POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

FPGA-Assisted Verification of Digitally-Assisted Analog-to-Digital Converter Calibration Algorithms

Tesi di Laurea Magistrale in
Electronics Engineering - Ingegneria Elettronica

Author: **Simone Guglielmino**

Student ID: 969891
Advisor: Prof. Andrea G. Bonfanti
Co-advisors: Gabriele Be'
Academic Year: 2021-2022

Abstract

This *master thesis* presents a new verification method that implements an 8-channel Time-Interleaved (TI) Analog-to-Digital Converter (ADC), which requires extensive digital calibrations, on a Field-Programmable Gate Array (FPGA). The focus of this *thesis* is the verification of the calibration algorithms of the converter, which are needed to compensate for the effects of gain and offset mismatch between the cores. The implementation on the FPGA allows simulating in nearly real-time these calibration algorithms for hours/days in order to detect the presence of misbehaviors that can occur in such a long time, like, for instance, saturation or drift of the accumulators or integration of errors caused by the finite precision of the digital implementation. These flaws of the algorithms are difficult to detect in standard VHDL or MATLAB simulations. In fact, VHDL simulations are too slow to detect these errors in a reasonable amount of time, whereas MATLAB is not suited to accurately and realistically describe a digital system of reasonable complexity. To implement the ADC on an FPGA we have developed a digital approximation of the analog sections of the converter. The implementation on the FPGA has been carried out by considering the different architectures possible, modelling and comparing them when needed in MATLAB, before eventually synthesizing them on the FPGA. The results obtained with the model implemented on the FPGA have been then compared with the VHDL and MATLAB simulation outputs. The comparison has shown a comparable accuracy but a simulation time improvement of a factor 1185 with respect to VHDL simulations. This improvement can be even larger (up to about 10000) if a higher-speed communication interface, i.e., PCIe is adopted.

Keywords: FPGA, Mixed-signal, Verification, Prototyping, Analog-to-Digital Converter, ADC, Time-Interleaved, Digitally-Assisted Analog Circuit

Abstract in lingua italiana

Questa *tesi* presenta un nuovo metodo di verifica basato sull'implementazione di un Convertitore Analogico-Digitale (ADC) Time-Interleaved (TI) ad 8 canali, che fa largo uso di algoritmi di calibrazione digitali, su un Field-Programmable Gate Array (FPGA). L'obiettivo principale di questa *tesi* è la verifica degli algoritmi di calibrazione del convertitore, necessari per compensare gli effetti dei mismatch di offset e gain dei vari canali. L'implementazione su FPGA permette di simulare quasi in tempo reale gli algoritmi di calibrazione per ore/giorni in modo da rilevare la presenza di eventuali comportamenti errati che possono verificarsi in intervalli di tempo estesi, come, ad esempio, la saturazione o la deriva degli accumulatori, o l'integrazione di errori dovuti alla precisione finita dell'implementazione digitale. Questi difetti degli algoritmi sono difficili da rilevare usando simulazioni VHDL o MATLAB standard. Infatti, le simulazioni VHDL sono troppo lente per rilevare questo tipo di errori in tempi ragionevoli, mentre le simulazioni MATLAB non sono adatte a rappresentare accuratamente e realisticamente sistemi digitali complessi. Per implementare l'ADC su un FPGA abbiamo sviluppato un'approssimazione digitale delle sezioni analogiche del convertitore. L'implementazione è stata condotta considerando le diverse architetture possibili, modellandole e comparandole in MATLAB quando necessario, prima di sintetizzarle su un FPGA. I risultati ottenuti usando il modello implementato su FPGA sono stati confrontati con i risultati delle simulazioni VHDL e MATLAB. Il confronto ha mostrato un'accuratezza comparabile, con una riduzione del tempo di simulazione di un fattore 1185 rispetto alle simulazioni VHDL. Tale miglioramento può essere ancora maggiore (fino a circa 10000) utilizzando un'interfaccia di comunicazione ad alta velocità, ad esempio, PCIe.

Parole chiave: FPGA, Mixed-signal, Verifica, Prototipazione, Convertitore Analogico Digitale, ADC, Time-Interleaved, Circuito Analogico Assistito Digitalmente

Contents

Introduction

Modern wireless communication standards like *5G* and *Wi-Fi 6/6E/7* pose significant challenges to the physical layer of the protocols, requiring fast, small, accurate, and low-power transceivers[1]. In transceivers, the interface between the real world and the rest of the (digital) system is implemented by Analog-to-Digital Converters (ADCs). Thus, the performances of the ADC have the utmost importance for the performances of the transceiver.

To push forward the performances of ADCs, W.C. Black and D.A. Hodges proposed in 1980 [1] the concept of Time-Interleaved (TI) converter. This kind of converter uses multiple interleaved sub-converters to achieve a higher conversion rate. The main issue of such converters is that they are subject to a degradation of the performances due to the presence of mismatches (e.g., gain and offset) between the core sub-converters. To correct these mismatches, and improve the performances of the whole converter there is a need for calibration. Multiple approaches to calibration are found in literature, some implementing the calibration in the analog domain, some others in the digital one, and a few using mixed-signal techniques.

Digital electronics benefits from the technological scaling of CMOS process, thus it makes sense to try to calibrate the non-idealities in the digital domain. This approach has a few noteworthy advantages. First of all, the aforementioned technological scaling continuously improves the performances (area occupation, energy efficiency, etc.) of digital electronics. The second advantage is that moving the complexity of the design into the digital domain relaxes some of the constraints of analog designs in a so-called digitally-assisted analog design [2].

The focus of this *master thesis* is the verification of the calibration algorithms of the TI converter, which are needed to compensate for the effects of gain and offset mismatch between the cores. In this *thesis* we propose a novel verification methodology that implements the ADC (in this case an 8-channel TI-ADC), and its calibration algorithms on

[1] transceiver: a transceiver (*portmanteau* of transmitter and receiver) is a communication device that can both transmit and receive.

a Field-Programmable Gate Array (FPGA). The advantages of this technique are presented in chapter 1 and are related to the faster simulation/emulation speed and greater reliability thanks to the execution on a real hardware.

This *master thesis* is organized as in the following.

Chapter 1 introduces the concept of verification in an IC design flow, focusing on the evolution and state-of-the-art of digital verification and simulation/emulation techniques. It briefly describes analog and mixed-signal methodologies before presenting the proposed verification method (which is to use an FPGA to verify a digital-assisted A/D converter), and its advantages. Then the chapter briefly introduces ADCs before focusing on the concept of TI converter, providing an analytical model of an ideal M-channel TI-ADC. The chapter concludes with an introduction to the concept of digitally-assisted analog circuit and its main advantages.

Chapter 2 analyzes the different non-idealities of the TI-ADC. Starting from the ideal model of an M-channel TI-ADC introduced in chapter 1, we consider the effects of offset and gain mismatches and time-skew separately providing an analytical description of each of them. This section concludes with a brief analysis of the effect of nonlinearity. The analytical models of the non-idealities are followed by a section showing the results of the MATLAB simulations, which are used to confirm the model. The second part of the chapter focuses on the calibration algorithms, listing a few important results in the literature before describing the working principle of the selected algorithms. As for the analysis of the non-idealities, MATLAB simulations results are shown to highlight the behavior of the algorithms.

Chapter 3 describes the implementation of the on-FPGA model of the TI-ADC. The chapter begins with a description of the development environment, which is the FPGA, and a brief description of its capabilities and limitations. Then it analyzes the implementation of the different blocks that compose the on-FPGA model of the TI-ADC, starting with the Numerically Controlled Oscillator (NCO), which models the behavior of a real S&H circuit. An analysis of three different implementations of the NCO is provided before selecting the most appropriate one. The third section of the chapter focuses on the implementation of the ADC block, describing in detail how the different non-idealities analyzed in chapter 2 are implemented in the on-FPGA model. Following the implementation of the ADC, we show an analysis of the Background Calibration Algorithms (BCAs), starting from the working principles described in chapter 2. This chapter concludes with a description of the auxiliary entities required for the on-FPGA model to work. These entities include a communication interface between the on-FPGA model and the rest of the

FPGA, an output buffer to store the samples, and a set of configuration registers.

Chapter 4 is divided into three sections. In the first one we provide a description of the third-party IPs implemented in the FPGA to communicate and interface with the on-FPGA model of the TI-ADC described in chapter 3. In the second section, the results of multiple simulations run using the on-FPGA model are presented, comparing them with the results of the MATLAB simulations shown in chapter 2. The last section summarizes the performances achieved by this verification technique and the advantages over classical VHDL or MATLAB simulations. The chapter concludes with a brief analysis of a few interesting future developments worth being researched.

1 | Verification of ICs

This chapter introduces the concept of verification in an Integrated Circuit (IC) design flow and Time-Interleaved Analog-To-Digital Converters (TI-ADC).

The first sections explore the current state-of-the-art for digital, analog, and mixed-signal verification methodologies before introducing the proposed method and its advantages.

In the last section, ADCs are briefly introduced before presenting the concept of Time-Interleaved converter and digitally-assisted ADC. The section concludes with a mathematical model of an ideal Time-Interleaved ADC.

1.1. Digital Verification

In this section, we introduce the general concept of verification, before exploring the state-of-the-art for digital verification.

Verification (sometimes referred to as validation) is the process of evaluating if a system, in this case, an IC, complies with the specifications [3]. It is a fundamental step of the IC design flow and its engineering cost is quickly becoming dominant due to the explosive growth in the complexity of ICs as shown in figure 1.1 [4].

In figure 1.1 we can observe the main components of the design cost of a digital IC, namely:

- Architecture: cost of the early design phases.
- Software: includes the cost of the tools and the HW infrastructure.
- Verification
- Physical: costs related to the actual manufacturing of the ICs (includes masks).
- IP qualification: the cost of validating an existing IP for integration in the new IC.

In figure 1.1 it is possible to notice how the cost of verification is quickly becoming dominant, representing up to the 50% of the design cost of modern ICs [4].

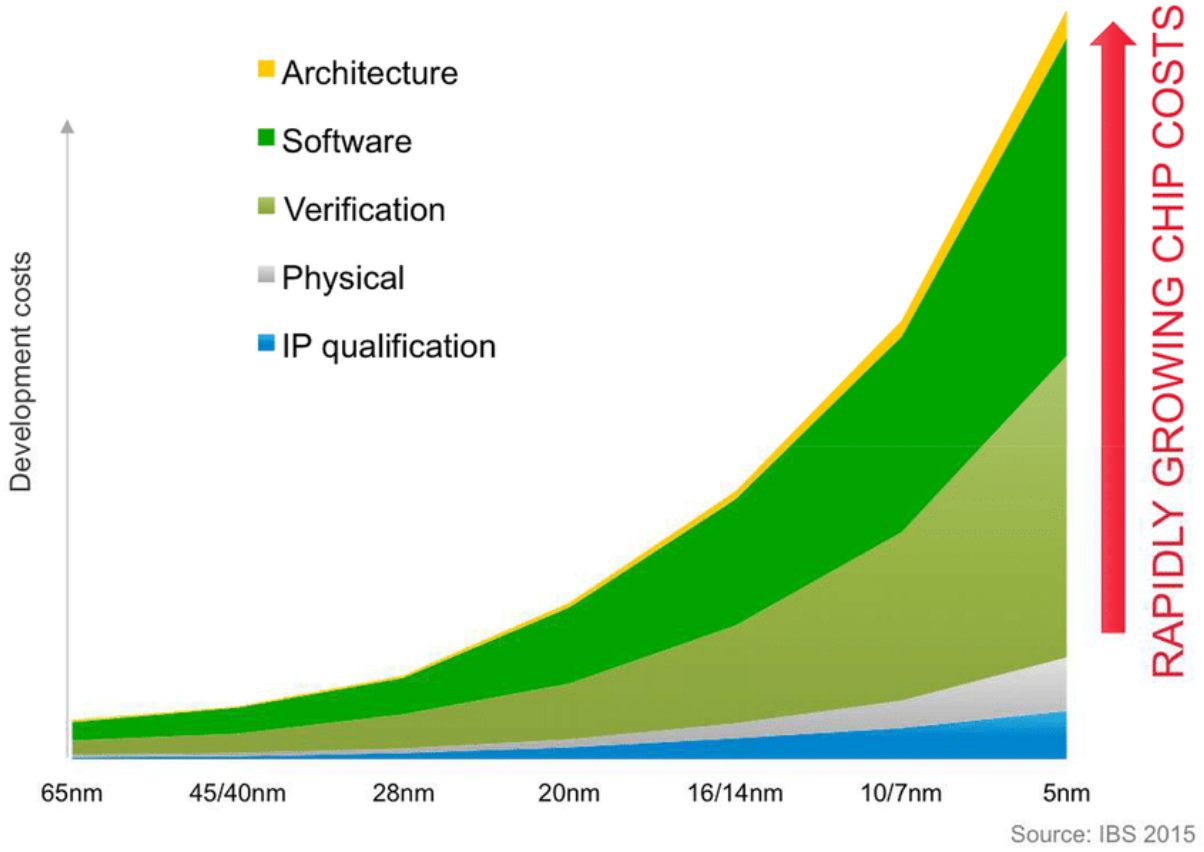

Figure 1.1: Design cost of digital ICs Vs. technological node [4].

Note that while the costs shown in figure 1.1 are referred to digital ICs, the concept of verification also applies to analog and mixed-signal ICs. The actual objectives of the verifications might be different, and the methodologies definitely are, but all kinds of complex systems (and ICs are typical examples of complex systems) require some kind of validation to ensure the manufactured product corresponds to the specifications. The more complex the system, the more advanced the verification techniques.

The importance of a correct and complete verification of the IC cannot be underestimated. Bugs undetected during verification require a new set of photolithographic masks in a so-called *re-spin*. This is a common occurrence in the semiconductor industries as less than 35% of the ASICs are able to reach a so-called *first silicon success* as reported in the largest survey of ASIC/FPGA industries [5] and shown in figure 1.2.

Verification itself includes multiple different aspects and can be divided into *pre-silicon* and *post-silicon* verification. Pre-silicon verification acts before manufacturing the ICs to validate the design, whereas post-silicon one uses manufactured ICs (for instance, early production batches) to ensure the correct behavior of the device.

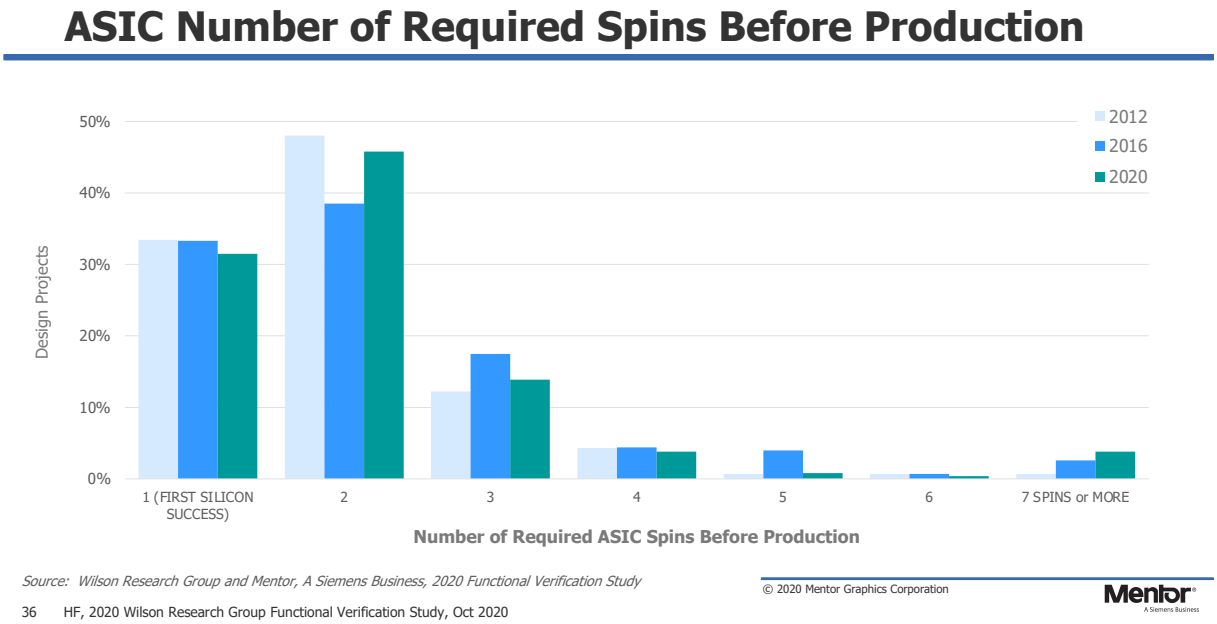

Figure 1.2: Number of *re-spins* required before production [5].

We will focus on pre-silicon, and more precisely on functional verification, which is defined as the process of ensuring that the system behaves as intended. Functional flaws cause about 50% of the *re-spins* as shown in figure 1.3.

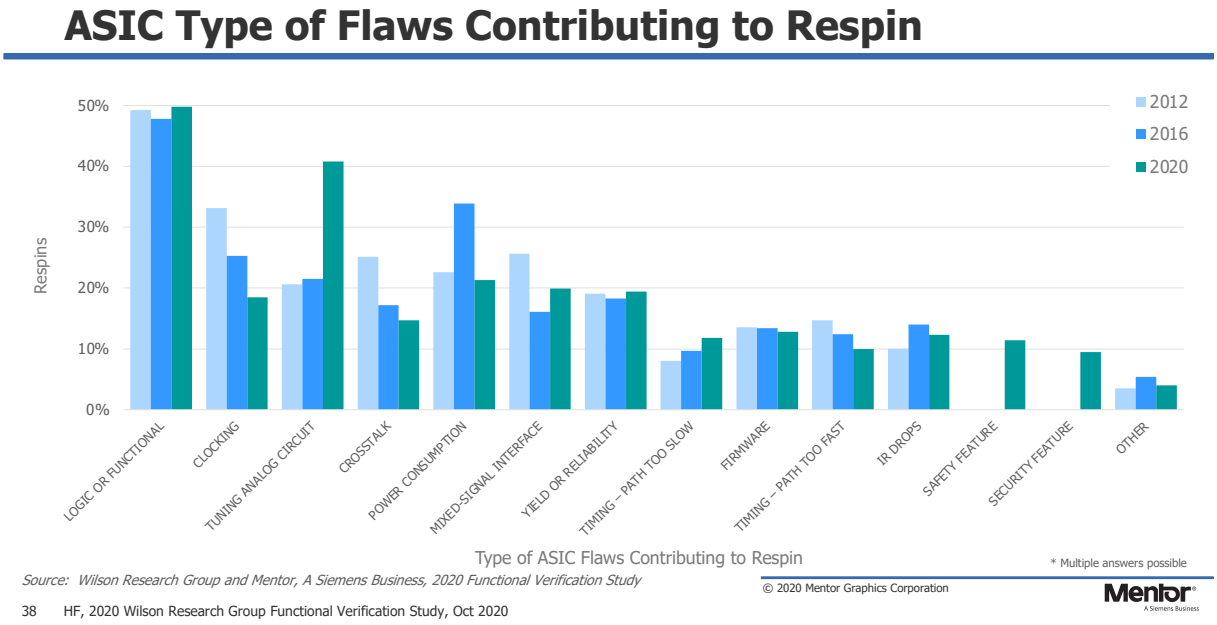

Figure 1.3: Types of flaws contributing to a *re-spins* [5].

Verification of purely digital designs is far more automated than analog verification. This is mainly a consequence of the explosive growth in the number of gates characterizing digital ICs as dictated by Moore's Law and shown in figure 1.4.

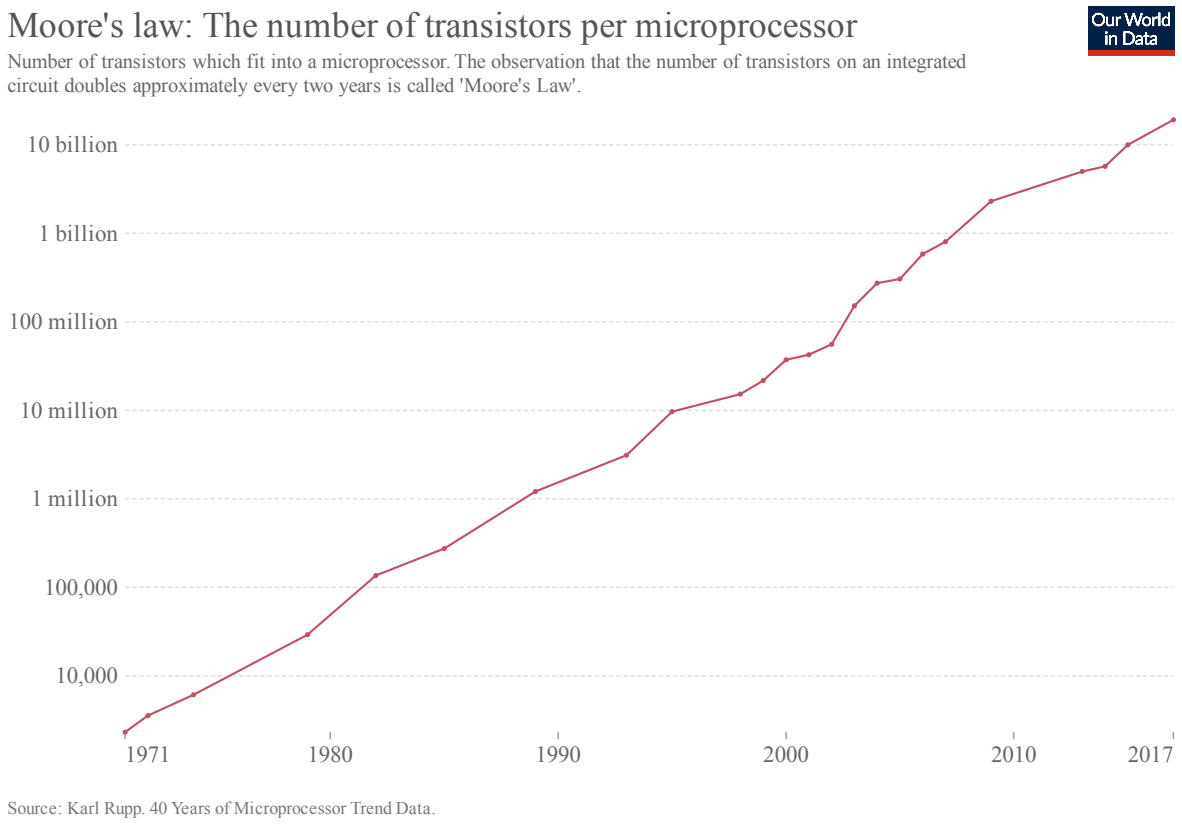

Figure 1.4: Number of transistors Vs. year (log scale) [6].

From figure 1.4 it is possible to observe that digital ICs closely follow Moore's Law, something that was made possible by the quick progress of digital Electronic Design Automation[1] (EDA) tools, used both to design and validate the ICs.

The first obstacle to the design of more complex ICs was the so-called design-productivity gap. Announced in 1997 by SEMATECH, the design-productivity gap refers to the different rates of growth of the manufacturing capability (i.e., what can be manufactured, which doubles every 18-24 months according to Moore's law), and the design productivity (i.e., what can be designed, which doubles every 40 months) as shown in figure 1.5 [7].

[1]EDA: an Electronic Design Automation tool is a CAD software used to design electronic system like ICs and PCBs

This design-productivity gap actually never manifested in industry, as the quick evolution of EDA tools, and the diffusion of design-reuse concepts allowed for the design productivity to keep up with the manufacturing capabilities.

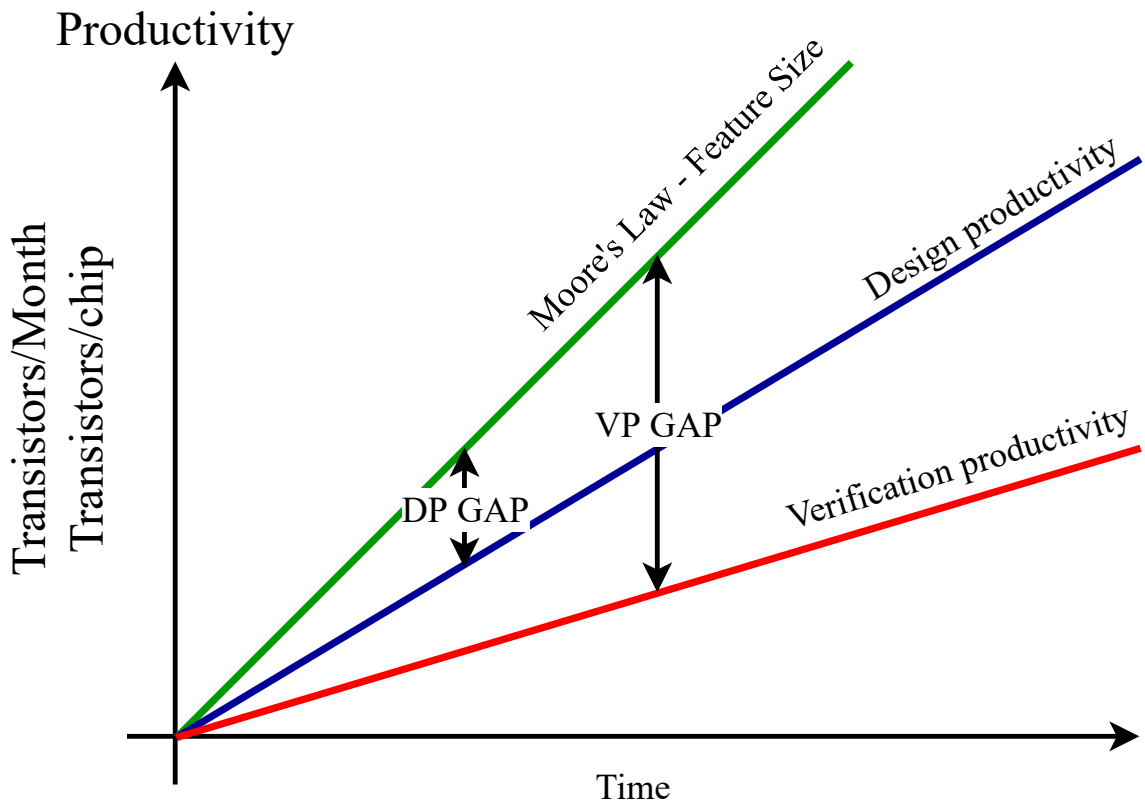

Figure 1.5: Illustration of design-productivity and verification-productivity gaps.

A second, more recent, obstacle to the design of more complex ICs is represented by the verification-productivity gap, which refers to the different rates of growth of the design productivity, and the verification productivity (i.e., what can be verified) as shown in figure 1.5 [7].

For this reason, digital verification is probably the electronic design automation sub-field that has undergone the fastest evolution in the last decades. We will list a few noteworthy steps of this evolution below.

It is worth defining three terms that will be found multiple times in the following analysis. The *verification plan* represents the properties of the design that needs to be completely and exhaustively verified (i.e., what should be verified). *(Verification) coverage* is defined as the set of tests required to satisfy the verification plan (i.e., what is verified), whereas *(verification) closure* is the fulfillment of the verification plan.

It is mandatory to begin the analysis from the origin of modern digital design, i.e., with the development of Hardware Description Languages[2] (HDL).

The first HDLs developed in the early 80's like Verilog[3] (1984) and VHDL[4] (1983) already included a subset of functionalities for the verification of the described design. Actually, the synthesizable section (that is the part of the language that describes a behavior that can be mapped to real hardware) is a subset of the whole language.

Early verification techniques used the same HDL to describe both the Device Under Test[5] (DUT) and the Test Bench[6] (TB) in a so-called direct testing. Each TB was custom-developed for each IP[7] resulting in an engineering time spent on verification with a more-than-linear proportionality both to the engineering time spent on design and to the required verification plan [7].

As the complexity of the ICs quickly grew outside the capabilities of such a basic system, the engineering time needed to manually develop the TBs for the verification coverage became impractical, and other verification methodologies were developed.

The Open Verification Methodology[8] (OVM) and Universal Verification Methodology[9] (UVM) are industry standard methodologies for the testing of digital designs. Among the main benefits of these methodologies, there is the concept of reusability, i.e., TBs are not custom-developed anymore for each different IP. A TB is now made of different modules that can be shared across different projects and combined to verify different designs with little overhead. OVM and UVM allowed for the verification coverage to keep up with the growing complexity of the test plans.

Figure 1.6 shows the adoption in the semiconductor industries of UVM/OVM methodologies (together with a few other similar standards).

[2]HDL: a Hardware Description Language is a language used to describe electronic circuits.

[3]Verilog: is an HDL developed and later made open by Cadence Design Systems, it is widely used nowadays for logic design and verification.

[4]VHDL: VHSIC Hardware Description Language is an HDL originated from VHSIC research program of the U.S. DoD, it is widely used nowadays for logic design and verification.

[5]DUT: the Device Under Test is the system subject to verification/validation

[6]TB: a Test Bench is a test environment used to validate a design

[7]IP: an Intellectual Property is a (reusable) unit of a circuit.

[8]OVM: the Open Verification Methodology is a standardized methodology and library for testing digital designs.

[9]UVM: the Universal Verification Methodology in an evolution of OVM that improves test automation

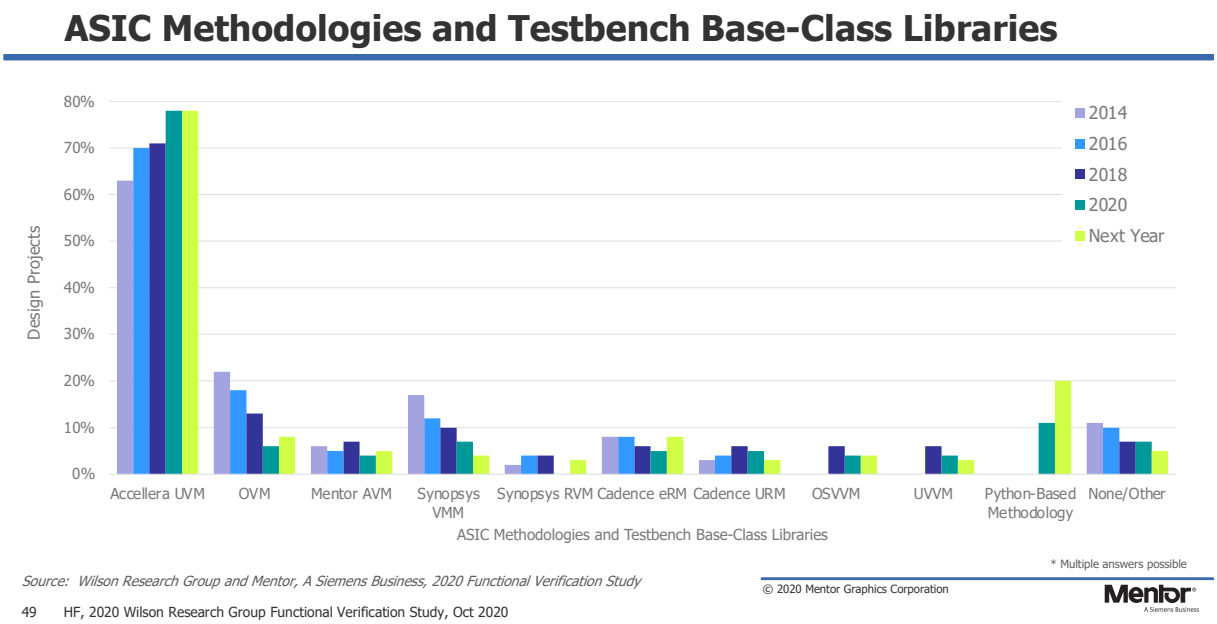

Figure 1.6: Diffusion of different verification methodologies in the semiconductor industries [5].

It is possible to notice how UVM reaches market penetrations as high as 80% [5]. This can be explained by its widespread support among the tools of the different EDA companies.

Both direct testing and OVM/UVM TBs use RTL[10] simulators to verify the correct working of a design. The main idea behind RTL simulation is to exploit the hierarchical architecture of the design. The lowest level blocks (the single standard cells for instance) are simulated at gate level, but once verified they are modelled as black boxes with *ad hoc* parameters specifying their performances (logic function, delay, etc.). Repeating this process moving up the hierarchy allows simulating large designs with reasonable times.

Though the aforementioned methodologies allow for the verification coverage to keep up with the increased complexity of the design (and thus of the verification plan), they still leave the task of defining the adequate verification plan, a task that also became impractical with the continuous growth in ICs complexity.

Formal verification is a relatively new set of techniques (introduced in the last 20 years), which has gained popularity in the last years aiming to solve this issue [8]. The idea is to use formal mathematics methods (theorem proving, induction, model checking, etc.) to prove (or disprove) the correct working of a system [9].

[10]RTL: the Register Transfer Level is an abstraction that models the dataflow of digital designs.

In standard formal verification (referred to as formal property checking in figure 1.7), the properties of the system have to be described using dedicated formal languages. Whereas in automated formal verification the properties are derived automatically from a high-level HDL like SystemVerilog[11] (SV).

Compared to standard RTL simulations, formal verification is equivalent to simulating a design in all possible ways, resulting in a 100% verification coverage.

Figure 1.7 shows the adoption in the industrial environment of formal verification [5]. It is possible to notice a 15% increase between 2012 and 2020 of the adoption of this technique.

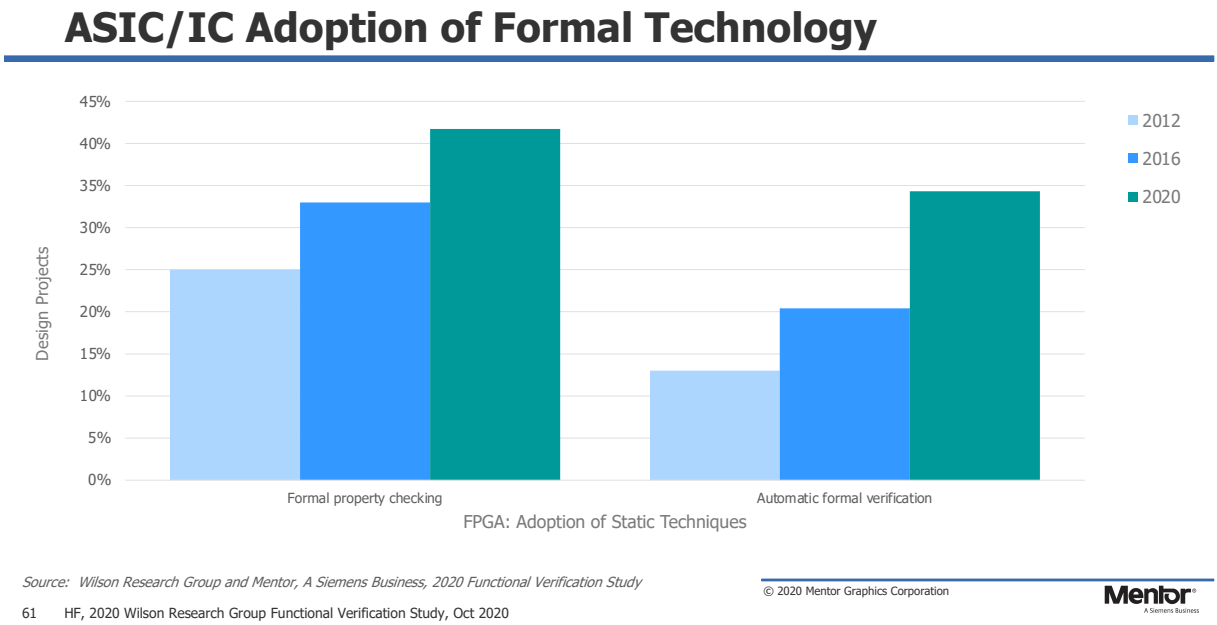

Figure 1.7: Adoption of formal verification methodologies in industries [5].

So far we have shown how verification methodologies have evolved to increase both the coverage (introducing the concept of reusability, and automation) and to ease the definition of a verification plan, reducing the engineering time required for the task.

However, the explosive growth of the complexity of the circuits still poses another issue. Even with computers getting more powerful every year, the time required to simulate, even at behavioral level, an entire IC keeps growing year after year, with similar effects on the time-to-market and design costs.

[11] SV: SystemVerilog is an evolution of Verilog (part of the same standard since 2008) focused on higher level description

A possible solution to reduce the time required for digital verification is to use specialized hardware. To be more precise, there are two different alternatives:

- Hardware (HW) emulation: Custom hardware accelerators (custom ASICs[12]) are used to execute an opportunely compiled version of the RTL code. These ASICs are highly-parallel CPU purposedly designed to accelerate RTL simulations and are able to achieve performances and efficiencies not reachable by standard CPUs.
- FPGA prototyping: The RTL code is implemented into one or more FPGAs[13] and run as actual hardware (with the necessary modifications to the code, if any). This allows verifying the design on real hardware, at close-to-real speed.

Both solutions have advantages and disadvantages. Hardware emulation is usually slower than FPGA prototyping but allows full exposure of the design (all the signals are visible to the user). Furthermore, once the emulators are set up, it is possible to modify the simulation (for instance, to change the RTL code after a bug is found) with a little time penalty.

FPGA prototyping, on the other hand, is faster, but only selected and predetermined signals are available externally. Furthermore, FPGA prototyping usually requires manual modifications to the original RTL code (targeting an ASIC) to adapt it to an FPGA.

Method	Performances $[instruction/s]$
C model	2 M/s
RTL simulation	200/s
Cadence Palladium (HW emulation)	700 K/s
Cadence Protium (FPGA prototyping)	4 M/s
Custom FPGA prototype	42 M/s

Table 1.1: Performances of different simulations methods [10].

Table 1.1 shows an example provided by Cadence Design System of the performances achieved by one of their customers with RTL simulations, HW emulation, and FPGA prototyping. It also lists a C model and a custom-developed FPGA prototype. The simulations/emulations shown in table 1.1 refer to a CPU, and the column labeled "performances" refers to the number of software instructions the simulations/emulations are able to execute per second.

[12]ASIC: an Application Specific IC is an IC custom designed for a particular task
[13]FPGA: a Field Programmable Gate Array is an IC designed to be reconfigured after manufacturing

The difference between Cadence Protium and the custom FPGA prototype is that the latter requires significant modifications to the original RTL, whereas Cadence Protium is highly automated in this aspect.

Usually, HW emulation is used to test single IP, whereas FPGA prototyping is used to verify whole systems (i.e., composed of multiple IPs). This is both due to the aforementioned limitation in the number of signals the user has access to, but also due to the constraint on the number of I/O available on the FPGA. HW emulation and FPGA prototyping are, respectively, 3500 and 20000 times faster than an RTL simulation, and approach the speed of the real hardware.

This allows both a reduction of the time-to-market and the possibility to verify the system for large time intervals, detecting any long-term bug, which might not be discovered in a shorter simulation run on a computer. Furthermore, the simulation/emulation speed close to the one of the real hardware allows for the development and testing of drivers and microcode before the fabrication of the first ICs.

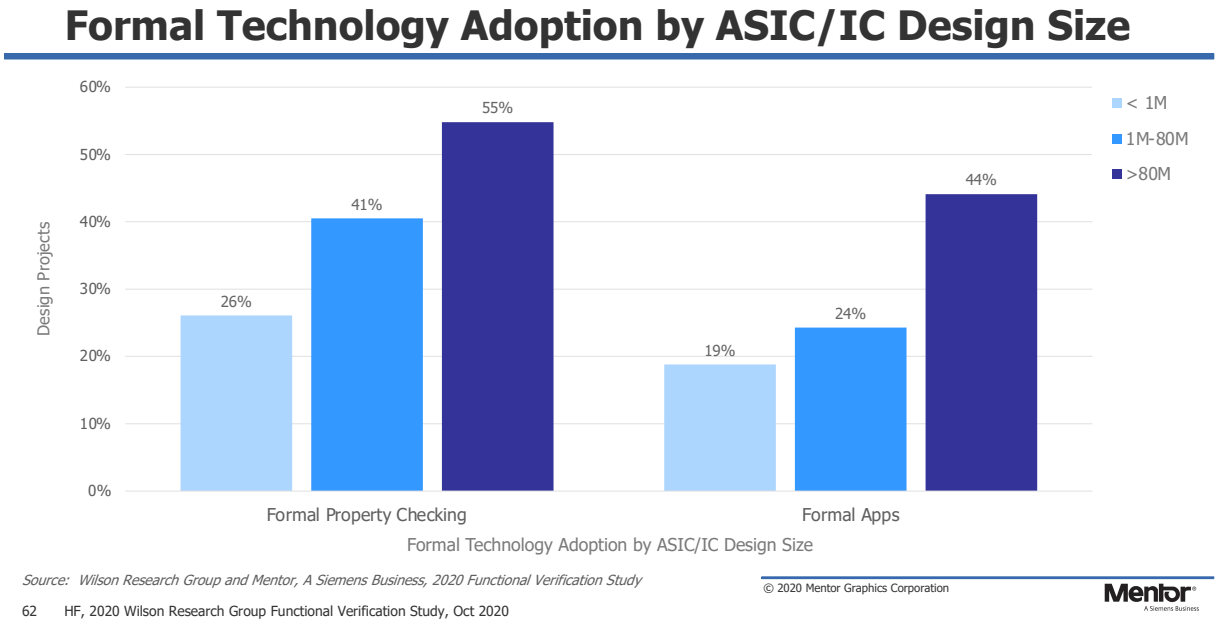

Figure 1.8: Adoption of hardware emulation and FPGA prototyping for different design sizes (measured in M of gates) in the industries [5].

Figure 1.8 shows the adoption of HW emulation and FPGA prototyping in semiconductor industries for different design sizes (measured in millions of gates). It is possible to notice that HW emulation is preferred to FPGA prototyping for larger designs [5].

This can be explained by the engineering efforts required to modify the larger design to implement the system in an FPGA, not to mention that a single FPGA might not have enough resources for such a system (it is possible to use multiple FPGAs connected together to prototype a design, but this introduces further, and more intensive, modifications to the original RTL code). HW emulators, on the other hand, only suffer a performance penalty for larger designs.

These kinds of tool are extremely powerful to verify digital systems. In fact, all three major semiconductor EDA companies provide similar systems: Cadence Design Systems with the aforementioned Palladium (HW emulation) and Protium (FPGA prototyping), Synopsys with ZeBu (HW emulation) and HAPS (FPGA prototyping), and Siemens EDA with the Veloce Strato (HW emulation) and Veloce Prototyping (FPGA prototyping) platforms.

Table 1.2 summarizes the simulation/emulation methodologies for digital ICs analyzed in this sub-section and their advantages/disadvantages. RTL (or HDL) simulation includes both the *ad hoc* TBs and the ones using UVM/OVM.

	HDL simulation	HW emulation	FPGA prototyping
Speed	low	medium-high	high
Visibility	full	high	low
Cost	low	high	medium-low
Automation	full (with UVM/OVM)	high	medium
Design cycle	immediate	minutes/hours	hours/days

Table 1.2: Summary of the different digital simulation and emulation methodologies.

The design cycle refers to the time required to begin a new verification cycle after the RTL code has been modified. HDL simulators and HW emulators require only a new compilation of the source RTLs (HW emulation requires a more complex compilation). FPGA prototyping, on the other hand, requires a new synthesis and implementation.

Formal verification is not listed in table 1.2 because it is fundamentally different in its principle. It is not based on a simulation/emulation of the design, but on demonstrating its equivalence to the design specifications.

1.2. Analog Verification

This sub-section briefly analyzes the different requirements of analog verification with respect to the digital one mentioned before.

Verification of analog circuits is radically different from the verification of digital designs described before due to the different levels of detail required [11]. HDL simulators are based on hierarchical abstraction to reduce the simulation times. This concept is not applicable to analog circuits due to the complex and interdependent behavior of this kind of circuits.

Analog simulators are usually transistor-level simulators like SPICE[14] and its derivatives. The single devices are described by extensive models that can also take into account the process corners, temperature, variations in the power supplies, and so on.

Anyhow, at least until a few years ago, verification of analog blocks was a manual task. Usually, the designers themselves were in charge of verifying the correct functioning of the block they have designed. Nowadays, the complexity of analog blocks is approaching the limits of what it is possible to verify manually (like digital blocks did decades ago), and many methodologies are being adapted from digital to analog and mixed-signal verification.

Note that the minor degree of automation found in analog verification is not only related to the smaller density of analog ICs with respect to digital ones. Another, perhaps more important obstacle to automated analog verification lies in the difficulty of describing clearly and unequivocally an analog circuit, something that was solved decades ago for digital circuits with the use of HDLs.

1.3. Mixed-signal Verification

Large digital design containing tightly integrated analog sections like, for instance, an RF modem with integrated front-ends or a System on a Chip[15] (SoC) with embedded radios (known as RF-SoC) are quite difficult to verify [12–14].

Figure 1.9 shows the evolution in the complexity of ICs to combine large digital sections with analog functions [15].

[14]SPICE: Simulation Program with Integrated Circuit Emphasis is the first open-source analog circuit simulator

[15]SoC: a System on a Chip is an electronic system contained in a single IC.

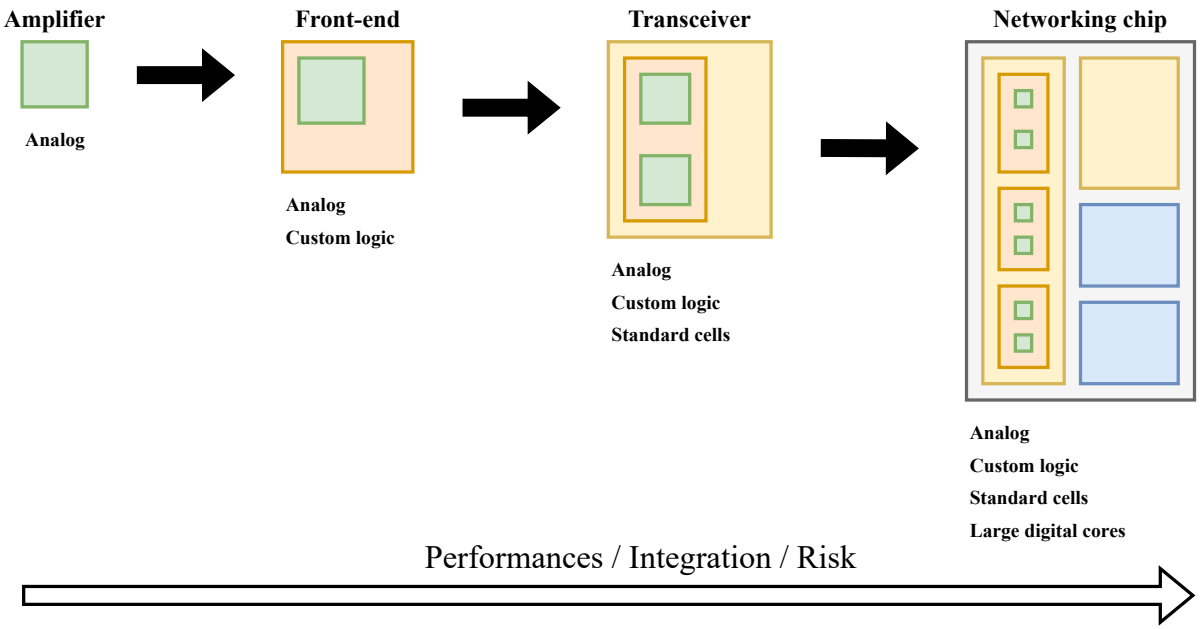

Figure 1.9: Trend to combine analog and digital sections in a single IC to implement complex functions.

It is possible to verify the single components separately in the respective environment, but, though this is reasonable in loosely coupled systems like a micro-controller with an analog peripheral like an ADC or a DAC, it results completely inadequate to tightly coupled systems like an integrated RF front-end. In the latter case, in fact, the correct working of the single parts does not imply necessarily the correct functioning of the whole system [16].

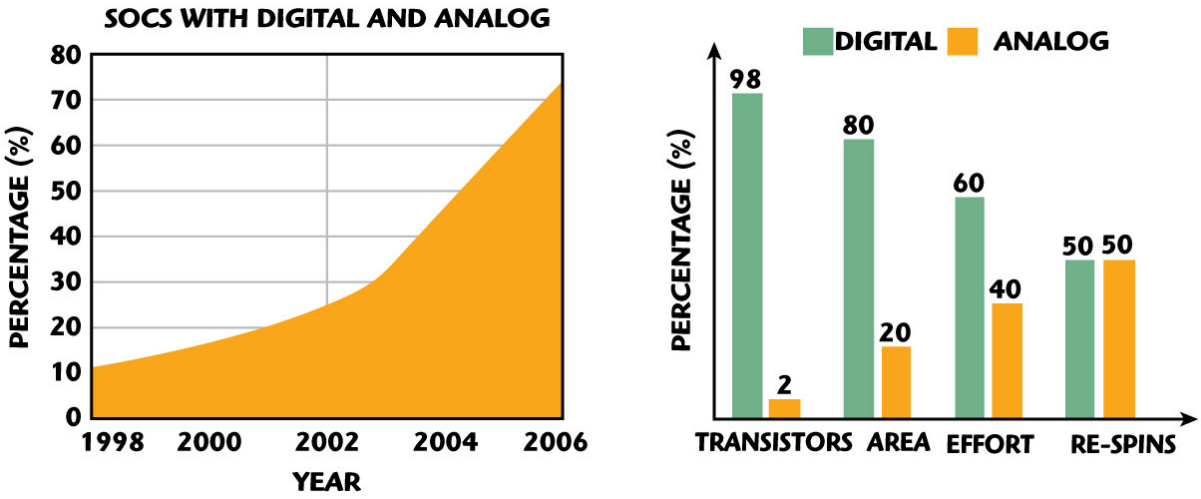

Figure 1.10: Percentage of SoC containing analog components (left), and influences of the digital and analog section on different design aspects (right) [15].

Figure 1.10 shows the percentage of SoC containing both analog and digital sections (left), and the influence that the two domains have on different design aspects (right) [15]. *Re-spins* refer to flaws that require a new set of masks to be corrected. *Re-spins* are extremely expensive and time-consuming, and, therefore, must be avoided at all costs.

Verifying complex mixed-signal SoC poses serious challenges due to the different issues and specifications the two domains need to satisfy.

From one point of view simulations of digital designs are more difficult due to the larger number of transistors. Still in digital circuits, we are not interested in the actual voltage, but only in the logic level. This is exploited by the simulators to simplify the analysis and reduce the simulation times, together with the event-driven nature of a clocked system, and with the hierarchical abstraction mentioned before.

On the other hand, analog designs might be made up of a smaller number of elements, but the elements need to be characterized orders of magnitude more accurately than in digital simulations to provide realistic results, thus increasing the simulation times. This higher level of detail makes simulating an entire SoC impossible.

In the last years, mixed-mode (mixing transistor and behavioral HDL) and mixed-level (mixing model and transistors) simulators started appearing to speed-up simulation times, but they achieve this by trading accuracy for speed as shown in figure 1.11 [13].

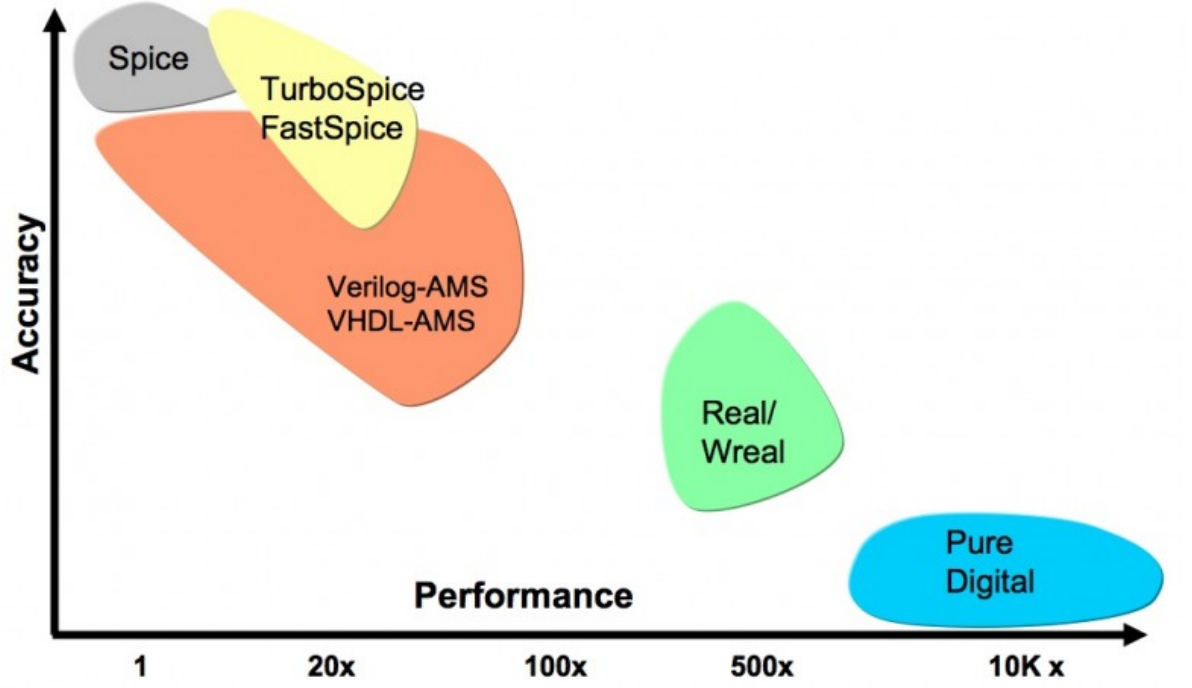

Figure 1.11: Accuracy-speed trade-off of circuit simulators [17].

For example: **1)** HDLs have been extended to include analog and mixed-signal capabilities (VHDL with VHDL-AMS and Verilog with Verilog-A and Verilog-AMS). These extensions allow defining the behavior of mixed-signal blocks. **2)** Verification methodologies like UVM received the same treatment with UVM-AMS that allows the automated testing of analog blocks [18], even if only as a complement to the standard analog verification of the circuit due to the limitations on its reliability [19].

Note that the aim of mixed-signal verification is to verify the correct functioning of the whole complex system. Analog and digital sections are still tested separately using their dedicated techniques, but, as mentioned before, this is not sufficient to guarantee a functioning system when combined together.

Note that simulation times of complex mixed-signal ICs can reach days/weeks when using behavioral description and weeks/months for block levels simulations depending on the complexity of the IC. For this reason, given the large and continuously growing size of nowadays SoC, there are some attempts to exploit digital techniques like FPGA prototyping to validate mixed-signal systems. Some are based on the use of a CPU integrated in an FPGA to run the simulations of the analog section of the system [20], others make use of an FPGA implementable model of the analog sections [21–23].

1.4. Proposed Verification Method

The verification methodology proposed in this *thesis* consists of using an FPGA implementable model of the analog circuit to prototype its digital logic section. This allows the validation of the digital logic with a throughput comparable to the one of the real system [21–23]. The main advantages of this technique are:

- **Arbitrarily long run times**: this methodology allows running the prototype for an arbitrary amount of time with throughput close to the real hardware.
- **Testing the design on real hardware**: using real hardware to verify the design might reveal implementation bugs that cannot occur in computer simulations.

The first point is probably the greatest advantage because it allows checking long-term behaviors like, for instance, ensuring that the BER[16] of a communication interface satisfies the requirements, or even revealing long-term bugs like the saturation of accumulators, or error accumulation. The design runs on a single FPGA instead of a workstation or a server. Thus, this method is also more efficient from a power consumption standpoint.

[16]BER: the Bit Error Rate is the number of bits per unit time that have been detected incorrectly

The second point is also extremely important. Using real hardware means that we don't need to purposefully model the effects of the hardware (i.e., finite precision, unknown initialization state, etc.).

Furthermore, if the selected FPGA has enough resources to implement the design (and this is the case, as the number of gates in FPGAs is subject to the same rate of growth that characterizes ICs due to Moore's law), we will have that the throughput, meant as emulation speed, is independent of the complexity of the design.

Note that this method allows for the verification of digital design containing also analog parts. The target is still the verification of the digital part only, while representing the analog block with adequate realism, and is, therefore, useful to validate whole systems.

1.5. Analog-to-Digital Converters

The verification method proposed in the previous section has been implemented targeting the verification of Analog-to-Digital Converters (ADCs).

This type of circuit was selected as a demonstrator because it is a typical example of a mixed-signal circuit. To be more precise, we selected a 10-bit 8-channel 2-GS/s Time-Interleaved ADC for $5G$ wireless applications, which makes extensive use of digital calibration algorithms to correct some non-idealities and improve its performances.

ADCs are very common mixed-signal systems. They are everywhere, (transceivers, microphones, cameras, radios, etc.) and it is not an exaggeration to say that without ADCs we would not have much of the technology we have now.

An ADC is an electronic system that converts an analog and time-continuous input signal into a digital one. In the process the signal is also sampled, i.e., it becomes a time-discrete signal.

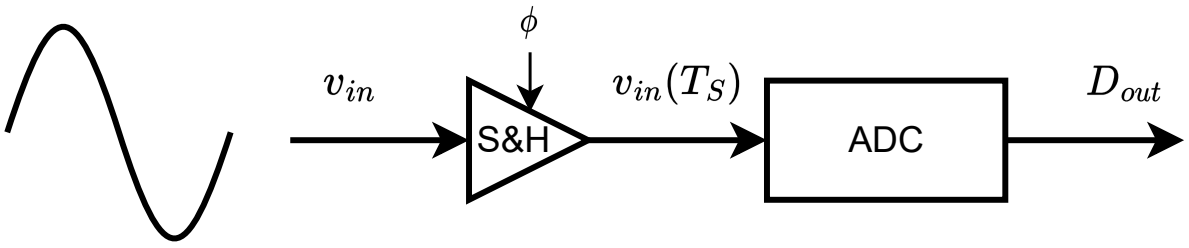

Figure 1.12: Block diagram of an ADC.

Figure 1.12 shows a block diagram of an ADC. Typically, an ADC is preceded by a sampler circuit (S&H) controlled by the clock signal ϕ. Its role is to sample the input signal ($\phi = 1$) and hold it for the duration of the conversion ($\phi = 0$).

1.5.1. Time-Interleaved ADCs

In this *thesis*, we focus on an FPGA implementable model of a particular type of ADC known as Time-Interleaved ADC.

The first occurrence in the literature of TI converters is in 1980 in a paper written by W.C. Black and D.A. Hodges [1]. The advantages of this type of converter are presented by B. Razavi in [24] together with the upper bound of the maximum achievable performances. Buchwald in [25] provides a similar analysis while focusing on the different design opportunities of TI converters.

A TI-ADC is built by placing multiple sub-converters (cores, channels, or slices) in parallel. Each channel is driven by different phases of the same clock signal, setting a different sampling instant for each of the cores.

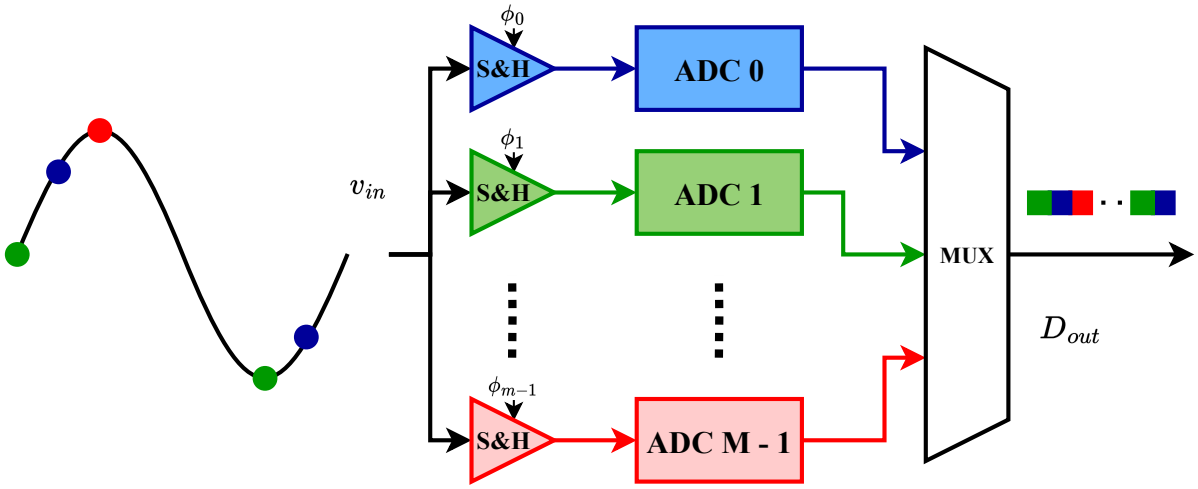

Figure 1.13: Block diagram of an ideal M-channel TI-ADC.

Figure 1.13 shows a block diagram of an ideal M-channel TI-ADC. Each channel is clocked with a different phase ϕ_m (with $m = 0, 1, ..., M-1$ index of the channel) as shown in figure 1.14.

This effectively results in a total sampling frequency $f_{S,TI}$ equal to $M \cdot f_{S,CORE}$, where M and $f_{S,CORE}$ are the interleaving factor (i.e., the number of cores), and the sampling frequency of the single core, respectively.

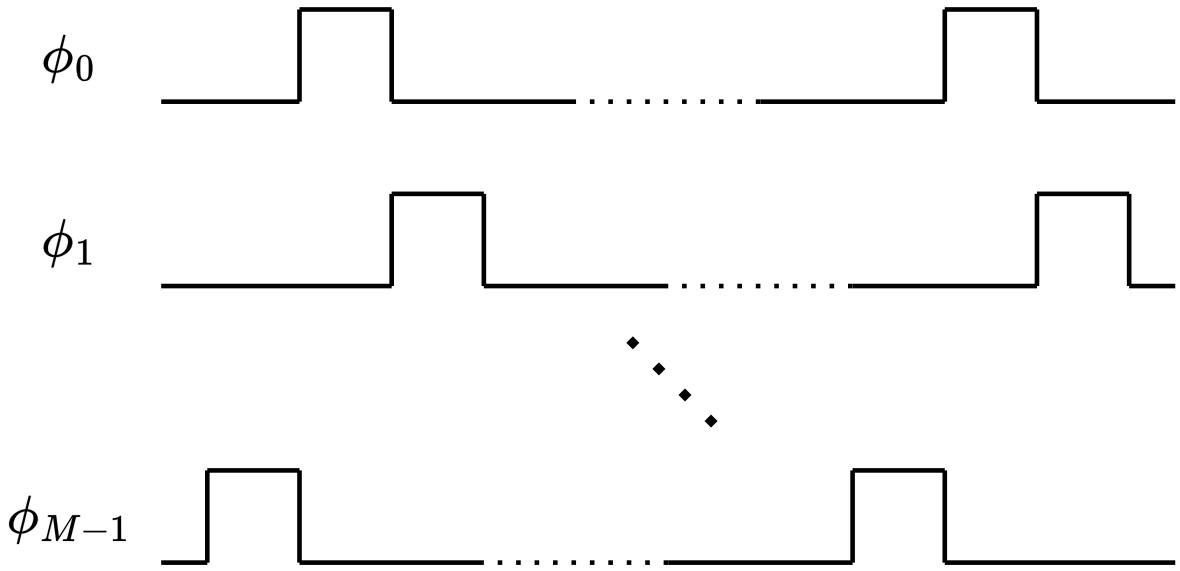

Figure 1.14: Clock phases of an ideal M-channel TI-ADC.

Figure 1.14 shows the clock phases of an ideal M-channel TI-ADC.

The main advantage of time interleaving lies in the reduced performances required for each sub-converter. In fact, for the same sampling frequency f_S, a TI-ADC with an interleaving factor M requires a converter M times slower.

This was exploited in 1992 [26, 27] for the first time to implement ADCs with high sample rate in standard CMOS[17] process instead than a more exotic one. The possibility of implementing a high-speed converter with a standard CMOS process must not be underestimated. In fact, the compatibility with a CMOS process allows the implementation of the analog components (i.e., the ADC in this case) and the digital ones in the same die, greatly reducing the energy lost in the inter-IC interconnections [28, 29]. This shift from a System on a Module[18] (SoM)/ System on a Package[19] (SoP) approach to an SoC one (as shown in figure 1.9) has its main benefits in the smaller form-factor, higher power efficiency and lower costs [30].

The use of standard CMOS processes also has the advantage to allow analog circuits benefitting from the technological scaling of the fabrication processes pushed by digital applications.

[17]CMOS: the Complementary Metal Oxide Semiconductor is a family of fabrication processes widely used nowadays

[18]SoM: a System on a Module is an electronic system (i.e., multiple ICs and passive components) contained within a Printed Circuit Board

[19]SoP: a System on a Package is an electronic system (i.e., multiple ICs and passive components) contained within a chip carrier (i.e., a single physical package)

Though analog circuits do not benefit from scaling, they still benefit from the improvement in the manufacturing process (better control, better uniformity, etc) that are a consequence of the technology scaling.

Another, perhaps more interesting, advantage of TI-ADCs is the additional degree of freedom in the design space of the converter to optimize the power-speed trade-off as shown in [31, 32]. Traditionally ADCs have a non-linear relationship between efficiency and conversion speed. Beyond a certain sampling frequency, it is more efficient to increase the interleaving factor instead of pushing the single core at a larger conversion rate [24].

1.5.2. Model of an Ideal TI-ADC

From here on, the sampling time of the interleaved converter $T_{S,TI}$ will be referred to as T_S for the sake of simplicity. Likewise, the sampling frequency of the whole converter will be referred to as f_S.

It is possible to express the output signal of each sub-converter as the product of the input signal $x(t)$ with an opportunely time-shifted Dirac comb (sampling function) with steps equal to $M \cdot T_S$:

$$y_m(t) = \sum_{n=-\infty}^{\infty} x(t) \cdot \delta\left(t - (nM + m) \cdot T_S\right), \tag{1.1}$$

where $m = 0, 1, ..., M-1$ represents the index of the channel.

The time domain expression of the sampled signal of all the sub-converters is given by the summation of all the outputs of the cores, thus:

$$y(t) = \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} x(t) \cdot \delta\left(t - (nM + m) \cdot T_S\right). \tag{1.2}$$

Reordering equation 1.2, we get:

$$\begin{aligned} y(t) &= x(t) \sum_{n=-\infty}^{\infty} \sum_{m=0}^{M-1} \delta\left(t - (nM + m) \cdot T_S\right) \\ &= x(t) \sum_{n=-\infty}^{\infty} \delta\left(t - nT_S\right). \end{aligned} \tag{1.3}$$

The expression of the sampled signal for an ideal M-channel TI-ADC, with each sub-converter sampling at $f_S/M = 1/MT_S$, is the same of a monolithic ADC sampling at $f_S = 1/T_S$.

Moving into the frequency domain, equation 1.3 becomes:

$$\begin{aligned} Y(f) &= \mathscr{F}\left[x(t)\sum_{n=-\infty}^{\infty}\delta\left(t-nT_S\right)\right] \\ &= X(f) * \mathscr{F}\left[\sum_{n=-\infty}^{\infty}\delta\left(t-nT_S\right)\right]. \end{aligned} \tag{1.4}$$

The Fourier transform of a Dirac comb is a Dirac comb, thus, equation 1.4 becomes:

$$\begin{aligned} Y(f) &= X(f) * \frac{1}{T_S}\sum_{n=-\infty}^{\infty}\delta\left(f-\frac{n}{T_S}\right) \\ &= \frac{1}{T_S}\sum_{n=-\infty}^{\infty}X\left(f-\frac{n}{T_S}\right). \end{aligned} \tag{1.5}$$

Equation 1.5 expresses the well-known output spectrum of a sampled signal, showing replicas of the input spectrum at $\pm nf_S$.

In our case we are interested only in what falls within the Nyquist band, which is (in a unilateral spectrum) $0 \leq f < f_S/2$.

1.5.3. Digitally-Assisted ADCs

The compatibility with standard CMOS processes also opened another frontier of the IC design, which is the concept of digitally-assisted analog circuits (in this particular case digitally-assisted converter). Unlike digital circuits, analog ones are usually constrained by noise, matching, and linearity requirements. The idea is to relax these constraints where possible (usually the matching and linearity ones) and correct the errors introduced with a post-processing in the digital domain with multiple benefits [2, 33, 34]:

First of all, it greatly simplifies the analog section of the circuit. For example, it is possible to replace complex high-gain amplifiers with negative feedback loops (usually required to ensure high linearity) with simpler single-stage amplifiers without feedback. Relaxing the requirement in the matching also reduces the area occupation of the circuit (matching error is inversely proportional to the square root of the component area). In both cases, the errors can be corrected by a downstream calibration logic.

Another advantage of digitally assisted designs is a consequence of the first point. We are trading the analog complexity for the digital one, but digital logic benefits much more than analog circuits from technological scaling.

This digitally assisted design method is used in the TI-ADC selected as the target of this *thesis* to correct some errors due to the time interleaving that will be analyzed in the next chapter. The digital assisting logic of the TI-ADC is the exact target of the verification methodology proposed in this chapter.

2 | Non-Idealities of TI-ADC and Calibration Algorithms

This chapter shows a mathematical analysis of the non-idealities of the TI-ADC and the calibration algorithms used to correct them.

In the first section, the ideal model presented in the previous chapter is extended to take into account different sources of error separately, before showing the MATLAB simulations of analyzed the non-idealities.

In the second section calibration algorithms are presented and analyzed for two of the aforementioned non-idealities, namely, offset and gain mismatch.

The chapter concludes with MATLAB simulations of the aforementioned background calibration algorithms.

2.1. Non idealities

In the previous chapter, an ideal TI-ADC was described. However, real devices are characterized by non-idealities that limit the performances of the converter.

The importance of those non-idealities is well-known in literature since the early works on the topic by W.C. Black and D.A. Hodges [1]. Y.C. Jenq in [35] provides a theoretical analysis of the effect of a non-uniform sampling of sinusoidal signals. Starting from those results, [36] introduced the effects of channel mismatches.

A general analysis of the limitations of Time-Interleaved ADCs is provided by Buchwald in [37]. Explicit formulas for the effects of the offset, skew, and timing mismatches are presented in different works [36, 38–42]. [43] provides a detailed analysis of the effects of the time-skew, whereas [44] presents explicit formulas for the effects of combined mismatches.

C. Vogel in [45] provides a closed-form expression of the SNDR of a M-channel TI-ADC affected by channel mismatch while laying out a framework for the analysis of TI-ADC. In [46] C. Vogel and G. Kubin analyze the effect of nonlinearity on a TI-ADC.

In this *thesis* the following non-idealities are taken into account:

- Offset mismatch
- Gain mismatch
- Skew error
- Nonlinearity of the core

Each non-ideality is analyzed separately in the following subsections, deriving an analytical model of the output spectrum where possible.

In the following section, an algorithm for the calibration of the first two mismatches (offset and gain) will be analyzed. The implementation of the aforementioned algorithms into an FPGA is detailed in the next chapter.

2.1.1. Offset Error

The offset error of an ADC is defined as the difference between the first ideal code transition and the first actual code transition [47] as shown in figure 2.1.

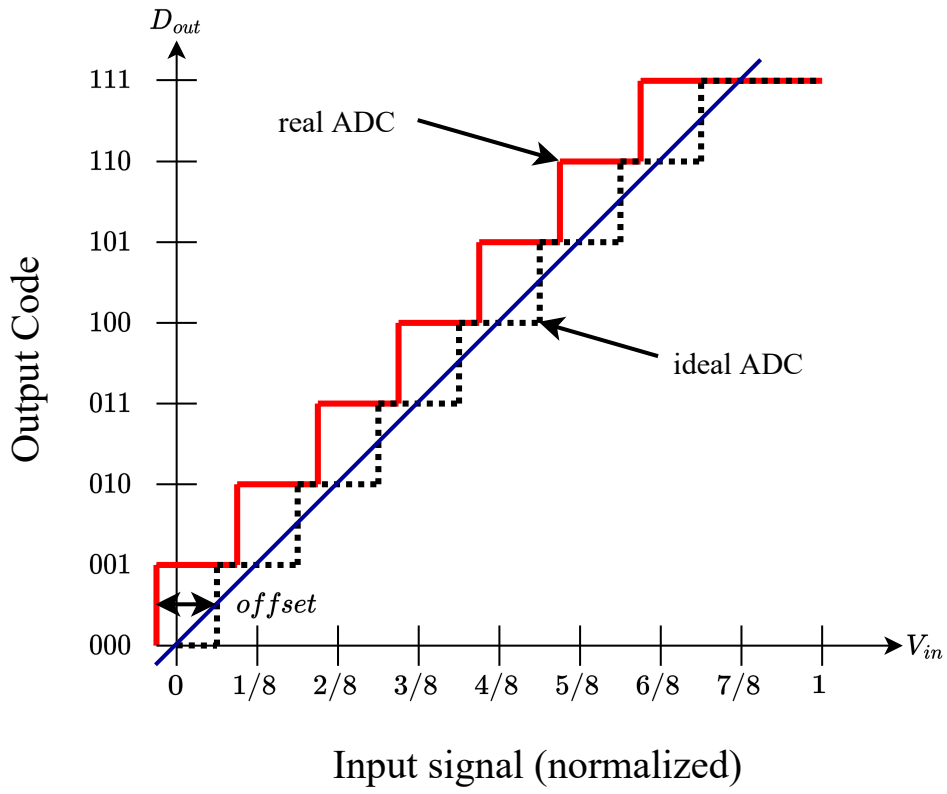

Figure 2.1: Example of ADC characteristic affected by offset error.

Each core has its own DC offset. The actual sources of the offset depend on the architecture of the converter but they are usually (and this is the case for SAR ADCs) related to the comparator [48].

In a SAR ADC where there is a single comparator, the offset can be modelled as a global offset. In principle, in a single SAR ADC, this is not an issue, but in a TI-ADC, because of the periodic alternation of the channels, the different offsets of the cores determine a fixed pattern noise, lowering the SNDR and SFDR of the converter [24, 36, 40].

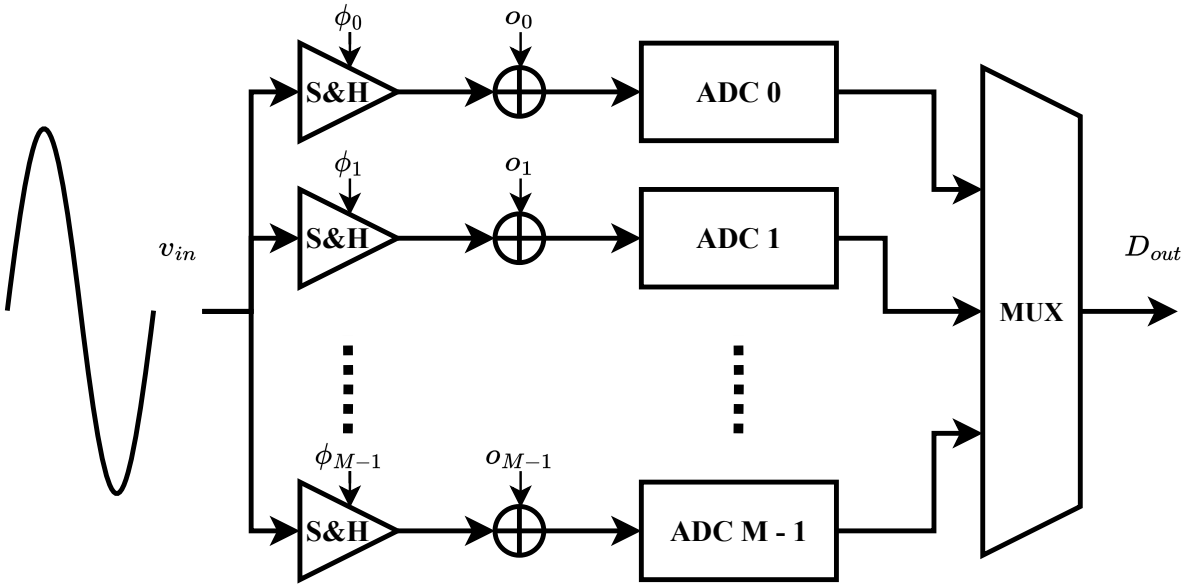

Figure 2.2: Block diagram of a M-channel TI-ADC affected by offset mismatch.

Figure 2.2 shows the block diagram of a M-channel TI-ADC with offset mismatch. Each channel is characterized by a different offset o_m.

It is possible to intuitively explain the effect of the offset mismatch considering a 2-channel TI-ADC. Let's suppose the input signal is zero (i.e., $x(t) = 0$). In this condition, the output signal of each channel is the DC offset of the channel. The continuous alternation of the channels will result in harmonic at $f_S/2$ with an amplitude proportional to the difference between the two DC offsets. In fact, the overall converter output signal results:

$$y(t) = ... + o_1 \cdot \delta(t-0) + o_2 \cdot \delta(t-T_S) + o_1 \cdot \delta(t-2\cdot T_S) + o_2 \cdot \delta(t-3\cdot T_S) + ..., \quad (2.1)$$

where o_1 and o_2 represent the offset of the first and second channel, respectively.

Let's now provide a more rigorous analysis of the output spectrum due to an offset considering a generic M-channel TI-ADC like the one represented in figure 2.2. We can model the effect of the offset mismatch by adding a core offset o_m, different for each channel, to the input signal:

$$\hat{x}_m(t) = x(t) + o_m \,. \tag{2.2}$$

Note that the use of the $\hat{}$ symbolizes a signal affected by a non-ideality, in this case offset.

The sampled signal of the m^{th} channel can be written as the convolution of the input signal $\hat{x}_m(t)$ with a sampling function as expressed in equation 1.1 and reported below for the sake of simplicity:

$$\hat{y}_m(t) = \hat{x}_m(t) \cdot \sum_{n=-\infty}^{\infty} \delta\left(t - (nM + m) \cdot T_S\right). \tag{2.3}$$

Note that in this case, the signal convoluted with the sampling function is different for each channel.

As done for the ideal model, it is possible to reconstruct the complete output signal (of the interleaved ADC) by adding the output of each channel:

$$\hat{y}(t) = \sum_{m=0}^{M-1} \hat{y}_m(t) = \sum_{m=0}^{M-1} \hat{x}_m(t) \sum_{n=-\infty}^{\infty} \delta\left(t - (nM + m) \cdot T_S\right). \tag{2.4}$$

It is possible to split equation 2.4 in two parts as:

$$\hat{y}(t) = \sum_{m=0}^{M-1} x(t) \sum_{n=-\infty}^{\infty} \delta\left(t - (nM + m) \cdot T_S\right) + \sum_{m=0}^{M-1} o_m \sum_{n=-\infty}^{\infty} \delta\left(t - (nM + m) \cdot T_S\right). \tag{2.5}$$

Reordering equation 2.5, we get:

$$\hat{y}(t) = x(t) \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} \delta\left(t - (nM + m) \cdot T_S\right) + \sum_{m=0}^{M-1} o_m \sum_{n=-\infty}^{\infty} \delta\left(t - (nM + m) \cdot T_S\right) \tag{2.6}$$

$$\hat{y}(t) = x(t) \sum_{n=-\infty}^{\infty} \delta\left(t - nT_S\right) + \sum_{m=0}^{M-1} o_m \sum_{n=-\infty}^{\infty} \delta\left(t - (nM+m)\cdot T_S\right). \tag{2.7}$$

The first part of equation 2.7 represents the ideal sampled signal (without offset mismatch), thus:

$$\hat{y}(t) = y(t) + \sum_{m=0}^{M-1} o_m \sum_{n=-\infty}^{\infty} \delta\left(t - (nM+m)\cdot T_S\right). \tag{2.8}$$

In the frequency domain we have:

$$\begin{aligned} \hat{Y}(f) &= Y(f) + \mathscr{F}\left[\sum_{m=0}^{M-1} o_m \sum_{n=-\infty}^{\infty} \delta\left(t - (nM+m)\cdot T_S\right)\right] \\ &= Y(f) + \sum_{m=0}^{M-1} o_m \cdot \mathscr{F}\left[\sum_{n=-\infty}^{\infty} \delta\left(t - (nM+m)\cdot T_S\right)\right]. \end{aligned} \tag{2.9}$$

Recalling, as mentioned in chapter 1, that the Fourier transform of a Dirac comb is a Dirac comb, it results:

$$\hat{Y}(f) = Y(f) + \sum_{m=0}^{M-1} o_m \cdot \frac{2\pi}{MT_S} \cdot e^{-j2\pi f m T_S} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{MT_S}\right). \tag{2.10}$$

Reordering equation 2.10, we get:

$$\hat{Y}(f) = Y(f) + \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} \left[\sum_{m=0}^{M-1} o_m \cdot e^{-j2\pi f m T_S}\right] \delta\left(f - \frac{n}{MT_S}\right). \tag{2.11}$$

The part within the square brackets is a Discrete Fourier Transform (DFT):

$$O_n = \sum_{m=0}^{M-1} o_m \cdot e^{-j2\pi n m/M}. \tag{2.12}$$

Substituting equation 2.12 into equation 2.11, it results:

$$\hat{Y}(f) = Y(f) + \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} O_n \cdot \delta\left(f - \frac{n}{MT_S}\right). \tag{2.13}$$

From equation 2.13 we can highlight the output spectrum due to the offset only, which is:

$$E_{offset}(f) = \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} O_n \cdot \delta\left(f - \frac{n}{MT_S}\right). \tag{2.14}$$

Thus, the output spectrum of a TI-ADC with an offset mismatch between the channels has spurious tones at nf_S/M. The amplitude of these tones is given by the term O_n, which depends on the offset sequence.

2.1.2. Gain Error

The gain error of an ADC is defined as the difference between the last step's midpoint of the actual ADC and the last step's midpoint of the ideal ADC, once compensating for the offset error [47]. Figure 2.3 shows the characteristic of an ADC affected by gain error.

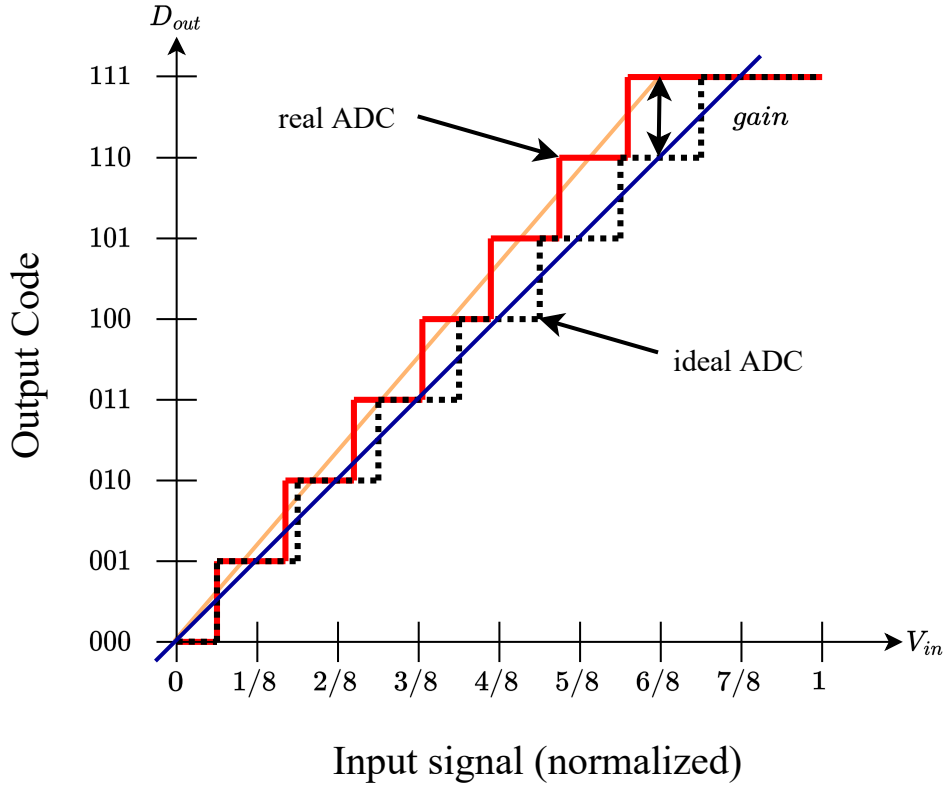

Figure 2.3: Example of ADC characteristic affected by gain error.

The gain error is a specific case of transfer function mismatch in which only the DC gain is taken into account. In the case of SAR converters, gain errors are mainly due to differences in the capacitive array and in the reference voltages [48].

As before, a multiplicative coefficient in a single-core SAR ADC is not an issue, but, in a TI converter, because of the periodic alternation of the channels, it modulates the amplitude of the signal lowering the SNDR and SFDR of the converter [24, 36, 40].

Figure 2.4 shows the block diagram of a M-channel TI-ADC affected by gain error. As in the case of the offset error, each channel is characterized by a different gain $1 + g_m$.

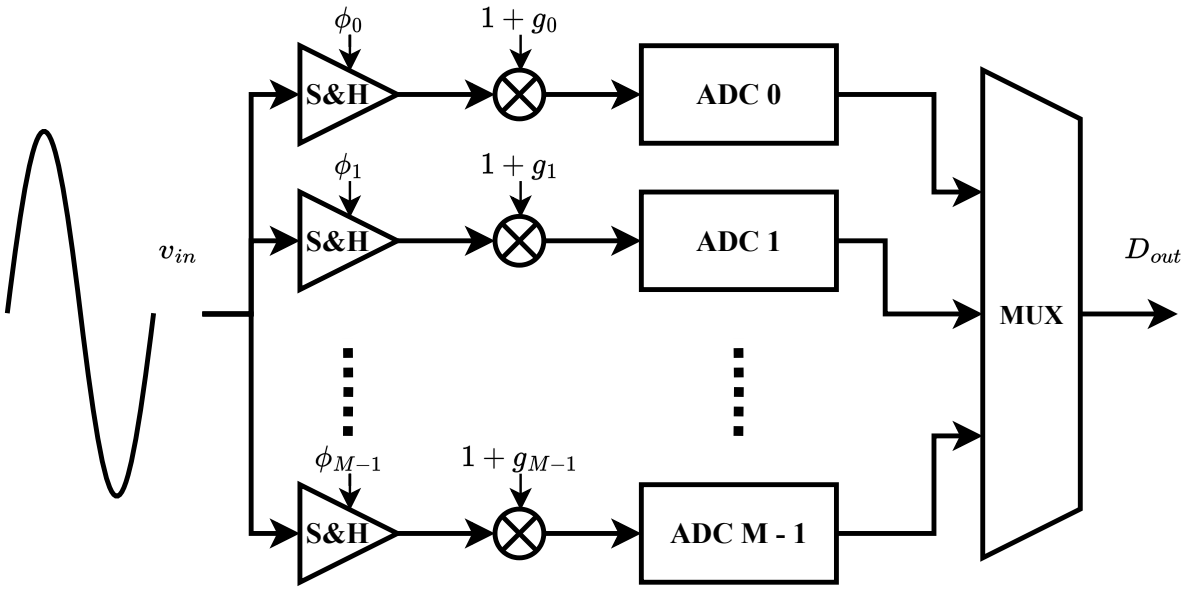

Figure 2.4: Block diagram of a M-channel TI-ADC affected by gain mismatch.

We can model the gain mismatch by multiplying each sampled signal by gain $1 + g_m$ as shown in figure 2.4. Thus, the input signal of each core is:

$$\hat{x}_m(t) = (1 + g_m) \cdot x(t)\,. \tag{2.15}$$

The use of a $1 + g_m$ multiplicative factor instead of a simple g_m allows to easily separate at a later stage the expression of the output spectrum due to the gain mismatch only.

Repeating the same algebraic operations done for the offset error, we end up with:

$$\hat{y}(t) = x(t) \sum_{n=-\infty}^{\infty} \delta\left(t - nT_S\right) + x(t) \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} g_m \delta\left(t - (nM + m) \cdot T_S\right). \tag{2.16}$$

The first part of equation 2.16 represents the ideal sampled signal (i.e., without gain mismatch), thus:

$$\hat{y}(t) = y(t) + x(t) \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} g_m \delta \left(t - (nM + m) \cdot T_S\right). \tag{2.17}$$

Moving now to the frequency domain, we can write:

$$\begin{aligned}
\hat{Y}(f) &= Y(f) + \mathscr{F}\left[x(t) \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} g_m \delta \left(t - (nM + m) \cdot T_S\right)\right] \\
&= Y(f) + X(f) * \sum_{m=0}^{M-1} g_m \cdot \mathscr{F}\left[\sum_{n=-\infty}^{\infty} \delta \left(t - (nM + m) \cdot T_S\right)\right] \\
&= Y(f) + X(f) * \sum_{m=0}^{M-1} g_m \cdot \frac{2\pi}{MT_S} \cdot e^{-j2\pi f m T_S} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{MT_S}\right) \\
&= Y(f) + X(f) * \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} \left[\sum_{m=0}^{M-1} g_m \cdot e^{-j2\pi f m T_S}\right] \delta\left(f - \frac{n}{MT_S}\right).
\end{aligned} \tag{2.18}$$

As for offset expression, the part within the square brackets is a DFT:

$$G_n = \sum_{m=0}^{M-1} g_m \cdot e^{-j2\pi nm/M}. \tag{2.19}$$

Substituting equation 2.19 into equation 2.18, it results:

$$\hat{Y}(f) = Y(f) + X(f) * \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} G_n \cdot \delta\left(f - \frac{n}{MT_S}\right). \tag{2.20}$$

From equation 2.20, it is possible to highlight the output spectrum due to the gain mismatch only:

$$E_{gain}(f) = X(f) * \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} G_n \cdot \delta\left(f - \frac{n}{MT_S}\right). \tag{2.21}$$

Now, in case the input signal is a sinewave with a frequency f_{in}, the input spectrum is:

$$X(f) = \frac{1}{2}\left[\delta(f - f_{in}) + \delta(f + f_{in})\right]. \tag{2.22}$$

If we plug equation 2.22 into equation 2.21, we get:

$$\begin{aligned} E_{gain}(f) &= \frac{1}{2}\left[\delta(f-f_{in})+\delta(f+f_{in})\right] * \frac{2\pi}{MT_S}\sum_{n=-\infty}^{\infty} G_n \cdot \delta\left(f-\frac{n}{MT_S}\right) \\ &= \frac{\pi}{MT_S}\sum_{n=-\infty}^{\infty} G_n \cdot \left[\delta\left(f-f_{in}-\frac{n}{MT_S}\right)+\delta\left(f+f_{in}-\frac{n}{MT_S}\right)\right]. \end{aligned} \tag{2.23}$$

Thus, the error in the output spectrum of a TI-ADC introduced by the gain mismatch is represented by a series of spurious tones at $nf_S/M \pm f_{in}$. The amplitude of the tones is given by the term G_n, which depends on the gain sequence.

2.1.3. Skew Error

In an ideal TI-ADC, each channel samples the input signal after a time T_S from the previous core. In a real application, the time interval between consecutive channels is not constant due to the presence of skew between the different phases of the clock [24].

The clock skew has multiple origins, some are deterministic, like differences in the clock distribution network, others are statistical, and others, like PVT[1] are slowly varying [49].

Wann and Franklin in [50] list four main sources of clock skew:

- Difference in the length of the paths from the clock source to the clock sinks.
- Differences in the delays through active elements (i.e., clock buffers, clock multiplexers, etc.) in the paths.
- Differences in the parameters (i.e., resistivity, dielectric constant, width, thickness, parasitic capacitance, etc.) of the paths.
- Differences in the parameters (i.e., threshold voltage, channel mobility, etc.) of the active elements in the paths.

Furthermore, in a real case, this time shift is also time-dependent due to the presence of a clock jitter, resulting in a sampling instant that varies also in time. For our purpose, it is enough to examine the effects of a deterministic time-skew.

[1]PVT: Process, Voltage, and Temperature are some of the conditions that influence the performances of an IC. The Process is related to the variations in the device parameters (threshold voltage, mobility, etc.). These variations are fixed at fabrication time. Voltage is related to variations in the power supplies, which might be caused for instance, by a time-dependent IR drop. Temperature influences the properties of the devices by acting on their parameter. Mobility, for instance, is highly temperature dependant. The last two conditions (voltage and temperature) can change during the operation of the IC.

Figure 2.5 shows a block diagram of a M-channel TI-ADC affected by clock skew, where the skew is represented by the term $\delta\phi_m$ (with $m = 0, 1, ..., M-1$), added to each sampling phase ϕ_m.

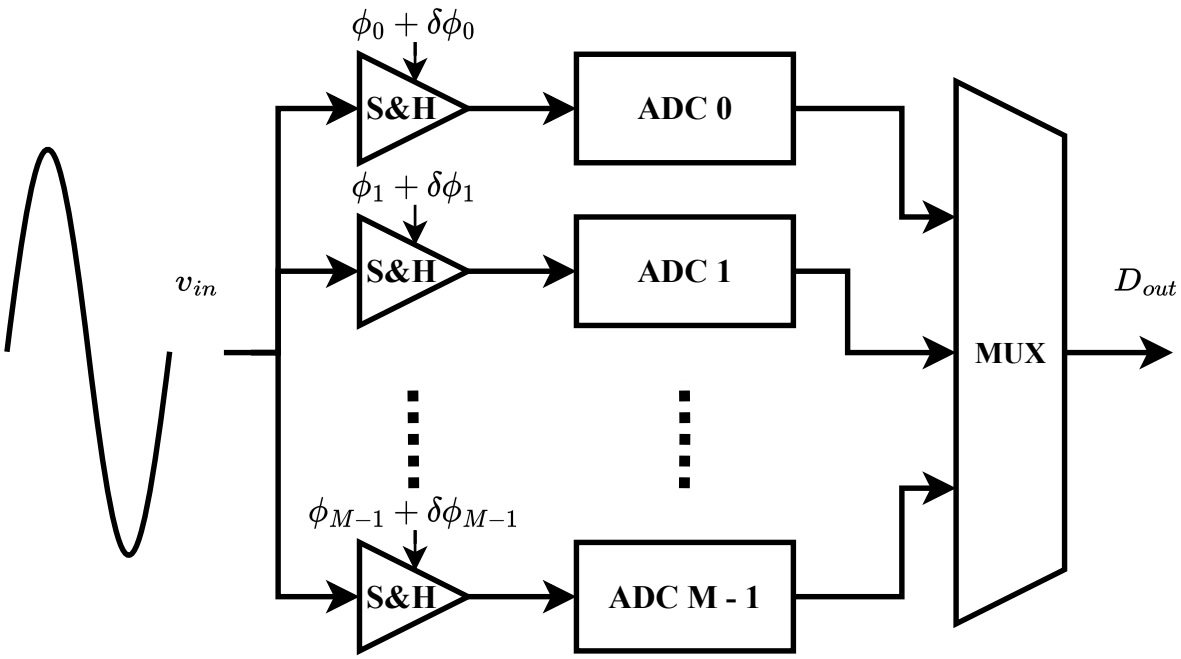

Figure 2.5: Block diagram of a M channel TI-ADC affected by skew error.

An example of the clock phases of a TI-ADC affected by skew is shown in figure 2.6.

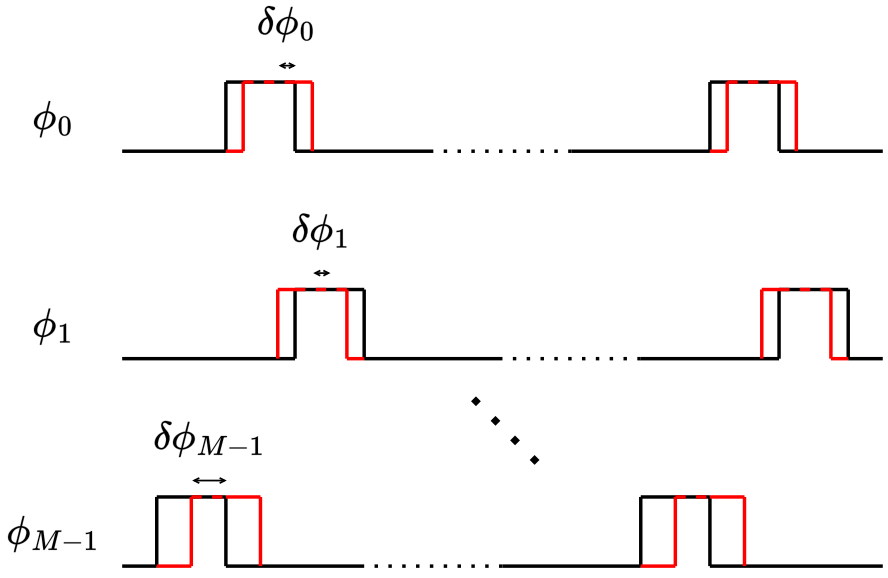

Figure 2.6: Clock phases of a M-channel TI-ADC affected by clock skew.

It is possible to model the effect of the time-skew by acting on the sampled signal of each channel, introducing a delay t_m, different from each channel, thus, we can write:

$$\hat{y}_m(t) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - (nM + m) \cdot T_S - t_m)\,, \tag{2.24}$$

t_m being the time-skew corresponding to the phase-skew $\delta\phi_m$.

The overall output signal is given by the summation of the contributions of each channel, being:

$$\hat{y}(t) = \sum_{m=0}^{M-1} \hat{y}_m(t) = \sum_{m=0}^{M-1} x(t) \sum_{n=-\infty}^{\infty} \delta(t - (nM + m) \cdot T_S - t_m)\,. \tag{2.25}$$

Reordering equation 2.25, we get:

$$\hat{y}(t) = x(t) \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} \delta(t - (nM + m) \cdot T_S - t_m)\,. \tag{2.26}$$

Moving now to the frequency domain:

$$\begin{aligned}
\hat{Y}(f) &= \mathscr{F}\left[x(t) \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} \delta(t - (nM + m) \cdot T_S - t_m)\right] \\
&= X(f) * \sum_{m=0}^{M-1} \mathscr{F}\left[\sum_{n=-\infty}^{\infty} \delta(t - (nM + m) \cdot T_S - t_m)\right] \\
&= X(f) * \sum_{m=0}^{M-1} \frac{2\pi}{MT_S} \cdot e^{-j2\pi f(mT_S - t_m)} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{MT_S}\right) \\
&= X(f) * \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} \left[\sum_{m=0}^{M-1} e^{-j2\pi f t_m} \cdot e^{-j2\pi f m T_S}\right] \delta\left(f - \frac{n}{MT_S}\right).
\end{aligned} \tag{2.27}$$

The part within the square brackets is a Discrete Fourier Transform:

$$T_n = \sum_{m=0}^{M-1} e^{-j2\pi f t_m} \cdot e^{-j2\pi f m T_S}\,. \tag{2.28}$$

Plugging equation 2.28 into equation 2.27, it results:

$$\hat{Y}(f) = X(f) * \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} T_n \cdot \delta\left(f - \frac{n}{MT_S}\right). \tag{2.29}$$

Now, let's suppose our input signal is a sinewave at a frequency f_{in}, its spectrum being the one expressed by equation 2.22. If we plug equation 2.22 into equation 2.29, we get:

$$\begin{aligned} \hat{Y}(f) &= \frac{1}{2}\left[\delta(f - f_{in}) + \delta(f + f_{in})\right] * \frac{2\pi}{MT_S} \sum_{n=-\infty}^{\infty} T_n \cdot \delta\left(f - \frac{n}{MT_S}\right) \\ &= \frac{\pi}{MT_S} \sum_{n=-\infty}^{\infty} T_n \cdot \left[\delta\left(f - f_{in} - \frac{n}{MT_S}\right) + \delta\left(f + f_{in} - \frac{n}{MT_S}\right)\right]. \end{aligned} \tag{2.30}$$

The output spectrum of a TI-ADC affected by sampling skew is characterized by a series of spurious tones at $nf_S/M \pm f_{in}$, like the error introduced by the gain mismatch. The amplitude of the tones is given by the term T_n, which depends on the skew sequence.

Note that unlike the case of gain error, the SNDR degradation due to time-skew depends on the frequency of the input signal as shown in [36, 39, 45].

2.1.4. Non-ideal Characteristic of the ADC

Aside from the three sources of error analyzed in the previous sections (offset, gain, and time-skew), ADCs are also affected by nonlinearity errors (DNL, INL) [46].

The transfer function of an ideal ADC is a staircase in which each step corresponds to a particular digital word. Ideally, each step has the same amplitude equal to an LSB.

In a real ADC, there are deviations from the ideal characteristic. Those deviations are characterized using two parameters:

- Differential Non-Linearity (DNL): defined as the difference in the step width between the actual transfer function and the ideal transfer function [47].
- Integral Non-Linearity (INL): defined as the vertical difference between the actual and the ideal transfer functions [47].

In a SAR ADC, those non-linearities are mainly related to mismatches in the capacitive array [48]. These deviations from the ideal characteristics lower the SNDR and SFDR of the converter and represent an issue even for a single SAR.

Figures 2.7 and 2.8 show the characteristics of a real ADC, compared to an ideal one, highlighting the concept of DNL and INL, respectively.

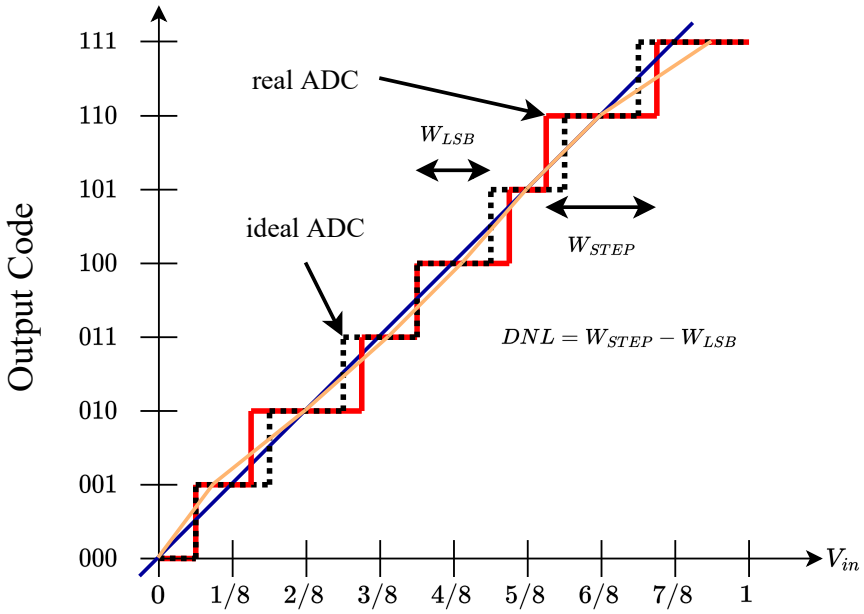

Figure 2.7: Definition of DNL of an ADC.

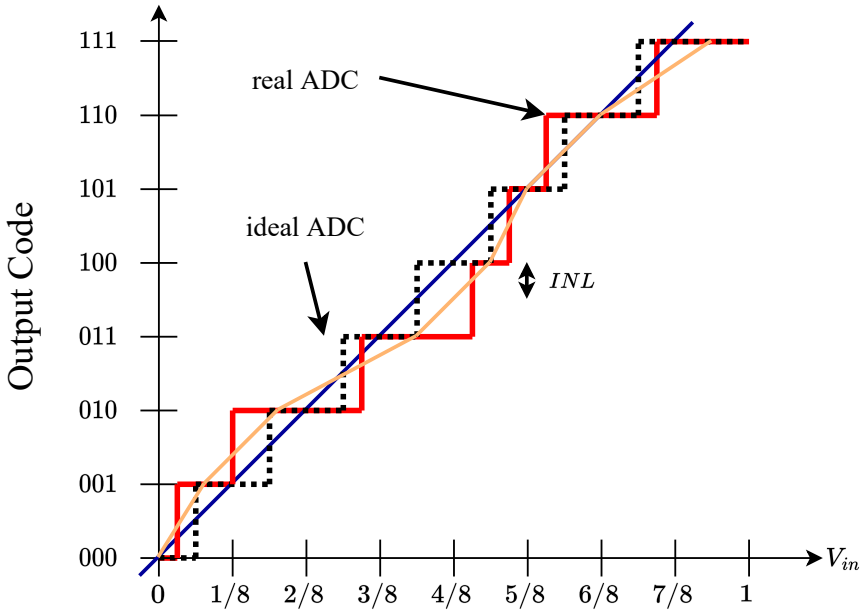

Figure 2.8: Definition of INL of an ADC.

In order to understand the effect of the core nonlinearity on the overall converter, it's possible to resort to *ad hoc* simulations since it's difficult to treat this non-ideality analytically.

2.1.5. MATLAB Simulations of the Non-idealities

A MATLAB model has been developed to validate the analytical models presented in the previous sections.

The model describes a 10-bit 8-channel 2-GS/s TI-ADC. All four non-idealities described in the first sections of this chapter can be simulated both individually and together.

The input signal is a sinewave with an amplitude between -1 and 1. Offset, gain, and time-skew errors are modelled exactly as shown in the respective sections, which results in an input signal for each channel that can be written as:

$$S_m[k] = o_m + (1 + g_m) \cdot sin\left(2\pi f_{in} \cdot \left(\frac{M \cdot k + m}{f_S} + t_m\right)\right), \tag{2.31}$$

where t_m is the time-skew corresponding to the phase-skew $\delta\phi_m$.

The nonlinearity of each core is modelled according to:

$$S_{nonlinear}[k] = \alpha \cdot tanh\left(\frac{S_{linear}[k]}{\alpha}\right), \tag{2.32}$$

where α is used to control the magnitude of the nonlinearity effect.

All the parameters can be both specified or randomly generated. The simulations shown in this chapter have been run with randomly generated parameters. The standard deviations used for the different parameters are listed in table 2.1.

Parameter	μ	σ	Note	average SNDR
o_m	0	$8 \cdot 10^{-3}$	16 mV	40.1 dB
$G_m = 1 + g_m$	1	$1.2 \cdot 10^{-2}$		39.68 dB
t_m	0	$20 \cdot 10^{-12}$	20 ps	49.9 dB
α	5	$5 \cdot 10^{-2}$		49.4 dB

Table 2.1: Summary of the parameters used for the MATLAB simulation.

The average SNDR reported in the table has been calculated as the average over 100 simulations with the parameters reported in the respective row considered independently. All the results have been simulated with a sinewave input at $f_{in} = 27\ MHz$.

Figure 2.9 shows the output spectrum of an 8-channel TI-ADC affected by offset mismatch.

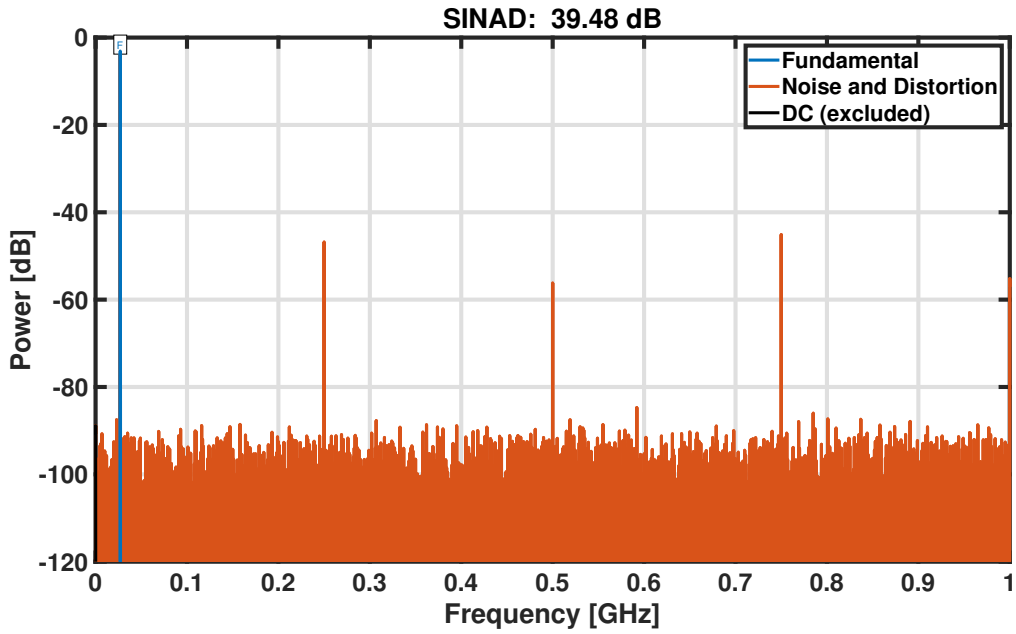

Figure 2.9: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by offset mismatch. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

The spectrum has spurious tones at $mf_S/8$, consistently with the mathematical model. Note that the spectrum is limited to the Nyquist frequency, thus tones at higher frequencies are not shown.

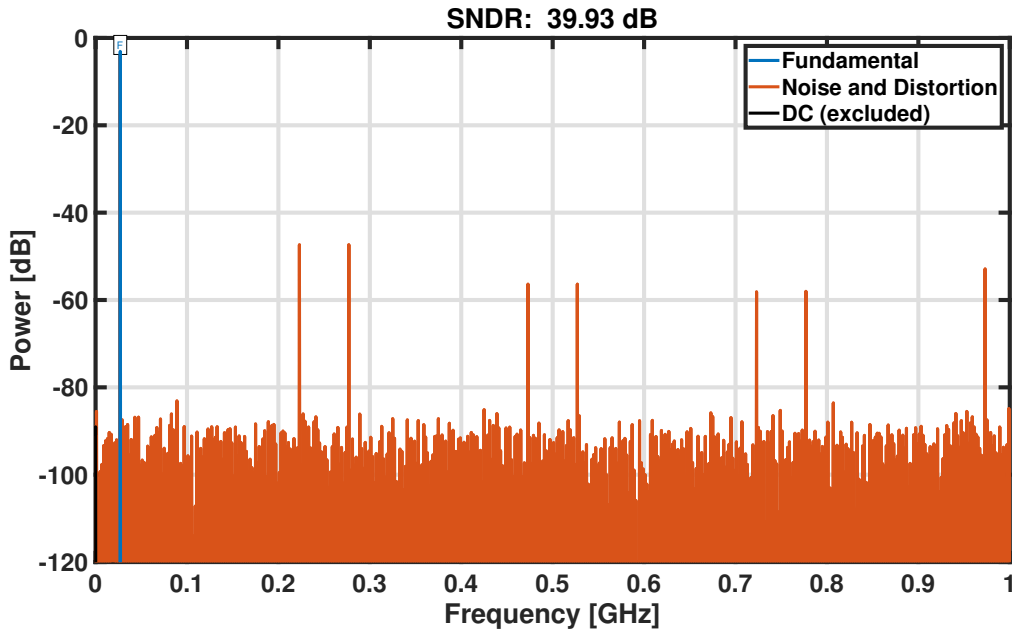

Figure 2.10: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by gain mismatch. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

Figure 2.10 shows the output spectrum of an 8-channel TI-ADC affected by gain mismatch. The spurious tones are at $mf_S/8 \pm f_{in}$ according with the analytical model derived in the first section of this chapter.

Figure 2.11 shows the output spectrum of an 8-channel TI-ADC affected by time-skew.

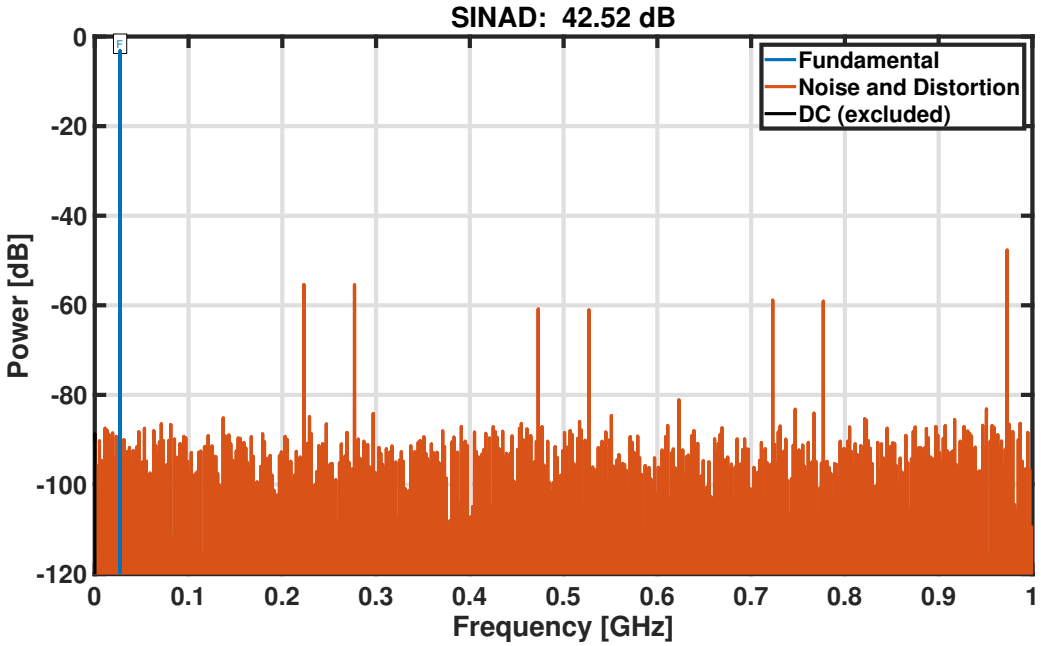

Figure 2.11: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by time-skew. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

Again, the resulting output spectrum shows spurious tones at $mf_S/8 \pm f_{in}$, as predicted by the model.

Figure 2.12 shows two output spectra of an 8-channel TI-ADC, affected by nonlinearity, without (left) and with (right) nonlinearity mismatches between the cores. For the right plot, in fact, different α values have been considered for the 8 cores.

From the two plots in figure 2.12, we can see that a mismatch of the nonlinearity between the cores has negligible effects (a difference of only 0.06 dB) with respect to the one due to the presence of the nonlinearity itself, as shown in [51].

Note that this effect is not introduced by the time-interleaved architecture as with the offset, gain, and time-skew, but it is something that characterizes also each core ADC.

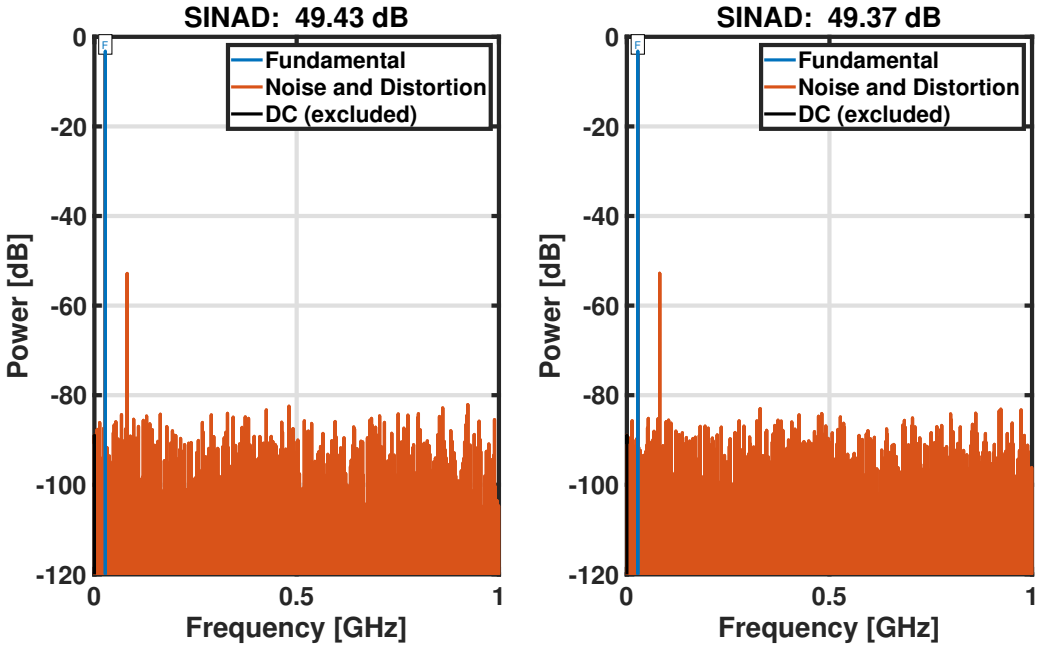

Figure 2.12: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by nonlinearity, without (left) and with (right) nonlinearity mismatches between the cores. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

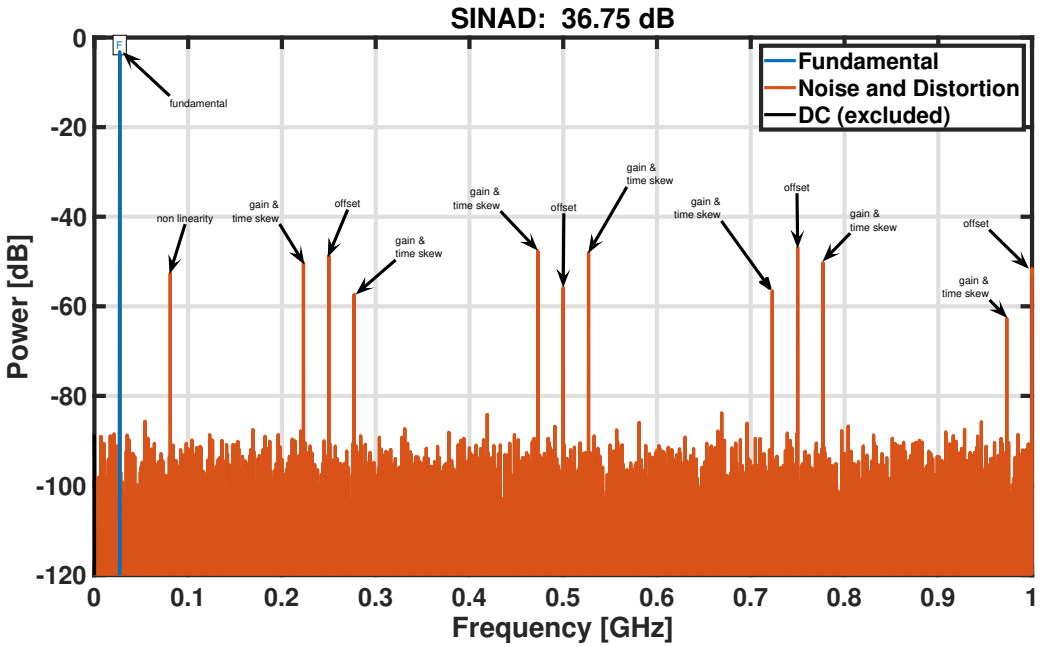

Figure 2.13: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by offset and gain mismatches, time-skew, and nonlinearity. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

Considering all the non-idealities of the TI converter, we obtain the overall output spectrum shown in figure 2.13.

The output spectrum shows spurious tones due to the offset mismatch at $mf_S/8$ (250 MHz, 500 MHz, 750 MHz, 1 GHz, ...), like in figure 2.9, and at $mf_S/8 \pm f_{in}$ (223 MHz, 277 MHz, 473 MHz, 527 MHz, 723 MHz, 777 MHz, 973 MHz, ...) due to gain mismatch and time-skew as shown in figures 2.10 (gain) and 2.11 (time-skew).

Moreover, a large tone at $3f_{in} = 81$ MHz is present, which can be ascribed to the effect of the core nonlinearity, as shown in figure 2.12.

Figure 2.14 shows the SNDR resulting from a frequency sweep of the TI-converter considering the three non-idealities related to the TI architecture (offset, gain, and time-skew) independently.

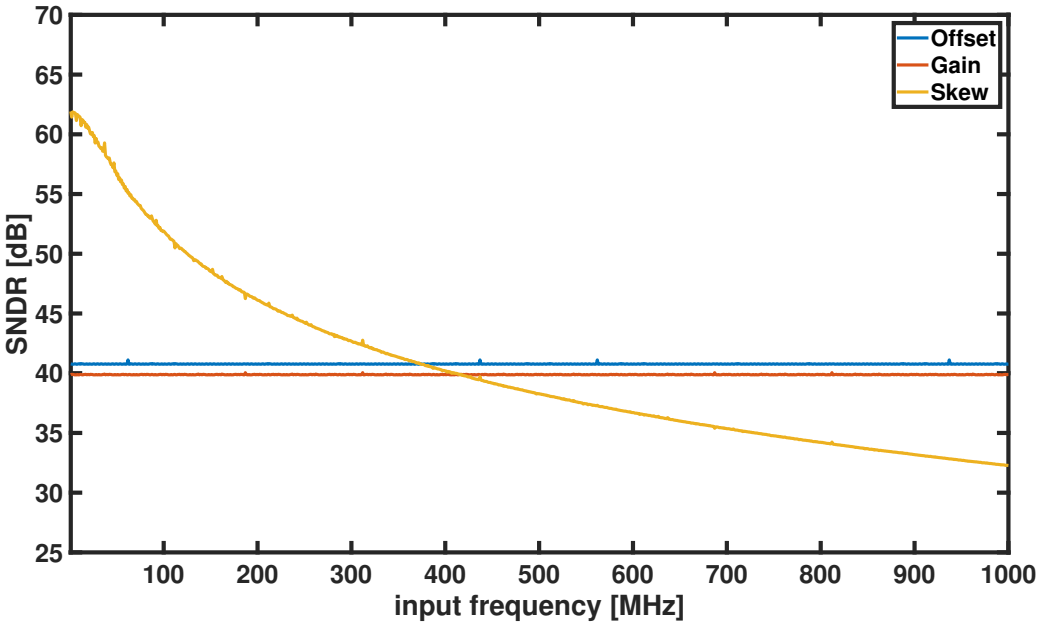

Figure 2.14: Frequency sweep simulated in MATLAB of a 10-bit 2-GS/s 8-channel TI-ADC affected separately by offset mismatches, gain mismatches, and time-skew. The input is a sinewave whose frequency sweeps from 1.5 MHz to 999.5 MHz in 1.0-MHz steps.

As mentioned in section 2.1.3, the degradation of the performances of the TI converter introduced by the skew is frequency dependant [36, 39, 45]. The effects of offset and gain mismatches are, instead, independent of the input frequency.

2.2. Calibration Algorithms

The need for calibration of the mismatches is clear from the analysis presented in the previous section. The principal limiting factors of the performances of a TI-ADC are the offset, gain, and timing mismatches [24, 52]. Among the three errors, offset and again mismatches are frequency independent, whereas timing mismatches become more relevant at high-frequency [36, 39, 45]. Furthermore, gain and time-skew errors are also dependent on the amplitude of the input signal.

There are multiple different approaches found in literature, depending both on the domain of implementation (analog or digital) and in the mode of operation (background or foreground) [52]. Analog calibration methods detect the error and act directly on its source, changing the operating point of the analog circuit to minimize the non-idealities. Digital approaches detect and correct the error directly in the digital domain [53–55]. A third, mixed-signal approach detects the errors in the digital domain and corrects them in the analog one. Each non-ideality is better calibrated in its own domains. For instance, time-skew calibration is predominantly corrected in the analog domain (and detected in the digital one) [56]. On the other hand, offset and gain mismatches are usually corrected in the digital domain [52]. However, there are opposite examples in literature, like [57–59], which propose a fully-digital calibration also for time-skew error.

From what concerns the difference between background and foreground calibration we have that background (BG) calibration occurs during the normal operation of the converter and is, therefore, invisible to the user. Foreground (FG) calibration, on the other hand, interrupts the normal operation of the converter. Another advantage of background calibrations consists in their capability of intrinsically tracking PVT variations of the IC.

From this brief description, it looks like background calibration is superior to foreground calibration. However, background calibration more often than not poses some constraints on the input signal [52].

The earliest reference regarding methods for mitigating the effects of TI converter focuses on a two-rank SH architecture to avoid timing error [60]. Dyer, Fu, Lewis, and Hurst provide in [61, 62] and [63, 64], respectively, a digital and an analog background calibration of offset and gain mismatches. In both cases, the time-skew error is avoided using a first-rank S&H architecture.

In this work, we analyze only digital calibration algorithms, which can be implemented on an FPGA.

2.2.1. Offset Correction

Let's start with the calibration of the offset mismatch. The idea is to estimate the DC offset of the channel and then subtract it from the output, resulting in:

$$S_{m,corrected}[k] = S_{m,non\ corrected}[k] - O_m[k]\,, \tag{2.33}$$

with $O_m[k]$ being the estimate of the offset of the m^{th} channel at the instant k.

A possible method to achieve this task is to average a certain number of samples from the channel using a Finite Impulse Response[2] (FIR) filter. For instance, using a moving average FIR results in an estimation of the offset of the channel expressed by:

$$O_m[k] = \frac{1}{N}\sum_{n=1}^{N} S_{m,non\ corrected}[k-n]\,. \tag{2.34}$$

Actually, we are not interested in removing the offset of each core. It is enough that all the cores have the same offset, which means the mismatch is eliminated.

It is possible to use one of the channels as a reference and calibrate all other channels with respect to it. Thus, equation 2.34 becomes:

$$o_m[k] = \frac{1}{N}\sum_{n=1}^{N} \left(S_{m,non\ corrected}[k-n] - S_{reference}[k-n]\right)\,, \tag{2.35}$$

with $o_m[k]$ representing the estimation of the offset mismatch of the m^{th} channel with respect to the reference channel at the instant k.

Note that in this case, the reference channel is not subject to calibration.

Though an FIR filter is usually simpler to implement and is inherently stable, it provides only an approximation of the offset mismatch of the channel and its accuracy depends on the number N of samples used to calculate the moving average. Furthermore, it also requires a large number of resources. Referring to the aforementioned moving average, we need to store a number of samples equal to the length of the filter in a shift register, a requirement that becomes quickly prohibitive, especially for the low cut-off frequency required for this kind of application.

[2]FIR: a Finite Impulse Response is a filter whose impulse response is finite, which means that the response of the filter to a finite duration input signal has a finite duration.

A possible solution is to use Infinite Impulse Response[3] (IIR) filters [55]. Equations 2.33 and 2.35 become respectively:

$$S_{m,corrected}[k] = S_{m,non\ corrected}[k] - \gamma_{offset} \cdot o_m[k] \tag{2.36}$$

and

$$o_m[k] = \sum_{n=1}^{\infty} \left(S_{m,corrected}[k-n] - S_{reference}[k-n]\right) . \tag{2.37}$$

Equation 2.37 can be rewritten in the more clear form of:

$$o_m[k] = o_m[k-1] + S_{m,corrected}[k-1] - S_{reference}[k-1] . \tag{2.38}$$

IIR filters use feedback to achieve low cut-off frequencies and sharp cut-offs using few hardware resources. However, the presence of the feedback loops introduces stability issues. Figure 2.15 shows the block diagram of the offset compensation algorithm using an IIR filter.

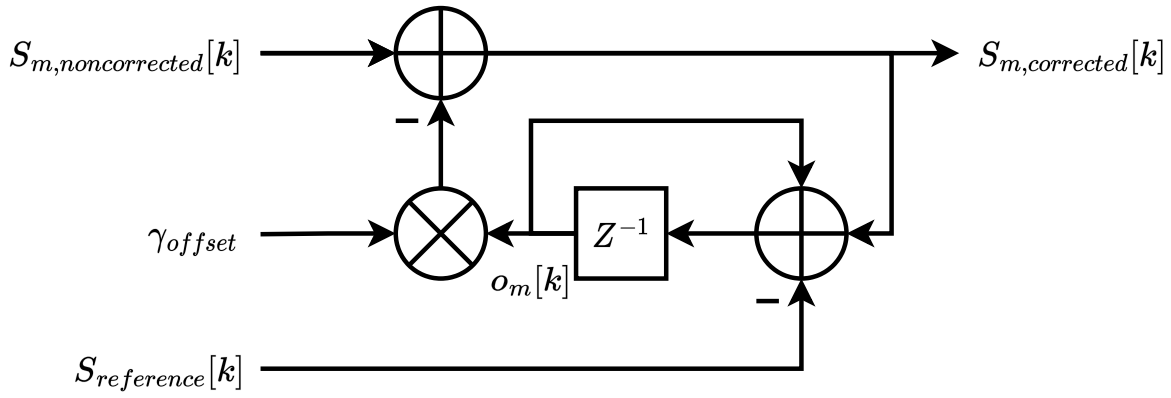

Figure 2.15: Block diagram of offset compensation algorithm using an IIR filter.

The choice of the γ_{offset} factor is critical for the stability and convergence speed of the offset calibration algorithm. Large γ_{offset} values might cause instability, whereas small values cause long convergence times. The effects of different values of γ_{offset} have been investigated using MATLAB simulations.

[3]IIR: an Infinite Impulse Response is a filter whose impulse response is infinite, this means the response of the filter to a finite duration input signal has infinite duration

2.2.2. Gain Correction

The same analysis done for the offset mismatch can be repeated for the gain mismatch. We limit ourselves to reporting the results. As for the offset mismatch, we are not interested to remove the gain error from each core, but it is enough to force all cores to have the same gain.

In this case, we want to correct a multiplicative error, thus equation 2.33 becomes:

$$S_{m,corrected}[k] = S_{m,non\ corrected}[k] \cdot \gamma_{gain} \cdot g_m[k]\,. \tag{2.39}$$

Equation 2.38 can be modified to use the absolute value (or the square value) of the signals [55]:

$$g_m[k] = g_m[k-1] - |S_{m,corrected}[k-1]| + |S_{reference}[k-1]|\ . \tag{2.40}$$

We prefer the use of the absolute value since it does not involve multipliers, which requires a lot of hardware. Figure 2.16 shows the block diagram of the gain calibration algorithm using an IIR filter:

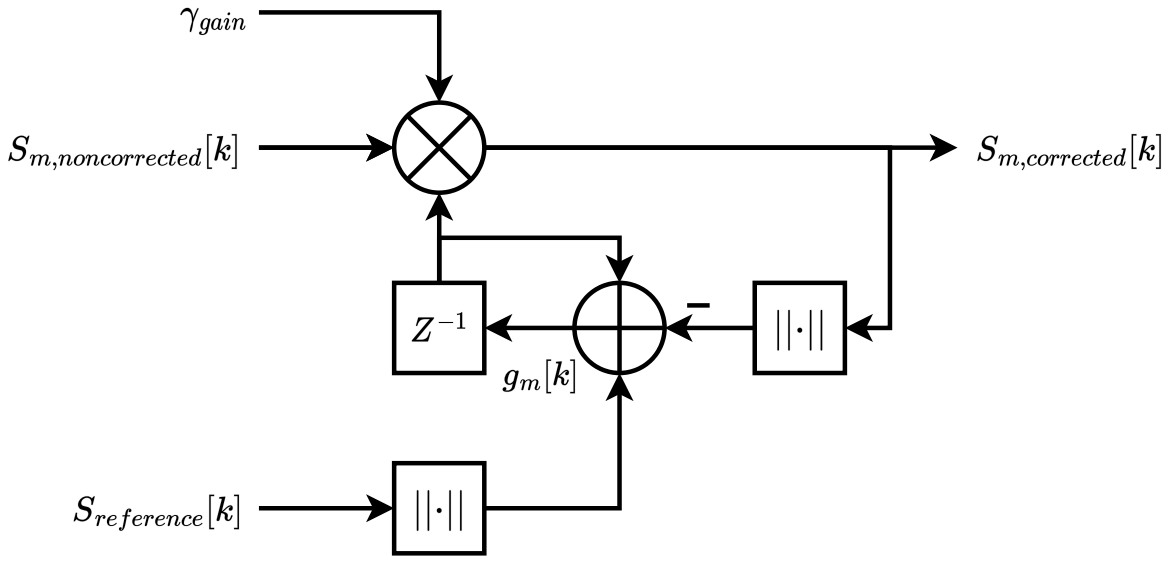

Figure 2.16: Block diagram of gain compensation algorithm using an IIR filter.

As for the offset calibration, the choice of the γ_{gain} factor has influences on stability and convergence speed of the system [55].

2.2.3. MATLAB Simulations of Correction Algorithms

This sub-section presents the MATLAB simulations of the calibration algorithms used to correct the output of the TI-ADC affected by non-idealities.

Figure 2.17 shows the output spectrum of an 8-channel TI-ADC affected by offset mismatch, before (left) and after (right) the calibration algorithm convergence.

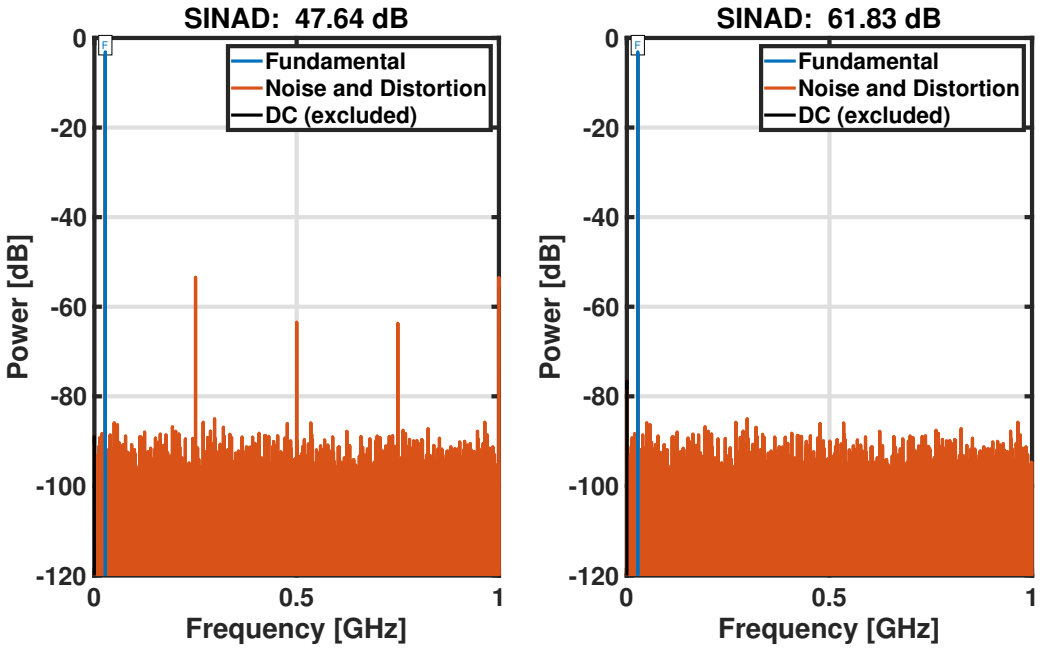

Figure 2.17: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by offset mismatch before (left), and after (right) the offset calibration algorithm convergence. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

It is possible to notice how the spurious tones caused by the offset mismatch are not present in the right plot.

The parameters used to model the offsets, and the expected values of $\gamma_{offset} \cdot o_m$ of all the channels are summarized in table 2.2. The use of $\hat{}$ denotes the coefficients used to model the non-idealities.

CH	0	1	2	3	4	5	6	7
$\hat{o_m}$	0.001	0.0031	-0.004	-0.0014	-0.005	0.002	0.001	0.0027
$\gamma_o o_m$	0	-0.0021	0.005	0.0024	0.006	-0.001	0	0.0017

Table 2.2: Summary of the parameters used for the MATLAB simulation of the offset mismatch and the respective expected offset calibration coefficients.

Figures 2.18, 2.19, and 2.20 shows the evolution of the $\gamma_{offset} \cdot o_m[k]$ coefficients of all the channels for γ_{offset} equals to 2^{-12}, 2^{-16}, and 2^{-20}, respectively. Note that the x-axes of the plots are different.

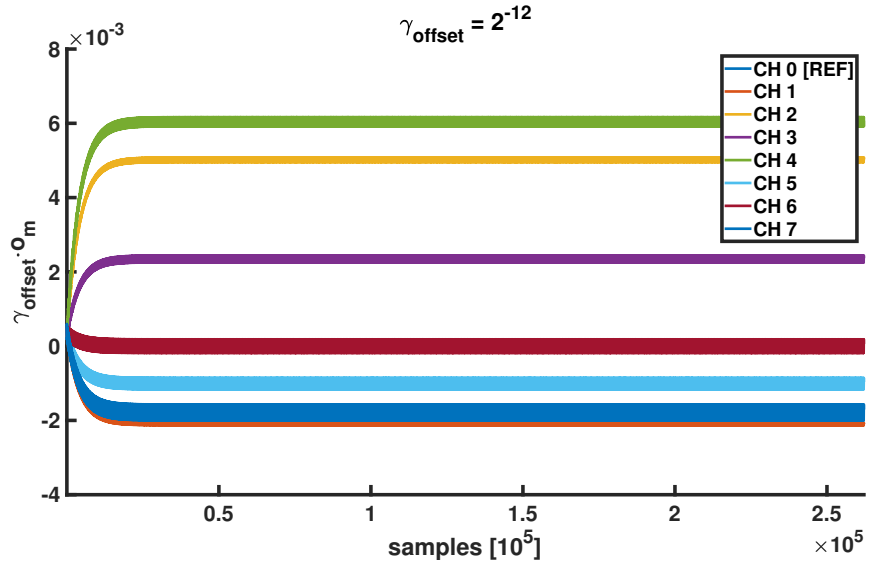

Figure 2.18: MATLAB simulation of the offset calibration algorithm showing the trend of the $\gamma_{offset} o_m$ coefficients. $\gamma_{offset} = 2^{-12}$.

In figure 2.18 it is possible to notice the oscillation of the $\gamma_{offset} o_m$ coefficients around their expected values, a symptom that the γ_{offset} coefficient is too small.

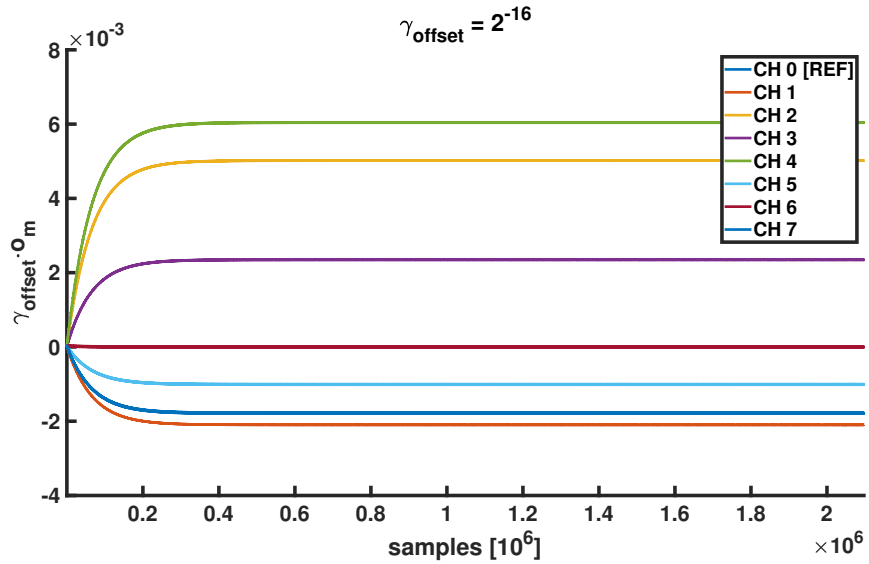

Figure 2.19: MATLAB simulation of the offset calibration algorithm showing the trend of the $\gamma_{offset} o_m$ coefficients. $\gamma_{offset} = 2^{-16}$.

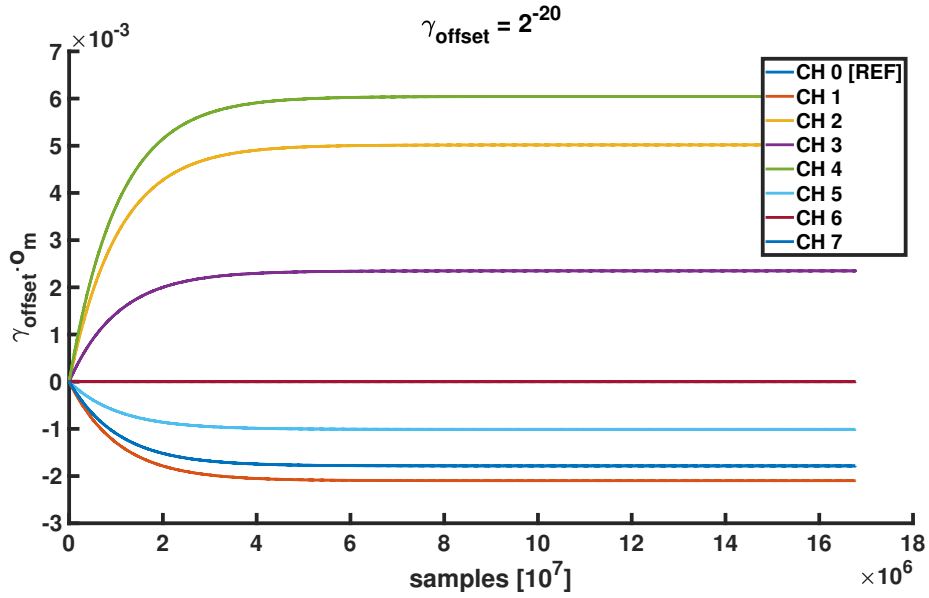

Figure 2.20: MATLAB simulation of the offset calibration algorithm showing the trend of the $\gamma_{offset} o_m$ coefficients. $\gamma_{offset} = 2^{-20}$.

As mentioned in section 2.2, the value of γ_{offset} influences the stability and the convergence speed of the algorithm.

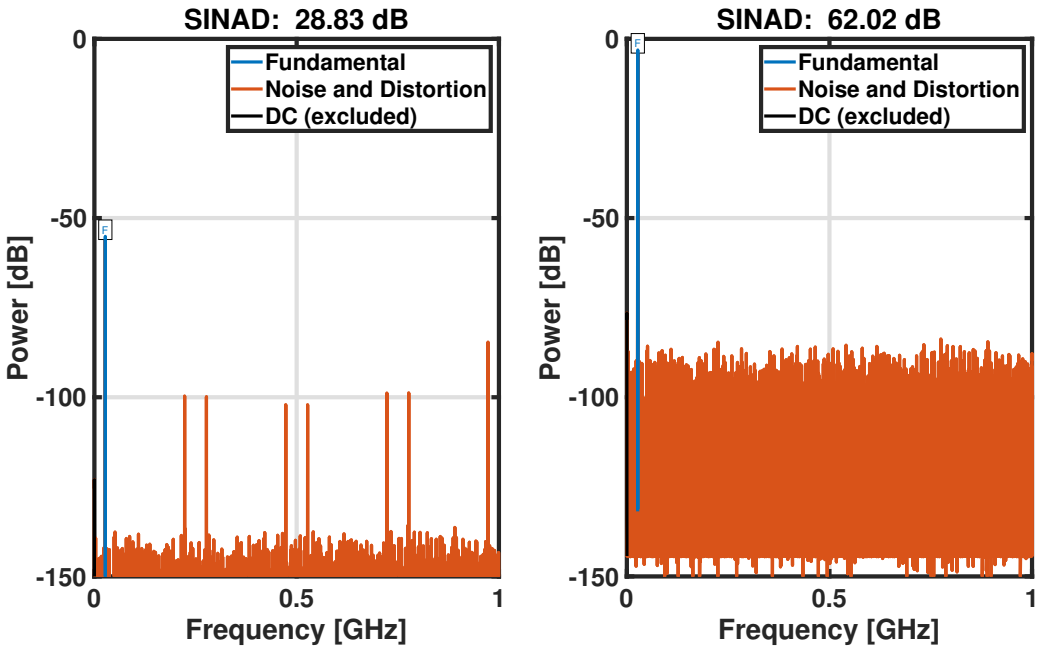

Figure 2.21: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by gain mismatch before (left), and after (right) the gain calibration algorithm convergence. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

Figure 2.21 shows the output spectrum of an 8-channel TI-ADC affected by gain mismatch, before (left) and after (right) the gain calibration algorithm convergence.

As for the offset calibration algorithm, the spurious tones introduced in the output spectrum by the gain mismatch are not present in the corrected output spectrum (right).

It is also possible to notice (on the left plot in figure 2.21) the initial attenuation of the fundamental (of the whole spectrum actually) caused by the zero initial value of $\gamma_{gain} \cdot g_m$ coefficients (as shown in figure 2.22).

In fact, the output sample of the gain calibration algorithm (expressed by equation 2.39, and reported here for the sake of clarity) can be written as:

$$S_{out} = S_{in} \cdot \gamma_{gain} g_m \,, \tag{2.41}$$

thus, at the first cycle of the calibration algorithm (i.e., when $g_m = 0$), there is no transfer to the output.

The transfer to the output increases until the $\gamma_{gain} \cdot g_m$ coefficients reach their expected values as shown in figure 2.22.

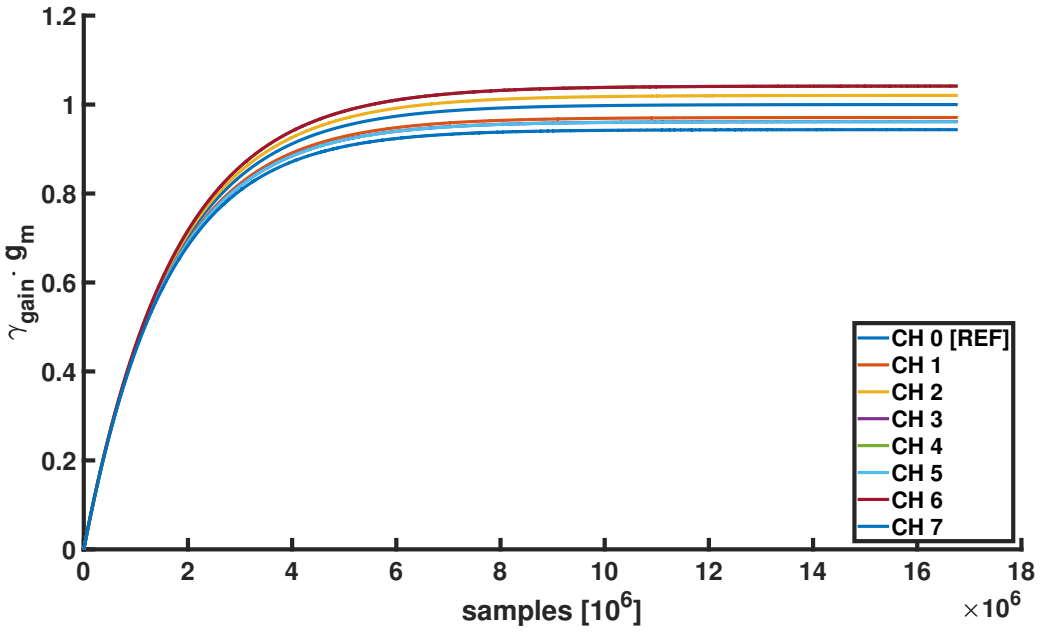

Figure 2.22: MATLAB simulation of the gain calibration algorithm showing the trend of the $\gamma_{gain} g_m$ coefficients. $\gamma_{gain} = 2^{-20}$.

It is possible to notice how the $\gamma_{gain} g_m$ coefficients converge to the expected values summarized in table 2.3. As before, the use of $\hat{}$ denotes the coefficients used to model the non-idealities.

CH	0	1	2	3	4	5	6	7
$\hat{g_m}$	1	1.03	0.98	1.04	0.96	1.04	0.96	1.06
$\gamma_g g_m$	1	0.97	1.02	0.96	1.04	0.96	1.04	0.94

Table 2.3: Summary of the parameters used for the MATLAB simulation of the gain mismatch and the respective expected gain calibration coefficients.

The gain calibration algorithm behaves similarly to the offset calibration one (shown in figures 2.18, 2.19, and 2.20) with respect to the stability-convergence speed trade-off of the $\gamma_{gain} g_m$ coefficients, therefore the results of the simulations are not reported here for the sake of brevity.

Figure 2.23 shows the output spectrum of an 8-channel TI-ADC affected by both gain and offset mismatches, before (left) and after (right) the calibration algorithms convergence. The two algorithms act in series, the first one is the offset compensation, followed by the gain calibration.

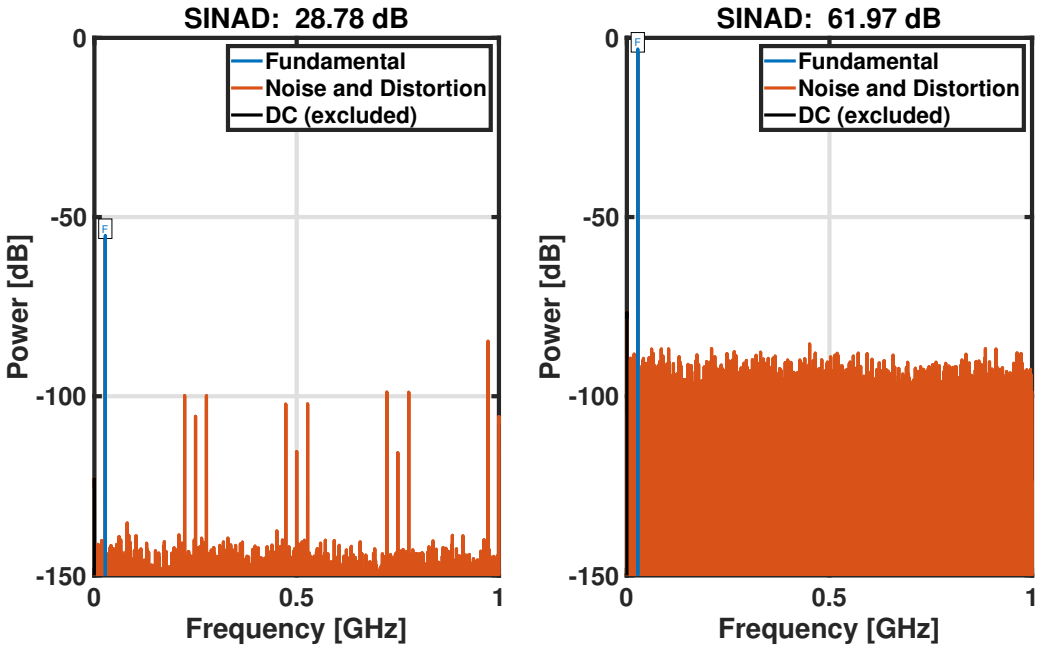

Figure 2.23: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by offset and gain mismatch before (left), and after (right) the calibration algorithms convergence. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

Again, we can notice (on the left in figure 2.23) the initial attenuation of the spectrum caused by the zero initial value of the $\gamma_{gain} \cdot g_m$ coefficients.

The calibration algorithms correct the offset and gain mismatches even when working together (i.e., in series) as shown by the lack of spurious tones in the right plot of figure 2.23.

As for the case where the single non-idealities were considered separately, the SNDR of the calibrated spectrum tends to the theoretical one of $SNDR \approx 10 \cdot 6.02\ dB + 1.76\ dB \approx 61.96\ dB$.

The values of the coefficients used to model the non-idealities in the simulations shown in this chapter are summarized in tables 2.2 (offset) and 2.3 (gain). The respective coefficients converge to their expected values as shown in figures 2.18, 2.19, 2.20 (for the offset) and 2.22 (for the gain).

For the sake of completeness, figure 2.24 shows the output spectrum of an 8-channel TI-ADC affected by all the non-idealities analyzed in this chapter (offset, gain, time-skew, and nonlinearity), before (left) and after (right) the calibration algorithms convergence. For this simulation, the same α value has been considered for the 8 cores.

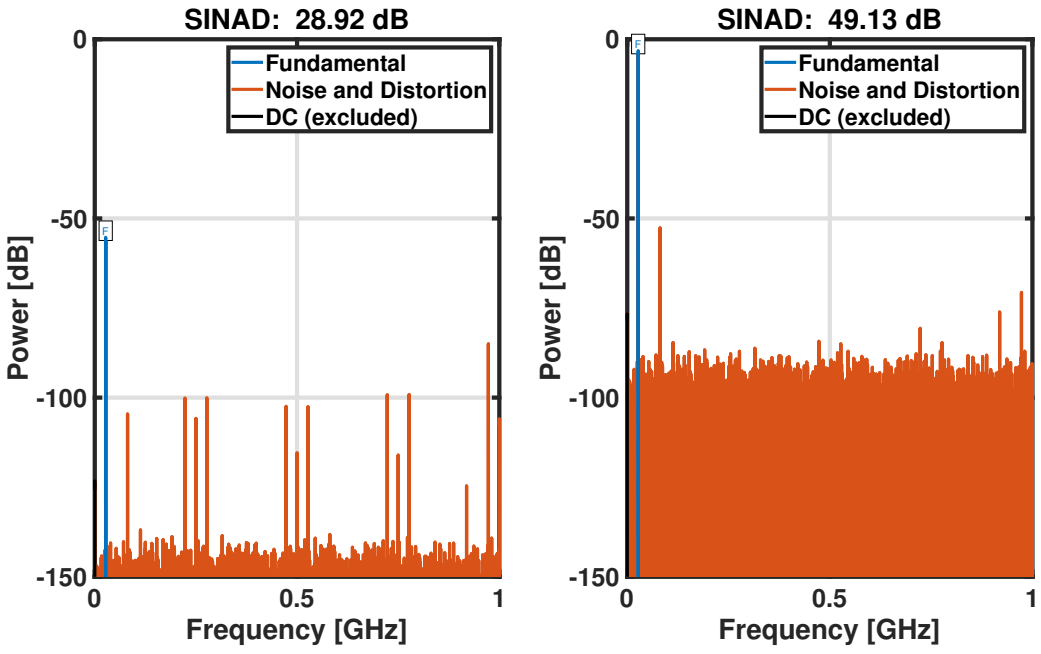

Figure 2.24: MATLAB simulation results of a 10-bit 2-GS/s 8-channel TI-ADC affected by offset and gain mismatch, time-skew, and core nonlinearity, before (left), and after (right) the calibration algorithms convergence. The input is a sinewave at a frequency $f_{in} = 27\ MHz$.

In this case, we have some spurious tones related to the nonlinearity and to the time-skew in the output spectrum. The offset and gain calibration algorithms are (rightly so) unable to correct these non-idealities.

3 | Implementation of the TI-ADC Model on FPGA

This chapter contains the details of the implementation of the TI converter and calibration algorithms discussed in chapter 2.

We start with a brief description of what is an FPGA before going into the details of the architecture of the model implemented in the FPGA, and the design considerations made during the implementation of each sub-module.

Any numeric consideration is referred to the particular implementation used to realize the results illustrated in chapter 4, which is a 10-bit 8-channel TI-ADC.

3.1. Development Environment

A Field-Programmable Gate Array (FPGA) is a type of digital IC designed to be configured after manufacturing (hence the field-programmable in the acronym). FPGAs are composed internally of logic blocks (the gate array), connected by a programmable interconnect matrix. The logic blocks (known as Configurable Logic Blocks (CLBs) in Xilinx's FPGAs) are able to implement the most different logic functions, and, in modern devices, they are also accompanied by blocks implementing specific functions like DSP[1] engines and memories. FPGAs also have plenty of distributed registers that can be used for the most diverse purposes (pipelining, accumulators, shift registers, etc.).

The configuration is usually described using HDLs in a similar fashion to digital ICs. This, together with the unlimited reconfigurability of modern FPGAs, makes the use of FPGAs for prototyping digital ICs quite obvious.

The model of the TI-ADC has been developed on an Arty A7-100T development board[2], (shown in figure 3.1).

[1] DSP: Digital Signal Processing consists in the elaboration of a signal in the digital domain.

[2] A development board is an off-the-shelf solution that can be used to test a design before mass production. They are also useful for familiarizing a designer with a particular device.

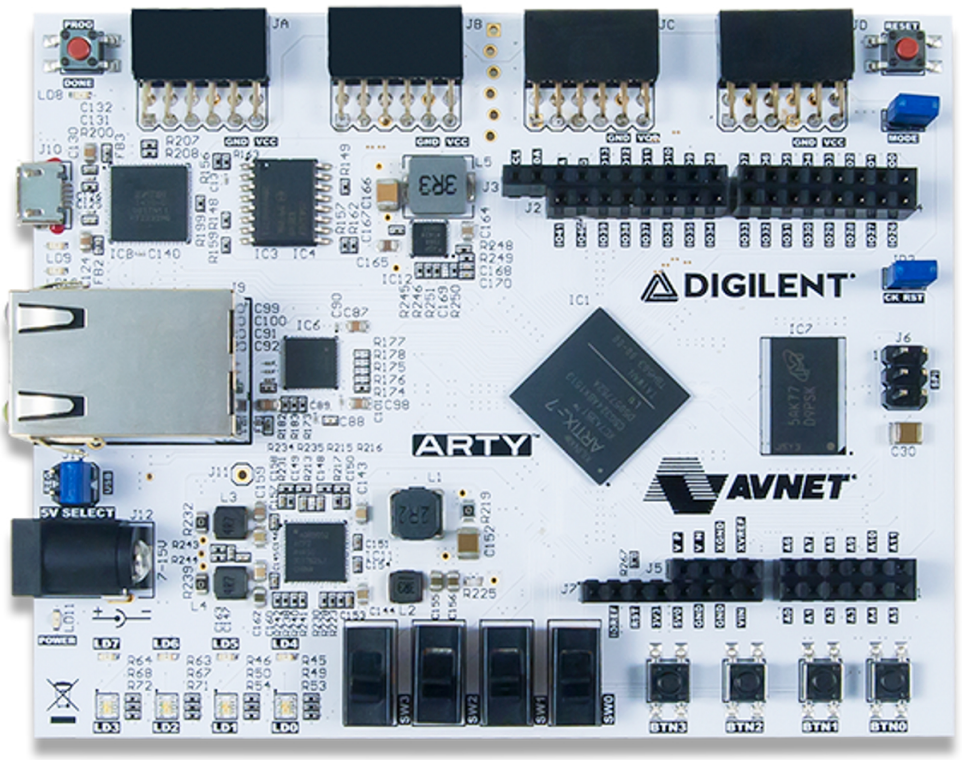

Figure 3.1: Top view of an Arty development board from Digilent [65].

This board is manufactured by Digilent. The resources available in the FPGA are shown in the Series-7 product lineup [66] and summarized in table 3.1.

	Artix-7-A100T
Total Block RAM (kb)	4.860
BRAM (36 kb each)	135
Total Distributed RAM (kb)	1.188
DSP	240
Slices	15.850
Logic cells	101.440
Flip-flops	126.800

Table 3.1: Summary of resources available on the Artix-7 A100T FPGA [65].

The Xilinx-A100T FPGA is part of the Artix-7 family [67] of Xilinx's Series 7 FPGAs. This means that we can easily adapt this project to other FPGAs of the same series like the Kintex-7 and Virtex-7 [68] without any change to the RTL. Furthermore, the project is easily editable to be implemented in newer generations of Xilinx's FPGAs like Ultrascale and Ultrascale+ as shown in Xilinx's migration guide [69].

The last aspect is particularly important because many of the design choices (i.e., the number of the channels of the TI converter, the resolutions of the ADC and of the Numerically Controlled Oscillator (NCO)) are constrained by the resources available on the FPGA (mainly the amount of available memory).

It is important to remember that this works aims to present only a proof-of-concept of the proposed verification technique. If a more capable FPGA is available, we could easily synthesize a higher resolution ADC model, NCO with better phase or amplitude resolution, or even a TI-ADC with more channels, and so on.

For instance, the most capable Series-7 FPGA, the Virtex-7 XC7VX1140T, has more than ten times the memory (67.8 Mb Vs. 4.8 Mb) of the Artix-7 A100T [66], whereas newer generations have orders of magnitude more.

Note that many development boards (and this is the case of the ARTY-A7 boards) also embed an external RAM[3] (SDR[4]-RAM, DDR[5]-RAM or HBM[6] in high-end devices). However, the time required to fetch data from external memory is orders of magnitude more than the one required to access internal memory (hundreds of clock cycles versus 1-2 clock cycles), introducing a severe performance bottleneck. For this reason, we are bound to use the internal memory.

More capable FPGAs might also be useful to fully exploit the throughput the FPGA TI-ADC model is able to provide. In fact, the effective throughput of the TI-ADC model is limited by the speed of the communication interface available. For a proof-of-concept this is not an issue, but it is absolutely required should this system be further developed into a practical verification instrument.

[3]RAM: a Random Access Memory is a type of computer memory that can be accessed in any order and whose access time is independent on the accessed address.

[4]SDR: Single Data Rate RAM is a type of RAM that works on a single clock edge (either rising or falling edge).

[5]DDR: Double Data Rate RAM is a type of RAM that works on both clock edges (rising and falling edges), doubling the bandwidth with respect to SDR-RAM for the same clock frequency.

[6]HBM: High Bandwidth Memory is a new generation computer memory made up of multiple DDR-RAM modules stacked-up in a single package. It is characterized by extremely large bus widths and bandwidths.

3.2. General Architecture

At a high level, the architecture of the on-FPGA model closely resembles the block diagram of the TI-ADC shown in figure 1.13. Like the real TI converter, the on-FPGA model consists of multiple channels as shown in figure 3.2.

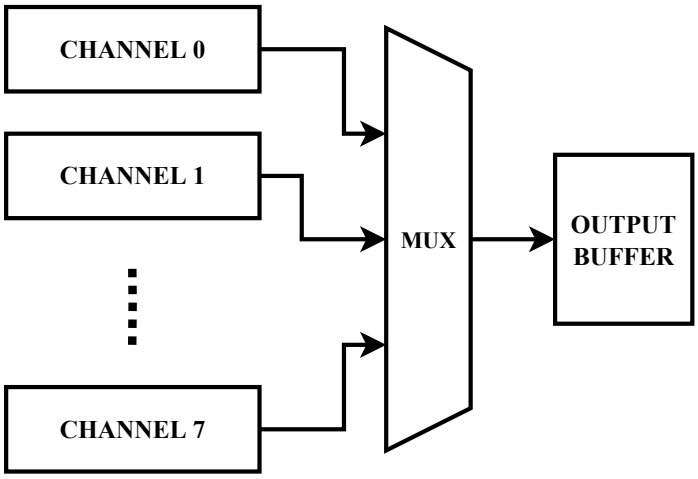

Figure 3.2: High-level block diagram of the on-FPGA model.

Differently from the real TI-ADC, in the on-FPGA model the channels work in parallel to leverage the advantages of an FPGA implementation. The samples are then ordered correctly by an opportunely managed multiplexer and stored in an output buffer.

The number of channels can be specified at the time of the RTL synthesis (like other specifications of the on-FPGA model). The design has been successfully synthesized and tested for different numbers of channels ranging from 4 to 16.

Each channel is functionally identical to the other. It is composed of an NCO, the ADC itself, and the Background Calibration Algorithm (BCA) block. The BCA itself is composed of two independent parts correcting the offset and gain, respectively. Figure 3.3 shows a high-level block diagram of the structure of each channel.

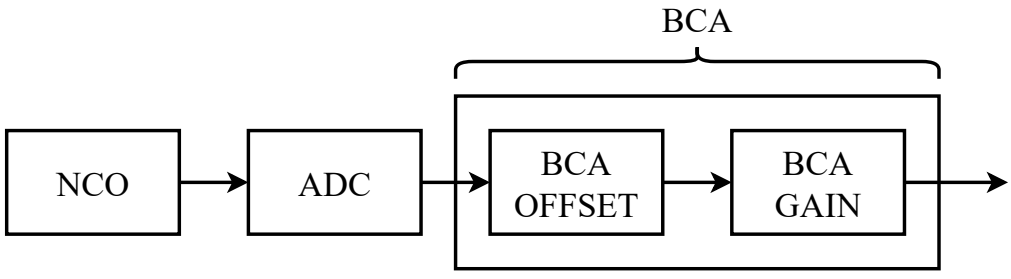

Figure 3.3: High-level diagram of each channel of the TI-ADC modelled on the FPGA.

A detailed description of each block of the processing chain will be provided in the following sections.

Aside from the aforementioned blocks composing the channels, there are also other auxiliary blocks needed to interface with and configure the on-FPGA model.

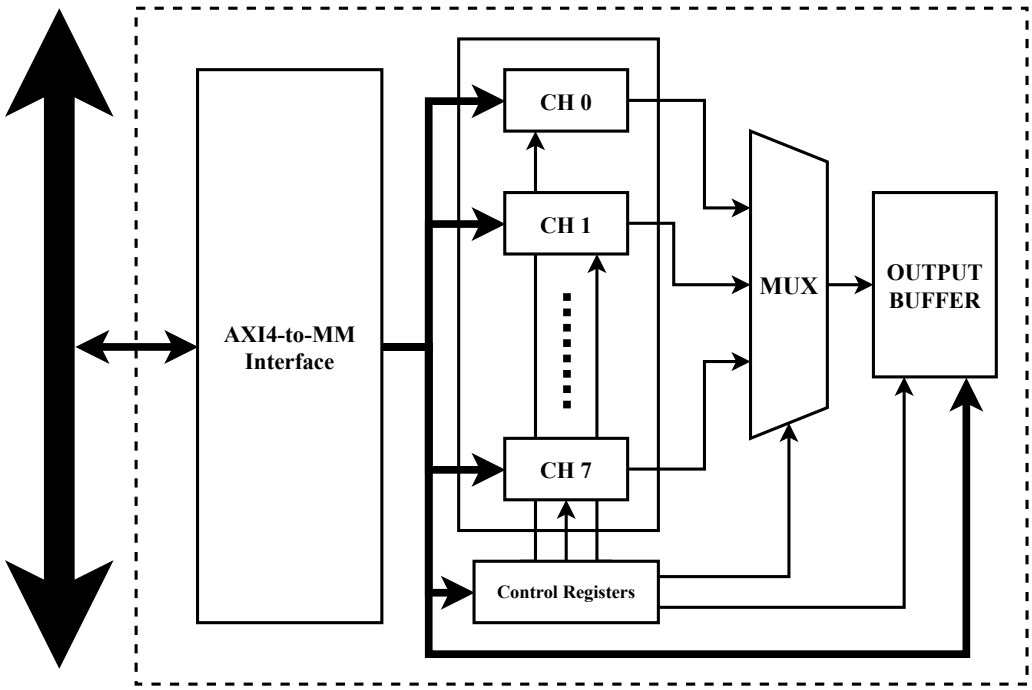

Figure 3.4: High-level diagram of the on-FPGA model showing auxiliary blocks.

In figure 3.4 we can see the different auxiliary blocks in the on-FPGA model, namely:

- AXI4-Lite to Memory Mapped interface: the external interface of the whole on-FPGA model translates the global AXI4[7] interface to a local Memory Mapped[8] (MM) interface.
- Control Logic: a set of memory-mapped registers that store the parameters of the processing chain.

All the blocks within the dotted line shown in figure 3.4 compose the on-FPGA model itself. There are other blocks implemented in the FPGA but those are third-party IPs and will be briefly analyzed in the next chapter.

[7]AXI4: the Advanced eXtensible Interface is an on-chip communication protocol developed by ARM. AXI is royalty-free and in the last few years has become the *de-facto* standard in the industry.

[8]MM: a Memory Mapped interface is a type of communication interface in which different peripherals are mapped by their memory address.

3.3. Numerically Controlled Oscillator Model

In a real ADC the input signal is analog and time-continuous, properties that are not reproducible in a fully-digital system like an FPGA (but the former can be approximated with arbitrary precision).

The idea is to replace the Sample and Hold (S&H) circuit that samples the analog, time-continuous signal into an analog, time-discrete one with an NCO, which directly provides a digital, time-discrete signal. A comparison of the two models is shown in figure 3.5.

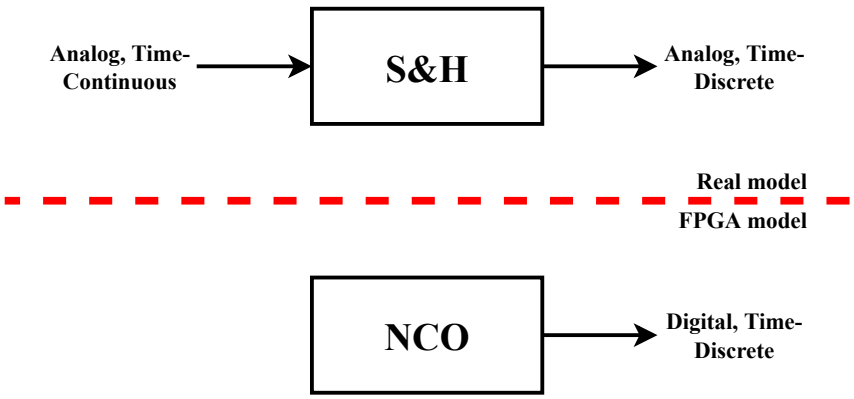

Figure 3.5: Comparison of the real and on-FPGA model of the input signals of an ADC.

AMD - Xilinx provides its own configurable NCO implementation known as Direct Digital synthesis [70], which is capable of synthesizing sinusoidal signals from 3 to 26 bits. However we decided to implement our own custom NCO to have better control over it (i.e., select the desired implementation) and closely couple it (i.e., have access to the whole internal state of the NCO) with the rest of the system.

Three different implementations have been considered for the NCO, before selecting the most appropriate one for the task:

- Unstable filter NCO
- Look-Up Table[9] (LUT) NCO
- Quarter-LUT NCO

Each different implementation has its own advantages and shortcomings. In the following sub-sections they are analyzed and explained in detail.

[9]LUT: a Look-Up Table is a table filled with the results of a particular computation. Instead of doing the computation at run-time, we look for the result stored in the table.

3.3.1. Unstable Filter NCO

The first type of NCO considered is an unstable filter NCO (also known as recursive filter NCO) [71–74], like the one shown in figure 3.6.

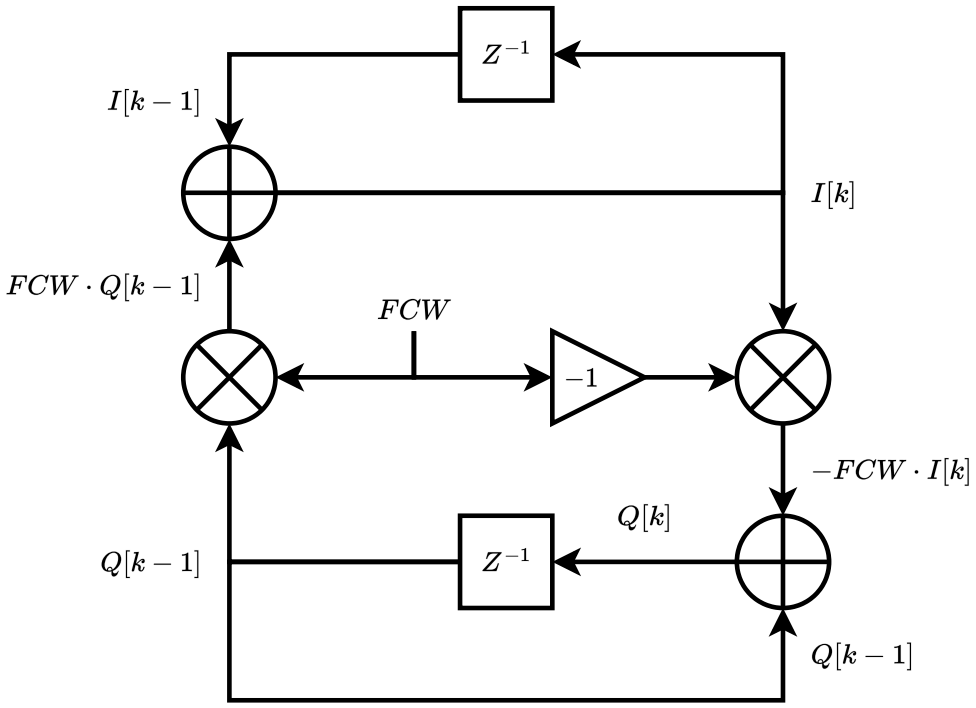

Figure 3.6: Block diagram of an unstable filter NCO.

This type of numerically controlled oscillator provides both the in-phase ($I[k]$) and the quadrature ($Q[k]$) sinewaves. It is possible to express the equations governing this NCO as a function of the Frequency Control Word (FCW), resulting in:

$$I[k] = I[k-1] + FCW \cdot Q[k-1] \tag{3.1}$$

$$Q[k] = Q[k-1] - FCW \cdot I[K]\,, \tag{3.2}$$

with $0 < FCW < 2$.

This implementation is quite convenient from a hardware perspective as it requires just two multipliers, two adders, and some registers, resources that are abundant in an FPGA. However it has a few major shortcomings, some depending on the architecture itself, and others related to the FPGA implementation.

First of all, the dependence of the frequency of the output tone on the FCW parameter is non-linear.

Another issue is that the amplitude of the sinewave depends on FCW. This implies that the SNDR depends on the FCW and thus on the frequency of the output tone.

Figure 3.7 shows the results of a sweep of FCW from 0 to 2 (excluded) for a 24-bit unstable filter NCO. On the left, we can see the non-linear FCW-frequency relationship, on the right we can observe the normalized amplitude of the output sinewave as a function of FCW.

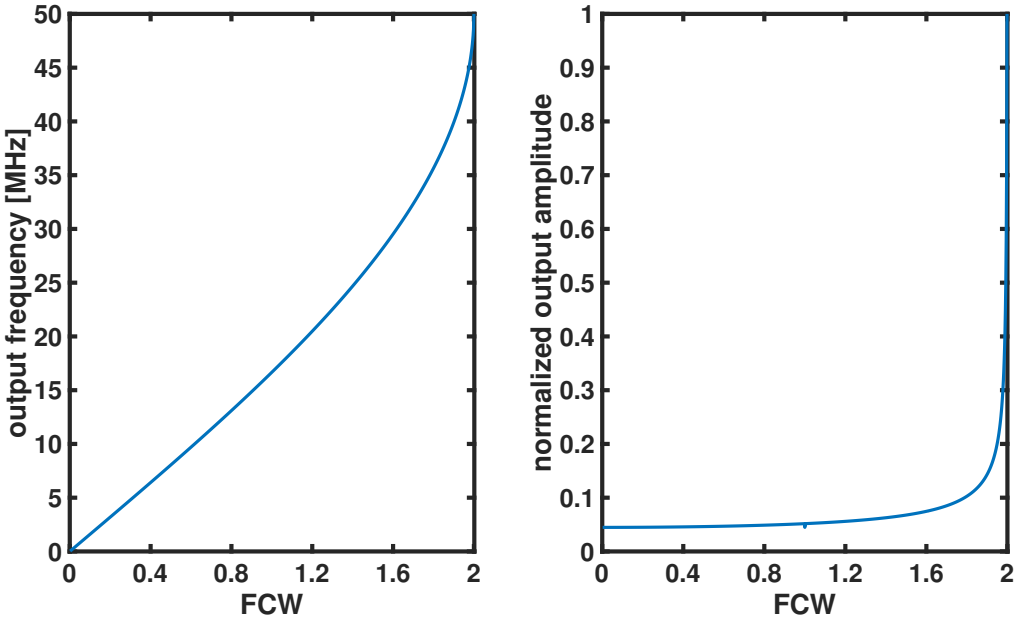

Figure 3.7: Output frequency (left) and normalized output amplitude (right) of a 24-bit unstable filter NCO simulated in MATLAB for FCW values between 0 and 2 (excluded).

Figure 3.8 shows the SNDR of the output tone of a 24-bit unstable filter NCO resulting from a sweep of FCW from 0 to 2 (excluded).

We can see how the ENOB of the output tone varies by more than 1 bit (6 dB in SNDR) across the FCW sweep. This alone is enough to discard the implementation. We need our NCO to provide a signal with a quality (i.e., SNDR, SFDR) as constant as possible to verify the calibration algorithms.

We can also see how the effective SNDR of the output tones is lower than the theoretical $24 \cdot 6.02\ dB + 1.76\ dB \approx 146\ dB$.

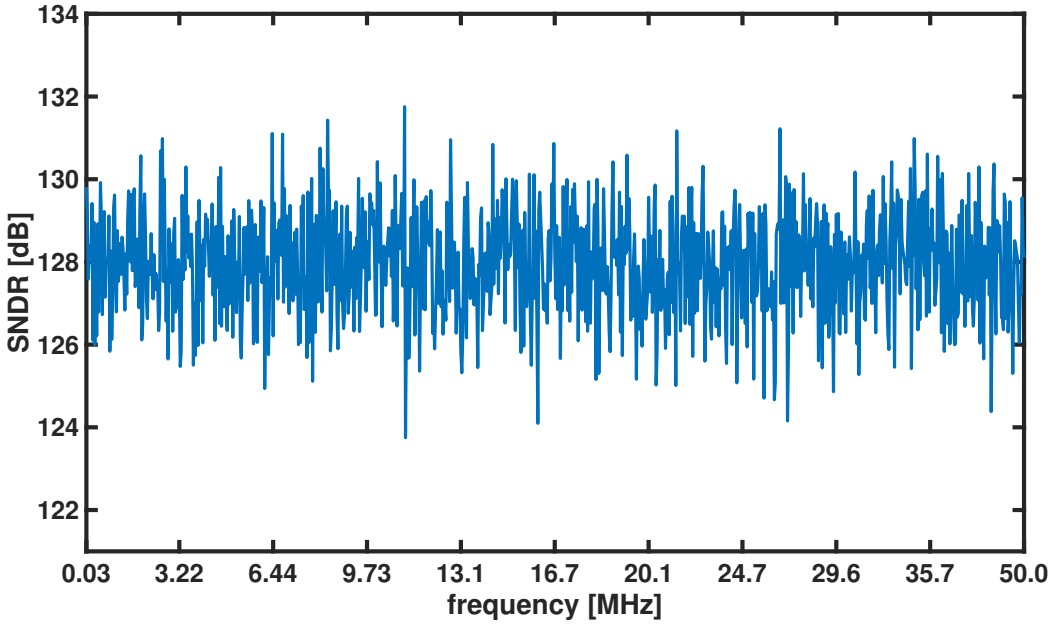

Figure 3.8: SNDR of a 24-bit unstable filter NCO simulated in MATLAB for FCW values between 0 and 2 (excluded).

Note that, for the purpose of the analysis of the oscillators, the sampling frequency f_S was considered equal to the clock frequency $f_{clk} = 100\ MHz$.

Let's focus now on the limitations related to the actual FPGA implementation.

The number of bits available for the multiplication is limited by the dedicated hardware we use to implement the operation. In Xilinx's Series-7 FPGAs products are implemented using DSP48E1 primitives. The DSP48E1 primitive that implements the multiplier in the FPGA is limited to 25 $\times$ 18 two's complement multiplications as shown in the DSP48E1 User Guide [75]. This means that we can multiply at most two numbers of 25 and 18 bits, respectively.

Though this resolution seems satisfactory (we target a 10-bit resolution for the ADC, thus a 25-bit resolution for the input sample is more than enough), we have to remember that the unstable filter NCO is based on a recursive approach. This means that the result of the multiplication at clock cycle k is used as an operand at clock cycle $k + 1$. Thus we need to truncate the result of the $I[k] \cdot FCW$ multiplication, introducing an error that gets integrated with the evolution of the filter. This means that the effective quality of the output tone is not only frequency-dependent but also time-dependent.

Because of these two limitations, we explored alternative implementations for the NCO.

3.3.2. LUT-based NCOs

Numerically Controlled Oscillators based on a LUT are probably the simplest implementation of a frequency synthesizer. The idea is to keep track of the phase and use a phase-to-amplitude converter to obtain the output waveform. The simplest phase-to-amplitude converter is a LUT in which we store the waveform samples.

Figure 3.9 shows the general block diagram of a LUT-based NCO. Note that this NCO has no constraints on the waveforms it can generate, as shown by the use of $f(\cdot)$ in the phase-to-amplitude converter. ϕ_S is the phase step.

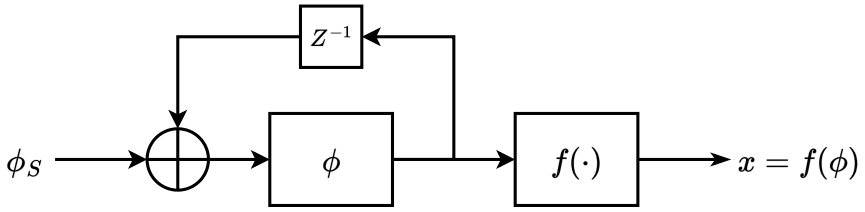

Figure 3.9: Block diagram of a generic LUT-based NCO.

Full-LUT NCO

In a Full-LUT NCO (or simply LUT NCO) a whole period of the waveform of interest (in our case a sine or a cosine) is stored in the LUT. A block diagram of simple Full-LUT NCO is shown in figure 3.10.

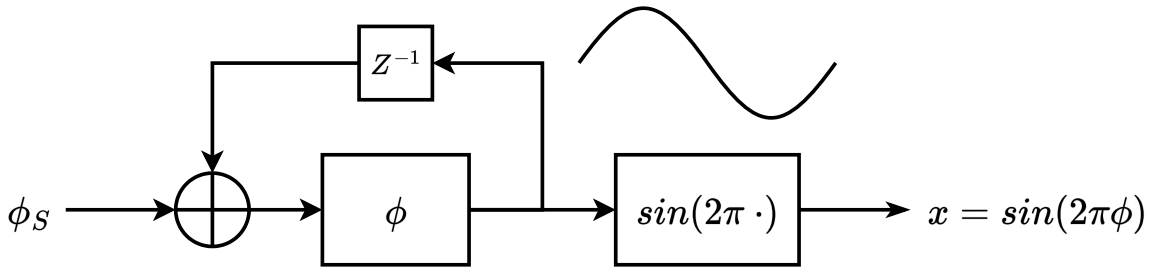

Figure 3.10: Block diagram of a Full-LUT sinewave NCO.

At each discrete time instant k, the output signal of the NCO will be:

$$x[k] = sin\left(2\pi \cdot \phi[k]\right) , \tag{3.3}$$

where we use $sin(2\pi\phi)$ instead of $sin(\phi)$ to express ϕ as a normalized phase. The reason why it is useful to have a normalized phase will be made clear later in this section.

The normalized phase (simply called phase from now on) represents the number of times we have gone around the unit circle as shown in figure 3.11.

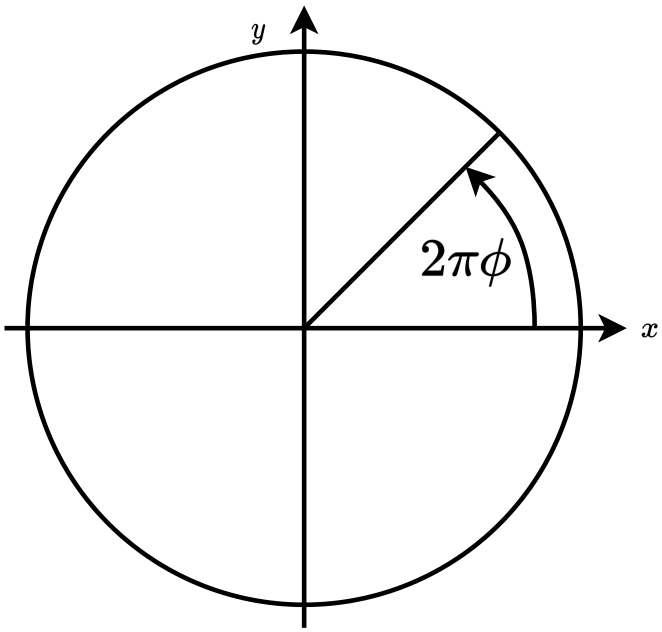

Figure 3.11: Unit circle and definition of normalized phase.

However, since the sine is a periodic function we don't care about how many integer times the sinewave went around the unit circle. This means we can discard the integer part of the phase accumulator as shown in figure 3.12.

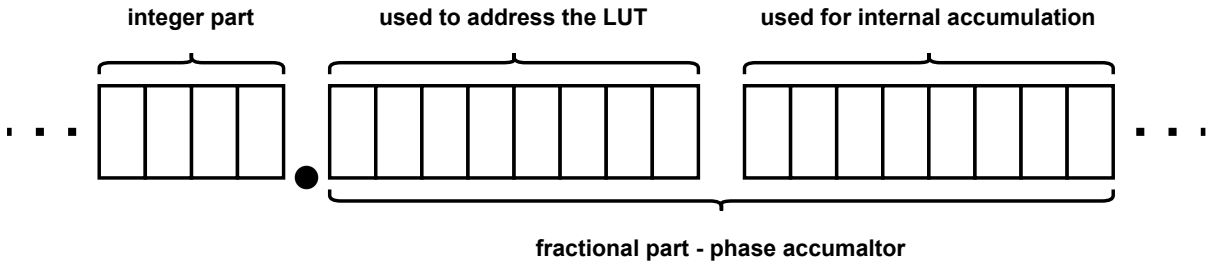

Figure 3.12: Structure of the phase accumulator.

Note that we don't even need to implement or store the integer portion of the phase as it is completely meaningless for our purpose.

It is also possible to decouple the number of bits used to store the phase from the number of bits used to address the LUT as shown in figure 3.12.

This allows for example to represent frequencies that are not achievable with integer steps through the LUT [76] as shown in figure 3.13.

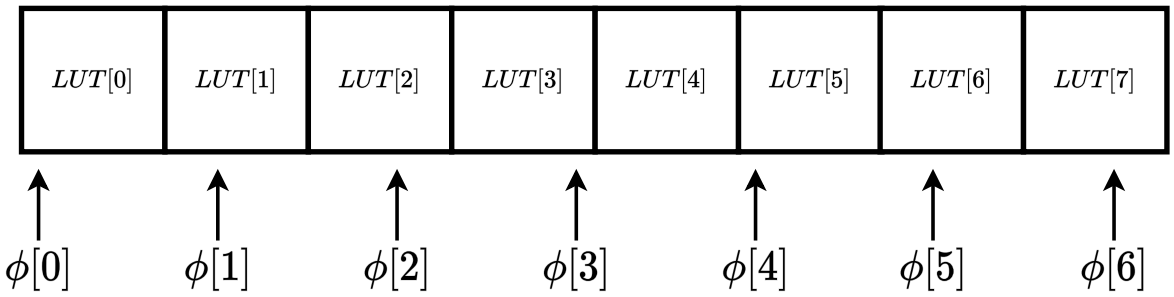

Figure 3.13: Representation of the fractional phase accumulation.

The extra fraction (the part not used to address the LUT in figure 3.12) accumulates causing a different addressed position with respect to the one we would have using only the addressing bits for the phase accumulator.

At each cycle, the phase is updated according to:

$$\phi[k] = \phi[k-1] + \phi_S \,, \tag{3.4}$$

ϕ_S being the phase step.

Any overflow of the fractional part is handled automatically by the unsigned arithmetic (this is why it is useful to have a normalized phase, otherwise we would have to subtract 2π after each period). By acting on the value of ϕ_S it is possible to select the frequency of the output sine tone as:

$$f_{out} = f_S \cdot \left(\frac{\phi_S}{2^{LUT\ BITS}} \right) \,, \tag{3.5}$$

where f_S is the sampling frequency and $LUT\ BITS$ is the number of bits used to address the LUT.

This implementation has the advantage of a linear relationship between the value of ϕ_S and the frequency of the output tone. Furthermore, in this case, the quality (i.e., SNDR, SFDR) of the output sinewave is independent of the ϕ_S chosen (and thus from the output frequency).

Again, the sampling frequency f_S has been considered equal to the clock frequency $f_{clk} = 100\ MHz$ for the analysis of the NCO.

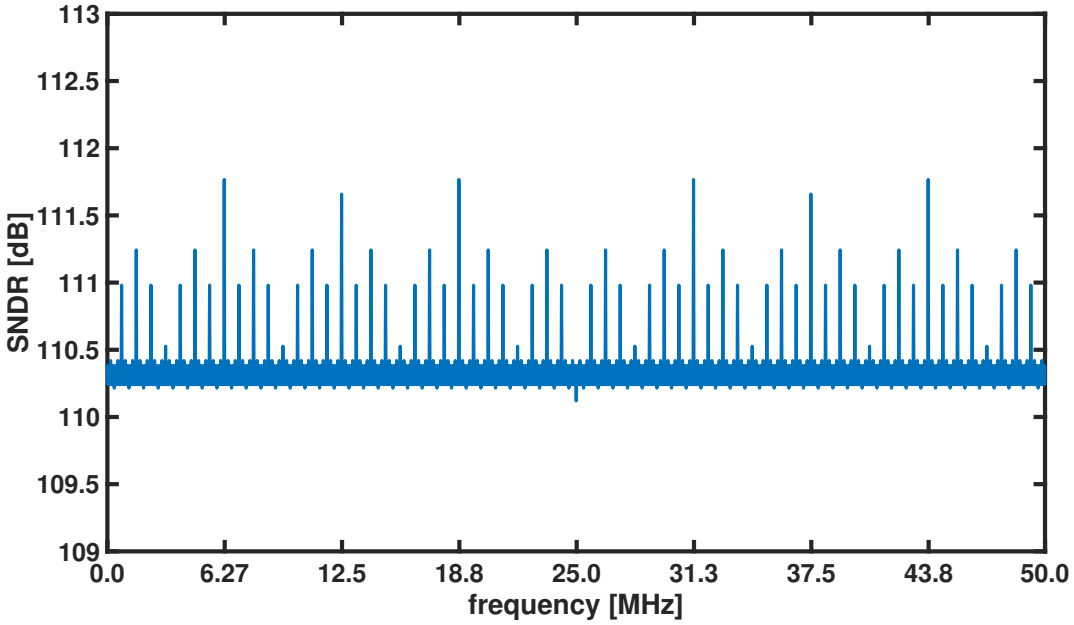

Figure 3.14: SNDR of the Full-LUT NCO simulated in MATLAB for output frequency ranging from 0 to 50.0 MHz.

The SNDR varies by about 1 dB across the whole sweep of frequency (0-50 MHz) as shown in figure 3.14, much better than with the unstable filter NCO shown in figure 3.8.

This NCO implementation has better and more consistent performances than the unstable filter one seen in the previous subsection. This is, however, paid with a more resource-intensive implementation. In fact, we need to store the sampled waveform, and each additional bit of phase resolution doubles the amount of memory required.

The most obvious hardware that can be used to implement a LUT is block RAM. Block RAMs are dedicated hardware resources that do not consume additional CLBs (as opposed to distributed RAMs, which are implemented using them). In Xilinx's Series-7 FPGAs block RAMs are available as BRAM36K [77]. A BRAM36K is a 36-kb memory array that can be configured with multiple aspect ratios (32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36, or 512 x 72). For write/read width larger than 8 bits we gain an additional bit for each byte resulting from the additional Error Correction Code[10] (ECC) bit. We can use this extra free bit to increase the resolution of the output sample. This, however, constrains the output samples to a resolution multiple of 9 bits. Given the target resolution for the ADC of 10 bits, an 18-bit resolution is enough to approximate an analog signal.

[10]ECC: an Error Correction Code is a redundant encoding of the data that allows detecting and/or correcting data corruption

Quarter-LUT NCO

A Quarter-LUT NCO (or QLUT NCO) is an improvement of the Full-LUT NCO described in the previous sub-section. The idea is to exploit the symmetry of the sine function to reduce the number of samples we need to store at the cost of some extra logic.

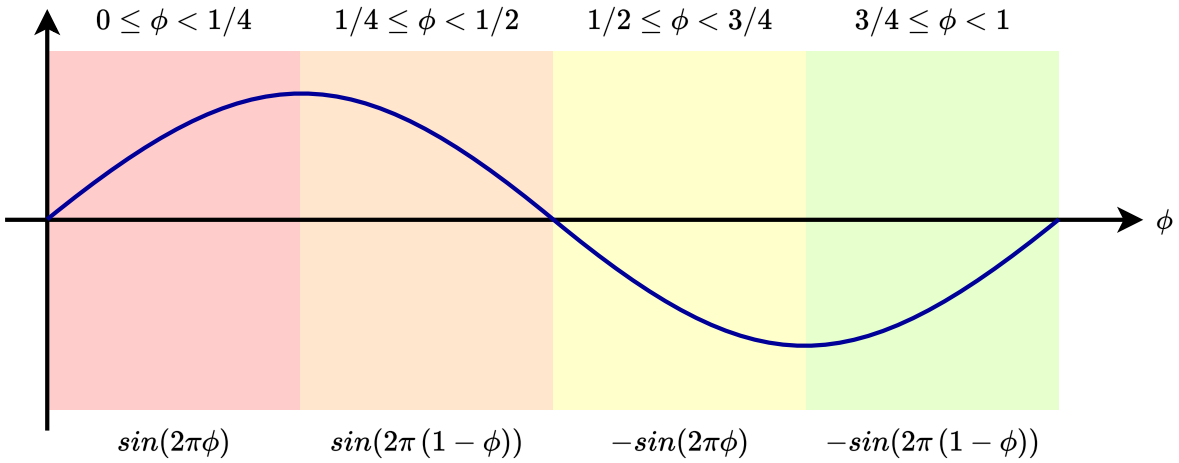

Figure 3.15: Partitioning of the sinewave according to the symmetry of the function.

Figure 3.15 shows how to partition the period of the sinewave to exploit its symmetry. By storing only a quarter of the period we are anyhow able to reconstruct the whole period.

To achieve this goal we need to change the sign of either the output sample or the lockup address according to the value of the phase. To be more precise, we use the two MSBs of the fractional part to identify the quadrant (or quarter) of the unit circle, whereas the remaining bits are used to address the LUT as shown in figure 3.16.

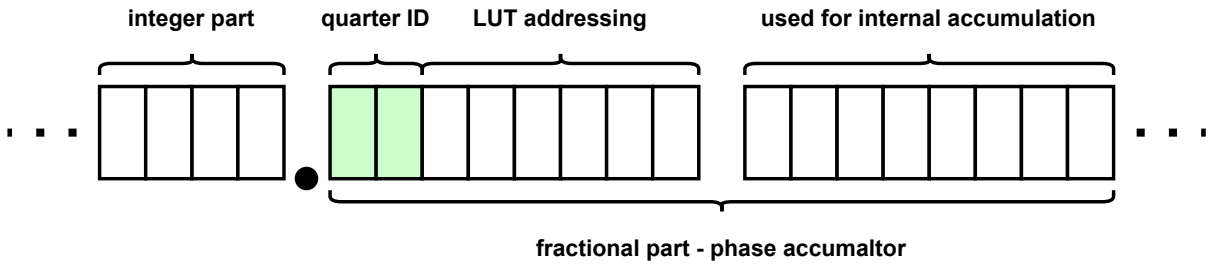

Figure 3.16: Structure of the phase accumulator of a QLUT NCO.

In the first quarter (that is, for $0 \leq \phi < 1/4$) the QLUT NCO behaves as the Full-LUT NCO, with the only difference being that we do not use the whole phase to address the LUT.

During the second quarter ($1/4 \leq \phi < 1/2$) we should use $1-\phi$ to address the LUT (with 1 being the last element of the LUT in a Full-LUT NCO). If we express it as a binary address, it results:

$$ADDR = LUT\ LENGTH - ADDR_\phi\,, \tag{3.6}$$

where $ADDR_\phi$ is the address of the element of the LUT indexed by ϕ.

Now, let's suppose that the length of the LUT is a power of 2. We can write $LUT\ LENGTH$ as a function of the number of bits used to address the LUT ($LUT\ BITS$), resulting in:

$$ADDR = 2^{LUT\ BITS} - 1 - ADDR_\phi\,. \tag{3.7}$$

However, in two's complement arithmetic this is equivalent to simply complementing the phase. In fact, to change the sign of a number we complement it and add 1:

$$-ADDR_\phi = \overline{ADDR_\phi} + 1\,, \tag{3.8}$$

which becomes:

$$-1 - ADDR_\phi = \overline{ADDR_\phi}\,. \tag{3.9}$$

Plugging equation 3.9 into equation 3.7, we get:

$$ADDR = \overline{ADDR_\phi}\,. \tag{3.10}$$

Thus, it is possible to flip the waveform by using a simple logic complement operation, which is easily implementable in an FPGA.

In the third quarter ($1/2 \leq \phi < 3/4$) we should use ϕ to address the LUT. This results in the same expression used in the first quadrant. The only difference is that we need to change the sign of the output samples.

Last, in the fourth quadrant ($3/4 \leq \phi < 1$) we should use again $1 - \phi$ to address the LUT. The result of this operation is the same as equation 3.10. Like in the third quadrant, in the fourth one we also have to change the sign of the output sample.

Table 3.2 summarizes the logic operations needed in a QLUT NCO.

Quadrant	Change S sign $[\phi[-1]]$	Complement $ADDR_\phi$ $[\phi[-2]]$
1^{st}	NO [0]	NO [0]
2^{nd}	NO [0]	YES [1]
3^{rd}	YES [1]	NO [0]
4^{th}	YES [1]	YES [1]

Table 3.2: Summary of logic operations needed to reconstruct the whole period of a sinewave from a single quarter of the period.

It is quite evident from table 3.2 that we can easily control the operations that have to be performed on the lockup address ($ADDR$) and on the output sample (S) using the two MSBs of the fractional part as shown in figure 3.16. In table 3.2 $\phi[-1]$ and $\phi[-2]$ represent respectively the first and second bits of the fractional part of ϕ (the 0^{th} bit is conventionally assumed to be the first bit of the integer part).

The performances of a QLUT NCO match the ones of the Full-LUT NCO and therefore are not reported here. Like the Full-LUT NCO, the SNDR of the output tone is much more stable than the one of the unstable filter NCO. The frequency dependence on the chosen phase step ϕ_S is still linear.

A Quarter-LUT NCO allows cutting the used memory by a factor of 4 while keeping the same phase resolution, or, vice versa, to gain 2 bits of phase resolution while using the same memory resources as a Full-LUT NCO. Still, this technique is less flexible than Full-LUT NCOs as it poses further constraints on the signal stored in the LUT as it will be clear in the next sub-section.

As for Full-LUT NCOs, also in the QLUT NCO the LUT is implemented using one or more BRAM36K. Thanks to the reduction in the number of samples that need to be stored, we can increase of a factor 4 the phase resolution of the NCO (in the same memory array we can fit four times the samples as before). Furthermore, in this case we need to store only positive samples since during the negative half of the sinewave we will directly change the sign of the output samples. This gives us one more bit of resolution for the NCO *gratis*, resulting in a 19-bit approximation of an analog signal.

The logic complementation of $ADDR_\phi$ and the change of the sign of the output samples required for this type of NCOs are easily implementable in the FPGA using Series-7 distributed CLBs [78] without using dedicated hardware like the DSP48E1 needed for the unstable filter NCO.

3.3.3. Comparison and Implementation

Comparison of LUT and QLUT NCOs

As mentioned in chapter 1, the ADC taken as an example for this work is designed for a transceiver application. Thus, it can be useful to generate input signals that are not ideal sinewave (like, for instance, a modulated signal). With a Full-LUT NCO, it is possible to load a period of an arbitrary waveform in the LUT and supply it to the ADC. In this case, the NCO behaves more as an input buffer than as a Numerically Controlled Oscillator.

Another, perhaps more interesting, capability of the Full-LUT NCO consists in the possibility to model the time-skew. Assuming we are interested in a single sine tone, we will have that the LUT is filled with:

$$LUT[ADDR_\phi] = sin\left(2\pi \cdot \frac{ADDR_\phi}{LUT\ LENGTH_{FLUT}}\right) . \tag{3.11}$$

Now, recalling that in our on-FPGA model we have M different NCOs placed in parallel, we can load a different waveform into each NCO:

$$LUT_m[ADDR_\phi] = sin\left(2\pi \cdot \frac{ADDR_\phi}{LUT\ LENGTH} + \delta\phi_m\right) , \tag{3.12}$$

where $\delta\phi_m$ represents the skew that affects each channel as modeled in chapter 2.

This gives us the possibility to assign an arbitrary skew to each channel. This is not possible in a Quarter-LUT approach as symmetry considerations restrict the values of $\delta\phi_m$ we can select [79] as shown in figure 3.17.

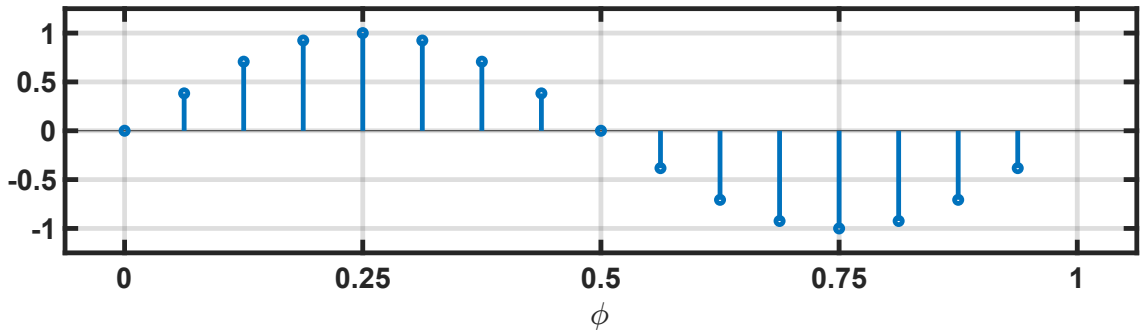

Figure 3.17: Effect of the phase quantization on the symmetry of the output waveform.

The sampled waveform shown in figure 3.17 is not compatible with a QLUT NCO, as the 5^{th} sample (or the 1^{st}, 9^{th} and 13^{th}) is not strictly part of any quadrant.

Instead, if we use $\delta\phi_m = \pi/LUT\ LENGTH$ we can recover the symmetry of the output waveform as shown in figure 3.18

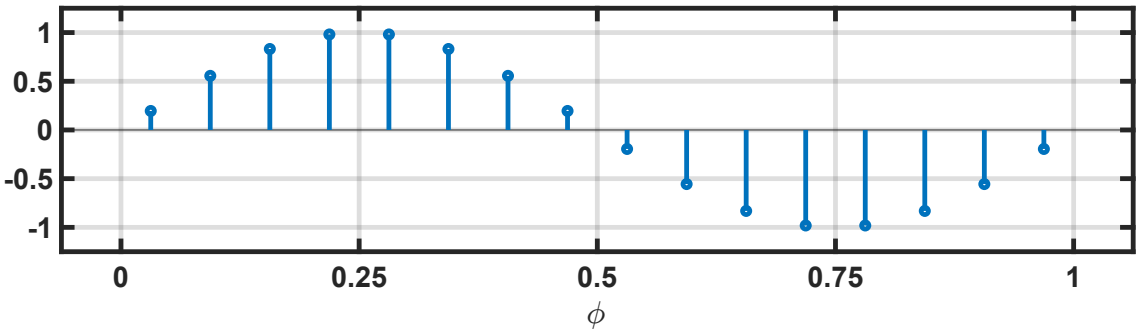

Figure 3.18: Effect of the phase quantization on the symmetry of the output waveform with a $\delta\phi_m = \pi/LUT\ LENGTH$ phase shift.

Perhaps, it is possible to better explain this concept using the unit circle.

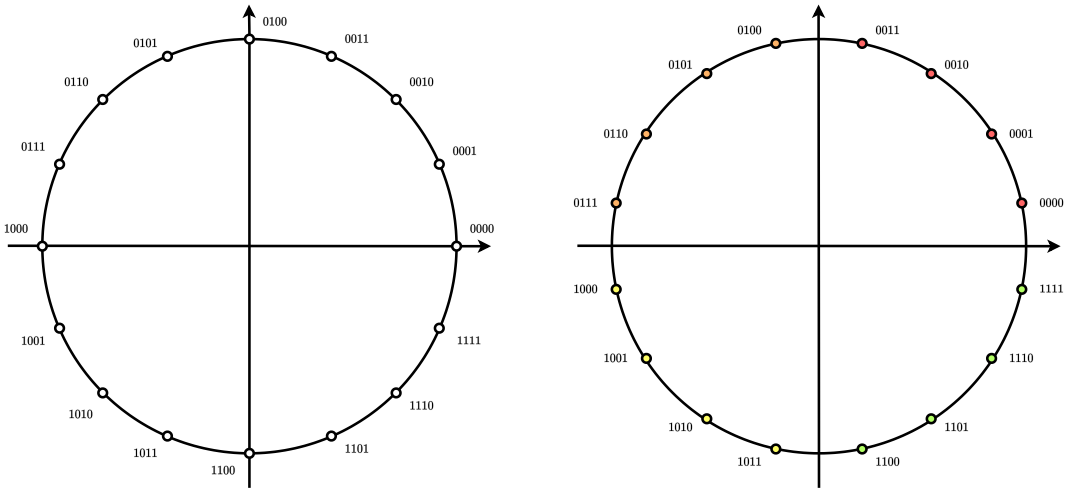

Figure 3.19: Unit circle representation of a 4-bit phase NCO for Full-LUT (left) and Quarter-LUT (right) NCOs.

Figure 3.19 shows the unit circle divided into 16 different steps, coherently with the sampled waveforms shown in figures 3.17 (left) and 3.18 (right) respectively. Introducing a time-skew $\delta\phi_m$ means adding a phase offset to the waveforms, which is equivalent to a clockwise rotation on the unit circle (for positive $\delta\phi_m$). However, while in a Full-LUT NCO the whole unit circle rotates clockwise, in a QLUT NCO we have that the first and third quadrants rotate clockwise, while the second and fourth rotate counterclockwise (vice versa for a negative $\delta\phi_m$), introducing a distortion in the output sinewave.

With a QLUT NCO, it is not possible to have a different value of $\delta\phi_m$ for each channel.

Acting on the initial values of the phase for each of the channels ($\phi_{0,m}$) it is possible to synchronize the sampling of the different channels. Thus, assuming a sinusoidal signal, the output of each channel will be:

$$S_{m,n} = sin\left(2\pi \cdot \frac{\phi_{0,m} + n \cdot \phi_{S,m}}{LUT\ LENGTH} + \delta\phi_m\right) . \tag{3.13}$$

Let's now consider $\delta\phi_m = 0$. We can choose $\phi_{0,m}$ such that:

$$\phi_{0,m} = \phi_{0,0} + \frac{\phi_{S,m}}{M} \cdot m\,. \tag{3.14}$$

Assuming $\phi_{0,0} = 0$ for the sake of simplicity and plugging equation 3.14 into equation 3.13, it results:

$$S_{m,n} = sin\left(\frac{2\pi}{LUT\ LENGTH} \cdot \frac{\phi_{S,m}}{M} \cdot (M \cdot n + m)\right) , \tag{3.15}$$

which closely resembles the equation describing the signal sampled by the TI-ADC.

Acting on the value of $\phi_{0,m}$ it is possible to model the time-skew also with QLUT NCOs. This, however, restricts the values of the skew that can be represented to discrete intervals with a maximum resolution equal to:

$$\delta\phi_{min} = \frac{2\pi}{LUT\ LENGTH}\,. \tag{3.16}$$

Assuming a reasonable size for the LUT equal to $LUT\ LENGTH = 2^{13}$ and recalling the relationship between phase and time delay ($\delta\phi = \delta t \cdot f_S$), it results that the minimum resolution at $f_S = 27\ MHz$ (the same frequency used for the simulations in chapter 2) is equal to:

$$\delta t_{min} = \frac{\delta\phi_{min}}{f_S} \approx 15\ ps\,, \tag{3.17}$$

which is too large for our purpose (the standard deviation used for the simulations reported in chapter 2 has the same order of magnitude).

Furthermore, this resolution is also frequency dependent.

Implementation of the NCO

Eventually, we decided to implement the NCO as a Full-LUT NCO. In fact, even if QLUT NCO offers unquestionable advantages in the resources required for the implementation, it also poses strict requirements on the signal that can be generated.

Taking into account that we need the Block RAM also to implement the LUT in the ADC and the output buffer, we decided to dedicate to the NCOs about a third of the memory resources available in the Xilinx-A100T FPGA (135 BRAM36Ks) [67]. Considering an implementation with 8 channels, each channel can use a maximum of $135/(3 \cdot 8) \approx 5$ BRAM36Ks. Recalling the 18 bits of resolution for the NCO mentioned when discussing the Full-LUT NCO, we end up with a maximum (rounding down to the nearest power of 2) of $2^{13} = 8192$ samples for each NCO. The use of other resources (CLBs, registers, etc.) is negligible.

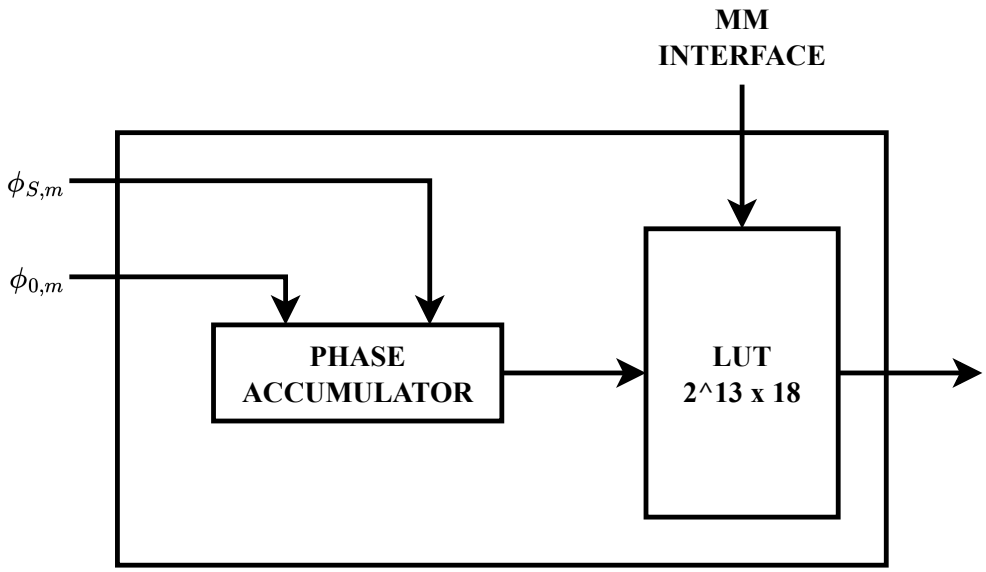

Figure 3.20: Block diagram of the RTL implementation of the NCO.

A block diagram of the implemented design is shown in figure 3.20. The initialization value of the phase ($\phi_{0,m}$) and the amplitude of the phase step ($\phi_{S,m}$) are stored within the control logic block and are configurable from the PC.

The LUT is implemented as a Simple Dual Port (SDP) RAM. This means that the RAM has two different ports: one, the read port is addressed using the phase, the other, the write port, is connected to the AXI4-Lite-to-MM interface, allowing the LUT to be written from the PC.

The output interface follows AMBA AXI4-Stream specifications [80].

3.4. Analog-to-Digital Converter Model

The Analog to Digital Converter is the second block in the channel flow diagram shown in figure 3.3.

A real ADC receives as input an analog, time-discrete signal, and provides its digital representation at the output. The on-FPGA model, on the other hand, receives a digital (albeit at higher resolution), time discrete signal, thus, the "quantization" procedure is in principle extremely simple as it is enough to discard a number of LSBs to model the effect of the quantization error. Figure 3.21 shows a high-level comparison of the two models.

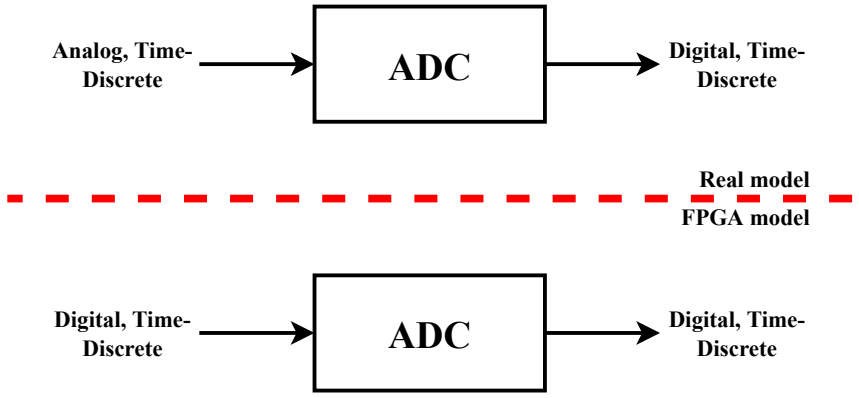

Figure 3.21: Comparison of the real and on-FPGA model of the ADC.

However, the aim of this work is to demonstrate a novel verification technique for calibration algorithms, thus we need some kinds of error or non-ideality to calibrate. In the previous chapter we showed the effects of four different non-idealities (offset mismatch, gain mismatch, timing skew, and nonlinearity of the single core). The time-skew is modelled within the NCO, whereas the remaining non-idealities are modelled within the ADC itself as shown in figure 3.22.

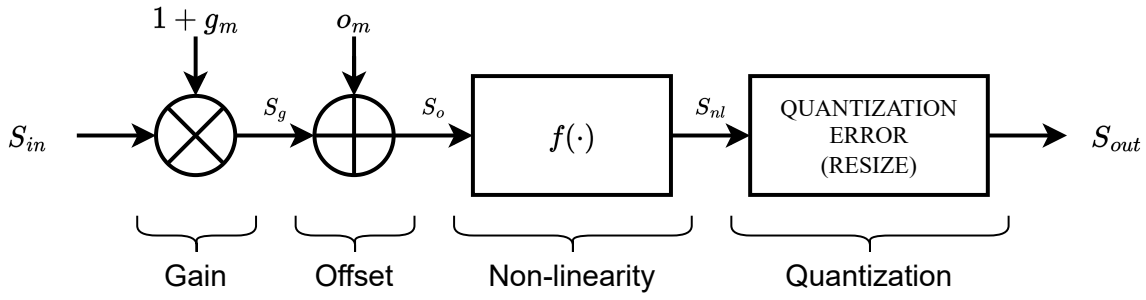

Figure 3.22: Block diagram of the on-FPGA model of the ADC.

The first modelled non-ideality is the gain, otherwise the offset would be multiplied by the gain coefficient too. In the previous chapter we have described the effect of the gain error using a $1 + g_m$ coefficient, i.e.,

$$S_g = S_{in} \cdot (1 + g_m)\,. \tag{3.18}$$

As mentioned in the previous chapter, in a SAR ADC the offset can be modelled as a global offset, that is, we need a single coefficient to model this error. This is simply done by adding an offset o_m to the sample:

$$S_o = S_g + o_m = S_{in} \cdot (1 + g_m) + o_m\,. \tag{3.19}$$

Regarding the implementation of the nonlinearity, we have considered two different approaches.

The first consists of using a series of multipliers to implement a Taylor series approximation of the characteristic:

$$S_{nl} = f(S_g) = S_g + k_2 \cdot S_g^2 + k_3 \cdot S_g^3 + k_4 \cdot S_g^4 + k_5 \cdot S_g^5 + \ldots\,. \tag{3.20}$$

This solution was quickly discarded due to the prohibitive number of multipliers required to implement a pipelined version of the operation for a reasonable degree of the approximate series. Furthermore, the limited width of the multipliers (25×18 as mentioned while discussing the unstable filter NCOs) poses hard limits to the achievable precision due to the truncation needed to cascade multiple DSPs.

A more elegant solution consists again in using a LUT in which we store the desired characteristic. In this case, the sample affected by nonlinearity is simply:

$$S_{nl} = f(S_g) = LUT(S_g)\,. \tag{3.21}$$

This allows the use of arbitrarily complex characteristics to model the nonlinearity. It is also possible to export an ADC characteristic taken from an EDA tool or even from a measurement of a real device and load it on the LUT to validate the calibration algorithms against more realistic scenarios. As for the NCO, this improved flexibility is paid with a greater requirement of resources (mainly the block RAMs).

Taking into account the three operations (in order, gain, offset, and nonlinearity), the sample at the output of the ADC can be written as:

$$S_{out} = LUT\left((1 + g_m) \cdot S_{in} + o_m\right) . \tag{3.22}$$

The effect of the quantization is modelled directly by designing the LUT with the desired output width. This is achieved by using a number of addressing bits larger than the data width of the LUT.

It is legitimate to ask at this point what is the need for the first two operations (gain and offset) if there is a LUT available. We could, in principle, simply load a characteristic already affected by an offset and a gain error into the LUT.

Though this solution obviously works, it makes impossible to separate the effects of the three non-idealities without requiring a new initialization of the verification run. Furthermore, changing the value of the offset or gain coefficient would require writing the whole LUT from the PC instead of a single value in a register.

3.4.1. Implementation of the ADC

The product used to model the gain effect is implemented using a DSP48E1 slice [75] that is capable of a 25×18 two's complement multiplications in a single clock cycle. Both S_{in} and $1 + g_m$ are signed numbers in two's complement representation.

Furthermore, the operation was implemented in such a way to saturate should the product of the two numbers exceed the maximum (or minimum) value that can be represented with an 18-bit two's complement representation. This introduces a small timing penalty but prevents distortion when amplitudes close to the maximum value are used.

The adder used to represent the offset is instead implemented in the FPGA using the CLBs that come with a dedicated carry-propagation logic to speed up the algebraic operations [78]. Both operands are signed numbers in two's complement representation. As with the product operation, the adder is also implemented to saturate should the result of the sum fall outside the values that can be represented with an 18-bit two's complement representation.

Nevertheless, a scale factor $k < 1$ is used when generating the samples for LUT of the NCO such that for reasonable values of offset and gain coefficients it is verified that $(1 + g_m) \cdot S_{in} + o_m$ is within the range of values that can be represented with an 18-bit two's complement representation.

Regarding the nonlinearity effect, the LUT is implemented using BRAM36Ks as done with the LUT of the NCO. Again, we have considered about a third of the total BRAM36Ks available on the FPGA for this task, resulting in the same 5 BRAM36Ks per channel as before.

In this case, the aspect ratio (i.e., the number of elements and the width of each element) of the memory array is constrained by the target resolution of the ADC. Furthermore, there is a trade-off between the resolution of the output sample (i.e., the resolution of the ADC) and the resolution of the input signal, which limits the number of bits of the input signal effectively used.

The resolution of the input signal is 18 bits (constrained by the NCO), and the first two operations in the ADC (gain and offset) are also 18 bits wide.

Let's now assume a target resolution for the ADC equal to 10 bits. Using all the 18 bits to address the LUT would require $2^{18} \cdot 10 = 2.6\ Mb \approx 71$ BRAM36Ks per channel. Such an amount of memory is unfortunately not available on the FPGA. Taking into account the 5 BRAM36Ks per channel budget calculated above, and the same 10 bits of resolution for the output sample, we end up with a maximum (rounding down to the closest power of 2) of $2^{14} = 16384$ size for the LUT.

Thus, we discard 4 LSBs from the input sample when addressing the LUT, and equation 3.21 becomes:

$$S_{nl} = LUT(S_g[17:4])\,, \tag{3.23}$$

where $[17:4]$ indicates that we take only bits form the 4^{th} to the 17^{th} of S_g (the LSB is the 0^{th} bit and the MSB is the 17^{th} bit).

It is possible to change those settings at the time of the synthesis by changing the value of some generics[11]. For example, it is possible to halve the number of channels and double the addressing space of the LUT without variation in the used resources.

Like the NCO's LUT, this LUT is implemented as an SDP RAM. The read port is addressed using the sample S_g. The write port is connected to the AXI4-Lite-to-MM interface allowing the LUT to be written from the PC as shown in figure 3.23.

The values of the offset and gain coefficients are stored within the control logic block and are configurable from the PC. Each operation can also be bypassed independently using multiplexers controlled by the control logic (not shown in figure 3.23).

[11]Generics: Generics are parameters of the entity that can be specified at the time of the synthesis.

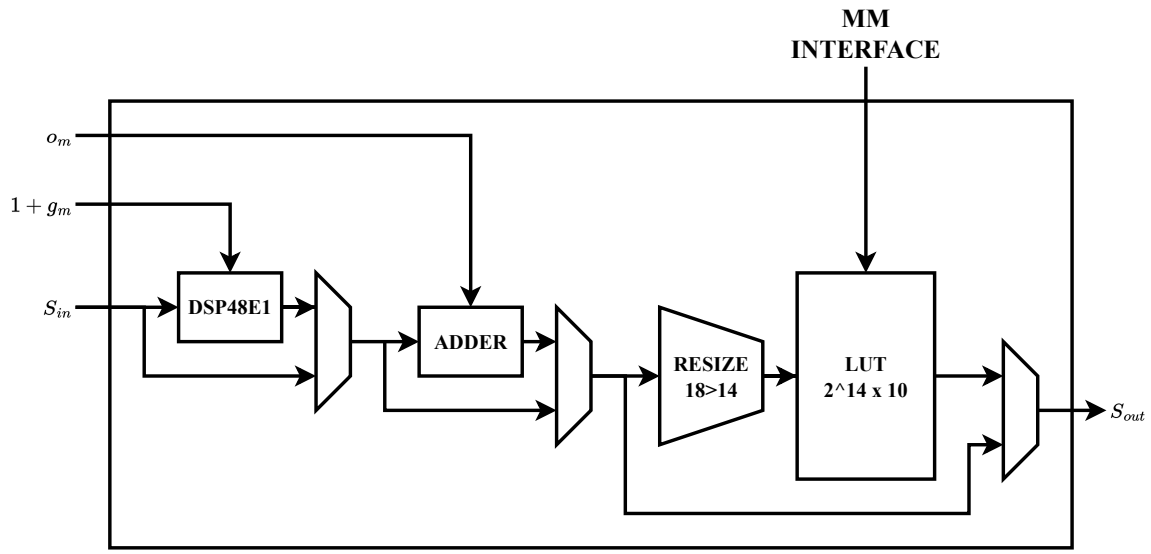

Figure 3.23: Block diagram of the RTL implementation of the ADC.

Both the input and output interfaces follow AMBA AXI4-Stream specifications [80] to implement the flow-control[12]. However, both interfaces have a width equal to 18 bits, which is different than the "multiple of 8 bits" width specified by the standard. This choice is due to the possibility of bypassing the processing done by the module, resulting in:

$$S_{out} = S_{in}\,. \tag{3.24}$$

Thus, the output interface must be able to represent all the signals that the input interface is able to represent.

This capability is provided in all the modules along the processing chain (except for the NCO), both as a debugging tool and to provide a benchmark against which we can compare the calibration algorithms.

The output sample is aligned to the MSB (i.e., the LSBs are filled with zeros). This is useful because it extends the output sample, operation required by the Background Calibration Algorithms.

Each ADC entity requires (for this specific configuration) 5 BRAM36Ks, a single DSP48E1, 106 registers, and 56 slices[13] LUTs.

[12]flow-control: flow-control is a process between two communication nodes that regulates the data rate through the interface.

[13]Slice: in Xilinx's Series-7 FPGAs a slice is a hierarchical unit below the CLB [78]. each CLB is composed of two slices that can be used independently. Each slice contains 4 LUTs and 8 registers.

3.5. Background Calibration Algorithms

Differently from the two previous blocks, the BCAs are completely digital even in a real digitally calibrated TI-ADC, thus there isn't any need to find a way to model or approximate its working. In this section, we provide a brief description of the HW implementation of the two algorithms discussed in the last section of chapter 2.

To simplify the implementation, the two algorithms (offset and gain) are implemented as a series of two different entities rather than as a single monolithic one, as shown in figure 3.3.

Both blocks use an 18-bit wide modified AXI4-Stream as input and output interface, implementing a complete flow control system using the AXI4-Stream Valid/Ready handshake [80].

3.5.1. Offset Background Calibration Algorithm

The Offset BCA implements the offset calibration algorithm explained in chapter 2, whose basic equations (2.36 and 2.38) are reported below for the sake of simplicity:

$$S_{m,corrected}[k] = S_{m,non\ corrected}[k] - \gamma_{offset} \cdot o_m[k] \tag{3.25}$$

$$o_m[k] = o_m[k-1] + S_{m,corrected}[k-1] - S_{reference}[k-1]\,, \tag{3.26}$$

with γ_{offset} being a negative power of 2 to transform the hardware-intensive division operation into a simple bit-shifting one.

Equation 3.26 represents an accumulator, which is implemented with a 32-bit width, whereas γ_{offset} is fixed at the time of the synthesis.

The effects of the value of γ_{offset} on the stability and convergence of the calibration algorithm have been analyzed in chapter 2.

The two adders are implemented within the FPGA exploiting the dedicated carry-propagation resources of the CLBs [78].

Figure 3.24 shows a high-level RTL representation of this entity.

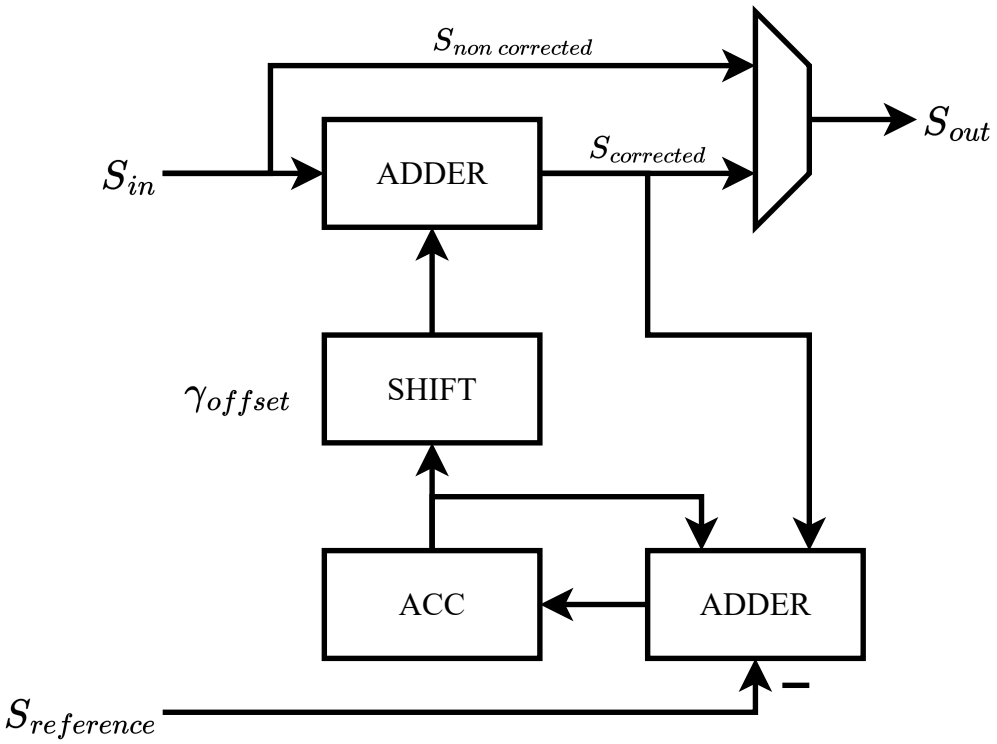

Figure 3.24: Block diagram of the RTL implementation of the offset BCA.

As for the ADC, it is possible to completely bypass the block, so that $S_{out} = S_{in}$.

3.5.2. Gain Background Calibration Algorithm

The Gain BCA implements the gain calibration algorithm shown in chapter 2. The equations (2.39 and 2.40) describing its working principle are reported below for the sake of simplicity:

$$S_{m,corrected}[k] = S_{m,non\ corrected}[k] \cdot \gamma_{gain} \cdot g_m[k] \tag{3.27}$$

$$g_m[k] = g_m[k-1] - |S_{m,corrected}[k-1]| + |S_{reference}[k-1]| \ , \tag{3.28}$$

with γ_{gain} being a negative power of 2 for the same reason of γ_{offset}. The accumulator is still implemented with a 32-bit width and γ_{gain} is fixed at the time of the synthesis.

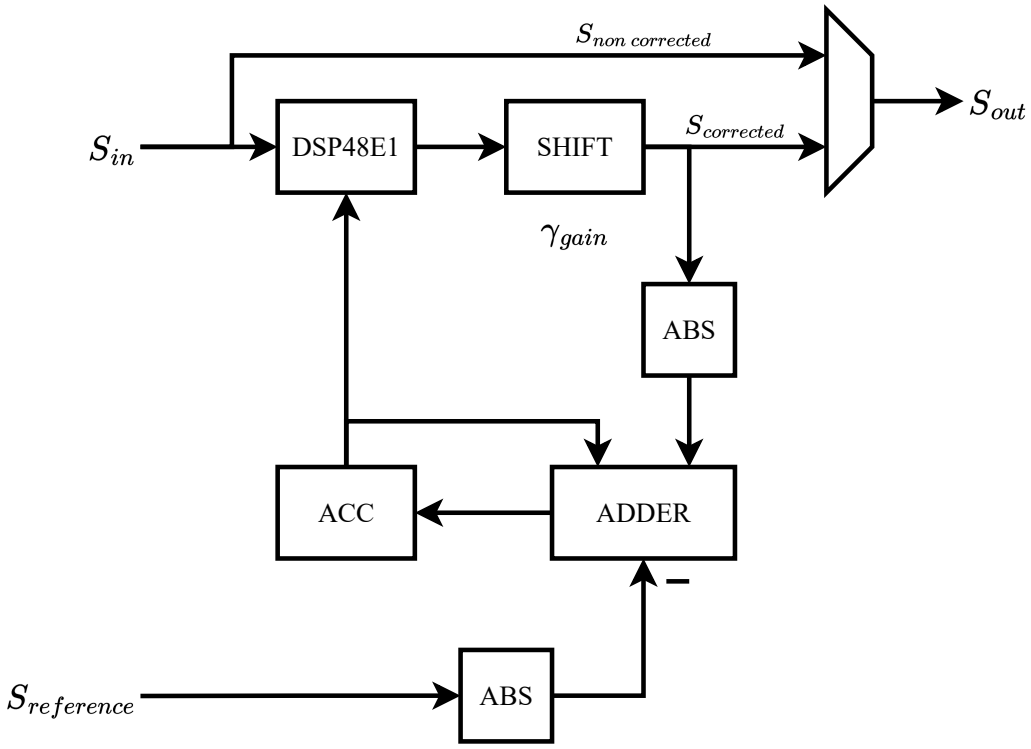

Figure 3.25: Block diagram of the RTL implementation of the gain BCA.

The shift operation is moved after the multiplication (i.e., the DSP) as shown in figure 3.25 (in the offset BCA the shift is done before the sum). This is done to improve the resolution of the system as the LSBs that would have been discarded now contribute to the final result.

This however results in operands wider than the 25 bits allowed by the DSP48E1 [75] (to be more precise the operands are respectively 32-bit and 18-bit wide). This issue is solved by combining two DSPs to implement a wider multiplier. The multiplication can be split into a sum of smaller operations opportunity shifted, which is:

$$P = A \cdot B = (A_H + A_L) \cdot (B_H + B_L) = A_H \cdot A_H + A_H \cdot B_L + A_L \cdot B_H + B_L \cdot A_L \,, \tag{3.29}$$

where the subscripts H and L indicate respectively the upper and lower bits of A and B.

In this case, the operand B is S_{in}, which is an 18-bit number, thus $B_H = 0$ and only 2 DSP48E1s are needed for each gain BCA.

As for the offset BCA, this block can be bypassed entirely, so that $S_{out} = S_{in}$

3.6. Auxiliary logic

Aside from the three entities analyzed above that compose each of the channels in the on-FPGA model, there are also other entities that are needed to configure and communicate with the system. In the following sub-sections, these entities are briefly analyzed.

3.6.1. AXI4-Lite to Memory Mapped Interface

This entity implements a bridge between the AXI4-Lite bus used in the FPGA (but externally to the on-FPGA model) and a custom Memory Mapped bus internal to the on-FPGA model. AXI4-Lite is a lightweight version of the popular AMBA AXI interface [81]. The choice of this interface is due to the fact that is the *de facto* industry standard.

The AXI4 (Full spec or lite) is made of five different channels:

- Write Address (WA) channel
- Write Data (WD) channel
- Write Response (WR) channel
- Read Address (RA) channel
- Read Data (RD) channel

The first three channels (WA, WD, and WR) are part of the write interface, whereas the last two (RA and RD) constitute the read interface. The two interfaces are independent of each other and it is possible to read and write concurrently.

The write interface waits for a valid transfer of the address (WA) and data (WD) before converting the received address to an internal one and starting the write operation on the MM interface. The response (WR) is used to signal the outcome of the operation to the master. The WD transaction[14] can occur at the same clock cycle or at a later one with respect to the WA transaction, but not before as specified in the AMBA AXI4 Protocol Specification [81].

The read interface waits for a valid transfer of the address (RA) before converting it to an internal address and starting the read operation on the MM interface. Once the read operation is carried out, the resulting datum is provided on the data channel (RD).

The translation mechanism of the address is the same for both write and read transactions. Figure 3.26 shows the translation scheme used in this project.

[14]transaction: a transaction is an interaction between two elements on the bus.

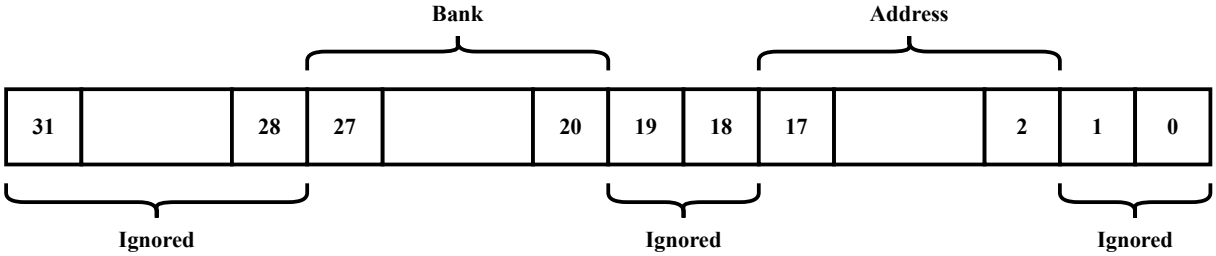

Figure 3.26: Scheme of the address translation performed by the AXI4-Lite to MM interface.

The bits extending outside the address space of the whole on-FPGA model (bits 28-31) are simply ignored (actually transactions that fall outside the address space of a slave are automatically discarded by the AXI interconnect). The remaining bits are used to select the right bank (bits 27-20) and address (bits 17-2). Bit 1-0 are discarded because AXI4 is byte addressed, whereas the on-FPGA model is word-addressed internally.

Each memory space within the on-FPGA model exposed to the MM interface is selected using a dedicated enable signal, whereas the address and data buses (separate write and read busses) are shared by all the modules.

A map of the address space of the on-FPGA model is shown in figure 3.27. Sections colored in gray fall within the address space of the on-FPGA model but are not mapped to any module.

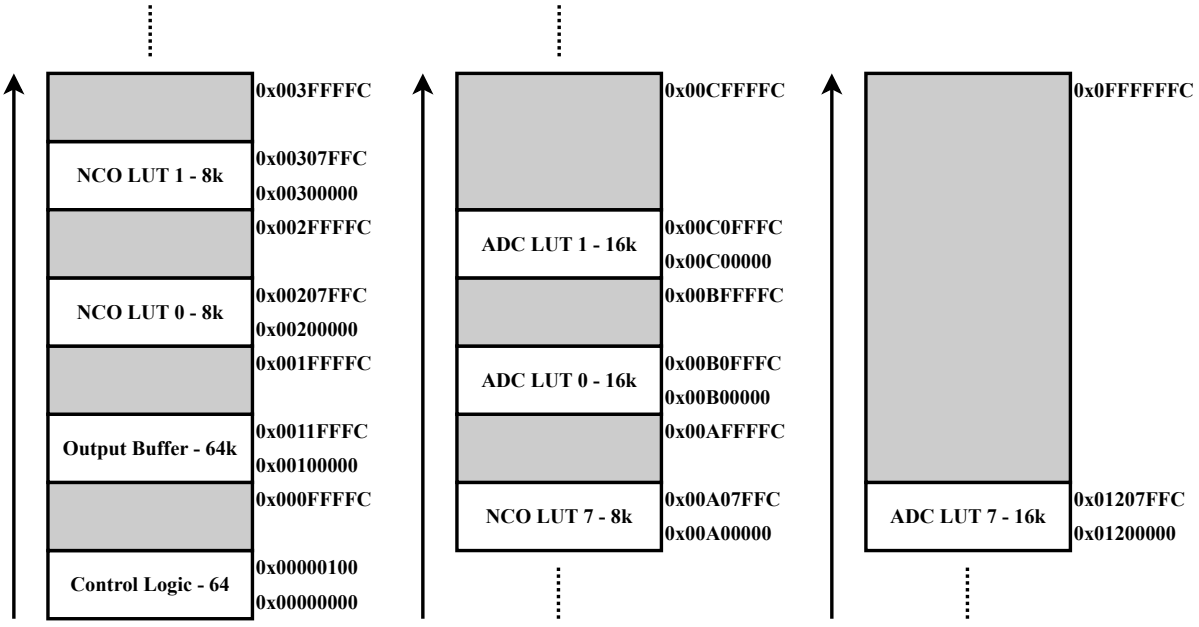

Figure 3.27: on-FPGA model address space.

3.6.2. Control Logic

The control logic is a memory-mapped set of registers that can be accessed from the PC to set the parameters of the verification run.

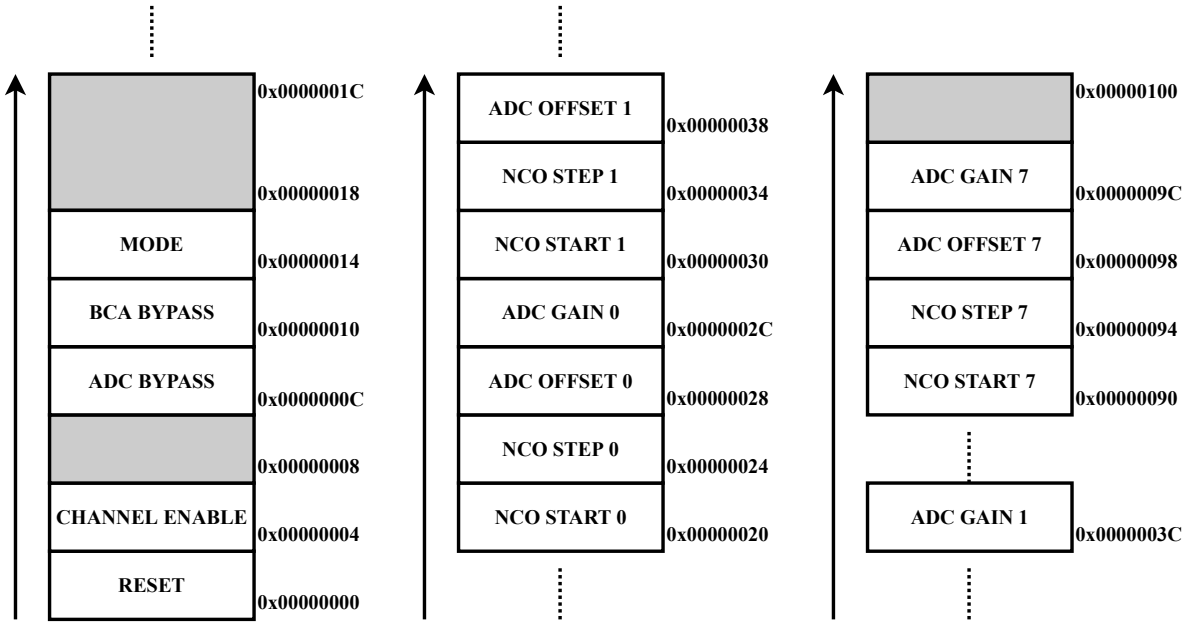

Figure 3.28: Control registers address map.

A map of the address space of the registers is shown in figure 3.28. Sections colored in gray exist but are not connected to any entity in the on-FPGA model. Each register can be accessed by the memory-mapped interface from the PC both to be read or written.

Note that not all the bits of all the registers are used. For instance, only the first bit of the **RESET** register is used to reset the on-FPGA model. The **CHANNEL ENABLE** register exploits a number of bits equal to the number of channels in the particular implementation (in this case 8). Each bit enables the respective channel. This is useful, for example, if we want to simulate a TI-ADC with a number of channels smaller than the synthesized one without starting a new synthesis.

ADC BYPASS and **BCA BYPASS** contain, respectively, the bypass selection bit for the ADC (gain bypass, offset bypass, LUT bypass) and for the BCA (gain calibration bypass and offset calibration bypass). **MODE** is used to select the operating mode of the output buffer.

The other registers store the configuration values for the NCO (**NCO START** and **NCO STEP**) and for the ADC (**ADC OFFSET** and **ADC GAIN**) of each channel.

3.6.3. Output Buffer

The output buffer stores the samples processed by the on-FPGA model before they are read by the PC. It consists of a multiplexer (or reorderer) that orders the output of the channels and a memory that stores them.

Like every other block along the processing chain, the blocks communicate using a slightly modified version of the AXI4-Stream interface.

Reorderer

The reorderer is essentially an opportunely controlled multiplexer with dedicated control logic. A block diagram of the implementation is shown in figure 3.29.

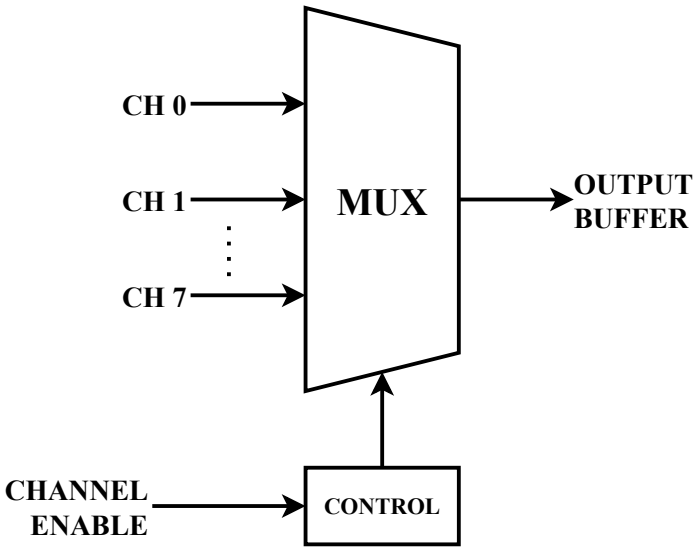

Figure 3.29: Block diagram of the RTL implementation of the reorderer.

In principle, at each clock cycle, each of the channels is able to provide a new sample. The flow-control mechanism implemented in each entity blocks the channel when the master (i.e., the output buffer) is not ready to receive the data (for instance, because it is full).

The control logic selects which of the different channels the multiplexer has to connect to the output buffer. If all the channels are enabled the multiplexer iterates through the channels one at a time. If one or more channels are disabled, the reorderer ignores them.

The control logic waits for a valid transaction (READY and VALID high at the same rising edge) before switching to the next channel.

MM-FIFO

The MM-FIFO[15] is the effective memory of the output buffer. We decided to dedicate about a third of the 135 BRAM36Ks available on the FPGA to the output buffer, resulting in about 45 BRAM36Ks.

Actually, there is a more stringent constraint that limits the amount of BRAM36Ks we can cascade. In fact, it is possible to cascade only BRAM36Ks within the same "column" [77]. The Artix-7-A100T at disposal has a maximum of 40 BRAMs per column. This leaves us with a maximum of (rounding down to the closest power of two) $2^{16} = 65536$ samples. This number of samples is satisfactory for the characterization of an ADC.

In fact, the FFT of an N-sample signal has a frequency resolution equal to:

$$\Delta f = \frac{f_S}{N}\,, \tag{3.30}$$

where, in our case, $f_S = f_{clk}$. If we consider a target clock frequency of $f_{clk} = 100\ MHz$, equation 3.30 becomes:

$$\Delta f_{FFT} = \frac{f_{clk}}{N} = \frac{100\ MHz}{2^{16}} \approx 1.5\ kHz\,, \tag{3.31}$$

whereas the Numerically Controlled Oscillator has a frequency resolution equal to:

$$\Delta f_{NCO} = \frac{f_{clk}}{LUT\ LENGTH} = \frac{100\ MHz}{2^{13}} \approx 12.2\ kHz\,, \tag{3.32}$$

which is bigger than the resolution of the FFT. Thus, the number of samples is satisfactory for our purposes.

The memory (and thus the on-FPGA model) can operate in two different modes:

- Standard mode (or FIFO mode);
- Continuous mode (or circular buffer mode).

When working in FIFO mode, the memory is filled with the samples coming from the processing chain (i.e., the channels). Once the memory is full, the on-FPGA model is stopped until the memory is emptied by reading it from the PC. At this point, the on-FPGA model resumes, and the output buffer is filled with the next 2^{16} samples.

[15] FIFO: a First-In-First-Out is a type of memory in which the samples are read in the same order they are written

When working in circular buffer mode, the memory is still filled with the samples from the on-FPGA model, but now, once the memory is full, the on-FPGA model is not stopped. The oldest sample is overwritten without stopping the process.

This is achieved by moving forward both the write and read pointers as shown in figure 3.30.

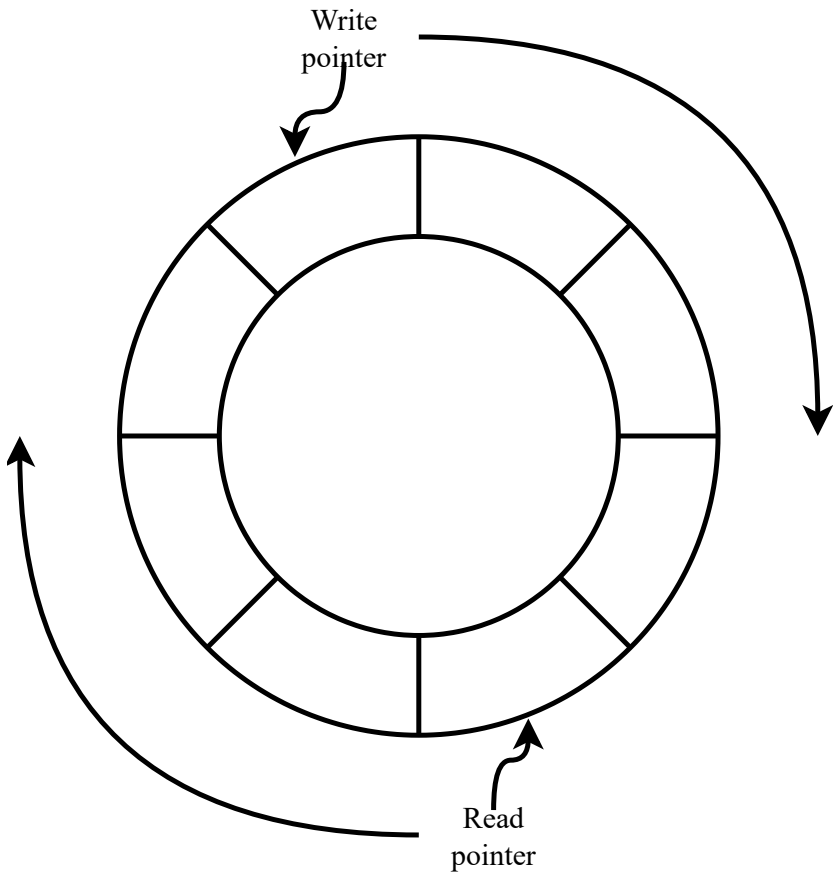

Figure 3.30: Example of the behaviors of the pointers in a circular buffer.

In a standard FIFO, two pointers, the read pointer (RP) and write pointer (WP), are used to keep track of the data. When a datum is written, WP is increased by 1, and when a datum is read, RP is increased by the same amount. By calculating the difference $OCC = WP - RP$ it is possible to know the occupancy (OCC) of the FIFO. If the occupancy is 0 the FIFO is empty, and vice versa, when the occupancy is equal to the size of the FIFO, the FIFO is full. In a classical FIFO, when OCC is equal to the size of the FIFO further write operations are forbidden (i.e., the FIFO signals upstream that it is not ready to accept new data). In a circular buffer, write operations continue being accepted and RP is increased together with WP, overwriting the oldest datum.

This last working mode is useful to fully exploit the throughput of the on-FPGA model. In fact, as it will be explained in the next chapter, the throughput of the on-FPGA model when working in FIFO mode is severely restricted by the speed of the communication interface with the PC.

4 | System-level Architecture and Results

This chapter is divided into two main sections.

The first section consists of a description of the system architecture and the communication interface. The system architecture includes all the third-party IPs implemented on the FPGA, which is everything that was not described in chapter 3.

The second part of the chapter lists the results of this *thesis*. The results obtained with the on-FPGA model are compared first with the MATLAB model of the TI-ADC and the calibration algorithms, and then with VHDL simulations.

4.1. System Architecture

This section contains a description of the design implemented on the FPGA, focusing on the third-party IPs that surround the on-FPGA model of the TI-ADC.

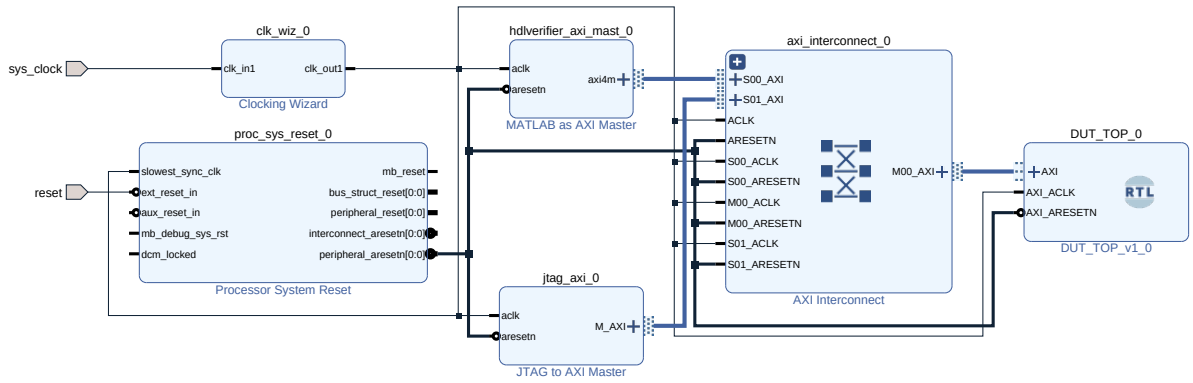

Figure 4.1: Screenshot of the Block Design view of the project from Vivado 2022.2.

Figure 4.1 shows a screenshot taken from the Block Design[1] view of Xilinx Vivado 2022.2.

[1]Block Design: the Block Design view is the main interface to Vivado IP integrator tool.

On the far right, we can see the on-FPGA model (**DUT_TOP_0**) described in the previous chapter, which communicates with the rest of the system using an AXI4-Lite interface.

On the left, we can see two service blocks, **clk_wiz_0** and **proc_sys_reset_0**. The former generates the clock signal [82], whereas the latter handles the reset signal [83] generation and propagation to the various IPs. These two blocks are IPs provided by Xilinx.

The two blocks in the middle are the communication interface, **hdlverifier_axi_master_0**, which will be briefly described in the next sub-section, and a debug module, **jtag_axi_0**. The latter allows the generation of AXI transactions to the FPGA directly from Vivado and has been extensively used during the early phases of the design. We decided to leave it in the final design due to its low requirement of resources, especially when compared to the utility of having a debug tool available.

The central block **axi_interconnect_0** represents the AXI interconnection. This IP provided by Xilinx implements the AXI interconnection matrix [84].

The AXI interconnect IP automatically takes care (among the others) of:

- Data Width Conversion: connects AXI Memory Mapped interfaces with different widths[2], splitting or merging multiple transactions if needed.
- Protocol Converter: connects an AXI (AXI3, AXI4 or AXI4-Lite) interface to another AXI interface using a different protocol (AXI3, AXI4 or AXI4-Lite). Burst[3] transactions supported by the full AXI standards (AXI3 and AXI4) are split into multiple AXI4-Lite transactions.
- Clock Domain Crossing: handles transactions across different clock domains. This is useful, for instance, to run the interconnect at a higher clock frequency than the rest of the FPGA logic to achieve higher bandwidth.
- Register Slice: Automatically places register slices to facilitate timing closure at the cost of increased latency.
- Address Range Checking: Automatically check if the address of the transaction falls within the assigned address space, if not the transaction is discarded.

[2]AXI widths: AMBA AXI standard [81] specifies 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide data buses for AXI3/AXI4 interfaces, whereas AXI4-Lite is limited to 32- or 64-bit wide data buses. The width of the address bus is instead limited to 32 or 64 bits for both standards.

[3]Burst: a burst transaction is a type of transaction in which a certain number of sequential addresses are written/read without requiring a new address [81]. AXI3 supports burst lengths from 1 to 16, whereas AXI4 supports lengths from 1 to 256. AXI4-Lite does not support burst transactions.

The AXI interconnect IP automatically selects the best bus topology (crossbar, shared crossbar, shared access, etc.) to achieve the best performance-resource trade-off [84].

Figure 4.2 shows a high-level diagram of the same design represented in figure 4.1 with the clocking and reset resources not shown.

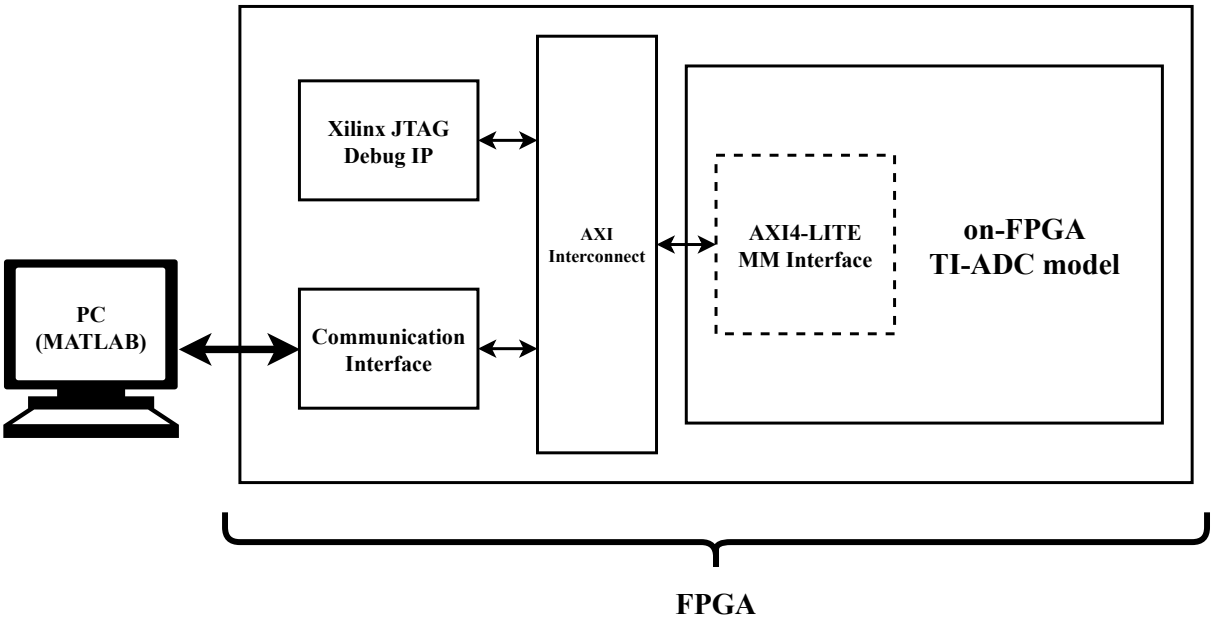

Figure 4.2: Block diagram of the whole test apparatus.

In 4.2 we can also see how the on-FPGA model interfaces with the PC through the communication interface.

4.1.1. Communication Interface

The communication interface is implemented using the AXI manager IP provided by MathWorks in the HDL Verifier Toolbox [85]. This IP allows accessing the FPGA from MATLAB. Actually, there are three different IPs within the HDL Verifier Toolbox:

- MATLAB PCIe[4] AXI Master
- MATLAB UDP[5] AXI Master
- MATLAB JTAG[6] AXI Master

[4]PCIe: the Peripheral Component Interconnect Express (PCIe) is a high-speed bus used to interconnect computing devices. It is the *de facto* industry standard for high throughput communications.

[5]UDP: the User Datagram Protocol is one of the most common communication protocols. It implements the transport layer of the ISO/OSI interconnection model.

[6]JTAG: it is an industry-standard interface used to test, debug, and program ICs.

Of the three, the PCIe version has not been considered due to the lack of a PCIe interface in the Arty-7 development board [65].

The UDP AXI Master implements a UDP to AXI4 interface. MathWorks also provides an auxiliary IP that implements the UDP and MAC[7] functionalities [86] connecting directly with the PHY[8] using a GMII/MII[9] interface (depending on the PHY available in the development board). The Ethernet PHY included in the Arty-7 development board (a DP83848J from Texas Instruments as shown in figure 4.3) supports only 10/100 Mb/s. The PHY communicates with the PC through an RJ-45 connector.

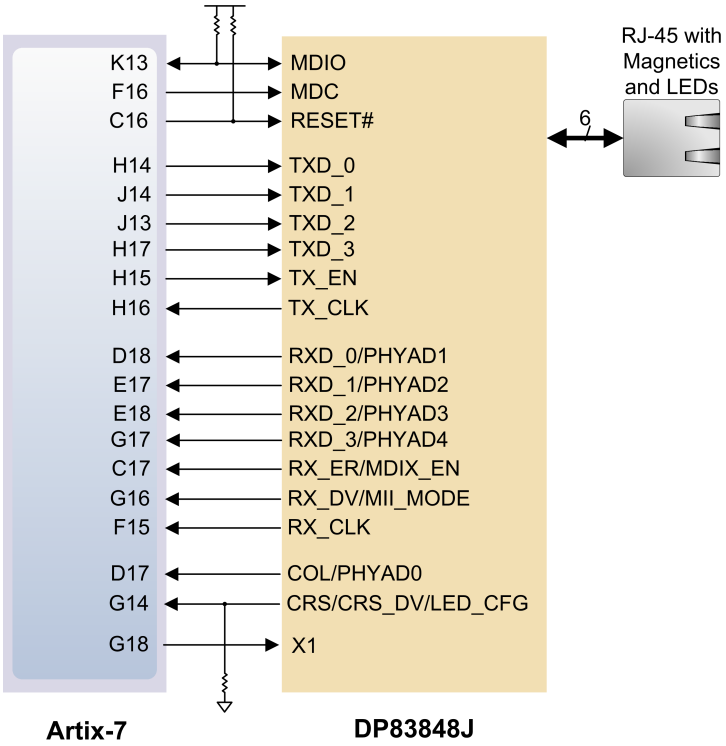

Figure 4.3: Arty A7 Ethernet schematic [65].

Once implemented, the UDP Master can be controlled from MATLAB using specific functions within the HDL Verifier package. From multiple tests, the actual net communication speed of the interface is about 5 Mb/s. Such a communication speed is sufficient to demonstrate a proof of concept, but severely constrains the throughput of the on-FPGA model.

[7]MAC: the Medium Access Control is a layer of the ISO/OSI that is responsible of the interaction with the transmission medium.

[8]PHY: a PHY (from PHYsical) is a device that implements the physical layer of the ISO/OSI interconnection model.

[9]GMII/MII: the Media Independent Interface and its Gigabit version are interfaces used to connect the PHY to the MAC module.

Furthermore, the UDP AXI master requires a moderate amount of BRAM36Ks (about 10 of them) to implement the TX/RX buffers in the MAC module, further reducing the amount of memory available for the on-FPGA model.

Likewise, the JTAX AXI Master implements a JTAG to AXI4 Interface. Differently from the UDP AXI Master, the JTAG one is standalone (it does not require any other module).

The JTAG interface is connected on dedicated configuration pins of the FPGA as shown in figure 4.4. The JTAG is interfaced with the PC through an FTDI[10] module that acts as a USB controller.

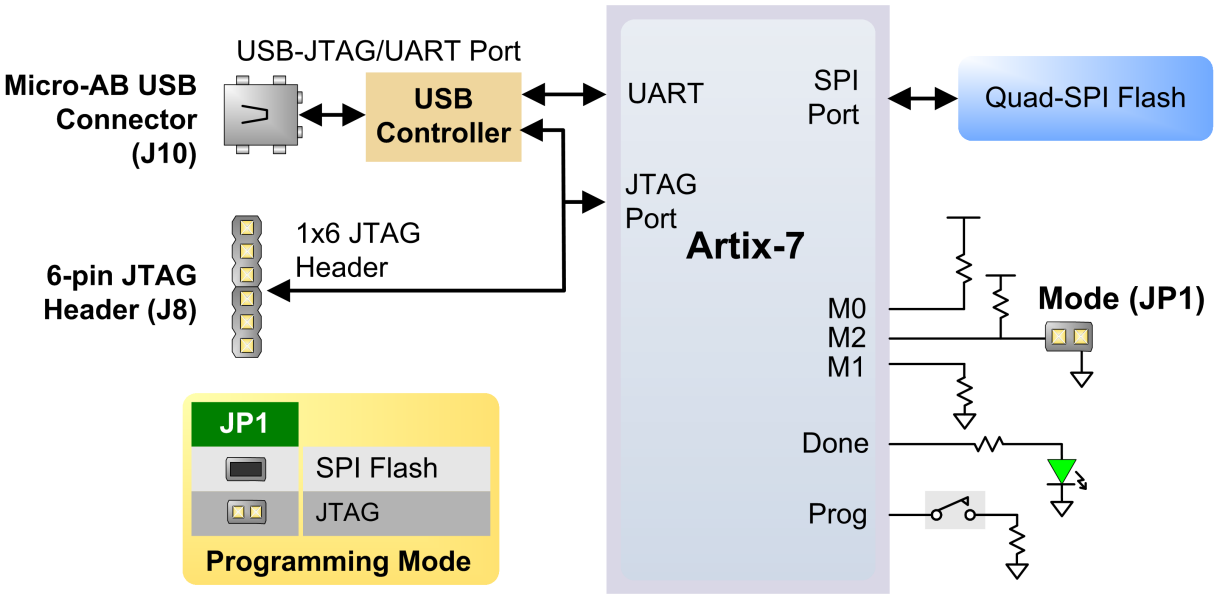

Figure 4.4: Arty A7 configuration schematic [65].

Like the UDP Master, the JTAG Master can be controlled from MATLAB using specific functions.

The actual net communication speed of the JTAG interface depends on the frequency of the JTAG itself. At the maximum possible frequency supported by the FPGA (66 MHz) it is about 4 Mb/s.

The communication speed of the JTAG is slightly lower than the one achieved by the UDP interface. Nevertheless, the amount of BRAM36Ks required by the former is much smaller than the one required by the latter (a single BRAM36K versus 10). For this reason, we decided to use the JTAG interface rather than the UDP one.

[10]FTDI: a company that produces USB ICs. It is usual to refer to the interface ICs themselves as FTDI

There is only one minor shortcoming in the use of the JTAG interface. The JTAG interface is also used to program the FPGA, thus it is not possible to connect the FPGA to Vivado and to MATLAB at once. However, this is not a real issue since the FPGA will never be connected to Vivado and MATLAB at the same time.

At this point, it is worth understanding and quantifying the type of constraint the speed of the communication interface poses on the on-FPGA model.

Taking into account the effect of the reorderer, at each clock cycle, the on-FPGA model is able to provide a new sample. Let's assume we are using a target clock frequency equal to $f_{clk} = 100\ MHz$ and 18-bit samples. The amount of data (bits) generated per second is equal to:

$$f_{clk} \cdot 18\ bit = 1.8\ Gb/s\,, \tag{4.1}$$

which is about 500 times the throughput of the JTAG and UDP interfaces.

Furthermore, the AXI Manager reads/writes 32-bit words, thus the 18-bit samples are extended to 32-bit words before being transmitted, resulting in an actual amount of data generated per second of 3.2 Gb/s.

Proceeding backward, i.e., starting from the bandwidth of the JTAG interface, and taking everything into account, the number of samples that can be transmitted per unit of time results:

$$\frac{4\ Mb/s}{32\ bit} \approx 125.000\ Samples/s\,, \tag{4.2}$$

which is about a sample transmitted every 800 samples processed.

It is possible to further parallelize the on-FPGA model. For example, by removing the reorderer and using a dedicated output buffer for each channel (as shown in figure 4.5), we can improve the throughput of the on-FPGA model from 1 sample per clock cycle to M samples per clock cycle, reaching in this case of $M = 8$ a transmitted-to-processed ratio of $1 : 6400$.

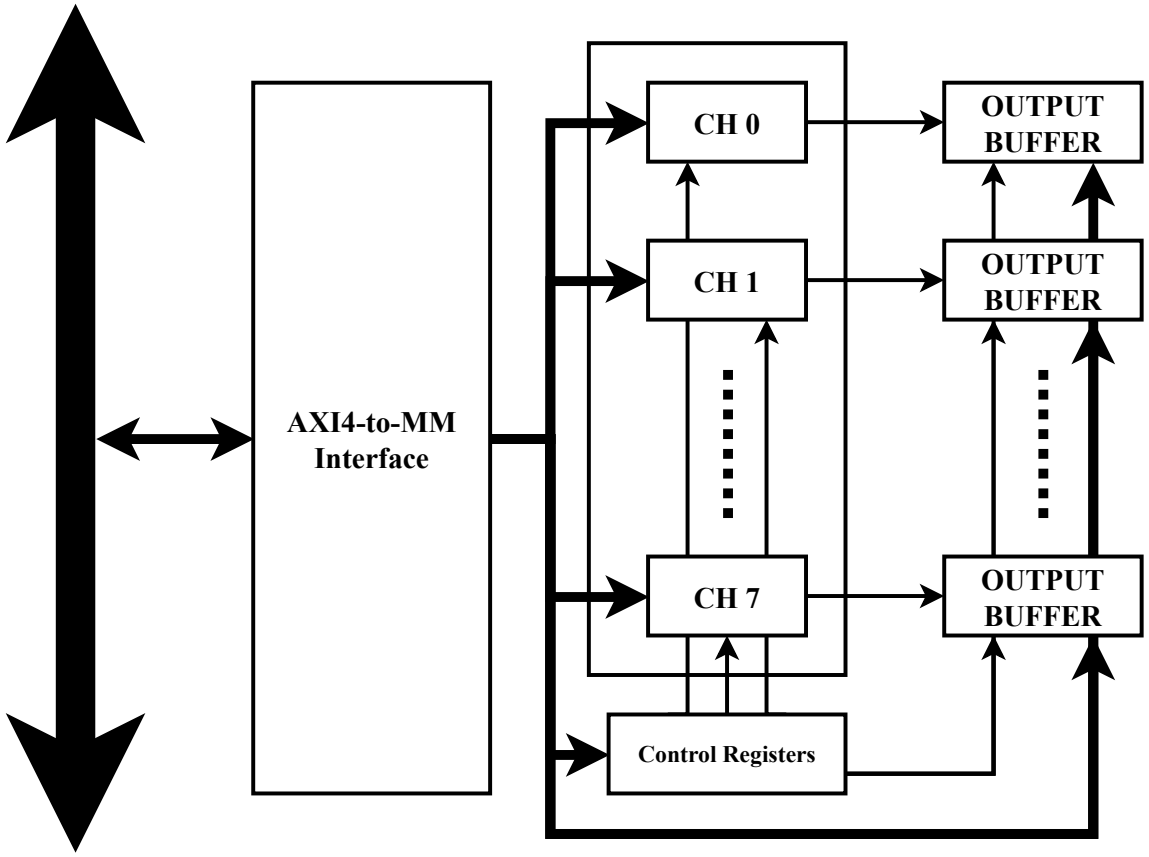

Figure 4.5: High-level block diagram of a hypothetical on-FPGA model with dedicated output buffers for each channel.

Note that the architecture proposed in figure 4.5 has not been implemented due to the aforementioned limitations. The design is already severely constrained in its capabilities by the lack of a high-speed interface. Implementing an even faster version of the on-FPGA model makes little sense. Furthermore, in this hypothetical 8-way on-FPGA model, there is no reorderer, which means we have to reorder the samples on the PC side (for instance, using MATLAB *reshape()* function).

Resorting to even wider implementations further exacerbates the issue.

It is clear that the communication interface required to fully exploit such a verification system needs to be orders of magnitude faster than the JTAG interface used to demonstrate the concept.

It is also quite evident the need for an output buffer to store the processed samples waiting to be transmitted.

4.1.2. Synthesis and Resource Utilization

The final specifications for the synthesized design are:

- Number of channels (M): 8
- NCO LUT: $2^{13} = 8192 \times 18$ *bit* samples
- ADC LUT: $2^{14} = 16384 \times 10$ *bit* samples
- Output buffer: $2^{16} = 65536 \times 18$ *bit* samples

Table 4.1 reports the amount of resources exploited for the implementation of the design with the parameters listed above.

	Used	Available	%
Slice LUT	5.741	63.400	9,06
Slice Register	8.884	126.800	7,01
BRAM36K	115.5	135	85,56
DSP48E1	24	240	10,00

Table 4.1: Resources utilization for the implemented design.

Please note that it is possible to split a single BRAM36K into two independent BRAM18Ks with little reduction of their capabilities [77]. This explains the partial utilization of a BRAM36K in table 4.1.

As previously stated, the design is heavily reliant on the amount of BRAM36Ks available on the FPGA. During the preliminary design phase, the use of the external 256-MB RAM available on the board has been considered (as mentioned in chapter 3). In the end, it was decided not to use it because it would have been useful only as an output buffer. In fact, the time (meant as the number of clock cycles) needed to access external memory makes impossible its use to implement the LUTs. Since the output buffer already has a satisfactory size it has been decided not to use the external RAM to simplify the design.

The RTL synthesis[11] and implementation[12] has been carried out using "Vivado synthesis Default" and "Vivado Implementation Default" strategies in Vivado 2022.2.2 targeting an Artix-7 XC7A100TCSG324-1 FPGA.

[11] synthesis: a process that converts the high-level hardware described using an HDL into a technology-independent gate-list

[12] Implementation: a process that maps the technology-independent gate-list resulting from the synthesis into the actual hardware available.

The clock frequency targeted for this design is $f_{clk} = 100\ MHz$, which means the on-FPGA model is able to process samples at a 1/20 of the speed of the real 10-bit 8-channel 2-GS/s TI-ADC we have used as reference.

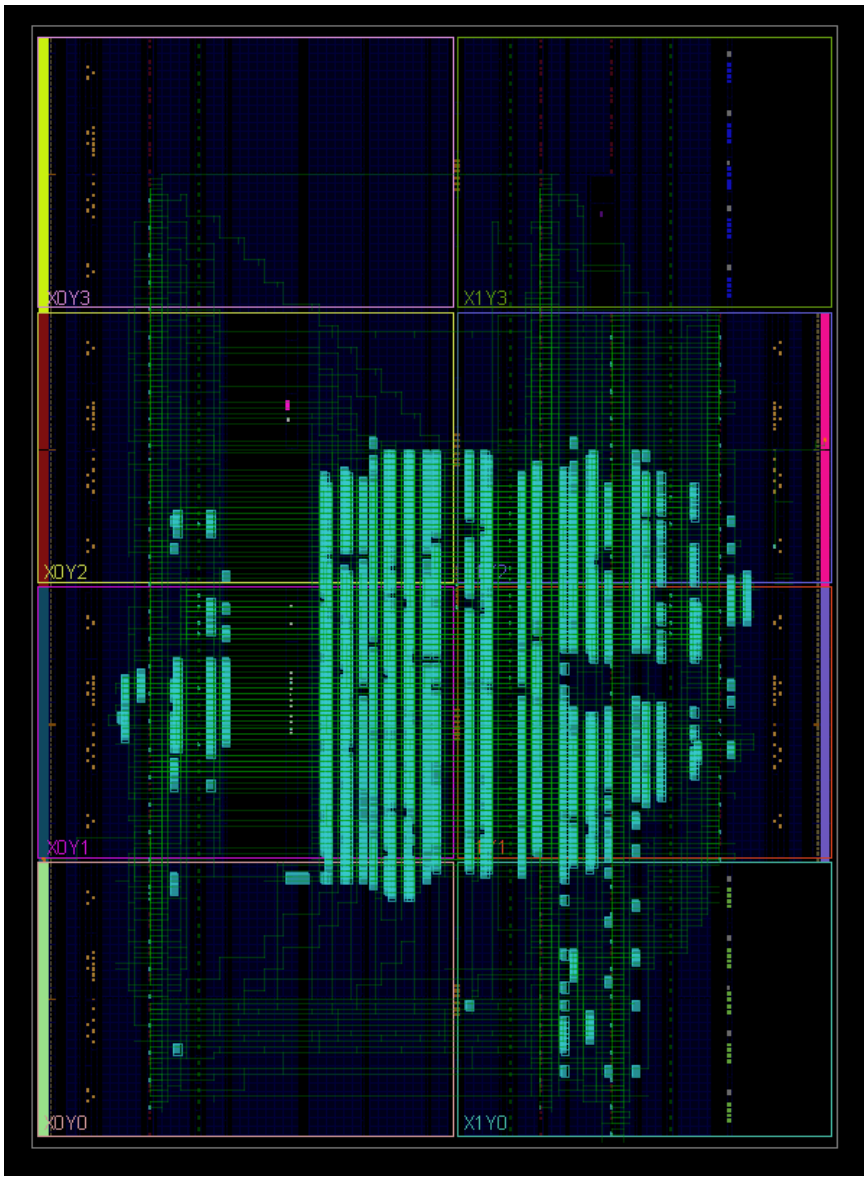

Figure 4.6: Screenshot of the Implementation view of the project from Vivado 2022.2.

Figure 4.6 shows the actual implementation of the design into the FPGA. It is possible to observe the interconnects (shown in green) that branch out from the logic-heavy region in the center (highlighted in cyan) to reach the BRAM resources spread over the device (the vertical columns on the side). Each of the eight rectangular sections shown in 4.6 represents a different Super Logic Region[13] (SLR).

[13]SLR: a Super Logic Region is a hierarchical unit of the FPGA that shares the same clocking resources.

4.2. Results

In this section we show the results of different simulations run on the on-FPGA model. All the graphs refer to samples processed by the on-FPGA model except where specifically indicated. The values of the offset, gain, and skew coefficients are kept constant between the different runs except when explicitly specified.

First of all, we need to verify if the on-FPGA model developed in chapter 3 is faithful to the MATLAB model. Figure 4.7 shows the spectra of the on-FPGA model (left) and MATLAB model (right) when considering only offset and gain non-idealities.

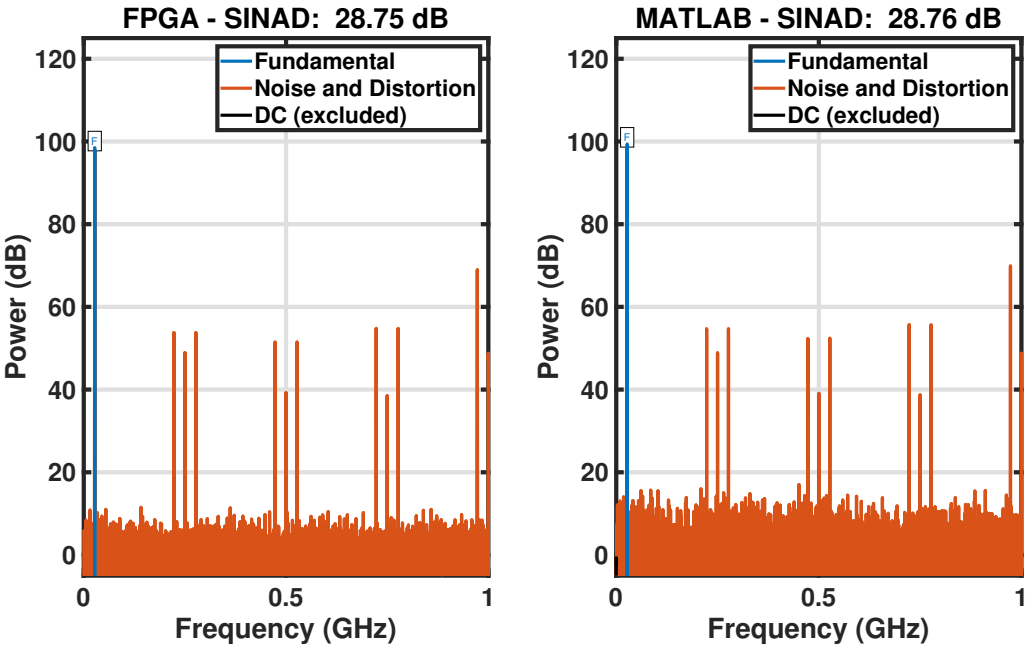

Figure 4.7: Output spectra of the on-FPGA model (left) and MATLAB model (right). The input is a sinewave at frequency $f_{in} = 27\ MHz$. Only offset and gain non-idealities are considered.

Both the MATLAB simulation and the on-FPGA emulation use the offset and gain coefficients listed in tables 2.2 (for the offset) and 2.3 (for the gain).

It is possible to notice how the two spectra are virtually identical, and there is only a 0.01 dB difference between the SNDR (SINAD) of the MATLAB simulation and the one of the on-FPGA emulation.

It is also possible to compare the spectrum from the on-FPGA model (the left plot in figure 4.7) with the one resulting from the MATLAB simulations (shown in figures 2.9 and 2.10 for offset and gain, respectively) finding the same spurious tones.

Figure 4.8 shows the output spectrum of the on-FPGA model affected by offset and gain mismatches obtained considering the first interval of 2^{16} samples (left) and the 128^{th} interval of 2^{16} samples (right).

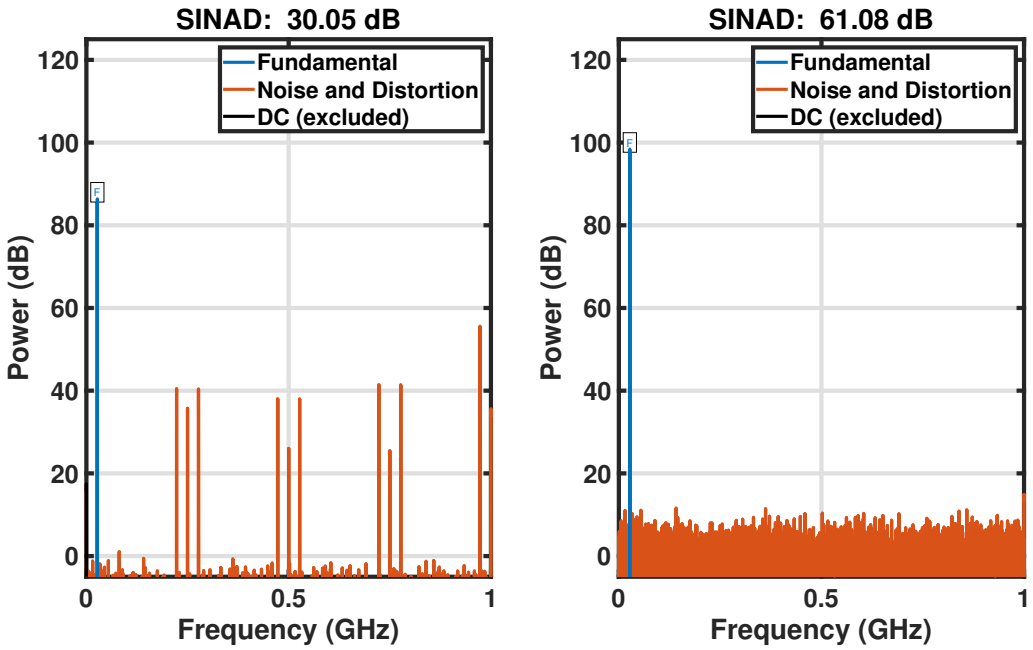

Figure 4.8: FFT of 2^{16} samples of the on-FPGA model, considering the first 2^{16} samples (left) and the 128^{th} 2^{16} samples (right). The input is a sinewave at frequency $f_{in} = 27\ MHz$. Only offset and gain non-idealities are considered.

It is possible to notice how the spurious tones are still present on the left, indicating that the BCAs have not yet reached their steady state, whereas on the right the spurious tones are completely removed. The BCAs are able to recover about 31 dB of SNDR (SINAD), equivalent to about 5.2 bits.

Figures 4.9, 4.10 and 4.11 show the evolution of the output spectrum of the on-FPGA model with BCAs enabled. The nine plots represent the output spectrum of the on-FPGA model of the TI-ADC taken at different intervals. Each interval is composed of 2^{16} samples, and the intervals are sequential. In other words, the output samples are windowed every 2^{16} samples, and the FFT is calculated.

The gain calibration algorithm converges faster than the offset one as shown by the lack of harmonics at $f_S/M \pm f_{in}$ in the 12^{th} interval (right plot in figure 4.10). In the same plot, we can still observe the harmonics at f_S/M due to the offset mismatches, which are still present also in the 16^{th} interval (central plot in figure 4.11) before disappearing in then 18^{th} interval (right plot in figure 4.11).

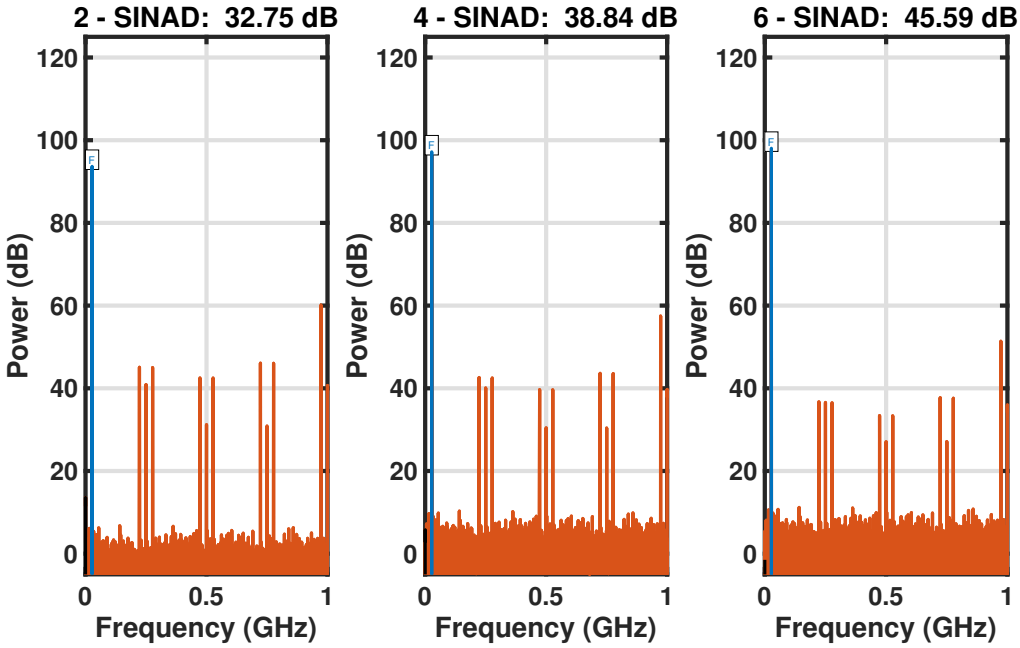

Figure 4.9: FFT of 2^{16} samples of the on-FPGA model. The input is a sinewave at frequency $f_{in} = 27\ MHz$. Only offset and gain non-idealities are considered. 2^{nd}, 4^{thd}, and 6^{th} intervals of 2^{16} samples (from left to right) are considered.

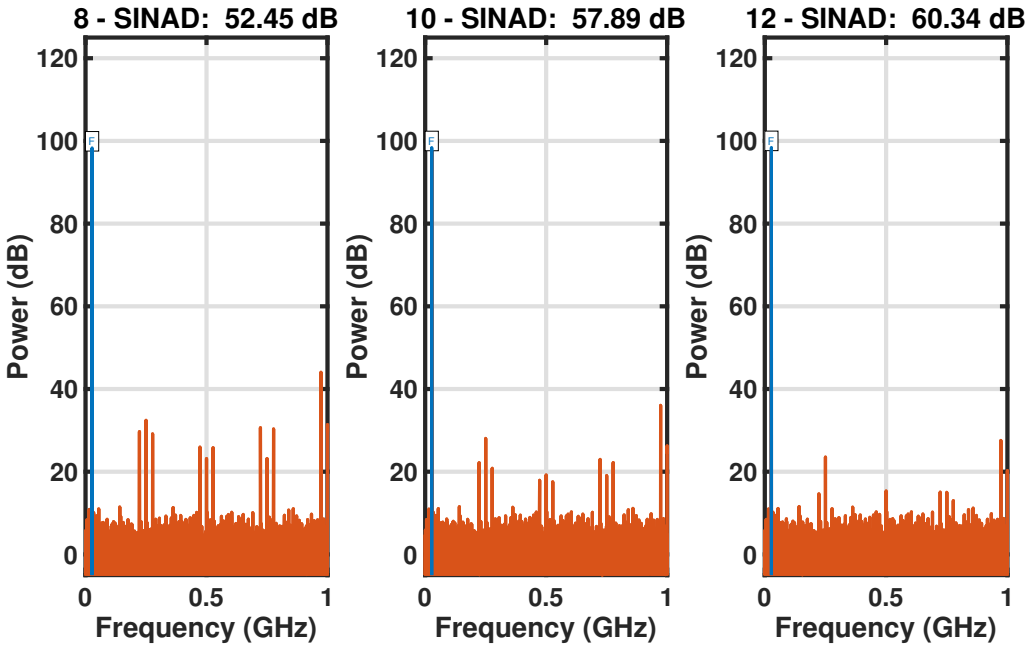

Figure 4.10: FFT of 2^{16} samples of the on-FPGA model. The input is a sinewave at frequency $f_{in} = 27\ MHz$. Only offset and gain non-idealities are considered. 8^{th}, 10^{th}, and 12^{th} intervals of 2^{16} samples (from left to right) are considered.

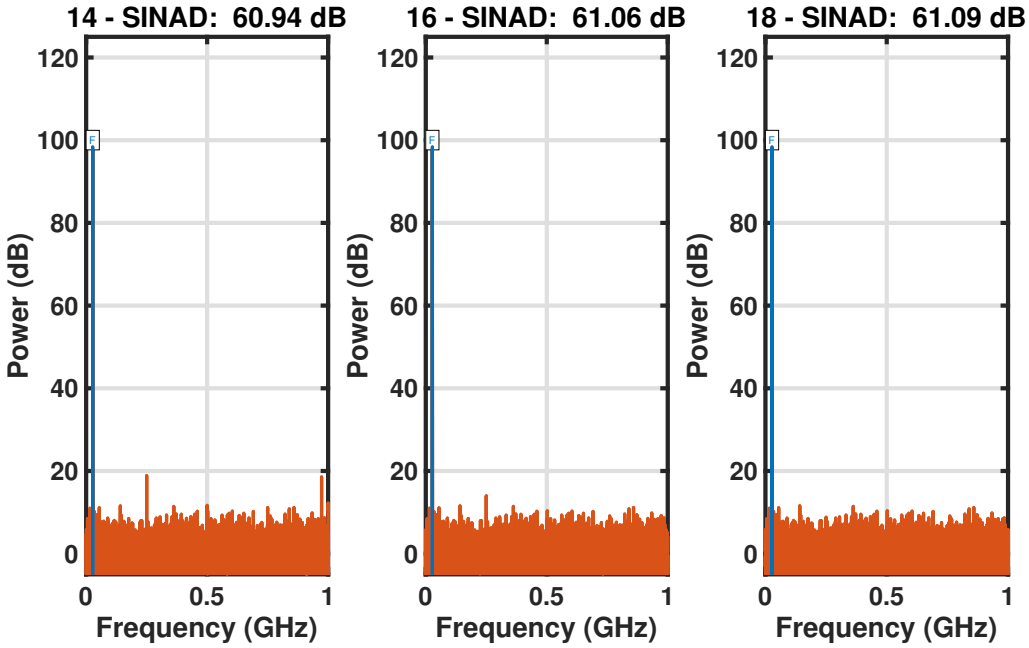

Figure 4.11: FFT of 2^{16} samples of the on-FPGA model. The input is a sinewave at frequency $f_{in} = 27\ MHz$. Only offset and gain non-idealities are considered. 14^{th}, 16^{th}, and 18^{th} intervals of 2^{16} samples (from left to right) are considered.

Figure 4.12 shows the evolution of the SNDR of the TI-ADC model affected only by offset and gain mismatches. The blue curve refers to the ADC with BCAs enabled.

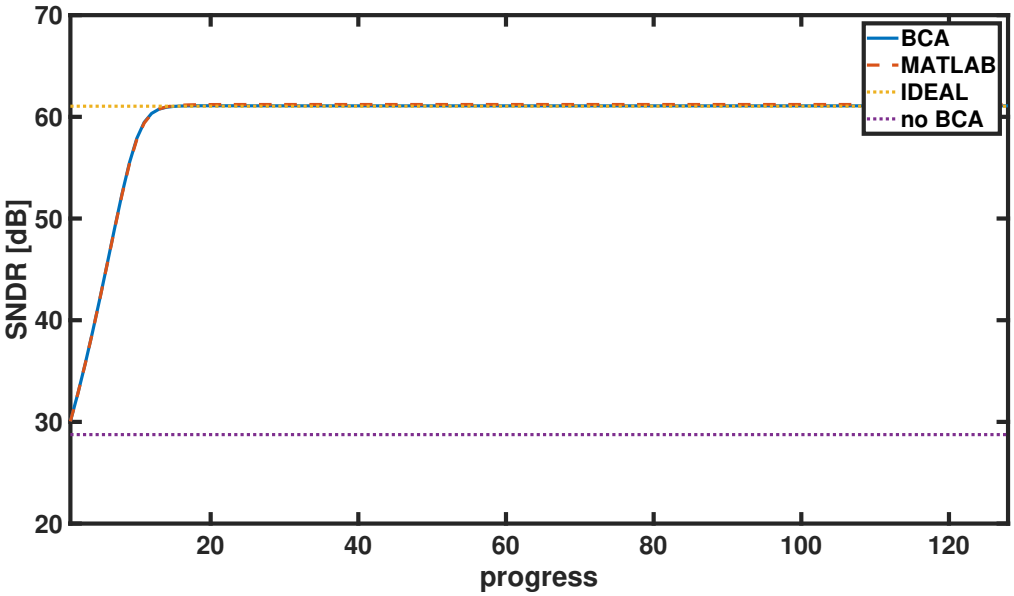

Figure 4.12: SNDR of the TI-ADC measured every 2^{16} samples for a total of 2^{23} samples. Only offset and gain non-idealities are considered.

In figure 4.12 it is possible to notice how the on-FPGA model (in blue) closely follows the evolution of MATLAB simulation (in orange, dashed). The SNDR of both models also converges to the ideal value (in yellow, dotted), which for a 10-bit ADC (taking into account the reduced amplitude of the input signal) results equal to:

$$SNDR = 10 \cdot 6.02\ dB + 1.76\ dB + 20 \cdot log_{10}(0.9) \approx 61.05\ dB\,, \tag{4.3}$$

where 0.9 represents the scaling factor k introduced in chapter 3 to guarantee that the processing chain does not saturate.

For the sake of completeness, figure 4.13 shows the same output spectra shown in figure 4.8 but taking into account also timing mismatches.

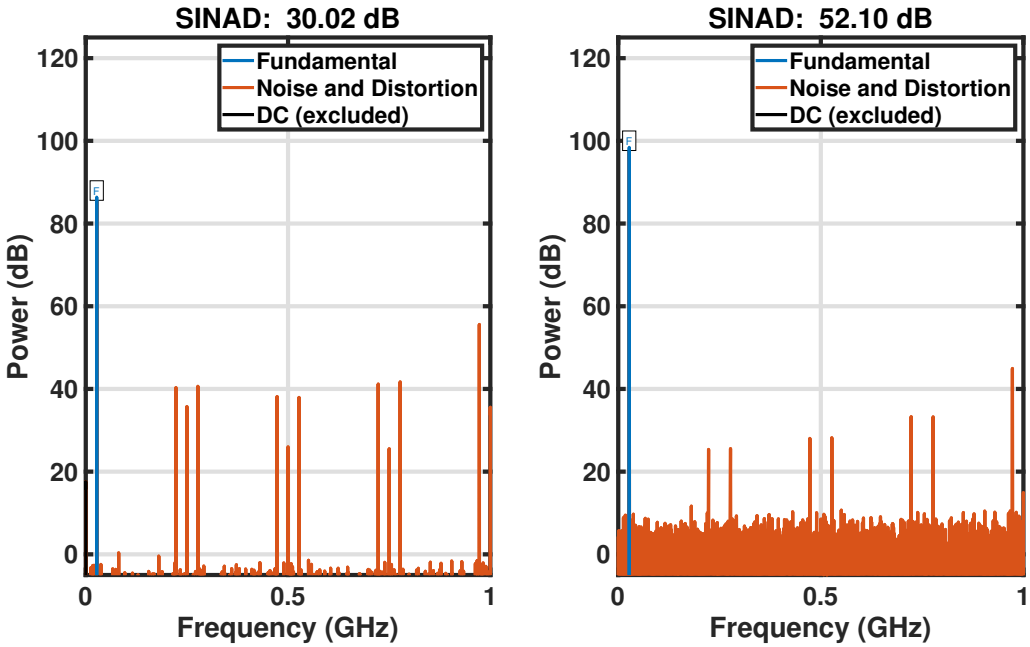

Figure 4.13: FFT of 2^{16} samples of the on-FPGA model, considering the first 2^{16} samples (left) and the 128^{th} 2^{16} samples (right). The input is a sinewave at frequency $f_{in} = 27\ MHz$. Offset, gain, and time skew non-idealities are considered.

We can notice how the harmonics at f_S/M due to the offset mismatches are attenuated by the offset BCA, but we still have some harmonics (albeit with a reduced amplitude) at $f_S/M \pm f_{in}$. The latters are due to both gain mismatch and time skew. The spurious tones due to the gain mismatch are filtered out by the gain BCA, whereas the ones due to the time skew are left unchanged. The timing error is not corrected limiting the SNDR of the converter as shown in figure 4.14.

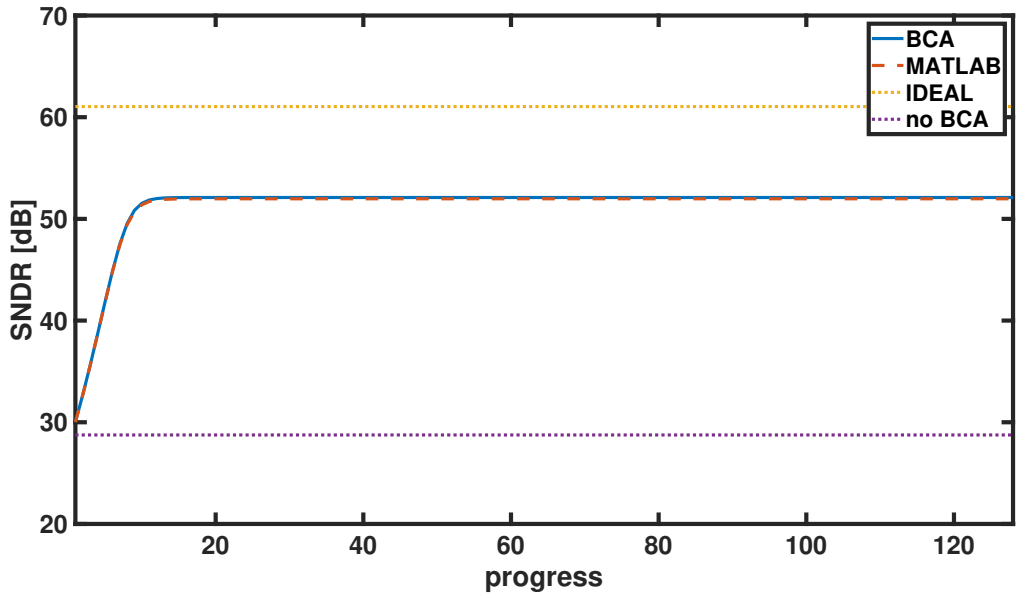

Figure 4.14: SNDR of the TI-ADC measured every 2^{16} samples for a total of 2^{23} samples. Offset, gain and time-skew non-idealities are considered.

Figure 4.15 shows the same output spectra shown in figure 4.13 but takes into account nonlinearity instead of time skew. The nonlinearity is modelled as shown in chapter 2.

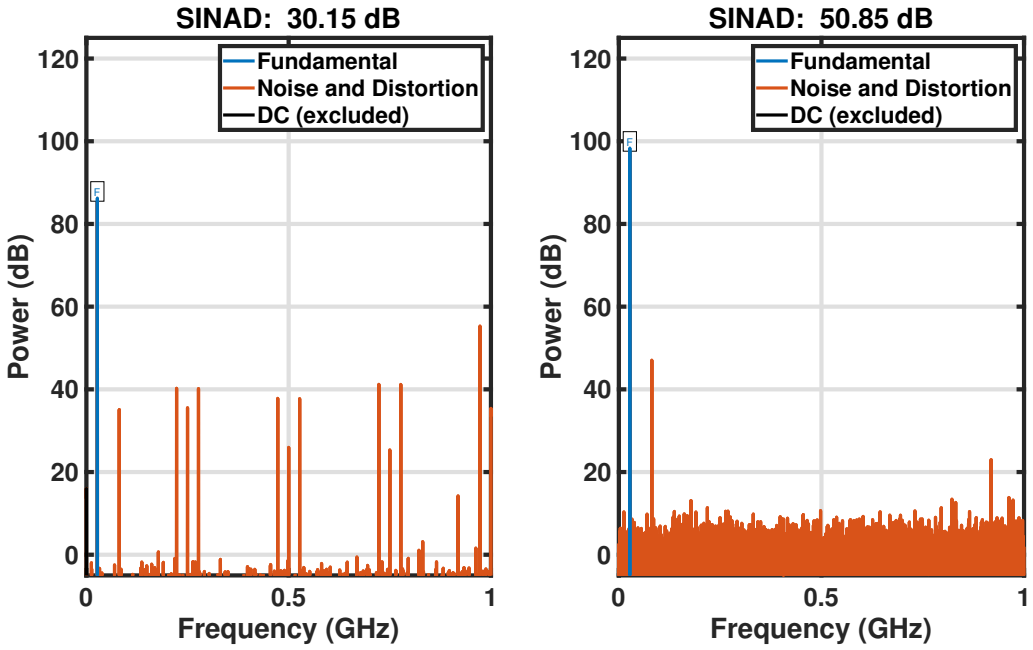

Figure 4.15: FFT of 2^{16} samples of the on-FPGA model, considering the first 2^{16} samples (left) and the 128^{th} 2^{16} samples (right). The input is a sinewave at frequency $f_{in} = 27\ MHz$. Offset, gain, and nonlinearity non-idealities are considered.

As for the time skew, the BCAs cannot correct the nonlinearity of the cores, and the spurious harmonics are visible even in the 128^{th} interval, whereas the spurious tones due to the offset and gain mismatches are completely removed.

Figure 4.16 shows the evolution of the SNDR of the 8-channel TI-ADC model affected by offset and gain mismatches and nonlinearity of the cores.

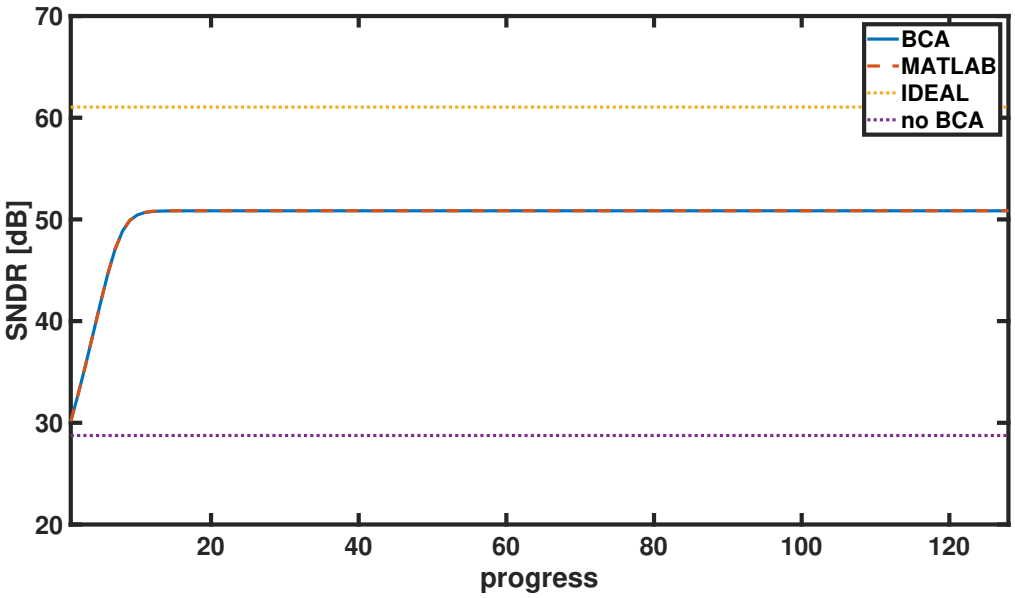

Figure 4.16: SNDR of the TI-ADC measured every 2^{16} samples for a total of 2^{23} samples. Offset, gain and nonlinearity non-idealities are considered.

As for the case of time skew, the SNDR of the converter does not reach the ideal value of 61.05 dB due to the inability of the calibration algorithms to correct non-idealities other than gain and offset mismatch.

All the simulations presented in this section were performed using the on-FPGA model in standard mode (or FIFO mode, as explained in chapter 3), i.e., the FPGA processed 2^{16} samples (the size of the output buffer), and then waited for a readout from the PC. We have decided to use the on-FPGA model in this mode to better characterize the reliability of the on-FPGA model with respect to the MATLAB one. In fact, plots like the ones reported in figures 4.12, 4.14, and 4.16 are achievable only using the standard operation mode. A similar constraint exists for the results shown in figures 4.9, 4.10, and 4.11.

Nevertheless, the more interesting operating mode is the continuous-run one, which allows emulating the calibration algorithms on real hardware for large time intervals (hours, days) at close (1/20) to real speeds.

This artifice is needed due to the limited speed of the communication interface. In fact, the speed of the communication interface limits the number of samples per second that can be sent to the PC. Thus, it is either possible to slow down the on-FPGA model to keep up with the interface (i.e., using it in standard mode), or discard most of the samples using the on-FPGA model at its maximum speed (i.e., using it in continuous run mode).

4.2.1. Comparison with MATLAB and VHDL Models

In this section, the results obtained with the on-FPGA model are compared with the ones obtained from the MATLAB model and from the simulations of the VHDL code.

Comparison with MATLAB model

The MATLAB model is meant as a reference to check the on-FPGA model. In chapter 2 we have presented extensive simulations performed with the MATLAB model. Here we compare a few of the results for the sake of clarity.

From figure 4.7 we can already appreciate the fidelity of the on-FPGA model to the MATLAB one. Figures 4.12 (offset and gain), 4.14 (offset, gain and time skew), and 4.16 (offset, gain and core nonlinearity) further confirm its reliability even when the calibration algorithms are running.

The run time of the MATLAB simulation[14] for a 2^{23} samples is about 1 s. The on-FPGA model is able to process the same number of samples (assuming the lack of any external bottleneck) in about:

$$t_{sim} = \frac{2^{23}}{f_{clk}} \approx 84\ ms\,, \tag{4.4}$$

providing a factor 12 speed-up.

Aside from the shorter simulation time, which is not exploitable in this case due to the lack of a high-speed interface, it's worth noting that the on-FPGA model also has the advantage of executing the algorithms on real hardware, thus, all the effects related to the hardware itself (saturation of the accumulators, finite precision, accumulation of truncation errors, long-term effects, etc.) are better simulated, without requiring *ad hoc* models. Furthermore, it is possible to leave the on-FPGA model running for hours/days, simulating time intervals that are not practical to simulate with MATLAB.

[14] average run time of 64 simulations on an 8 cores / 16 threads CPU with 64 GB of RAM, Windows 11, MATLAB R2021b.

Comparison with VHDL model

The VHDL model has by definition the same behavior as the implemented design and has been only used for verification purposes. This comparison is better suited for measuring the performances of the on-FPGA model with respect to a VHDL simulation performed on a computer, i.e., compare the time required to simulate a certain number of samples using the VHDL simulations and the on-FPGA model.

To process the same 2^{23} samples elaborated by the on-FPGA model in 84 ms, a VHDL simulator[15] requires nearly 100 seconds. The on-FPGA model implementation provides a factor 1185 speed-up. More practically, this means that if we are interested in verifying the behavior of the calibration algorithms for an hour (i.e., one hour for the 2-GS/s TI-ADC), we would need about 23700 hours (just over 33 months) using ModelSim, but only 20 hours using the on-FPGA emulation.

4.2.2. Achieved Performances

The average access time required to access the buffer (i.e., to read 2^{16} samples) is about 1.3 s, resulting in a simulation speed equal to:

$$\frac{2^{16}}{1.3\ s} \approx 50.000\ Samples/s\,. \tag{4.5}$$

This performance is reached when using the on-FPGA model in standard mode, which is how the simulations in the previous section have been performed.

The result in equation 4.2 is less than half the one in 4.5. This can be explained by the overhead of the functions provided by MathWorks, i.e., the MATLAB functions used to communicate with the FPGA do not saturate the JTAG interface. This result is quite modest and it is easily overcome even by the VHDL simulations.

However, we have to remember that we are heavily bottle-necked by the communication interface. Should this limitation be removed, the simulation speed will be equal to:

$$f_{clk} \cdot 1\ Sample/clk \approx 100\ MSamples/s\,, \tag{4.6}$$

which is a factor 2000 of improvement. This performance is actually reached in this work when using the on-FPGA model in continuous run mode (as explained in chapter 3).

[15] average run time of 10 simulations on an 8 cores / 16 threads CPU with 64 GB of RAM, Windows 11, ModelSim Intel FPGA edition 2020.1

Furthermore, as stated before, it is possible to redesign the reorderer and the output buffer to allow all channels to work in parallel instead than interleaved, as shown in figure 4.5. This would result in another factor 8 of improvement, resulting in a simulation speed of:

$$f_{clk} \cdot 8\ Sample/clk \approx 800\ MSamples/s\,. \tag{4.7}$$

Such an improved version has not been implemented because even the actual on-FPGA model is constrained by the communication interface.

To compare the various simulations methods, we defined a Figure of Merit (FoM) as the number of samples divided by the time needed to process them:

$$FoM = \frac{N_{samples}}{t_{sim}}\,. \tag{4.8}$$

It would be useful to compare the simulation speed of the different methods directly to the processing speed of the real converter (2-GS/s). In order to ease the comparison, we define a normalized Figure of Merit (FoM_n) as:

$$FoM_n = \frac{FoM_{method}}{FoM_{TI-ADC}}\,. \tag{4.9}$$

The higher FoM_n the faster the method. A FoM_n equal to 1 implies that the simulation/verification method is as fast as the real TI-ADC. A FoM_n higher than 1 means the simulation/verification is faster than the original TI-ADC.

Note that it is possible to remove the dependance on $N_{samples}$ in equation 4.9, resulting in:

$$FoM_n = \frac{T_{method}}{2\ GS/s}\,, \tag{4.10}$$

where T_{method} represents the throughput (in samples per second) of the method under consideration.

Table 4.2 highlights the performances shown by the on-FPGA model compared to other simulation methods, and with the real ADC this work aims to model.

Method	Simulation time [t_{sim}]	FoM_n
on-FPGA model (standard mode)	169.1 s	0.000025
on-FPGA model (continuous run mode)	0.084 s	0.05
on-FPGA model (hypothetical 8 way)	0.011 s	0.4
MATLAB model	1.004 s	0.0042
behavioral VHDL	99.4 s	0.000042
Real ADC	0.0042 s	1

Table 4.2: Summary of the simulation times of the verification methodologies considered for 2^{23} samples, and corresponding normalized FoM (FoM_n).

Note that the hypothetical 8-way on-FPGA model is the improved version described in figure 4.5. This version has not been implemented in this work due to the external limitations to the throughput of the on-FPGA model previously stated.

The achieved performances (in terms of throughput and improvement over a MATLAB or VHDL simulation) are comparable with the ones reached by similar works [21–23] (albeit not targeting a TI-ADC).

4.2.3. Advantages and Limitations of the Proposed Method

This sub-section presents the advantages and the limitations of the proposed method, aside from the high speed of the emulation enabled by the on-FPGA implementation, which has been highlighted enough in the previous chapters and sections.

Advantages

The other main advantages of the proposed verification methodology have already been presented in chapter 1, and are reported here for the sake of completeness:

- **Verification of long-term behaviors**: the close to real-time (1/20 of the speed of the 2-GS/s TI-ADC) emulation speed enables the investigation of long-term behaviors that might require hours/days before manifesting.
- **Prototyping on real hardware**: the use of real hardware allows verifying the effects of hardware-related characteristics, without requiring *ad hoc* models.
- **Simulation time independent on the design size**: if the FPGA has enough resources to implement the design, then the simulation speed is independent of the size of the verification target.

Limitations

Aside from the limitation on the performances achieved by this proof of concept due to the limited speed of the communication interface, there is another, perhaps more important, limitation.

In a computer simulation, like one performed in MATLAB using the model developed in chapter 2, or using a HDL simulator, or even a real schematic-level simulation performed in an EDA tool like Cadence Virtuoso, all the variables are exposed to the user (i.e., it is possible to see the evolution of all the signals in the design). This allows a greater insight into the inner workings of the Device Under Test[16] (DUT), since it is possible to select any arbitrary signal and track its evolution.

Such a capability is difficult to be obtained in a hardware system like the on-FPGA model presented in this *thesis*. Only predetermined signals (in this work the output samples) are saved. Others like, for instance, the accumulators storing the correction coefficients of the BCAs are not saved. It is always possible to reconfigure the on-FPGA model to expose different signals, but this requires a new synthesis and implementation.

Another possibility consists in instantiating an Integrated Logical Analyzer[17] (ILA) to monitor the internal signals [87]. This solution has a few major shortcomings. First of all, the exposed signals are determined at the time of the synthesis. This alone makes the solution much less versatile. Furthermore, the ILA has a large requirement of resources (mainly BRAM36Ks due to the buffer needed to store the sampled signals), which are better used to implement the on-FPGA model. In fact, it is better to use the available BRAM36Ks to increase the size of the LUTs (either the NCO or the ADC one), which have an effect on the reliability and accuracy of the on-FPGA model. Last but not least, the ILA is able to sample only small intervals (in the order of 2^{10}-2^{12} usually). Though this amount of samples is enough for debugging purposes, it is obviously not sufficient to verify a design. All those limitations make the ILA not suitable for the scope of this *thesis*.

4.3. Future Developments

This section briefly discusses some aspects mentioned in this *thesis* that deserve further investigation.

[16] DUT: a Device Under Test is a technical term referring to a design, product, or system undergoing a test.

[17] ILA: an Integrated Logic Analyzer is an IP provided by Xilinx that can be used to observe the internal signals of a design

Fully-Digital Time-Skew Calibration

As mentioned in chapter 2, there is a trend to calibrate offset and gain mismatches in the digital domain, but there is an opposite trend for the calibration of the time skew error, which is usually avoided using a first-rank S&H architecture [61–64] or detected in the digital domain and corrected in the analog one [88]. Only few works suggest fully-digital approaches to time-skew error calibration [57–59].

It would be interesting to develop a fully-digital BCA for time skew errors and use the proposed verification method to validate it.

Mixed-signal time-skew calibration algorithms are difficult, if not impossible to verify using the proposed verification method due to the clocked nature of the FPGA environment. In fact, the vast majority of mixed-signal calibration algorithms for time skew make use of Digital-to-Time Converters (DTCs), something that is not approximable in a satisfactory way within a clocked system.

Machine-Learning Assisted Error Calibration

Another, perhaps more interesting, research opportunity related to calibration algorithms is the use of Machine Learning (ML) (also referred to as Neural Network (NN)) to detect and correct the non-idealities [89, 90]. ML is perfectly suited to tasks like pattern recognition, which makes them an ideal candidate to detect and reduce spurious harmonics.

In this case, the on-FPGA model could be used both to verify the ML algorithms and, perhaps more interestingly, to train them. The training process consists in evaluating the performances of the ML algorithm, estimating the error from the desired behavior, and updating the coefficients of the ML algorithms. This process is iterated hundreds of times until the desired performances are met.

FPGAs represent an ideal environment to train the NN due to their reconfigurability [91].

High-speed Interface

To fully exploit the power of this verification technique, a high-speed communication interface is required.

Some estimations on the required bandwidth have been provided in section 4.1 and subsection 4.2.2. The required multi-Gb/s throughput leaves few possible interfaces among the suitable ones. More precisely, the interfaces better suited to communicate with the on-FPGA model are PCIe, 10G Ethernet, and USB 3.x.

Unfortunately, the development board is not equipped with any of the aforementioned interfaces. Furthermore, the task of implementing the whole communication stack (for instance, the MAC/TCP/IP stack for the Ethernet) is comparable in complexity to the entire work done in this *thesis*.

Nevertheless, it would be interesting to expand the proof of concept presented in this *thesis* to a verification instrument capable of reducing simulation and verification times by orders of magnitude.

Conclusions

This *thesis* focused on the verification of the digital calibration algorithms of a Time-Interleaved (TI) Analog-to-Digital Converter (ADC) by implementing them on a Field Programmable Gate Array (FPGA). We showed how the use of FPGA prototyping can shorten the verification times, enabling a nearly real-time simulation, while also providing new insight into long-term behavior of the calibration algorithms (like, for instance, the saturation of the accumulators or integration of errors caused by the finite precision of the digital implementation). We also listed a few scenarios in which standard VHDL or MATLAB simulations are not able to satisfy the verification requirements. In fact, VHDL simulations allow to reasonably investigate only short time intervals (due to the extremely long times required to simulate larger time intervals), whereas MATLAB is not suited to realistically describe a complex digital system.

To explain the concept of verification we introduced its *raison d'être*, before presenting its history, its evolution, and the current state-of-the-art in the semiconductor industries. We then introduced the advantages of the proposed verification method, which is to use an FPGA to verify the digital logic of TI-ADC. To achieve this goal we had to implement a digital approximation of the TI-ADC on an FPGA.

Before going into the details of the actual implementation on-FPGA, we introduced the concept of time-interleaved converter, discussing its advantages, and disadvantages. Among the main advantages, we find: **1)** the reduced performances required for the single sub-converters. **2)** the possibility (which is also a consequence of the first one) of implementing high-speed converters in standard CMOS processes. This second advantage enables tight integration of the converter with the digital functions, which nowadays constitute the vast majority of the systems. Then we focused on the disadvantage of time-interleaving, which is the degradation of the performances of the converter due to the presence of mismatches between the different sub-converters. We analyzed the different sources of error (namely gain, offset, time-skew, and nonlinearity) one at the time, providing an analytical expression for their effects on the output spectrum of the TI-ADC when possible. The analytical models were compared with an *ad hoc* MATLAB model, showing their correct behaviour.

Once investigated these limitations to the performances of the TI-ADC, we focused on the calibration algorithms of two of them, namely, the offset and gain mismatches. We listed a few noteworthy results found in literature before describing the background calibration algorithms selected for this *thesis*. We explained their behaviors, and showed the results of the MATLAB simulations using the same *ad hoc* model developed for verifying the correct behaviour of the analysis of the non-idealities.

At this point, we began analyzing the implementation on the FPGA of a digital approximation of the aforementioned TI-ADC and its calibration algorithms. After a description of the general architecture of the on-FPGA model, we analyzed separately the implementation of the different blocks that compose the model of the TI-ADC, starting with the Numerically Controlled Oscillator (NCO) used to model the sample & hold circuit. Then, we showed the implementation of the ADC itself, including in the model the aforementioned non-idealities, and the calibration algorithms. Last, we briefly discuss the implementations of all the auxiliary blocks required to interface and communicate with the on-FPGA model of the TI-ADC, i.e., the AXI4-to-MM interface, the reorderer, and the output buffer.

The proposed design was then synthesized and implemented on a Xilinx Artix-100T FPGA. The results obtained with the model implemented on the FPGA have been then compared with the VHDL and MATLAB simulations, showing a comparable accuracy but a simulation time improvement of a factor 1185 with respect to VHDL simulations. The results and performances achieved by this verification method are shown and analyzed before listing the advantages of this verification technique.

This *master thesis* has demonstrated how verification methodologies originally developed for fully-digital ICs can be repurposed for the verification of the digital sections of a mixed-signal circuit, in this case, a 10-bit 8-channel 2GS/s Time-Interleaved Analog-to-Digital Converter. The main advantages of the proposed digital verification techniques are reported below for the sake of completeness:

- **Close to real-time simulation speed**: the on-FPGA model is able to simulate the digital calibration algorithms at 1/20 of the speed of the real TI-ADC this *thesis* targets with the same level of detail.
- **Analysis of long-term behaviors**: the close to real-time simulation speed allows investigating long-term behaviors of the calibration algorithms. Simulating the on-FPGA model for hours/days results in a similar amount of simulation times, whereas a computer simulation requires hours to simulate a few seconds.

- **Prototyping on real hardware**: the use of real hardware to prototype the design allows verifying the effects of hardware-related characteristics (like, for instance, errors due to the finite precision of the digital implementation), without requiring complex *ad hoc* models.
- **Simulation time independent on the design size**: the use of a parallel device like an FPGA makes the simulation time independent on the size of the design. If the FPGA has enough resources to implement the design, then it will run at the same speed of a smaller one.

Of the advantages listed above, the first one (which is the simulation speed) was constrained by the low throughput of the available communication interface, and limited to about 50000 samples per second when all the samples are streamed to the PC. Should a communication interface featuring a higher speed, like PCIe, be available, a greater improvement could be attained, reaching a simulation time improvement factor of about 10000 with respect to VHDL simulations.

Bibliography

[1] W.C. Black and D.A. Hodges. Time interleaved converter arrays. *IEEE Journal of Solid-State Circuits*, 15(6):1022–1029, 1980. doi: 10.1109/JSSC.1980.1051512.

[2] B. Murmann. Digitally assisted analog circuits. *IEEE Micro*, 26(2):38–47, 2006. doi: 10.1109/MM.2006.33.

[3] IEEE. Ieee draft guide: Adoption of the project management institute (pmi) standard: A guide to the project management body of knowledge (pmbok guide)-2008 (4th edition). *IEEE P1490/D1, May 2011*, pages 1–505, 2011. doi: 10.1109/IEEESTD.2011.5937011.

[4] Umer Farooq. Pre-silicon verification using multi-fpga platforms: A review. *Journal of Electronic Testing*, 37:1–18, 02 2021. doi: 10.1007/s10836-021-05929-1.

[5] Foster. 2020 wilson research group functional verification study, 2020. last accessed on 2022-11-17.

[6] Our World In Data. Moore's law: The number of transistors per microprocessor, 2020. URL `https://ourworldindata.org/grapher/transistors-per-microprocessor`. last accessed on 2022-11-22.

[7] Harry D. Foster. Why the design productivity gap never happened. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 581–584, 2013. doi: 10.1109/ICCAD.2013.6691175.

[8] J. Harrison. Formal verification at intel. In *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, pages 45–54, 2003. doi: 10.1109/LICS.2003.1210044.

[9] Bohuslav Křena and Tomáš Vojnar. Automated formal analysis and verification: an overview. *International Journal of General Systems*, 42(4):335–365, 2013. doi: 10.1080/03081079.2012.757437. URL `https://doi.org/10.1080/03081079.2012.757437`.

[10] Foster. What's the difference between emulation and pro-

totyping?, 2015. URL https://semiwiki.com/eda/cadence/4980-whats-the-difference-between-emulation-and-prototyping/. last accessed on 2022-11-17.

[11] James A. Lear. Digital and mixed-signal verification differences. In *2009 10th International Workshop on Microprocessor Test and Verification*, pages 91–94, 2009. doi: 10.1109/MTV.2009.15.

[12] K. Muhammad, T. Murphy, and R.B. Staszewski. Verification of rf socs: Rf, analog, baseband and software. In *IEEE Radio Frequency Integrated Circuits (RFIC) Symposium, 2006*, pages 4 pp.–364, 2006. doi: 10.1109/RFIC.2006.1651166.

[13] Henry Chang and Ken Kundert. Verification of complex analog and rf ic designs. *Proceedings of the IEEE*, 95(3):622–639, 2007. doi: 10.1109/JPROC.2006.889384.

[14] Khurram Muhammad, Thomas Murphy, and Robert Bogdan Staszewski. Verification of digital rf processors: Rf, analog, baseband, and software. *IEEE Journal of Solid-State Circuits*, 42(5):992–1002, 2007. doi: 10.1109/JSSC.2007.894327.

[15] Cadence Design Systems William Dunham. Rf-a/ms ic functional verification: Requirements and methodology, 2020. URL https://www.microwavejournal.com/articles/print/5194-rf-a-ms-ic-functional-verification-requirements-and-methodology. last accessed on 2022-11-22.

[16] Chao Liang. Mixed-signal verification methods for multi-power mixed-signal system-on-chip (soc) design. In *2013 IEEE 10th International Conference on ASIC*, pages 1–4, 2013. doi: 10.1109/ASICON.2013.6812042.

[17] Tech Design Forums. Real-number or wreal modeling, 2020. URL https://www.techdesignforums.com/practice/guides/real-value-wreal-modelling/. last accessed on 2022-11-22.

[18] Chao Liang, Geng Zhong, Song Huang, and Bei Xia. Uvm-ams based sub-system verification of wireless power receiver soc. In *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 1–3, 2014. doi: 10.1109/ICSICT.2014.7021458.

[19] Wilmer Ramirez, Hector Gomez, and Elkim Roa. On uvm reliability in mixed-signal verification. In *2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)*, pages 233–236, 2019. doi: 10.1109/LASCAS.2019.8667543.

[20] A. Fernandez-Alvarez, M. Portela-Garcia, and M. Garcia-Valderas. Fpga-based

hw/sw co-simulation system for mixed-signal circuits. In *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6, 2016. doi: 10.1109/DCIS.2016.7845364.

[21] Steven Herbst, Byong Chan Lim, and Mark Horowitz. Fast fpga emulation of analog dynamics in digitally-driven systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2018. doi: 10.1145/3240765.3240808.

[22] Steven Herbst, Gabriel Rutsch, Wolfgang Ecker, and Mark Horowitz. An open-source framework for fpga emulation of analog/mixed-signal integrated circuit designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(7): 2223–2236, 2022. doi: 10.1109/TCAD.2021.3102516.

[23] Daniel Stanley, Can Wang, Sung-Jin Kim, Steven Herbst, Jaeha Kim, and Mark Horowitz. Fast validation of mixed-signal socs. *IEEE Open Journal of the Solid-State Circuits Society*, 1:184–195, 2021. doi: 10.1109/OJSSCS.2021.3122397.

[24] Behzad Razavi. Design considerations for interleaved adcs. *IEEE Journal of Solid-State Circuits*, 48(8):1806–1817, 2013. doi: 10.1109/JSSC.2013.2258814.

[25] Aaron Buchwald. High-speed time interleaved adcs. *IEEE Communications Magazine*, 54(4):71–77, 2016. doi: 10.1109/MCOM.2016.7452269.

[26] C.S.G. Conroy, D.W. Cline, and P.R. Gray. An 8-b 85-ms/s parallel pipeline a/d converter in 1- mu m cmos. *IEEE Journal of Solid-State Circuits*, 28(4):447–454, 1993. doi: 10.1109/4.210027.

[27] C.S.G. Conroy, D.W. Cline, and P.R. Gray. A high-speed parallel pipelined adc technique in cmos. In *1992 Symposium on VLSI Circuits Digest of Technical Papers*, pages 96–97, 1992. doi: 10.1109/VLSIC.1992.229274.

[28] Christophe Erdmann, Bob Verbruggen, Bruno Vaz, Roberto Pelliconi, John Mcgrath, Ryan Kinnerk, Ronnie De La Torre, John O'Dwyer, Pat Lynch, Padraig Kelly, Peng Lim, Daire Breathnach, and Brendan Farley. A modular 16nm direct-rf tx/rx embedding 9gs/s dac and 4.5gs/s adc with 90db isolation and sub-80ps channel alignment for monolithic integration in 5g base-station soc. In *2018 IEEE Symposium on VLSI Circuits*, pages 219–220, 2018. doi: 10.1109/VLSIC.2018.8502292.

[29] Tim J. Ridgers, Claire Boucey, Jan-Peter Frambach, Laure Rolland du Roscoat, and Patrice Gamand. Challenges in integrating embedded rf within a soc. In *2008*

IEEE Radio and Wireless Symposium, pages 547–550, 2008. doi: 10.1109/RWS.2008.4463550.

[30] Brendan Farley, Christophe Erdmann, Bruno Vaz, John McGrath, Edward Cullen, Bob Verbruggen, Roberto Pelliconi, Daire Breathnach, Peng Lim, Ali Boumaalif, Patrick Lynch, Conrado Mesadri, David Melinn, Kwee Peng Yap, and Liam Madden. A programmable rfsoc in 16nm finfet technology for wideband communications. In *2017 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 1–4, 2017. doi: 10.1109/ASSCC.2017.8240201.

[31] L. Sumanen, M. Waltari, and K.A.I. Halonen. A 10-bit 200-ms/s cmos parallel pipeline a/d converter. *IEEE Journal of Solid-State Circuits*, 36(7):1048–1055, 2001. doi: 10.1109/4.933460.

[32] Dušan Stepanovic and Borivoje Nikolic. A 2.8 gs/s 44.6 mw time-interleaved adc achieving 50.9 db sndr and 3 db effective resolution bandwidth of 1.5 ghz in 65 nm cmos. *IEEE Journal of Solid-State Circuits*, 48(4):971–982, 2013. doi: 10.1109/JSSC.2013.2239005.

[33] Boris Murmann. The race for the extra decibel: A brief review of current adc performance trajectories. *IEEE Solid-State Circuits Magazine*, 7(3):58–66, 2015. doi: 10.1109/MSSC.2015.2442393.

[34] B. Murmann. A/d converter trends: Power dissipation, scaling and digitally assisted architectures. In *2008 IEEE Custom Integrated Circuits Conference*, pages 105–112, 2008. doi: 10.1109/CICC.2008.4672032.

[35] Y.-C. Jenq. Digital spectra of nonuniformly sampled signals: fundamentals and high-speed waveform digitizers. *IEEE Transactions on Instrumentation and Measurement*, 37(2):245–251, 1988. doi: 10.1109/19.6060.

[36] Baiying Yu and W.C. Black. Error analysis for time-interleaved analog channels. In *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, volume 1, pages 468–471 vol. 1, 2001. doi: 10.1109/ISCAS.2001.921894.

[37] Aaron Buchwald. Practical considerations for application specific time interleaved adcs. In *2015 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8, 2015. doi: 10.1109/CICC.2015.7338383.

[38] Yifan Ping, Xinquan Yang, Yin Kuang, and Wei Lin. Error detection and calibration

for time-interleaved adcs. In *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*, pages 1–4, 2014. doi: 10.1109/URSIGASS.2014.6929250.

[39] N. Kurosawa, H. Kobayashi, K. Maruyama, H. Sugawara, and K. Kobayashi. Explicit analysis of channel mismatch effects in time-interleaved adc systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48 (3):261–271, 2001. doi: 10.1109/81.915383.

[40] A. Petraglia and S.K. Mitra. Analysis of mismatch effects among a/d converters in a time-interleaved waveform digitizer. *IEEE Transactions on Instrumentation and Measurement*, 40(5):831–835, 1991. doi: 10.1109/19.106306.

[41] Mojtaba Bagheri, Filippo Schembari, Hashem Zare-Hoseini, Robert Bogdan Staszewski, and Arokia Nathan. Interchannel mismatch calibration techniques for time-interleaved sar adcs. *IEEE Open Journal of Circuits and Systems*, 2:420–433, 2021. doi: 10.1109/OJCAS.2021.3083680.

[42] David G. Nairn. Time-interleaved analog-to-digital converters. In *2008 IEEE Custom Integrated Circuits Conference*, pages 289–296, 2008. doi: 10.1109/CICC.2008.4672079.

[43] Manar El-Chammas and Boris Murmann. General analysis on the impact of phase-skew in time-interleaved adcs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(5):902–910, 2009. doi: 10.1109/TCSI.2009.2015206.

[44] Hao Zhang, Yibing Shi, and Zhigang Wang. Analysis of combined channel mismatch effects in time-interleaved adc systems. In *2009 IEEE Circuits and Systems International Conference on Testing and Diagnosis*, pages 1–4, 2009. doi: 10.1109/CAS-ICTD.2009.4960782.

[45] C. Vogel. The impact of combined channel mismatch effects in time-interleaved adcs. *IEEE Transactions on Instrumentation and Measurement*, 54(1):415–427, 2005. doi: 10.1109/TIM.2004.834046.

[46] C. Vogel and G. Kubin. Analysis and compensation of nonlinearity mismatches in time-interleaved adc arrays. In *2004 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages I–593, 2004. doi: 10.1109/ISCAS.2004.1328264.

[47] Microchip. Adc dc specifications, 2021. URL `https://microchipdeveloper.com/adc:adc-dc-spec`. last accessed on 2022-11-19.

[48] Chun-Po Huang, Hsin-Wen Ting, and Soon-Jyh Chang. Analysis of nonideal behav-

iors based on inl/dnl plots for sar adcs. *IEEE Transactions on Instrumentation and Measurement*, 65(8):1804–1817, 2016. doi: 10.1109/TIM.2016.2562198.

[49] Behzad Razavi. Problem of timing mismatch in interleaved adcs. In *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pages 1–8, 2012. doi: 10.1109/CICC.2012.6330655.

[50] Wann and Franklin. Asynchronous and clocked control structures for vsli based interconnection networks. *IEEE Transactions on Computers*, C-32(3):284–293, 1983. doi: 10.1109/TC.1983.1676220.

[51] Koji Asami, Hidetaka Suzuki, Hiroyuki Miyajima, Tetsuya Taura, and Haruo Kobayashi. Technique to improve the performance of time-interleaved a-d converters with mismatches of non-linearity. In *2008 17th Asian Test Symposium*, pages 105–110, 2008. doi: 10.1109/ATS.2008.34.

[52] Kenneth C. Dyer, John P. Keane, and Stephen Lewis. Calibration and dynamic matching in data converters: Part 2: Time-interleaved analog-to-digital converters and background-calibration challenges. *IEEE Solid-State Circuits Magazine*, 10(3): 61–70, 2018. doi: 10.1109/MSSC.2018.2844609.

[53] Tsung-Chih Hung, Fan-Wei Liao, and Tai-Haur Kuo. A 12-bit time-interleaved 400-ms/s pipelined adc with split-adc digital background calibration in 4,000 conversions/channel. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66 (11):1810–1814, 2019. doi: 10.1109/TCSII.2019.2895694.

[54] Sadeque Reza Khan, Arifa Ferdousi, and GoangSeog Choi. Hardware feasible offset and gain error correction for time-interleaved adc. In *2017 International SoC Design Conference (ISOCC)*, pages 254–255, 2017. doi: 10.1109/ISOCC.2017.8368881.

[55] Asgar Abbaszadeh and Khosrov Dabbagh-Sadeghipour. An efficient postprocessor architecture for channel mismatch correction of time interleaved adcs. In *2010 18th Iranian Conference on Electrical Engineering*, pages 382–385, 2010. doi: 10.1109/IRANIANCEE.2010.5507040.

[56] Gabriele Bè, Angelo Parisi, Luca Bertulessi, Luca Ricci, Lorenzo Scaletti, Mario Mercandelli, Andrea L. Lacaita, Salvatore Levantino, Carlo Samori, and Andrea Bonfanti. A 900-ms/s sar-based time-interleaved adc with a fully programmable interleaving factor and on-chip scalable background calibrations. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(9):3645–3649, 2022. doi: 10.1109/TCSII.2022.3182217.

[57] Nicolas Le Dortz, Jean-Pierre Blanc, Thierry Simon, Sarah Verhaeren, Emmanuel Rouat, Pascal Urard, Stéphane Le Tual, Dimitri Goguet, Caroline Lelandais-Perrault, and Philippe Benabes. 22.5 a 1.62gs/s time-interleaved sar adc with digital background mismatch calibration achieving interleaving spurs below 70dbfs. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 386–388, 2014. doi: 10.1109/ISSCC.2014.6757481.

[58] Jian Shen, Bo Yan, Shuisheng Lin, and Liang Zhou. A digital correction approach of mismatches for m-channel time-interleaved adcs. In *2013 International Conference on Communications, Circuits and Systems (ICCCAS)*, volume 1, pages 311–314, 2013. doi: 10.1109/ICCCAS.2013.6765240.

[59] Jie Pu, Daiguo Xu, Yuxin Wang, and Ruitao Zhang. A 10-bit 800ms/s low power time-interleaved sar adc with background calibration. In *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 1470–1472, 2016. doi: 10.1109/ICSICT.2016.7998772.

[60] K. Poulton, J.J. Corcoran, and T. Hornak. A 1-ghz 6-bit adc system. *IEEE Journal of Solid-State Circuits*, 22(6):962–970, 1987. doi: 10.1109/JSSC.1987.1052844.

[61] D. Fu, K. Dyer, S. Lewis, and P. Hurst. Digital background calibration of a 10 b 40 m sample/s parallel pipelined adc. In *1998 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, ISSCC. First Edition (Cat. No.98CH36156)*, pages 140–141, 1998. doi: 10.1109/ISSCC.1998.672407.

[62] Daihong Fu, K.C. Dyer, S.H. Lewis, and P.J. Hurst. A digital background calibration technique for time-interleaved analog-to-digital converters. *IEEE Journal of Solid-State Circuits*, 33(12):1904–1911, 1998. doi: 10.1109/4.735530.

[63] K. Dyer, D. Fu, S. Lewis, and P. Hurst. Analog background calibration of a 10 b 40 m sample/s parallel pipelined adc. In *1998 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, ISSCC. First Edition (Cat. No.98CH36156)*, pages 142–143, 1998. doi: 10.1109/ISSCC.1998.672408.

[64] K.C. Dyer, Daihong Fu, S.H. Lewis, and P.J. Hurst. An analog background calibration technique for time-interleaved analog-to-digital converters. *IEEE Journal of Solid-State Circuits*, 33(12):1912–1919, 1998. doi: 10.1109/4.735531.

[65] Digilent. Arty A7 Reference Manual, 2022. URL `https://digilent.com/reference/programmable-logic/arty-a7/reference-manual`. last accessed on 2022-11-10.

[66] AMD - Xilinx. Series 7 Product Selection Guide, 2021. URL `https://docs.xilinx.com/v/u/en-US/7-series-product-selection-guide`. last accessed on 2022-11-10.

[67] AMD - Xilinx. DS181 - Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics, 2022. URL `https://docs.xilinx.com/v/u/en-US/ds181_Artix_7_Data_Sheet`. last accessed on 2022-11-10.

[68] AMD - Xilinx. DS180 - 7 Series FPGAs Data Sheet: Overview, 2020. URL `https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview`. last accessed on 2022-11-10.

[69] AMD - Xilinx. UG1026 - Ultrascale Migration Guide, 2022. URL `https://docs.xilinx.com/v/u/en-US/ug1026-ultrascale-migration-guide`. last accessed on 2022-11-10.

[70] AMD - Xilinx. PG141 - DDS Compiler, 2022. URL `https://docs.xilinx.com/v/u/en-US/pg141-dds-compiler`. last accessed on 2022-11-10.

[71] Milan Stork and Jiri Hammerbauer. Recursive sine wave digital oscillator with new method used for amplitude control. In *2022 International Conference on Applied Electronics (AE)*, pages 1–4, 2022. doi: 10.1109/AE54730.2022.9920027.

[72] Milan Stork. Digital sinusoidal recursive oscillators with quadrature and three phase outputs. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2020. doi: 10.1109/MECO49872.2020.9134231.

[73] Florean Curticăpean and Jarkko Niittylahti. Complex digital oscillator with absolute periodicity. In *2000 10th European Signal Processing Conference*, pages 1–4, 2000.

[74] C.S. Turner. Recursive discrete-time sinusoidal oscillators. *IEEE Signal Processing Magazine*, 20(3):103–111, 2003. doi: 10.1109/MSP.2003.1203213.

[75] AMD - Xilinx. UG479 - 7 Series DSP48E1 Slice, 2018. URL `https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1`. last accessed on 2022-11-10.

[76] ZipCPU. Building a Numerically Controlled Oscillator, 2022. URL `https://zipcpu.com/dsp/2017/12/09/nco.html`. last accessed on 2022-11-10.

[77] AMD - Xilinx. UG473 - 7 Series FPGAs Memory Resources, 2019. URL `https://docs.xilinx.com/v/u/en-US/ug473_7Series_Memory_Resources`. last accessed on 2022-11-10.

[78] AMD - Xilinx. UG474 - 7 Series FPGAs Configurable Logic Block, 2016. URL

https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB. last accessed on 2022-11-10.

[79] ZipCPU. Building a quarter sine-wave lookup table, 2022. URL https://zipcpu.com/dsp/2017/08/26/quarterwave.html. last accessed on 2022-11-11.

[80] ARM. AMBA 4 AXI4-Stream Protocol Specification, 2010. URL https://developer.arm.com/documentation/ihi0051/a. last accessed on 2022-11-11.

[81] ARM. Part 1. AMBA AXI3 and AXI4 Protocol Specification, 2010. URL https://developer.arm.com/documentation/ihi0022/e. last accessed on 2022-11-11.

[82] AMD - Xilinx. UG472 - 7 Series FPGAs Clocking Resources, 2001. URL https://docs.xilinx.com/v/u/en-US/ug472_7Series_Clocking. last accessed on 2022-11-12.

[83] AMD - Xilinx. PG164 - Processor System Reset Module v5.0, 2015. URL https://docs.xilinx.com/v/u/en-US/pg164-proc-sys-reset. last accessed on 2022-11-12.

[84] AMD - Xilinx. PG059 - LogiCORE IP AXI Interconnect v2.0, 2013. URL https://docs.xilinx.com/v/u/2.0-English/pg059-axi-interconnect. last accessed on 2022-11-12.

[85] MathWorks. AXI Manager, 2022. URL https://it.mathworks.com/help/supportpkg/xilinxfpgaboards/axi-manager.html. last accessed on 2022-11-12.

[86] MathWorks. Ethernet AXI Manager, 2022. URL https://it.mathworks.com/help/supportpkg/xilinxfpgaboards/ug/ethernet-axi-manager.html. last accessed on 2022-11-12.

[87] AMD - Xilinx. DS875 - Integrated Logic Analyzer (ILA) v2.0, 2012. URL https://docs.xilinx.com/v/u/en-US/ds875-ila. last accessed on 2022-11-12.

[88] Sunghyuk Lee, Anantha P. Chandrakasan, and Hae-Seung Lee. 22.4 a 1gs/s 10b 18.9mw time-interleaved sar adc with background timing-skew calibration. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 384–385, 2014. doi: 10.1109/ISSCC.2014.6757480.

[89] Ce Fu and Hui Wang. Adaptive neural network filter in compensation of time interleaved a/d converter system. In *Proceedings of ICSSSM '05. 2005 International Conference on Services Systems and Services Management, 2005.*, volume 2, pages 1027–1030 Vol. 2, 2005. doi: 10.1109/ICSSSM.2005.1500148.

[90] Danfeng Zhai, Wenning Jiang, Xinru Jia, Jingchao Lan, Mingqiang Guo, Sai-Weng Sin, Fan Ye, Qi Liu, Junyan Ren, and Chixiao Chen. High-speed and time-interleaved adcs using additive-neural-network-based calibration for nonlinear amplitude and phase distortion. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–14, 2022. doi: 10.1109/TCSI.2022.3201016.

[91] Shreyas K. Venkataramanaiah, Shihui Yin, Yu Cao, and Jae-Sun Seo. Deep neural network training accelerator designs in asic and fpga. In *2020 International SoC Design Conference (ISOCC)*, pages 21–22, 2020. doi: 10.1109/ISOCC50952.2020.9333063.

List of Figures

List of Tables

List of Symbols

Variable	Description
T_S	sampling time
f_S	sampling frequency
M	interleaving factor
f_{in}	input frequency
ϕ	phase
ϕ_S	phase step
o_m	channel offset
O_n	DFT of the channel offsets
g_m	channel gain
G_n	DFT of the channel gains
t_m	channel phase skew
T_n	DFT of the channel time skews
α	core nonlinearity magnitude control
μ	average value
σ	standard deviation
γ_{offset}	offset calibration coefficient
γ_{gain}	gain calibration coefficient
f_{clk}	clock frequency
t_{sim}	simulation time
FoM	Figure of Merit
FoM_n	Normalized Figure of Merit

List of Acronyms

Acronym	Description
ADC	Analog-to-Digital-Converter
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BCA	Background Calibration Algorithm
BER	Bit Error Rate
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DC	Direct Current
DDR	Double Data Rate
DNL	Differential Non-Linearity
DSP	Digital Signal Processing
DTC	Digital-to-Time Converter
DUT	Device Under Test
EDA	Electronic Design Automation
ENOB	Equivalent Number Of Bit
FCW	Frequency Control Word
FIFO	First-In-First-Out
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
GMII	Gigabit Media Indipendent Interface
HBM	High Bandwidth Memory
HDL	Hardware Description Language
HW	Hardware
IC	Integrated Circuit
IIR	Infinite Impulse Response

Acronym	Description
ILA	Integrated Logic Analyzer
INL	Integral Non-Linearity
IP	Intellectual Property
LSB	Least Significative Bit
LUT	Look-Up Table
MAC	Medium Access Control
MII	Media Indipendent Interface
ML	Machine Learning
MM	Memory-Mapped
MSB	Most Significative Bit
NCO	Numerically Controlled Oscillator
NN	Neural Network
OVM	Open Verification Methodology
PCIe	Peripheral Component Interconnect express
PVT	Process, Voltage, Temperature
QLUT	Quarter LUT
RAM	Random Acess Memory
RA	Read Address
RD	Read Data
RP	Read Pointer
RF	Radio Frequency
RTL	Register Transfer Level
SAR	Successive Approximation Registers
SDP	Simple Dual Port
SDR	Single Data Rate
SINAD	SIgnal to Noise And Distortion Ratio
SFDR	Spurious Free Dynamic Range
SLR	Super Logic Region
SNDR	Signal to Noise and Distortion Ratio
SoC	System on a Chip
SoM	System on a Module
SoP	System on a Package
SPICE	Simulation Program with Integrated Circuit Emphasis
SV	System Verilog
S&H	Sample & Hold

Acronym	Description
TB	Test Bench
TCP	Transmission Control Protocol
TI	Time-Interleaved
UDP	User Datagram Protocol
USB	Universal Serial Bus
UVM	Universal Verification Methodology
VHDL	VHSIC Hardware Description Language
WA	Write Address
WD	Write Data
WP	Write Pointer
WR	Write Response

Ringraziamenti

Desidero ringraziare prima di tutto il mio relatore, Prof. Andrea G. Bonfanti, per l'immensa pazienza, la precisione e la disponibilità dimostrata durante la fase più importante del mio percorso accademico. Ringrazio anche il mio correlatore, Gabriele Be', per i continui suggerimenti e confronti durante tutte le fasi di questa tesi.

Un grande ringraziamento ai miei genitori, Guglielmo e Katia per il continuo sostegno ed incoraggiamento che non è mai venuto a mancare durante questi anni. Un enorme grazie anche a mia sorella, Flavia, e al mio piccolo fratellino peloso, Max.

Grazie anche a tutti i miei amici, con cui ho condiviso attimi di gioia e di tristezza, e che mi hanno supportato durante tutte le fasi della mia carriera universitaria.

Vorrei ringraziare anche i miei compagni di corso, per avermi sempre incoraggiato fin dall'inizio del percorso universitario, senza di voi non sarei mai arrivato fin qui.

In fine, un enorme grazie ai ragazzi di Skyward, con cui in questi anni ho condiviso grandi soddisfazioni.

Un sentito grazie a tutti!

Simone Guglielmino

Milano, 20 Dicembre 2022