and *reach_cycle_i* are set to one because counting is complete (Fig. 9.8), the state changes to either *ret* or *ret_tf* depending on signal *twiddle_valid_i*. If this is equal to one, then the transition is towards *ret*: this state is analogous to *comp*, except that *over_o* is set to one. This is an output to the external world, to signal that the computation is complete. Similarly, *ret_tf* matches *comp_tf*, but in the last case, outputs are unchanged. Indeed, from the following clock cycle after the reaching of *ret*, *dataout_o* is the bit-reversed output of the FFT, thus the change of signal *over_o* can be used as a trigger to sample the final output of the FFT. If *over_o* was high in state *ret_tf* this advantage would not be exploitable. A transition from *ret* to *ret_tf* is not possible because the twiddle factor generator outputs a constant value throughout the whole last cycle. If the implemented FFT algorithm was radix-2 DIT, the TFs of the last stage would have been different one to the other, with an increase of computational time. Moreover, a more complex protocol should have been used. This is not the implemented case, nonethless the Moore machine considers the eventuality of a transition for improved robustness.

When the counter in Fig. 9.8 sets *reach_cycle_i* to one, the computation terminates and the processor re-initialises. This is also possible when *stop_i* is driven to one by the user, requesting immediate interruption of data return, but not in any state other than *ret* and *ret_tf*.

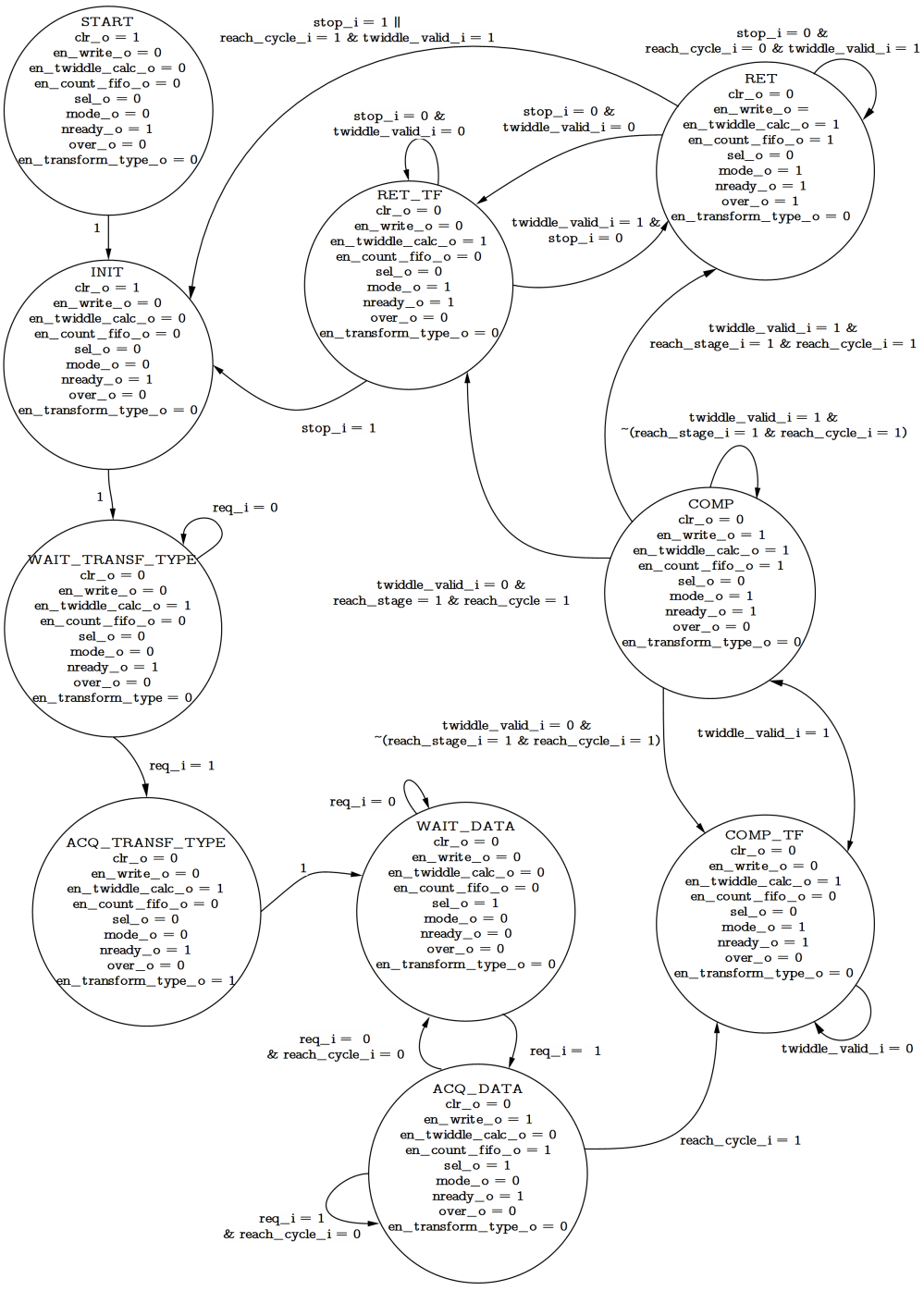

Figure 9.9: Dataflow diagram of the control of the processor.

9.2 TF module general structure and LUT-based architecture

The conventional approach to obtain Twiddle Factors (TF) for FFT computation is to have them stored in a Lookup Table (LUT). When a particular phase rotation is required, the index of the corresponding TF is called, according to B.10. In this chapter, the overall description of the TF system module is given. Then, the particular case of LUT-based structure is analysed in its implementation detail.

9.2.1 Inside *twiddle_factors_system*

Fig. 9.10 shows the architecture of module *twiddle_factors_system_structural* and how Alg. 1 fits into this structure. Two counters (*cycle_count_i* and *stage_count_i*) keep trace of the current cycle and stage in the computation of the FFT. Modules *addr_calc* implement Eq. (9.10) and are enabled by the *ready_for_new_address_s* wire. Notably, each *addr_calc* matches a PE and generates the indexes of TFs that feed it throughout the whole computation. In the diagram, custom data type *address_ty* is the binary representation of a TF index and its matching array type is *address_vector_ty*. Both are defined in package *fft_global_types*.

Module *tfc* is fed with *address_ty* data, a bit indicating the transform type and the external enable signal. Both the proposed implementations of Alg. 1 are architectures of this module. Signal *ready_for_new_address_s* is set to one each time new addresses are needed. When the computation of the a set of TFs is complete, signal *valid_o* is driven to one, and in the following clock cycle results can be found in array *dataout_o*.

There are two possible architectures for module *tfc*: *tfc_lut*, which is illustrated in the following section, and *tfc_generator*, whose detail are shown in chapter 9.3.

9.2.2 LUT-based TF generation

Datapath

The datapath of architecture *tfc_lut* is shown in Fig.9.11. The *tfc_mapper* block receives the requested addresses (i.e. indexes) of TFs. Each index i satisfies

$$0 \leq i \leq \frac{N}{2} - 1 \tag{9.5}$$

and matches a complex number in the third and fourth quadrant of Gauss' plane. As a consequence, it is possible to *map* any twiddle factor to the circular segment of angles $0 \leq \phi \leq \frac{\pi}{4}$ by using simple trigonometry rules. The output of the module is so a *reduced_address_ty* value, which is an integer value expressed with less bits than the *address_ty*, and a *mapping_code_ty* value. This is a three-digit code such that:

- if the phase of the mapped TF and the number of digits for its representation are so small that it is undistinguishable from TF of index 0, then the first digit is 1. Else, it is 0;
- the remaining two digits enumerate the original $\frac{\pi}{4}$-sized sector the TF is mapped from.

With this encoding, it is possible to perform the mapping and storing just the TFs of indexes $i = 0, \ldots, \frac{N}{8}$ in module *lut_rom*. Moreover, considering the number of digits for TF

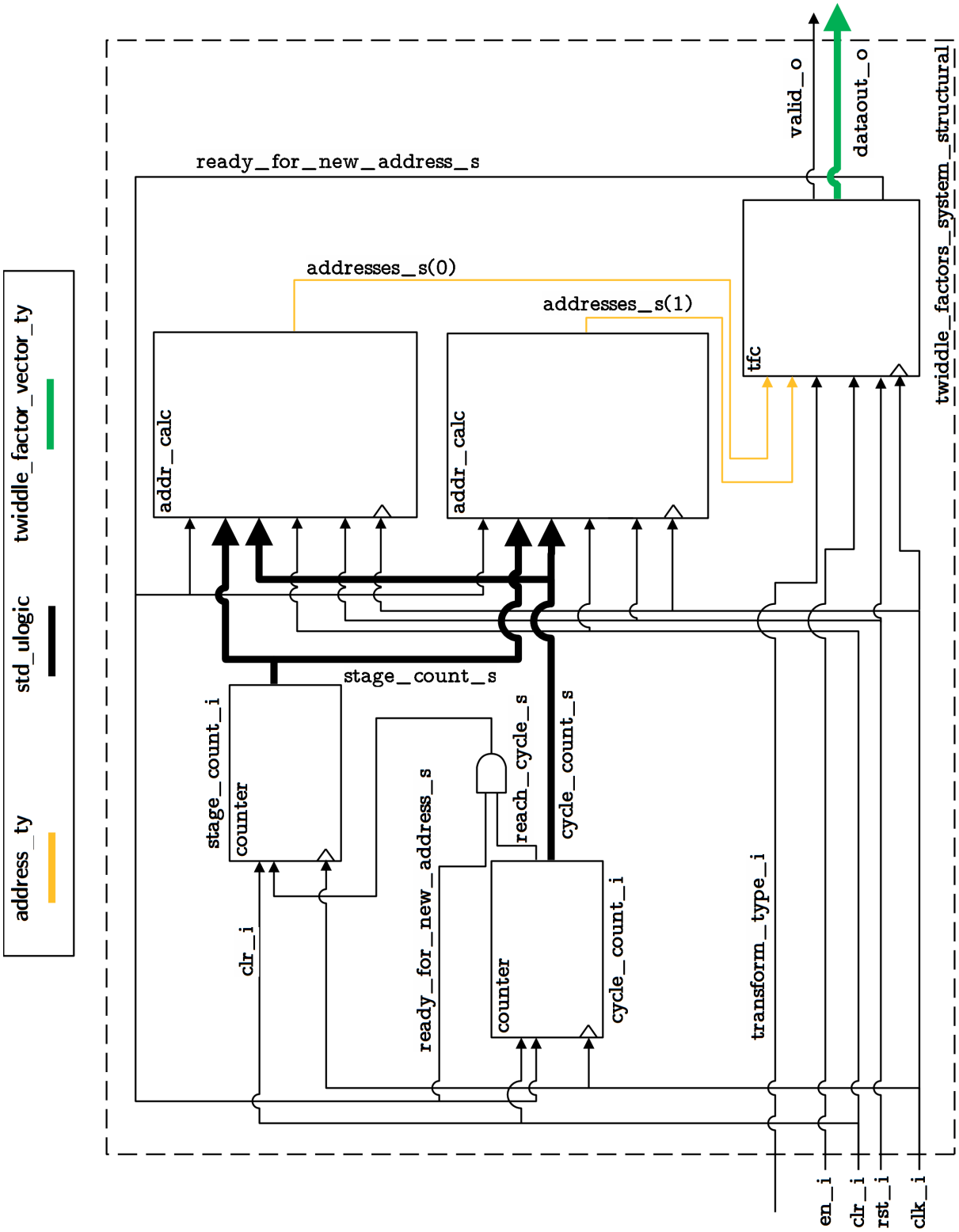

Figure 9.10: Structural diagram of architecture *twiddle_factors_structural*, $P = 2$.

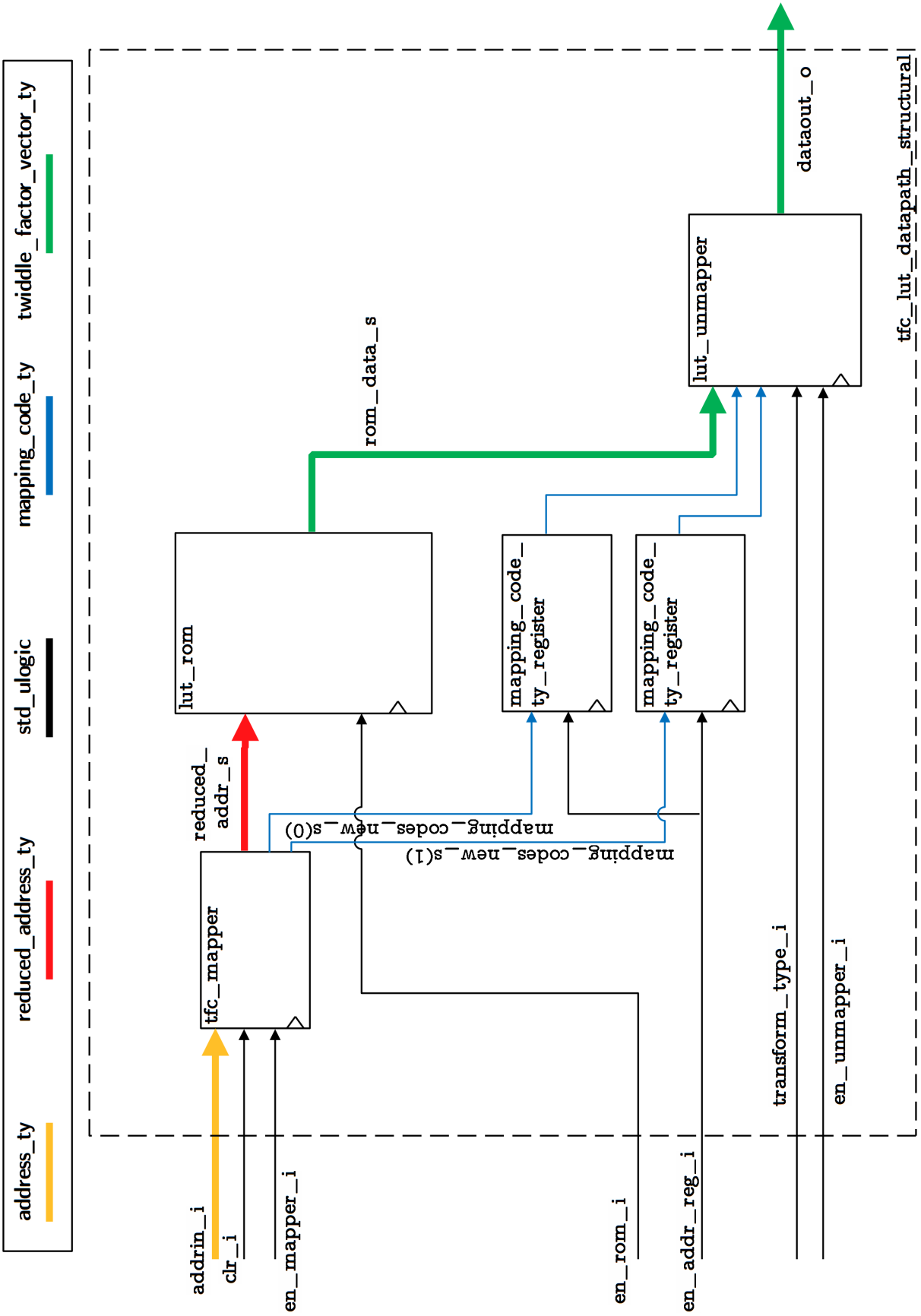

Figure 9.11: Structural diagram of *tfc_datapath_structural*, $P = 2$. Clock and asynchronous reset are omitted.

representation allows the output of the module to identical to the one of a naive $\frac{N}{2}$-sized LUT. There are two possible architectures for this ROM: a *behavioral* implementation and a *bram* version, which exploits BRAM blocks of FPGAs to have multiple inputs as well as multiple outputs.

The *mapping_code_ty* registers synchronise inputs to module *lut_unmapper*, which simply performs the inverse mapping of TFs.

Control

The control of architecture *tfc_lut* is made by a simple Moore machine that allows to reach a throughput of P twiddle factors per clock cycle, where P is the number of PEs. Two states (*mapping* and *load_first*) are necessary to fill the pipeline in the very first stage of computation.

9.3 Hybrid CORDIC-LUT twiddle factor generator

Classic approaches to FFT designs require a large amount of memory for storing precomputed twiddle factors. The CORDIC iterative algorithm presented in [115] allows computation of twiddle factors at runtime. To this purpose many architectures introduce this algorithm inside the PEs by substituting the complex multipliers with iterative phase rotations. With this approach, that we call *in-processor rotator*, in [129] the number of iterations has been reduced by using optimized sequences and corresponding scale factors both stored in a LUT. In [121] a multi-bank RAM structure to reduce memory logic is presented. On the other hand, some systems replace the storage ROM with a CORDIC-based *twiddle factor generation* system [3]. Nonethless, CORDIC hardware implementations can be very expensive in terms of area usage [10, 33], consequently hardly suitable for a scalable FFT approach.

In this chapter we discuss two novel hybrid CORDIC-LUT systems designed to generate twiddle factors in FFTs. In the next section mathematical considerations are drawn on the addressing scheme for twiddle factors in CG-FFT, in order to reduce the dimension of the considered set of TFs. In section 9.3.1 a novel algorithm to compute twiddle factors taking advantage of PE-scalability is introduced. In the further sections two PE-scalable CORDIC architectures are presented with their VHDL implementation.

9.3.1 Properties of the CG-FFT

An important feature of radix-2 DIF CG-FFT is the sequence of twiddle factors in the flowgraph in Fig. 9.12. In general, the twiddle factor of index i is defined as in expression (B.10), here reported:

$$W_N^i = e^{-j\frac{2\pi}{N}i}, \tag{9.6}$$

where N is the number of input points to the FFT. i ranges from 0 to $\frac{N}{2}-1$. In Radix-2 implementations, the FFT is computed in $\log_2 N$ stages, indexed considering s ranging from 0 to $\log_2 N-1$. For each stage $\frac{N}{2}$ twiddle factors must be considered. If the base algorithm is radix-2 DIF the sequence of indexes $\mathscr{I}_s$ is the following:

$$\mathscr{I}_s = \{\underbrace{0,\ldots,0}_{2^s},\underbrace{1\cdot 2^s,\ldots,1\cdot 2^s}_{2^s},\underbrace{2\cdot 2^s,\ldots,2\cdot 2^s}_{2^s},\ldots\}, \tag{9.7}$$

and can be summarized in $\mathscr{I}_s = \{i^{(0)}, \ldots, i^{(N/2-1)}\}$. The derivation for radix-2 DIT is similar, and can be obtained by using index $s' = \log_2 N - 1 - s$: the last sequence in the DIF case will be the first of the DIT and vice-versa for the first one.

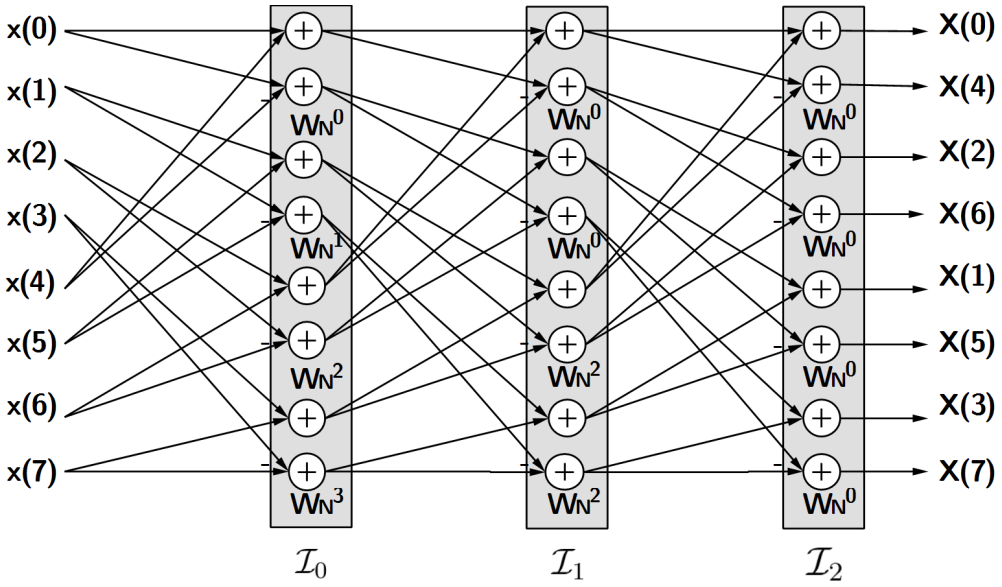

Figure 9.12: Flowgraph of an 8-point Radix-2 DIF CG-FFT with underlined $\mathscr{I}_s$ sequences

Let us define

$$P \in \{x \in \mathbb{N} \mid x = 2^y, y \geq 0\} \tag{9.8}$$

equal to the number of parallel PEs, which are enumerated by using index p ranging from 0 to $P-1$. The twiddle factor generation system must output P values at the same instant, one for each PE. Consequently, as discussed in section 9.1.1, every stage s is split in $C = \frac{N}{2P}$ cycles, indexed as $c = 0, \ldots, C-1$, and similarly sequence $\mathscr{I}_s$ is divided into C subsequences $\mathscr{J}_{sc}$ of indexes:

$$\mathscr{I}_s = \{\mathscr{J}_{s0}, \ldots, \mathscr{J}_{sc}, \ldots, \mathscr{J}_{s(C-1)}\}, \tag{9.9}$$

where $\mathscr{J}_{sc} = \{i_{sc0}, \ldots, i_{scp}, \ldots, i_{sc(P-1)}\}$. The twiddle factor needed on step c of stage s by the p-th PE has index

$$i_{scp} = \left\lfloor \frac{p + cP}{2^s} \right\rfloor 2^s. \tag{9.10}$$

The number of considered indexes can be reduced from $\frac{N}{2}$ to $\log_2 N - 2$, by taking advantage of the following observations.

Observation 9.3.1. *Consecutive elements $i^{(k)}$ in $\mathscr{I}_s$ are either the same index or differ by 2^s.*

Proof. Two consecutive elements in $\mathscr{I}_s$ can be expressed either by

$$\{i_{scp}, i_{sc(p+1)}\} \qquad p = 0, \ldots, P-2 \tag{9.11a}$$

$$\{i_{sc(P-1)}, i_{s(c+1)0}\}. \tag{9.11b}$$

In the first case, the elements belong to the same subsequence $\mathscr{J}_{sc}$. If we write

$$\frac{p + cP}{2^s} = m + \gamma \qquad m \in \mathbb{N}, \quad 0 \leq \gamma < 1$$

we obtain, by substituting in (9.10),

$$\begin{aligned} i_{sc(p+1)} - i_{scp} &= \left(\left\lfloor \frac{p+1+cP}{2^s} \right\rfloor - \left\lfloor \frac{p+cP}{2^s} \right\rfloor \right) 2^s \\ &= \left(\left\lfloor m+\gamma+\frac{1}{2^s} \right\rfloor - \left\lfloor m+\gamma \right\rfloor \right) 2^s \\ &= \left(m + \left\lfloor \gamma+\frac{1}{2^s} \right\rfloor - m \right) 2^s \\ &= \left\lfloor \gamma+\frac{1}{2^s} \right\rfloor 2^s. \end{aligned}$$

Being $0 \leq \gamma < 1$ and $\frac{1}{2^s} \leq 1$, statement 9.3.1 holds.

We now consider the case in Eq. (9.11b), corresponding to the last element of subsequence $\mathscr{J}_{sc}$ and the first of subsequence $\mathscr{J}_{s(c+1)}$. If we write

$$\frac{(c+1)P-1}{2^s} = m' + \gamma' \qquad m' \in \mathbb{N}, \quad 0 \leq \gamma' < 1,$$

we have similarly

$$\begin{aligned} i_{s(c+1)0} - i_{sc(P-1)} &= \left(\left\lfloor \frac{(c+1)P}{2^s} \right\rfloor - \left\lfloor \frac{P-1+cP}{2^s} \right\rfloor \right) 2^s \\ &= \left(\left\lfloor \frac{(c+1)P-1+1}{2^s} \right\rfloor - \left\lfloor \frac{(c+1)P-1}{2^s} \right\rfloor \right) 2^s \\ &= \left(\left\lfloor m'+\gamma'+\frac{1}{2^s} \right\rfloor - \left\lfloor m'+\gamma' \right\rfloor \right) 2^s \\ &= \left(m' + \left\lfloor \gamma'+\frac{1}{2^s} \right\rfloor - m' \right) 2^s. \end{aligned}$$

This leads to

$$i_{s(c+1)0} - i_{sc(P-1)} = \left\lfloor \gamma'+\frac{1}{2^s} \right\rfloor 2^s = \begin{cases} 0, & \text{if } \gamma'+\frac{1}{2^s} < 1, \\ 2^s, & \text{if } \gamma'+\frac{1}{2^s} \geq 1. \end{cases}$$

□

Observation 9.3.2. *The first elements i_{sc0} of each subsequence $\mathscr{J}_{sc}$ (which represent inputs to the first PE) can be the same or differ by a power of 2.*

Proof. To prove this, we proceed similarly to the proof of observation 9.3.1 and write

$$\frac{cP}{2^s} = m + \gamma \qquad m \in \mathbb{N}, \quad 0 \leq \gamma < 1.$$

Then, we manipulate the following expression

$$\begin{aligned} i_{s(c+1)0} - i_{sc0} &= \left(\left\lfloor \frac{(c+1)P}{2^s} \right\rfloor - \left\lfloor \frac{cP}{2^s} \right\rfloor \right) 2^s \\ &= \left(\left\lfloor m+\gamma+\frac{P}{2^s} \right\rfloor - \left\lfloor m+\gamma \right\rfloor \right) 2^s \\ &= \left(m + \left\lfloor \gamma'+\frac{P}{2^s} \right\rfloor - m \right) 2^s \\ &= \left\lfloor \gamma+\frac{P}{2^s} \right\rfloor 2^s. \end{aligned} \tag{9.12}$$

Substituting Eq. (9.8) in (9.12) we have

$$i_{s(c+1)0} - i_{sc0} = \left\lfloor \gamma + \frac{P}{2^s} \right\rfloor 2^s = \lfloor \gamma + 2^{y-s} \rfloor 2^s. \tag{9.13}$$

If $y \geq s$ the result is 2^y, which is of course a power of two. If $y < s$ we have

$$i_{s(c+1)0} - i_{sc0} = \begin{cases} 0, & \text{if } \gamma + 2^{y-s} < 1 \\ 2^s, & \text{if } \gamma + 2^{y-s} \geq 1. \end{cases}$$

This confims that observation 9.3.2 is correct. □

Observation 9.3.3. *For every i, W_N^i is a complex number with unitary module falling either in the third or fourth quadrant of the Gauss plane. With basic trigonometry all $\frac{N}{2}$ different twiddle factor values necessary during computation can be calculated using indexes in the set $M = \{0, \ldots, \frac{N}{8}\}$.*

The first remark is immediate from Eq. (9.6), while the considerations on set M can be drawn by observing Fig. 9.13. We can notice that twiddle factor $W_N^{\frac{N}{8}}$ corresponds to an angle of $-\frac{\pi}{4}$.

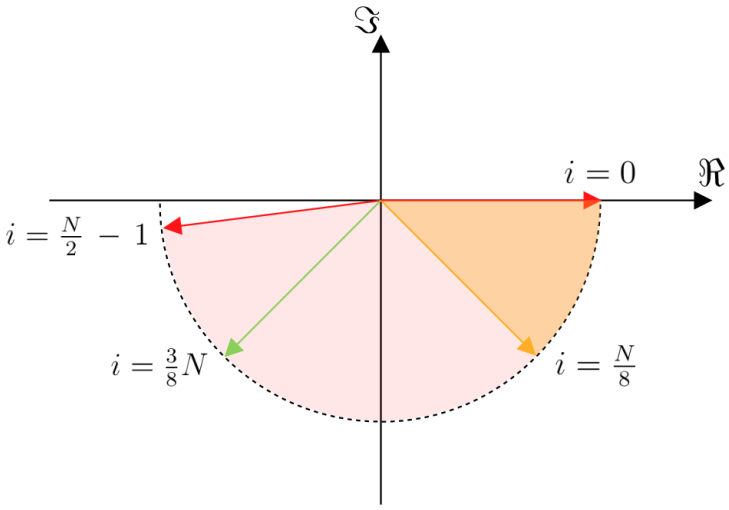

Figure 9.13: Distribution of twiddle factors in the Gauss'plane.

Observation 9.3.4. *Multiplication by W_N^i is an angular rotation of $-j\frac{2\pi}{N}i$.*

This is well known from the basic theory of complex numbers.

As a consequence of the forementioned observations, the smallest set of indexes that will be considered in the derived models is the subset $M' \subset M$

$$M' = \{m = 0 \vee m = 2^x, x = 0, 1, \ldots, \log_2 N - 3\}. \tag{9.14}$$

In other terms, we consider only TFs with an index that is equal to a power of two, plus twiddle factor W_N^0, in the first quarter of the negative half circumference.

Scalable rotational algorithm

We present here a scalable rotational algorithm for computing the twiddle factor sequence $\{W_N^{(k)}\}$. The procedure, illustrated in Alg. 1, requires a set $W(M')$ of precomputed twiddle factors with indexes in M'.

Algorithm 1 The proposed scalable rotational algorithm.

Require: $\mathscr{I}_s, N, C, P, W(M')$
  $k \leftarrow 0$
  for $c = 0$ to $C-1$ **do**
    for $p = 0$ to $P-1$ **do**
      $q_{scp} \leftarrow i_{scp}$ reflected in $\{0, \ldots, \frac{N}{8}\}$
5: **if** $c = 0$ **and** $p = 0$ **then**
        $l \leftarrow 0$
      else if $p = 0$ **then**
        $l \leftarrow q_{s(c-1)(P-1)}$ or $l \leftarrow q_{s(c-1)0}$
      else
10: $l \leftarrow q_{sc(p-1)}$
      end if
      if $q_{scp} \in M'$ **then**
        $W_N^{(k)} \leftarrow W(q_{scp})$
      else if $q_{scp} = l$ **then**
15: $W_N^{(k)} \leftarrow W_N^{(k-1)}$
      else {CORDIC iteration}
        $\alpha \leftarrow \pm(q_{scp} - l), |\alpha| \in M'$
        $W_N^{(k)} \leftarrow W_N^{(k-1)} \cdot W(\alpha)$
      end if
20: $W_N^{(k)} \leftarrow W_N^{(k)}$ reflected in $(-\pi, 0)$
      $k \leftarrow k+1$
    end for
  end for
Ensure: $\{W_N^{(k)}\}$.

In most approaches, CORDIC algorithms need precomputed scale factors and converge towards results in a number of steps depending on the resolution bits B [115, 129]. Taking advantage of statement 9.3.4, each element of the twiddle factor output sequence is obtained in only one iteration, without the need of any gain correction. Also, observation 9.3.3 has been used in line 4, in order reduce the number of indexes to $\frac{N}{8}+1$. Moreover, exploiting considerations given in 9.3.1, the presented algorithm (lines 12 and 14) does not require any arithmetic operation both when reflected index q_{scp} belongs to M' and when q_{scp} is equal to the index l of the previous step.

We distinguish between two versions of the same algorithm: if we decide to exploit the observation given in 9.3.1 then $l \leftarrow q_{s(c-1)(P-1)}$, otherwise considering 9.3.2, $l \leftarrow q_{s(c-1)0}$.

9.3.2 Overall architectural remarks

The architectures presented in this section are twiddle factor generation modules implementing the two versions of Alg. 1 respectively. Both these modules receive P parallel un-

signed signals corresponding to subsequences $\mathscr{J}_{sc}$ and output P twiddle factors belonging to sequence $\{W_N^{(k)}\}$ in signed fixed-point format.

We refer to *shared core* (Fig. 9.14) when $l \leftarrow q_{s(c-1)(P-1)}$. This means that the computation on subsequence $\mathscr{J}_{s(c-1)}$ must be completed before the start of calculations on $\mathscr{J}_{sc}$.

When $l \leftarrow q_{s(c-1)0}$ the first element of subsequence $\mathscr{J}_{sc}$ can be processed before the end of calculations on subsequence $\mathscr{J}_{s(c-1)}$, using only the data from the computation of the first element ($p = 0$) of $\mathscr{J}_{s(c-1)}$. This leads to another architecture, referred to as *pipelined* (Fig. 9.18), which parallelizes the for loop on line 2 of the algorithm by using different pipeline stages.

Alg. 1 can require to perform a rotation by using twiddle factors resulting from CORDIC iterations themselves. This can be an issue in terms of error propagation. In order to compensate this effect, we perform internal operations by using $B' \geq B$ bits and then rounding to B-bit words, where B is the resolution of the output twiddle factors. To this purpose, a new data type, *high_res_twiddle_factor_ty* is defined in package *cordic_types* as follows:

```
subtype high_res_twiddle_factor_part_ty is
  u_sfixed( global_tf_accuracy_int_c-1  downto
          -cordic_high_res_tf_frac_accuracy_c);
```

Constant *cordic_high_res_tf_frac_accuracy_c* is defined in package *fft_cordic_settings* and its default value is 17.

Referring to the structure of module *twiddle_factors_system* described in the previous chapter, both twiddle factor generators are included in architecture *tfc_generator*. This is made of two modules: *tfc_mapandstep* and *tfc_cordic*.

tfc_mapandstep

This module performs mapping operations and computes the difference between indexes for CORDIC iterations. The component blocks are:

- Module *tfc_mapper*, which realizes the first step of Alg. 1 (line 4). This is in order to output the mapping of the incoming requested array of P indexes to the set M, representing the portion $(-\frac{\pi}{4}, \ldots, 0)$ of the complex plane.
- Block *tfc_step_computer_pipeline*, that analyses the output array of *tfc_mapper* with the purpose of determining α (line 17 of Alg. 1). That is the ROM address where *step data* is stored, needed to compute the CORDIC iteration.
- *mapping_code_ty_registers* behave like described in the previous chapter, and synchronise outputs of *tfc_mapandstep*.
- A counter, which keeps trace of how many subsequences are currently being processed in the module.

The whole *tfc_mapandstep* is a pipeline structure, which must be filled at the beginning of computation. With this technique the downstream *tfc_cordic* module can *pop* a subsequence out of it whenever it needs, by just driving signal *ready_for_new_address_i*.

The two hybrid TF generators are architectures of module *tfc_cordic*.

9.3.3 Hybrid CORDIC-LUT architecture *shared-core*

Datapath

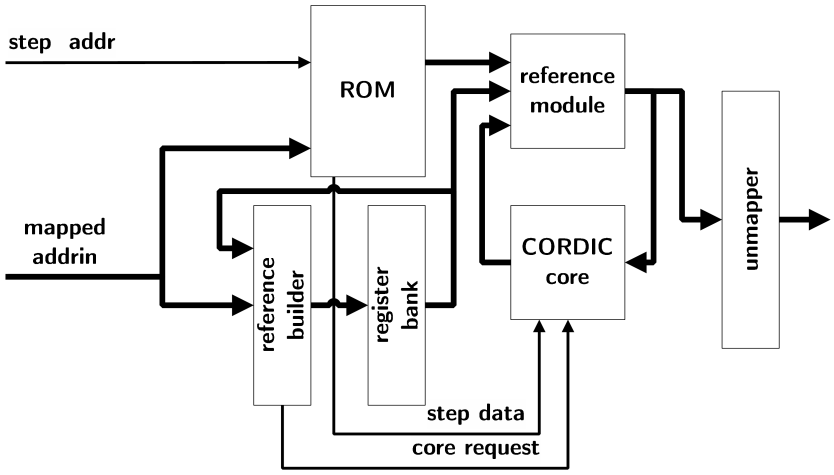

Figure 9.14: Functional diagram of the *shared core* scalable CORDIC architecture

Fig. 9.14 illustrates the hardware structure implementing *shared_core* with a simple block diagram. In this section, we will describe the general behaviour of each module before giving a detailed analysis in terms of VHDL architectures.

Then *reference builder* scans input addresses comparing every value with the preceeding ones, determining if this array needs computing of a new twiddle factor or not. If computation is not necessary, data can be immediately retrieved because either the twiddle factor has already been obtained by CORDIC iteration or it belongs to $W(M')$. The implemented procedure is formalized in Alg. 2, where $\boldsymbol{a}$ is a P-sized array of requested TF indexes, i is the the number of the current request, l is the P-th request of the past clock cycle. In the realised design r is called *reference* and s is the *core request* signal. *Reference builder* outputs to *register bank* the information for selecting data used by *reference module*. Request of CORDIC core usage is done by propagating signal *core request*. This is high when all the following conditions occur at the same time:

- the index of the current called TF is not in M',
- the index of the current called TF is not equal to the last request l (line 14 of Alg. 2)
- no other request before the current involves the same TF.

The $P+1$ port ROM stores the set $W(M')$: it outputs a nonzero value only when the address belongs to M'. Implementing this module is an issue on FPGAs: the majority of them contains only dual-port block RAMs, also called BRAMs in Xilinx products. Synthesizing a bare multi-port RAM consequently results in expanding the memory into many slices, with an overhead in terms of resource usage. The adopted solution is to pack couples of requests and direct each of them to a BRAM port, consequently expecting the synthesizer to instantiate $\frac{P}{2}+1$ memory blocks. On the other hand implementation on ASIC technology does not imply any particular expedient.

Algorithm 2 Building of references and request of CORDIC iteration.

Require: $\boldsymbol{a}, i, l, P, M'$
  $k \leftarrow i$
  while $a_{k-1} = a_i$ **and** $k > 0$ **do**
    $k \leftarrow k - 1$
  end while
  if $k = 0$ **then**
    if $a_i = l$ **then**
      $r \leftarrow -1$
    else
      $r \leftarrow k$
    end if
  else
    $r \leftarrow -1$
  end if
  $s \leftarrow (a_i = l)$ **or** $(a_i = 2^x, x \in \mathbb{N})$ **or** $(a_i = 0)$
  $s \leftarrow (r = k)$ **and not** s
Ensure: r, s

Combinatorial block *reference module* selects data from the ROM or from CORDIC results and manages the case of consecutive same-valued indexes (line 14 of Alg. 1) by reading reference information in *register bank*. The behaviour of this module can be analysed in more detailed terms by considering VHDL Lst. 9.4. A first set of multiplexers is driven by a signal coming from the control, *sel_data_i*, selecting between data from the memory or from the CORDIC core. A second set of multiplexers solves the references: each of them outputs the data pointed by signal *curr_references_i* (whose value is r from Alg. 2). Consequently, if the reference for request p has value p, data is propagated directly from the previous multiplexer. Else, if the reference has value $k \neq p$, data is taken from the result of the k-th request.

The CORDIC core reads the *core request* signal. Then, it serves the pending requests by using a complex multiplier, composed of three real multipliers. Initialization data is taken from the output of *reference module* and the product is performed by using *step data* according to line 18 of Alg. 1. The structural diagram of the datapath is illustrated in Fig. 9.15. Module *cordic_fifo* implements a classic FIFO queue and is loaded when *fifo_ready* is driven to high by the *cordic_core* control. Elements inside the queue are in increasing order of request number (i.e. p). Starting from the next clock cycle, elements of *cordic_fifo* are output one by one to *caller_s*. Depending on the value of this signal, the multiplexers select the data to initialise the CORDIC rotation following line 18 of Alg. 1, and input signal *step_sign_i* determines the direction of rotation. Moreover, de-multiplexers select an output register for writing and set a bit of output array *cordic_end_o* to one. This signals to other modules that computation is complete and that starting from the following clock cycle data will be available in the matching output register.

When computations are executed, twiddle factors feed block *unmapper* reflecting the obtained results to the third and fourth quadrant of the complex plane. This block implements line 20 of Alg. 1. Furthermore, it converts TF data from the B'-bits format to the B-bit format. By default, this is performed by rounding and not by truncation, thus increasing the accuracy of obtained results. The used routines are part of the *fixed_pkg* IEEE VHDL

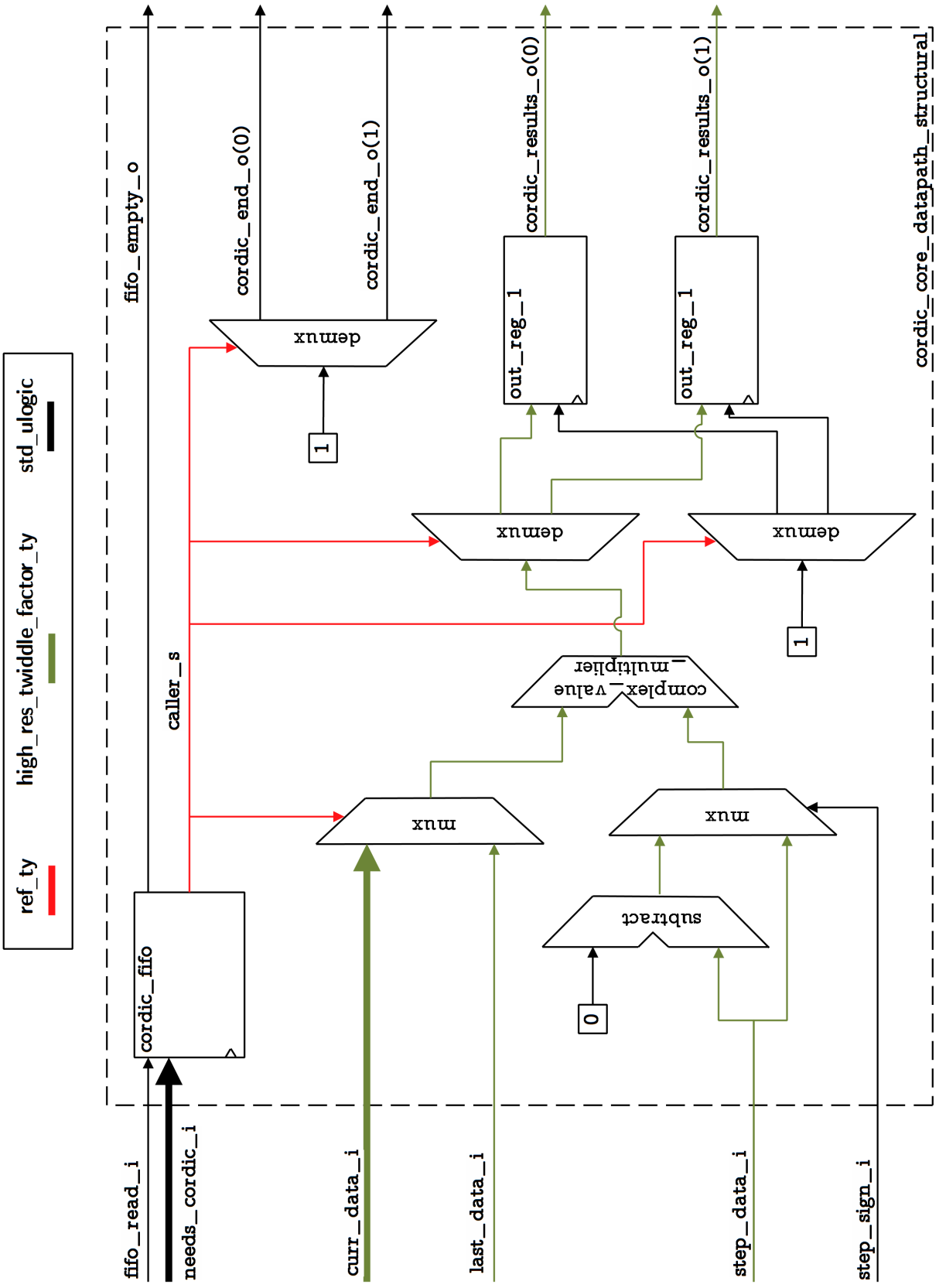

Figure 9.15: Structural diagram of module *cordic_core*. $P = 2$. Clock and asynchronous reset are omitted for simplicity.

Listing 9.4: Behavioral architecture of *reference_module*.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.fft_global_settings.all;
use work.math_utils.all;
use work.fft_types.all;
use work.cordic_types.all;

architecture  cordic_reference_module_structural of
              cordic_reference_module is
  signal memory_or_cordic_data_s
    : high_res_twiddle_factor_vector_ty(0 to global_pes_c-1);
begin

  -- Choosing between data coming from memory or from the
  -- CORDIC core
  memory_cordic_mux_gen : for i in 0 to global_pes_c-1 generate
    with sel_data_i(i) select
      memory_or_cordic_data_s(i)  <=
        mem_data_i(i)                    when '0',
        cordic_results_i(i)              when '1',
        high_res_twiddle_factor_zero_c when others;
  end generate;

  -- Solving references
  references_mux_gen : for i in 0 to global_pes_c-1 generate
    curr_data_o(i)  <=
      memory_or_cordic_data_s(
        to_integer(signed(curr_references_i(i))))
      when (signed(curr_references_i(i))>=0) else last_data_i;
  end generate;

end architecture cordic_reference_module_structural;
```

package. If the *transform_type_i* bit is set to one, then module *unmapper* also computes the conjugate of the TFs by inverting their imaginary part.

Control

The control of the implemeted *cordic_core* architecture is realised by P Moore machines, each one matching a request of TF computation. In order to illustrate its mechanism, we consider Fig. 9.16, picturing the complete implemented datapath, and Fig. 9.17 showing a dataflow diagram of the control.

When the overall computation of the FFT starts, all the Moore machines are in state *idle*. Then, the control of the processor sets signal *en_twiddle_calc_i* to high, and a transition occurs for all the machines towards state *load_addr*. In this state, all the controls set a single bit signal, *en_addreg_o*, to one, requesting to sample data in the *cordic_ref_ty* and *cordic_mapping_cod_ty* registers. Moreover they request the enabling of the ROM. The register bank in the simplified diagram of Fig. 9.14, is implemented by registers *Cordic_ref_ty* and *cordic_mapping_cod_ty*.

When all Moore machines are in state *load_addr*, signal *tot_en_addreg_i*, which is obtained by a bitwise *and* of all *en_addreg_o* signals, is driven to one. Complete information about the next steps are thus available: if a request needs usage of module *cordic_core* this is initialised, if data is available from another request the proper reference has been built.

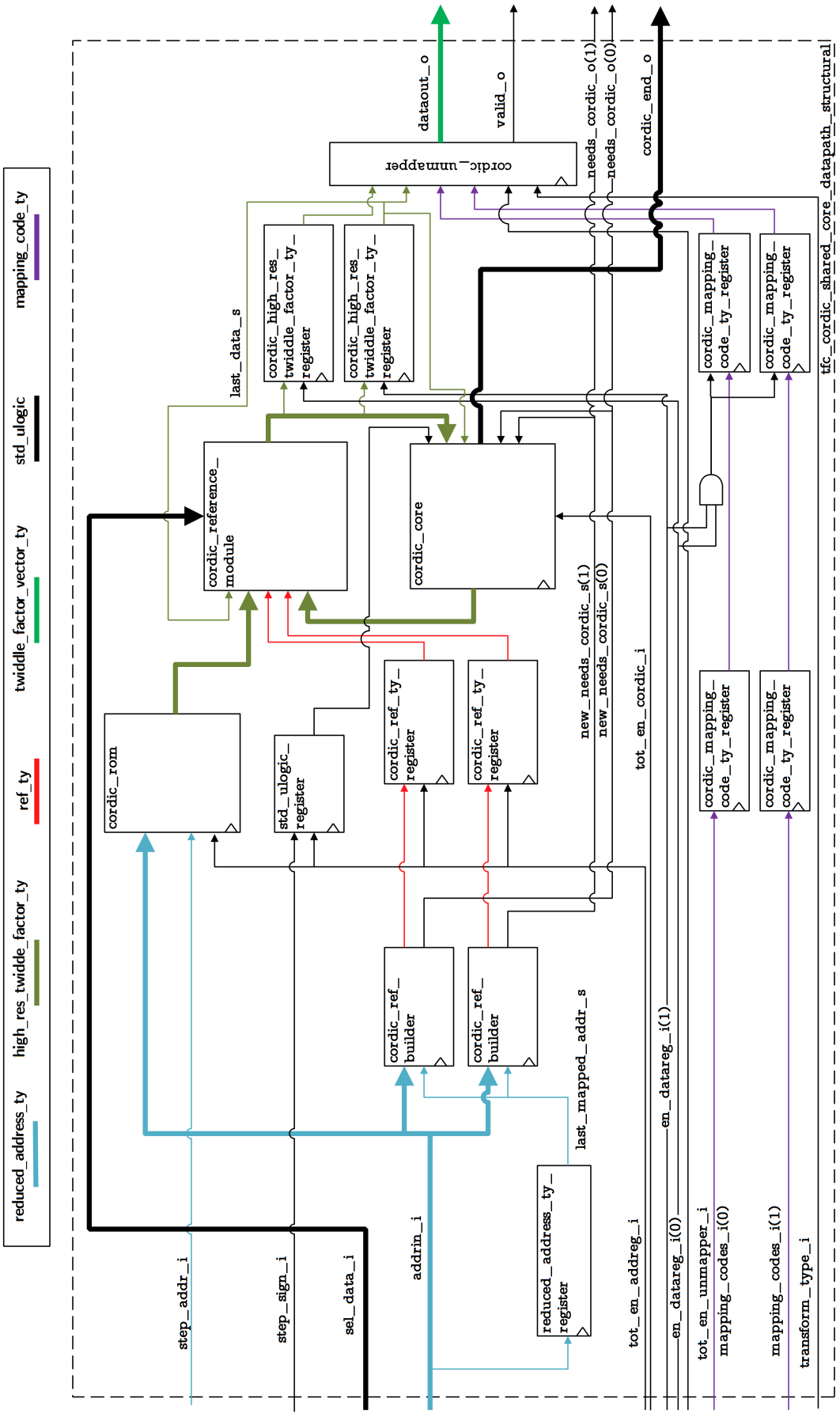

Figure 9.16: Structural diagram of the datapath of *shared core* with $P = 2$. Clock and asynchronous reset are omitted for simplicity.

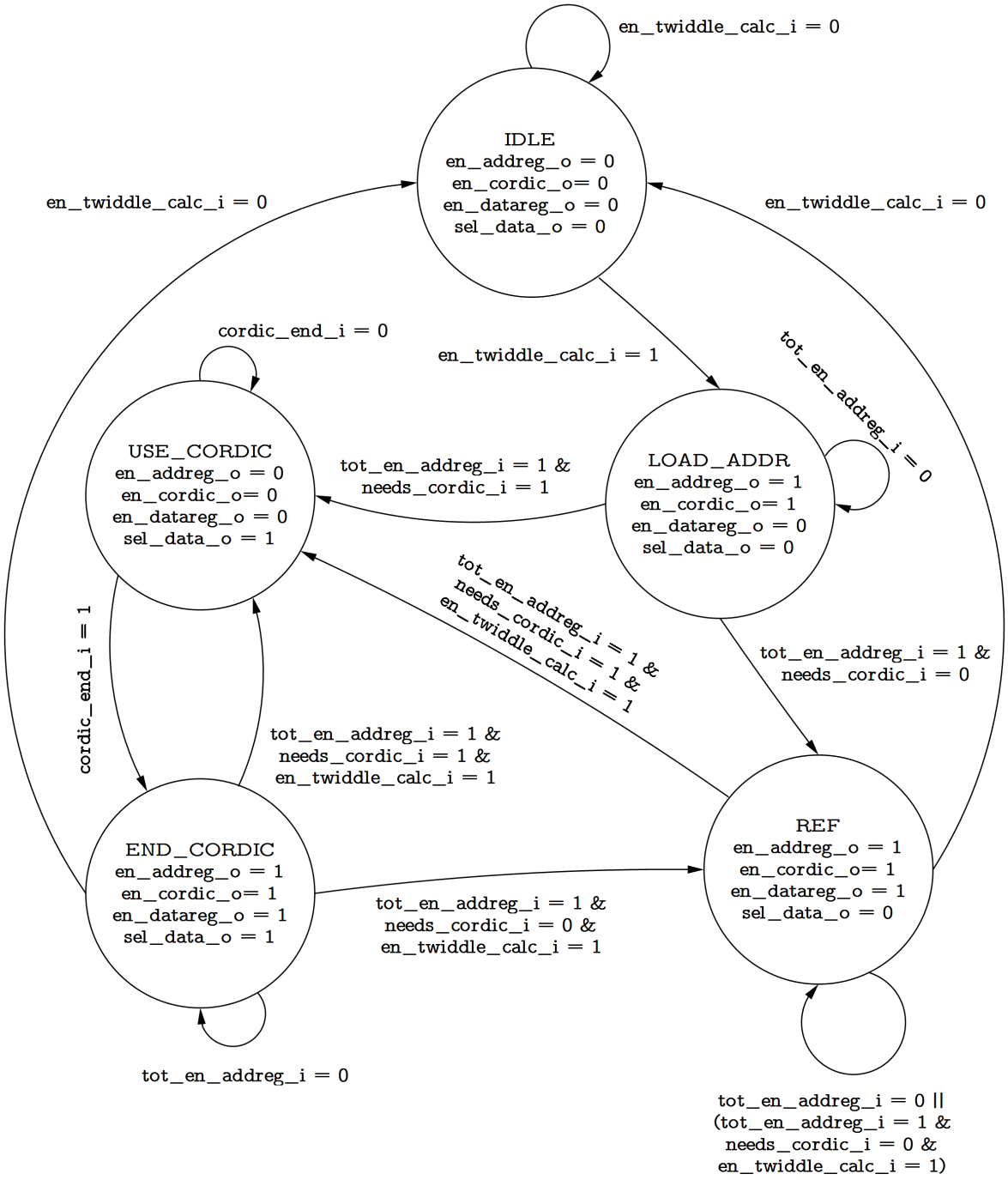

Figure 9.17: Dataflow diagram of the *shared core* control.

Thus, depending on the request type, each machine reaches state *use_cordic* or *ref*. In the first case, it waits until the CORDIC iteration is complete, by listening to single bit signal *cordic_end_i*. When data is ready, state becomes *end_cordic*, and the computed TF flows through the *reference_module* (Lst. 9.4) according to signal *sel_data_o*. In the case of state *ref* no operation is performed. The machine simply requests sampling of data at the output of *cordic_reference_module*, in its matching *cordic_high_res_twiddle_factor_ty_register* (see Fig. 9.16).

At the same time, both in the *end_cordic* and *ref* states, a new sampling of addresses and references is enabled, allowing a continous chain of request serving. The *needs_cordic_i* signal specifies the new state for the transition so that no clock cycle is wasted for interpretation of new requests.

9.3.4 Hybrid CORDIC-LUT architecture *pipelined*

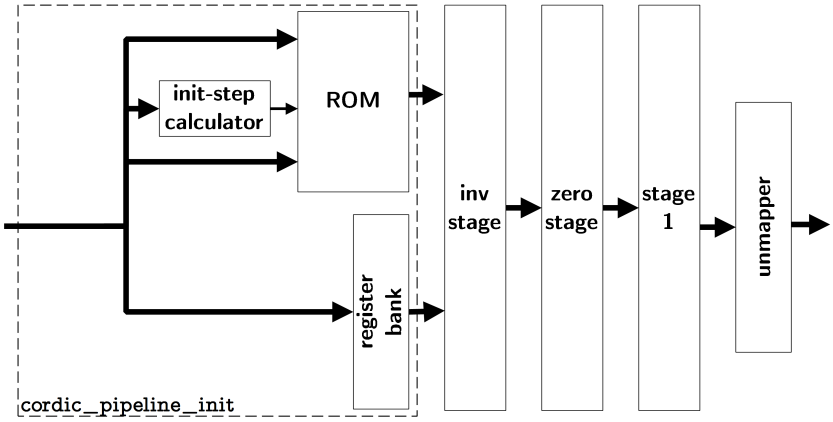

Figure 9.18: Functional diagram of the *pipelined* scalable CORDIC architecture. $P = 2$.

The structure of architecture *pipelined* is illustrated in Fig. 9.18. Modules *mapper*, *step calculator* and *unmapper* are the same used in architecture *shared core*. The main difference between the two solutions is that in the *pipelined* one the mechanism of *references* is unnecessary. Consequently synchronisation of modules and control is much easier. Moreover, the *register bank* is less hardware expensive if compared to the implementation given in Fig. 9.16. This is because no register is needed to store obtained references, but it is always necessary to propagate the proper information for TF unmapping.

Module *init_step_calculator* computes the rotational step considering only the first elements of adjacent $\mathcal{J}_{sc}$ request arrays. That is, it computes α using $l = q_{s(c-1)0}$ in Alg. 1 by taking advantage of observation 9.3.2.

Another element of difference between the *pipelined* and the *shared core* structures is in the ROM. In this case a $(P+2)$-port block is needed. The implementation considerations remain the same of the former solution, both for FPGA and ASIC technologies.

The actual pipeline structure is composed of $P+2$ stages. The first one, called *inv stage*, performs the inversion of the imaginary part of the step data (α in Alg. 1) used in CORDIC iterations, depending on the direction of rotation. This operation was performed inside the *cordic core* module in the *shared core* architecture. Being the *pipelined* solution expressly

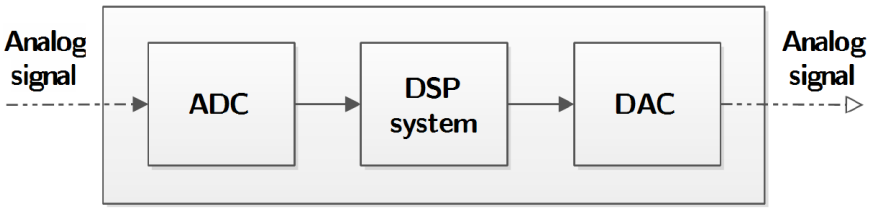

Figure 9.19: Digital Signal Processing model.

oriented towards higher throughput applications, it is separated from CORDIC iterations. With this expedient we expect to achieve an increase of the operating clock frequency.

Zero stage serves the requests for $p = 0$, by either reading data propagated from the ROM or performing rotations. In the latter case, step data conforms to the address computed by *init-step calculator*. The behaviour of the following stages is similar, but each stage matches a different value of p, from 1 to $P-1$. For these stages iteration step data accords to the output of *step calculator*. Following Alg. 1, three events can occur:

1. CORDIC iteration is needed. The iteration is performed from the TF data of the last stage by using a *complex_value_multiplier*, module whose code is in Lst. 9.2. Consequently, each stage block contains three real multipliers and the total number of multipliers expected to be synthesized is $3P$.

2. The requested TF has the same index as the one in the last stage. If p is the number of the current served request, then data is selected from request $p-1$, and propagated to the following stage.

3. The index of the current requested TF is in set M'. Data is directly propagated without any change because it was already retrieved in the ROM.

Similarly to architecture *shared core*, the last pipeline stage is the *unmapper* block.

The control of each stage is easily implemented with a two state Moore machine. In the first state, called *idle*, the module simply waits to become functional. When it is enabled by the previous stage, it reaches state *comp*. Here, it enables the internal registers of the block and propagates a *valid_o* signal, in order to enable the following stage.

The two implemented *shared core* and *pipelined* solutions differ from the FFT CORDIC approaches in the state of the art, both because they take into account system scalability in terms of PE and because TF phase rotations are performed without needing any scale factor correction. We expect this to allow an improved scalability in terms of hardware resource usage both in the case of FPGA and ASIC implementations. Nevertheless the choice between the two architectures is expected to condition overall system performances critically. The *shared core* structure has been designed for severely tight hardware usage requirements. On the other hand the *pipelined* architecture takes into account resource efficiency but is thought for achieving improved timing performances. Simulation and synthesis results in a following chapter will formalise our expectations and give evaluation metrics to this purpose.

9.4 PSD computer

The examined processor has been embedded in a PSD computer, with application to the self-tuning algorithm for digital control of switched-mode power supplies (SMPS). These devices perform a conversion of a DC input voltage from one level to another [91]. The general scheme of the system is associated to the conventional DSP problem that is illustrated in Fig. 9.19: the processing of a sampled signal to obtain another signal. In the commissioned design, part of the processing is based on the autocorrelation, which is performed by using a FFT-based technique.

The digital control is implemented on FPGA, as a first development step. Being part of a research project, the PSD computer must guarantee good implementation flexibility. Moreover, the lowest bound of the operating frequency given by the customer is $70MHz$. This value takes into account the final application of the design, which requires low resource usage more than high speed. Simultaneously, computation time must be small enough for the control to achieve a correct response. The implemented PE-scalable architecture has been chosen to fullfil all these requirements, but, being it part of a more complex PSD computer, the design of all the other blocks must follow the same approach. Furthermore, the whole project will migrate to ASIC platform in the future, thus it is necessary for the DSP solution to have good performances when synthesized in both technologies.

In this chapter the implemented autocorrelation PSD computer for the PSD computation is analysed. First, the algorithm is illustrated and some important remarks are drawn. These, lead to the hardware structure which is analysed block by block in the next section. Following subsection 9.4.2 explores the control of the system.

9.4.1 Autocorrelation-based DSP algorithm: PSD computation

The PSD computer evaluates the operation

$$\text{autocorrelation}\left|\text{fft}(\boldsymbol{x})\right|, \tag{9.15}$$

where $\boldsymbol{x}$ is a $(2^{10} \times 1)$ array of real samples. In order to understand the amount of resources that have to be instantiated in the implementation, it is necessary to make some observations on the autocorrelation function.

The processing performed by the PSD computer is related to the PSD computation, it has been shown in Alg. 3. Computations require an array $\boldsymbol{x}$ of 1024 samples and a constant s, which is called *scaling factor* and is needed for hardware reasons. This will be discussed in the next section.

The first stage of the processing is the computation of the absolute value of the FFT of input samples. The second and third stages need calculations on 2^{11}-sized sequences, so it will be necessary to instantiate a 2048-point processor. In order to use this hardware, computation is performed during stage one on the doubled vector $[\boldsymbol{d}_{\text{first}}; \boldsymbol{d}_{\text{first}}]$. This is illustrated in line 2. This results in a sequence interleaved with zero-valued data. Noticing that the output of the processor is bit-reversed, only the first half of the output cycles will contain valid data, so only the first 1024 output complex values are considered. After the computation of the modulus, the resulting real data is scaled and then stored in bit-reversed order (line 6).

In the second processing stage, the 1024 data from the previous stage is input to the processor, immediately followed by 1024 zero values. This implements the zero-padding as discussed in section B.3. After the computation of the square modulus of complex data and

Algorithm 3 Algorithm executed by the PSD computer for the PSD computation.

Require: $\boldsymbol{x} \in \mathbb{R}^{1024}, s$
$\boldsymbol{d}_{\text{first}} \leftarrow \boldsymbol{x}$ {First stage: FFT}
$\boldsymbol{r}_{\text{first}} \leftarrow \text{FFT}([\boldsymbol{d}_{\text{first}}; \boldsymbol{d}_{\text{first}}])$
$\boldsymbol{r}_{\text{first}} \leftarrow [r_{\text{first},0}, \ldots, r_{\text{first},1023}]^\intercal$
$\boldsymbol{r}_{\text{first}} \leftarrow |\boldsymbol{r}_{\text{first}}|$
5: $\boldsymbol{r}_{\text{first}} \leftarrow \boldsymbol{r}_{\text{first}} / s$
$\boldsymbol{d}_{\text{second}} \leftarrow \text{bitrevorder}(\boldsymbol{r}_{\text{first}})$ {Second stage: FFT}
$\boldsymbol{r}_{\text{second}} \leftarrow \text{FFT}([\boldsymbol{d}_{\text{second}}; \boldsymbol{0}])$
$\boldsymbol{r}_{\text{second}} \leftarrow |\boldsymbol{r}_{\text{second}}|^2$
$\boldsymbol{r}_{\text{second}} \leftarrow \boldsymbol{r}_{\text{second}} / s$
10: $\boldsymbol{r}_{\text{second}} \leftarrow \text{bitrevorder}(\boldsymbol{r}_{\text{second}})$
$\boldsymbol{d}_{\text{third}} \leftarrow [r_{\text{second},0}, \ldots, r_{\text{second},1024}]^\intercal$ {Third stage: IFFT}
$\boldsymbol{r}_{\text{third}} \leftarrow \text{IFFT}([d_{\text{third},0}, \ldots, d_{\text{third},1024}, d_{\text{third},1023}, \ldots, d_{\text{third},0}])$
$\boldsymbol{r}_{\text{third}} \leftarrow |\boldsymbol{r}_{\text{third}}|$
$\boldsymbol{r}_{\text{third}} \leftarrow \boldsymbol{r}_{\text{third}} / s$
15: $\boldsymbol{r}_{\text{third}} \leftarrow \text{bitrevorder}(\boldsymbol{r}_{\text{third}})$
$\boldsymbol{y} \leftarrow [r_{\text{second},0}, \ldots, r_{\text{second},1023}]^\intercal$
Ensure: $\boldsymbol{y}$

scaling, only the first 1025 values are stored. This expedient takes advantage of the simmetry of the DFT. Storage is done with bit-reversed ordering (line 11).

The third stage computes the IFFT of stored data. 2048 values are given to the processor by first considering elements from 0 to 1024, and then from 1023 to 0. The modulus of processed data is computed and then scaled. After another bit-reversed ordering, data is ready to be output.

9.4.2 System architecture

The design principles used for the PSD computation are the same used in chapter 9.1. A hierarchy of packages definining constants and types is used ad a structure for coding. Also, naming convention in Tab. 9.1 holds.

Package *autocorrelation_settings* defines all the global constants used in the design. It is comparable to *fft_types* from the FFT processor design. On the counterpart, *fft_types* is encapsulated in the VHDL code of the processor with all the other packages used in the previous design. In order to separate the code of the processor from the design of the autocorrelation-based PSD computer, constants are re-defined in *autocorrelation_settings*. Thus, in case of editing this last package, it is necessary to ensure the correct match of settings in *fft_global_settings*. Among the additional editable constants defined in *autocorrelation_settings* we have

input_sample_int_c Defines the number of bits for the integer part of input data. The default value is 4.

input_sample_frac_c Is the number of bits of for the fractional part of input data, equal to 0 by default.

parallel_words_c Being the processor based on a parallel structure, this value is equal to the number of words in the bus. *global_pes_c* in package *fft_global_settings* must match this value: the number of words propagating in parallel in the bus is equal to the number of PEs of the processor multiplied by two. As a consequence, throughout the rest of this chapter, this parameter will be indicated with $2B$. By default the value of *parallel_words_c* is equal to 8.

scaling_factor_c This natural value is related to the scaling factor s applied during the processing (see Alg. 3) and, by default, this is equal to 7.

Also, considering the discussion in the previous section, package *autocorrelation_types* defines non-editable parameters such as:

ext_address_size_c is the length of the address field used to retrieve data. Given that 1024 data must be addressed, this is fixed to 10.

ram_address_size_c Is the length of the address field in the internal bus. Noticing that 1025 memory locations are addressed, its value is fixed to 11.

The implemented system architecture is based on the same custom types described in chapter 9.1. Although package *autocorrelation_types* re-defines them, it also adds other types. Package *modulus_types* defines other custom data types which are necessary to understand the datapath of the PSD computer. The following list describes the most important new types introduced in this design.

ram_address_ty This data type is defined in *autocorrelation_types* and represents the input address to the RAM. It is defined with the following VHDL code

```
subtype ram_address_ty is
  std_ulogic_vector(ram_address_size_c-1 downto 0);
```

An array of *ram_address_ty* elements is a *ram_address_vector_ty*.

square_complex_part_ty This type is defined in *modulus_types* as a signed fixed-point number. It is sized so that a squared *complex_part_ty* can fit into it without any rounding. It is defined with the following VHDL code, which takes advantage of the IEEE *fixed_pkg* standard package.

```
subtype square_complex_part_ty is
  sfixed(sfixed_high( complex_part_ty'high,complex_part_ty'low,
                '*',
                complex_part_ty'high,complex_part_ty'low)
      downto
      sfixed_low(   complex_part_ty'high,complex_part_ty'low,
                '*',
                complex_part_ty'high,complex_part_ty'low));
```

square_modulus_ty Defined in package *modulus_types*, this is an unsigned fixed-point number representing the square modulus of a *complex_ty* element. Consequently, it is sized to contain the sum of two *square_complex_part_ty* numbers.

```
subtype square_modulus_ty is
  ufixed(ufixed_high( square_complex_part_ty'high,
                square_complex_part_ty'low,
                '+',
                square_complex_part_ty'high,
```

```
            square_complex_part_ty'low)
downto
ufixed_low(   square_complex_part_ty'high,
            square_complex_part_ty'low,
            '+',
            square_complex_part_ty'high,
            square_complex_part_ty'low));
```

An array of *square_modulus_ty* elements is a data of type *square_modulus_vector_ty*.

Datapath

The structure of the datapath is shown in Fig. 9.20. The input to the system is given by a serial to parallel converter, *s2p*, that converts samples to the *complex_part_ty* custom type and builds an array. The output is taken from the external port of module *autocorrelation_ram*, and is addressed with signal *ext_addrin_i*. This choice allows the downstream processing to exploit already instantiated RAM resources and takes into consideration the requests from the custormer.

The *complex_ty_conv_mux* block is a particular multiplexer. Depending on input signal *sel_in_mux_i* different operations are performed.

- When *sel_in_mux_i* is '00', data is selected from input *A*. Each *complex_part_ty* element of the array is packed with a zero-valued imaginary part into a *complex_ty* element.
- When *sel_in_mux_i* is '01', data is selected from input *B*. The behaviour of the module is similar to the former case.
- In all other cases, the output is composed of zero-valued *complex_ty* elements. This allows to perform the zero-padding in an easy way.

Module *fft_processor* is the processor designed in chapter 9.1. This is instantiated in the design by using its interface, illustrated in Lst. 9.5. In order to make the design as portable as possible, all inputs and outputs are defined by using standard types *std_ulogic_vector* and *std_ulogic*. The length of arrays is managed by loading package *fft_global_settings*. The content of this package does not conflict with *autocorrelation_settings* because common constants have been defined with different names. Moreover, in Lst. 9.5 the *fft_types* package is not loaded, thus avoiding any possible type mismatch. A negative aspect of this solution is the necessity of defining conversion functions from and to the *complex_ty* and *complex_vector_ty* custom types and the standard types. These are in the package body of *autocorrelation_types*. Another approach that could have been used to interface the designed processor to the rest of the PSD computer is based on generics. This technique introduces the possibility of a mismatch between processor settings and instance parameters, because of the lack of a *package generics* mechanism like the one in VHDL200X. Moreover, this usage of generics differs from the followed design philosophy. In the whole design generics are employed only when the same block is used in different parts of the architecture or instantiated many times.

Block *square_modulus* computes the square modulus of the input *complex_part_ty* by using two real multipliers. Its output register is enabled with the *en_modulus_i* signal.

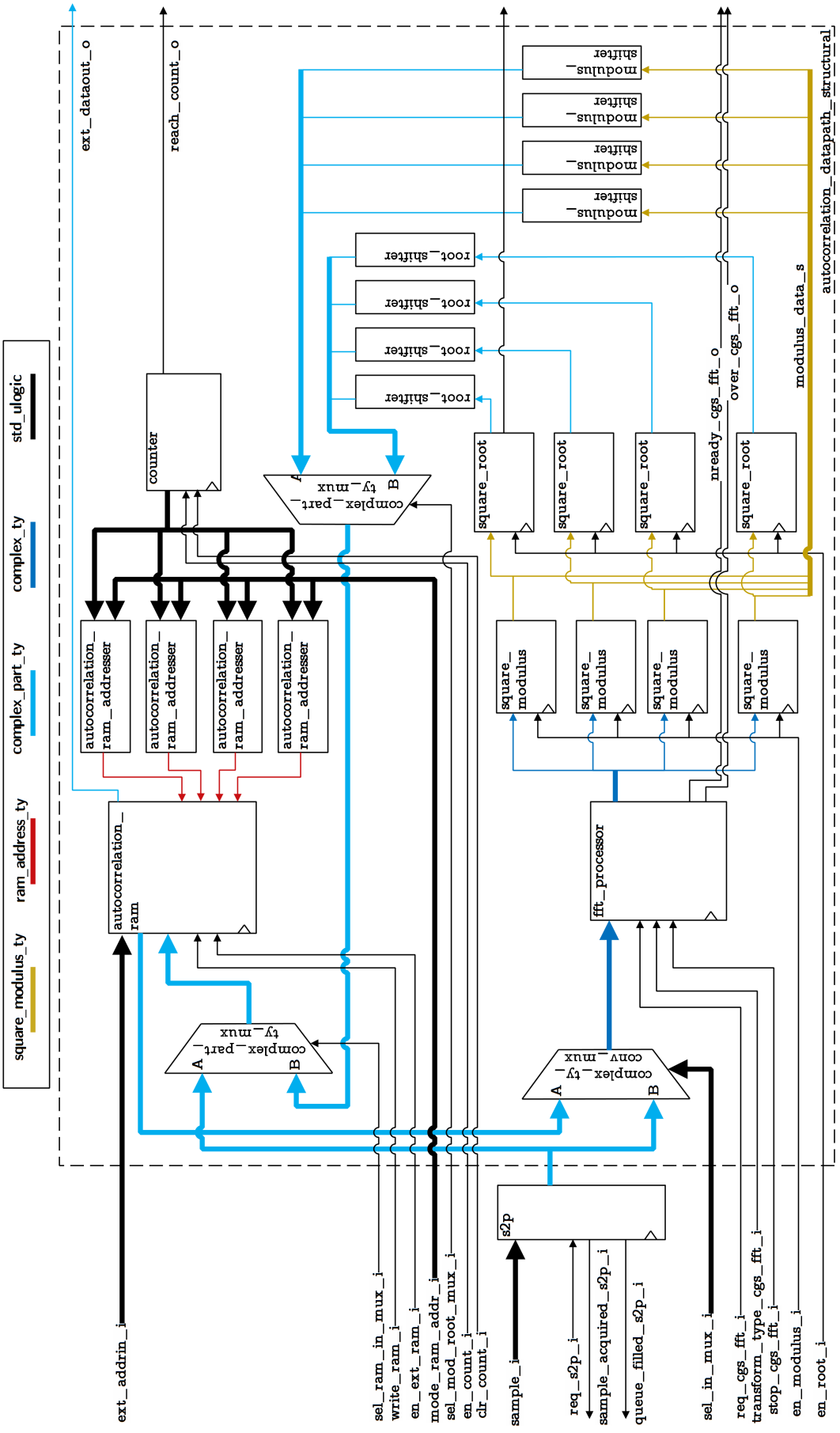

Figure 9.20: Structural diagram of the datapath of the PSD computer. $2P = 4$. Clock and asynchronous reset are omitted for simplicity.

Listing 9.5: Interface of entity *fft_processor*.

```
library ieee;
use ieee.std_logic_1164.all;
use work.fft_global_settings.all;

entity fft_processor is
   port( clk_i                : in std_ulogic;
         rst_i                : in std_ulogic;
         req_i                : in std_ulogic;
         transform_type_i     : in std_ulogic;
         stop_i               : in std_ulogic;
         datain_i             : in std_ulogic_vector(
               0 to (2*global_pes_c)*(global_accuracy_tot_c*2)-1);
         dataout_o            : out std_ulogic_vector(
               0 to (2*global_pes_c)*(global_accuracy_tot_c*2)-1);
         nready_o             : out std_ulogic;
         over_o               : out std_ulogic);
end entity fft_processor;
```

The square root of each element of signal *modulus_data_s* is computed inside the *square_root* modules. The implemented algorithm is called non-restoring square root algorithm [53]. In order to describe the algorithm, an integer $2B$-bit value D is considered [22]. If Q is its integer square root and R the remainder, the following equation holds:

$$D = Q^2 + R, \tag{9.16}$$

where Q is expressed with B bits and R with $B+1$ bits. Defininning $r_0 = D2^{-2B}$, at every step the remainder can be computed by inverting Eq. (9.16) and obtainining

$$r_{i+1} = r_0 2^{2(i+1)} - q_{i+1}^2. \tag{9.17}$$

Noticeably, at step $B-1$ Eq. (9.16) and Eq. (9.17) correspond. In the here illustrated algorithm, the quotient q is updated as follows

$$q_{i+1} = 2q_i + Q_{i+1} \qquad Q_{i+1} = \begin{cases} 1, & \text{if } r_{i+1} \geq 0 \\ 0, & \text{otherwise} \end{cases}. \tag{9.18}$$

This means that, at every step, data bits are shifted left by one position and the last bit of the obtained word is set depending on the remainder. Substituting Eq. (9.18) into Eq. (9.16) it results

$$\begin{aligned} r_{i+1} &= r_0 2^{2(i+1)} - (2q_i + Q_{i+1})^2 \\ &= 4r_0 2^{2i} - (4q_i^2 + Q_{i+1}^2 + 4Q_{i+1}q_i) \\ &= 4r_i - (4q_i + Q_{i+1})Q_{i+1}. \end{aligned} \tag{9.19}$$

When $Q_{i+1} = 1$, then $r_{i+1} = 4r_i - (4q_i + 1)$. But if the result is negative, it means that the iteration made q_{i+1} too big. Thus a compensation must be performed by adding $(4q_i + 1)$ back, obtaining $r_{i+1} = 4r_i$, and setting $Q_{i+1} = 0$. Considering $r_i < 0$, this should be restored by setting $r_i = 4r_{i-1}$ and $q_i = 2q_{i-1} + 0$. Instead, in the following iteration the remainder is

computed as

$$\begin{aligned} r_{i+1} &= 4r_i - (4q_i + 1) \\ &= 4(4r_{i-1}) - (4q_i + 1) \\ &= 4\big[r_i + (4q_{i-1} + 1)\big] - (4q_{i-1} + 1) \\ &= 4\big[r_i + (2q_i + 1)\big] - (4q_{i-1} + 1) \\ &= 4r_i + (4q_i + 3). \end{aligned} \tag{9.20}$$

The result is that in the non-restoring square root algorithm there is no interruption of computation, but this continues with $r_{i+1} = 4r_i + (4q_i + 3)$. The complete procedure is illustrated in Alg. 4. Among the advantages of this algorithm, if compared with other approaches, is that its hardware implementation does not need additional registers and control logic to manage the case of a negative remainder. Moreover, it is suitable for a pipelined architecture.

Algorithm 4 The non-restoring square root algorithm. From [22].

Require: D, B
  $r_0 \leftarrow D \times 2^{-2B}$
  $q_0 \leftarrow 0$
  for $i = 0$ **to** $B - 1$ **do**
    if $r_i \geq 0$ **then**
      $r_{i+1} \leftarrow 4r_i - (4q_i + 1)$
    else
      $r_{i+1} \leftarrow 4r_i + (4q_i + 3)$
    end if
    if $r_{i+1} \geq 0$ **then**
      $q_{i+1} \leftarrow 2q_i + 1$
    else
      $q_{i+1} \leftarrow 2q_i$
    end if
  end for
Ensure: q_B

The realised implementation of Alg. 4 takes advantage of its structure and is composed of B pipeline stages, where B is the number of bits in a *complex _part_ty* data. The number of stages can not be reduced due to operating frequency constraints. In fact, although multiplications in lines 5 and 7 of Alg. 4 can be performed by a simple arithmetic left shift, the operation requires a cascade of two B-bit adders. Furthermore its result is used in order to compute q_{i+1}. From these consideration we expect that including two iteration steps in a single pipeline stage would decrease the clock frequency below the requirements.

In order to perform the computation, it is necessary that the input word to block *square_root* is composed of an even number of bits. Being this value an editable parameter of the system, the chosen approach is shown in Lst. 9.6. Fixed-point *square_modulus_ty* data is extended in such a way that both its integer and fractional part are composed of an even number of bits. This allows to obtain the correct value from Alg. 4 by simply shifting the obtained result left or right. Moreover, function *root_size_low* allows to add more bits at the end of the word to obtain a longer result that can be rounded, thus achieving an improved accuracy of results.

Listing 9.6: VHDL functions and constants in architecture *square_root_structural*.

```
-- Function "root_size_low" computes the lower bound for the
-- radicand of the squared root so that the number of bits of
-- the fractional part is even.
-- Two bits are added to have a better resolution.
function root_size_low(square_low_bound : integer) return integer is
begin
   if ((-square_low_bound mod 2)=0) then
      return square_low_bound - 2*square_root_guard_bits_c;
   else
      return square_low_bound-1 - 2*square_root_guard_bits_c;
   end if;
end function root_size_low;
-- Function "root_size_high" computes the high bound for the
-- radicand of the squared root so that the number of bits of
-- the integer part is even.
function root_size_high(square_high_bound : integer)
   return integer is
begin
   if ((square_high_bound mod 2)=0) then
      return square_high_bound+1;
   else
      return square_high_bound;
   end if;
end function root_size_high;

-- Bounds for the data. They are such that for the radicand the
-- number of bits of both the integer and the fractional part
-- are even.
constant radic_bound_high_c   : integer
   := root_size_high(square_modulus_ty'high);
constant radic_bound_low_c    : integer
   := root_size_low(square_modulus_ty'low);
constant radicand_int_c       : integer := radic_bound_high_c+1;
constant radicand_frac_c      : integer
   := radic_bound_high_c+1-radic_bound_low_c;
constant remainder_int_c      : integer
   := radic_bound_high_c+1-radic_bound_low_c;
constant remainder_frac_c     : integer
   := radic_bound_high_c+1-radic_bound_low_c;
constant quotient_int_c       : integer
    := (radic_bound_high_c+1-radic_bound_low_c)/2;
constant quotient_frac_c      : integer := (-radic_bound_low_c)/2;
```

The rescaling operation is performed either by modules *root_shifter* or *modulus_shifter* as an arithmetic left shift of *scaling_factor_c* positions. Consequently, the value of s in Alg. 3 results set to

$$s = 2^{\text{scaling_factor_c}}.$$

Also, it is possible to choose the rouding type by editing constant *scaling_rounding_c*. This constant is defined in package *autocorrelation_settings* and allows an improved accuracy at the expense of resource saving. By default, this parameter is set to *fixed_round*. This allows the right accuracy for the control system to achieve its function.

Another fundamental is the *autocorrelation_ram_addresser*. This module implements different addressing schemes depending on input signal *mode_ram_addr_i*.

- When it is '000', addressing is linear. All the addresses that are propagated in parallel in signal *systemside_addr_i* have consecutive values.

- When *mode_ram_addr_i* is '001' the address is initially computed as case '001'. Then, all the bits but the first are reversed. This allows to perform the bit-reversed addressing of 1024 values and is necessary to implement line 6 of Alg. 3.
- Case '010' is similar to the former, but all address bits are involved in the reversal. With this technique the bit-reversed ordering of 2048 values can be achieved as required in line 10 of Alg. 3.
- Addressing scheme is bit-reversed and mirrored when *mode_ram_addr_i* is '011'. This means that data referring to location 1023 in the simple bit-reversed case is addressed to location 0. This mode is not used by default, but it can be by modifying the control. Its application is related to the ordering of final data.
- When *mode_ram_addr_i* is '100' addressing is linear but inverted. The first address is thus 1024, and adjacent values differ of -1. This scheme allows to read the RAM from the last location to the first, and is necessary to exploit spectral simmetry.

The design of block *autocorrelation_ram* is essentially based on timing considerations. The achieved goal is that, at every clock cycle, all read and write operations are performed simultaneously. Given the sequence of data accesses in Alg. 3, the RAM must be capable of serving $2P$ requests at the same time. This guarantees the correct timing for the whole PSD computer. The only constraint on synchronization is that when a read operation is requested, the result must be available on the next clock cycle. Fig. 9.21 shows the internal structure of module *autocorrelation_ram*. Block *complex_part_ty_register* implements data location 1025, while all the others are contained in the *ram_core* module.

The *ram_router* block selects data depending on its address and manages the routing of output data. It is composed of a decoding logic, a register bank and a set of multiplexers. The decoding logic determines the data to write into the register bank. At every clock cycle, if *en_ext_i* is set to zero, the module considers requests coming from the internal side of the PSD computer. If the i-th address, with $0 \leq i \leq 2P-1$, refers to location 1025, then a binary word referring to the *complex_part_ty_register* is written in register i. Else, the word refers to port i of the *autocorrelation_ram* block, either to write signal *sys2core_data_s* or to read *core2sys_data_s*. The first bit of the address is cut away, and the remaining bits define a *internal_ram_address_ty* type. These feed the RAM module. The register bank drives the multiplexers, so that in the following clock cycle address data is correctly available at output port i. If *en_ext_i* is high, the system-side ports of the RAM are disabled, and only read requests from the external port are served. All non-valid read requests are managed by simply referring to the RAM ports, but the impossibility of this eventuality comes from the discussion on block *autocorrelation_ram_addresser*.

Non-valid write requests are managed by block and *write_checker*, as seen in Fig. 9.21. Its function is simply to set the write bit to zero whenever the called location has an address above 1025.

The implementation of module *ram_core* can be different, depending on the chosen technology. If the design is implemented on an ASIC platform, this block is no more than a simple $2P$-port RAM. All the addressing schemes explored when analysing block *autocorrelation_ram_addresser* and Alg. 3 itself guarantee that no overlap between read and write steps is possible. Moreover, it is not possible that the same address is requested simultaneously. The only problem holds when the *write_checker* module modifies some requests

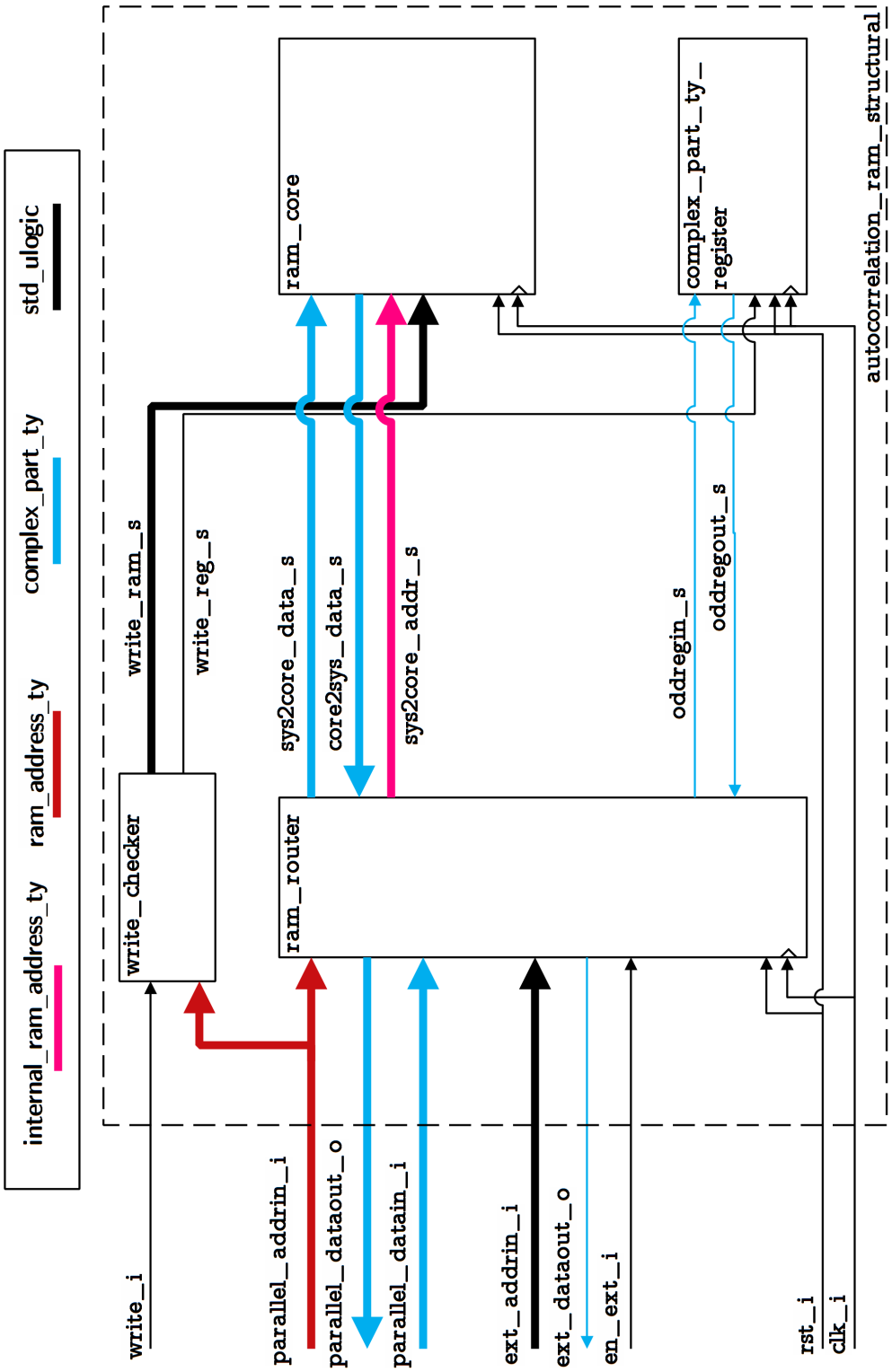

Figure 9.21: Structural diagram of architecture *autocorrelation_ram_structural*.

from write to read. But in this case, the system is expecting to perform a write operation, thus the effect of the forementioned module is simply to let the request fall unserved. By consequence, the memory coherence management is extremely simple. Nevertheless, on an ASIC platform, two-port RAM blocks place a fixed limit on the implementation. In order to understand the design of this module, we examine Fig. 9.22. From Alg. 3 we know that the bit-reversed addressing is necessary whenever a write operation must be performed, while the linear addressing is essentially used for reading. Holding the fact that both operations must be simultaneous on $2P$ ports, it is possible to give an interpretation of address *internal_ram_address_ty* based on fields. Supposing to have default settings, it is $2P = 8$ and thus we consider

$$\underbrace{a_9 a_8 a_7}_{w}\underbrace{a_6 a_5 a_4 a_3}_{l}\underbrace{a_2 a_1 a_0}_{r}, \tag{9.21}$$

where w is the *write index*, r is the *read index* and l is the BRAM location addressed. Consequently, in the FPGA case the problem is solved by implementing a $(2P)^2$ matrix of BRAM blocks in which every element is addressed with (w, r). This approach may lead to hardware overhead, and also decrease the operating clock frequency, depending on how the scheme is realised. On the other hand, considering the final ASIC orientation of the overall design, the given solution may be considered as a compromise for a rapid FPGA prototipation.

0000000000	0000000000
0000000001	1000000000
0000000010	0100000000
0000000011	1100000000
0000000100	0010000000
0000000101	1010000000
0000000110	0110000000
0000000111	1110000000

Figure 9.22: Examples of addressing and BRAM selection fields.

The hardware structure implementing the address scheme described by Eq. (9.21), follows the same architecture of *autocorrelation_ram*. A *bram_router* module, whose behaviour is similar to the one of *ram_router*, selects the correct BRAM block depending on fields w and r of the address. Thus, location l of block (w, r) is requested. The same mechanism involving a register bank and a set of multiplexers, as described before, allows the correct timing for read operations.

Control

The control of the PSD computer is performed by a Moore machine. Because of its complexity both in terms of number of states and in terms of signals involved, its behaviour is illustrated in Fig. 9.23 with a functional diagram, instead of a full dataflow.

The first stage of the control inolves initialisation of the FFT processor and communication with the serial to parallel converter. Whenever this module completes an entire column of $2P$ samples, these are sent through the *complex_ty_conv_mux* in Fig. 9.20 to the processor.

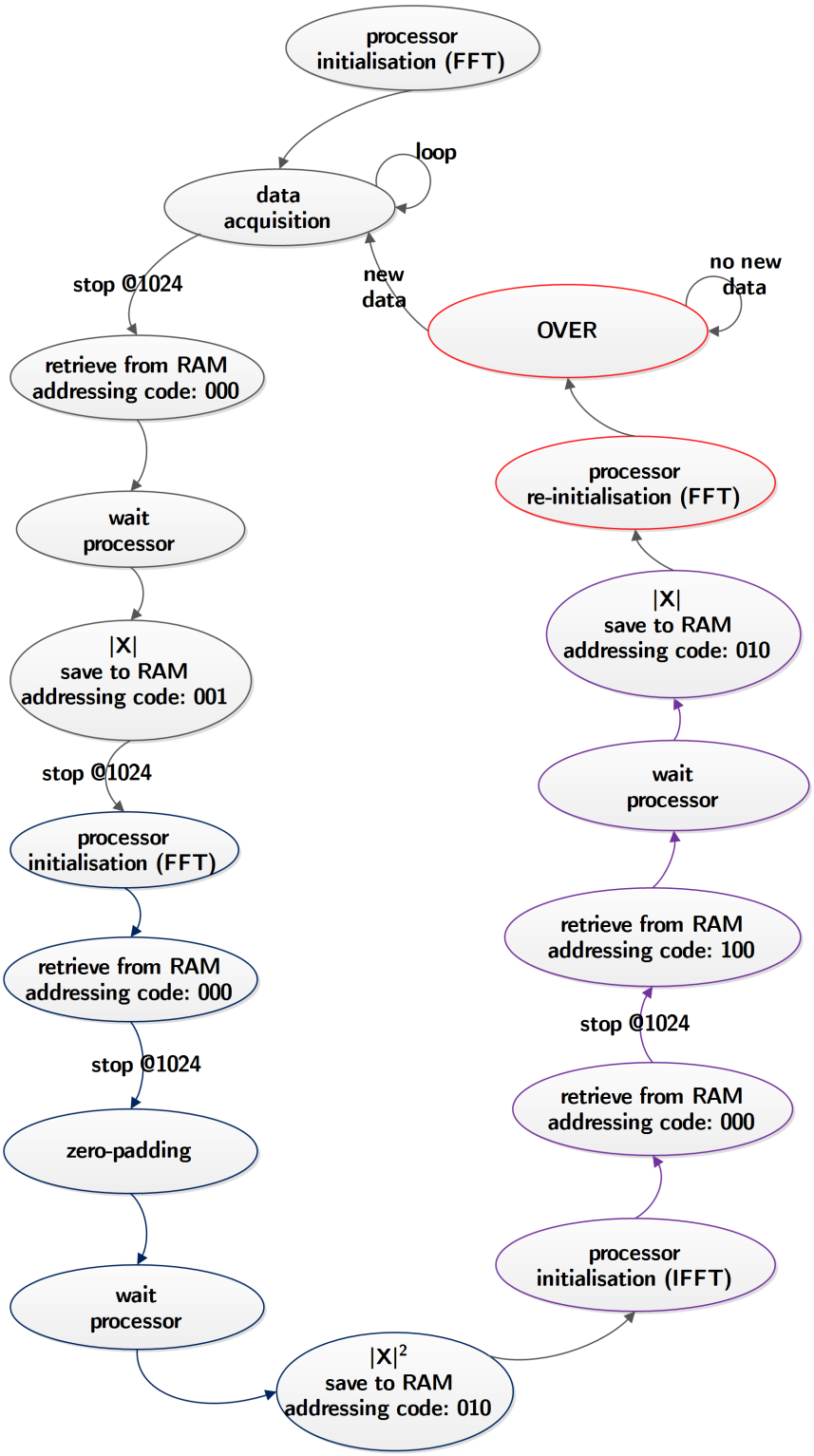

Figure 9.23: Functional diagram of the control of the PSD computer.

Also, these samples are sent to the RAM to be stored in linear-addressed order. Note that this is the only case in which writing is performed with linear addressing. When 1024 samples are read, the counter in Fig. 9.20 propagates a reach signal. More samples from the serial to parallel converter are ignored and samples stored in the RAM are read in linear order and then sent to the processor.

When the processor signals the end of computation by driving signal *over_cgs_fft_o* to one, modules *square_modulus* and *square_root* in Fig. 9.20 are initialised. After that, while square moduli and square roots are computed and then scaled, data is written into the RAM with bit-reversed addressing scheme specified by *mode_ram_addr_i* set to '001'. When 1024 values are read, the interruption of FFT data return is achieved by driving signal *stop_cgs_fft_o* to one.

The second stage of processing, as described by Alg. 3, starts with the initialisation of the FFT processor in order to perform the direct transform. Subsequently, the first 1024 data are read by direct linear addressing from the RAM. The remaining 1024 values are zeros, and are input to the processor by exploiting the features of block *complex_ty_conv_mux*. With this expedient, the zero-padding is performed as in line 7 of Alg. 3.

When the processor is ready to output results, the *square_modulus* block in Fig. 9.20 is initialised. The square moduli of all the 2048 complex data are computed and then scaled, before being written to the RAM. The requested addressing scheme in this case is the conventional bit-reversed one, specified by setting *mode_ram_addr_i* to '010'.

The last processing step is the computation of the IFFT. As in the previous algorithmic stages, the FFT processor is initialised first, and the inverse transform is requested by setting signal *transform_type_cgs_fft_i* to one. Then, data feeding the processor are read first by linear addressing and then by inverse linear addressing from the RAM. When the processor is ready to output results, both the *square_modulus* and *square_root* modules are initialised for the last time. Values are then scaled and written to the RAM in bit-reversed order.

When signal *over_o* is driven to one by the control, valid data is available in the RAM of the PSD computer. The external port of block *autocorrelation_ram* is enabled on the following clock cycle and read operations can be performed. This is possible until new data is available from the serial to parallel converter and signal *queue_filled_i* is set to one again. The communication protocol of the PSD computer is thus interely managed by the *over_o* and *nready_o* signals.

over_o This signal refers to the output of the processing system. It his high whenever valid data is contained in the ROM.

nready_o This signal refers to the input port of the system. It is high whenever the system is processing data. When *nready_o* is set to one, the external port of the RAM is disabled and every input from the user is ignored.

With this remarks, it is possible to use *over_o* as a trigger to start reading computation results, and *nready_o* as a confirmation signal to validate user inputs.

The implemented PSD computer takes advantage of the parallelism, which is a feature of the scalable FFT processor described in the former chapters. By a simple editing of global settings, we expect to achieve design flexibility and resource saving as requested by the customer. Moreover, the communication protocol is kept easy in order to simplify the interface with downstream modules. Nonetheless, a major complexity of the overall design is in the Moore machine that implements the control. We expect this to allow an easier editing and

improving of the single blocks, while keeping all the possible synchronization issues confined to the global control.

9.4.3 Frequency feature extraction

When the sample array is processed by module *autocorrelation*, its result is located inside the module's RAM. Another block, *extraction_data*, performs a linear search on it in order to extract some features which are necessary for SMPS control purposes.

The behaviour of *extraction_data* can be summarised few steps:

1. it initialises some internal registers containing data and addresses to be requested.
2. it performs a backwards one-by-one scan from the last element of the processed array to element specified by generic *end_search_g*. If a local maximum is found, the module stops earlier.
3. if the found maximum value is above *f0_threshold_g*, then another search is performed starting from *start_esr_search_g* to *end_search_g*.
4. If a local minimum is found, then another search starts from this element towards *end_search_g* in order to find another local maximum.
5. When data feature extraction is complete, signal *extraction_data_ready* is set to 1 for one clock cycle.

9.5 Testing and simulation environment

As seen in the past chapters, the design of a FFT scalable processor is complex both in terms of modules and synchronisation of components. The development of the autocorrelation-based PSD computer described in chapter 9.4 is even more critical. The main reason is that the design must be tested for a lot of different configurations to check its correct behaviour. For example, in the case of scalable hybrid TF generators, it is necessary to vary the number P of processing elements, the number of word bits B' and the number of FFT points N, in order to ensure a good coding of the architecture. The possible combinations are various, and a bug can be evident in only one of them. Also, the forementioned systems are supposed to process many points, in different stages. It is possible that a peculiar type of error occurs only when a certain set of data is given to the module. From these issues arises the necessity of a structured Testing and Simulation Environment (TSE).

In this chapter, the organsation of the employed TSE is covered. First of all, the structure of VHDL testbenches is analysed, then MATLAB simulation scripts are explained. The developed *fixed-point binary* library allows to simulate the VHDL *fixed_pkg* package and to create a full software model of the systems.

9.5.1 VHDL testbenches

The VHDL part of the TSE is organised as shown in Fig. 9.24. The illustration refers to the FFT processor, but the TSE for the PSD computer follows the same approach. All custom global packages are contained in the *work* library of the design and are related one to the

other as discussed in Sec. 9.1.2. Standard VHDL packages are contained in a separate library, *ieee_proposed*. Every module instantiates one or more of these packages. In VHDL terms, every module is called *entity* and has its own declaration construct in the language. An entity is like a black box, with one or more *architectures* associated to it. In the developed TSE, a testbench is coupled with the entity too.

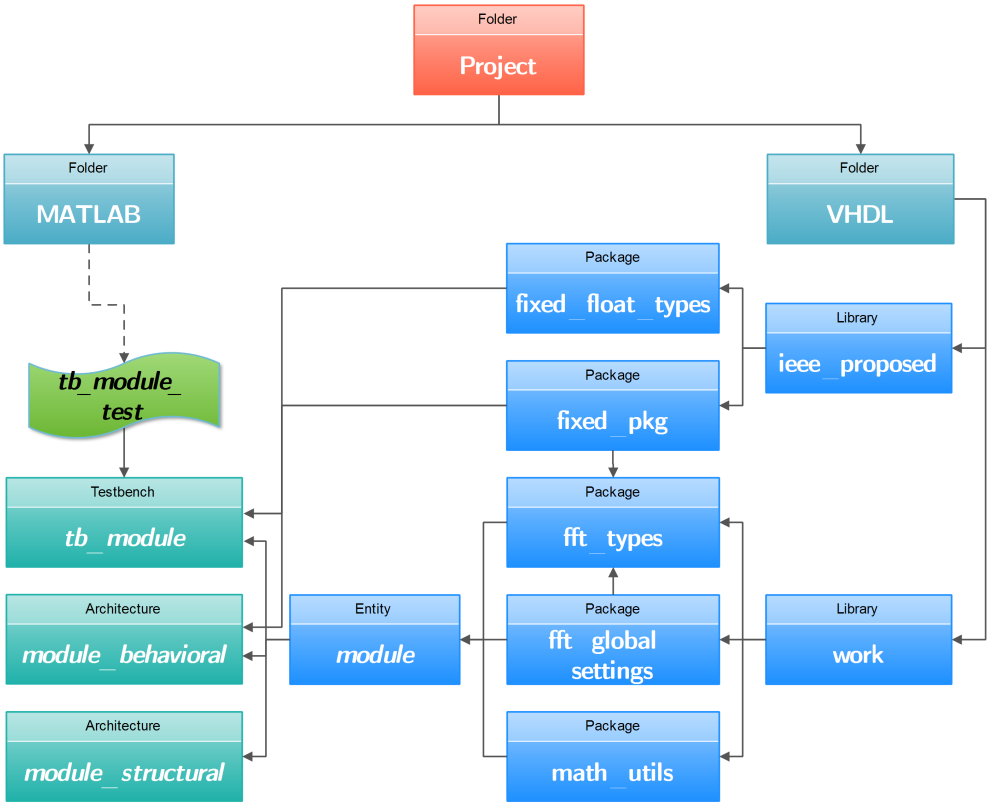

Figure 9.24: Organisation of the TSE on the VHDL side.

All testbenches follow the same structure.

1. All signals in the testbench are declared by using either standard types or custom types defined in the forementioned packages.

2. Clock management with the following statement:

   ```
   clk_s <= not clk_s after clock_period_c/2;
   ```

   This allows to set the clock frequency, by modifying constant *clock_period_c*.

3. Instantiation of the Unit Under Test (UUT).

4. A *test_p* process reads file *tb_module_test*, containing both inputs and test vector. An example test file is shown in Lst. 9.7. The file format can change for every testbench, but its file header always specifies it clearly. Inside the process, inputs are always given to the UUT before checking results. If a mismatch is found, this is notified to the designer by exploiting the presented VHDL standard function. When test is complete, message '*Test completed*' is printed to screen.

Listing 9.7: Testfile for entity *square_root*.

```
--------------------------------------------------------------------
-- tb_square_root_test.txt
-- This file contains stimuli for testbench "tb_square_root.vhd".
-- Generated using "tb_square_root_gen.m".
-- Guard bits : "3".
--
-- Template:
-- <time> <rst> <en>
-- <datain> <dataout>
--------------------------------------------------------------------
-- Resetting
 0 1 0
 00000000000000000.0000000000000000000   00000000.0000000000
-- Requesting radix of 0.00000
 110 0 1
 00000000000000000.0000000000000000000   00000000.0000000000
-- Requesting radix of 131.20320
 120 0 1
 00000000010000011.00110100000001010010   00000000.0000000000
-- Requesting radix of 262.40641
 130 0 1
 00000000100000110.01101000000010100100   00000000.0000000000
-- Requesting radix of 393.60961
 140 0 1
 00000000110001001.10011100000011110110   00000000.0000000000
-- Requesting radix of 524.81281
 150 0 1
 00000001000001100.11010000000101001000   00000000.0000000000
```

The software used to run testbenches is the ISE Simulator (ISim) from Xilinx. This tool is freely available with Xilinx ISE Project Manager, which is the software used for the whole coding and project organisation.

9.5.2 MATLAB simulation scripts

Given the previously discussed issues about testing different combinations of parameters and inputs, some software tools were developed in order to validate simulation results.

First of all, the *fixed-point binary* MATLAB library was coded to allow simulation of all fixed-point mathematical operations. The standard MATLAB data used in this library to represent a fixed-point binary number is the *string*. No object-orientation was adopted for the design of the library. A string is accepted as valid by *fixed-point binary* if it contains only the characters '0', '1' and a dot, although the dot may not be present if the value is an integer. The *fixed-point binary* library also takes advantage of MATLAB built-in *fixed-point toolbox* for performing mathematical operations. Nonetheless, many functions have been written *from scratch* to make hardware debugging easier and more punctual.

Among the most important functions of the library we have *bin2dec_mod*. This was developed to substitute the built-in MATLAB *bin2dec* function, which can convert binary dot-free strings to decimals only if the string has less than 52 characters. This is an issue especially when simulating the hybrid TF generator with a high number of guard bits.
A central role is covered by the *resize_binary* function, which emulates the behaviour of VHDL *resize* function. This allows to change the number of bits of the integer or fractional part of a fixed-point value, and takes care of resulting overflows and underflows. In particular, when rounding from a word with a longer fractional part to a shorter one, the remainder

is checked first. If the most significant bit of the remainder is a '1' and the less significant bit of the unrounded result is a '1' or the lower bits of the remainder include a '1', then the result will be rounded. This means that '1' is added to the lowest bit of the unrounded result. If saturation is enabled, whenever a value exceedes the largest representable number, the maximum possible value is returned, given its integer and fractional part bits.

The structure of the TSE on the MATLAB side is shown in Fig. 9.25. The *fixed-point binary* library is employed either by a library of components and by scripts. Components are implemented as functions and are called inside simulation scripts. When the test of a module is passed, then a function is made out of the script and it becomes a component. This allows a *bottom-up* approach to design testing.

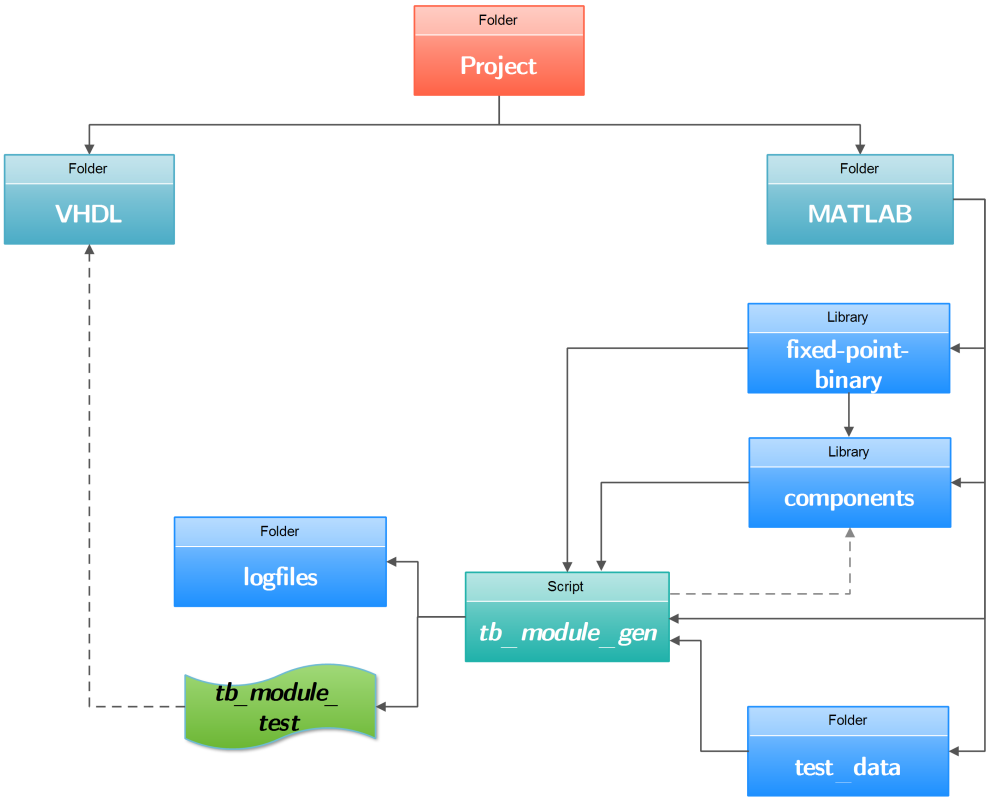

Figure 9.25: Organisation of the TSE on the MATLAB side.

The structure of a MATLAB test script is the following:

1. Script initialisation. Previous data is removed from the MATLAB workspace and simulation constants are declared.

2. The test file header is written, specifying all simulation parameters. If the simulated system is particularly complicated, a logfile is also opened and its header written.

3. Data is generated either in a random way or by importing a file from the *test_data* folder. This allows to repeat the simulation on the same data but changing the parameters.

4. Following the test scheme of the script, the actual simulation is performed. At every step, the test file is compiled according to its format. Also, the possible logfile keeps trace of variables, whose names are identical to the ones used in the RTL design.

5. When simulating complex systems, such as the DSP, the FFT processor or a TF generator, a plot of simulation results is shown. This depicts data during every computational stage and final statistics on error.

6. Logfiles and test files are closed. A message saying '*Job done!*' is printed to screen.

Being the TSE organised in two separate VHDL and MATLAB parts that must match, an issue occurs whenever a test fails. The problem can be not in the RTL coding but on the MATLAB side, in the custom *fixed-point binary* library, in the simulation scripts or in a component. The followed approach in this case is cross-validation of both parts: with the developement of MATLAB code, a deeper understanding of hardware bugs is achieved and solutions are prototyped more easily. Moreover, the advantage of the used TSE is in the possibility of running many simulations sequentially and automatically collecting results.

9.6 Simulation and synthesis results

In this chapter, all results related to the implemented architectures are covered. We can distinguish between accuracy, timing and synthesis results. These are illustrated for the hybrid twiddle factors generators, for the FFT processor and for the overall PSD computer. Considering scalable architectures, the results are presented by highliting the variation of performances and resource occupation with different parametric configurations. The effect of editing parameters is analysed in depth for all the metrics.

Because of the orientation of the whole project towards a future ASIC implementation, other metrics than FPGA-based values may be necessary. When possible, an expression describing the expected value of the metric is derived and then validated through experimental results.

9.6.1 Hybrid twiddle factors generators

Accuracy

The hybrid CORDIC-LUT twiddle factors generators described in chapter 9.3 was simulated with the testing and simulation environment discussed in chapter 9.5.

In section 9.3.2 it was noted that the scalable rotational algorithm computes TFs by using results of previous computations, and this can be an issue in terms of error propagation. The adopted solution is to use $B' \geq B$ bits, where B is the bit length of outputs, for the internal representation of twiddle factors. We expect the problem to be the most relevant in the first stage, $s = 0$, following the notation introduced in chapter 9.3. In this stage all TF indexes are adjacent, thus error propagation is more visible. Given sequence $\{W^{(k)}_{N,\text{obtained}}\}$ of twiddle factors obtained at stage zero, and sequence $\{W^{(k)}_{N,\text{exact}}\}$, of twiddle factors computed in MATLAB with full resolution, the relative error is defined as follows:

$$E = \frac{|W^{(k)}_{N,\text{obtained}} - W^{(k)}_{N,\text{exact}}|}{|W^{(k)}_{N,\text{exact}}|} = |W^{(k)}_{N,\text{obtained}} - W^{(k)}_{N,\text{exact}}|. \tag{9.22}$$

Fig. 9.26 shows mean value and standard deviation of E as a function of $B' - B$. These values are computed considering B fixed to 16 and N set to 1024. The curves related to the classic CORDIC algorithm are obtained with MATLAB *cordicsincos* built-in function. It is evident that in hybrid CORDIC-LUT architectures a certain amount of error must be accepted, if the number of bits of the internal representation of data is not increased from B. Nonetheless, by incrementing this value of few units, comparable results to the conventional approach are met. Values in Fig. 9.26 are obtained when rounding inside the complex multiplier is disabled, thus with default settings.

When rounding is enabled, we expect the hybrid architectures to achieve better accuracy if compared to the former case. A validation of this hypothesis is shown in Fig. 9.27. Conventionally in the classic CORDIC approach the number of steps that are taken in the computation is equal to B, and no rounding is performed at the end of the procedure [115, 33]. The first points of the curves in figures 9.26 and 9.27 refer to this solution. The mean relative error is of about 0.004% and the standard deviation of the relative error is fixed at 0.0023%. Thus, we note that a comparable result between pure CORDIC and the proposed hybrid solution in terms of mean error can be achieved by choosing $B' - B = 2$ in the case of Fig. 9.26, or $B' = B$ in the case of Fig. 9.27, for the pipelined architecture. If the shared-core architecture is chosen, then values are $B' - B = 4$ in the case of Fig. 9.26, or $B' - B = 3$ in the case of Fig. 9.27. Approximately equal results in terms of standard deviation in the rounding disabled case require B' to be at least one bit bigger than the value considered before. In the case of Fig. 9.27, it must be at least $B' - B = 3$ for the scalable pipelined architecture and $B' - B = 4$ for the shared-core structure. In order to minimise hardware resource usage, we require the mean relative error to be near the conventional CORDIC value and accept a bigger standard deviation. Consequently, we consider the values for $B - B'$ listed in Tab. 9.2.

Table 9.2: Considered values of $B - B'$ depending on architecture and rounding.

Architecture	Rounding off	Rounding on
Shared core	3	4
Pipelined	2	0

Notably, in figures 9.26 and 9.27 all the curves tend to stabilise to the same value when the difference $B' - B$ increases. This is because the effect of error propagation between adjacent terms of sequences is easily compensated by the large number of guard bits. However, using such a representation of computational data can lead to both severely hardware-espensive implementations and low operational frequencies. Sure enough, it must be remembered that setting B' also means using larger multipliers and adders. Similarly, enabling rounding means increasing the depth of the critical path, either inside the shared rotational core or inside the pipeline stages. Consequently, accepting a certain amount of error as a drawback may be preferred in some cases.

Speedup effect

While the pipelined architecture can generate P twiddle factors per clock cycle, the shared core solution has a variable troughput until it reaches the maximum value P. The trend is shown in Fig. 9.28. This is because as the stage number increases, the sequence of $\frac{N}{2}$ twiddle

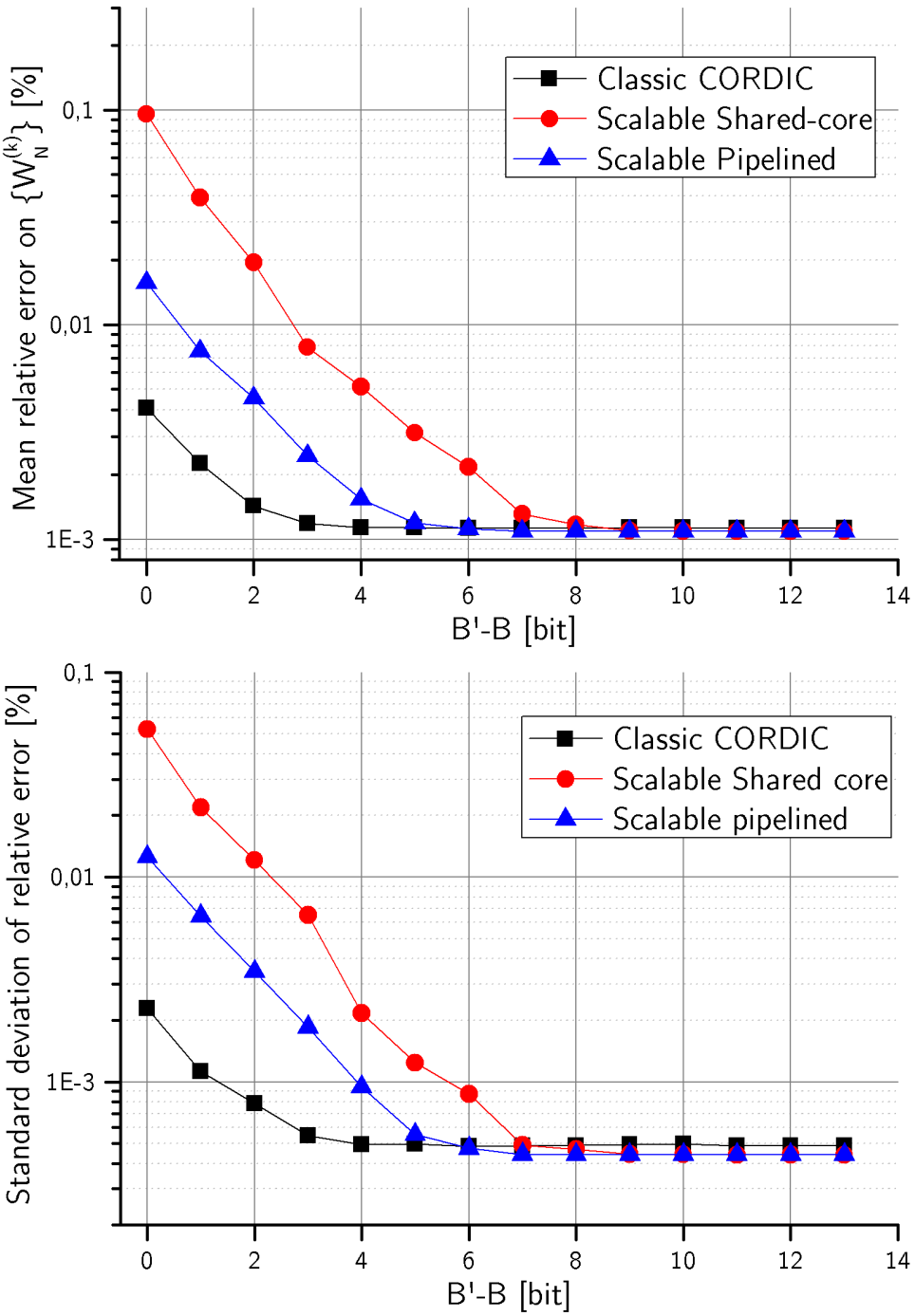

Figure 9.26: Relative error statistics on TF output sequence $\{W_N^{(k)}\}$ with rounding disabled. $B = 16$, $N = 2^{10}$, $s = 0$.

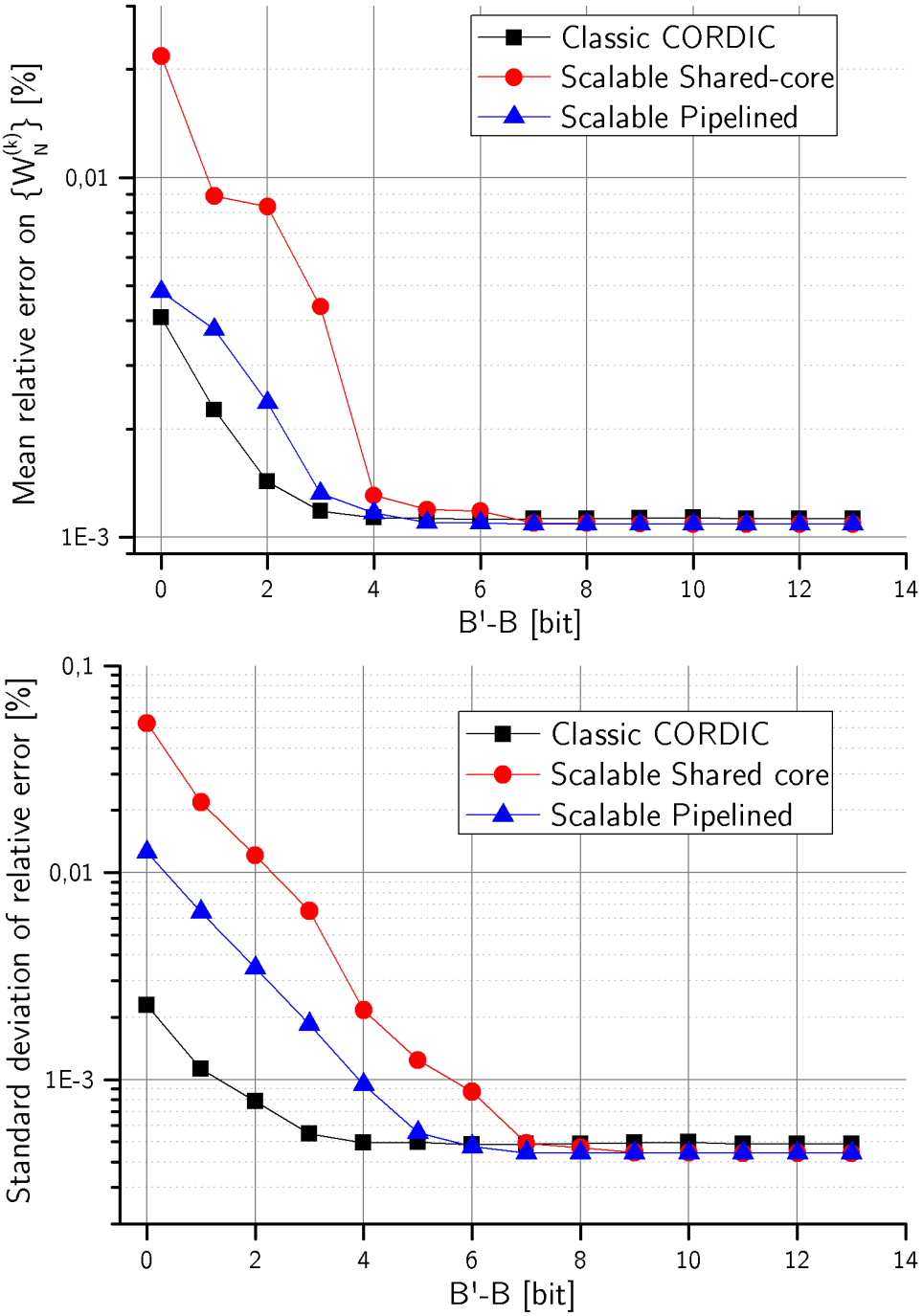

Figure 9.27: Relative error statistics on TF output sequence $\{W_N^{(k)}\}$ with rounding enabled. $B = 16$, $N = 2^{10}$, $s = 0$.

factors contains less different elements. Consequently, less iterations have to be performed and immediate retrieval of data becomes more effective. We call this the *speedup effect* of the shared core architecture.

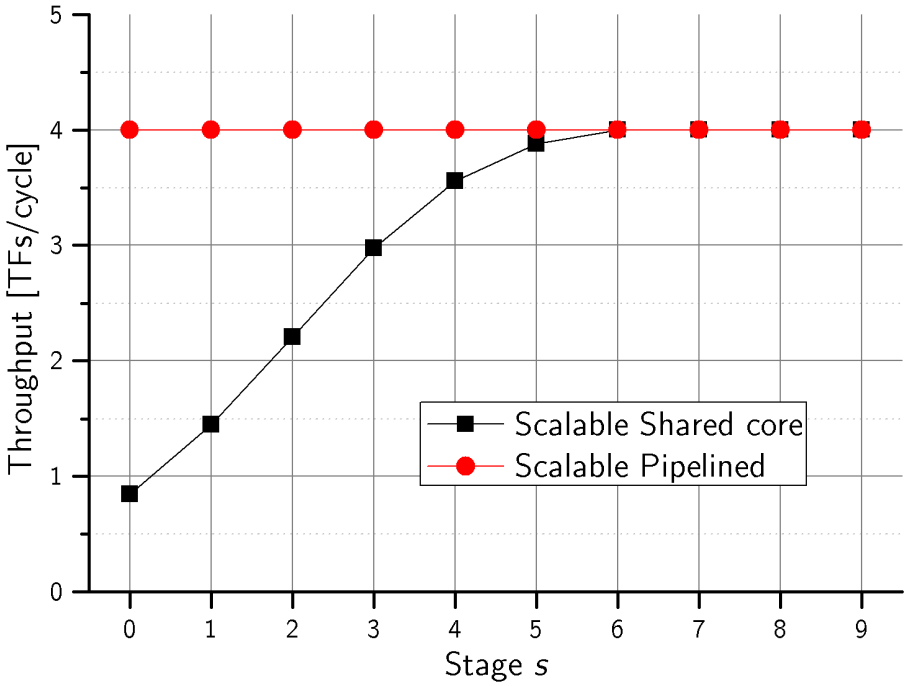

Figure 9.28: Shared core architecture speedup. $N = 1024$, $P = 4$.

This feature of the hybrid CORDIC-LUT system exploits the structure of the proposed scalable rotational algorithm (Alg. 1). In [129] the number of iterations, thus the latency of the system, is reduced by using a lookup-table containing optimised rotational sequences. This leads to variable scale factors that must be correctly compensated in order to perform the FFT. The minimum number of iterations achieved in [129] is $\frac{B}{2}$, where B is the bit length of twiddle factors. In the analysed architecture scaling is not necessary because rotations are performed with complex multiplications. Moreover, all parallel requests involving the same TF index are satisfied at the same time, consequently latency is not a function of B.

Synthesis results

In order to have a technology-independent estimate of resource usage, we want to derive an expression for the number of stored bits in both architectures. We consider only bits of computational data (i.e. twiddle factors) in any representation. For the shared core architecture, as shown in Fig. 9.14, we count the following contributes:

- $2BP$ bits in the output registers of module *unmapper*.
- $2B'P$ bits in the registers that store the output of the *reference module*, as illustrated in Fig. 9.16,
- $2B'P$ bits in the output registers of the CORDIC core.
- $2B'(\log_2 N - 1)$ bits inside the ROM.

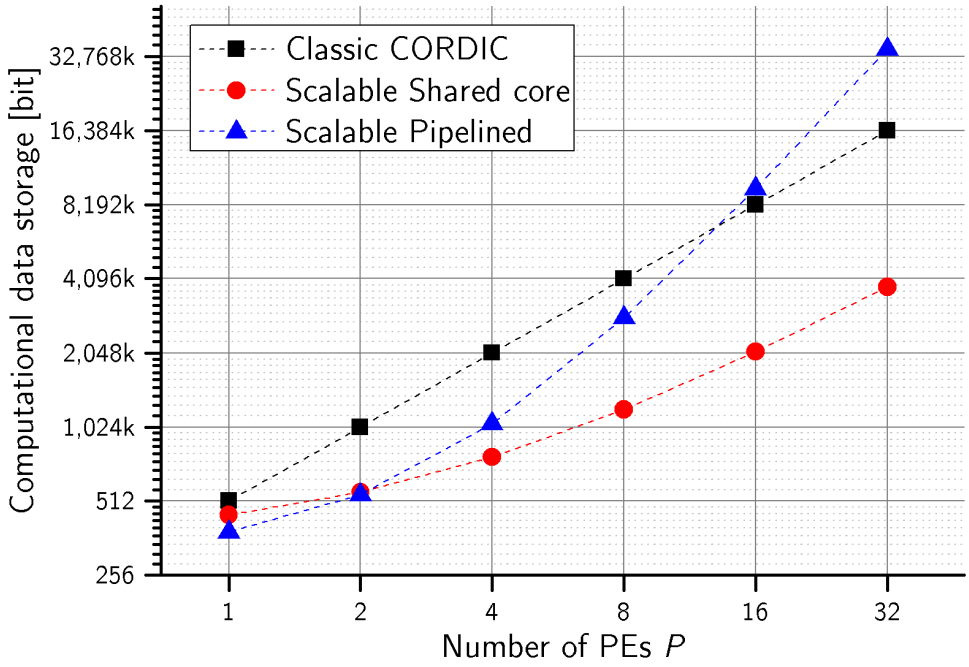

Figure 9.29: Computational data storage estimates as a function of P. $B = 16$ and $N = 1024$.

After summing and ordering the contributes, we obtain the estimate β_{shared} expressed by the equation

$$\beta_{\text{shared}} = 2(2B' + B)P + 2B'(\log_2 N - 1). \tag{9.23}$$

From Fig. 9.18 we obtain the contributes that give the estimate for the pipelined scalable hybrid architecture.

- $2BP$ bits in the output registers of module *unmapper*.
- $2B'(P+1)P$ bits in the output registers of the P+1 pipeline stages composing the structure.
- $2B'(\log_2 N - 1)$ bits inside the ROM.

Thus, the estimate in the pipelined case is

$$\beta_{\text{pipelined}} = 2\left[B'(P+1) + B\right]P + 2B'(\log_2 N - 1). \tag{9.24}$$

In the conventional CORDIC approaches [81, 108, 121] or even in the particularly efficient solution [33], B pipeline stages containing a $2B$-bit register are instantiated for each of the P output elements. Consequently we consider

$$\beta_{\text{classic}} = 2B^2P \tag{9.25}$$

as our estimate of computational data storage. Fig. 9.29 is obtained by evaluating equations (9.23), (9.24) and (9.25), for a 1024 point system with $B = 16$ and $B' = 19$. In the shared core case $B' = 19$ is chosen, while $B' = 16$ is set for the pipelined architecture. This takes into account the remarks related to accuracy in subsection 9.6.1.

From Fig. 9.29 we note that when the number of PEs is below the value of B, then the proposed scalable architectures are more efficient in terms of computational data storage. When $P = B$, then the hybrid pipelined architecture is expected to require about the same storage resources of the conventional CORDIC. However, the estimate for the shared core architecture is always lower if compared to the other architectures. It is also noticeable that when $P = 1$ all architecture use almost the same amount of computational data storage.

Another technology-independent metric that can be given is the number of real multipliers used in the designs. This value can be deduced from the discussion in in chapter 9.3 and is 3 in for the shared core structure and $3P$ for the pipelined architecture.

In order to validate the estimate in equations (9.23) and (9.24), we now illustrate synthesis results for both the proposed scalable hybrid systems and confront them with a classic CORDIC architecture freely available at Opencores.com [41]. These are synthesized on a Xilinx Virtex-5 XC5VLX330T-2FF1738 FPGA. The used synthesis tool is Xilinx XST, with speed as the optimization goal. The settings for all the systems are $N = 1024$ and $B = 16$, or rather the defaults.

The shared core architecture is synthesized with both rounding disabled (thus $B' - B = 4$) and rounding enabled ($B' - B = 3$). Results are shown in figures 9.30 and 9.31. We first note that the trend for the slice registers occupation is almost perfectly exponential, and no bending like in Fig. 9.29 is visible. This is because this value does not take into account the contribute given by the ROM, which was considered in Eq. (9.23). However, register resource saving is deduced from the figure both in the case with rounding enabled and the one with rounding disabled. Slice LUT occupation is instead much similar for all the examined cases. This is due to the resource mapping performed by the synthesizer, that must instantiate bigger multiplexers as P increases. Also, hardware for mathematical operations is much heavier in the shared core architecture, because B' is bigger than B of some bits.

Another important figure of merit that must be taken into consideration is operating frequency, whose plot is in Fig. 9.31. The classic CORDIC approach in [41] works at frequencies of about $400MHz$. On the other hand, the shared core solution has a maximum frequency around $95MHz$ when $P < 8$ and rounding is disabled, while when rounding is enabled, operating frequency is around $75MHz$ in the same range. This is expected from the chosen design philosophy, which tries to achieve hardware resource saving if compared to the conventional approach.

Synthesis results for the pipelined architecture are shown in Fig. 9.32. In this case, synthesis parameters are $B' - B = 2$ for the structure with rounding disabled and $B' = B$ for the one with rounding enabled, according to Tab. 9.2. Fig. 9.6.1 shows the same feature as discussed for the shared core architecture regarding the bending of the estimate curve. The approximation is well followed, and slice register saving is shown until $P = B$, or rather until the number of pipeline stages of the hybrid architecture becomes equal to the one of the conventional CORDIC approach. Interesting results are also illustrated in Fig. 9.6.1: the plot proves that, independently from P, the hybrid pipelined architecture always uses less LUTs on the FPGA if compared to the classic solution.

Moreover, Fig. 9.33 shows the low sensitivity of operating frequency with P. This is always around $114MHz$ in the case of rounding turned off and $91MHz$ in the other one. Both curves prove that the given design requirements can be satisfied with the designed structures, thus guaranteeing resource saving as well as good throughput.

The synthesis figures show that, being the mean relative error approximately equal, the shared core solution is heavier in terms of FPGA LUTs than the pipelined one. The main

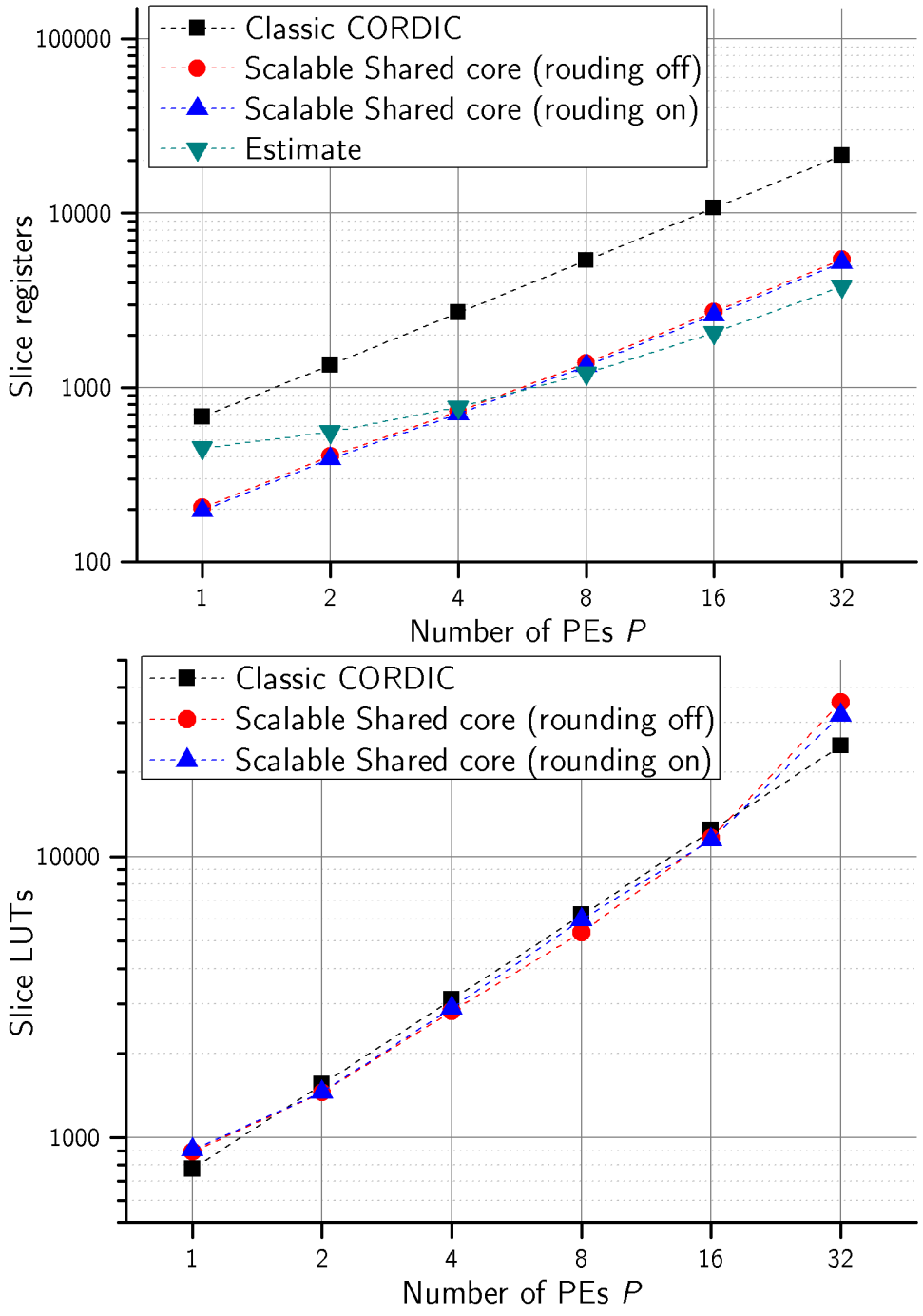

Figure 9.30: Synthesis results for the shared core architecture as a function of P.

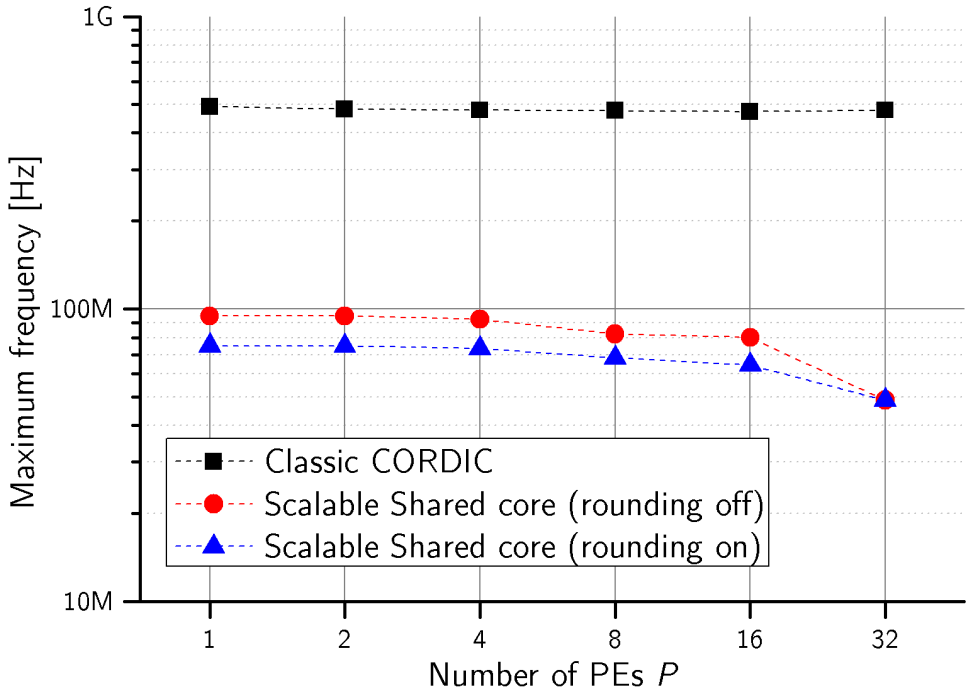

Figure 9.31: Maximum operating frequency of the shared core architecture as a function of P.

reason is that the shared core architecture needs to instantiate much hardware to manage the serving of requests. This includes many multiplexers, which are also wider, in relation to word length, because of error compensation. This does not happen in the pipelined structure, which also results to be less sensitive to variations of P in the maximum operating frequency. Another fact must be taken into account, which is the usage of DSP blocks, or rather real multipliers. As discussed before, this happens to be three times bigger in the pipelined architecture than in the shared core one. This can be an issue when implementing the design on ASIC, and possibly leads to different values of metrics than expected. However, ASIC synthesis must be done to validate estimates in this case. In general, we can state that the shared core architecture saves more hardware resources, compared to the pipelined one, if a bigger error can be accepted on output TFs. In this case B' can be reduced and LUTs saved, as shown in Fig. 9.34. The plots show that, being P fixed, the shared core architecture requires both less slice registers and less slice LUTs if compared to the pipelined solution. However a remark regarding the pipelined architecture with rounding disabled can be done. In Fig. 9.6.1, values referring to this architecture are below the ones of the shared core TF generator with rounding enabled, until $B - B' = 3$. This can be explained by observing that in the figure $B = 16$, and each DSP block of the FPGA contains a 18×18 real multiplier. When $B' > 18$ signals must be routed so that two DSP can be used to perform a real multiplication, and this becomes an issue in the pipelined architecture, which instantiates 24 DSPs instead of 12. The shared core system besides uses only 6 DSP blocks instead of 3, so resource mapping on the FPGA chip is more efficient. From Fig. 9.34 we also note that the pipelined hybrid architecture with rounding enabled requires more slice LUTs than the others when $B' = B$. But Fig.9.6.1 proves that, in order to achieve the same accuracy performances, other solutions require $B' > B$, thus using more LUTs and registers.

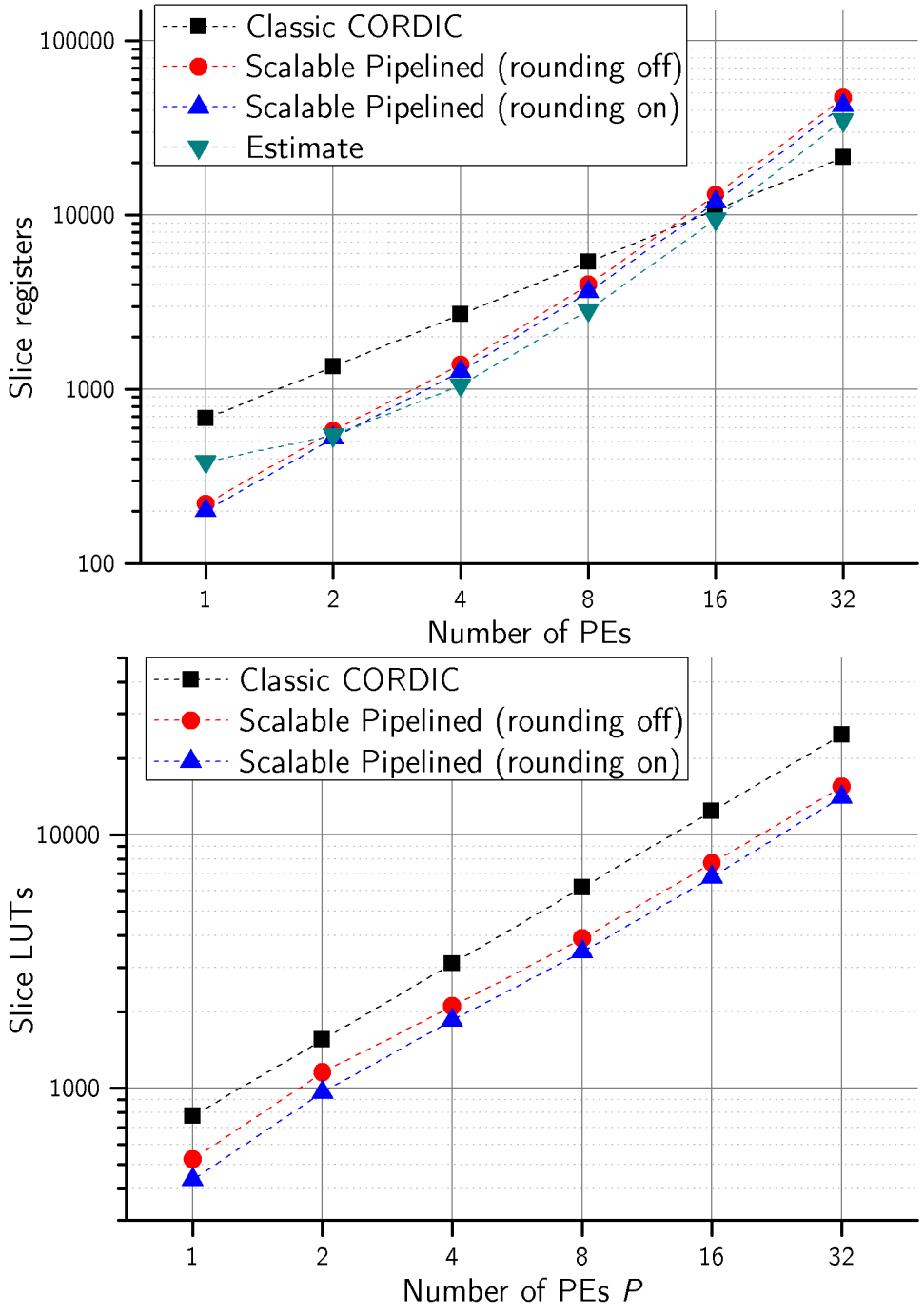

Figure 9.32: Synthesis results for the pipelined architecture as a function of P.

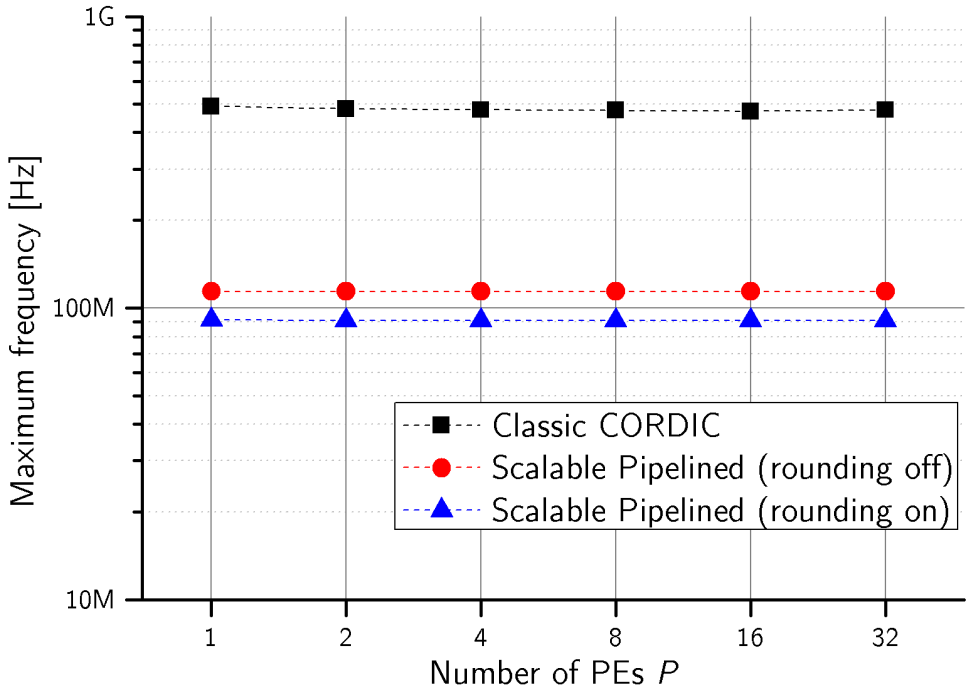

Figure 9.33: Maximum operating frequency of the pipelined architecture as a function of P.

Fig. 9.35 shows the plot of maximum operating frequency as a function of $B' - B$, being N fixed at 1024, $P = 4$ and $B = 16$. As expected, the architectures with the highest value of this figure of merit are the ones with rounding turned off. In particular, the pipelined architecture achieves a frequency above $110MHz$ when $B - B' \leq 2$. On the other hand, the shared core architecture with rounding enabled falls welow the required $70MHz$ when B' is bigger than B of five bits. We also note that the pipelined architecture with rounding enabled operates in a range of clock frequencies of about $10MHz$ around $90MHz$ when $B' = B$ and $80MHz$ when $B' - B = 5$.

Given the considerations in Sec. 9.3.3, synthesized ROM blocks are in a number of $P+1$ for the shared core architecture and $P+2$ for the pipelined one. These are implemented into LUTs by XST in order to maximize timing performances. The multi-port ROM of both hybrid architectures stores $\log_2 N - 1$ values, like the LUT described in [129]. The CORDIC architecture described in [33] uses no ROM, but hardware resources requested by the control are noted to be a drawback for FFTs of less than 1024 points. Furthermore, being based on Alg. 1, the proposed architectures do not perform any gain correction. Their drawback is in the maximum frequency, which is lower than in the forementioned architectures in the state of the art.

Considering required hardware efficiency, operating frequency constraints and accuracy, the best choice in the case of the commissioned project is the pipelined design with rounding enabled. This achieves both low register and LUT usage while setting the operating frequency above the lower limit of $70MHz$. In the general case, the most suitable architecture is determined by considering error constraints in the first place and then choosing the desired tradeoff between accuracy and resource usage.

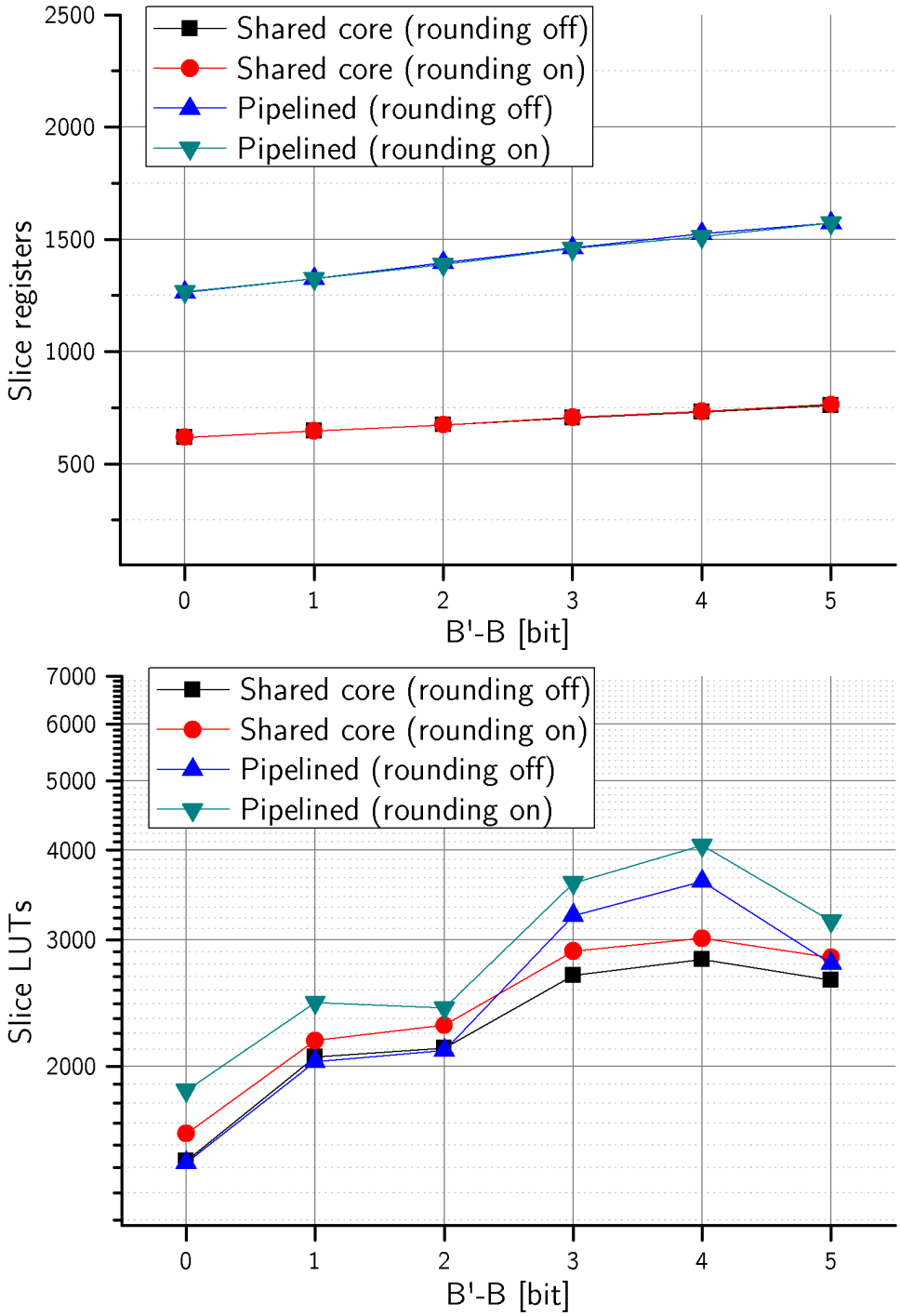

Figure 9.34: Synthesis results of hybrid architectures as a function of $B' - B$. $N = 1024$, $P = 4$ and $B = 16$.

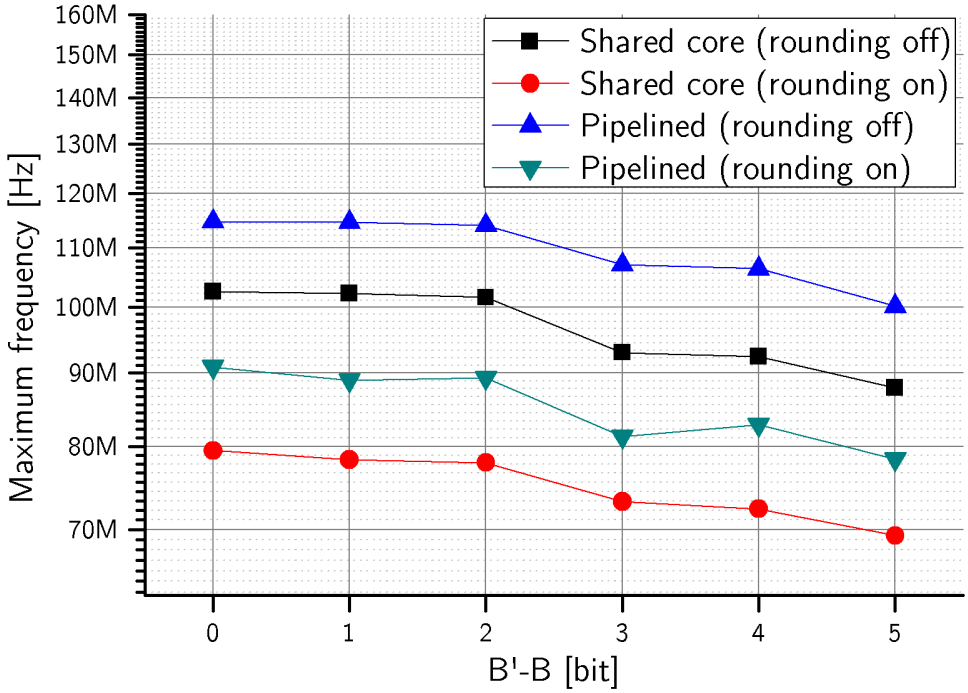

Figure 9.35: Maximum operating frequency of hybrid architectures as a function of $B' - B$. $N = 1024$, $P = 4$ and $B = 16$.

9.6.2 Scalable FFT processor

Accuracy

In order to study the accuracy of the implemented scalable FFT processor, we consider the set of input data given by the customer as a benchmark. The first 100 samples belonging to this array are shown in Fig. 9.36. These values are the output of an ADC converter and the parameters of the implemented processor are determined by considering the features of their processing.

Determination of bit width for both twiddle factors and coefficients data of the processor takes accuracy into consideration together with hardware resource saving. If $\boldsymbol{X}_{\text{exact}}$ is the exact result of the FFT, and $\boldsymbol{X}_{\text{obtained}}$ is the array of obtained complex values, we define the mean relative error as

$$E = \text{mean}\left(\frac{|\boldsymbol{X}_{\text{obtained}} - \boldsymbol{X}_{\text{exact}}|}{|\boldsymbol{X}_{\text{exact}}|}\right). \tag{9.26}$$

Fig. 9.37 shows the trend of E as a function of both TF and coefficient length expressed in bit. The plot is obtained by simulating the processor with a hybrid pipelined TF generator with rounding enabled. For the coefficient data, five bits are reserved to the integer part of the word, in order to avoid possible saturation when computing sequences other than the benchmark. Considering that the processor may operate multiple times on the same dataset, like required from Alg. 3, it is necessary to guarantee low error propagation. If we choose to impose $E \leq 1\%$, We note that at least 16 bits are needed both for TFs and coefficients. By setting the number of coefficient word width to $18bit$, we ensure E to be under 0.5%.

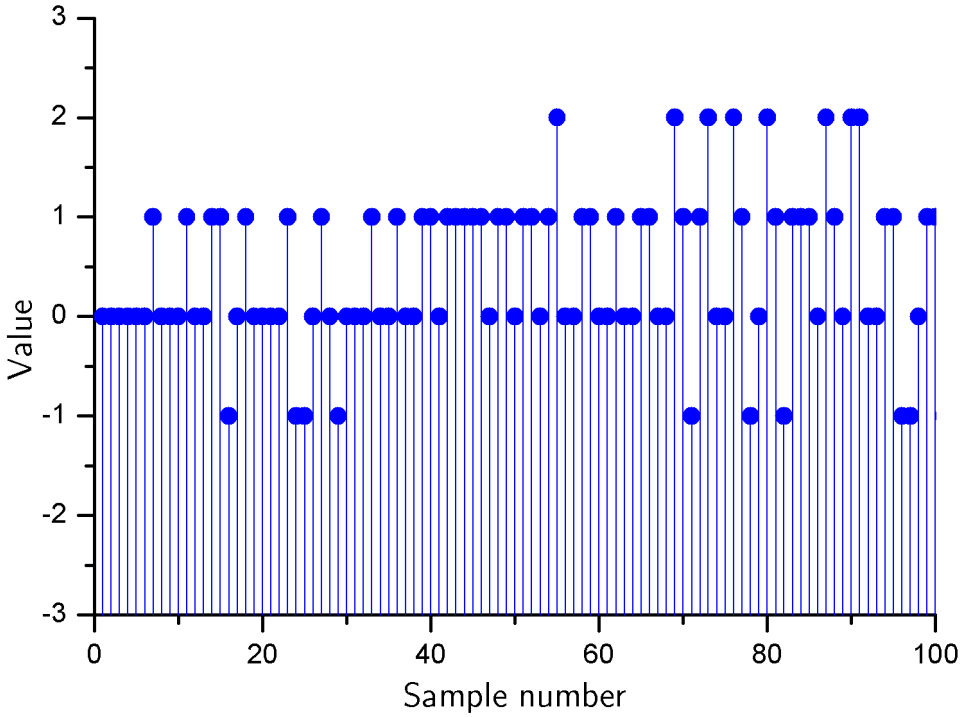

Figure 9.36: Samples from the dataset.

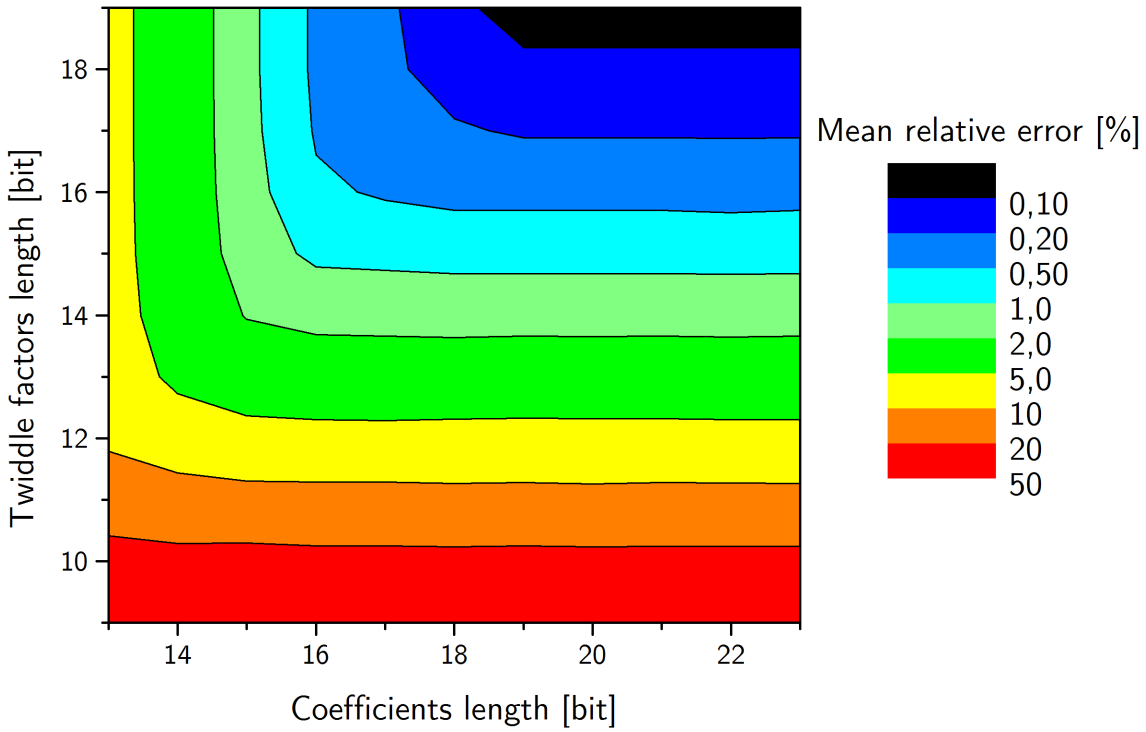

Figure 9.37: Mean relative error as a function of the length of twiddle factors and coefficients of the FFT processor. $N = 1024$.

Timing

As discussed in Sec. 9.1.1, the number of clock cycles to perform the computation of the FFT depends on the number of PEs P. Considering that a stage consists in calculating $\frac{N}{2}$ butterflies, we have

$$T_{\text{comp}} = \frac{N}{2P}(\log_2 N - 1), \tag{9.27}$$

where T_{comp} is the computation time in clock cycles. When the chosen architecture for the TF generator is the shared core one, Eq. (9.27) is not valid anymore. Considering the implementation discussed in Sec. 9.3.3, every time that a CORDIC iteration is performed, one clock cycle is used for the synchronisation between Moore machine and shared core. This is additional to the necessary cycles for the actual computation. Moreover the speedup effect must be taken into consideration. This makes the formula for T_{comp} a complicated function of P and N.

Evaluating Eq. (9.27) for $N = 1024$ and $N = 2048$ leads to the trend in Fig. 9.38. Notably, choosing $P = 4$ leads to $T_{\text{comp}} = 1152$ if $N = 1024$ and $T_{\text{comp}} = 2560$ if $N = 2048$. Given results in Sec. 9.6.1, we choose this number of PEs as a compromise between hardware efficiency and computational speed.

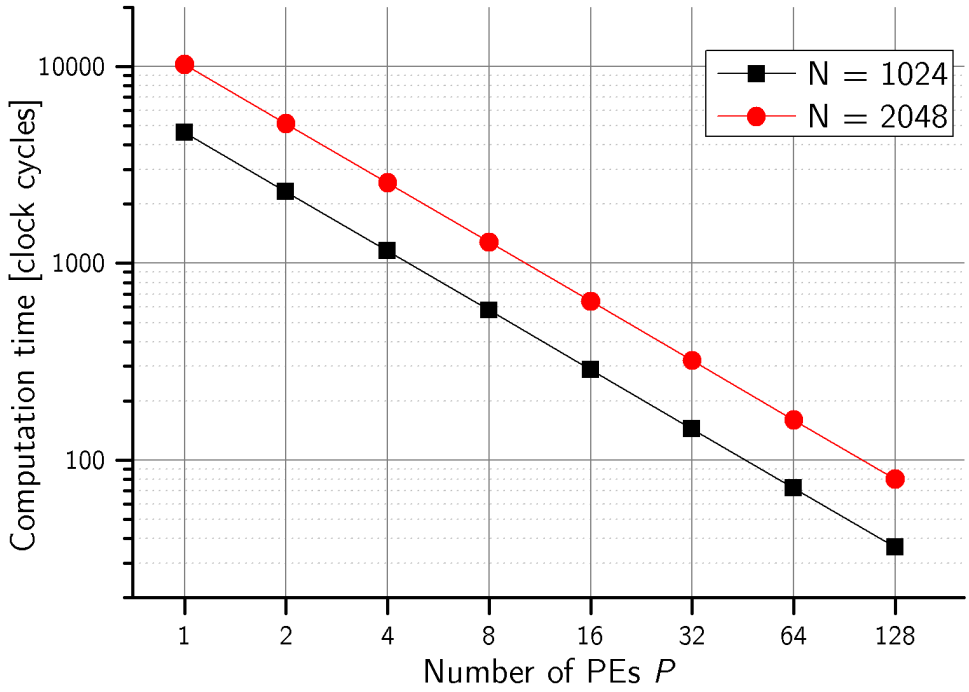

Figure 9.38: Computation time of the FFT processor as a function of P.

Synthesis results

The scalable FFT processor with hybrid TF generators is synthesized on the forementioned Xilinx Virtex-5 FPGA with different parametric configurations. In order to reduce the space of parameters, only the pipelined architecture with rounding enabled and internal representation of data of $16bit$ is considered. Likewise, the length of data words for the processor is fixed at $18bit$, given accuracy considerations in Sec. 9.6.2. Fig. 9.39 illustrates synthesis

results for different values of N as a function of the number of PEs P. Both the plot of of occupied slice registers and LUTs have an exponential trend, as expected. We can also note that, while data for $N = 512$ and $N = 1024$ do not differ very much for the same value of P, in order to achieve similar resource occupation with a bigger N it is necessary to reduce P. However, this also implies a reduction of T_{comp}, as discussed before. On the other hand, the trend for the maximum operating frequency (Fig. 9.40) is almost independent from P until this reaches $P = 8$. When such a number of parallel PEs is instantiated, the figure of merit is evidently reduced. This is because the *step_computer* block described in Sec. 9.3.3, which computes the needed step for rotational iterations, must examine more mapped addresses (see Alg. 1). Moreover, with the increase of N, these signals become also wider, and this implies an additional reduction of frequency. Despite this fact, the range of interest of P for practical applications falls inside the range $(0,4)$. If necessary, the frequency reduction effect can be easily compensated by implementing the *step_computer* block with a pipelined structure.

Discussed synthesis results show the basic advantage of the scalable architecture against pipelined FFTs, which is the possibility of choosing the number of PEs in function of desired hardware resource occupation. For example, the MDC pipelined architecture implemented in [79] on a Xilinx Virtex-5 FPGA for $N = 1024$ and $16bit$ for computational data uses much more DSP slices (i.e. real multipliers) if compared to the proposed approach. This can be seen in Tab. 9.3. Nonetheless, the scalable architecture with pipelined hybrid TF generator takes less clock cycless to perform the computation. On the contrary, the MDC structure requires less registers and LUTs but 20 BRAM blocks. As previously observed, the ROM in the hybrid TF generator is realised by the synthesizer with LUT blocks. Additionally, the whole scalable architecture is based on shifting registers and does not use any RAM. This explains hardware overhead observed in Tab. 9.3. We accept this considering the future ASIC implementation of the design, and the fact that the design will not be mapped on existing hardware like in the FPGA case. A remark can be done on the in place architectures discussed in [79], in particular on the one with five PEs. Although it requires less slice registers than the scalable FFT, it uses more LUTs together with BRAM blocks. Despite this, its computational latency is more than two times higher than the value referring to the scalable architecture.

Table 9.3: Comparison between the proposed architecture and [79]. $N = 1024$, $B = 16$.

	Scalable (4 PEs)	In place (1 PE)	In place (5 PEs)	MDC
Slice registers	6465	601	2384	2545
Slice LUTs	6901	4656	7453	2480
BRAM blocks	0	0	14	20
DSP slices	24	4	20	40
Computation time [clock cycles]	1152	6144	3687	2055

Another set of architectures specifically designed for automatic generators like [79] was proposed in [100]. The goal of these architectures is to put together hardware optimisation and design flexibility, like in the scalable FFT architecture. The difference is in the approach, which in [100] is fully-pipelined both in the radix-2 and radix-4 solutions. The used FPGA device belongs to the Xilinx Virtex-5 family, and synthesis results are given in Fig. 9.41. A synthesis with comparable parameters was performed by choosing $N = 256$, $8bit$ of reso-

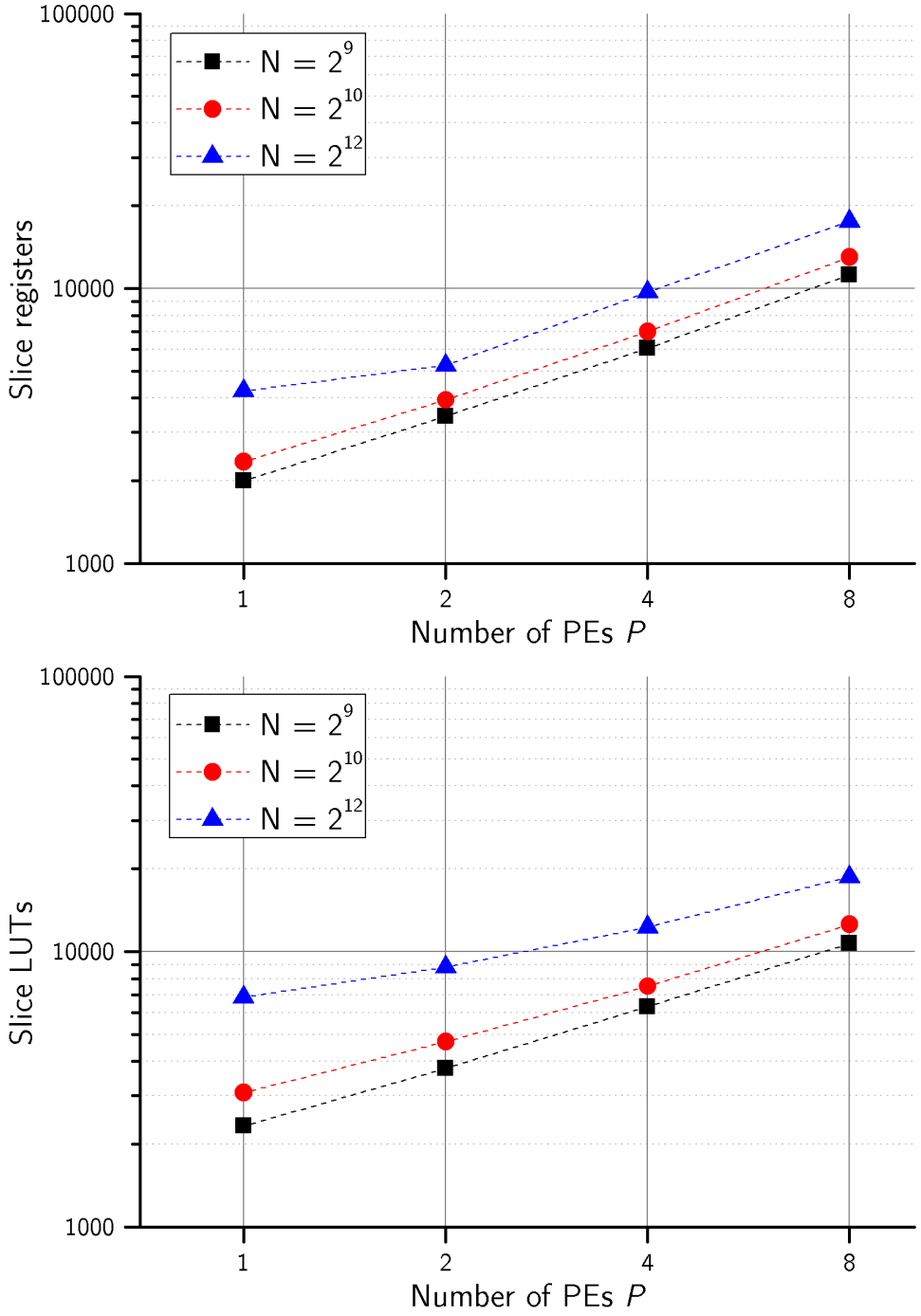

Figure 9.39: Synthesis results of the FFT processor as a function of P.

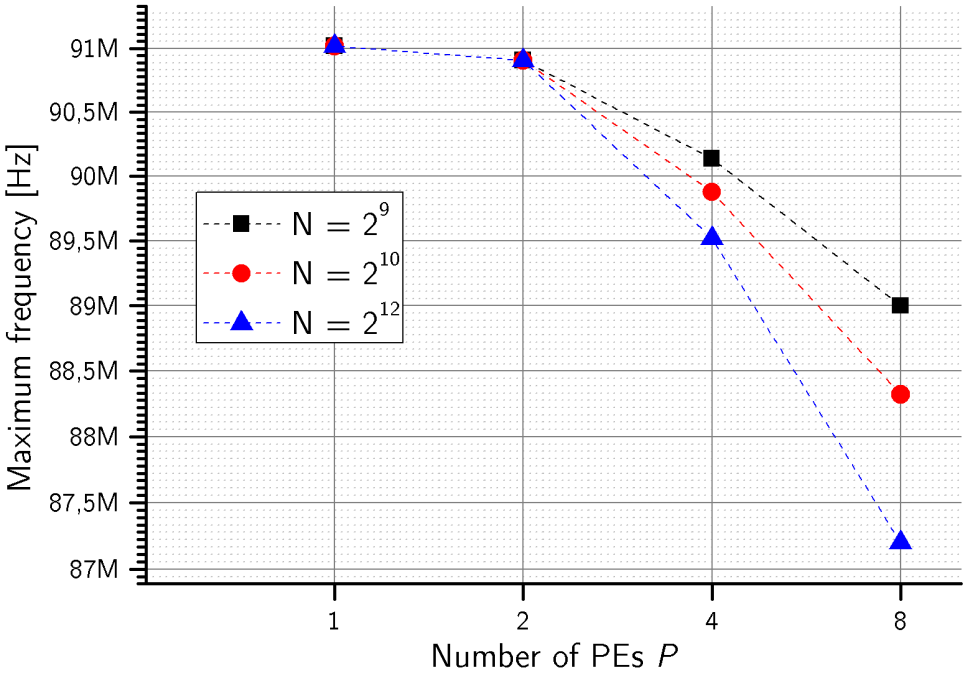

Figure 9.40: Maximum operating frequency of the FFT processor as a function of P.

lution of both twiddle factors and coefficients (i.e. computational data) and four PEs. The results are shown in Tab. 9.4. From the figures it is evident that the proposed architecture achieves improved slice saving as well as a similar operating requency. Because of its scalability, the architecture allows good timing performances. indeed, with 4 PEs it is possible to compute the transform in only 224 clock cycles.

Table 9.4: Synthesis results for the scalable archtiecture with $N = 256$, $B = 8$ for both TFs and coefficients and $P = 4$.

Metric	Value
Used slices	3114
Maximum frequency [MHz]	92

XionLogic [125] releases competitive FFT cores with an open source license. Among these, there is also a CG-FFT module. Synthesis results on a Xilinx Spartan-6 LX-16 FPGA are given by the producer. In Tab. 9.5, these are shown together with the ones of the scalable architecture, for a system with one PE, $10bit$ twiddle factors and $8bit$ computational data representation. The proposed architecture is proved to have lower operating frequency and greater slice register occupation if compared to all the considered architectures. This is the drawback of the FPGA implementation of the scalable architecture. This uses registers to perform data shuffling instead of memory addressing. On the other hand, it is worth noticing that the R2^{2}SDF architecture uses more LUTs, BRAM blocks and DSP if compared to the scalable approach. Also, all architectures implement memory with BRAM blocks, while, given the small dimensions of the ROM used by the hybrid TF generator, this is synthesized on other LUTs. Each BRAM block of a Spartan-6 FPGA contains up to $18Kbit$ of data [124], thus even if XionLogic designs are optimized for FPGA applications, hardware saving is not

guaranteed for ASIC platforms. Furthermore, CORDIC approaches are more effective as N increases, while conventional LUT-based solutions require bigger memories.

Table 9.5: Comparison between the proposed architecture and [125]. $N = 1024$, $B = 8$.

	Scalable (1 PE)	XionLogic R2^{2}SDF	XionLogic DIF	XionLogic CG
Slice registers	1256	927	438	392
Slice LUTs	1426	1987	388	331
BRAM blocks	0	2.5	4	6
DSP slices	6	16	4	4
Operating frequency [MHz]	52.982	100	100	100

In general, the advantage given by the scalable approach is in the tradeoff between computational speed and hardware saving. As shown in Tab. 9.6, hybrid CORDIC-LUT scalable architectures extend this concept in the state of the art [107, 75] by introducing scalability into TF generation. With this expedient, the total storage for twiddle factors is reduced depending on N from a linear to a logarithmic trend. Also, the choice of using three real multipliers instead of four, according to Eq. (9.4), allows to compensate their usage in hybrid TF generators at the expense of a reduced operating frequency. Being the design expressly oriented towards applications in which frequency is not critical, this is an acceptable compromise.

Table 9.6: Comparison between the proposed scalable hybrid CORDIC-LUT architecture and other architectures in the state of the art.

Architecture	total TF storage	Real Multipliers
Scalable hybrid shared core	$\log_2 N - 1$	$3(P+1)$
Scalable hybrid pipelined	$\log_2 N - 1$	$6P$
Scalable [107]	N	$4P$
Pipelined [32]	$\frac{3}{8}N - 1$	$4(2\log_2 N - 4)$
Scalable [75]	N	$4P$
In place [114]	$N/4$	3
In place [3]	N	4

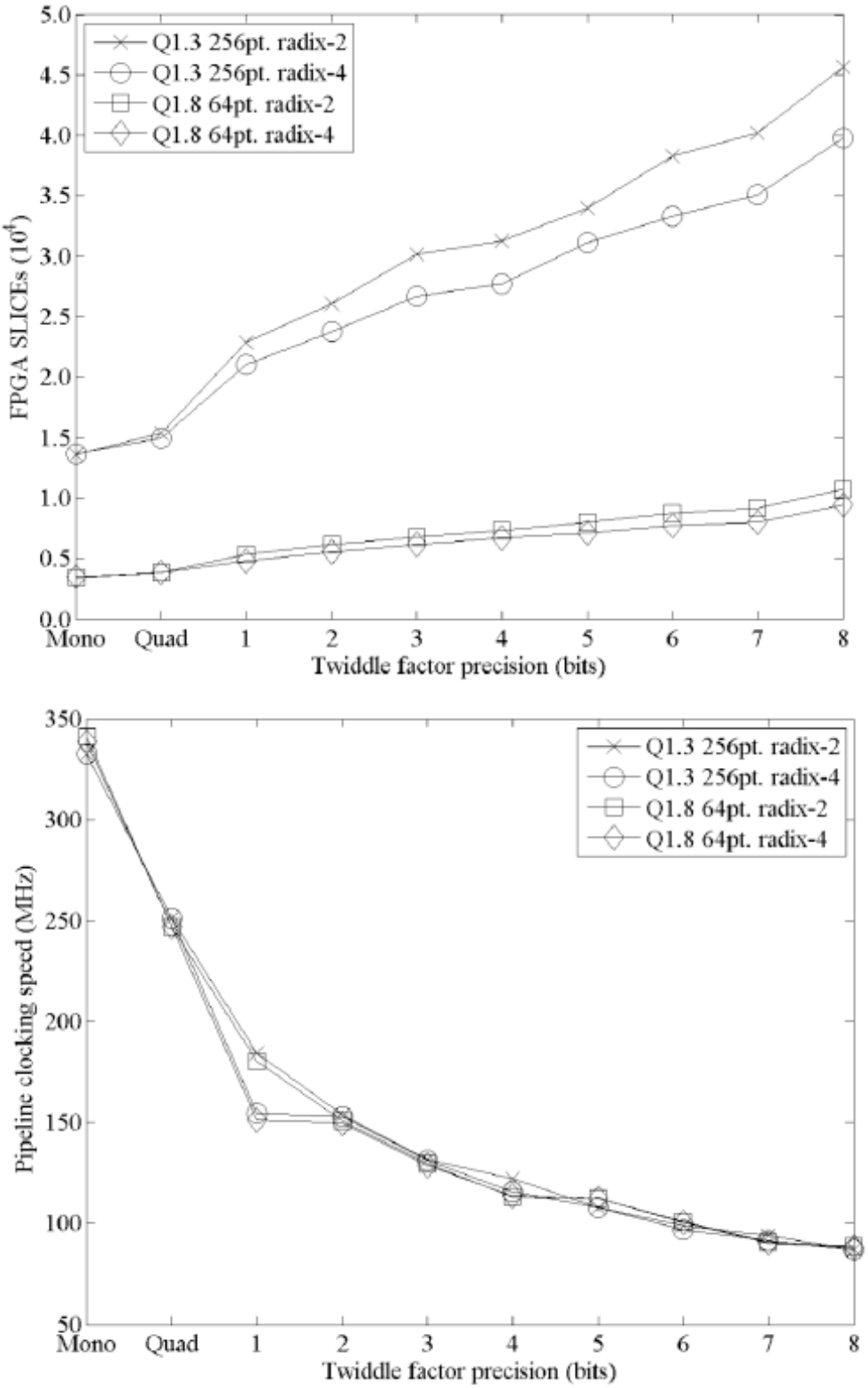

Figure 9.41: Performances of generated architectures. From [100].

9.6.3 PSD computer

Accuracy

The processing perfomed by the PSD computer is composed of several steps, each one involving the computation of a direct or inverse FFT, a square modulus, possibly a square root and a scaling. Moreover, if input data differs from the benchmark and parameters are too much optimized on the given data set, the system lacks of robustness. Therefore, finding metrics to evaluate the accuracy of the system can be an issue. We note that a slightly different sequence of samples for input, can lead to the saturation of a value in one step of the FFT and consequently lead to a big mean relative error. Thus, the mean relative error is not very suitable to the discussed case.

Considering the specific application of the PSD computer as given by the customer, what matters for the downstream modules is the identification of a trend. The chosen metric is subsequently the *correlation coefficient* [80], defined for two random sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ as

$$\rho_{x,y} = \frac{\mathrm{cov}(x,y)}{\sigma_x \sigma_y},$$

where $\mathrm{cov}(x,y)$ is the covariance of $\boldsymbol{x}$ and $\boldsymbol{y}$. In the discussed case, $\rho_{x,y}$ is a (2×2) matrix, and the actual coefficient can be found in the top right or bottom left elements. The correlation coefficient is a value between -1 and 1. In particular, its absolute value is 1 when $\boldsymbol{x} = \alpha \boldsymbol{y}$, with $\alpha \in \mathbb{R}$. We thus consider

$$\rho = |\rho_{\mathrm{obtained,exact}}| \tag{9.28}$$

as the accuracy metric for the PSD computer, where $\boldsymbol{a}_{\mathrm{exact}}$ and $\boldsymbol{a}_{\mathrm{obtained}}$ are respectively the evaluation of Eq (9.15) with MATLAB in full resolution and the output array.

Discussions from the former sections of this chapter have explored in detail the effect of varying the number of bits in the twiddle factor representation. In order to maximise the accuracy metric in Eq. (9.28), we simulate the behaviour of the system by taking advantage of the software environment described in Cha. 9.5. Experimentally, $18bit$ for data representation and a scaling factor value of 6 allow to achieve a high value of ρ. With these settings, data in the RAM memory has values in the range $(0,16)$ at the end of the first stage of the overall DSP processing, as shown in Fig. 9.42. Moreover, values are symmetrical from the center of the obtained array. Possibly the most critical stage is the second one. The output of the FFT processor can have very small square moduli, as illustrated in Fig. 9.42, consequently the value of the scaling factor is essential to achieve a good result in the following processing. By setting parameter *scaling_factor_c* to 3, at the end of the third stage of Alg. 3, data is like in Fig. 9.44. We can see that the profile of the reference result computed with MATLAB is well followed, and the accuracy metric ρ is equal to 0.9978. Consequently we can say that accuracy requirements are fulfilled by the implemented architecture.

Synthesis results

The PSD computer was synthesized on the forementioned Xilinx Virtex-5 FPGA. Being accuracy fixed from previous considerations, the only parameter that can vary depending on desired timing performances is the number of parallel words of the bus P. Increasing P means not only instantiating a bigger FFT processor, but also implementing a more dense matrix of BRAM blocks inside the ROM, together with more modules working in parallel. This is deducible from the datapath of the system in Fig. 9.20.

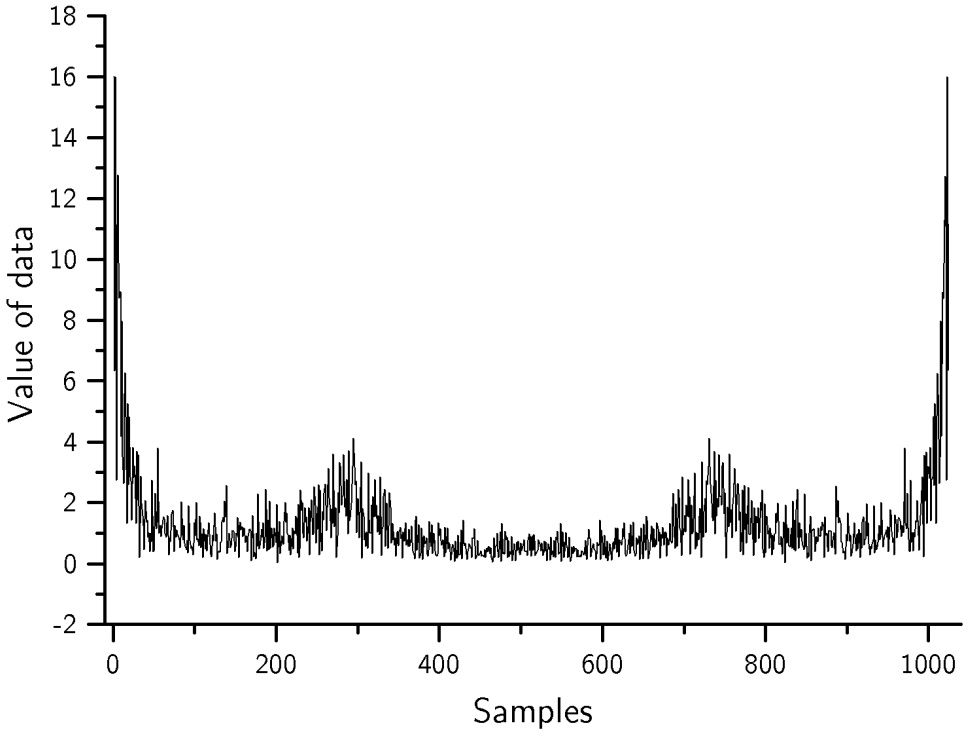

Figure 9.42: Data at the end of the first processing stage.

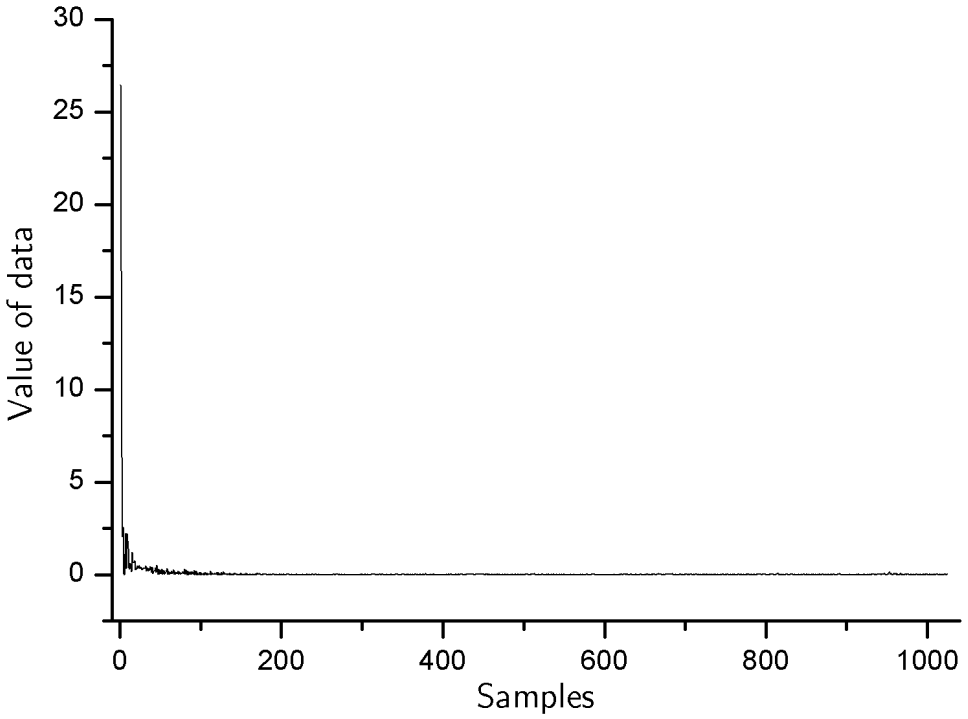

Figure 9.43: Square moduli at the second processing stage before scaling.

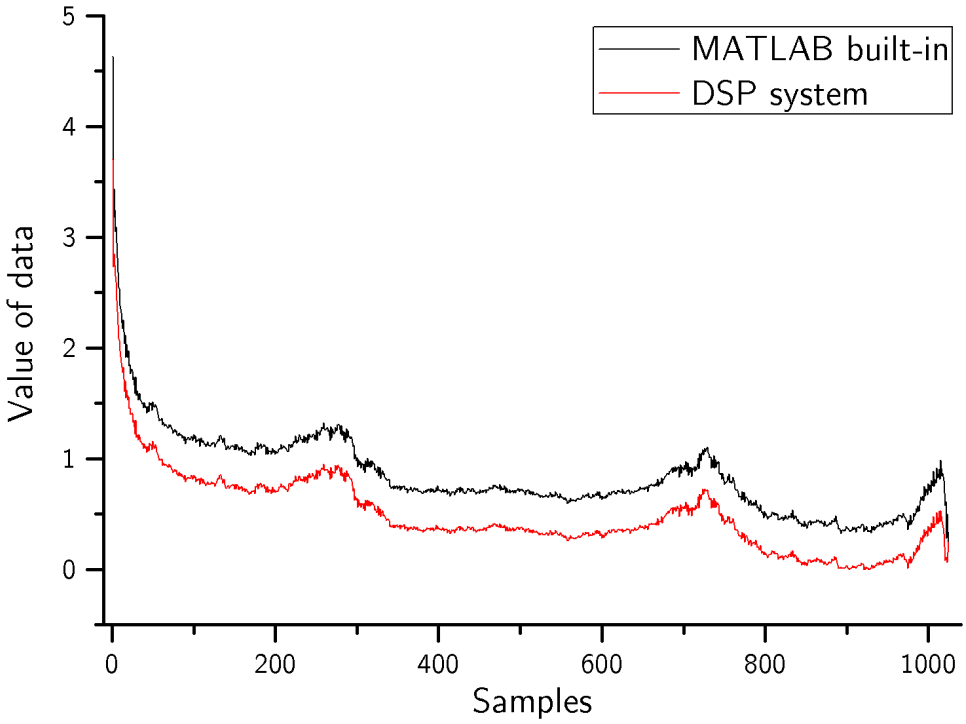

Figure 9.44: Data at the end of processing.

Slice resource occupation of the synthesized PSD computer is shown in Fig. 9.45. From the figures, it is evident that the number of requested LUTs grows faster than slice registers. This is because the increase of P leads to the instanstiation of more logic for routing of data among different channels inside the system bus. In particular, the multiplexers that manage data routing towards BRAM blocks become wider and less hardware-effective. The complexity of the system is evident if we consider that the Virtex-5 model used is one of the largest FPGAs of its device family, but it is nearly saturated by setting $P = 8$. Moreover, the matrix of RAM units described in Sec. 9.4.2 is implemented with BRAMs only in the case $P = 1$, else it is expanded into LUTs in order to achieve the speed goal. The usage of DSP blocks is shown in Fig. 9.46: as expected, the followed trend is exponential.

The maximum operating frequency of the system as a function of P is shown in Fig. 9.47. The plot shows fulfilment of frequency costraints in all the cases, as expected from the previous analysis.

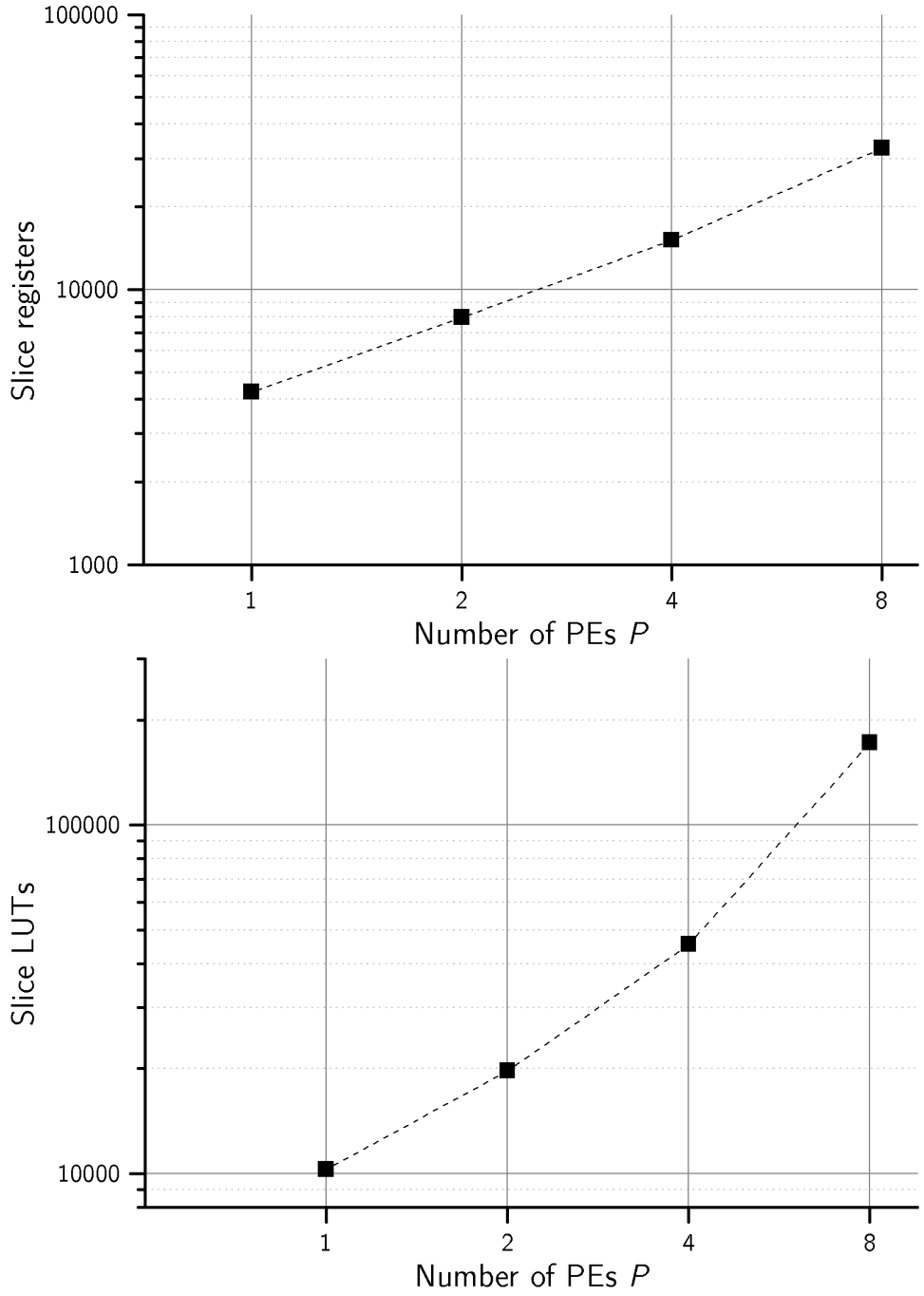

Figure 9.45: Slice resource occupation of the PSD computer as a function of P.

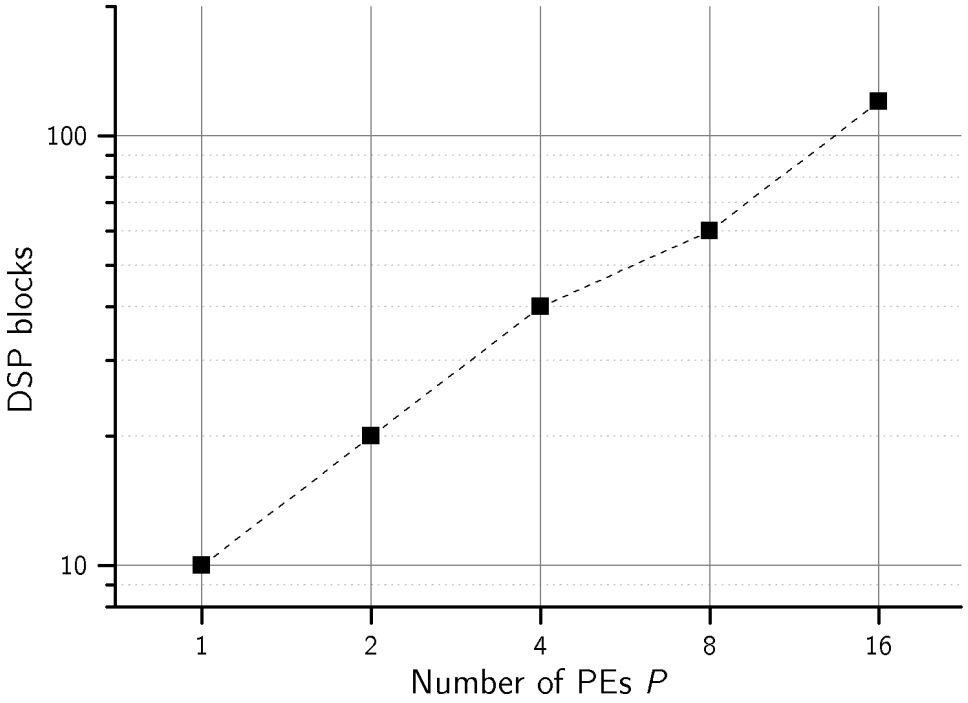

Figure 9.46: DSP blocks occupation as a function of P.

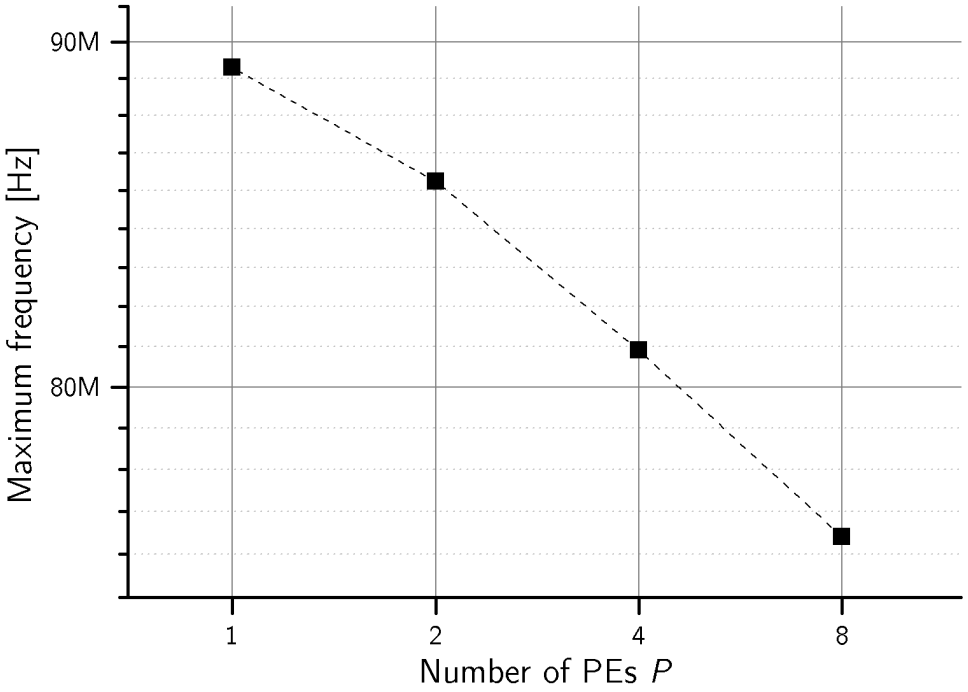

Figure 9.47: PSD computer maximum frequency as a function of P.

Chapter 10

Conclusions

The entire research presented in this thesis has been focused on Innovation Project 2010, shared between the EOLAB group of the University of Cagliari and the Automotive Department of Infineon Technologies AG (Villach-Austria). Aim of this activity is to develop and execute an innovative self-tuning prototype (on-line controller) for future automation technologies to allow the integration of power supplies for different set of applications reducing development time and R & D costs. On-line controllers for digitally controlled Switching Mode Power Supplies (SMPS), are able to automatically set the PID compensator gains for a specific identified load, while, off-line controllers have static compensator parameters tuned over a wide range of loads. Self-tuning algorithms for SMSP are part the on-line controllers. They have the capability to identify the output filter configuration of the converter and, consequently, adjust both bandwidth and margin of the system. The load identification (or system identification) phase is usually performed by injecting a perturbation in the system (non-parametric approach). In the realised self-tuning prototype, the system identification (SI) can be accomplished during steady state operations of the converter. The algorithm, is able to set the best PID configuration even if non-idealities occur during the converter operations. When temperature variations occurs the system dynamic can be compromised by the ESR contribution on the output capacitor. Steady state system identification (SI) algorithms are able to compensate this non-idealities, the zero introduced by the ESR contribution can be detected and the both margin and bandwidth can be modified by changing the compensator structure. On the counter part, open loop SI approaches do not run during the steady state, and load identification is done before or during the system start up. Consequently, the PID configuration is set for the identified load but cannot be updated during steady state operations. The main advantage of open loop SI methods is, that the perturbation is not injected during converter operations, however, steady steady SI approach has to perturb the system without compromising the nominal operating conditions of the converter.
The self-tuning prototype has been designed for automotive applications, it has been accomplished and qualified integrating a digitally controlled buck converter (off-line controller) with a full-scalable VHDL-code PSD computer for the Self-Tuning algorithm execution. It offers a fully customizable prototype as a development platform for future concepts.

With this purpose, a FPGA-based digital control loop for SMPS has been design. The prototype has been mapped on Virtex6 FPGA presenting a maximum frequency of $124.748 MHz$,

using respectively 402 and 1123 of slice registers and slice LUTs. The implemented control loop has been designed for the FPGA prototyping. In order to increase the maximum frequency of the digital control (system speed up), the signals among the main control blocks have to be registered and the obtained latency of computation is of four clock cycles. Avoiding output registers in the design, the digital feedback prototype could be used for low-latency ASIC implementations. However, the obtained maximum frequency is enough for automotive purposes.
The digital resolution of the control feedback is of 28 bits for the PID compensator and 12 bits respectively for the $\Delta\Sigma$ modulator and the DPWM. The tuning of these values have been first addressed comparing fixed and floating point Matlab/Simulink closed loop models. The resolution validation has been accomplished with mixed signal hardware-software (VHDL-Matlab) FPGA-based co-simulations. The robustness of the designed control loop has been proven with the capability to recover the dc output voltage level when a load step occurs.

A robust digitally controlled buck converter (off-line controller), has been realised interfacing the aforementioned FPGA-based digital control loop with a Test Chip (TC) where analog blocks were already implemented. Having the feature to be externally driven, the TC permits to verify the closed loop implementation. The system robustness has been confirmed. The designed FPGA-TC digitally controlled buck converter prototype has been validated and it can be exploited for future applications up to $124.748MHz$.

Two novel non-parametric system identification (SI) algorithms have been theoretically introduced and then validate into a fixed-point Matlab/Simulink closed loop model. The open loop SI technique is based on the system step response, while, the steady state SI method on the amplification of dithering effects on the signal path of the digital control feedback. In both cases, output filter information are obtained in the frequency domain in terms of resonant frequency f_0 and possible ESR contribution at frequency f_z. When the respective perturbations occur, the output voltage is observed on the digital side at the ADC output and processed with PSD computation. The resonant frequency contribution represents the maximum in the PSD output while the f_z can be detected as second peak in the processing output. All identification results have been presented for systems having typical automotive specifications. A clock frequency $f_{clk} = 70MHz$ and a switching frequency $f_s = 450kHz$, are considered for buck converters having a corner frequency in the range of frequencies below the upper limit of $f_s/20$. The PSD computation is considered on a dataset of $N = 128$ samples, introducing a finite resolution $\Delta f = f_s/2N = 1.754kHz$ for the identified f_0 and f_z.
The open loop SI method identification outputs very precise results for all considered f_0 identification, the little amount of error in the identification result is due to the finite resolution of the processing. For instance, considering a buck converter configuration having resonant frequency $f_{0,buck} = 4.9kHz$, the obtained maximum in the PSD is $f_0 = 4.2kHz$. When ESR contributions are considered inside the buck convert bandwidth or very close to its corner frequency, a maximum error of $2\Delta f$ is introduced in the identified f_z. Moreover, the resonant frequencies obtained with ESR contributions are still precise, only an error of $1\Delta f$ is obtained for one buck converter configuration. A maximum error in the f_z identification, occur when the ESR contribution is approaching the system bandwidth ($f_z = 3f_0$).
The steady state SI analysis have been deeper characterized with multiple acquisitions. The first validation of the algorithm has been done in Matlab/Simulink closed loop model, then

dithering amplification effects have been evaluated into the off-line controller prototype. The perturbations considered for this non-parametric method have been characterised through a mathematical model and confirmed in every step, up to the self-tuning prototype realisation. The dithering amplification factor α has been studied in the Matlab/Simulink model, both in terms of perturbation of the output voltage and of impact on the load identification results. For considered system configuration ($70MHz$ clock frequency and 449 kHz switching frequency) about $100mV$ of average perturbation on the output voltage is introduced doubling the dithering effects ($\alpha = 2$), and f_0 identification results very precise for all considered converter configurations. To validate identification results with $\alpha = 2$, multiple processing have been considered. The related averaged values are very close to the desired ones and the little amount of introduced error is related only to the finite resolution Δf. The f_z identification results, have been obtained trough comparison between two different third order noise shaper structures when dithering effects are doubled. With a classical third order noise shaper, the identified resonant frequencies are very precise while the ESR contributions cannot be detect. Inserting a notch at frequency $f = f_0$ in the noise shaper, permits to detects the f_z contributions as peak in the PSD output. In this way, the dithering amplification effects are more concentrated on frequency where the ESR contribution is present. This trend has been confirmed considering four different values of ESR and precise results are obtained. When the ESR contribution is approaching the bandwidth, the introduced error on f_z is of about $3\Delta f$.
The identification results based on dithering amplification during steady state, have been validated inserting the α factor in the off-line controller. Results for the digitally controlled FPGA-TC, have been shown considering a fixed-point emulation of the PSD computer. In this case, the resonant frequency identification for $\alpha = 2$ has been averaged among eight acquisitions for each considered buck converter configuration. The obtained mean values are $f_{01,avg} = 4.38kHz$, $f_{02,avg} = 6.32kHz$ and $f_{03,avg} = 9.82kHz$ respectively for the buck converter configurations $f_{0,buck1} = 4.9kHz$, $f_{0,buck2} = 7.3kHz$ and $f_{0,buck3} = 10.7kHz$. Identified results are very close to the desired ones, and the error is lower the $1\Delta f$ when averaging is computed. The ESR contribution has been obtained with the same approach used for the Matlab/Simulink fixed-point model. The ESR contribution can be detected as a second peak thanks to the combined effects of both dithering amplification and modified noise shaper. If for instance a $ESR = 1\Omega$ is considered for $f_{0,buck} = 7.3kHz$, the zero contribution is identified at $f_z = 15.8kHz$ for an expected value of $f_{z,buck} = 15.9$. The bigger error is obtained when the ESR contribution is approaching the bandwidth ($4\Delta f$), however, the ESR contribution in this case is vary little and has been considered for a frequency f_z three times bigger than f_0.
After that the steady state SI method has been validated, this algorithm has been integrated into the digitally controlled buck converter. An on-line steady state self-tuning prototype has been realised. This controller is able to identify the resonant frequency and, when the zero contribution is present, it extracts f_z modifying the noise shaper. In this case two steps are needed, the first to identify f_0 and the second to modify the third order noise shaper inserting a notch at $f_n = f_0$. To prototype the entire self-tuning SI algorithm, a full-scalable PSD computer has been VHDL-coded and integrated into the FPGA. Identification results can be observed directly on the FPGA and presented through a Chipscope Pro Analyzer interface, where the extracted f_0 and f_Z are shown in terms of frequency corresponding to the related sample of the PSD output ($f = f_s/2N$). When eight successive acquisitions are performed, the identified results are averaged. The user can activates the SI algorithm, after every iden-

tification he can decide either to repeat the identification or to change the PID coefficients. New compensator gains are retrieved from a LUT in this implementation.
The trend for the resonant frequency identification is confirmed, precise average results can by obtained by doubling dithering effects. Moreover, the extracted resonant frequency are not affected by the ESR contribution. When ESR contribution are considered, the two step SI algorithm is able to output results very close to the desired f_z.
During the identification, PID gains are the same used with the off-line controller approach. They are fixed for the entire range of considered loads. The on-line controller permits to tune the PID configuration for the identified load. Once that the identification is computed, the PID gains can be updated consequently to the identification to adjust system dynamics in term of bandwidth. In this case, the recovery time when a load step occurs, can be more than $1ms$ shorter than the PID gains configured for the off-line controller.
The steady state SI algorithm have the main advantage to identify non-idealities during the converter operations, without adding any overhead in terms of resources. Dithering amplification can be easily obtained playing on the quantizer resolution, the hardware overhead in the on-line prototype is added by the PSD computer. For a Virtex6 FPGA, the maximum frequency of the digital control loop is reduced at $91.86MHz$ when the PSD computer is introduced, while, the FPGA resource usage is 21583 and 21135 respectively for Slice Registers and Slice LUTs. However, the PSD computer is composed by Processing Elements (PEs) which are operating in parallel, its scalability permits to reduce the resource usage increasing the latency of the computation. For these reasons, different scalable FFT structures have been VHDL-coded and integrated into the PSD computer to find the best compromise between computation latency and resource usage.
Considering this requirements of scalability, a scalable FFT architecture in the state of the art was modified and extended by using a novel approach to twiddle factor generation. The procedure, takes into consideration the structure of the CG-FFT algorithm in order to compute twiddle factors from a diminutive set sized $\log_2 N - 1$, where N is the length of the FFT. Each element of the obtained sequence of TFs is either computed starting from another one or directly retrieved from the forementioned set. This also contains the data which is necessary to perform rotations. There are two possible variations of the algorithm, depending on the interpretation followed and on the chosen property of the CG-FFT that one wants to exploit. Each of them leads to a different architecture for the TF generator. Being both based on the usage of a LUT and iterations, these systems have been called hybrid CORDIC-LUT twiddle factor generators.
The shared core architecture uses a single rotational engine to perform all computations. Requests for TFs are scanned and then organised to allow the shared core to compute each different value only once. Thus, if two requests are referred to the same twiddle factor, only one iteration is performed and both are satisfied at the same time. The pipelined architecture is composed instead of a number of stages equal to the PEs of the system. Each stage performs the computation of a request by possibly applying phase rotations. Both architectures achieve the calculation of TFs by using complex multipliers realised with three real multipliers, as in Gauss'complex product algorithm. The design of the processor is based on a tradeoff between operating frequency and hardware resource saving. By exploiting the novel scalable rotational algorithm, the number of hardware stages used to compute TFs on the fly during the FFT is reduced according to the number of PEs. Depending on desired timing and hardware usage, the parameters of the processor can be adjusted to fulfil all requirements.

The whole system and all its components were tested by using a simulation and testing environment, composed of a set of MATLAB software and VHDL testbenches. The environment was developed to allow an easy testing of many combinations of parameters.
The designed processor and the PSD computer were both synthesized on a Xilinx Virtex-5 FPGA. While the pipelined structure achieves the highest possible throughput, the shared core architecture must serve different requests in sequence, and thus it is slower. But as the processing of the FFT goes on, the number of different requested TFs decreases, so the system shows a speed up. If a bigger error is acceptable, the shared core architecture is proved to be even more hardware efficient if compared to the pipelined one.
Given the same mean relative error of the classic CORDIC case, the pipelined architecture uses less hardware because it can more easily compensate error propagation than the shared core structure. Compared to the conventional CORDIC, both solutions are proved to be more hardware efficient as the number of PEs of the system increases. Estimates based on stored computational data bits are derived and validated in order to figure resource occupation depending on the variation of parameters. All CORDIC structures in the state of the art are based on the same basic hardware structure, thus the latency of their processing is a function of the number of resolution bits of the word. Moreover, they require additional logic to perform the compensation of a scale factor, which can be different from iteration to iteration depending on the particular architecture. The proposed structures do not need any gain compensation, and their latency does not depend on data word length. The drawback of the proposed approach is in frequency reduction.
A study of the accuracy of the FFT is firstly given by examining the mean relative error as a function of the number of bits of both twiddle factors and computational data representation. The processor can achieve a value of this figure of merit of less than 0.5%, if 16 bits are chosen for TFs and 18 bits are chosen for coefficients. Latency of FFT processing is confirmed to decrease as the number of PEs of the system increases, and four PEs are observed to be the best compromise in the discussed implementation case. The processor has been synthesized on the same FPGA device as the TF generators. The design of the scalable architecture is based on a set of registers that implement the perfect shuffle permutation, while other processors in the state of the art take advantage of the RAM blocks of the FPGA. For this reason, slice LUT occupation can be higher than in other architectures. On the contrary, the effects of scalability are well shown both in used multipliers, or rather DSP blocks, and in the ROM location count for TFs. The majority of FFT processors require a memory of a size which is proportional to the number of points of the transform. The implemented processor uses a memory whose size is the logarithm of the FFT length. This make the discussed approach more convenient if compared to other ones as the length increases. Also, both the designed versions of the FFT tend to have a lower operating frequency if compared to other approaches, that are usually oriented towards general purpose applications or telecommunications. This is expected, because the whole system is coded in order to achieve improved hardware efficiency at the expense of frequency, considering is particular application.
The complete PSD computer system was both tested and implemented on FPGA. The chosen metric for accuracy is the absolute value of the correlation coefficient between results obtained with MATLAB bult-in functions and actual output of the PSD computer. The achieved value is 0.9978, which means that the trend of the output follows almost perfectly reference values. Synthesis results prove the scalability of the whole design as well as requirements fulfilment.

Future developments of this work, can refer to an on-line controller which integrates the

open loop SI technique into the steady state self-tuning prototype. In this way, the SI can be performed before the system start up and the steady SI technique can be used to monitor possible non-idealities during the converter operations. Both presented SI approaches extract load parameters through the PSD computation, this solution can be easily implemented without adding any resource overhead. Furthermore, the fully-customizable self-tuning prototype can be applied for different application. The entire discussion have been addressed with automotive specifications, but the approach can be extended to other fields. The SI algorithms have been described for buck converters, but they can by applied to other converter configurations.

A possible direction of the following design effort on the scalable processor is in the derivation of a generalised hybrid architecture. This could be a compromise between the shared core and the pipelined structure. Such a system could be composed of a tunable number of pipeline stages, each one containing a shared cordic core that would serves a small set of adjacent requests. This approach would allow to achieve improved hardware saving if compared to the pipelined architecture while guaranteeing better timing performances than the shared core architecture.

Another possible improvement of the architecture is the increase of its operating frequency. This would allow the system to be more general purpose, and could be achieved by improving the design of existing modules and by substituting used multipliers with synchronous multipliers. The first case involves the study of efficient solutions in order to achieve computation of TFs in the smallest possible time, while breaking the critical path or possibly adding other pipeline stages. The second approach requires an in-depth analysis of available multiplier architectures and possibly the design of a module that would allow clock frequency increase without adding too much latency in the processing. This development process can also be tightly linked to the future ASIC implementation of the processor, with the use of optimized custom hardware, whose layout should be carefully designed.

Bibliography

[1] Fast fourier transforms. [cited at p. xviii, 270, 271]

[2] J.A. Abu Qahouq and V. Arikatla. Online closed-loop autotuning digital controller for switching power converters. *Industrial Electronics, IEEE Transactions on*, 60(5):1747–1758, 2013. [cited at p. iv]

[3] A.A. Al Sallab, H. Fahmy, and M. Rashwan. Optimized hardware implementation of fft processor. In *Design and Test Workshop (IDT), 2009 4th International*, pages 1–5, 2009. [cited at p. xviii, 166, 181, 230, 283, 284, 290]

[4] M. Algreer, M. Armstrong, and D. Giaouris. Adaptive pd+i control of a switch-mode dcdc power converter using a recursive fir predictor. *Industry Applications, IEEE Transactions on*, 47(5):2135–2144, 2011. [cited at p. iv, v, 78, 85]

[5] Maher Algreer, Matthew Armstrong, and D. Giaouris. Active online system identification of switch mode dcdc power converter based on efficient recursive dcd-iir adaptive filter. *Power Electronics, IEEE Transactions on*, 27(11):4425–4435, 2012. [cited at p. iv, v, 78, 85]

[6] Altera Corporation. *FFT MegaCore Function*. [cited at p. xviii, 280, 285, 289]

[7] J. Astola and D. Akopian. Architecture-oriented regular algorithms for discrete sine and cosine transforms. *Signal Processing, IEEE Transactions on*, 47(4):1109–1124, 1999. [cited at p. 161]

[8] J. Astola and D. Akopian. Architecture-oriented regular algorithms for discrete sine and cosine transforms. *Signal Processing, IEEE Transactions on*, 47(4):1109–1124, Apr. 1999. [cited at p. v, 126]

[9] Adam Barkley, Roger Dougal, and Enrico Santi. Adaptive control of power converters using digital network analyzer techniques. In *Applied Power Electronics Conference and Exposition (APEC), 2011 Twenty-Sixth Annual IEEE*, pages 1824–1832. IEEE, 2011. [cited at p. iv, v, 85, 86]

[10] R. Bhakthavatchalu, N. Abdul Kareem, and J. Arya. Comparison of reconfigurable fft processor implementation using cordic and multipliers. In *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE*, pages 343–347, 2011. [cited at p. 181, 291]

[11] R. Bhakthavatchalu, N. Abdul Kareem, and J. Arya. Comparison of reconfigurable fft processor implementation using cordic and multipliers. In *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE*, pages 343–347, Sept. [cited at p. v, 128]

[12] David Bishop. *Fixed Point Package User's Guide*. [cited at p. 166, 169]

[13] M. Botao, R. Zane, and D. Maksimovic. Automated digital controller design for switching converters. In *Power Electronics Specialists Conference, 2005. PESC '05. IEEE 36th*, pages 2729–2735, 2005. [cited at p. iv, v, 78, 79]

[14] M. Botao, R. Zane, and D. Maksimovic. Practical on-line identification of power converter dynamic responses. In *Applied Power Electronics Conference and Exposition, 2005. APEC 2005. Twentieth Annual IEEE*, volume 1, pages 57–62 Vol. 1, 2005. [cited at p. iv, v, 78, 79]

[15] T. Carosa, R. Zane, and D. Maksimovic. Digital multiphase modulator; a power d/a perspective. In *Power Electronics Specialists Conference, 2006. PESC '06. 37th IEEE*, pages 1–6, 2006. [cited at p. iii]

[16] Ye-Then Chang and Yen-Shin Lai. Novel on-line parameter tuning technique for predictive current mode control operating in boundary conduction mode. In *Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE*, pages 715–722, 2009. [cited at p. iv, v, 78]

[17] Ye-Then Chang and Yen-Shin Lai. Online parameter tuning technique for predictive current-mode control operating in boundary conduction mode. *Industrial Electronics, IEEE Transactions on*, 56(8):3214–3221, 2009. [cited at p. iv, v, 78]

[18] Yun-Nan Chang and K.K. Parhi. An efficient pipelined fft architecture. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 50(6):322–325, 2003. [cited at p. 277]

[19] Jingquan Chen, A. Prodic, R.W. Erickson, and D. Maksimovic. Predictive digital current programmed control. *Power Electronics, IEEE Transactions on*, 18(1):411–419, 2003. [cited at p. iii, iv]

[20] Chao Cheng and K.K. Parhi. Low-cost fast vlsi algorithm for discrete fourier transform. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 54(4):791–806, 2007. [cited at p. 282]

[21] Eleanor Chu and Alan George. *Inside the FFT Black Box. Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press, 2000. [cited at p. 169, 265, 267, 268]

[22] Wanming Chu and Yamin Li. Cost/performance tradeoff of n-select square root implementations. In *Computer Architecture Conference, 2000. ACAC 2000. 5th Australasian*, pages 9–16, 2000. [cited at p. 200, 201]

[23] A. Congiu, M. Barbaro, A. Picciau, E. Bodano, and D. Hammerschmidt. Prototype of a novel steady-state load identification technique for digitally controlled dc-dc power supplies. In *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on*, pages 355–356, 2013. [cited at p. xvi, 129, 138, 147, 154]

[24] A. Congiu, A. Picciau, M. Barbaro, and E. Bodano. Scalable hybrid cordic-lut architectures for cg-fft processors. In *Ph.D. Research in Microelectronics and Electronics (PRIME), 2013 9th Conference on*, pages 105–108, 2013. [cited at p. viii, 128, 138, 159, 262]

[25] L. Corradini, P. Mattavelli, and D. Maksimovic. Robust relay-feedback based autotuning for dc-dc converters. In *Power Electronics Specialists Conference, 2007. PESC 2007. IEEE*, pages 2196–2202, 2007. [cited at p. iv, v, 34, 78, 79, 81]

[26] Alessandro Costabeber, P. Mattavelli, S. Saggini, and A. Bianco. Digital autotuning of dcdc converters based on a model reference impulse response. *Power Electronics, IEEE Transactions on*, 26(10):2915–2924, 2011. [cited at p. iv, v, 78]

[27] Ognjien Djekic and Miki Brkovic. Synchronous rectifiers vs. schottky diodes in a buck topology for low voltage applications. In *Power Electronics Specialists Conference, 1997. PESC'97 Record., 28th Annual IEEE*, volume 2, pages 1374–1380. IEEE, 1997. [cited at p. 8]

[28] Robert Erickson and Dragan Maksimovic. High efficiency dc-dc converters for battery-operated systems with energy management. *Worldwide Wireless Communications, Annual Reviews on Telecommunications*, pages 1–10, 1995. [cited at p. 7]

[29] Robert W Erickson and Dragan Maksimovic. *Fundamentals of power electronics*. Springer, 2001. [cited at p. 8, 81, 85]

[30] CASPER (Collaboation for Astronomy Signal Processing and Electronics Research) project Wiki. The polyphase filter bank technique. Online, accessed 2nd April 2013. [cited at p. 283]

[31] Wei Fu, Siang Tong Tan, and A. Fayed. Switching and conduction loss analysis of buck converters operating in dcm-only scenarios. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 921–924, May 2013. [cited at p. 8]

[32] JesÃžs GarcÃŋa, Juan A. Michell, Gustavo Ruiz, and Angel M. BurÃşn. Fpga realization of a split radix fft processor. *Proceedings of SPIE*, 6590, 2007. [cited at p. 230, 278, 280]

[33] M. Garrido and J. Grajal. Efficient memoryless cordic for fft computation. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 2, pages II–113–II–116, 2007. [cited at p. xviii, 181, 213, 217, 222, 290, 291]

[34] M. Garrido and J. Grajal. Efficient memoryless cordic for fft computation. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 2, pages II–113–II–116, April. [cited at p. v, 128]

[35] M. Garrido, J. Grajal, M.A. Sanchez, and O. Gustafsson. Pipelined radix-2^k feedforward fft architectures. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(1):23–32, 2013. [cited at p. 280]

[36] Mario Garrido Galvez. Introduction to the fft algorithm and its hardware architectures. [cited at p. 277]

[37] Shuibao Guo, Yanxia Gao, Yanping Xu, Xuefang Lin-Shi, and B. Allard. Digital pwm controller for high-frequency low-power dc-dc switching mode power supply. In *Power Electronics and Motion Control Conference, 2009. IPEMC '09. IEEE 6th International*, pages 1340–1346, 2009. [cited at p. iii, 36]

[38] Jason HandUber. Systolic arrays, 2003. Online, accessed 2nd April 2013. [cited at p. 282]

[39] Shousheng He and M. Torkelson. A new approach to pipeline fft processor. In *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, pages 766–770, 1996. [cited at p. xviii, 277, 278]

[40] Shousheng He and M. Torkelson. Designing pipeline fft processor for ofdm (de)modulation. In *Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on*, pages 257–262, 1998. [cited at p. 277]

[41] Richard Herveille. Cordic core, 2009. Online, accessed 20th April 2013. [cited at p. 218]

[42] Benjamin Heyne and Jürgen Götze. A pure cordic based fft for reconfigurable digital signal processing. In *12th European Signal Processing Conference (Eusipco2004)*, volume 7, 2004. [cited at p. 281, 283]

[43] H. Ho, V. Szwarc, and T. Kwasniewski. Hardware optimization for a reconfigurable polyphase-fft design using common sub-expression elimination. In *Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on*, pages 650–653, 2007. [cited at p. 283]

[44] Y.H. Hu. Cordic-based vlsi architectures for digital signal processing. *Signal Processing Magazine, IEEE*, 9(3):16–35, 1992. [cited at p. 269]

[45] M.M. Islam, D.R. Allee, S. Konasani, and A.A. Rodriguez. A low-cost digital controller for a switching dc converter with improved voltage regulation. *Power Electronics Letters, IEEE*, 2(4):121–124, 2004. [cited at p. iii]

[46] Gary Kacmarcik. Perfect shuffles. Online, accessed 6th April 2013. [cited at p. 162]

[47] A. Kelly and K. Rinne. Control of dc-dc converters by direct pole placement and adaptive feed-forward gain adjustment. In *Applied Power Electronics Conference and Exposition, 2005. APEC 2005. Twentieth Annual IEEE*, volume 3, pages 1970–1975 Vol. 3, 2005. [cited at p. iv, v, 78, 85]

[48] Stewart Kenly and W Latham II Paul. Methods and systems for power supply adaptive control utilizing transfer function measurements, 2013. US Patent 8,344,716. [cited at p. iv, 79, 82]

[49] Volkan Kursun, Siva G Narendra, Vivek K De, and Eby G Friedman. Analysis of buck converters for on-chip integration with a dual supply voltage microprocessor. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(3):514–522, 2003. [cited at p. 8]

[50] Tze Leung Lai and Ching Zong Wei. Least squares estimates in stochastic regression models with applications to identification and control of dynamic systems. *The Annals of Statistics*, pages 154–166, 1982. [cited at p. iv, v, 85]

[51] Pui-Kei Leong, Chun-Hung Yang, Chi-Wai Leng, and Chien-Hung Tsai. Design and implementation of sigma-delta dpwm controller for switching converter. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 3074–3077, 2009. [cited at p. iii, 36]

[52] Jian Li, Feng Liu, Teng Long, and Erke Mao. Research on pipeline r2^{2}sdf fft. In *Radar Conference, 2009 IET International*, pages 1–5, 2009. [cited at p. 280]

[53] Yamin Li and Wanming Chu. A new non-restoring square root algorithm and its vlsi implementations. In *Computer Design: VLSI in Computers and Processors, 1996. ICCD '96. Proceedings., 1996 IEEE International Conference on*, pages 538–544, 1996. [cited at p. 200]

[54] Yan-Fei Liu, E. Meyer, and Xiaodong Liu. Recent developments in digital control strategies for dc/dc switching power converters. *Power Electronics, IEEE Transactions on*, 24(11):2567–2577, 2009. [cited at p. iii]

[55] Yan-Fei Liu and P.C. Sen. Digital control of switching power converters. In *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*, pages 635–640, 2005. [cited at p. iii]

[56] Yan-Fei Liu and P.C. Sen. Digital control of switching power converters. In *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*, pages 635–640, 2005. [cited at p. iii]

[57] Lennart Ljung. *System identification*. Springer, 1998. [cited at p. iv, v, 77, 78, 85]

[58] Lennart Ljung and Keith Glover. Frequency domain versus time domain methods in system identification. *Automatica*, 17(1):71 – 86, 1981. [cited at p. iv, v, 77, 78]

[59] Z. Lukic, S.S. Ahsanuzzaman, A. ProdicÌĄ, and Zhenyu Zhao. Self-tuning sensorless digital current-mode controller with accurate current sharing for multi-phase dc-dc converters. In *Applied Power Electronics Conference and Exposition, 2009. APEC 2009. Twenty-Fourth Annual IEEE*, pages 264–268, 2009. [cited at p. iv, v, 78]

[60] Z. Lukic, Zhenyu Zhao, S.S. Ahsanuzzaman, and A. ProdicÌĄ. Self-tuning digital current estimator for low-power switching converters. In *Applied Power Electronics Conference and Exposition, 2008. APEC 2008. Twenty-Third Annual IEEE*, pages 529–534, 2008. [cited at p. iv, v, 78]

[61] Zdravko Lukic, Nabeel Rahman, and Aleksandar Prodic. Multibit σ–âĹĘ pwm digital controller ic for dc–dc converters operating at switching frequencies beyond 10 mhz. *Power Electronics, IEEE Transactions on*, 22(5):1693–1707, 2007. [cited at p. iii, 36]

[62] F Jessie MacWilliams and Neil JA Sloane. Pseudo-random sequences and arrays. *Proceedings of the IEEE*, 64(12):1715–1729, 1976. [cited at p. 84]

[63] D. Maksimovic and R. Zane. Small-signal discrete-time modeling of digitally controlled dc-dc converters. In *Computers in Power Electronics, 2006. COMPEL '06. IEEE Workshops on*, pages 231–235, 2006. [cited at p. iii, 25]

[64] D. Maksimovic, R. Zane, and R. Erickson. Impact of digital control in power electronics. In *Power Semiconductor Devices and ICs, 2004. Proceedings. ISPSD '04. The 16th International Symposium on*, pages 13–22, 2004. [cited at p. iii]

[65] M.W. May, M.R. May, and J.E. Willis. A synchronous dual-output switching dc-dc converter using multibit noise-shaped switch control. In *Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International*, pages 358–359, 2001. [cited at p. iii, 36]

[66] Botao Miao, R. Zane, and D. Maksimovic. A modified cross-correlation method for system identification of power converters with digital control. In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, volume 5, pages 3728–3733 Vol.5, 2004. [cited at p. iv, v, 78, 79]

[67] Botao Miao, R. Zane, and D. Maksimovic. System identification of power converters with digital control through cross-correlation methods. *Power Electronics, IEEE Transactions on*, 20(5):1093–1099, 2005. [cited at p. iv, v, 78, 79]

[68] Rais Miftakhutdinov and Joseph Zbib. Synchronous buck converter with increased efficiency. In *Applied Power Electronics Conference, APEC 2007-Twenty Second Annual IEEE*, pages 714–718. IEEE, 2007. [cited at p. 8]

[69] Rais Miftakhutdinov and Joseph Zbib. Synchronous buck converter with increased efficiency. In *Applied Power Electronics Conference, APEC 2007-Twenty Second Annual IEEE*, pages 714–718. IEEE, 2007. [cited at p. 8]

[70] P. A. Milder. Dft/fft ip core generator. Online, accessed 3rd April 2012. [cited at p. xviii, 280]

[71] Peter A. Milder, Franz Franchetti, James C. Hoe, and Markus Püschel. Formal datapath representation and manipulation for implementing DSP transforms. In *Design Automation Conference (DAC)*, pages 385–390, 2008. [cited at p. 279, 283]

[72] J. Morroni, L. Corradini, R. Zane, and D. Maksimovic. Adaptive tuning of switched-mode power supplies operating in discontinuous and continuous conduction modes. *Power Electronics, IEEE Transactions on*, 24(11):2603–2611, 2009. [cited at p. iv, v, 78]

[73] J. Morroni, R. Zane, and D. Maksimovic. Design and implementation of an adaptive tuning system based on desired phase margin for digitally controlled dcdc converters. *Power Electronics, IEEE Transactions on*, 24(2):559–564, 2009. [cited at p. iv, v, 78]

[74] J. Morroni, R. Zane, and D. Maksimovic. An online stability margin monitor for digitally controlled switched-mode power supplies. *Power Electronics, IEEE Transactions on*, 24(11):2639–2648, 2009. [cited at p. iv, 81]

[75] Grace Nordin, Peter A. Milder, James C. Hoe, and Markus Püschel. Automatic generation of customized discrete Fourier transform IPs. In *Design Automation Conference (DAC)*, pages 471–474, 2005. [cited at p. xviii, 166, 230, 277, 278, 279]

[76] M. Norris, L.M. Platon, E. Alarcon, and D. Maksimovic. Quantization noise shaping in digital pwm converters. In *Power Electronics Specialists Conference, 2008. PESC 2008. IEEE*, pages 127–133, 2008. [cited at p. iii, 36, 37, 38, 97, 99]

[77] 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, pages 53–59 Vol.1, 2004. [cited at p. iii]

[78] A. V. Oppenheim and R. W. Schafer. *Discrete-time Signal Processing*. Prentice Hall, 1989. [cited at p. xviii, 272, 273, 274]

[79] J. O'Sullivan, S. Weiss, and G. Rice. Automatic fft code generation for fpgas with high flexibility and human readability. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, pages 2197–2201, 2011. [cited at p. xviii, xix, 227, 284, 286]

[80] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables and Stochastic Processes - Fourth Edition*. Mc-Graw Hill. [cited at p. 232, 272]

[81] Sang Yoon Park, Nam-Ik Cho, Sang-Uk Lee, Kichul Kim, and Jisung Oh. Design of 2k/4k/8k-point fft processor based on cordic algorithm in ofdm receiver. In *Communications, Computers and signal Processing, 2001. PACRIM. 2001 IEEE Pacific Rim Conference on*, volume 2, pages 457–460 vol.2, 2001. [cited at p. 217, 280, 283]

[82] B.J. Patella, A. Prodic, A. Zirger, and D. Maksimovic. High-frequency digital controller ic for dc/dc converters. In *Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventeenth Annual IEEE*, volume 1, pages 374–380 vol.1, 2002. [cited at p. iii, iv]

[83] Marshall C. Pease. An adaptation of the fast fourier transform for parallel processing. *J. ACM*, 15(2):252–264, April 1968. [cited at p. 161, 279]

[84] Hao Peng and D. Maksimovic. Digital current-mode controller for dc-dc converters. In *Applied Power Electronics Conference and Exposition, 2005. APEC 2005. Twentieth Annual IEEE*, volume 2, pages 899–905 Vol. 2, 2005. [cited at p. iii, iv]

[85] Hao Peng, A. Prodic, E. Alarcon, and D. Maksimovic. Modeling of quantization effects in digitally controlled dc ndash;dc converters. *Power Electronics, IEEE Transactions on*, 22(1):208–215, 2007. [cited at p. iii, 24, 32, 33]

[86] A.V. Peterchev and S.R. Sanders. Quantization resolution and limit cycling in digitally controlled pwm converters. *Power Electronics, IEEE Transactions on*, 18(1):301–308, 2003. [cited at p. iii, 24, 32, 33, 105]

[87] A.V. Peterchev, J. Xiao, and S.R. Sanders. Architecture and ic implementation of a digital vrm controller. *Power Electronics, IEEE Transactions on*, 18(1):356–364, 2003. [cited at p. iii, iv]

[88] A.A. Petrovsky and S.L. Shkredov. Automatic generation of split-radix 2-4 parallel-pipeline fft processors: Hardware reconfiguration and core optimizations. In *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, pages 181–186, 2006. [cited at p. 277]

[89] Andrea Picciau. Design and implementation of a novel scalable fft processor with hybrid cordic-lut twiddle factors generator. [cited at p. 159, 261, 262]

[90] G.E. Pitel and P.T. Krein. Real-time system identification for load monitoring and transient handling of dc-dc supplies. In *Power Electronics Specialists Conference, 2008. PESC 2008. IEEE*, pages 3807–3813, 2008. [cited at p. iv, v, 78, 85]

[91] Dinkar Prasad. Introduction to switched-mode power supply (smps) circuits. Online, accessed 14th April 2013. [cited at p. 195]

[92] A. Prodic and D. Maksimovic. Design of a digital pid regulator based on look-up tables for control of high-frequency dc-dc converters. In *Computers in Power Electronics, 2002. Proceedings. 2002 IEEE Workshop on*, pages 18–22, 2002. [cited at p. iv]

[93] Aleksandar Prodic, Dragan Maksimovic, and Robert W Erickson. Design and implementation of a digital pwm controller for a high-frequency switching dc-dc power converter. In *Industrial Electronics Society, 2001. IECON'01. The 27th Annual Conference of the IEEE*, volume 2, pages 893–898. IEEE, 2001. [cited at p. iii, 24, 26]

[94] M. Puschel, J. M F Moura, J.R. Johnson, D. Padua, M.M. Veloso, B.W. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W. Johnson, and N. Rizzolo. Spiral: Code generation for dsp transforms. *Proceedings of the IEEE*, 93(2):232–275, 2005. [cited at p. 278]

[95] S. Saggini, M. Ghioni, and A. Geraci. An innovative digital control architecture for low-voltage, high-current dc-dc converters with tight voltage regulation. *Power Electronics, IEEE Transactions on*, 19(1):210–218, 2004. [cited at p. iii]

[96] S. Saggini, P. Mattavelli, and M. Ghioni. High-performance mixed-signal voltage-mode control for dc-dc converters with inherent analog derivative action. In *Applied Power Electronics Conference, APEC 2007 - Twenty Second Annual IEEE*, pages 28–33, 2007. [cited at p. v]

[97] S. Saggini, W. Stefanutti, E. Tedeschi, and P. Mattavelli. Digital deadbeat control tuning for dc-dc converters using error correlation. *Power Electronics, IEEE Transactions on*, 22(4):1566–1570, 2007. [cited at p. iv, v, 78, 79]

[98] S. Saggini, D. Trevisan, P. Mattavelli, and M. Ghioni. Synchronous ndash;asynchronous digital voltage-mode control for dc ndash;dc converters. *Power Electronics, IEEE Transactions on*, 22(4):1261–1268, 2007. [cited at p. v]

[99] Biranchinath Sahu and Gabriel A Rincon-Mora. An accurate, low-voltage, cmos switching power supply with adaptive on-time pulse-frequency modulation (pfm) control. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 54(2):312–321, 2007. [cited at p. 7]

[100] T.E. Schmuland, M.B. Longbrake, P.E. Buxa, and M.M. Jamali. Automatic vhdl generation software tool for parameterized fpga based fft architectures. In *Aerospace and Electronics Conference (NAECON), Proceedings of the IEEE 2010 National*, pages 306–309, 2010. [cited at p. xvii, 227, 231, 280]

[101] M. Shirazi, R. Zane, and D. Maksimovic. An autotuning digital controller for dc-dc power converters based on online frequency-response measurement. *Power Electronics, IEEE Transactions on*, 24(11):2578–2588, 2009. [cited at p. iv, v, vi, 78, 79]

[102] M. Shirazi, R. Zane, D. Maksimovic, L. Corradini, and P. Mattavelli. Autotuning techniques for digitally-controlled point-of-load converters with wide range of capacitive loads. In *Applied Power Electronics Conference, APEC 2007 - Twenty Second Annual IEEE*, pages 14–20, 2007. [cited at p. iv, v, 34, 78, 79]

[103] Mariko Shirazi, Jeffrey Morroni, Arseny Dolgov, Regan Zane, and Dragan Maksimovic. Integration of frequency response measurement capabilities in digital controllers for dc–dc converters. *Power Electronics, IEEE Transactions on*, 23(5):2524–2535, 2008. [cited at p. iv, v, 78, 79, 85]

[104] W. Stefanutti, P. Mattavelli, S. Saggini, and M. Ghioni. Autotuning of digitally controlled buck converters based on relay feedback. In *Power Electronics Specialists Conference, 2005. PESC '05. IEEE 36th*, pages 2140–2145, 2005. [cited at p. iv, v, 34, 78, 79]

[105] W. Stefanutti, S. Saggini, L. Corradini, E. Tedeschi, P. Mattavelli, and D. Trevisan. Closed-loop model reference tuning of pid regulators for digitally controlled dc-dc converters based on duty-cycle perturbation. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 1553–1558, 2007. [cited at p. iv, v, 78]

[106] W. Stefanutti, S. Saggini, E. Tedeschi, P. Mattavelli, and P. Tenti. Simplified model reference tuning of pid regulators of digitally controlled dc-dc converters based on crossover frequency analysis. In *Power Electronics Specialists Conference, 2007. PESC 2007. IEEE*, pages 785–791, 2007. [cited at p. iv]

[107] A. Suleiman, A. Hussein, K. Bataineh, and D. Akopian. Scalable fft architecture vs. multiple pipeline fft architectures; hardware implementation and cost. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 3792–3796, Oct. [cited at p. v, 126, 161, 162, 164, 166, 169, 230, 277, 283]

[108] T. Y Sung. Memory-efficient and high-speed split-radix fft/ifft processor based on pipelined cordic rotations. *Vision, Image and Signal Processing, IEE Proceedings* -, 153(4):405–410, August. [cited at p. xviii, 217, 282, 283]

[109] A. Syed, E. Ahmed, and D. Maksimovic. Digital pwm controller with feed-forward compensation. In *Applied Power Electronics Conference and Exposition, 2004. APEC '04. Nineteenth Annual IEEE*, volume 1, pages 60–66 Vol.1, 2004. [cited at p. iii]

[110] A. Syed, E. Ahmed, D. Maksimovic, and E. Alarcon. Digital pulse width modulator architectures. In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, volume 6, pages 4689–4695 Vol.6, 2004. [cited at p. iii, 26]

[111] T. Takayama and D. Maksimovic. Digitally controlled 10 mhz monolithic buck converter. In *Computers in Power Electronics, 2006. COMPEL '06. IEEE Workshops on*, pages 154–158, 2006. [cited at p. iii, 25]

[112] O. Trescases, Z. Lukic, Wai-Tung Ng, and A. Prodic. A low-power mixed-signal current-mode dc-dc converter using a one-bit delta; sigma; dac. In *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE*, pages 5 pp.–, 2006. [cited at p. v]

[113] D. Trevisan, S. Saggini, P. Mattavelli, L. Corradini, and P. Tenti. Analysis of a mixed-signal control for dc-dc converters based on hysteresis modulation and estimated inductor current. In *Power Electronics and Drive Systems, 2007. PEDS '07. 7th International Conference on*, pages 391–397, 2007. [cited at p. v]

[114] J.A. Vite-Frias, Rd.J. Romero-Troncoso, and A. Ordaz-Moreno. Vhdl core for 1024-point radix-4 fft computation. In *Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on*, pages 4 pp.–24, 2005. [cited at p. xviii, 230, 282, 283]

[115] Jack E. Volder. The cordic trigonometric computing technique. *Electronic Computers, IRE Transactions on*, EC-8(3):330–334, Sept. 1959. [cited at p. v, 128, 181, 185, 213, 269]

[116] K. Wang, N. Rahman, Z. Lukic, and A. Prodic. All-digital dpwm/dpfm controller for low-power dc-dc converters. In *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE*, pages 5 pp.–, 2006. [cited at p. iii]

[117] G.W. Wester and R.D. Middlebrook. Low-frequency characterization of switched dc-dc converters. *Aerospace and Electronic Systems, IEEE Transactions on*, AES-9(3):376–385, May 1973. [cited at p. 11]

[118] A.M. Wu, J. Xiao, D. Markovic, and S.R. Sanders. Digital pwm control: application in voltage regulation modules. In *Power Electronics Specialists Conference, 1999. PESC 99. 30th Annual IEEE*, volume 1, pages 77–83 vol.1, 1999. [cited at p. iii]

[119] A.M. Wu, J. Xiao, D. Markovic, and S.R. Sanders. Digital pwm control: application in voltage regulation modules. In *Power Electronics Specialists Conference, 1999. PESC 99. 30th Annual IEEE*, volume 1, pages 77–83 vol.1, 1999. [cited at p. iii]

[120] J. Xiao, A.V. Peterchev, Jianhui Zhang, and S.R. Sanders. A 4-μa quiescent-current dual-mode digitally controlled buck converter ic for cellular phone applications. *Solid-State Circuits, IEEE Journal of*, 39(12):2342–2348, 2004. [cited at p. iii, iv]

[121] Xin Xiao, E. Oruklu, and J. Saniie. Reduced memory architecture for cordic-based fft. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2690–2693, 2010. [cited at p. xviii, 181, 217, 284, 285]

[122] Xin Xiao, E. Oruklu, and J. Saniie. Reduced memory architecture for cordic-based fft. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2690–2693, 30 2010-June 2. [cited at p. v, 128]

[123] Xilinx. *LogiCORE IP Fast Fourier Transform v.7.1*. [cited at p. xviii, 280, 285, 286, 287, 288]

[124] Xilinx. *Spartan-6 FPGA Block RAM Resources*. [cited at p. 229]

[125] XionLogic. Radix-2^2sdf fft core, 2012. Online, accessed 2nd April 2013. [cited at p. xix, 229, 230, 280]

[126] V. Yousefzadeh and S. Choudhury. Nonlinear digital pid controller for dc-dc converters. In *Applied Power Electronics Conference and Exposition, 2008. APEC 2008. Twenty-Third Annual IEEE*, pages 1704–1709, 2008. [cited at p. iv]

[127] V. Yousefzadeh, T. Takayama, and D. Maksimovic. Hybrid dpwm with digital delay-locked loop. In *Computers in Power Electronics, 2006. COMPEL '06. IEEE Workshops on*, pages 142–148, 2006. [cited at p. iii, 25, 26]

[128] V. Yousefzadeh, Narisi Wang, D. Maksimovic, and Zoya Popovic. Digitally controlled dc-dc converter for rf power amplifier. In *Applied Power Electronics Conference and Exposition, 2004. APEC' 04. Nineteenth Annual IEEE*, volume 1, pages 81–87 Vol.1, 2004. [cited at p. iii]

[129] Cheng-Ying Yu, Sau-Gee Chen, and J. C. Chih. Efficient cordic designs for multi-mode ofdm fft, 2006. [cited at p. xviii, 181, 185, 216, 222, 290]

[130] Cheng-Ying Yu, Sau-Gee Chen, and J.-C. Chih. Efficient cordic designs for multi-mode ofdm fft. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 3, pages III–III, May. [cited at p. v, 128]

[131] E.L. Zapata and F. Arguello. A vlsi constant geometry architecture for the fast hartley and fourier transforms. *Parallel and Distributed Systems, IEEE Transactions on*, 3(1):58–70, 1992. [cited at p. 163]

[132] Guoping Zhang and F. Chen. Parallel fft with cordic for ultra wide band. In *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*, volume 2, pages 1173–1177 Vol.2, 2004. [cited at p. xviii, 166, 281, 283, 290]

[133] Jianhui Zhang and S.R. Sanders. A digital multi-mode multi-phase ic controller for voltage regulator application. In *Applied Power Electronics Conference, APEC 2007 - Twenty Second Annual IEEE*, pages 719–726, 2007. [cited at p. iii, iv]

[134] Yang Zhang, Xu Zhang, R. Zane, and D. Maksimovic. Wide-bandwidth digital multi-phase controller. In *Power Electronics Specialists Conference, 2006. PESC '06. 37th IEEE*, pages 1–7, 2006. [cited at p. iii, 27, 42]

[135] Zhenyu Zhao, Huawei Li, A. Feizmohammadi, and A. Prodic. Limit-cycle based auto-tuning system for digitally controlled low-power smps. In *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE*, pages 1143–1147, 2006. [cited at p. iv, v, 34, 78, 79, 81]

[136] Zhenyu Zhao and A. Prodic. Limit-cycle oscillations based auto-tuning system for digitally controlled dc-dc power supplies. *Power Electronics, IEEE Transactions on*, 22(6):2211–2222, 2007. [cited at p. iv, v, 34, 78, 79]

[137] Zhenyu Zhao and A. Prodic. Continuous-time digital controller for high-frequency dc-dc converters. *Power Electronics, IEEE Transactions on*, 23(2):564–573, 2008. [cited at p. iv]

[138] Zhenyu Zhao and A. ProdicÌĄ. Non-zero error method for improving output voltage regulation of low-resolution digital controllers for smps. In *Applied Power Electronics Conference and Exposition, 2008. APEC 2008. Twenty-Third Annual IEEE*, pages 1106–1110, 2008. [cited at p. iii, 36]

[139] Zhenyu Zhao, A. Prodic, and P. Mattavelli. Self-programmable pid compensator for digitally controlled smps. In *Computers in Power Electronics, 2006. COMPEL '06. IEEE Workshops on*, pages 112–116, 2006. [cited at p. iv, v, 34, 78, 79]

[140] Zhenyu Zhao, V. Smolyakov, and A. Prodic. Continuous-time digital signal processing based controller for high-frequency dc-dc converters. In *Applied Power Electronics Conference, APEC 2007 - Twenty Second Annual IEEE*, pages 882–886, 2007. [cited at p. iv]

[141] Bin Zhou, Yingning Peng, and David Hwang. Pipeline fft architectures optimized for fpgas. *International Journal of Reconfigurable Computing*, 2009:9, 2009. [cited at p. 280]

List of Publications Related to the Thesis

Patents

- A. Congiu, E. Bodano, D. Hammerschmidt, *Converter Circuit and Method for Converting an Input Voltage to an Output Voltage.* United States Application Number or PCT International Appln. No.61/815,787, April 2013. (Relation to Chapter 7)

Published papers

Conference papers

- A. Congiu, M. Barbaro, E. Bodano, D. Hammerschmidt, *Low-Perturbation Load Identification Techniques for Digitally Controlled DC-DC Power Supplies,* in 45th *annual meeting of the Associazione Gruppo Italiano di Elettronica,* pp.123–124, Udine, Italy, June 2013. (Relation to Chapter 7)

- A. Congiu, M. Barbaro, A. Picciau, E. Bodano, *Scalable Hybrid CORDIC-LUT Architectures for CG-FFT Processors,* in *IEEE Proc. of* 9th *Conf. on Ph.D. Research in Microelectronics and Electronics,* pp.105–108, Villach, Austria, June 2013. (Relation to Chapter 9)

- A. Congiu, M. Barbaro, A. Picciau, E. Bodano, D. Hammerschmidt, *Prototype of a novel Steady-State Load Identification Technique for Digitally Controlled DC-DC Power Supplies,* in *IEEE Proc. of DASIP 2013 Conf. on Design and Architectures for Signal and Image Processing,* pp.355–356, Cagliari, Italy, October 2013. (Relation to Chapter 8)

Appendices

Appendix

Appendix A

Quick configuration guide

How to use this guide

This Appendix explains how to set-up the whole PSD computer, without deepening the details of each configuration or module. More details can be found in [89].

The process can be resumed in the following steps:

1. Configure the PSD computer
    a) Set the parameters of the *autocorrelation* system.
    b) Select the architecture of the RAM.
2. Configure the FFT processor
    a) Set the parameters of the *fft_processor* according to step 1.
    b) Select the architecture of the *twiddle factor generator*.

A.1 Configuring the autocorrelation system

A.1.1 Setting the parameters

Configuration of the global PSD computer in terms of parameters is easily performed by editing the file *autocorrelation_settings.vhd*. Each entry in this file is briefly described in the same configuration file, and default values are given.

A.2 Selecting the architecture of the RAM

Selection of the RAM architecture is performed by editing file *autocorrelation_ram_structural*. Unser the declaration of the *ram_core* component, a configuration declaration allows to choose between

ram_core_fpga This architecture is necessary for FPGA implementation. The multi-port RAM is realised as a matrix of BRAM blocks.

ram_core_simple A simple multi-port RAM code, which is preferred for an ASIC implementation.

A.3 Configuring the FFT processor

A.3.1 Setting the parameters

The parameters of the processor can be set by editing *fft_global_settings.vhd* according to *fft_global_settings.vhd*, as shown in Tab. A.1. Other parameters, such as *global_tf_accuracy* costants can be set in order to achieve the proper tradeoff between accuracy and hardware resource saving.

Table A.1: Matching of *fft_global_settings.vhd* and *autocorrelation_settings.vhd*.

FFT processor	Autocorrelation
global_points_c	sequence_size_c*2
global_pes_c	parallel_words_c/2
global_accuracy_int_c	data_int_c
global_accuracy_frac_c	data_frac_c

A.3.2 Selecting the twiddle factor generation architecture

Twiddle factors can be generated in two different ways. This can be selected by editing a line in the declaration section of architecture *twiddle_factors_system_structural*. Two options are available:

tfc_lut Conventional approach, a LUT is used to retrieve twiddle factors.

tfc_generator A novel system based on the CG-FFT twiddle factor scheme taking advantage of the scalability of the processor. This is more hardware efficient as the number of points to process increases. Two different sub-architectures can be selected by editing a line in the declaration section of file *tfc_generator.vhd*:

- *tfc_cordic_shared_core*
- *tfc_cordic_pipelined*

The settings of these two sub-architectures can be edited by modifying the file *fft_cordic_settings.vhd*.

The proposed architectures are slightly different than the ones in [89] and [24], but their behaviour is similar. The only difference is in the fact that the *step-computer* module here is pipelined, follows the *mapper*, but preceedes all the other blocks. A functional diagram of twiddle factor generators is shown in Fig. A.1

Once the twiddle factor generation mechanism is selected (either if it is the LUT or one of the hybrid CORDIC systems), the architecture of the internal ROM must be chosen. This is achieved by editing a file and configuring a component in its declaration section according to Tab. A.2. Architectures of the *fpga* type implement a multi-port ROM by using more BRAM blocks, while the *behavioral* versions are more suitable to an ASIC implementation.

A.3.3 Initialising the ROM

The internal ROM must be initialised according to accuracy and transform size settings. This is performed by running a MATLAB script which can be found in the *sw* folder. The name of the script is:

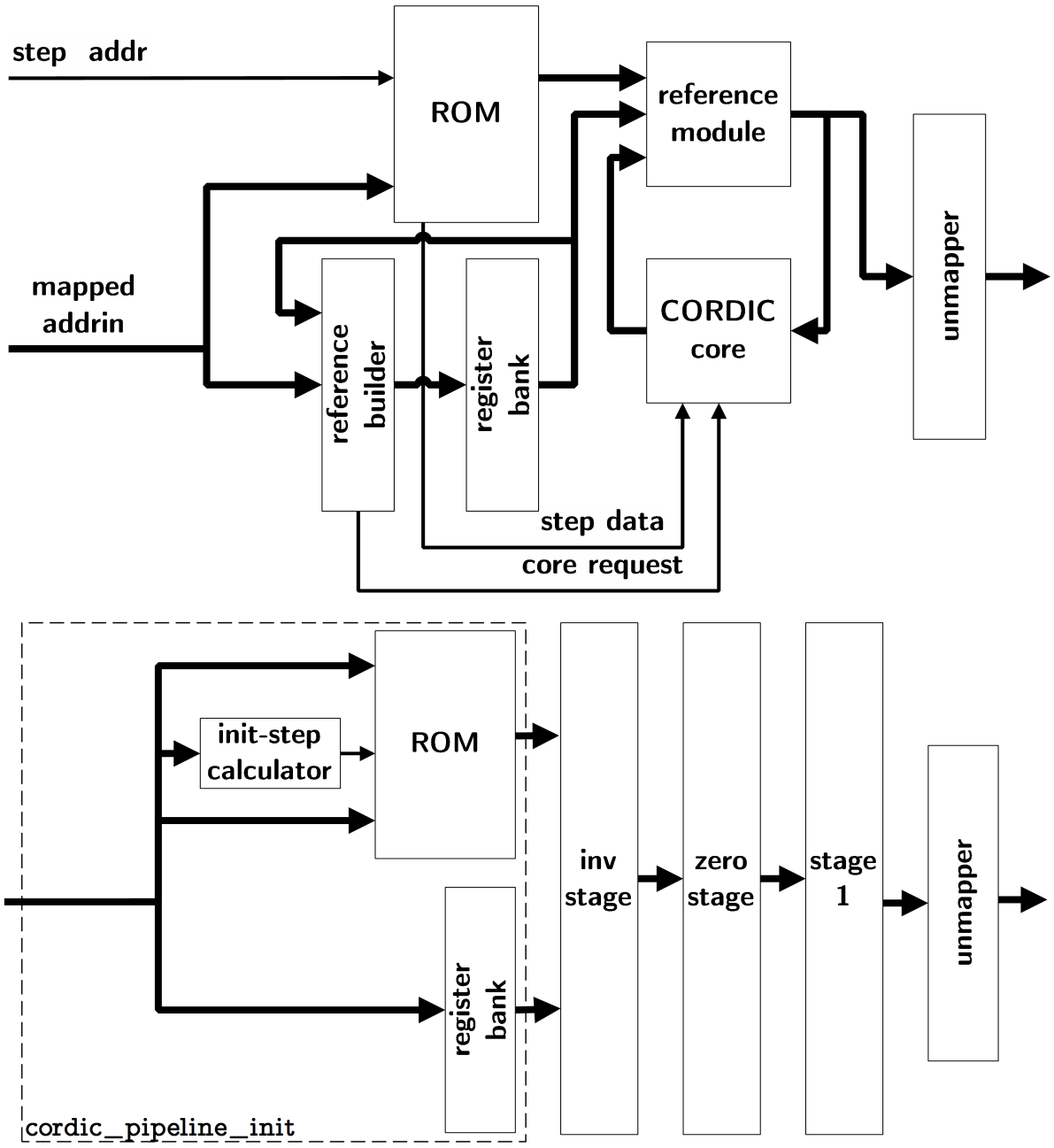

Figure A.1: Functional diagrams of twiddle factor generators.

Table A.2: Selection of the architecture for the internal ROM.

Architecture	File to edit	Component to configure	Possible architectures
LUT	tfc_lut_datapath_structural.vhd	lut_rom	*lut_rom_fpga* or *lut_rom_behavioral*
shared-core	tfc_cordic_shared_core_datapath_structural	cordic_rom	*cordic_rom_fpga* or *cordic_rom_behavioral*
pipelined	cordic_pipeline_init_datapath_structural	cordic_rom	*cordic_rom_fpga* or *cordic_rom_behavioral*

lut_rom_content_synth_gen if *tfc_lut* is selected.

cordic_rom_content_synth_gen if *tfc_generator* is selected.

In both cases the parameters of the script must be set according to the parameters of the processor. Each one of the scripts creates a file in the *rtl\vhdl* directory, called either *lut_rom_content_synth.txt* or *cordic_rom_content_synth.txt*. The content of the file must be copied in the declaration section of the specific ROM, defining costant *rom_c*. The files to copy data into has the same name of the chosen ROM architecture (see Tab. A.2).

Appendix B

Background mathematical concepts

B.1 The Fast Fourier Transform

In order to understand how the processor works and draw further mathematical considerations, it is necessary to derive the expression of the Discrete Fourier Transform (DFT). An interesting technique uses function interpolation theory [21].
Let us consider $f(\theta)$ a periodic function defined on $(0,2\pi)$ such that $f(\theta) = f(\theta + 2\pi)$. The goal of the mathematical derivation is to interpolate this function with the trigonometric polynomial

$$p(\theta) = a_0 + \sum_{k=1}^{n} a_k \cos k\theta + b_k \sin k\theta. \tag{B.1}$$

The coefficients of the polynomial are $N = 2n+1$, thus interpolation is done at N points θ_l that we assume to be equally spaced

$$\theta_l = l\frac{2\pi}{N} \qquad l = 0,\dots,N-1. \tag{B.2}$$

By using Euler identities

$$\cos\theta = \frac{e^{j\theta} + e^{-j\theta}}{2} \qquad \sin\theta = \frac{e^{j\theta} - e^{-j\theta}}{2j} \tag{B.3}$$

and substituting in expression B.1 we obtain

$$\begin{aligned} p(\theta) &= \left(\frac{a_n + jb_n}{2}\right)e^{-nj\theta} + \cdots + \left(\frac{a_l + jb_l}{2}\right)e^{jl\theta} + \cdots + \left(\frac{a_n - jb_n}{2}\right)e^{nj\theta} \\ &= X_{-n}e^{-jn\theta} + \dots X_l e^{jl\theta} + \cdots + X_n e^{jn\theta}. \end{aligned} \tag{B.4}$$

Using B.2 note that

$$e^{j\theta_l} = e^{jl\theta_1} = W_N^l \qquad l = 0,\dots,N-1.$$

Also, the following properties are valid

$$W_N^{l\pm N} = W_N^l \tag{B.5a}$$
$$W_N^{-l\pm N} = W_N^{-l} \tag{B.5b}$$
$$W_N^N = 1 \tag{B.5c}$$
$$(W_N^1)^{\frac{N}{2}} = -1 \tag{B.5d}$$
$$W_{\frac{N}{2}}^1 = W_N^2 \tag{B.5e}$$

because all the W_N^l coefficients are complex values equispaced in the unitary module circle of the Gauss plane. By exploiting these properties and letting $x_l = p(\theta_l)$ one can obtain the following from Eq. B.4:

$$x_l = X_0 + X_1 W_N^l + \cdots + X_l W_N^{l^2} + \cdots + X_{2n} W_N^{2nl}.$$

Considering $l = 0, \ldots, 2n$ this leads to the system of equations

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & W_N^1 & W_N^2 & W_N^3 & \ldots & W_N^{2n} \\ 1 & W_N^2 & W_N^4 & W_N^6 & \ldots & W_N^{4n} \\ 1 & W_N^3 & W_N^6 & W_N^9 & \ldots & W_N^{6n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{2n} & W_N^{4n} & W_N^{6n} & \ldots & W_N^{4n^2} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_l \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_l \\ \vdots \\ x_{N-1} \end{bmatrix},$$

that can be written as the matrix equation

$$M\boldsymbol{X} = \boldsymbol{x}. \tag{B.6}$$

Introducing $\overline{M}$ as the matrix in which every element is the complex conjugate of the matching element in M, and observing that $\overline{W_N^l} = W_N^{-l}$, it is possible to compute

$$\overline{M}M = \begin{bmatrix} N & 0 & 0 & 0 & \ldots & 0 \\ 0 & N & 0 & 0 & \ldots & 0 \\ 0 & 0 & N & 0 & \ldots & 0 \\ 0 & 0 & 0 & N & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & N \end{bmatrix}.$$

This leads to $M^{-1} = \frac{1}{N}\overline{M}$. Inverting B.6 we obtain

$$\begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_l \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & W_N^{-3} & \ldots & W_N^{-2n} \\ 1 & W_N^{-2} & W_N^{-4} & W_N^{-6} & \ldots & W_N^{-4n} \\ 1 & W_N^{-3} & W_N^{-6} & W_N^{-9} & \ldots & W_N^{-6n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{-2n} & W_N^{-4n} & W_N^{-6n} & \ldots & W_N^{-4n^2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_l \\ \vdots \\ x_{N-1} \end{bmatrix}. \tag{B.7}$$

Eq. B.7 leads to the following formula

$$X_r = \frac{1}{N} \sum_{l=0}^{N-1} x_l W_N^{-rl} \qquad r = 0, \ldots, N-1. \tag{B.8}$$

that is also called the Discrete Fourier Transform (DFT). Its inverse counterpart (IDFT) can be expressed as

$$x_l = \sum_{r=0}^{N-1} X_r W_N^{rl} \qquad l = 0, \ldots, N-1. \tag{B.9}$$

It is worth noticing that during the whole demonstration N is supposed to be odd. DFTs with $N = 2^s \quad s \in \mathbb{N}, s \geq 0$ are the most efficient to compute. One solution to this issue is to choose the following interpolator trigonometric polynomial

$$p(\theta) = \frac{a_0}{2} + \frac{a_{n+1}}{2} \cos(n+1)\theta + \sum_{k=1}^{n} a_k \cos k\theta + b_k \sin k\theta,$$

which allows to consider $N = 2n+2$ and does not affect B.8 nor B.9.

The computational complexity of B.8 is $O(N^2)$, such as a multiplication of a matrix by a vector. To improve this figure of merit, the basic idea is to exploit the *divide and conquer* paradigm: the problem is divided into two or more problems of a smaller size that are solved recursively or iteratively. When these sub-problems are small enough, the recursion terminates, and the solution of the original problem is obtained as a combination of theirs. This approach is used to obtain the Fast Fourier Transform (FFT).

An important parameter discriminating classes of FFT algorithms is the *radix* [21]. An algorithm is said to be *radix-2* when a problem is divided into two sub-problems of half the size. This type of algorithm is probably the most used in practical applications, but it is possible to find *radix-4*, *radix-8*, *mixed-radix* and *split-radix* FFTs, which combine different radix types to achieve improved computational complexity. Each division is called a *step* of the algorithm, consequently if N is the length of the array that must be processed and R is the radix of the algorithm, $\log_R N$ steps are necessary to perform the whole computation. Once the radix is defined, another distinction is based on the *decimation* type [21], that can be *Decimation-In-Time* (DIT) or *Decimation-In-Frequency* (DIF). Decimation refers to how sub-problems are defined during the execution of the algorithm.

Our implementation is based on radix-2 DIF. It is obtained by splitting X_r into two separate sets: the even-indexed set $\{X_{2k} \mid k = 0,\dots,\frac{N}{2}-1\}$ and the odd-indexed set $\{X_{2k+1} \mid k = 0,\dots,\frac{N}{2}-1\}$. To derive this basic FFT algorithm firstly it is necessary to define, with a slight abuse of notation,

$$W_N^i = e^{-ji\theta_1} = e^{-j\frac{2\pi}{N}i} \qquad i \in \mathbb{N} \tag{B.10}$$

as the *twiddle factor* (TF) *of index* i. Being properties B.5 valid for twiddle factors, they are intensely exploited throughout the whole derivation of every FFT algorithm. With B.10 Eq. B.8 becomes

$$X_r = \frac{1}{N}\sum_{l=0}^{N-1} x_l W_N^{rl} \qquad r = 0,\dots,N-1. \tag{B.11}$$

Splitting X_r into the two sets we obtain for $r = 0,\dots,N-1$

$$\begin{aligned} X_r &= \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1} x_l W_N^{rl} + \frac{1}{N}\sum_{l=\frac{N}{2}}^{N-1} x_l W_N^{rl} \\ &= \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1} x_l W_N^{rl} + \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1} x_{l+\frac{N}{2}} W_N^{r(l+\frac{N}{2})} \\ &= \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1} \left(x_l + x_{l+\frac{N}{2}} W_N^{r\frac{N}{2}}\right) W_N^{rl}. \end{aligned} \tag{B.12}$$

For r even, using B.5d and B.5e in B.12 yelds to

$$\begin{aligned} X_{2k} &= \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1} \left(x_l + x_{l+\frac{N}{2}} W_N^{kN}\right) W_N^{2kl} \\ &= \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1} \left(x_l + x_{l+\frac{N}{2}}\right) W_{\frac{N}{2}}^{kl} \end{aligned} \qquad k = 0,\dots,\frac{N}{2}-1.$$

By defining $Y_k = X_{2k}$ and $y_l = x_l + x_{l+\frac{N}{2}}$ one half-size sub-problem is written as:

$$Y_k = \sum_{l=0}^{\frac{N}{2}-1} y_l W_{\frac{N}{2}}^{kl} \qquad k = 0,1,\dots,\frac{N}{2}-1. \tag{B.13}$$

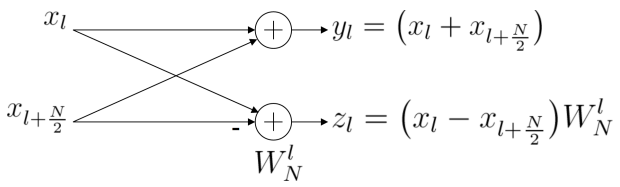

Figure B.1: The Gentleman-Sande butterfly.

For r odd, using B.5c and B.5e in B.12 results in

$$\begin{aligned} X_{2k+1} &= \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1}\left(x_l + x_{l+\frac{N}{2}} W_N^{(2k+1)\frac{N}{2}}\right) W_N^{(2k+1)l} \\ &= \frac{1}{N}\sum_{l=0}^{\frac{N}{2}-1}\left(\left(x_l - x_{l+\frac{N}{2}}\right)W_N^l\right) W_{\frac{N}{2}}^{kl} \end{aligned} \qquad k = 0,\ldots,\frac{N}{2}-1.$$

Thus, with $Z_k = X_{2k+1}$ and $z_l = \left(x_l - x_{l+\frac{N}{2}}\right)W_N^l$ the second half-size sub-problem is written as:

$$Z_k = \sum_{l=0}^{\frac{N}{2}-1} z_l W_{\frac{N}{2}}^{kl} \qquad k = 0,1,\ldots,\frac{N}{2}-1. \tag{B.14}$$

After the two sub-problems have been solved, no further operation is needed to obtain final results. The division in sub-problems and computation of y_l and z_l as in B.13 and B.14 is called the Gentleman-Sande *butterfly* [21] and is graphically represented as in Fig. B.1.

It has been observed that computation of y_l and z_l in B.13 and B.14 requires N complex additions and $\frac{N}{2}$ complex multiplications. Each complex addition is equal to two real additions and each complex multiplication can be thought as equal to three real multiplications and three real additions. By counting each real addition or multiplication as one operation one computes the arithmetic cost of the FFT. It is possible to show that this is

$$T(N) = 5N\log_2 N = \mathrm{O}\left(N\log_2 N\right), \tag{B.15}$$

much less than the DFT basic formula which has a $\mathrm{O}(N^2)$ complexity [21].

A pseudocode version of the radix-2 DIF FFT algorithm is illustrated in Alg 5.

Every FFT algorithm is very often featured with a *flowgraph* or *dataflow diagram* that gives a visual depiction of how data is managed during processing. An example flowgraph of radix-2 DIF is shown in Fig. B.2. Decimation in frequency is visible, and in the last step butterflies are computed by using adjacent values. By contrast, a dataflow diagram of the DIT conterpart of radix-2 is illustrated in Fig. B.3. This different algorithm is derived by splitting x_l (and not X_r) into two separate sets, thus obtaining a different butterfly structure called *Cooley-Tukey butterfly* [21].

Another characterization of FFT algorithms, more related to their implementation, can be done by discriminating the order of input and output values. In classic FFTs one of the two is *bit-reversed*, or rather the actual sequence of values can be obtained by reading the address of each of them in reversed order. Permutation of input or output values is thus a computational step that must be taken into account when implementing FFT hardware. A common practice is to discern types of algorithms by using two subscript characters, the first for input and the second for output ordering. N is used when no permutation is needed while R indicates that bit-reversal must be performed. The FFT algorithm implemented in the hardware design for this thesis is based on *radix-2 DIF*$_{NR}$.

Algorithm 5 The iterative radix-2 DIF FFT algorithm.

Require: $\boldsymbol{x}, \boldsymbol{W}$
  $N \leftarrow \text{length}(\boldsymbol{x})$
  $NumPr \leftarrow 1$
  $PrSize \leftarrow N$
  while $PrSize > 1$ **do**
    $HalfSize \leftarrow PrSize/2$
    for $k = 0$ **to** $NumPr - 1$ **do**
      $j_{first} \leftarrow kPrSize$
      $j_{last} \leftarrow j_{first} + HalfSize - 1$
      $l \leftarrow 0$
      for $j = j_{first}$ **to** j_{last} **do**
        $W \leftarrow W_l$
        $q \leftarrow x_j$
        $x_j \leftarrow q + x_{j+HalfSize}$
        $x_{j+HalfSize} \leftarrow W(q - x_{j+HalfSize})$
        $l \leftarrow l + NumPr$
      end for
    end for
    $NumPr \leftarrow 2NumPr$
    $PrSize \leftarrow HalfSize$
  end while
Ensure: $\boldsymbol{X}$

B.2 The CORDIC algorithm

The CORDIC (COordinate Rotation Digital Computer) was introduced in 1959 by Volder [115]. The iterative algorithm has two different working modes. *Rotation mode* computes sine and cosine of an angle $\beta \in (-\frac{\pi}{2}, \frac{\pi}{2})$, which are represented in a fixed-point format. The *vectoring mode* performs the inverse operation. For the purposes of this thesis, rotation mode is the most significant version of CORDIC, consequently it will be analysed in its general formulation [44]. Iterations are initialised with vector

$$\boldsymbol{v}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Every iteration can be represented by the rotation matrix

$$R_i = \begin{bmatrix} \cos\gamma_i & -\sin\gamma_i \\ \sin\gamma_i & \cos\gamma_i \end{bmatrix} \tag{B.16}$$

and written as a matrix equation:

$$\boldsymbol{v}_i = R_i \boldsymbol{v}_{i-1} \tag{B.17}$$

where $\boldsymbol{v}_i$ is the vector obtained at iteration i and $\boldsymbol{v}_{i-1}$ is the vector obtained from the previous iteration. By subsituting the trigonometric identities

$$\cos\alpha = \frac{1}{\sqrt{1+\tan^2\alpha}} \qquad \sin\alpha = \frac{\tan\alpha}{\sqrt{1+\tan^2\alpha}}$$

in Eq. B.17 we find

$$\boldsymbol{v}_i = \frac{1}{\sqrt{1+\tan^2\gamma_i}} \begin{bmatrix} 1 & -\tan\gamma_i \\ \tan\gamma_i & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}. \tag{B.18}$$

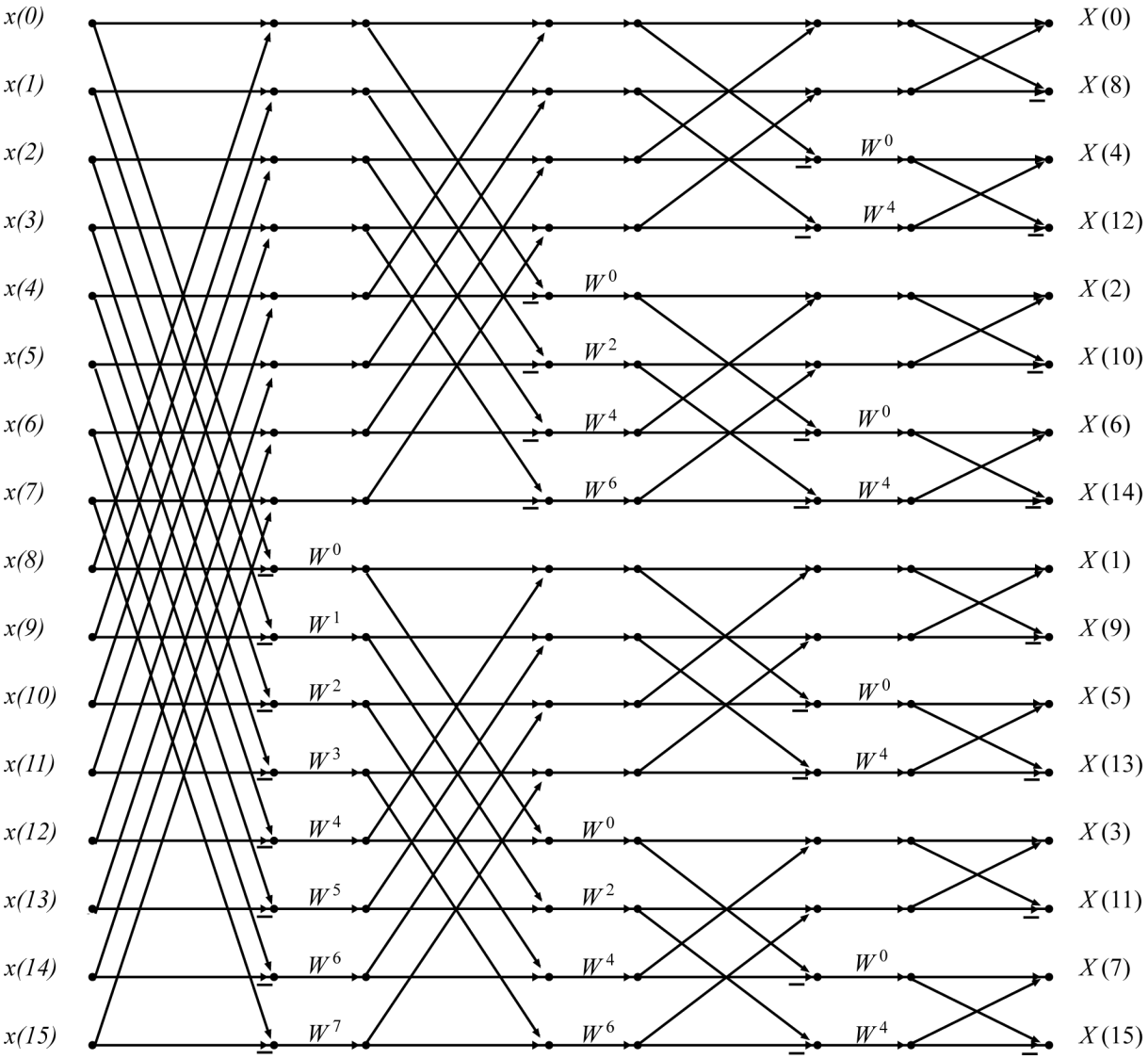

Figure B.2: Length-16, Decimation-in-Frequency, In-order input, Radix-2 FFT. From [1].

It is possible to choose rotation angles γ_i such that $\tan\gamma_i$ can be written as $\pm 2^{-i}$, thus obtaining the famous CORDIC relationship

$$\boldsymbol{v}_i = K_i \begin{bmatrix} 1 & -\sigma_i 2^- i \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}. \tag{B.19}$$

in whom $K_i = \frac{1}{\sqrt{1+2^{-2i}}}$. The direction of rotation at every step, thus the value of σ_i in B.19 is determined considering the remainder angle

$$\beta_i = \beta_{i-1} - \sigma_i \gamma_i$$

in the following expression

$$\sigma_i = \begin{cases} +1 & \text{if } \beta_{i-1} \geq 0, \\ -1 & \text{if } \beta_{i-1} < 0. \end{cases} \tag{B.20}$$

If n is the number of iterations, K_n is also called *CORDIC gain* or *scale factor*. This can be indifferently

- retrieved from a lookup-table,

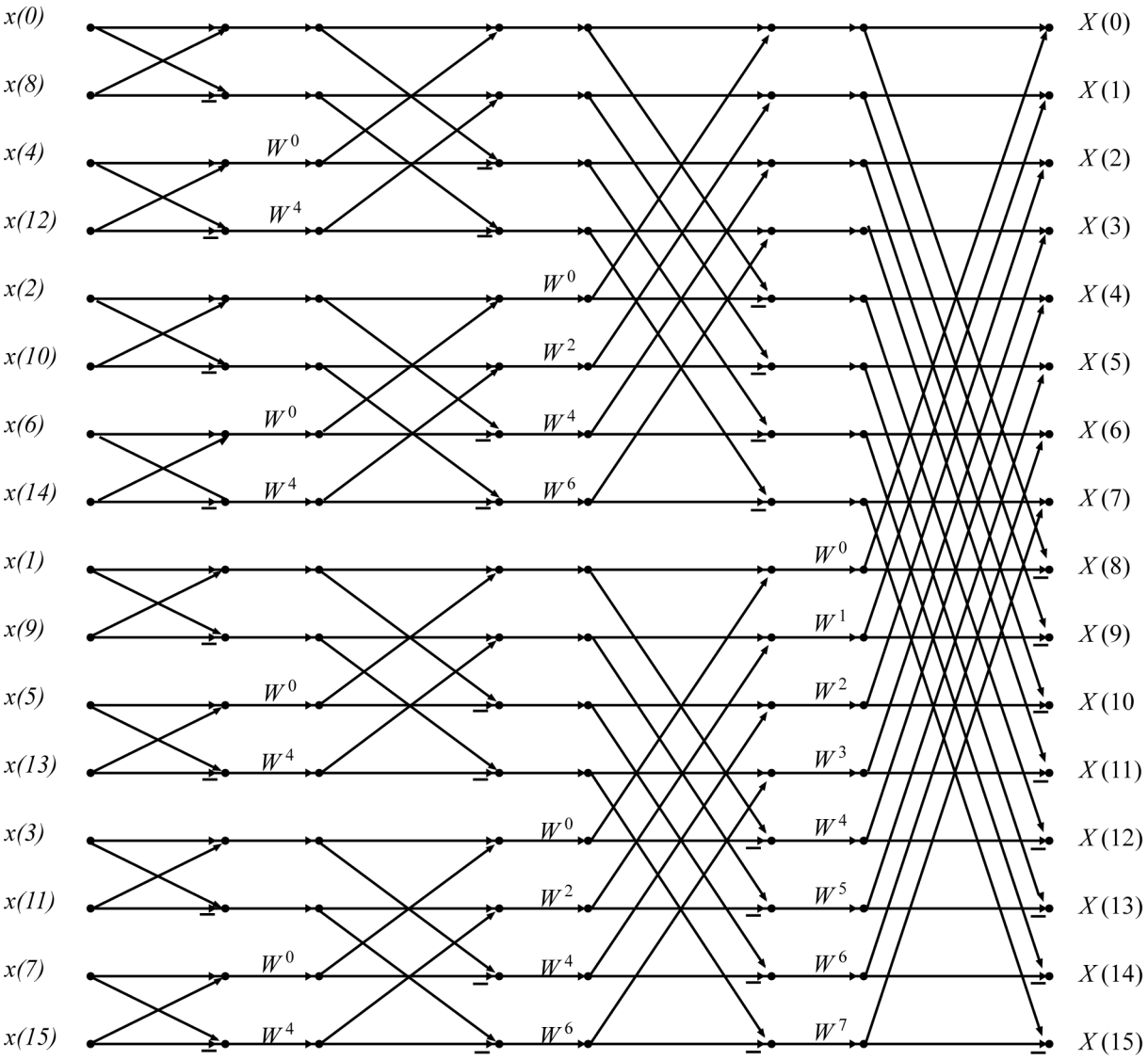

Figure B.3: Length-16, Decimation-in-Time, In-order output, Radix-2 FFT. From [1].

- ignored throughout the iterative process and compensated in the last iteration by multiplication,
- corrected in the first iteration by properly scaling $\boldsymbol{v}_0$.

An pseudocode example of CORDIC is shown in Alg. 6. It is worth noticing that the values of $\gamma_i = \tan^{-1} 2^{-i}$ and the scale factor K_n must be necessarily stored to perform iterations. The number of iterations n depends on the desired precision. Usually, it is assumed that n is equal to the number of bits B of output results.

Algorithm 6 The CORDIC algorithm

Require: $\beta, n, \boldsymbol{\gamma}, \boldsymbol{K}$
  $\boldsymbol{v}_0 \leftarrow [0;0]$
  $\beta_0 \leftarrow \beta$
  for $i = 0$ **to** $n-1$ **do**
    if $\beta_i \geq 0$ **then**
      $\sigma_i \leftarrow 1$
    else
      $\sigma_i \leftarrow -1$
    end if
    $q \leftarrow \sigma_i 2^{-i}$
    $R_i \leftarrow [1, -q; q, 1]$
    $\boldsymbol{v}_{i+1} \leftarrow R_i \boldsymbol{v}_i$
    $\beta_{i+1} \leftarrow \beta_i - \sigma_i \gamma_i$
  end for
  $\boldsymbol{v}_n \leftarrow \boldsymbol{v}_n K_n$
Ensure: $\boldsymbol{v}_n$

B.3 Theorical remarks on the autocorrelation function

The Wiener-Khinchin theorem states that, if $x(t)$ is a wide-sense stationary (WSS) process and $S(f)$ is its *power spectral density*. then its autocorrelation can be written as [80]

$$R(\tau) = \int_{-\infty}^{\infty} S(f) e^{j2\pi f \tau} \, df. \tag{B.21}$$

This means that the autocorrelation of a WSS process is the inverse Fourier Transform of its power spectral density. In the discrete case, an interesting derivation of the properties of the autocorrelation sequence is based on the forementioned theorem and the introduction of estimates [78]. If WSS signal $x(t)$ is sampled with sampling time T_s, sequence $x[n]$ is obtained as:

$$x[n] = x(nT_s).$$

Considering a finite record of $x[n]$ we have

$$v[n] = \begin{cases} x(nT_s), & \text{for } 0 \leq n \leq Q-1, \\ 0, & \text{otherwise.} \end{cases} \tag{B.22}$$

An estimate of the autocorrelation sequence is thus

$$\hat{\phi}_{xx}[m] = \frac{1}{Q} \sum_{n=0}^{Q-|m|-1} x[n] x[n+|m|], \tag{B.23}$$

for $|m| \leq M-1$. Observing that in B.23

$$\sum_{n=0}^{Q-|m|-1} x[n] x[n+|m|] = \sum_{n=0}^{Q-|m|-1} x[n] x[|m|-(-n)] = x[n] * x[-n],$$

we can say that $\hat{\phi}_{xx}$ is an *aperiodic discrete convolution*. An example of this operation is shown in Fig. B.4, with two different sequences $x[k]$ and $h[n-k]$.

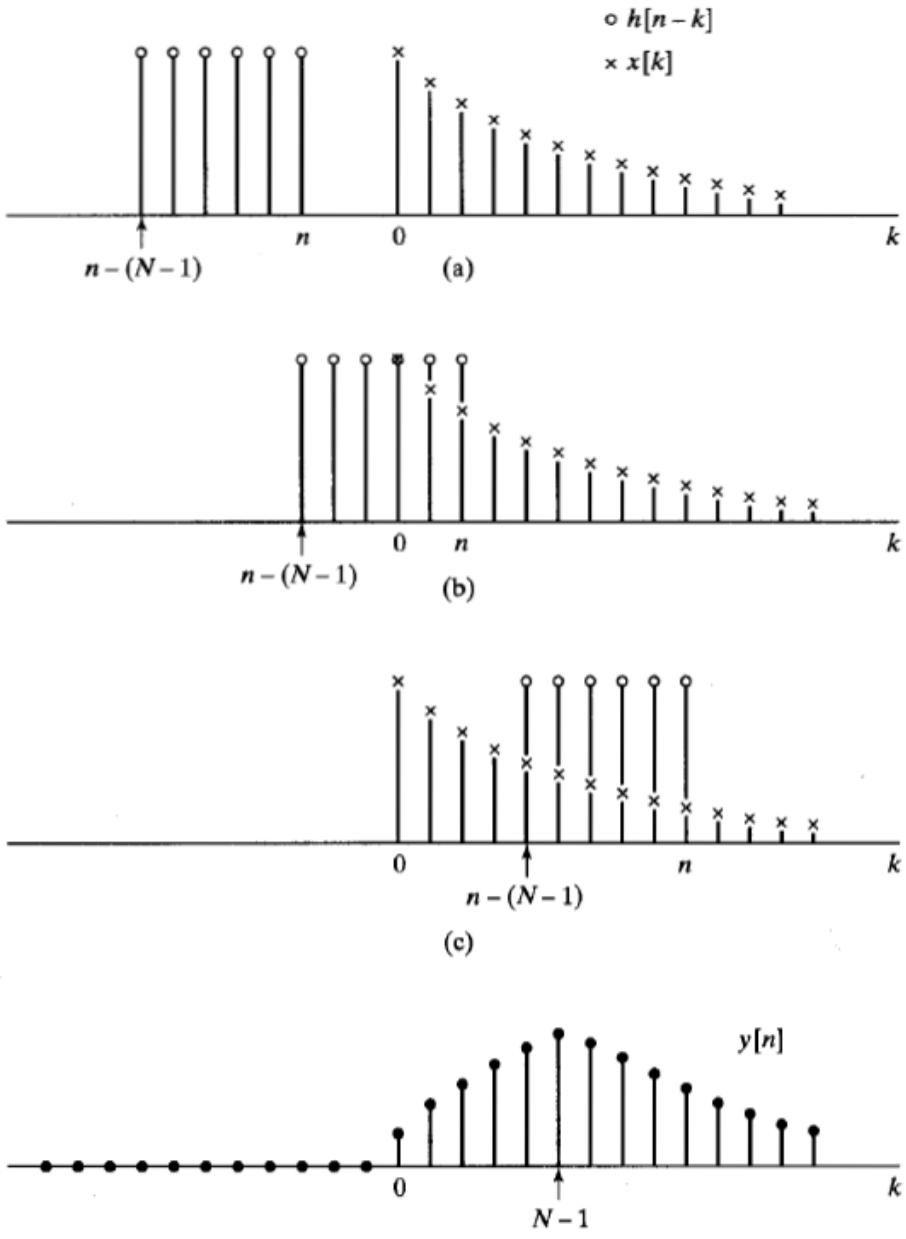

Figure B.4: Sequence involved in computing an aperiodic discrete convolution. From [78].

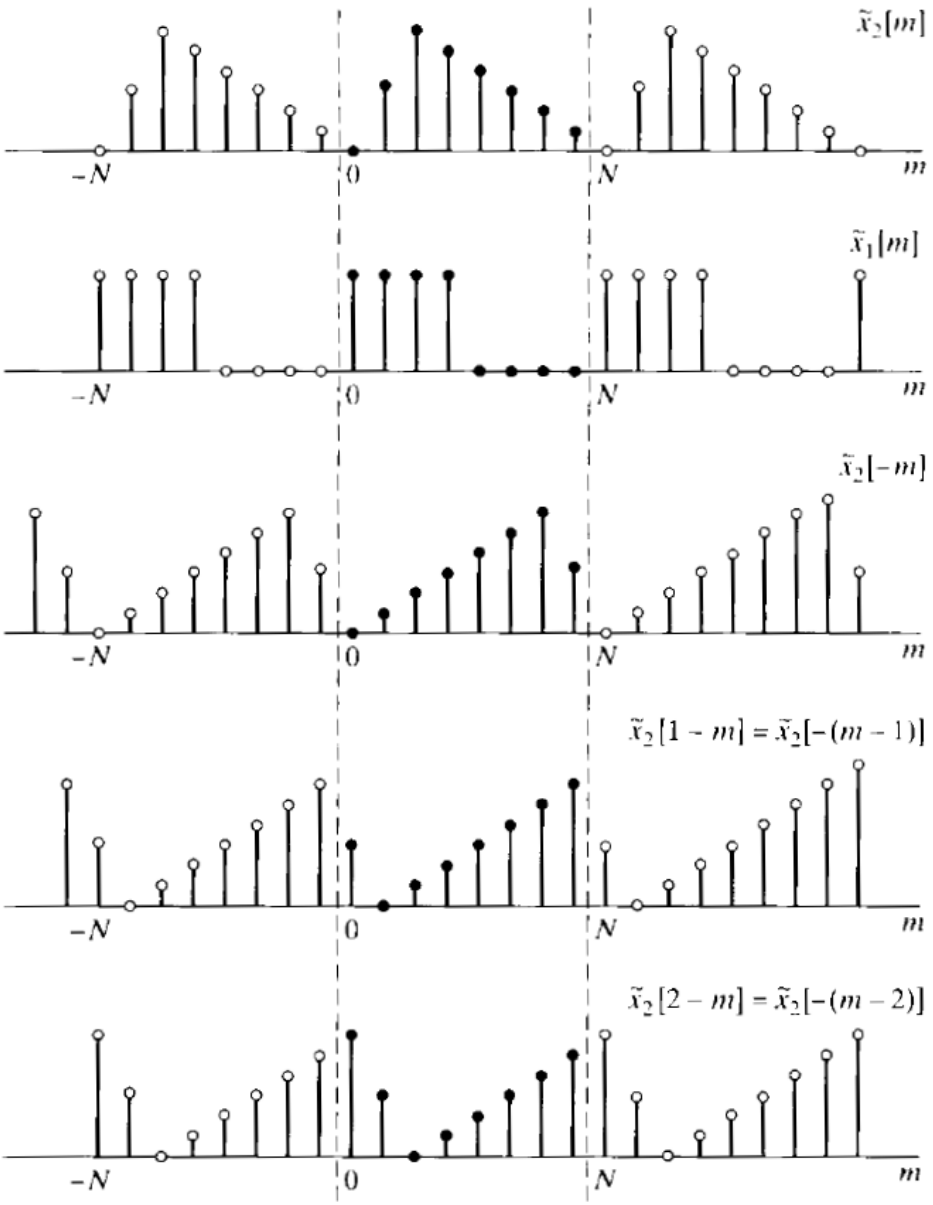

Figure B.5: Procedure for the periodic convolution of two periodic sequences. From [78].

A different type of convolution is the *circular convolution*. This is based on the Discrete Fourier Series (DFS), which is defined by

$$\tilde{X}[k] = \sum_{n=0}^{N-1} \tilde{x}[n] W_N^{kn}, \tag{B.24}$$

where W_N^{kn} is a twiddle factor, whose definition can be found in Eq. B.10, $\tilde{x}[n]$ is a periodic discrete sequence and $\tilde{X}[k]$ is the sequence of its DFS coefficients. Given two periodic sequences with period N, $\tilde{x}_1$ and $\tilde{x}_2$, their discrete Fourier series coefficients are denoted with $\tilde{X}_1[k]$ and $\tilde{X}_2[k]$. If the following equation holds

$$\tilde{X}_3[k] = \tilde{X}_1[k]\tilde{X}_2[k],$$

then the periodic sequence $\tilde{x}_3[n]$ with Fourier series coefficient $\tilde{X}_3[k]$ is obtained as

$$\tilde{x}_3[n] = \sum_{m=0}^{N-1} \tilde{x}_1[m]\tilde{x}_2[n-m]. \tag{B.25}$$

A convolution in the form of Eq. B.25 is called *periodic convolution*. An example illustrating some steps involved in the operation is in Fig. B.5. Similarly to the periodic case, one can consider two finite-duration sequences of length N, $x_1[n]$ and $x_2[n]$, corresponding to one period of $\tilde{x}_1$ and $\tilde{x}_2$ respectively, with DFTs $X_1[n]$ and $X_2[n]$. If $X_3[k] = X_1[k]X_2[k]$, sequence $x_3[n]$ whose DFT is $X_3[k]$, corresponds to one period of $\tilde{x}_3[k]$. Consequently, by using Eq. B.25,

$$x_3[n] = \sum_{m=0}^{N-1} \tilde{x}_1[m]\tilde{x}_2[n-m], \qquad 0 \le n \le N-1. \tag{B.26}$$

Introducing the notation

$$((m))_N = m \mod N,$$

leads to

$$\tilde{x}[n] = x[((n))_N], \tag{B.27}$$

and it is possible to note that that $((m))_N = m$ for $0 \le m \le N-1$. By using Eq. B.27, former Eq. B.26 can be written as

$$\begin{aligned} x_3 &= \sum_{m=0}^{N-1} x_1[((m))_N] x_2[((n-m))_N] \\ &= \sum_{m=0}^{N-1} x_1[m] x_2[((n-m))_N], \end{aligned} \qquad 0 \le n \le N-1. \tag{B.28}$$

Eq. B.28 represents the *circular convolution* of x_1 and x_2. As derived, a circular convolution is fundamentally a periodic convolution. Thus, by confronting Fig. B.4 and Fig. B.5, it is possible to show the differences between this approach and the aperiodic convolution. In the last case (Fig. B.4), one signal is time-reversed and then linearly shifted. The values of the superposition product are then summed. On the other hand, in the circular convolution (Fig. B.5) the second sequence is circularly time reversed and then circularly shifted before performing products and sums.

Another remark must be done on the simmetry property of the DFT. Considering periodic sequence $\tilde{x}$, it is easy to obtain, from the definition of the DFS,

$$\begin{aligned} \tilde{X}[k]^* &= \left(\sum_{n=0}^{N-1} \tilde{x}[n] W_N^{kn} \right)^* \\ &= \sum_{n=0}^{N-1} \tilde{x}^*[n] W_N^{k(-n)} \\ &= \sum_{n'=0}^{N-1} \tilde{x}^*[-n'] W_N^{kn'}, \end{aligned}$$

where $n' = -n$. By using Eq. B.27, the previous result can be extended to the DFT case, obtaining

$$X^*[k] \leftrightarrow \text{DFT}\, x^*[((-n))_N]. \tag{B.29}$$

Back to the estimate in Eq. B.23, one possibility to compute the aperiodic discrete convolution is to exploit equations B.28 and B.29. First, it can be noted that, being $\hat{\phi}_{xx}[-m] = \hat{\phi}_{xx}[m]$, one can consider only nonnegative values of m, thus $0 \leq m \leq M-1$. Considering the N-point DFT of $x[n]$, $X[k]$, computed with an FFT algorithm. If this is multiplied by $X^*[k]$, the DFT of the circular convolution of $x[n]$ and $x[((-n))_N]$ is obtained. This holds assuming sequence $x[n]$ to be real. By augmenting sequence $x[n]$ with zero-valued samples, the circular autocorrelation is forced to be equal to the aperiodic autocorrelation in Eq. B.23 for $0 \leq m \leq M-1$. This operation is called *zero-padding*. Such transition from the frequency domain to the time domain can be performed because of the preconditions on the stochastic process given by the Wiener-Khinchin theorem. Also, it is worth noticing that while the direct evaluation of Eq. B.23 has a $\text{O}(N^2)$ computational complexity, the FFT-based technique has a complexity of $\text{O}(2N\log_2 N)$. The value of N, which determines the number of zero samples to add to the sequence, can be chosen by noticing that it must be a power of two. Consequently, it results $M = Q$ and it is suitable to choose $N = 2Q$. The FFT-based technique for computing the autocorrelation is composed of the following steps.

1. Given $x[n]$, sequence of length Q, this is augmented with Q zero-valued samples (zero-padding), obtaining an N-valued sequence.
2. The N-point FFT of $x[n]$ is computed, obtaining $X[k]$.
3. $|X[k]|^2$ is evaluated.
4. Computation of the inverse FFT of $|X[k]|^2$.

Appendix C

State of the art in FFT hardware architectures

The two main approaches to FFT implementation in industry are called *pipelined* and *memory-based* [107, 36]. Pipelined architectures are characterised by the fact that all the butterflies of the same stage of the flowgraph are computed by the same *hardware butterfly*, also called *Processing Element* (PE). Thus, each stage of the pipeline computes a whole stage of the algorithm. These architectures process a continuous flow of data and are suitable for high-throughput applications. In memory-based, also called *in-place*, architectures only one butterfly is used and data is written to the same register or the same memory from which it is read. These structures tend to occupy less area than the former ones but they have a lower throughput. Also, they are not suitable for processing a continuous flow of data. Although this categorisation is generally valid, the literature also gives us examples of hybrid structures [75].

C.1 Pipelined structures

The most important innovation about pipelined architectures of the last years is possibly the introduction of the *radix-2^2* FFT Alg. [39]. In this paper, a distinction is made between *Multi-path Delay Commutator* (MDC), also called *feedforward*, and *Single-path Delay Feedback* (SDF), also called *feedback*, architectures. Feedback architectures have a throughput of $\frac{1 sample}{cycle}$ and need to store N locations, while feedforward architectures achieve a higher thoughput at the cost of a larger area occupation. The two types are illustrated in Fig. C.1. The radix-2^2 algorithm has the same multiplicative complexity as radix-4 algorithms, but retains the radix-2 butterfly structures. Consequently it uses less adders and multipliers if compared to radix-2 and radix-4 solutions. The flowgraph of the algorithm is shown in Fig. C.2. The concept behind radix-2^2 was later generalised with the class of radix-2^k algorithms [40].

An interesting pipelined architecture was designed in[18], by observing that the feedforward scheme can achieve better hardware utilization while the feedback scheme can lead to memory saving. Thus, a structure composed of two radix-2 DIT based SDF stages followed by radix-4 MDC is discussed. Operations inside the first two stages are bit-serialised. The scheme halves the required memory for TFs by exploiting symmetry in the complex plane.

An automatic generator of split-radix parallel pipeline FFT processors is described in [88]. The design is thought as a flexible part of speech-processing, noise and echo canceling systems and shows good timing performances if compared to commercial IPs. Operating frequency, throughput, CLB-

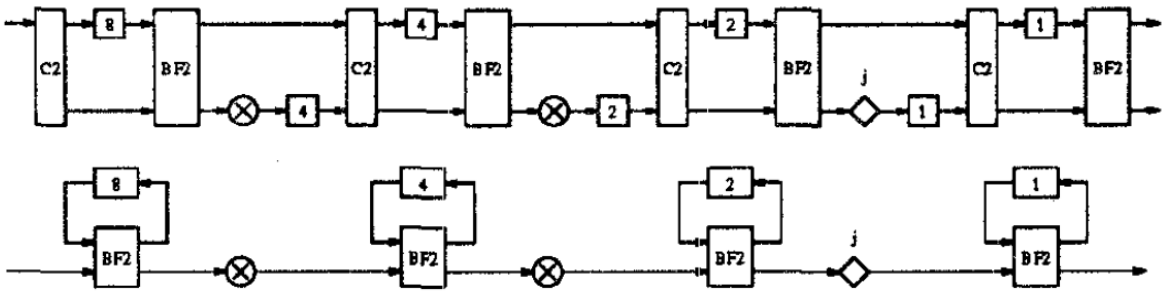

Figure C.1: Classic types of FFT pipelined architectures. From [39].

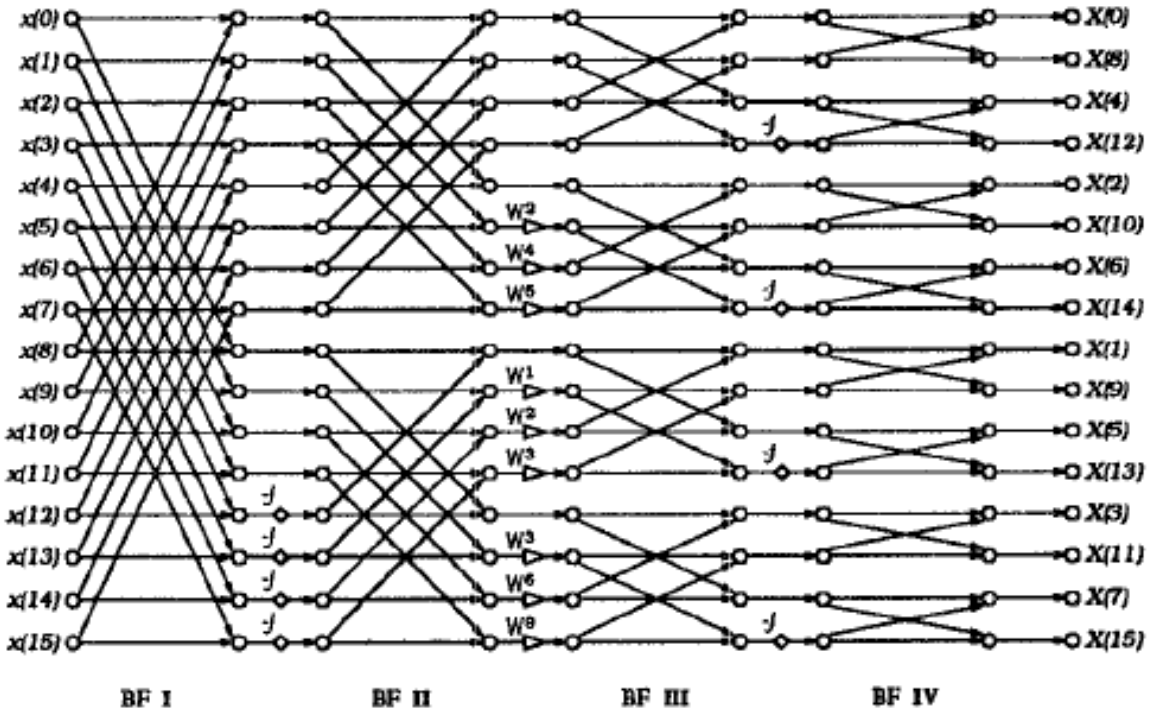

Figure C.2: Flowgraph of the radix-2^2 DIF FFT algorithm. $N = 16$. From [39].

count and power dissipation are selectable parameters of the discussed tool. But, being operating clock fixed at $65MHz$, figures suggest that FPGA resource occupation tends to be higher than in drop-in IP modules. Another split-radix-based design aiming at computation time minimization was proposed in [32]. The main blocks are of two types: *computing elements* (CE) and *delay commutators* (DC). The architecture is of the MDC type and it is obtained by alternating CE and DC in the strucutre of the pipeline. Although the minimum requirements for both data and signal length are of 16 bits (because of SNR features), the maximum clock rate is of $350MHz$ and computation time is well below values obtained with IPs of FPGA vendors. The drawback of the design is in resource occupation: the number of needed complex multipliers is $2\log_2 N - 4$, thus with $N = 256$ the count of used 18×18 FPGA multipliers is 48.

The SPIRAL project, started at Carnegie Mellon University [94] with the goal of developing a fully automatic trigonometric transform code generation system capable of optimizing its output on specified software architectures. In [75] this approach is extended to hardware thanks to the *Pease*

Alg. [83], given by the factorization

$$\boldsymbol{X} = \underbrace{R_N\Big(\prod_{i=0}^{\log_2 N-1} T_i (I_{\frac{N}{2}} \otimes F_2) L^N_{\frac{N}{2}}\Big)}_{\text{DFT}_N} \boldsymbol{x}. \tag{C.1}$$

In this equation R_N denotes the bit-reversal permutation matrix and T_i is a diagonal matrix that multiplies the elements of a vector by the proper TFs at stage i . $I_{\frac{N}{2}} \otimes F_2$ is a $(N \times N)$ block matrix with F_2 blocks in its diagonal and zeros elsewhere. $L^N_{\frac{N}{2}}$ is the matrix representing the *perfect shuffle* permutation which, in the case of $N = 8$ is

$$L^8_4 : [0,1,2,3,4,5,6,7]^\intercal \rightarrow [0,4,1,5,2,6,3,7]^\intercal.$$

Eq. C.1 states that the DFT operation consists in repeating $\log_2 N$ times the following sequence of steps, given input array $\boldsymbol{x}$:

1. apply the perfect shuffle permutation $L^N_{\frac{N}{2}}$ to the current data array.
2. for each couple of adjacent elements in the data array, apply the butterfly, which is represented by matrix F_2.
3. Multiply by the twiddle factors in T_i.

At the end of the procedure, the bit-reversal permutation R_N is applied. The flowgraph of the algorithm is shown in Fig. C.3 for the case $N = 8$ and butterflies based on radix-2 DIF. In [71] it was observed that the tensor product $I_{\frac{N}{2}} \otimes F_2$ which indicates $\frac{N}{2}$ parallel instantiations of block F_2 can be conceived as reuse in time or *streaming reuse*. Similarly, product $\prod_{i=0}^{\log_2 N-1} A_n$ can be thought not as a cascade of $\log_2 N$ equal A_n blocks, but as an *iterative reuse* of the same block. The combination of the two concepts leads to different architectures whose parameters are computed via sophisticated optimization algorithms depending on desired performances. Pipelined architectures can be thought of as a particular case in which full streaming reuse and no iterative reuse are selected as shown in Fig. C.4.

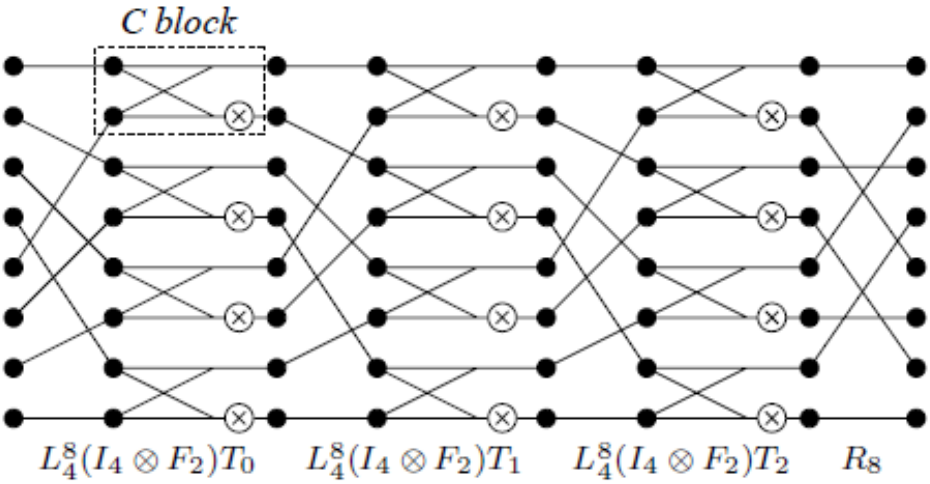

Figure C.3: Pease's algorithm flowgraph, $N = 8$. From [75].

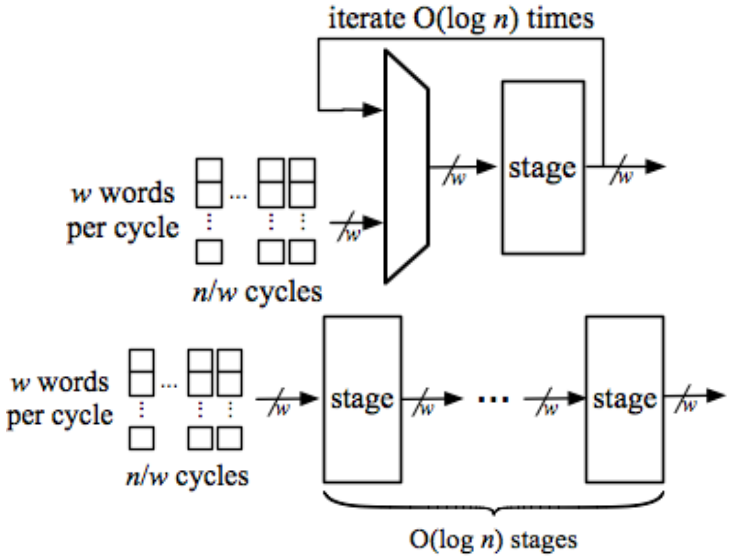

Figure C.4: SPIRAL project architectures. From [70].

The forementioned radix-2^2 algorithm was implemented on a Xilinx Virtex-4 FPGA in [52], where the resource usage were analysed for a 2^{14}-point processor. The obtained clock frequency is $105MHz$, but used multipliers are 24, well below the result obtained in the [32]. Anyway it is noted that to remove twiddle factors storage it is necessary to resort to the CORDIC algorithm. Better results in terms of clock frequency are achieved in [141], reaching $235.6MHz$ on the same FPGA family as the former paper. In this last paper it is also noted that the R2^2SDF is not suited for adding pipeline registers within individual butterfly elements because this would break the timing for the data feedback path. Thus CORDIC could not be used inside the butterfly in place of a complex multiplier.

In [100] another automatic RTL code generator of pipelined FFT processors is illustrated and extensively explored. The tool is also capable of simulating the generated VHDL core and extracting all the figures of merit of the selected implementation. Architectures are based on fixed-radix algorithms (radix-2 or radix-4) and use a fixed-point representation of internal data. Code for multiplication, addition, substraction, rounding and two's complement is contained in a package that is separated from the generated data, so that it is possible to use vendor-specific hardware code if desired. Flexibility is the strength of this approach, but sensitivity of clock frequency is its weakness. With $8bit$ precision of twiddle factor representation, pipeline clocking speed falls below $100MHz$ for all the generable architectures.

In most of the cases custom architectures (both pipelined and in-place) are compared to commercial IPs. Xilinx LogiCORE FFT [123] and Altera FFT MegaCore [6] are the most famous drop-in modules. A radix-2^2SDF FFT open-source IP core was also released by XionLogic in [125]. The most recent pipelined approaches such as [35] are oriented towards optimization of hardware resource usage by exploiting innovative variations of radix-2^k, without abandoning the strong MDC scheme.

C.2 Memory-based structures

The context of memory-based architectures is more varied than its counterpart. This is conceivably because the number of PEs is not necessarily fixed with N, so new solutions involving different overall structures can be explored, instead of focusing on the optimization of butterflies.

A processor for OFDM receivers with PEs based either on radix-4 or radix-8 was proposed in [81].

Complex multiplications are performed by using a complex multiplier, realised with three Booth real multipliers, or with a CORDIC rotator. The system uses two memories: a *transposition memory* (TM) and a *shuffle memory* (SM), and both reading and writing are managed by partitioning the clock cycle. The two approaches to multiplication are compared, showing that the CORDIC scheme uses more logic gates than the other solution. The drawback of the design is in the complexity of memory control.

A novel parallel-pipelined architecture for FFT using CORDIC was proposed in [132]. Its structure is illustrated in Fig. C.2 in the case of $N = 128$. Four radix-4 butterflies are fed in parallel by a multiplexer. Computations inside the PEs are pipelined, with the first stages implementing algebric sums and the last performing CORDIC rotations to compute complex multiplications, as shown in Fig. C.2. The feedback connection allows the feeding of input buffers again after data shuffling, which is performed thanks to cross-connections. The output stage is composed of radix-2 FFT hardware that contains memory elements. This architecture is particularly suitable for high-speed telecommunication systems, but its drawback is in the necessity of two different types of computing PEs. Also, the radix-2 stages are used only when radix-4 calculations are completed, thus hardware usage is not at its optimum.

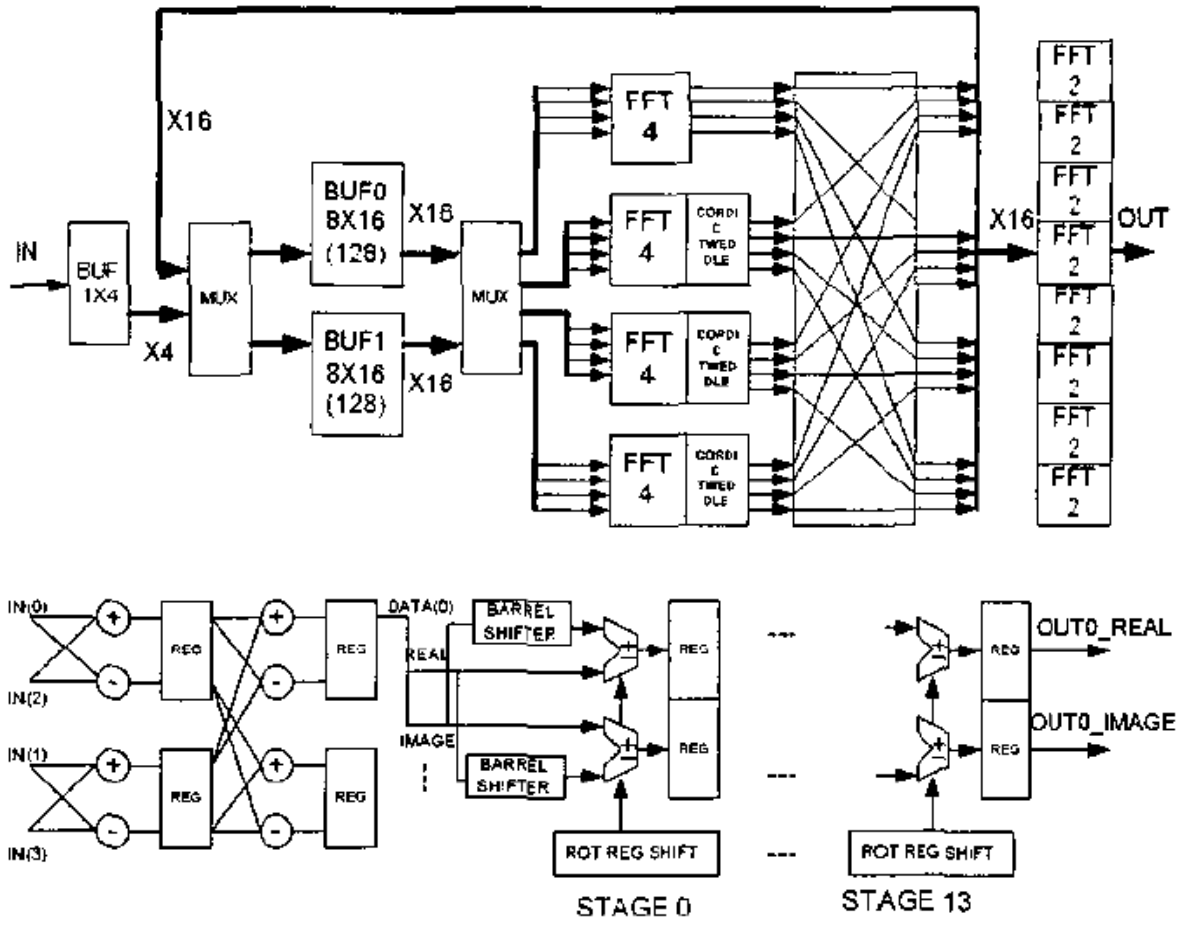

Figure C.5: zhang-chen's structure of parallel FFT with CORDIC. From [132].

A pure CORDIC-based FFT is obtained in [42] by expressing the DFT as seen in Eq. B.7 and relating the expression to CORDIC rotations. Scale factor correction is performed at the end of all the

computations. Compared to the conventional multiplier approach, this is slightly slower but has an improved accuracy.

The strength of memory-based architectures is in the compactness of the design. This is shown in [114], where architecture illustrated in Fig. C.6 is described. The proposed scheme is based on radix-4 DIT and uses a single butterfly engine. Operation of the PE is managed by a sequencer, which selects twiddle factors stored in a ROM, input data from a RAM, and output locations to a second RAM. The design was coded in VHDL, tested on a Xilinx Spartan-3 FPGA and compared to the Xilinx IP. Although this solution is slower than the device-optimized module of the vendor, improved hardware usage is shown.

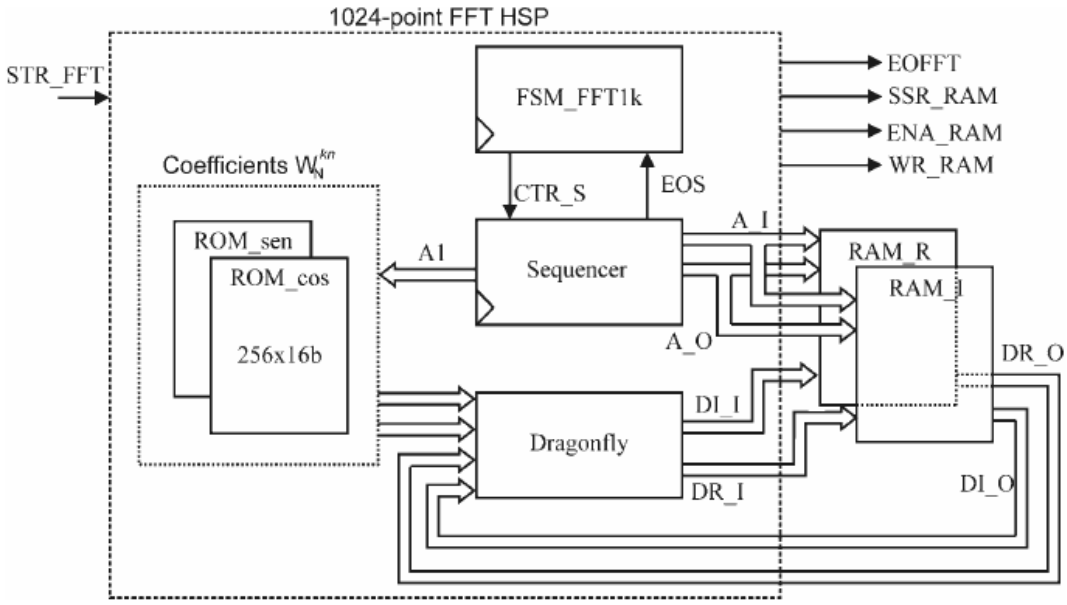

Figure C.6: Vite-Frias' architecture Memory-based radix-4 design scheme. From [114].

An innovative split-radix FFT processor with a modified CORDIC-based butterfly was proposed in [108]. The global structure is shown in Fig. C.7. Like in the previously discussed paper, a single PE performs the computations, and both memory access and twiddle factors are managed by an overall controller. By contrast, in this case all multiplications are performed with CORDIC engines, both inside the twiddle factor generator and inside the butterfly processor. An extra CORDIC pipeline stage is needed for gain correction. It is possible to implement a multi-butterfly system, thus having both parallelism and sequential processing. The design is oriented towards a VLSI implementation, consequently metrics are given in terms of equivalent gates. It is shown that the proposed CORDIC-based approach implements the arithmetic unit with $18000 equivalent gates$, against the $32000 equivalent gates$ of the classic approach with clocked Booth multipliers and ROM for TFs.

The case of prime N-length DFT is analyzed in [20]. An architecture based on cyclic convolution algorithm is used to solve small-length FFTs and the *Winograd* FFT algorithm is used to manage the case of a larger number of input points. The obtained structure is modeled by using *systolic arrays*. A systolic array is a regular array of simple processors connected in such a manner that each processor may exchange information with only its neighbour to the right and left and processors at the beginning and end of the row are used for input and output respectively [38]. With this technique, computational complexity of the DFT is reduced to $O(\log N)$, controlling at the same time the number of required multipliers. The advantage of this solution is in the possiblity of computing even and prime-length DFTs, but its drawback is in system complexity.

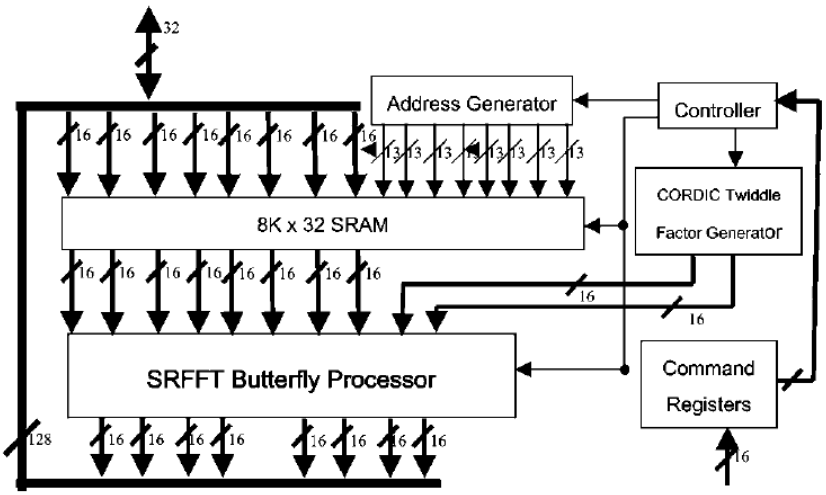

Figure C.7: Sung's architecture 2^{13}-point CORDIC-based split-radix design scheme. From [108].

A peculiar type of implementation is based on *polyphase filter banks* (PFB). These filters help reduce DFT leakage when it is an important issue [30] by lowering the sampling rate (*decimation*), filtering the signal and then interpolating. Among the architectures using PFB, one proposed by in[43] is worth noticing: *Common Sub-Expression Elimination* (CSE), a technique tipically used by compilers, is applied to hardware in order to reduce multipliers. The logic needed in comparison the conventional PFB approach is shown to be less, but usage of multipliers formed by blocks of look-up tables instead of built-in FPGA blocks leads to a lower throughput.

Another typical memory-based design was proposed in [3]. The base algorithm used is Radix-2 DIT_{RN} and the goal of the design is to achieve a complete butterfly operation in one clock cycle. The architecture design is *buffered*, meaning that all the input samples must be available before the execution can start, and its structure is illustrated in Fig. C.8. Samples are supplied to a bit-reversal module until an N-sized buffer is filled. Then, a butterfly operation is executed at every clock cycle until computation is complete. Similarly to [114] an address generator selects input and output addresses for data and the twiddle factor index. A State Machine Manager works as global controller. TFs can be retrieved from a look-up table (LUT) or from a CORDIC-based generation system.This approach differs from [81, 132, 42] because complex multiplications are still performed by using conventional multipliers inside the PE, and CORDIC is only used as a generation algorithm. This reduces butterfly computation latency while avoiding the usage of ROM. In [108] the CORDIC-based TF generator was conceived merely as a technique for computing the rotational angle θ_l, and the actual complex value was not calculated. The discussed memory-based design uses fixed-point configurable logic, and offers an optional magnitude calculation block, which can be used in applications where complex results are not needed. The architecture was implemented on an Altera Cyclone III device and tested against Altera's reference design and other architectures in literature: while the clock cycle count is near the reference design, memory and FPGA resource saving is evident from both the proposed structure and the figures.

Suleiman proposed a FFT hardware implementation of a Constant Geometry (CG) algorithm for trigonometric transforms [107]. The architecture is not pipelined, but its structure is similar to [71]. The designer of the overall PSD computer can choose how many PE to instantiate in the processor: the more the PE, the faster the execution of the algorithm. All the PEs compute butterflies belonging to the same stage of the flowgraph in parallel, and shuffling of data is performed thanks to a particular

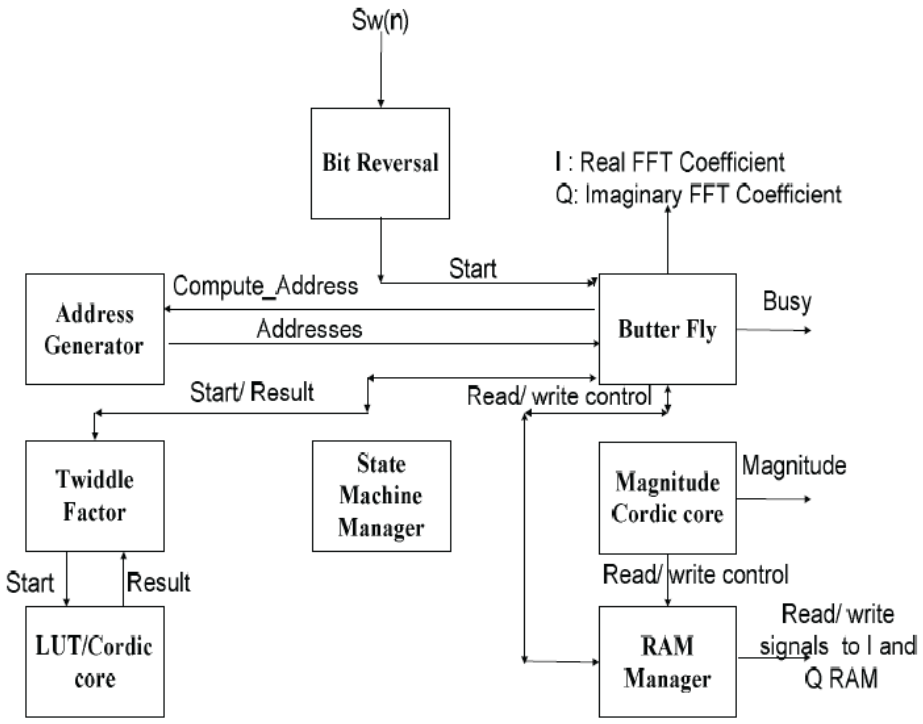

Figure C.8: Al Sallab's memory-optimized FFT architecture. From [3].

interconnection structure composed of a perfect shuffle network and a FIFO queue. No memory is used because shuffling is performed thanks to criss-cross connections between registers, thus the structure can be classified as *not-in-place* CG-FFT. The architecture's flexibility makes it particularly suitable for both high-speed and resource-saving designs.

In [121] an addressing scheme and matching generator logic expressely designed in order to avoid any ROM usage for TF was proposed and explored. First it is observed that for different FFT stages, the angles increase always one step per clock cycle. Thus, the structure of an angle generator, which is illustrated in Fig. C.2, is described. The accumulator is driven by a control depending on the current FFT butterfly stage and RAM address bits. This approach generalises the FFT algorithm from radix-2 to any radix. The symmetrical achitecture in Fig. C.2 takes advantage of this observation by reducing the total amount of memory bits up to 20% for radix-2 and up to 33% for radix-r. CORDIC micro-rotations are performed inside the butterfly which has an internal pipelined structure. The avoidance of TF or angle storage ROM leads to a reduction of total memory bits, and, in a lesser amount, of logic elements.

An automatic FFT code generator for FPGAs with the possiblity to choose between pipelined and in-place architectures, both oriented towards continuous throughput applications, was presented in [79]. The generated pipelined architecture is a conventional radix-2 MDC architecture with memory buffers for the reordering stage. The in-place counterpart is available both with a single butterfly engine and multiple PEs. In the last case, the number of PEs is obtained as $\lceil \frac{\log_2 N}{2} \rceil$, thus it is not a degree of freedom for the designer. The number of memory blocks is equal to the number of PEs plus two modules for input and output streaming. Fig. C.10 shows the discussed multi-butterfly structure. Each memory block must be accessible from each butterfly block, as well as input and output modules. Thus, for large values of N both large multiplexers and capacious memory blocks must be instantiated. Another issue is the necessity to have a TF LUT for each butterfly. Consequently, synthesis results on Xilinx Virtex-5 show that the MDC solution is more resource-efficient if compared to

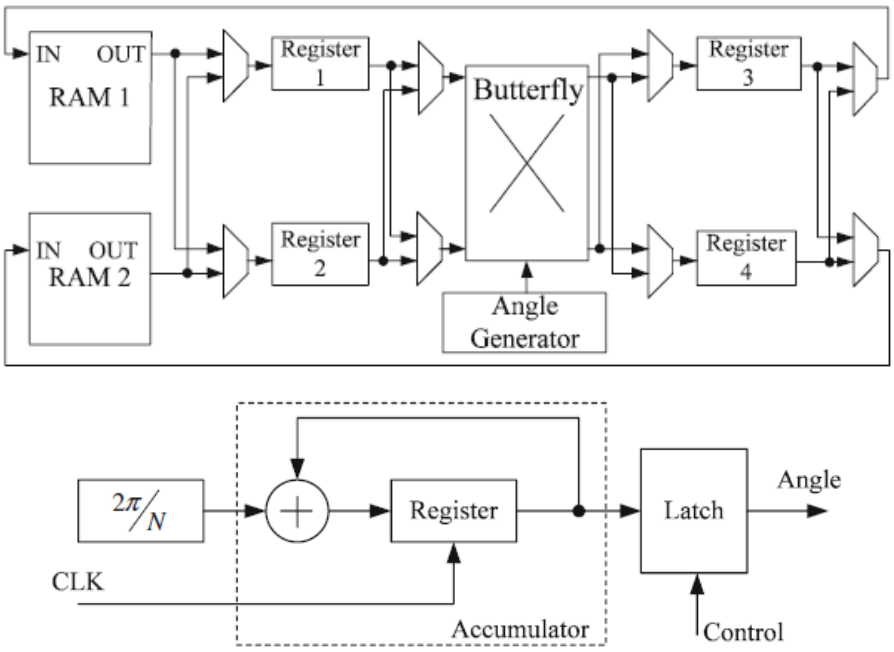

Figure C.9: Xiao's memory reduced CORDIC FFT. From [121].

the forementioned in-place approach.

The most cited FFT core is possibly LogiCORE IP FFT from Xilinx [123]. The vendor's IP is highly configurable, permitting many transform sizes and arithmetic types. Selectable architectures are from the fastest to the lightest (as shown in Fig. C.11):

- Pipelined, streaming I/O.
- Radix-4, burst I/O.
- Radix-2, burst I/O.
- Radix-2 Lite, burst I/O.

The first solution uses several radix-2 PEs to in order to offer continuous data processing. The architecture, which is represented in Fig. C.12, is of the radix-2 SDF type. The IP module expects the input to be in natural order, and if natural order is also requested for output values, a shuffling block using additional memory resource is utilized. The user has flexilbity to select the number of stages using block RAM for data and TF storage, and the remaining stages use distributed memory (FPGA LUTs). The counterpart burst I/O architectures use a single PE engine, as shown in Fig. C.13 that loads and unloads data separately from calculating the transform. Data loading and unloading from memory can be overlapped if data is unloaded in digit-reversed order. The radix-2 lite version of Fig. C.2 uses one shared adder/subtractor reducing resource usage but increasing computation time. Also, one cycle is used to multiply real values and the second to multiply complex values. Xilinx' competitor Altera proposes its MegaCore FFT [6]. This IP core allows the same customizability of the former, with different structural solutions. Four different I/O data flow architectures are selectable:

Streaming Allows continuous processing of input data, and outputs a continuous complex data stream.

Variable Streaming Produces a continuous stream of data, similarly to the Streaming architecture.

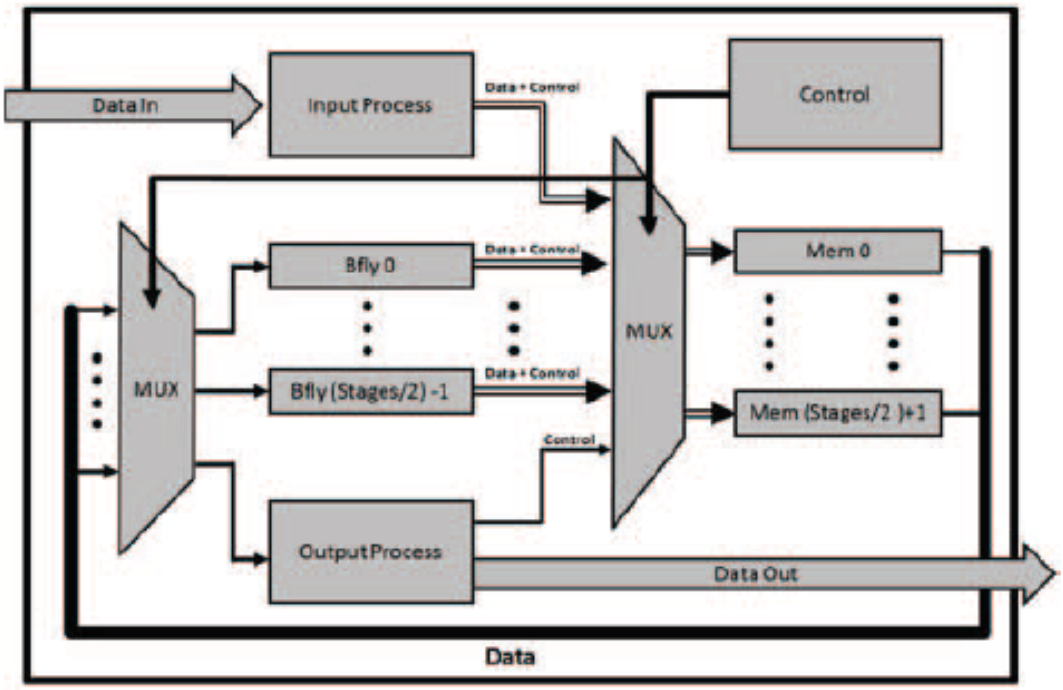

Figure C.10: O'sullivan's in-place FFT architecture. From [79].

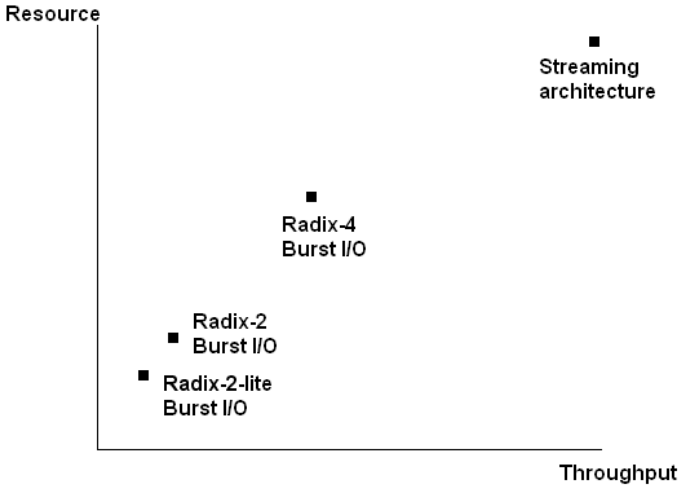

Figure C.11: Comparison of available architectures for Xilinx LogiCORE IP. From [123].

Buffered Burst Requires fewer memory resources than the streaming I/O data flow, but trades off with an average block throughput reduction.

Burst Operates similarly to the Buffered Burst architecture, except that the burst architecture requires even lower memory resources.

Block Floating Point (BFP) is used throughout the designs as a compromise between fixed-point and full floating-point to mantain a high signal to noise ratio.Depending on the I/O data flow architecture and the arithmetic chosen, the FFT MegaCore Function can implement different structures.

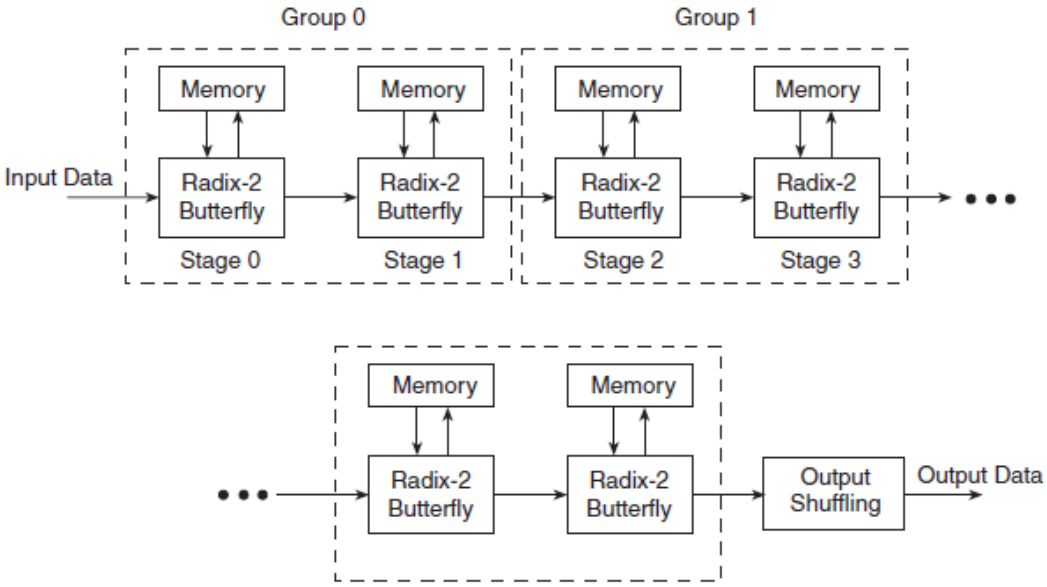

Figure C.12: Xilinx LogiCORE *pipelined, streaming I/O* architecture. From [123].

- Radix-2^2 SDF, as in Fig. C.1 is composed of $log_2 N$ pipeline stages and is used for fixed-point streaming variations.
- Mixed radix-4/2 is a pipelined architecture for floating point streaming variations. Each stage contains a single butterfly unit and a feedback delay unit. In each stage the number of cycles of delay set by the feedback delay unit is one quarter of the number of cycles of delay in the previous stage. This operation aligns input samples correctly for the calculations. The output of the pipeline is in index-reversed order.
- Quad-output FFT engine architecture is used for streaming, buffered burst and burst variations when transform time is to be minimized. The structure of the FFT is shown in Fig. C.2. Input complex values are read in parallel and re-ordered by a switch (SW). All the four outputs of the PE are processed together in a single clock cycle, by using TFs stored in three different ROMs. Three complex multipliers are needed. Block Floating-Point Units (BFPU) evaluate the results and perform a scaling of the representation depending on accuracy requirements.
- Single-output FFT engine architecture is used both for buffered burst and burst variations. The architectural structure is illustrated in Fig. C.2. Compared to the former solution, this occupies less resources because both SWs are substituted by Time Division Multiplexers (TDM), and the same approach is used to manage a single complex multiplier.

Although both vendor designs are very hardware-optimized and flexible, their drawback is in the lack of portability towards ASIC technology.

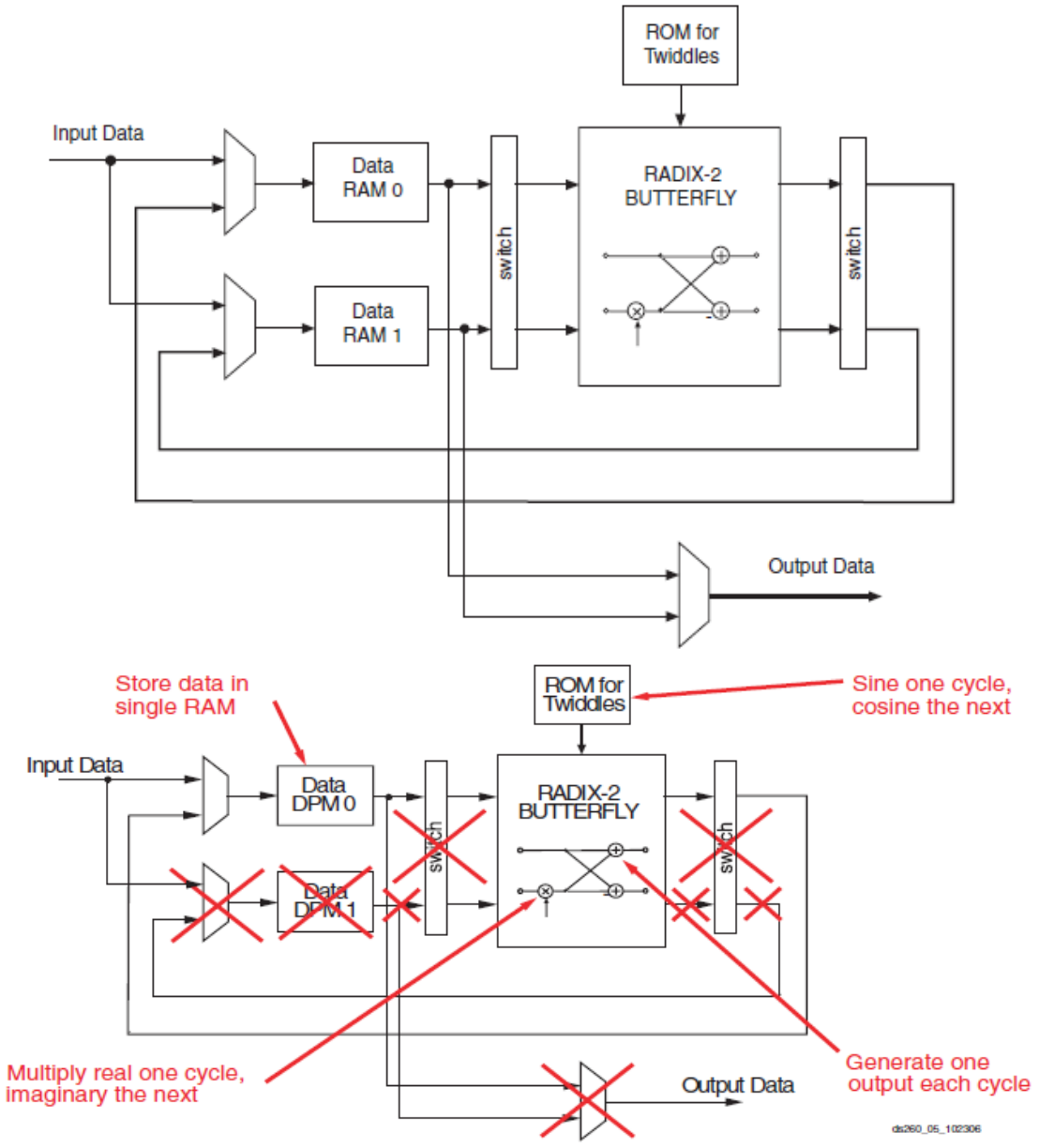

Figure C.13: Available Xilinx LogiCORE in-place *burst I/O* FFT architectures. From [123].

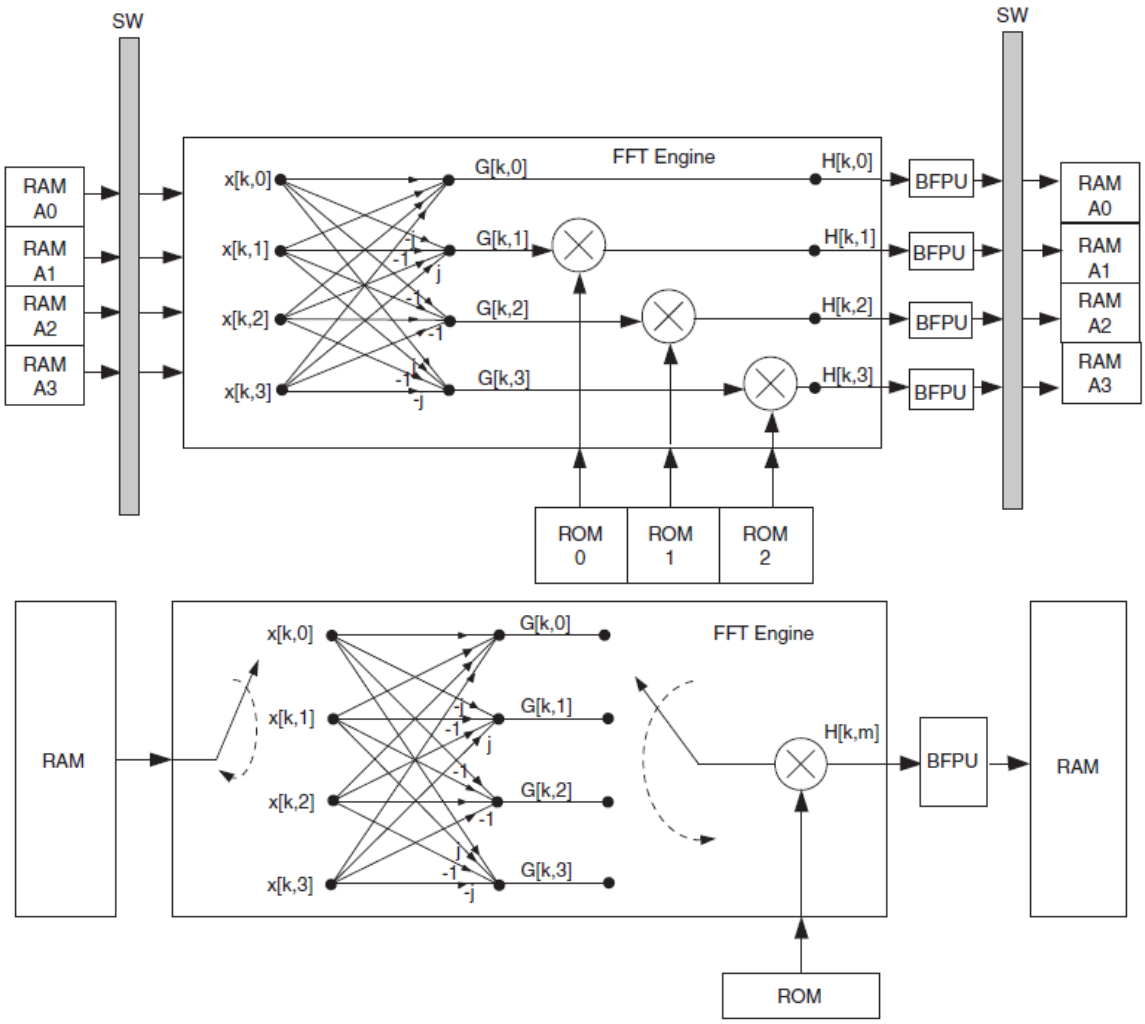

Figure C.14: Available Altera MegaCore in-place FFT architectures. From [6].

C.3 FFT-specific CORDIC

Considering papers analysed in sections C.1 and C.2, we discern two different approaches to CORDIC in FFT hardware architectures: the *in-processor rotator*, which is more common, and the the *twiddle factor generator*. An example of *in-processor rotator* can be seen in [132], where the CORDIC pipeline structure is embedded in the PE as a substitute for the complex multiplier. Thus, in this case computation of a butterfly requires two complex data and the TF angle. The counterpart of this approach is in [3], where the TF LUT is replaced with a CORDIC generation system, but the PE retains its structure. Consequently, the computation of a butterfly requires two complex data and the complex value of the TF. In the latter case, multiplication is usually performed by using conventional hardware multipliers.

In [129] the design of a CORDIC-based pipelined PE is explored. The proposed scheme uses only $\frac{B}{2}$ shift-and-add operations for the computation of each TF, where B is the length of data representation in bits. The idea behind the design is the decomposition of micro-rotations into *coarse component* and *fine component*, which satisfies the approximation rule $\tan 2^{-i} \approx 2^{-i}$ with $i \geq \frac{B}{3}$. The architecture of the proposed CORDIC system is illustrated in Fig. C.15. At stage $k = 0, \ldots, \log_2 N - 1$, requested twiddle factors are a multiple of $W_N^{2^{k-1}}$, so a $\log_2 N$-sized memory is necessary for storing both rotation angles and optimized rotation sequences. Moreover, it must be noted that in the design many redundant micro-rotations are skipped, thus it is necessary to compensate variable scale factors either using the LUT or computing the CORDIC gain depending on iteration step i.

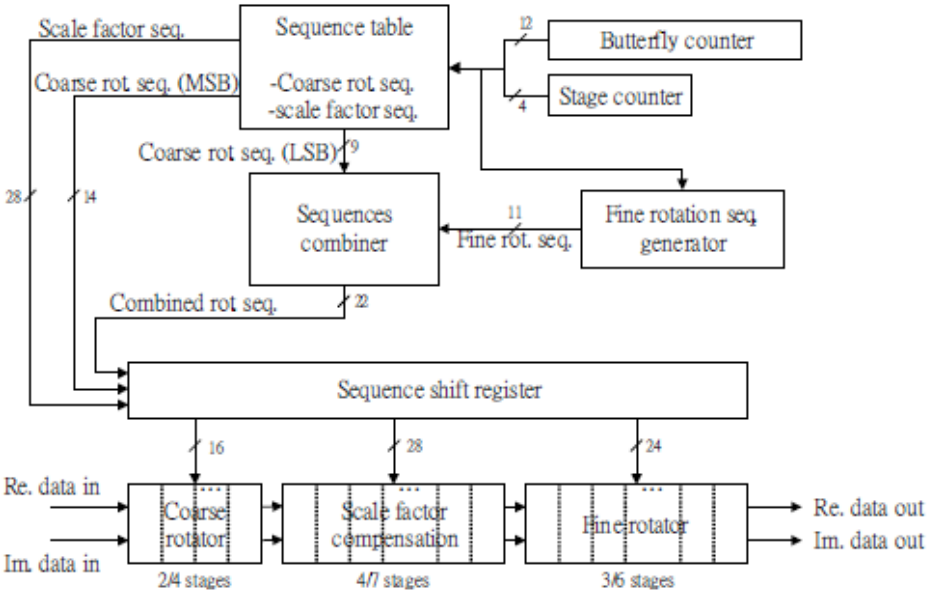

Figure C.15: Yu's CORDIC architecture. From [129].

The first category of CORDIC FFT hardware is also discussed in [33]. In this paper a CORDIC FFT-optimized algorithm is derived from the angle sequence of TFs for any radix. The rotation generator computes the first micro-rotations such as a conventional CORDIC, checking the remainder angle and setting the direction according to Eq. B.20. The rest of computations are performed considering the following approximations:

$$\alpha_i \approx \tan \alpha_i$$
$$\frac{\alpha_i}{\alpha_{i+1}} \approx 2.$$

Furthermore, in the proposed design the 45° rotation is not performed, so the scaling factor is set equal to

$$K = \prod_{i=1}^{M} \cos(\tan^{-1} 2^{-i}) \approx 0.8588$$

and afterwards approximated with

$$K \approx 0.8594 = 1 - 2^{-3} - 2^{-6}.$$

Consequently, CORDIC gain compensation is performed as a simple shift-and-add routine. The overall structure of the proposed CORDIC system is summarized in Fig. C.3. The advantage of this design, or rather its lack of a ROM for angles storage, becomes effective when the number of input points of the FFT is above 2^{10}, as shown in Fig. C.3. When N is below this value, the high complexity of the control makes the design less appropriate if compared to the conventional approach.

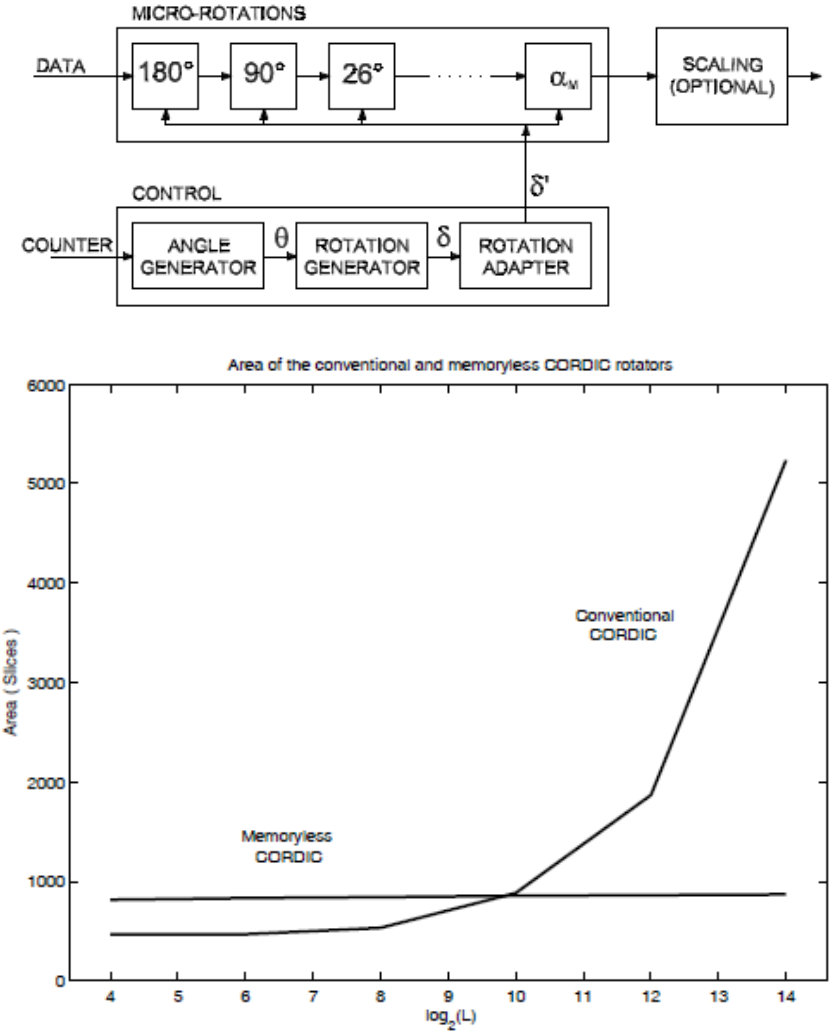

Figure C.16: Garrido's FFT-oriented CORDIC. From [33].

Implementation of CORDIC in FFT is a critical issue. It has been noted that since large-N computations require a number of processors, choosing a CORDIC-based PE increases the area of the design if compared to the classic multiplier-based approach [10]. CORDIC processors, on the counterpart, tend to have higher operating clock frequencies: the tradeoff determines the right structural approach.