Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo XXXII

Settore Concorsuale: 09/E3

Settore Scientifico Disciplinare: ING-INF/01

DESIGN TECHNIQUES TO ENHANCE LOW-POWER WIRELESS COMMUNICATION SOC WITH RECONFIGURABILITY AND WAKE UP RADIO

Presentata da: Francesco Renzini

Coordinatore Dottorato

Prof.ssa Alessandra Costanzo

Supervisore

Prof.ssa Eleonora Franchi Scarselli

Esame finale anno 2020

UNIVERSITÀ DI BOLOGNA

Abstract

ARCES-DEI

Doctor of Philosophy

Design Techniques to Enhance Low-Power Wireless Communication SoC with Reconfigurability and Wake Up Radio

by Francesco RENZINI

Nowadays, Internet of things applications are increasing, and each end-node has more demanding requirements such as energy efficiency and speed. The thesis proposes a heterogeneous elaboration unit for smart power applications, that consists of an ultra-low-power microcontroller coupled with a small (around 1k equivalent gates) soft-core of embedded FPGA. This digital system is implemented in 90-nm BCD technology of STMicroelectronics, and through the analysis presented in this thesis proves to have good performance in terms of power consumption and latency. The idea is to increase the system performance exploiting the embedded FPGA to managing smart power tasks. For the intended applications, a remarkable computational load is not required, it is just required the implementation of simple finite state machines, since they are event-driven applications. In this way, while the microcontroller deals with other system computations such as high-level communications, the eFPGA can efficiently manage smart power applications. An added value of the proposed elaboration unit is that a soft-core approach is applied to the whole digital system including the eFPGA, and hence, it is portable to different technologies. On the other hand, the configurability improvement has a straightforward drawback of about a 20–27% area overhead. The eFPGA usage to manage smart power applications, allows the system to reduce the required energy per task from about 400 to around 800 times compared to a processor implementation. The eFPGA utilization improves also the latency performance of the system reaching from 8 to 145 times less latency in terms of clock cycles. The thesis also introduces the architecture of a nano-watt wake-up radio integrated circuit implemented in 90-nm BCD technology of STMicroelectronics. The wake-up radio is an auxiliary always-on radio for medium-range applications that allows the IoT end-nodes to drastically reduce the power consumption during the node idle-listening communication phase.

Contents

List of Figures

List of Tables

List of Abbreviations

AMBA	**A**dvanced **M**icrocontroller **B**us **A**rchitecture
APB	**A**dvanced **P**eripheral **B**us
ASIC	**A**pplication-**S**pecific **I**ntegrated **C**ircuit
ASIP	**A**pplication **S**pecific **I**nstruction-set **P**rocessor
ASSP	**A**pplication **S**pecific **S**tandard **P**roduct
AXI	**A**dvanced e**X**tensible **I**nterface
BCD	**B**ipolar **C**MOS **D**MOS
CAD	**C**omputer **A**ided **D**esign
CLB	**C**onfigurable **L**ogic **B**lock
CMOS	**C**omplementary **MOS**
CRC	**C**yclic **R**edundancy **C**heck
CTS	**C**lock **T**ree **S**ynthesis
DMOS	**D**ouble-diffused **MOS**
DSP	**D**igital **S**ignal **P**rocessor
DTI	**D**eep **T**rench **I**solation
eFPGA	**e**mbedded **FPGA**
FD-SOI	**F**ully **D**epleted **S**ilicon **O**n **I**nsulator
FET	**F**ield **E**ffect **T**ransistor
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FPU	**F**loating **P**oint **U**nit
FSM	**F**inite **S**tate **M**achine
GPIO	**G**eneral **P**urpose **I**nput/**O**utput
HDL	**H**ardware **D**escription **L**anguage
IC	**I**ntegrated **C**ircuit
IoT	**I**nternet **o**f **T**hings
ISR	**I**nterrupt **S**ervice **R**outine
LB	**L**ogic **B**lock
LCD	**L**iquid **C**rystal **D**isplay
LFSR	**L**inear **F**eedback **S**hift **R**egister
MAC	**M**ultiplier **AC**cumulator
MOS	**M**etal **O**xide **S**emiconductor
MSSN	**M**ulti**S**tage **S**withing **N**etwork
NRE	**N**on-**R**ecurring **E**ngineering
OOK	**O**n-**O**ff **K**eying
PLD	**P**rogrammable **L**ogic **D**evice

PWM	**P**ulse **W**idth **M**odulation
RF	**R**adio **F**requency
RISC	**R**educed **I**nstruction **S**et **C**omputer
RNB	**R**earrangeable **N**on-**B**locking
RTL	**R**egister **T**ransfer **L**anguage
SE	**S**witching **E**lement
SoC	**S**ystem **o**n **C**hip
SRAM	**S**tatic **R**andom **A**ccess **M**emory
uC, μ**C**	**M**icro**C**ontroller
ULP	**U**ltra **L**ow-**P**ower
uP, μ**P**	**M**icro**P**rocessor
VCD	**V**alue **C**hange **D**ump
VTR	**V**erilog-**T**o-**R**outing
WSN	**W**ireless **S**ensor **N**etwork
WUR	**W**ake **U**p **R**adio

Chapter 1

Introduction

The well known Internet-of-Things (IoT) domain is becoming one of the most popular scenarios in every human activity field. In the IoT arena, everything, each object is connected to the others, composing a network. For instance, Wireless Sensor Networks (WSNs) are constantly growing in every kind of monitoring activities such as biomedical, structural, etc. IoT devices are becoming popular in many different applications for example domotic, industrial, automotive, etc. Hence, every network node should be able to carry out its native tasks and besides communicate with the other network elements. Thus, they can sense physical quantity, actuate some control policies and elaborate data based on the application requirements. In addition, they have a communication channel typically wireless (for obvious necessities) to interact with the other end-nodes. The development of these systems is subordinate to classical electronic system constraints such as energy efficiency, small size, and low cost. The energy restrictions and especially the required low power consumption, are due to both "green" environmental policies and evident technical issues since each element of the network could have battery power supply. Then, the research aims to improve both energy performance and energy efficiency in every end-node component. For instance, this thesis presents an architecture that provides configurability to the elaboration unit increasing its energy efficiency. It is also presented an additional circuit that allows the end-nodes to drastically reduce its power consumption during the idle-listening phase.

These systems are typically realized in printed circuit board systems combining many different integrated circuits as Application Specific Standard Products (ASSPs), but nowadays, the technology scaling allows one to integrate as much as possible in a unique integrated circuit resulting system-on-chip. These system-on-chips can integrate different kinds of programmable and non-programmable elaboration units, communication systems and a huge variety of dedicated peripherals. The elaboration unit can be non-programmable like Application-Specific Integrated Circuit (ASIC) allowing the best performances but implying both non-negligible Non-Recurring Engineering (NRE) costs and the impossibility to reuse the same SoC in different applications. On the other hand, there exist programmable elaboration units such as Application Specific Instruction-set Processors (ASIPs), CPUs or Programmable Logic Devices (PLDs) which allow one to drastically reduce NRE costs and thanks to programmability to reuse the same SoC for different applications. Typically, the communication protocols between each end-node can vary from the transmission medium (wired or wireless), data distribution (serial or parallel) and clock behavior (synchronous or asynchronous), etc. For instance, SoCs can have either industrial or automotive fieldbus interfaces, otherwise low-power wireless communication systems.

Communication interfaces have not negligible impact on the overall system power consumption, hence, are requested optimization design procedures at each layer of the communication system. For instance, every network node has an idle-listening phase where the node turns on the RF receiving circuit waiting for instruction or data from other network elements. In order to avoid data package lost, this phase should be well extended and may be redundant, which results in a non-negligible extra power consumption. Therefore, to increase energy efficiency could be necessary an architectural change of the classical system structure. Thus, semiconductor companies are designing different kinds of SoCs for different types of applications, to optimize the offered capabilities and hence, the performances. These ICs are literally system-on-chips, since they can also have embedded sensors and in some cases power devices necessary to the cyber-physical interaction.

In this thesis, we are focusing on the smart power arena which is an excellent scenario for the SoC development, since smart power combines all the SoC features, from physical quantity sensing, both data elaboration and communication, and decision actuation to power circuits. Thus, in these systems, there is a coexistence of analog, digital and power circuits. In this kind of application, the elaboration unit does not need extremely high computation performance, since smart power tasks require just simple finite state machines as a computational base model instead of complex parallel computing structures. Thus, the elaboration unit should generate output patterns based on inputs and not intensively process data. This type of controller is very diffused for example in motion control, lighting control, and power management. In order to realize integrated circuits provided with analog, digital and power transistors there exists the BCD (Bipolar CMOS DMOS) technology. Certainly, this coexistence implicates numerous issues in terms of isolation, reliability, etc. addressed by both process and circuit designers, making this technology appealing for widespread usage in the IoT arena. Moreover, CMOS technology node of BCD technology is not too scaled as standard CMOS technologies, historically targeting ASIC solutions, whereas currently, BCD is becoming appealing for higher complexity CMOS programmable circuits since it can have 130 nm and 90 nm transistors even if it provides just few metal layers for the routing.

In this scenario, the core of my research is to design and evaluate the performance of an elaboration unit that consists of a microcontroller and an embedded FPGA (eFPGA). Starting from the system HDL code, to the layout generation, the elaboration unit is implemented in 90 nm BCD technology of STMicroelectronics.

The performance evaluation is performed comparing data from back-annotate simulation and measurements (when possible) of different elaboration unit implementations, to carry out various kinds of applications. Starting from both microcontroller HDL code and eFPGA HDL code (presented by a previous Ph.D. student), through a standard cell-based digital design flow the system is implemented in BCD technology, which is, to the best of our knowledge the first reconfigurable heterogeneous system targeting smart power applications. The performance evaluation is carried out comparing both energy and latency data of different devices, in carrying out different kinds of applications, corresponding to different kinds of computational models.

Besides, the research work has also contributed to design a wake up radio integrated circuit in 90 nm BCD technology of STMicroelectronics, sent to fabrication. In this collaboration, the PhD work focuses on the design of some blocks of the wake up radio system at the transistor level, performing a full custom layout.

The proposed elaboration unit is a heterogeneous system (presented in [1]) composed of PULPino, which is an open-source ultra low-power microcontroller [2], coupled with a very small fully-synthesizable embedded FPGA [3], [4]. The idea is to use the eFPGA as a smart power application dedicated peripheral, while microprocessor can manage other kinds of more complex computations such as data processing, communication interface management, etc., or can be switched in sleep mode to reduce power consumption. This is also a new point of view for the eFPGA usage, typically used for high-density parallel computing and hence, usually designed with a hard-macro approach, optimizing the circuit at the transistor level to optimize density, performances and thus costs. On the other hand, the reconfigurability addition has a non-negligible area overhead which is around 20%.

In order to justify the area overhead due to the eFPGA addition, it is evaluated an energy-aware analysis comparing the performance of PULPino, of the eFPGA, of a very common commercial microcontroller (STM32) and an ASIC implementation to carry out different applications. The resulting data show that using the eFPGA to perform smart power tasks, hence implementing simple finite state machines, is possible to obtain better performance in terms of energy consumption. Obviously, the eFPGA does not have the same performances of an ASIC implementation but it allows the system to be reused in different applications without integrated circuit refabrication. The energy gain reason is located in the nature of the different architecture circuits. eFPGA directly maps in hardware circuit a finite state machine whereas processor needs to execute many load/store, jump/branch instructions just to update few bits of both finite state machine state and outputs. There is also an advantage in terms of circuit latency, in fact, the eFPGA updates its output at every clock cycle and hence, in this case, the latency depends on the eFPGA clock frequency. Then, the SoC user can program the eFPGA clock frequency based on the application latency requirements. Using a processor implementation, the latency is related to both the number of instructions needed to update the microprocessor outputs and the processor frequency (supposing the optimistic case that the processor can execute one instruction per clock cycle).

As already mentioned, the communication interface has a notable impact on system power consumption. Therefore, in order to improve the system-on-chip energy performance, we propose an additional always-on ultra low-power circuit for the wireless communication unit. The additional circuit aims to wake up the rest of the system after the receiving of a wake up radio signal transmitted from another element of the network. In this way, one can switch in sleep mode both the elaboration and the main radio transceiver, to reduce the power consumption, while keeping active just the wake up radio which is an ultra low-power circuit.

We propose a nano-watts wake up radio for On-Off Keying modulation working at 868 MHz, which consists of a demodulator/amplifier and a Schmitt trigger with 5-bits programmable threshold to generate a digital signal. The operating point of the overall wake up radio system is designed to be in subthreshold region. Then, exploiting the non-linearities of subthreshold MOSFET, the demodulator can detect the envelope of the received signal and provides the right amplification. Then, the detected envelope is digitalized through the Schmitt trigger. Thanks to enough level of amplification, the system performances are not restricted to the Schmitt trigger input offset voltage.

As already said, my research aims to analyze the proposed heterogeneous reconfigurable SoC in the smart power arena and to present our wake up radio integrated circuit. Therefore, the thesis is organized following a system top-down approach as follows:

- **Chapter 2** provides an introduction to the smart power arena. In particular, are explained the typical features of smart power applications and microelectronic technology which targets this kind of applications. In addition, are introduced different architectures to realize elaboration units.

- **Chapter 3** presents our proposed solution to implement the elaboration unit in the smart power arena, combining both an ultra low-power microcontroller and a fully-synthesizable embedded FPGA.

- **Chapter 4** reports our embedded FPGA architecture description.

- **Chapter 5** describes an energy-aware analysis in order to compare the performance of different digital units, such as eFPGA, PULPino, ASIC and STM32 to perform different kinds of applications. This chapter also addresses some considerations regarding the latency performance of the previous digital circuits.

- **Chapter 6** provides the architecture of our nano-watts wake up radio IC introducing the working principle of the demodulator/amplifier (designed by a colleague of mine) and analyzing blocks designed by me.

- **Chapter 7** finally summarizes the conclusions.
- **Appendix A** focuses on the proprieties of a multi-stage switching network used as an interconnection network of our proposed eFPGA.

Chapter 2

SoC Architectures for Smart Power Applications

Most of the material reported in this chapter is reused from [1] (©2019 IEEE), in agreement with IEEE copyright policy on theses and dissertations.

IoT, as introduced in the previous chapter is going to be one of the dominant scenarios of the next generation electronic systems. There exist many different kinds of connected applications, and the smart power area is entering among them thanks to the technology evolution as shown in the following of this chapter. This chapter provides an introduction to the smart power area and an extensive elaboration unit state of the art analysis, concluding with a system-on-chip architecture proposal.

2.1 Introduction

One of the definitions of smart power application is introduced from one of its inventors in [5], and it is the coexistence of both "force" - power electronic - and "intelligence" - digital circuits - in the same chip. Evidently, the cooperation between "force" and "intelligence" requires analog circuits too.

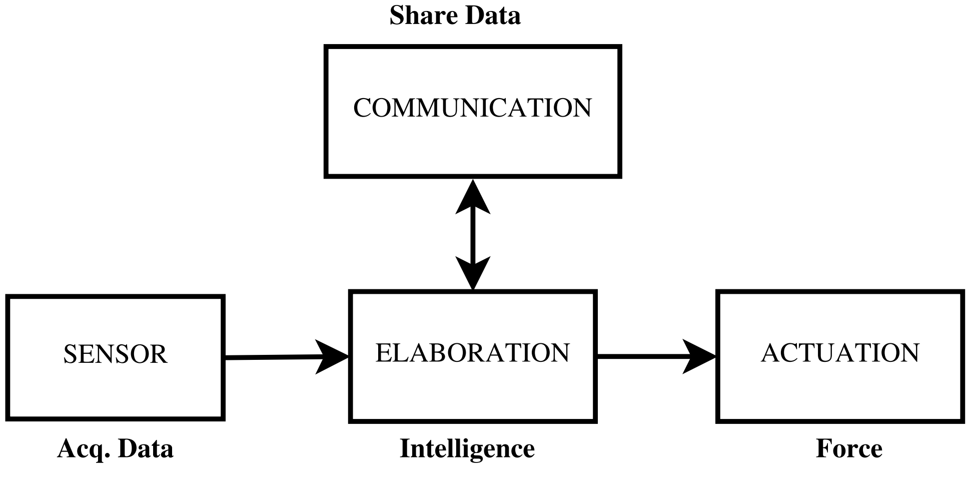


Figure 2.1: Smart power system diagram.

In Figure 2.1 is reported the basic diagram of a smart power system in the IoT scenario. Sensor and actuator allow the system to interact with the physical world. Sensors acquire data, an elaboration unit computes data based on the application specifications, hence a control policy for the actuation circuits. These three blocks constitute the classical feedback control system. For the IoT scenario, is needed a communication interface in order to share both data and instructions with the rest of the network. Sometimes the feedback control loop could be open or pass through the communication interface resulting in:

- *monitoring activities* where the system acquires, elaborates and shares data in the network (which could close the control loop) without local actuation circuits.
- *actuation activities* without local sensors, the system receives commands from the network as inputs.

Classical smart power applications are in the field of:

- power management
- motion control [6]
- lighting control [7]

where the end-nodes have to handle control policies and interact with the real world. As reported in Chapter 5, for instance, a common case is the application of the Pulse Width Modulation (PWM) in switching power converters and in both brushed motor and LED controllers.

In this scenario, the elaboration unit is relatively simple because the application computational complexity requirement is not extreme, which means that is not required high-performance computing. Typically in this kind of application, the elaboration unit decides some outputs for the actuators based on inputs from both sensors and input user. Hence, in this case, the computational base model of the elaboration unit is a Finite State Machine (FSM).

Generally, the elaboration unit can be either software-programmable, hardware-programmable or hardwired in an Application Specific Integrated Circuit (ASIC) as well as a combination of these solutions. The software-programmable solution executes software code and it is the most flexible and it allows one to use microprocessor (μP) or Digital Signal Processor (DSP). The hardware-programmable devices are Programmable Logic Devices (PLDs), and nowadays led by Field Programmable Gate Arrays (FPGAs). These devices have configurable architecture which is adapted based on the required functionality. PLDs are typically more powerful and efficient than software-programmable solutions to compute-intensive and parallel tasks since their architecture is configured to optimized the specific task, contrary to software-based devices which have a fixed architecture. The drawback of the PLD exploitation is the development time compared to the more easy of use software-based devices. ASIC implementations have the best performance in terms of power consumption and area occupation; however, the hardwired structure does not allow the device reuse in different applications involving substantial Non-Recurring Engineering (NRE) costs. The miniaturization of the devices allowed by the microelectronic technology, offers the possibility to realize microcontrollers (μCs) that integrate software-programmable solution such as DSPs and/or μPs with hardwired circuits used as dedicated peripherals. These peripherals, which are ASICs, are for example General Purpose I/Os (GPIOs), Pulse Width Modulation (PWM) controller, analog-to-digital and digital-to-analog converters, are necessary to interact with the real physical world. More complex devices are System-on-Chips (SoCs), which typically consist of high-performance computing microcontroller coupled with embedded FPGA (eFPGA). Usually, the elaboration unit of smart power ICs is based on ASIC implementation due to the relatively low computing complexity, but the constantly increase of application requirements and the reusability necessity are moving the smart power IC architecture to programmable solutions which are supported by technology evolution.

2.2 Smart Power Technology

Following the application evolution, microelectronics technologies are evolving too, specializing for the specific application requirements. For example, FinFET technology allows the extreme transistor scaling supporting the advanced digital circuit development. Regarding smart power applications, the best microelectronic technology is BCD technology. The BCD technology allows one to integrate into a unique die bipolar transistor (typically used in analog circuits), standard CMOS circuits for both digital and analog circuits, and DMOS transistors for high-power electronics [8] as reported in Figure 2.2.

BCD technology has proven to be suitable for sensors [10] and actuators [11]. Traditionally, the systems realized in BCD technologies were implemented as ASICs (without any programmable functionalities) due to the less scaled features of this technology with respect to traditional CMOS and to the limited amount of metal layers, which increase the overhead of the routing. However, the scaling of BCD's CMOS transistors, down to 90 nm and beyond, makes this technology suitable for the integration of mid-complexity digital circuits like μCs. This allows one to enable BCD utilization in the IoT arena [12], which requires additional features such as wireless stack implementation, radio interfaces, etc. The ICs realized in BCD technology are designed using an analog-on-top methodology, since the area occupation is usually dominated by both power and analog circuits. Hence, digital CMOS circuits are typically synthesizable in standard cell libraries, in order to adapt the digital

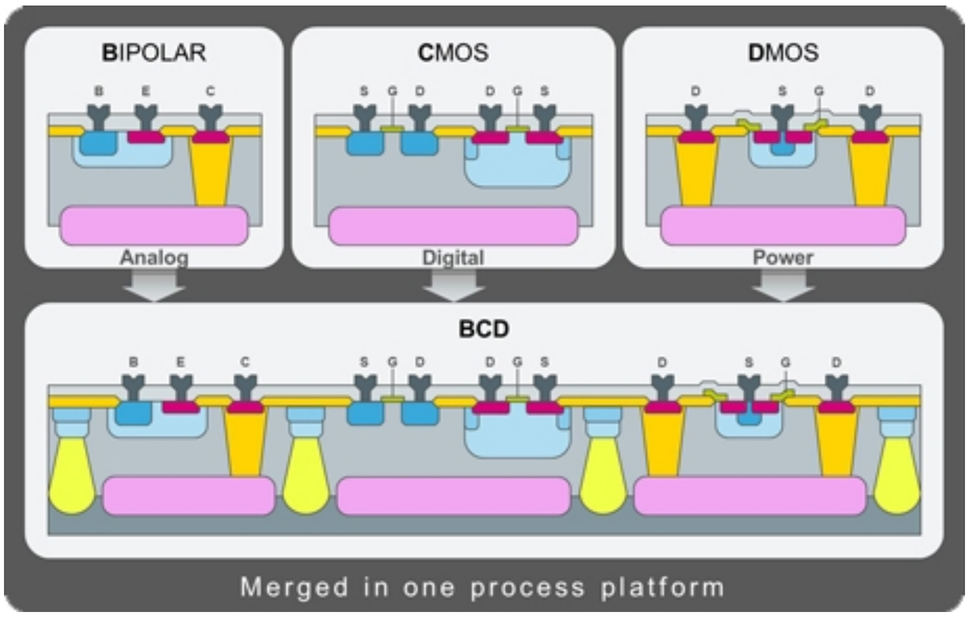


Figure 2.2: BCD technology [9].

circuit floorplan between the other more critically analog and power circuit layouts. For this reason, a hard-macro approach should be avoidable. In order to optimize the IC area occupation, led by both analog and power circuits, the smart power IC elaboration unit should be adaptable and hence synthesizable in standard cells. The limited amount of metal layers makes the device area density lower than standard CMOS technology. Besides, the coexistence of both, high power devices like DMOS and low power devices such as MOSFETs and bipolar transistors, requires very safe isolation technique, since power transistors can work kilovolts. In STMicroelectronics BCD technology the isolation is made through Deep Trench Isolation (DTI) which requires not negligible area overhead. Since smart power applications are very common in further scenarios such as automotive, BCD technology is qualified also for high temperatures, for example up to 150 °C for automotive.

Therefore, the digital circuit for smart power applications should be flexible, synthesizable and of course low-power. Next section shows some techniques to reduce the circuit power consumption.

2.3 ULP Techniques

On top of that, in the "green" era of the energy-saving, especially needed in the IoT scenario, Ultra Low-Power (ULP) strategies can be applied at every level (permitted by available technologies) from transistor level to system architecture level. Some of these ULP methodologies are for instance [13]:

- *Multi-V_{th}* allows one to use transistors with different threshold voltage in order to decrease the leakage current. Different threshold voltage can be realized, for instance, using transistors with different channel lengths.

- *Multi-L* is typically a technique for digital circuits, where are available different kinds of standard cells with different transistor channel length L. The usage of bigger channel lengths, reduces the power consumption related to the leakage current, but on the other hand, decreases the standard cell timing performance.
- *Multi-V_{DD}* permits to have different circuits with different supply voltages to optimize power consumption. The drawback of this technique is that it is necessary to use level shifter circuits between the different voltage islands.
- *Frequency scaling* allows one to modulate the operating clock frequency and hence, the dynamic power consumption, based on the application requirements, avoiding high dynamic power consumption when it is not needed. In the case of multi-frequency domains, at the interfaces of each clock domain are required synchronizer circuits.
- *Clock gating* allows one to switch off the block clock lines in order to eliminate the dynamic power consumption when the block activity is not required. In this case, are necessary clock gating circuits between clock sources and input clock lines of the blocks.
- *Power gating* consists of switching off the power supply of some circuits when they are not required. In this way, one can optimize the power consumption avoiding energy waste. On the other hand, are required both transistors properly design as power switches and a good power distribution network.

If it is possible they can be applied to every block of the electronic system of Figure 2.1. For instance, in BCD technology one can use some of the mentioned techniques, but we used just *multi-L* and *clock gating* methods since they are available in the standard cell libraries. The other techniques require additional and often complex supplementary circuits to manage them.

2.4 Programmable Device Scenario

In this section is provided an analysis of the available solutions to implement programmable elaboration units. In Table 2.1 is reported a summary of the programmable devices scenario, from a classical microcontroller (μC) to complex SoC enhanced with eFPGAs that allows one to implement a custom digital elaboration unit. Table 2.1 reports the field application of the programmable devices, the microelectronic technology used, its corresponding number of metal layers and the operating temperature range. It is also reported the used design technique which can be based on either a soft-core approach "soft" hence synthesizable in standard cell libraries, or a hard-macro approach "hard" optimizing the circuit at the transistor level. The area, coupled with both technological information and application scenario gives an

Table 2.1: Programmable Device Scenario. Reused from [1].

		Applications	Technology	Metal Layers	Temp. Range [°C]	IP	Area [mm^2]	embedded MCU/FPGA	eq. gates
μC	STM32 [14]	Gen. Purpose	CMOS	n.a.	-40–125	soft	medium	✓/✗	n.a.
	PULPino[a][2]	Gen. Purpose	CMOS 65 nm	9	-40–125[b]	soft	medium	✓/✗	n.a.
FPGA	Xilinx [15]	Gen. Purpose	Adv. CMOS	> 10	-55–125	hard	huge	✗/✓	$\leq$ M
	Intel [16]	Gen. Purpose	Adv. CMOS	> 10	-40–130	hard	huge	✗/✓	$\leq$ M
	QuickLogic [17]	Gen. Purpose	Adv. CMOS	> 10	-55–125	hard	huge	✗/✓	$<$ M
μFPGA	iCE40 [18]	Rec. I/Os/Accel.	CMOS 40 nm	n.a.	-55–125	hard	small	✗/✓	0.4–8 k
	IGLOO2 [19]	Rec. I/Os/Accel.	CMOS 65 nm	n.a.	-55–125	hard	small	✗/✓	$<$ 150 k
eFPGA	[17], [20], [21], [22]	Accelerator	Adv. CMOS	n.a.	-40–125	hard	big	✗/✓	$<$ M
	Kim2017 [23]	Accelerator	CMOS 65 nm	n.a.	n.a.	soft	n.a.	✗/✓	n.a.
	Cuppini2015 [3]	Accelerator	CMOS 65 nm	7	-40–125	soft	0.4/0.2[c]	✗/✓	1 k
	Cuppini2015 [3]	Smart Power	BCD 110 nm	4	-40–150	soft	1.2/0.7[c]	✗/✓	1 k
SoC	Borgatti2003 [24]	Reconf. I/Os	CMOS 180 nm	6	n.a.	hard	20 (8.2)[d]	✓/✓	15 k
	XiSystem [25]	Periph/Acceler	CMOS 130 nm	6	-40–125[b]	hard	42 (6)[d]	✓/✓	15 k
	Morpheus [26]	Periph/Acceler	CMOS 90 nm	7	-40–125[b]	hard	110	✓/✓	15–100 k
	[15], [16]	Gen. Purpose	Adv. CMOS	> 10	n.a.	hard	huge	✓/✓	$\leq$ M
	Proposed SoC	Smart Power	BCD 90 nm	5	-40–150	soft	1.78 (0.347)[d]	✓/✓	1 k

[a] Imperio implementation (`http://asic.ethz.ch/2015/Imperio.html`)
[b] Personal communication
[c] Max speed/Min area implementations
[d] System area (eFPGA area)

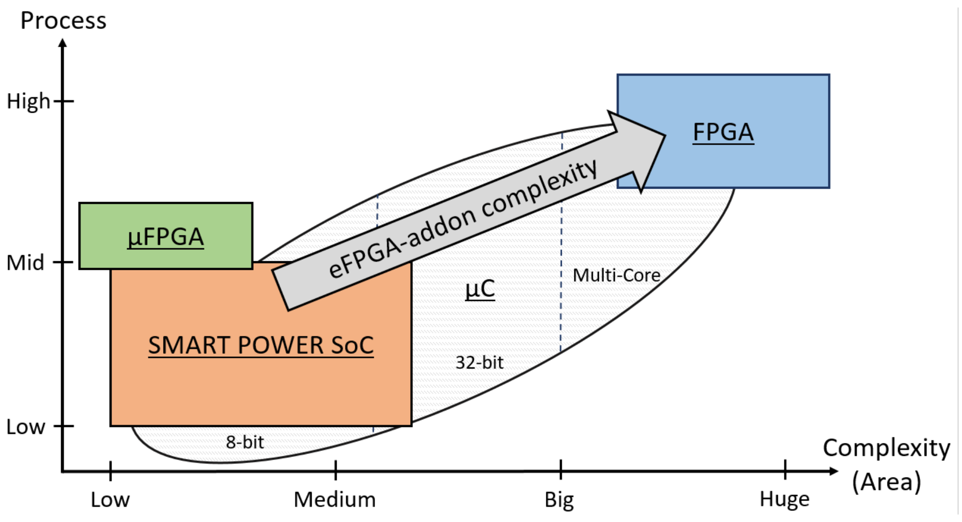


Figure 2.3: Programmable devices scenario. Picture reused from [1].

idea of the system complexity. In addition, it is reported which one between software and hardware programmable feature is present. The last column of Table 2.1 "eq. gates", expresses the computational capability of the hardware-programmable device if it is present in the device. In Figure 2.3 is showed a qualitative analysis of the manufacturing costs, stated as process complexity, versus system complexity for different programmable device solutions.

2.4.1 Microcontrollers

Nowadays, microcontrollers are probably the most diffused programmable devices thanks to their ease of software programmability. They integrate a classical microprocessor architecture coupled with a variety of different peripherals. Every semiconductor companies provide microcontrollers for different application requirements, from 8-bit core for simple applications to high-end 32-bit multi-core μC [27] tailored for high-performance computing. The application needs drive also both the amount and the complexity of the peripherals; almost every microcontrollers have a standard set of peripheral for instance programmable PWM controller, GPIOs, timers, etc. There are also available μC equipped for example with Floating Point Unit (FPU), camera controller, fieldbus interfaces, LCD controller, etc. [28] since the main idea is to move the computational task for a specific application from the processor to a more efficient peripheral/IP connected to the microcontroller bus. Microcontrollers have also been developed for either specific sensing activities for instance for ultrasonic applications [29], or actuation activities equipped with brushless motor controller [30] or for radio communication tasks providing the microcontroller of RF circuits [31]. In Table 2.1 we refer to a family of widespread commercial μC [14] and to an open-source solution [2] based on the RISC-V architecture. For their nature, they are general-purpose devices and both of them are realized in standard CMOS technology using a soft-core approach. As already explained one of the most attractive features of microcontrollers is the ease of programmability, but on the other hand, μCs do not allow one to design an ad-hoc dedicated controller. Microcontrollers are ASICs provided of software-programmability characteristics and then they have a non-reconfigurable hardware structure. Hence, if it is necessary to design an own custom controller it must be implemented in software (maybe interacting with peripherals) which results in possible energy waste and latency issues. As reported in Figure 2.3, the microcontroller manufacturing costs is obviously growing with the microcontroller complexity.

2.4.2 FPGAs

Nowadays FPGAs, the most diffused PLDs, are widely used in a wide range of applications. They were typically used for extreme computing thanks to their available parallel structure, which allows one to maps in a reconfigurable hardware structure a specific computing accelerator. In this way, one can design a custom and parallel accelerator instead of serialize the task in a microprocessor code execution. An example of parallel structure exploitation is a neural network accelerator [32]. In addition, FPGAs are used in power electronic applications in order to compensate for the increasing demand of reactivity [33]. Generally, FPGAs are designed following a hard-macro approach, which consists of the optimized basic block repetition. In order to optimize the FPGA performance in terms of power consumption, area occupation, and timing performance, one designs the basic block optimized at the transistor level. The area optimization is a key factor especially for this kind of device because the goal is to realize huge dimension FPGAs (order magnitude of mega equivalent gates) and hence, expensive technologies are used such as 130 to 40 nm CMOS, 22 nm FD-SOI ot advanced FinFET CMOS technology. In addition, these advanced technologies have a greater number of metal layers (more than 10) than standard technologies for both routing (which make the place and route easier) and power distribution. Therefore, as reported in Figure 2.3, FPGAs are very complex and for that reason very expensive devices. In Table 2.1 FPGAs are reported as devices for general purpose applications since they have huge computational capability even if they do not have embedded microprocessors. However, it is possible to add software-programmability features to FPGAs implementing soft-processors, which means configure the FPGA with a processor architecture. On the other hand, the adoption of soft-processor doesn't guarantee high energy efficiency because the processor is mapped in programmable-hardware and not in custom circuits. Thus, FPGAs are oversized for smart power applications which typically don't require high-density and parallel computing. Another drawback is that extremely advanced technologies used to fabricate FPGAs generally are not qualified for high temperatures such as 150 °C as shown in Table 2.1. In order to avoid some of the previous issues, some semiconductor companies such as [18] and [19] have designed small FPGAs, μFPGA, in non-advanced CMOS technologies in order to reduce the device costs. This kind of FPGAs targets both reconfigurable I/Os and simple accelerators and their computational capability goes from 0.4 kilo equivalent gates to less than 150 kilo equivalent gates as reported in Table 2.1. Just as reported in Figure 2.3, μFPGAs can be the right size for smart power applications, but they are still without processors limiting the flexibility.

2.4.3 Embedded FPGAs

Embedded FPGAs are useful to increase the system flexibility to have a hardware-programmable IP. In this way, one can design the own custom digital circuit to specific tasks, improving the performance in terms of speed and efficiency. They are typically used as hardware accelerators, for instance, specific algorithms sensor interface preprocessors for machine learning, etc. Several semiconductor companies are developing embedded FPGA IPs, such as [17], [20]–[22]. These commercial eFPGAs are large-size devices (up to 1 mega equivalent gates) usually designed with a hard-macro approach, optimizing the eFPGA at transistor level, obviously to optimize the area density and maximize speed performance as reported in Table 2.1. Vendors design eFPGAs based on the customers' needs which can be both a specific technology and needed device size. Since eFPGAs have typically big dimensions, the used technologies are from standard to advanced CMOS technologies as reported in Figure 2.3, in order to both employ as few metal layers as possible to reduce the manufacturing costs and increase density to improve the yield. Smart power ICs are generally analog-on-top because area occupation is dominated by both power and analog circuits, hence the use of a hard-macro eFPGA could be unworkable and therefore a soft-core approach is mandatory. There exist then synthesizable eFPGAs such as the one proposed in [23], which uses Verilog-to-Routing (VTR) CAD flow. VTR is an open-source tool that allows one to map a digital circuit on an island-style FPGA that consists of two-dimensional arrays of LB with vertical and horizontal routing channels. In addition, one can also find synthesizable eFPGAs based on a datapath-oriented architecture [34]. As shown in the following chapter and Table 2.1, our proposed eFPGA is fully-synthesizable [3] and it has a very small computational dimension (1 kilo equivalent gates) and can fit both standard CMOS technology at 65 nm with 7 metal layers and BCD technology with just 4 metal layers. Hence, the proposed eFPGA fits smart power application needs. As it is provided in Chapter 3 and Chapter 4, the interconnection network is based on a multi-stage switching network that allows full-routability and provides sustainable area overhead in small-size devices.

2.4.4 System-on-Chips

System-on-Chips (SoCs) provide the best system flexibility since they combine both software- (microprocessors) and hardware- (eFPGAs) programmability in a unique integrated circuit. Thus, they are attractive as digital elaboration unit. Theoretically SoCs can cover all the area of Figure 2.3, however, they are usually used as either

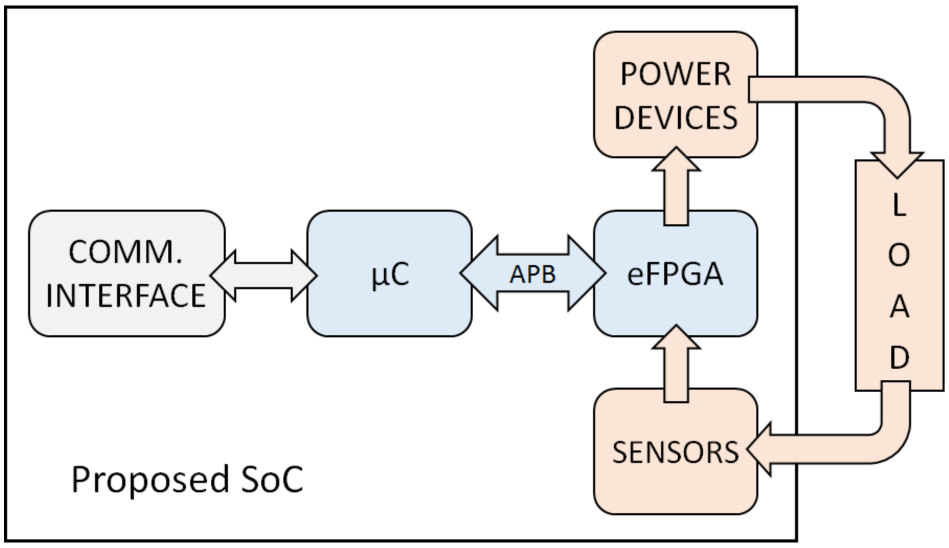


Figure 2.4: Proposed SoC architecture.

reconfigurable I/Os (as proposed in the pioneering work [24]) or both hardware and peripheral accelerators such as Ethernet MAC, binarization, etc. as presented in [25] and [35]. Nowadays, semiconductor companies such as [15] and [16] which historically produced high-end standalone FPGAs are now also producing high-end SoCs in advanced CMOS technologies provided with huge computational capabilities. Then, all the cited SoCs are really too complex, and hence expensive, for the smart power arena. In addition, at least the embedded hardware-programmable IPs are designed following a hard-macro approach and thus, most of the previous considerations made in Section 2.4.2 for standalone FPGAs are still applicable, as reported in Table 2.1.

2.5 Proposed SoC

In the smart power arena, we imagine the system-on-chip represented in Figure 2.4 which integrates the features of Figure 2.1. The proposed SoC elaboration unit is a heterogeneous system composed of a microcontroller and an eFPGA interfaced through the microcontroller peripheral bus. In this way, the eFPGA acts as a reconfigurable peripheral of the microcontroller and it should manage a specific task by itself, but also if it is required, the processor can easily interact with the eFPGA. For instance, in the smart power scenario, the eFPGA can implement some control policy and can directly manage the power devices for the actuation and receive feedback from sensors. For connected applications, the proposed SoC should also have a communication interface in order to interact with the other network elements. Therefore, the idea is to have a configurable peripheral that handle the smart power task without any cooperation with the processor. Thus, the microprocessor can manage both other kinds of computation useful to the system and the communication interface to send/receive data and instructions to/from the network. Besides, the microprocessor can be switched in sleep mode to reduce the power consumption while its peripherals are working on their specific tasks, and vice versa, the processor can switch off the eFPGA clock when it is not required for the application. In this case, the system designer can partition tasks addressing both latency and energy efficiency requirements as explained in the following of the thesis. The full-programmability of the elaboration unit allows one to reuse the same system-on-chip for different applications with a considerable impact on the costs.

As provided in Chapter 3, the proposed SoC has both software- and hardware-programmability features, in fact, it has a microcontroller and an eFPGA. The overall system-on-chip elaboration unit is fully-synthesizable in standard cell libraries and hence, it is designed using a soft-core approach. The system-on-chip is implemented in BCD technology at 90 nm of STMicroelectronics with 5 metal layers, as shown in Table 2.1. Considering the computational dimension of our SoC's eFPGA, which is 1 kilo equivalent gates, the proposed SoC elaboration unit fits the simpler smart power application requirements, becoming an interesting solution for the smart power scenario. The proposed elaboration unit for system-on-chip combines both system flexibility due to the fully-system-programmability and medium-complexity (as shown in Figure 2.3) since the elaboration unit integrates a "small" heterogeneous system. Indeed our proposed elaboration unit combines a 32-bit microcontroller coupled with our eFPGA, which, on the computational point of view, is comparable to a μFPGA described before. Nevertheless, the overall system is implemented in a smart power technology and not standard CMOS technology as the previous solutions.

Regarding the communication interface of Figure 2.4, Chapter 6 provides a description of the proposed technique to decrease the power consumption of the communication interface.

Chapter 3

SoC Elaboration Unit

Most of the material reported in this chapter is reused from [36] and [1] (©2018, 2019 IEEE), in agreement with IEEE copyright policy on theses and dissertations.

In this chapter is described the proposed elaboration unit architecture for smart power applications. The proposed digital system is an heterogeneous system, as introduced in [36] and highlighted in blue in Figure 2.4, which consists of the PULPino microcontroller [2] coupled with an embedded FPGA [3] [4]. The whole system-on-chip is fully-synthesizable which means that it is designed in synthesizable HDL code. The microcontroller is the master of the system and the eFPGA is interfaced through the Advanced Peripheral Bus (APB) of the Advanced Microcontroller Bus Architecture (AMBA) which is an open standard of digital system bus architecture. In this way, the embedded FPGA acts as a microcontroller peripheral since microprocessor is able to fully-manage the embedded FPGA writing and reading registers at the addresses defined in the RTL code. For instance, the microprocessor can program both the eFPGA clock frequency and the destination and the source of the embedded FPGA inputs/outputs. The idea is to have in the system a reconfigurable peripheral which can be programmed based on the application requirements, in order to handle the smart power tasks adopting only the eFPGA without any cooperation with the processor. Hence, microcontroller programs the eFPGA to manage the smart power application needs and then the processor can handle for instance high-level system communication, other computations or switches in sleep-mode to reduce the power consumption.

3.1 Microcontroller

The microcontroller used in the proposed reconfigurable SoC of Figure 2.4 is PULPino [2]. PULPino is an open-source ultra low-power microcontroller. Therefore, PULPino has a microprocessor unit and some standard peripherals such as general-purpose I/Os, timers, interrupt controller, serial communication interfaces. PULPino has a 32-bit single-core processor based on an implementation of the RISC-V instruction set architecture optimized for low-power and high-energy-efficient computing. Regarding processor implementation, it is possible to use:

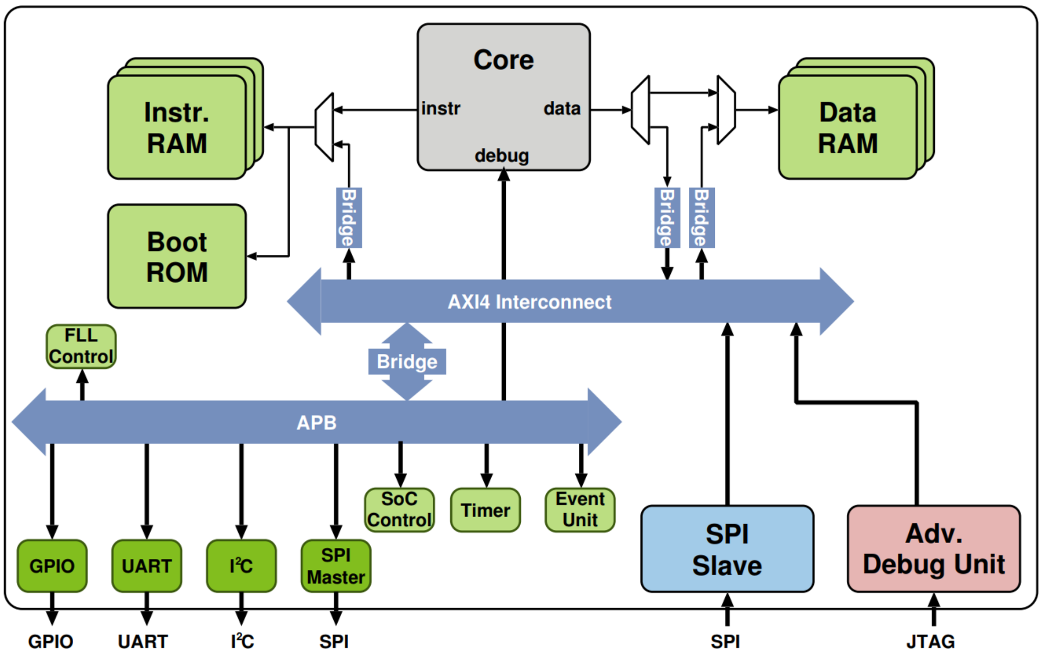


Figure 3.1: PULPino architecture scheme [38].

- 4-stage RI5CY pipeline
- 2-stage Zero-riscy pipeline
- 2-stage Micro-riscy pipeline without any hardware multiplier.

In this work, we used the 4-stage processor core because it is the one more similar to widespread commercial processor architecture such as ARM Cortex-M [37] and also because it allows the system to have good computing performance. As visible in Figure 3.1 the microcontroller is based on Harvard architecture since it has two different memory for both data (Data RAM) and instructions (Instr. RAM). Our realization of the proposed system has two 32-bit 4k-word static RAM, one for instructions and one for data memory. PULPino has two buses, one high-performance bus (Advanced eXtensible Interface AXI) for processor-memory communications and one at lower performance (APB) for processor-peripheral communications. The cited peripherals are connected to the APB bus, as reported in Figure 3.1.

3.2 Embedded FPGA Sub-System

Since eFPGA should be a microcontroller peripheral, it is connected to the APB with the other system peripherals. In addition to the actual eFPGA (PLD in Figure 3.2), there are also further digital blocks necessary to configure for instance the eFPGA clock frequency, eFPGA inputs, and outputs and program the eFPGA. Microcontroller interacts with the eFPGA system writing and reading registers located in the

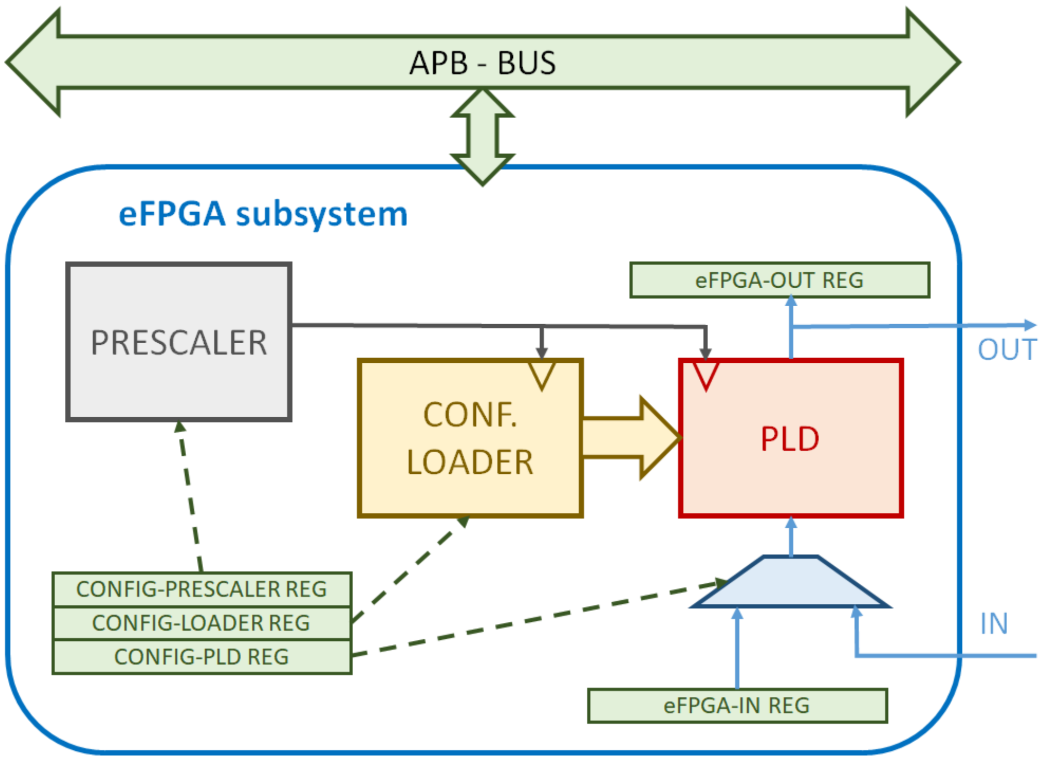


Figure 3.2: The eFPGA subsystem interfaced through APB bus. Picture reused from [1].

memory map. Hence, the eFPGA system, as reported in Figure 3.2 is provided of the actual eFPGA, a prescaler, a configuration loader, and some configuration registers.

3.2.1 Embedded FPGA Architecture

The embedded FPGA (PLD block of Fig. 3.2) is a soft-core Intellectual Property (IP), and hence fully-synthesizable in standard cell libraries, unlike the eFPGA available on the market (as shown in the previous chapter) which are designed using a hard-macro approach optimizing performance at transistor level. The eFPGA has a standard FPGA structure as presented in [3] and [4] which consists of an array of logic blocks connected via an interconnection network. As described in Figure 3.3, the proposed eFPGA has 16 Configurable Logic Blocks (CLBs) and the interconnection network is a Multi-Stage Switching Network (MSSN). The eFPGA has 64 inlets/outlets and the device dimension is about 100k equivalent gates. The eFPGA configuration memory is made of lathes replacing traditional SRAM cells (optimized at transistor level) to guarantee synthesizability. Each CLB has a structure reported in Figure 3.4, it has 12 inputs/outputs and 3 Basic Logic Elements (BLEs). The Basic Logic Element, as described in Chapter 4, can be configured as either 2$\times$LookUp Table (LUT) 4:2, 2$\times$LUT 5:1 or 1$\times$LUT 6:1, as well as either sequential or combinational circuit. As analyzed in the next chapter, the interconnection network is a Multi-Stage Switching Network with a butterfly-oriented topology, which provides

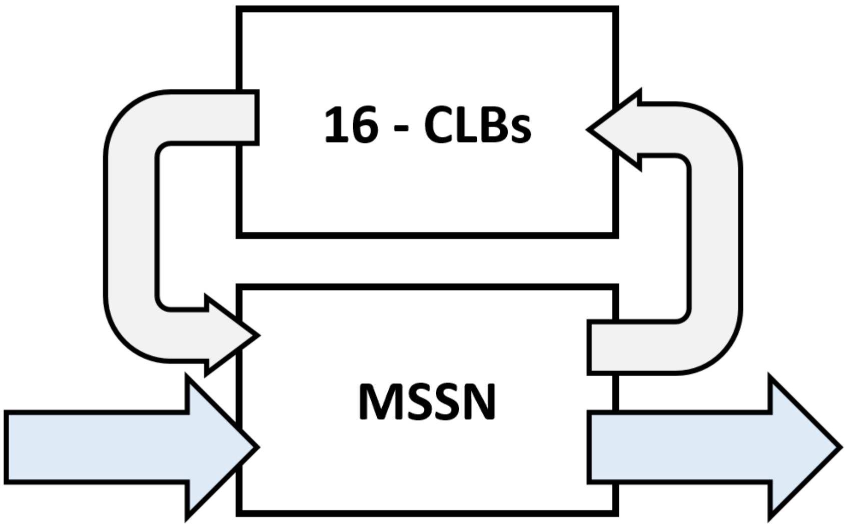


Figure 3.3: Diagram of the eFPGA. Picture reused from [1].

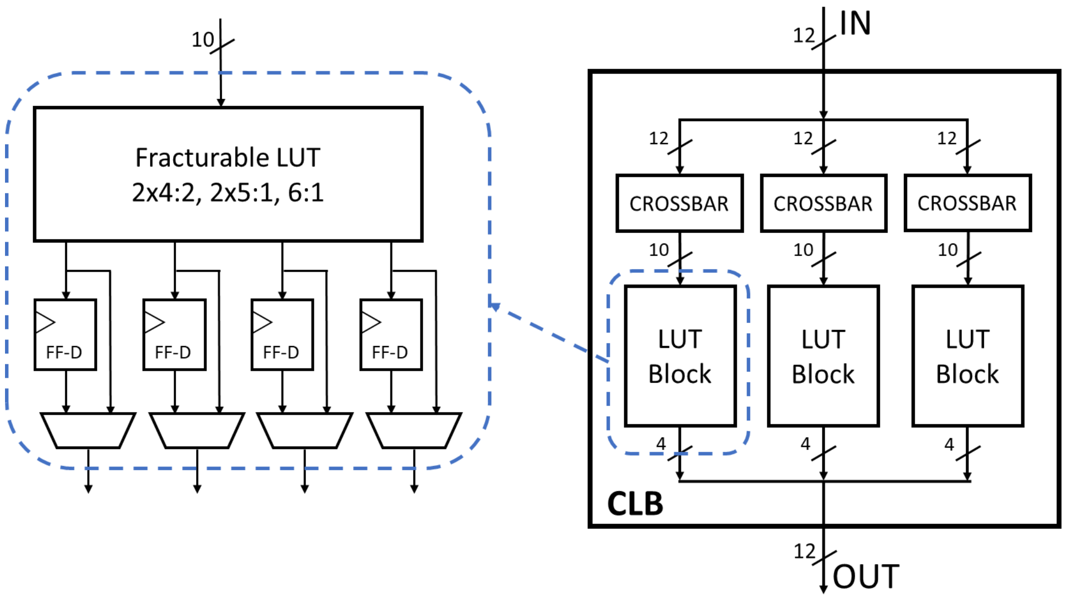


Figure 3.4: eFPGA CLB structure [3].

synthesizability and non-blocking routing features. The computational capability of the eFPGA is about 1k equivalent gates while the area occupation is about 100k equivalent gates. This under exploitation of the occupied area is obviously due to the reconfigurability. More details regarding the embedded FPGA architecture are provided in Chapter 4.

3.2.2 Prescaler

Prescaler (Figure 3.2) is an additional block that configures or switches off the eFPGA clock frequency. It takes the microcontroller clock line and divides it by a factor $n_{div} = 1\text{–}2^{16}$. It divides the system clock frequency for the eFPGA based on application reactivity needs, or it turns-off the eFPGA clock frequency when the eFPGA is not required, in order to reduce the dynamic power consumption. The processor fully-handles the prescaler configuration acting on its configuration register. The frequency division is simply performed programming the count of a digital counter.

3.2.3 Configuration Loader

The configuration loader configure the eFPGA CLBs and the interconnection network, putting the configuration bit-stream into the configuration memory of the eFPGA. Therefore, the configuration loader consists of a finite state machine that manages the PLD scan-chains which program the configuration memory. Since the eFPGA configuration phase is not both timing and power critical, because it occurs just one time at the system startup, it can be not too optimized. The easiest way is to serialize the bit-stream for the scan-chain, hence, the processor handles the programming phase, loading a byte of the configuration bit-stream in a configuration register. Then, activating a bit in the configuration register, the processor enables the configuration loader to put data into the scan-chain. Iterating the procedure the eFPGA is programmed.

3.2.4 Configuration Registers

Configuration registers are 32-bit registers used by the processor to both configure (writing) all the digital circuits of the eFPGA subsystem and check (reading) the status of the blocks or the eFPGA outputs. The configuration register addresses are organized in the address space designated to the peripherals. The configuration registers are:

- CONFIG-PRESCALER REG based on the application requirements in terms of responsiveness (latency) is configured by the processor. It programs the prescaler putting the value of the counter. Acting to an enable bit, the processor can switch off the PLD clock frequency if is not required.

Table 3.1: API Function Prototypes. Reused from [1].

Function Name	Description
reset_efpga()	resets the efpga
setup_efpga(efpga_addr, data_ptr)	manages configuration registers
set_in_efpga(bank_addr, value)	writes eFPGA inputs
read_out_efpga(bank_addr)	reads eFPGA outputs

- CONFIG-LOADER REG is used to program the configuration loader. The processor writes a byte of the configuration bit-stream in the register, and activating a bit flag notifies the configuration loader that the data are ready to be put in the configuration memory. Then, the configuration loader notifies the processor that the last byte it is correctly programmed and hence, the processor can load another byte of the configuration bit-stream until the final notification from the configuration loader. Since the eFPGA configuration procedure is a software procedure, it can be done at every time.
- CONFIG-PLD REG is used by the processor to both reset the eFPGA and configure the eFPGA inlets to be connected to either primary inputs or eFPGA-IN REG with 8-bit banks.
- eFPGA-OUT REG [1:0] are two registers containing the eFPGA outputs. The processor can read the register if it wants to either check or interact with the eFPGA activities.
- eFPGA-IN REG [1:0] are connected to an array of multiplexers, as shown in Figure 3.2, whose selectors are programmed in CONFIG-PLD REG. Depending on the CONFIG-PLD REG configuration, the eFPGA inputs can be connected either to primary inputs or to internal registers.

3.2.5 eFPGA Software Tools

A complete CAD flow for the proposed eFPGA was implemented as explained in [3] and [4]. The flow starts with an HDL code technology independent pre-synthesis in logic operators and flip-flop functionalities using Synopsys Design Compiler [39]. Then VTR provides the logic synthesis and LUT mapping and Versatile Place and Route (VPR) tool provides LUT packing and placement, while a custom tool is used to configure routing.

3.3 Application Programming Interface

In the proposed heterogeneous system, the task partitioning between eFPGA and microprocessor is arranged by the application designer, considering the eFPGA as a peripheral of the microcontroller. We developed the procedures for the PULPino code and they summarized in Table 3.1. These procedures use both read-memory and write-memory instructions at the corresponding eFPGA peripheral addresses. The reset_efpga procedure resets the eFPGA, which is necessary after the eFPGA configuration. The setup_efpga function can configure both the PLD (writing data into the CONFIG-PLD REG) and the prescaler (writing data into the CONFIG-PRESCALER REG) settings. The function iteration is used to program both the interconnection network and the CLB configuration memories writing in the CONFIG-LOADER REG. Using the set_in_efpga function microcontroller can manage the eFPGA inlets writing in the eFPGA-IN REG, while with the read_out_efpga function the processor can read the eFPGA outputs connected to the eFPGA-OUT REG.

Chapter 4

eFPGA Architecture Details

This chapter provides an architectural description of our embedded FPGA (designed by a previous Ph.D. student) already presented in [4] and [3]. A mathematical analysis of the eFPGA interconnection network properties is proposed in [40] and reported in this chapter and Appendix A.

4.1 Introduction

Field Programmable Gate Arrays (FPGAs) as embedded FPGAs are essentially arrays of some Computational Logic Blocks (CLBs) connected through some kind of interconnection network. The interconnection network plays an important role in the overall FPGA performances, in terms of area occupation and energy.

In Figure 4.1 are reported two typically field programmable gate array structures. In the island-style FPGAs (Figure 4.1(a)) the CLBs are distributed in a two-dimensional array, while in hierarchical FPGAs the CLBs are organized in different clusters as reported in Figure 4.1(b). In the island-style architecture the available interconnections are a notable diversity, and hence, usually used in commercial FPGAs. On the other hand, hierarchical FPGAs offer a very efficient connection between CLBs of the same cluster, but not if it is required to cross a considerable number of main connections between different clusters.

The red dots in Figure 4.1 are the switches that can connect the interconnection network wires, typically realized combining pass-transistors, multiplexers and tri-state buffers in order to create either unidirectional or bidirectional wires as depicted in Figure 4.2. Figure 4.2(a) reports a bidirectional switch composed of two back-to-back tri-state buffers. Since bidirectional switches require pass-transistors or tri-state circuits, they have been replaced by unidirectional switches, as for instance the one shown in Figure 4.2(b) [4].

CLBs are the computational elements of the FPGA and they can have various kinds of architectures, including different types of both storage and computational elements. Based on the design constraints, they can be implemented using different kinds of circuits such as standard logic gates, multiplexers, LookUp Tables (LUTs), pass-transistors, etc. Figure 4.3(a) depicts an example of a CLB architecture composed of various sub-blocks called Basic Logic Elements (BLEs) with I inlets and N

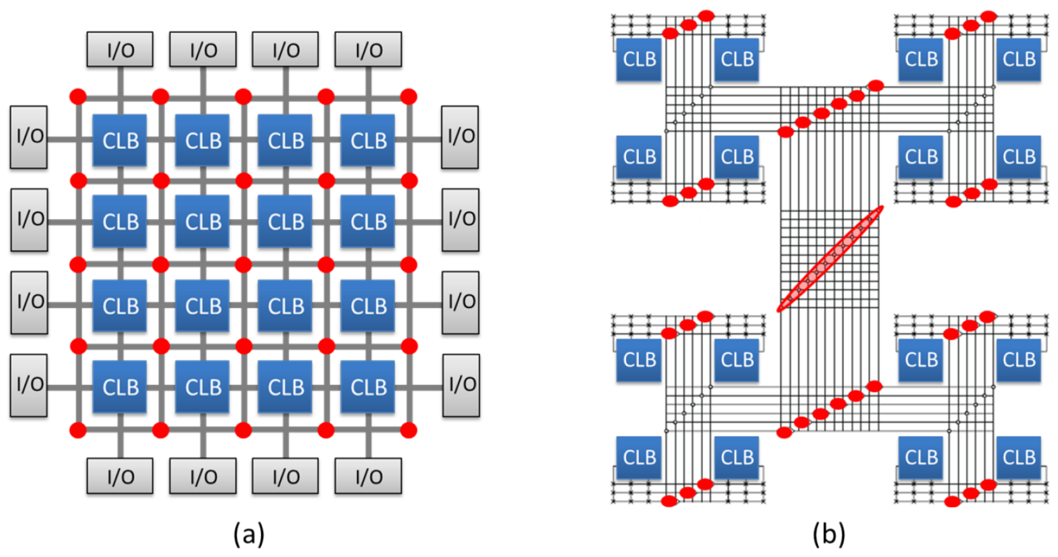


Figure 4.1: FPGA structures: island-style (a) and hierarchical (b) [4].

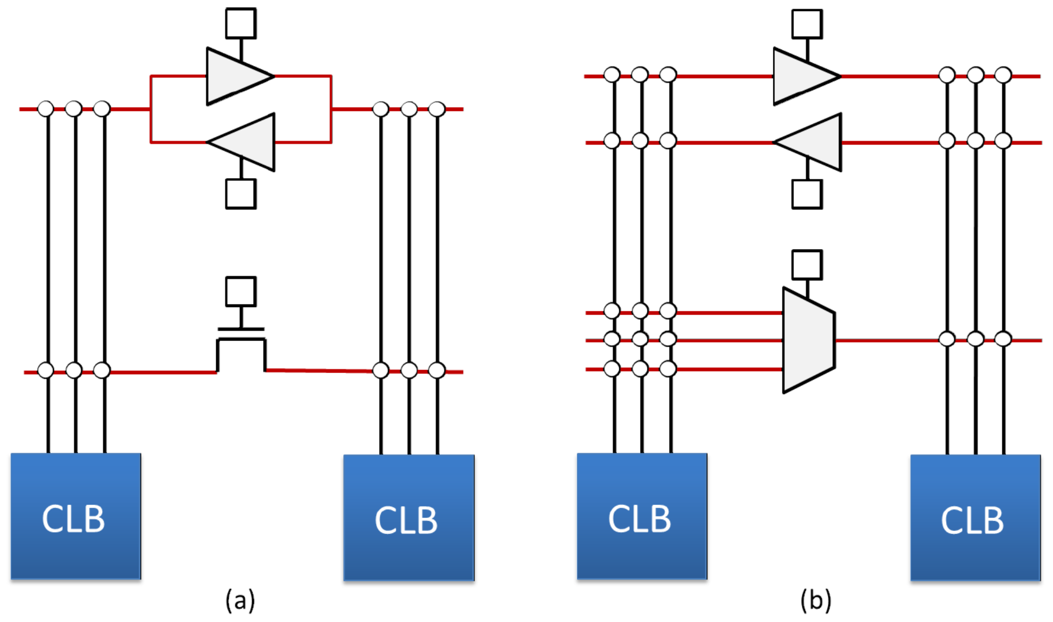


Figure 4.2: Bidirectional (a) and unidirectional (b) wires [4].

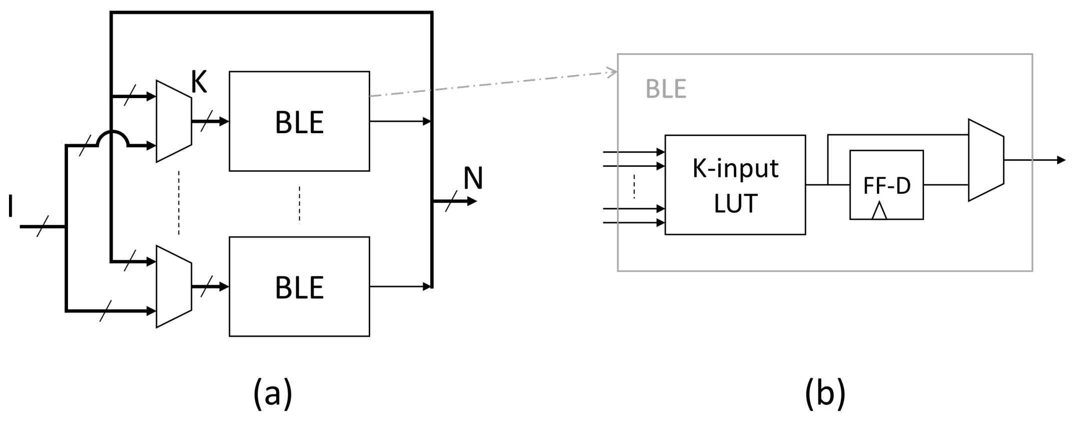


Figure 4.3: Example of both CLB architecture (a) and BLE (b).

outlets. Each BLE has K inputs directly connected to a lookup table as reported in Figure 4.3(b). In addition, a BLE has a Flip-Flop D to generate a sequential signal and a multiplexer that can be programmed to select either sequential or combinational path. In order to increase the versatility of the computational blocks, in some cases is possible to find specific blocks such as adders, multipliers, Digital Signal Processor (DSP) units.

As already said, FPGAs are arrays of CLBs connected using somehow programmable routing [41]. The overall field programmable gate array efficiency is strongly impacted by routing since it deeply affects area occupation and performances and is responsible for the efficiency of the interconnection [42], [43]. Due to the constant complexity application growth, researches try to mitigate the penalties due to both routing congestion issues and bit-level programmability.

In order to address our aim of portability to different technology, the only possible way is to design the eFPGA using standard logic gates from standard cell libraries.

As introduced in Chapter 3, our embedded FPGA is a small eFPGA with a computational capability of around 1 k equivalent gates and it is fully synthesizable. It has 16 CLBs and its architecture is based on a hierarchical organization of Figure 4.1(b). As presented in Chapter 3, each configurable logic block has a structure depicted in Figure 3.4. Each crossbar internal to CLB consists of an array of ten 12:1 multiplexers as shown in Figure 4.4(a). In Figure 4.4(b) is reported the structure of each 12:1 multiplexer of the crossbar. Each Basic Logic Element of the CLB has two lookup tables that can be configured as a LUT 6:1 or two fractured LUTs as depicted in Figure 4.5. It is still present the possibility to generate either sequential or combinational signal acting on the specific multiplexer selector of Figure 4.5.

As already mentioned, the interconnection network is based on a Multi-Stage Switching Network (MSSN), and in the following section, its proprieties are described.

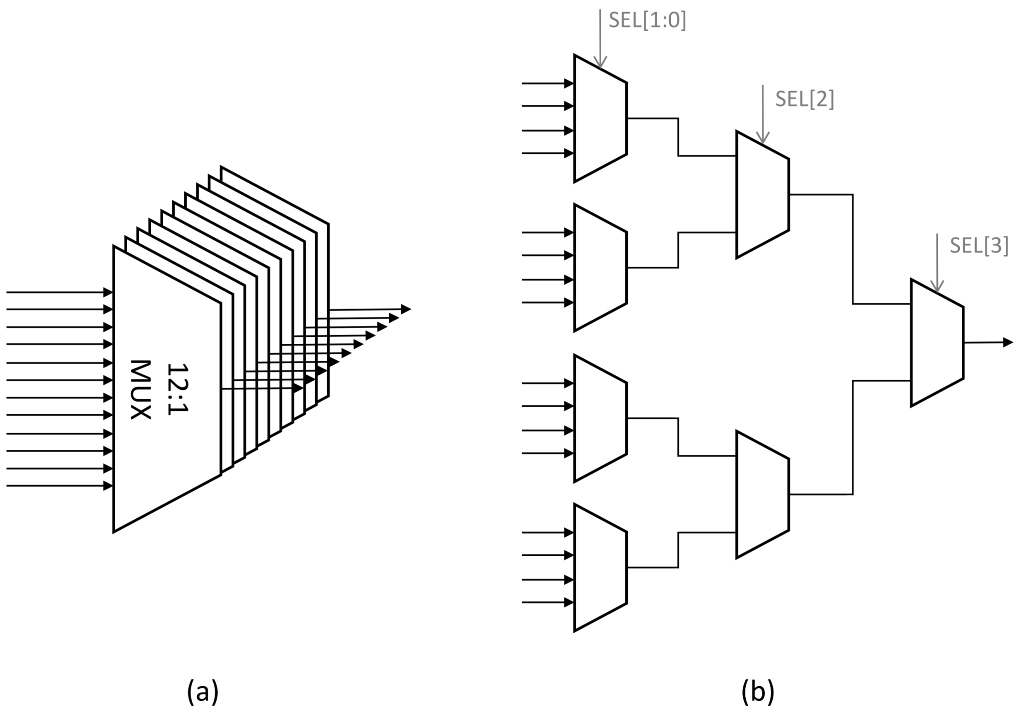
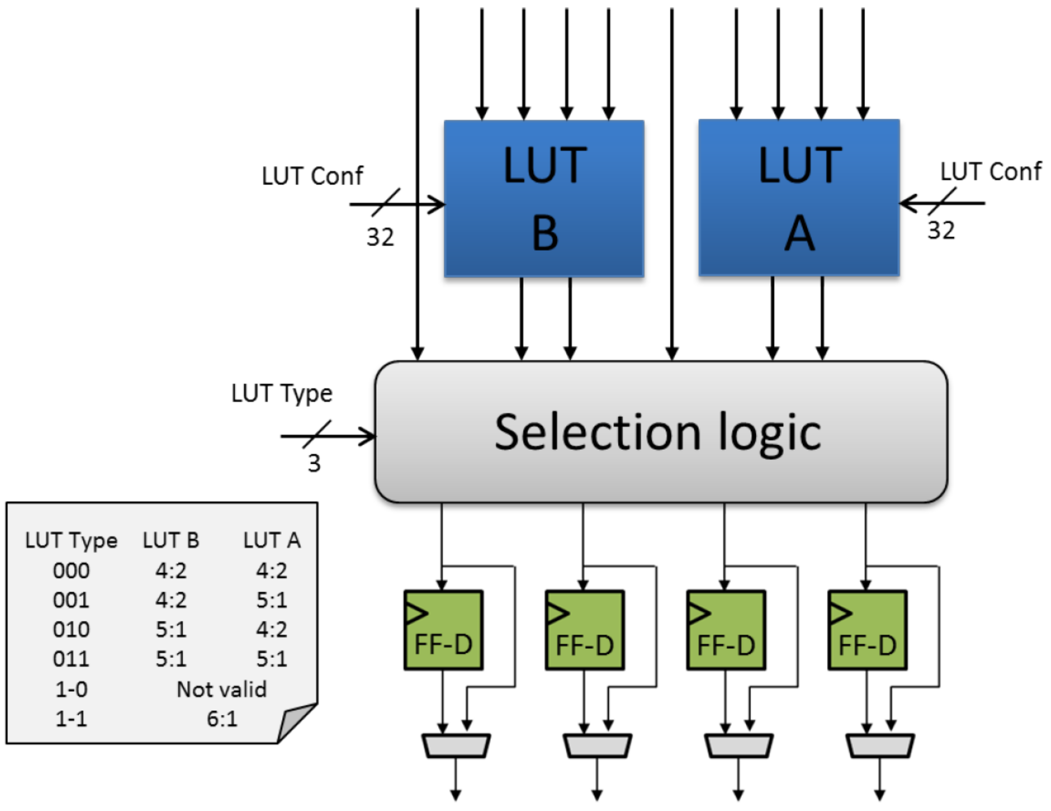


Figure 4.4: CLB internal Crossbar architecture (a). Multiplexer 12:1 structure (b).



LUT Type	LUT B	LUT A
000	4:2	4:2
001	4:2	5:1
010	5:1	4:2
011	5:1	5:1
1-0	Not valid	
1-1	6:1	

Figure 4.5: BLE architecture [4].

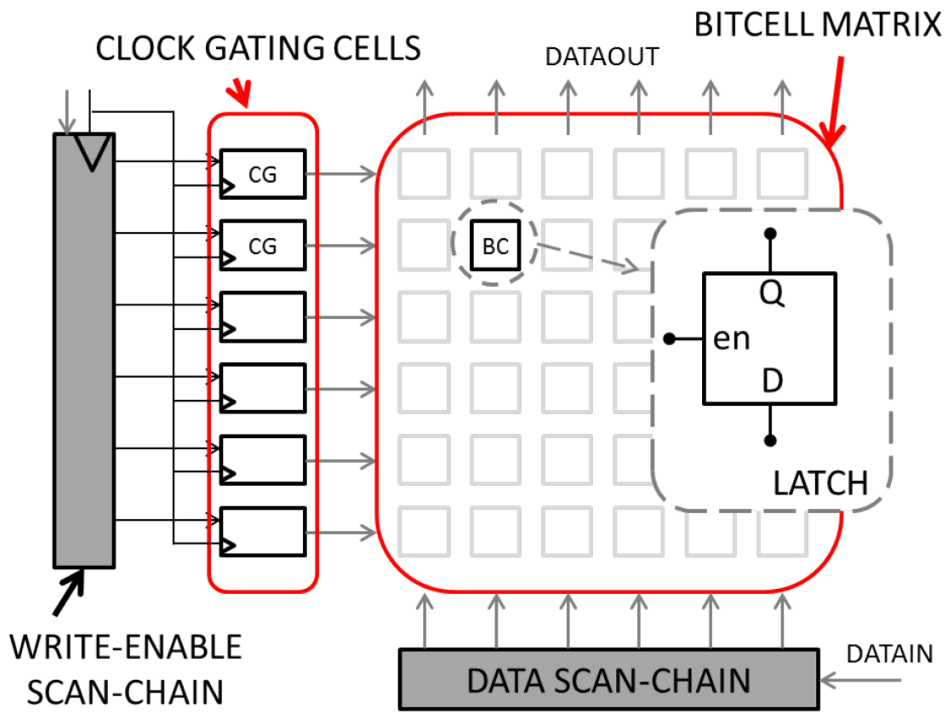


Figure 4.6: BLE architecture [4].

In order to preserve the fully-synthesizability of the embedded FPGA, the configuration memory is implemented through standard latches instead of RAM. In this way, the eFPGA block portability to different technology is guaranteed. For each CLB and MSSN level there is a configuration memory organized in a cluster of latches. The configuration memory is configured through two scan-chains as shown in Figure 4.6. One scan-chain is used to put data into latch memory, while the second one is used to select the row that should be programmed as a row write enable. Each Bit Cell (BC) of Figure 4.6 is driven to the enable signal by a clock-gating (CG) cell enabled by a configuration clock, which is substantially asynchronous with respect to the functional clock.

4.2 Embedded FPGA Interconnection Network

In order to both decrease complexity issues and enable a programmable routing area overhead reduction, hierarchical interconnection networks based on local crossbars and exploiting Rent's rules were studied [44]–[46]. Therefore, the eFPGA interconnection network that connects inputs, outputs and CLBs is a Multi-Stage Switching Network (MSSN). This network is a hierarchical interconnection network built combining many small crossbars, usually called switch elements (SE) (Figure 4.7). For instance, in Figure 4.7 is depicted a $N \times N$ Banyan MSSN with $N = 16$ inlets and outlets exploiting 2×2 switch elements, where it is possible to identify different sub-networks.

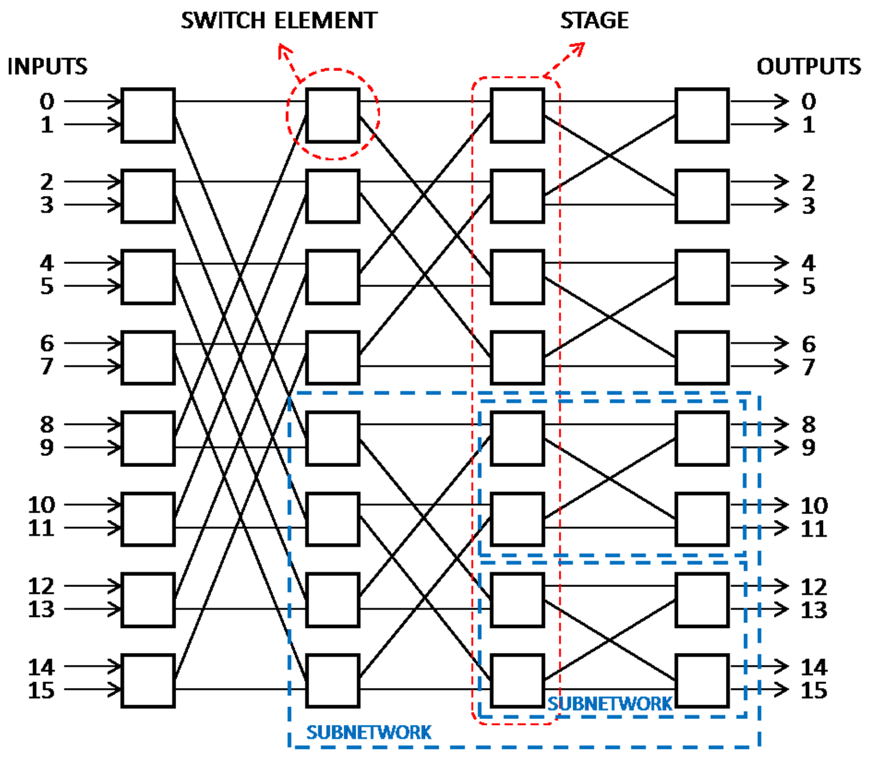


Figure 4.7: An $N \times N$ (with $N = 16$) Banyan multistage switching network (MSSN) featuring butterfly topology. Picture reused from [40].

A Multi-Stage Switching Network, depending on its topology and/or its ability to perform connection between inlets and outlets can be classified as:

- *non-blocking*, if the MSSN can connect each I/O pair regardless of the existing connections on the network.
- *Blocking*, if it cannot connect all the requests.

Besides, a non-blocking network can be further classified as:

- *strictly non-blocking* (SNB) if any connection can be set up incrementally without the need to rearrange (i.e., reroute) any of the connections already in place.
- *Rearrangeable non-blocking* (RNB) if one or more existing connections may have to be rearranged to permit a new required connection between one input and one or more outputs.

All the previous definition can be used for different kinds of traffic:

- *unicast* where a network input needs to be connected only to an output. For that reason is typically called one-to-one.
- *Multicast* connections are connections where a input is connected to many outputs (one-to-many).

- *Broadcast* connections require to connect an interconnect input to all the network outlets (one-to-all).

An eFPGA Multi-Stage Switching Network has to support multicast connectivity addressing rearrangeable non-blocking feature, since the device has static connections and hence, RNB characteristic is equivalent to strictly non-blocking. The required RNB property allows the interconnection network to reduce congestion using fewer resources. The usage of a MSSN as an eFPGA interconnection network has straightforward benefits in terms of:

1. network modularity which fits well a soft-core approach, since each switching element can be either implemented by standard digital circuits available in standard cell libraries or optimized at circuit level as a single coarse-grained cell, without affecting synthesizability.
2. Interconnection network routability properties, and hence, eFPGA design flexibility. A hierarchical interconnect allows one to customize the eFPGA size and the number of inlets/outlets following the same congestion analysis.
3. The routability analysis is simplified, since the blocking properties of an MSSN are well-defined and predictable in terms of topology, as discussed in Appendix A.1 [40].

Examples of hierarchical interconnection networks in eFPGA applications are M2000 (Abound Logic) which proposes an MSSN with local crossbars based on a Clos network [47], Leopard Logic proposed a butterfly-based hierarchical network [48]. In [43], [49] is discussed an MSSN based on butterfly topology with depopulation of the upper stages of the network and an isomorphic transformation to solve the *radix-boundary* problem [50], which is a limiting factor of MSSN exploitation in FPGAs. Nevertheless, area saving is balanced by the fact that this network is no more proven to be RNB, although authors indicate the availability of enough bandwidth based on Rent's rule. Another example of the application of Rent's rule to sizing a multi-level interconnection network is provided in [51], which overcomes the boundary-radix problem by adding shortcuts and staggering.

RNB multicast network theme is a quite extended topic [52], whereas for eFPGA application is restricted to architectures that provide a small area overhead for a number of inputs/outputs N that can reach some thousands of units. Among the variety of MSSNs used in telecommunication area [52]–[55], for embedded FPGA the starting point is the Banyan network [56], and its topological equivalent versions [56], [57] such as *Omega*, *Shuffle*, *Butterfly*, since they are low-latency multistage networks. Figure 4.7 shows an $N \times N$ Banyan network (with $N = 16$) built combining 2×2 switch elements (called *radix-2*) [58]. Due to its structure, a Banyan network is also called $\log_2 N$-network [59]. As visible in Figure 4.7, thanks to its hierarchical structure, the Banyan network can be recursively decomposed into sub-networks, besides its low-latency feature makes Banyan networks attractive for embedded FPGA. On the other hand, the main drawback of Banyan networks is that they are blocking also for unicast traffic. To improve the Banyan network blocking feature, additional resources to the baseline structure are mandatory (as explained in Appendix A.1), resulting in a network called *Multi*-$\log_2 N$ networks [59].

4.2.1 Proposed eFPGA Interconnection Network

The interconnection network architecture proposed in both [3] and [4], and intensively analyzed in [40] is a Multi-Stage Switching Network based on 2×2 switches, and it is shown in Figure 4.8(a) with 8 I/Os. The proposed network has a Benes-like topology with a butterfly connectivity. A topologically equivalent network [60], which means a network with the same graph, is depicted in Figure 4.8(b). The proposed interconnection network architecture has:

- $N_S = N$ switch elements per stage, where stage states as the number of columns composed by switches.
- $n_S = 2\log_2(N) + 1$ stages due to:
  - 1 input stage (IN$_{\text{stage}}$) that consists of N_S demultiplexers or 1×2 switches.
  - 1 output stage (OUT$_{\text{stage}}$) consisting of N_S multiplexers or 2×1 switches.
  - $2\log_2(N) - 1$ middle stages (MID$_{\text{stage}}$) with N_S 2×2 switches each. The middle stages are composed of:
    * n stages which are due to a $N \times N$ baseline $\log_2 N$ network, and hence, $n = \log_2 N$.
    * x extra stages to improve the blocking capability of the Banyan network (as shown in Appendix A.1) defined as $x = \log_2 N - 1$.

Hence, $n_s = 1_{INstage} + 1_{OUTstage} + (2\log_2(N) - 1)_{MIDstage}$.

The resulting Multi-Stage Switching Network has rearrangeable non-blocking feature has reported in Appendix A.1 and resulting from the intensive eFPGA testing. The proposed MSSN, as better visible in Figure 4.8(b), consists of two sub-network called planes (p_1 and p_2) connected to both inputs and outputs through both demultiplexing and multiplexing stages. Each sub-network or plane consists of a $N \times N$ Banyan network with x extra stages.

Considering our embedded FPGA composed of 16 CLBs with 12 inputs each, and targeting 64 primary eFPGA inputs/outputs, the necessary interconnection network has 256 I/Os, 17 total stages of 256 switch elements each.

The proposed MSSN architecture can be also generalized to a generic *radix-k*, using switch elements with a radix different to 2. If it is required by the application, one can design the interconnection network with a higher radix, like 4 for instance,

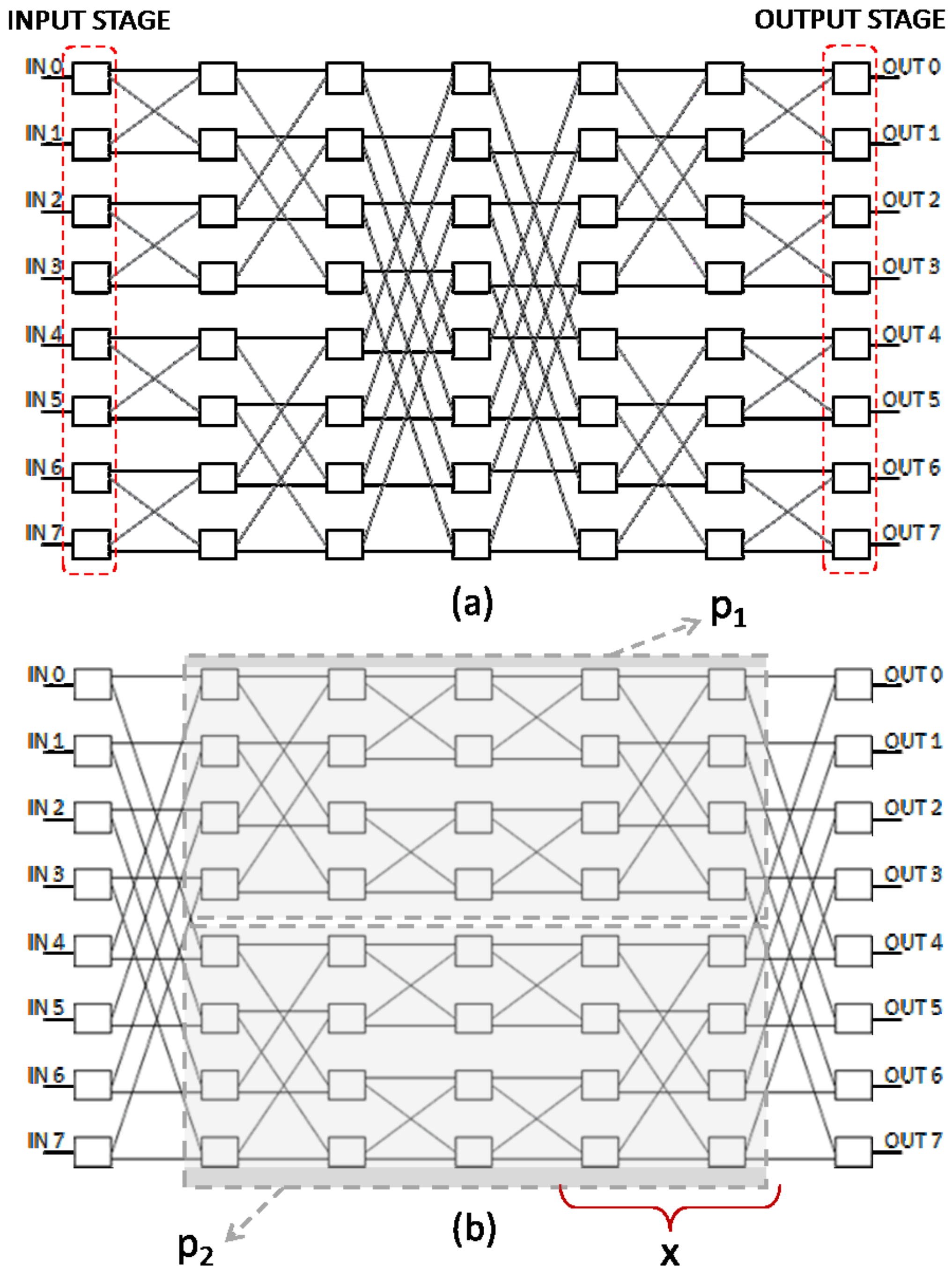


Figure 4.8: An $N \times N$ (with $N = 8$) *flat* version of the proposed MSSN: original architecture (a) and rearrangeably non-blocking (RNB)-proof unicast topologically equivalent version (b). Picture reused from [40].

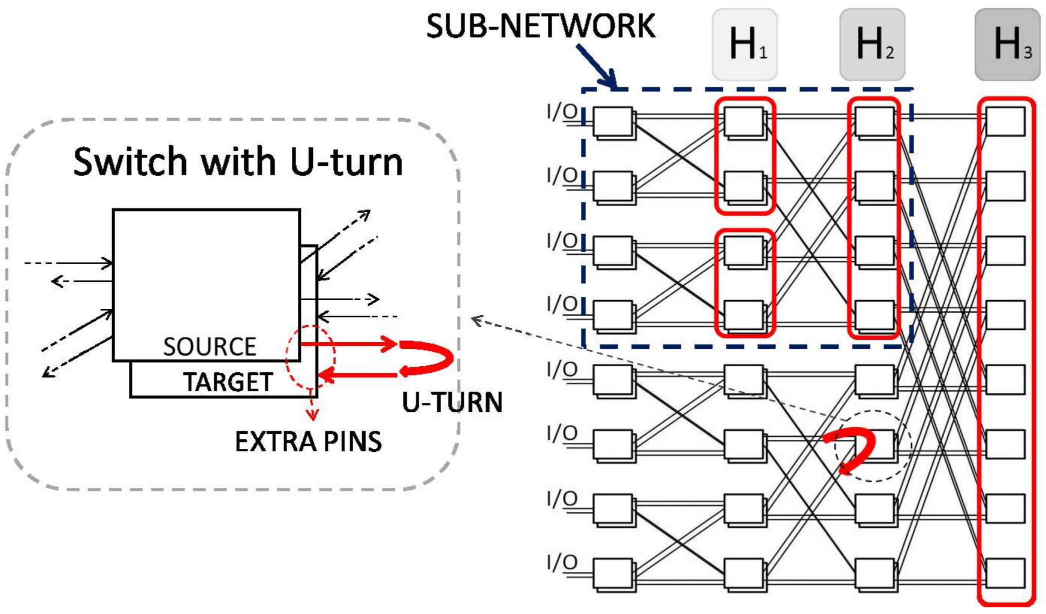


Figure 4.9: Folded bypassed MSSN enhanced with *U-turn* bypasses [3].

preserving the rearrangeable non-blocking proprieties, thanks to the network hierarchical structure. In Appendix A.2 are extended the consideration done for the *radix-2* MSSN to a generic *radix-k* MSSN.

MSSN with Bypass Enhancement

Thanks to the hierarchical interconnect structure, one can also improve the switch element features. The addition of x extra stages in the interconnection network to obtain rearrangeable non-blocking proprieties, impacts on the overall network latency. In addition, butterfly topology interconnects are not able to exploit local traffic. Therefore, an alternative hierarchy-aware folded MSSN version is introduced. Exploiting the network symmetry of the *flat* view of Figure 4.8(a), it is possible to fold the network at the central stage, creating a *bypassed* architecture shown in Figure 4.9 [3]. In this case, the addition of dedicated switches provided with a *U-turn* connection, allows the network to connect *U-switches* of the H_S group at the stage S. The *U-turn* exploitation allows the network to be divided into butterfly-based sub-networks, potentially allowing the upper-levels of the hierarchy to be bypassed. This enhancement permits the network to better exploit local connectivity without impacting on congestion. As an example, in Figure 4.9 the sub-network obtained grouping the H_2 upper switches allows any connections between I/O from 0 to 3 to be performed bypassing the upper level H_3 of switches [40].

The bypassed version of the interconnection network inherits the rearrangeable non-blocking properties from the flat interconnect architecture. Exploiting *U-turn*

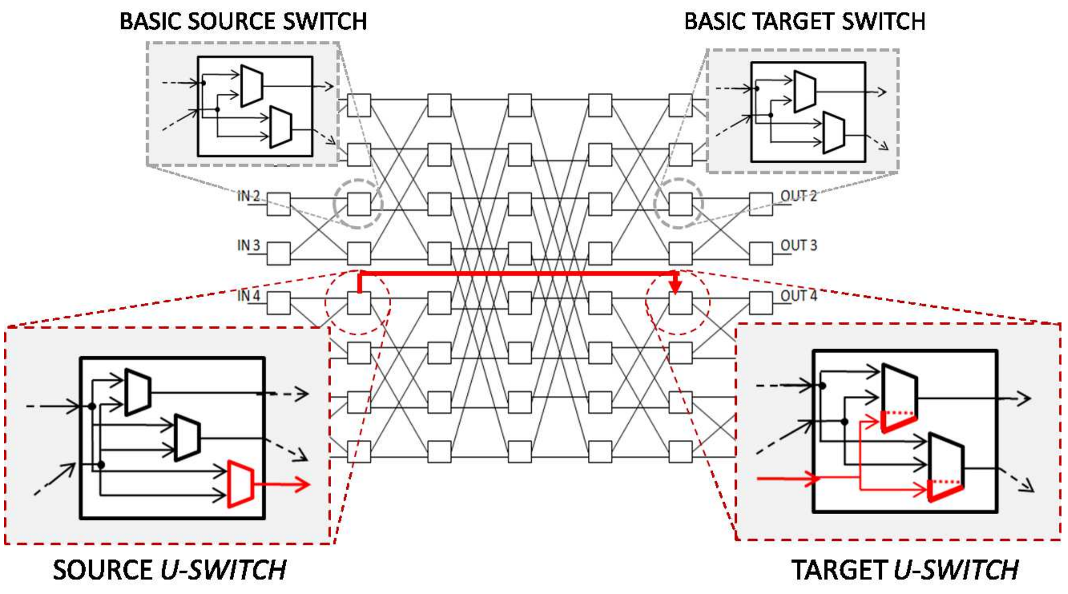


Figure 4.10: MSSN with bypass enhancement: unfolded view and *U-switch* structure description [4].

bypasses is possible to obtain more routing paths and available connections, inheriting the non-blocking features of the flat version to each *bypassed* sub-network (Figure 4.9). In this way, in an unfolded view of Figure 4.10, non-adjacent stages are bridged thanks to the addition of dedicated logic, and hence, additional resources.

The bypass exploitation results in a straightforward area overhead due to the additional circuits, interconnections and configuration bits. However, the area penalty is balanced by an improvement in performances, as analyzed in Appendix A.3.

4.2.2 MSSN Programming Strategy

The embedded FPGA software tool provides both the Register-Transfer-Level (RTL) description of the interconnection network and the network configuration bitstream as presented in [4]. One can describe the desired architecture specifying interconnect parameters, such as:

- number N of I/Os;
- number n_s of stages;
- switch element architecture i.e. *radix*
- enhanced stages with the bypass as in Figure 4.10.

Following [3], the MSSN routing engine is an iteration-based path-finding algorithm and the routing process split multi-fanout tracks into independently single-fanout nets sharing switch elements. After a net organization, the routing is sequential, and it takes into account the available resources, and thus, the available switch elements. The net sorting is based on their critical issue, which means that is listed the history of which nets have difficulty to be routed, and hence, the routing iteration success or fail, updating the list after each routing iteration. The process stops when all the tracks are successfully routed.

The path-finding strategy is based on Dijkstra's algorithm for weighted graphs [61]. The switch elements represent the graph nodes connected through weighted edges, which correspond to the wires used to connect interconnection network stages. The algorithm aims to find the cheapest path among the overall available ones, and the path cost is the sum of the weights of the traversed wires. In Figure 4.11 are reported three different ways to connect $IN2$ to $OUT0$, $OUT1$, and $OUT2$ sorted in order of cost. The considered multi-fanout connection can be implemented in the cheapest way of Figure 4.11(a) exploiting bypass, in *as-straight-as-possible* policy Figure 4.11(b) and using diagonal wires Figure 4.11(c). Clearly, the first two cases allow the algorithm to improve its convergence, and an *as-straight-as-possible* policy tends to increase the utilization of the bypasses. Therefore, both bypasses and straight links have lower weights in order to drive the router to maximize bypass and straight path usage. Thanks to this approach, we found that, on networks with dimensions up to 4K I/Os, the number of iterations needed to solve the routing lies on average between 5 and 10 [40].

The butterfly-based interconnection network usage changes the distance metric used on the placement algorithms. Traditional placement algorithms for both FPGAs and Application-Specific Integrated Circuits (ASICs) address the distance minimization between used logic blocks or standard cells, where the distance is measured in a 2D space (be it the L1 Manhattan distance or L2 distance) strictly related to the Elmore delay associated to the wire. On the other hand, in a butterfly-based interconnection network enhanced through bypasses, the delay is related to the number of stages to be crossed to connect an input to an output. Exploiting bypasses, as shown in Figure 4.11, is possible to reduce the number of crossed stages (path in Figure 4.11(a) traverses less stage than the path in both Figure 4.11(b) and Figure 4.11(c)), and hence, the delay. However, as shown in Figure 4.9, the distance between network I/Os is not linear but follows a step trend related to the network *radix*. Referencing to Figure 4.9, to connect the third input to the fourth output must be crossed only switches up to H_1, whereas to connect the fourth input to the fifth output is required to cross switches up to H_3. This topic is called the *boundary-radix* problem [43] and it is the main drawback of this kind of network usage in large dimension FPGA (eFPGA). In our case, of a small embedded FPGA, this problem is alleviated considering a wire length defined for a generic *radix-k* as:

$$\text{wire length} = \begin{cases} 2 \cdot S_{min}, & \text{for} \quad S_{min} = \log_k N \\ 2 \cdot S_{min} + 1, & \text{otherwise} \end{cases} \tag{4.1}$$

instead of a 2D distance metric. In equation (4.1) S_{min} (included from 1 to $\log_k N$) is the minimum stage S subtending all the points that must be connected as introduced

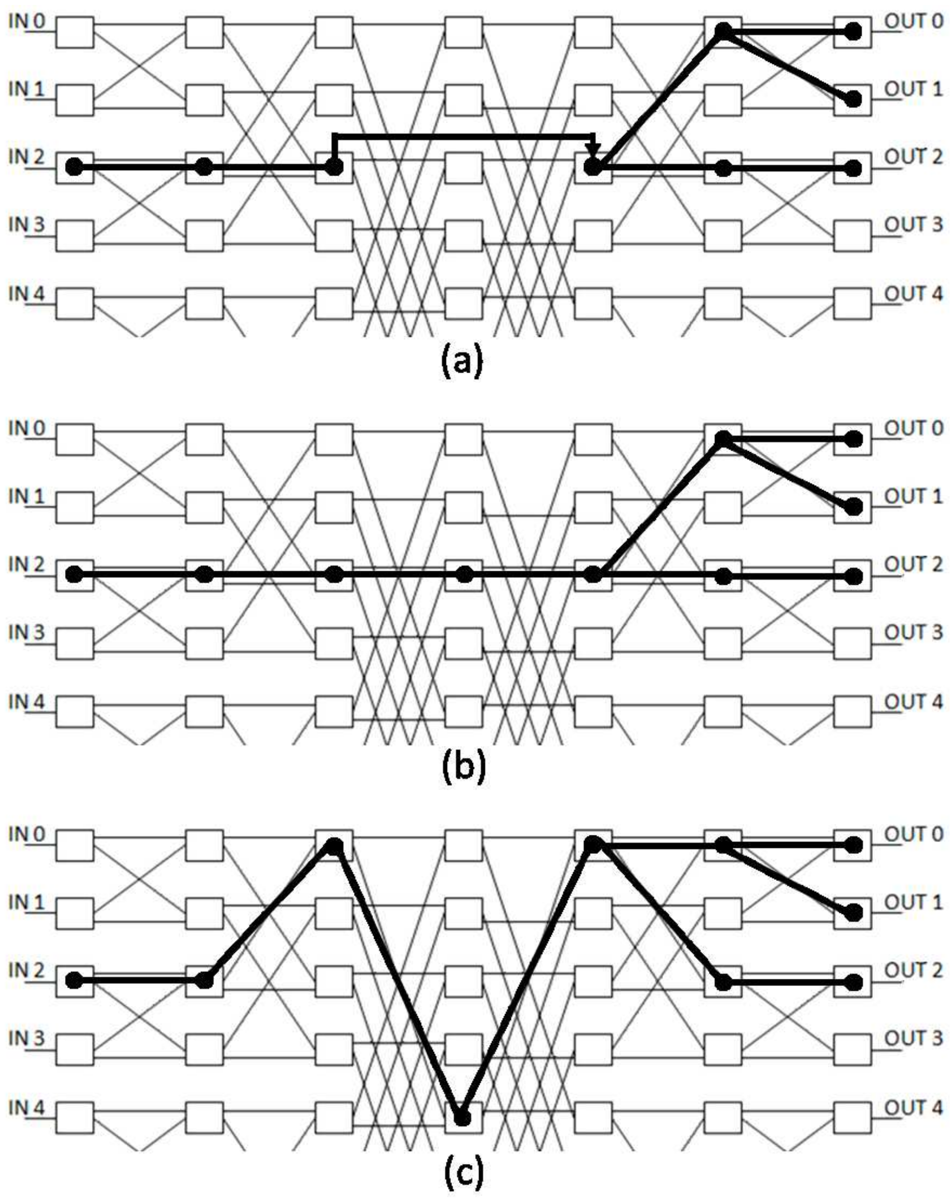


Figure 4.11: Three different multi-fanout paths connecting $IN2$ to $OUT0$, $OUT1$, and $OUT2$ in order of cost: with bypass enhancement (**a**), with the as-straight-as-possible policy (**b**), and using diagonal wires (**c**). Picture reused from [40].

in [3]. The "+1" term is due to the bypass connection. Therefore, the placement tool tries to limit the number of switches to be crossed to S_{min} in order to achieve the best enhanced hierarchical structure exploitation, and hence, to improve the eFPGA performance.

Chapter 5

Application Analysis

Most of the material reported in this chapter is reused from [1] (©2019 IEEE), in agreement with IEEE copyright policy on theses and dissertations.

This chapter provides an energy comparison of different implementations of the elaboration unit. Some considerations regarding the latency performance of the different solutions are also presented. The considered elaboration units are the eFPGA, PULPino and ASIC implemented in 90 nm BCD technology of STMicroelectronics and an STM32 microcontroller. Through an energy evaluation, are evaluated the energy performance of the different solutions to carry out different kinds of application. Hence, the goal of this chapter is to provide a proof in terms of performance of the validity of the proposed SoC (composed of both microcontroller and the eFPGA) in the smart power arena.

5.1 Implementation Choices

We present the different implementation units considered to evaluate both energy and latency performances. The first presented device is the proposed system-on-chip, then a general ASIC implementation and in the end a microcontroller of the STM32 family.

5.1.1 Proposed Reconfigurable SoC

The first considered elaboration unit is the proposed reconfigurable SoC (introduced in Chapter 3) composed of PULPino microcontroller and the eFPGA. This system has been implemented in 130 nm BCD technology of STMicroelectronics [36] and then presented in 90 nm BCD technology of STMicroelectronics in [1]. This technology is provided of 5 metal layers for the routing and a full-set of standard cell libraries with logic gates realized through transistors with two different channel lengths (*multi-L* technique). Figure 5.1 shows the whole digital reconfigurable system-on-chip floorplan and both the soft-core eFPGA sub-system and PULPino are highlighted targeting an implementation frequency of 50 MHz. In the floorplan, one can see two static RAM of 16 kB each for data and instructions (two rectangles in the bottom side of

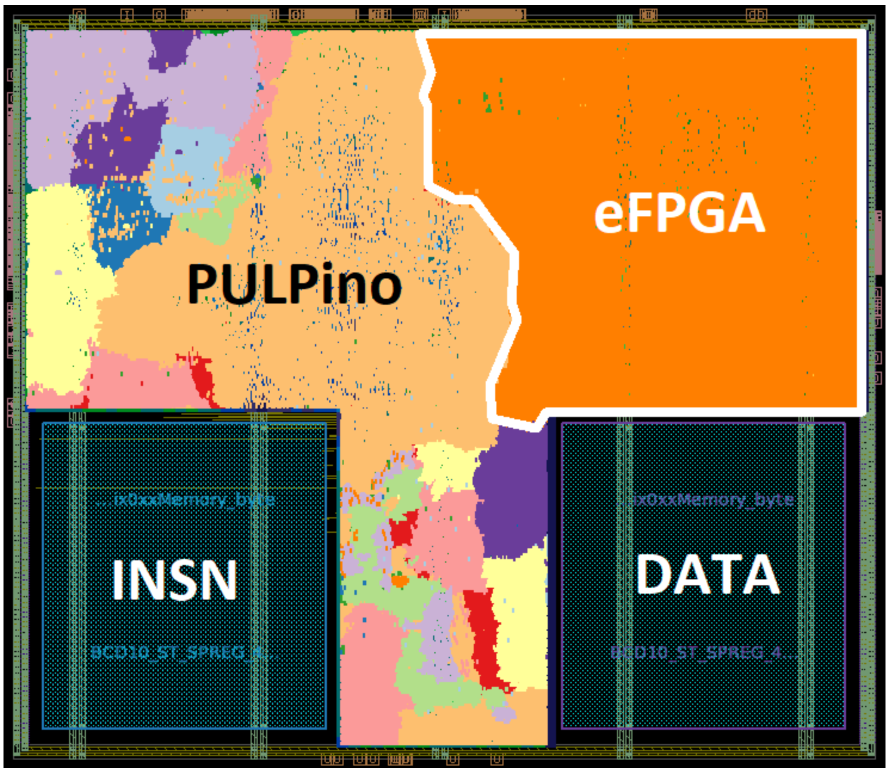


Figure 5.1: Proposed SoC implemented in BCD 90 nm. Reused from [1].

the picture). The bigger area of the PULPino system is the PULPino CPU which is the 4-stage RI5Y pipeline core, while the other smaller areas are the various micro-controller peripherals and bridge. The overall area occupation of the system-on-chip is 1.78 mm^2 with about 75 % of row utilization and 1.3 mm^2 without taking into account the memory area. The area overhead due to the eFPGA is about 20 % of the overall area and around 27 % without considering the memory area. The implementation results are summarized in Table 5.1 where the values in brackets do not consider the memory area. Figure 5.1 depicts the soft-core approach of the whole system-on-chip (except the two static RAMs which are hard-macros) and hence, the fully-synthesizability of the system. In the remainder of the chapter, all the evaluations for the proposed SoC are based on this physical implementation.

Table 5.1: Proposed SoC implementation Results. Reused from [1].

	Implementation
Technology	BCD 90 nm
Frequency	50 MHz
System Area	1.78 (1.3) mm^2
eFPGA Area	0.347 mm^2
eFPGA Area %	20 (27) %

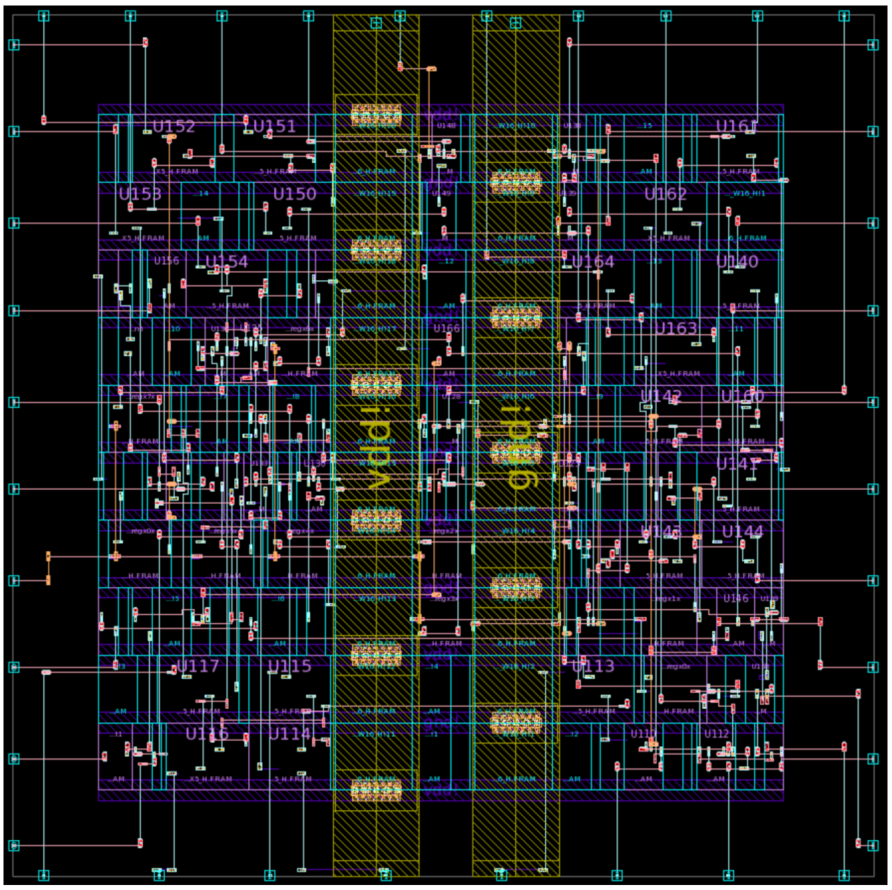


Figure 5.2: RGB LED controller ASIC layout.

5.1.2 ASIC

The ASIC implementation of the elaboration unit is obviously connected to the specific application. The considered applications are explained in the following paragraphs. For each application, starting from the HDL code, we have implemented the circuit in 90 nm BCD technology of STMicroelectronics, using a design flow based on standard cell libraries, and hence, the same as the previous case. The ASIC estimation area can be also expressed in equivalent gates, which is the ratio between the effective area occupation and the area of four transistors (typically a NAND gate). Figure 5.2 reports an example of an ASIC implementation in 90 nm BCD technology. In this example, the ASIC is a controller for RGB LED and it is better analyzed in the following.

5.1.3 STM32 Microcontroller

STM32 is a family of microcontrollers produced by STMicroelectronics. These microcontrollers are provided of many different ARM processors based on the application field of the microcontroller. In this work, we use an STM32L152RE which is equipped with an ARM Cortex-M3 [14]. The choice of this microcontroller was driven to find a commercial solution as much as possible similar to the PULPino microcontroller. Indeed, the ARM Cortex-M3 is a 32-bit RISC core with a 3-stage pipeline targeting embedded system applications [37]. The STM32 is also provided with numerous peripherals for interaction with the physical world. Microcontrollers can be configured to optimize either computing performance or reduce power consumption. For instance, one can reduce the operating clock frequency to reduce the dynamic power consumption, reduce the supply voltage to decrease power consumption and use different kinds of sleep mode. The STM32L1 microcontrollers are realized in 130 nm ultra low-leakage process [62], in order to address low-power consumption.

5.2 Energy and Latency Model

In this section we analyze the methodology used to a fair comparison between the different elaboration unit solutions. Hence, the way to figure out how different given circuits work is looking for the used energy to perform a specific task and in how much time. Thus, it is first evaluated the energy efficiency of the proposed eFPGA, PULPino, STM32 and, ASIC solutions, in carrying out various kinds of application and resulting also in a latency analysis. ASIC solutions implemented in 90 nm BCD technology give an idea of the best achievable performances that this technology can reach for these applications and hence, how the programmability deteriorates them. Since STM32 is realized in a different technology, its energy performance must be adapted to our reference technology, BCD at 90 nm.

5.2.1 eFPGA - Efficiency Model

The energy performance of the proposed eFPGA is based on the estimation of the average power consumption P_{eFPGA}. This estimation is performed on post-physical synthesis with a parasitic back-annotation, using Synopsys PrimeTime-PX, while the physical synthesis is performed in 90 nm BCD (L_{BCD}) technology of STMicroelectronics using Synopsys Design Compiler Graphical. The physical synthesis flow adopted is industrially qualified by the foundry to correlate well with implementation (presented in Section 5.1.1) and silicon. In order to have a power consumption really correlated with the application, it is considered a real switching activity, annotating Value Change Dump (VCD) files during the device simulation and including also Clock Tree Synthesis (CTS) power estimation. The power estimation P_{eFPGA} is performed considering 1.2 V of supply voltage $V_{DD_{BCD}}$ at room temperature of 25 °C. Therefore, from the average power estimation P_{eFPGA}, following the model introduced in [36], we define the power density $P_{d_{eFPGA}}$ as:

$$P_{d_{eFPGA}} = \frac{P_{eFPGA}}{f_{eFPGA}} \tag{5.1}$$

where f_{eFPGA} is the eFPGA operating clock frequency. The power density represents the average energy per clock cycle of the eFPGA. Since all the simulations are performed using the system introduced in Section 5.1.1, and reminding the proposed system-on-chip architecture from Chapter 3, the eFPGA clock frequency f_{eFPGA} is the PULPino clock frequency f_{PULP} scaled by a factor n_{div}. Consequently, multiplying the power density for the number of clock cycles n_{tick} needed to execute the task, we obtain the eFPGA energy per task:

$$E_{eFPGA} = P_{d_{eFPGA}} \cdot n_{tick} = \frac{P_{eFPGA}}{f_{eFPGA}} \cdot n_{tick} \tag{5.2}$$

In hardware programmable devices, n_{tick} represents the latency in terms of clock cycles and typically this kind of device updates the outputs in one clock cycle. Hence, the latency in terms of time is the number of clock cycles per task times the clock period.

5.2.2 PULPino - Efficiency Model

The energy performance of PULPino is estimate as the previous case of the eFPGA since they are in the same hardware physical implementation. Thus, starting from the PULPino average power estimation P_{PULP} to execute a specific task, we define the power density $P_{d_{PULP}}$ as the power consumption and operating clock frequency f_{PULP} ratio:

$$P_{d_{PULP}} = \frac{P_{PULP}}{f_{PULP}} \tag{5.3}$$

Considering the optimistic case that the processor executes one instruction per clock cycle and hence, without considering possible pipeline stalls, the power density $P_{d_{PULP}}$ represents the average energy per instruction. Therefore, considering that the processor executes n_{insn} number of instructions to execute e certain task, the average energy per task is:

$$E_{PULP} = P_{d_{PULP}} \cdot n_{insn} = \frac{P_{PULP}}{f_{PULP}} \cdot n_{insn} \tag{5.4}$$

supposing the mentioned optimistic case of one instruction per clock cycle. Typically, in a microcontroller implementation, one tries to use as much as possible the available peripherals in order to both reduce the power consumption and allow the processor to execute other computations. Since peripherals are specific and slightly configurable ASIC that they carry out their tasks more efficiently than a processor, application designers tend to move the computation over them. But, usually peripherals are not able to perform tasks without any cooperation with the processor, hence, interrupt-based paradigm is the most efficient way to manage the cooperation between both. However, in an Interrupt Service Routine (ISR) there are both prologue and epilogue to preserve the computation consistency, which means that the processor has to save and/or reload relevant registers, manage the stack pointer to avoid interrupt nesting.

From the assumption of one instruction per clock cycle, the PULPino latency in terms of clock cycles is the number of instructions n_{insn} to be executed, and hence in terms of time, the latency is the number of instructions times the clock period.

5.2.3 ASIC - Efficiency Model

The energy performances of the ASIC solutions are estimated as in the previous two cases, starting from the ASIC power consumption estimation P_{ASIC}, based on a real switching activity and annotating both parasitic and clock tree power consumption. Then, the ASIC power density $P_{d_{ASIC}}$ is defined as:

$$P_{d_{ASIC}} = \frac{P_{ASIC}}{f_{ASIC}} \tag{5.5}$$

where f_{ASIC} is the ASIC operating working frequency. The power density represents the average energy per clock cycle of the specific ASIC implementation. Hence, similar to the eFPGA case, the ASIC average energy per task E_{ASIC} is the power density times the number of required clock cycles to carry the task n_{tick}:

$$E_{ASIC} = P_{d_{ASIC}} \cdot n_{tick} = \frac{P_{ASIC}}{f_{ASIC}} \cdot n_{tick} \tag{5.6}$$

For the ASIC latency are still valid the eFPGA considerations, hence, the ASIC latency in terms of clock cycles is n_{tick}, while in terms of time is the number of required clock cycles times clock period. Generally, in ASIC implementations, the required clock cycle to perform a task is just one.

5.2.4 STM32 - Efficiency Model

The STM32 energy performances are performed from experimental measurements. Figure 5.3 reports the measurement setup which is mostly based on the STMicroelectronics X-NUCLEO-LPM01A. LPM01A is a dedicated system that provides the power supply to the STM32 and it is also capable of measuring the current consumption of STM32. Hence, as reported in Figure 5.3, we measure the average supply current to execute the task, waking up the processor from sleep mode, and thus, knowing the supply voltage, it is possible to compute the average power consumption of the specific task P_{MEAS}. Microcontroller is configured in low-power mode, using a Multi-Speed Internal (MSI) clock in Range 3 at 524.288 kHz, with a supply voltage of 1.8 V which corresponds to a core supply voltage of 1.2 V [14]. Since the STM32 is realized in different technology compared to the solutions implemented in BCD technology, it is needed to normalized the STM32 measurement data to BCD technology. In this way, it will be possible to compare all the four implementations. The STM32L152RE with the previous configuration works with a supply voltage of 1.8 V and its technology node L_{MEAS} is at 130 nm. Whereas, BCD technology has a

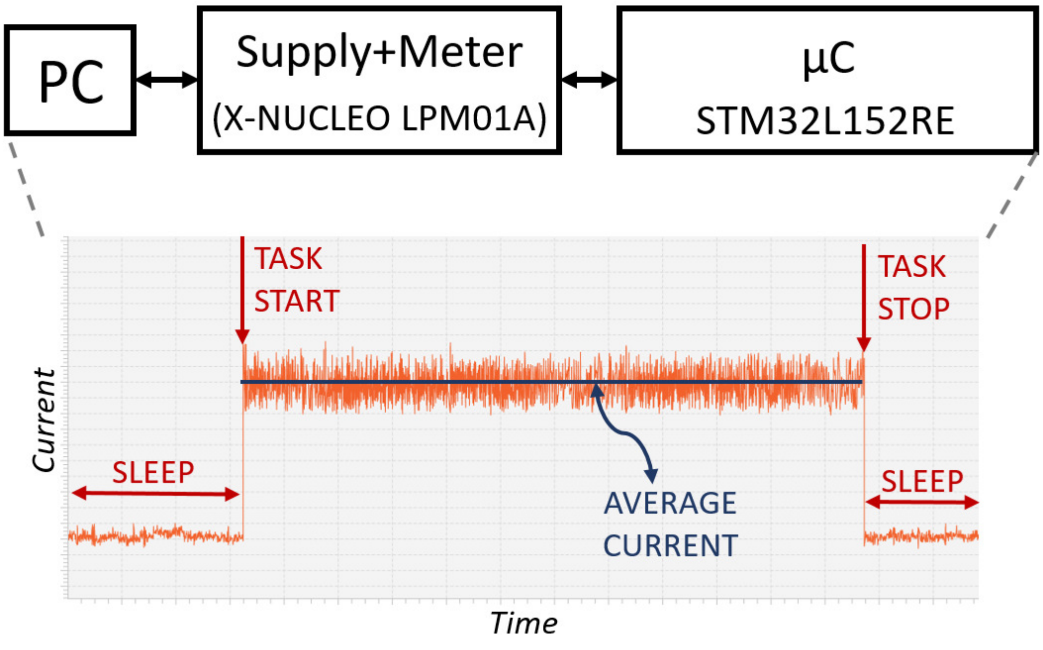


Figure 5.3: Measurement setup and current profile. Reused from [1].

supply voltage VDD_{BCD} of 1.2 V and the technology node L_{BCD} is at 90 nm. Therefore, through the generalized scaling theory [63] it is possible to scale the STM32 measured power consumption to the BCD, defining two scaling parameters, one electrical κ, and one geometrical λ:

$$\begin{cases} \kappa = \dfrac{VDD_{MEAS}}{VDD_{BCD}} = \dfrac{1.8\,V}{1.2\,V} \\ \lambda = \dfrac{L_{MEAS}}{L_{BCD}} = \dfrac{130\,nm}{90\,nm} \end{cases} \tag{5.7}$$

Hence, the STM32 measured power consumption P_{MEAS} scaled to BCD technology power consumption P_{STM32} is:

$$P_{STM32} = \frac{P_{MEAS}}{\lambda \cdot \kappa^2} \tag{5.8}$$

Again is possible to define the power density as power consumption P_{STM32} (scaled for BCD technology) over the operating clock frequency f_{STM32} (which is 524.288 kHz):

$$P_{d_{STM32}} = \frac{P_{STM32}}{f_{STM32}} \tag{5.9}$$

Then, considering the same optimistic hypotheses which are that processor executes one instruction per clock cycle and its execution is pipeline stalls free, one can define the STM32 average energy per task as:

$$E_{STM32} = P_{d_{STM32}} \cdot n_{insn} = \frac{P_{STM32}}{f_{STM32}} \cdot n_{insn} \tag{5.10}$$

where n_{insn} is the number of instructions needed to be executed to carry out the specific task.

As for the previous PULPino case, STM32 latency, expressed in clock cycles, is the number of instructions that have to be executed to perform the task (with the same optimistic supposition of one instruction per clock cycle). While in terms of time it is the number of instructions times the clock period.

As already mentioned, microcontrollers and especially commercial microcontrollers have a variety of dedicated peripherals such as Pulse-Width Modulation (PWM) controller, timers, etc., in order to increase the system performance in terms of power consumption and latency. In this way, it is possible to avoid the software-programmable device computational model, while exploiting hardware devices since dedicated peripherals tend to be "programmable" ASICs.

5.2.5 Energy Gain and Latency

As explained in Equation (5.2), (5.4), (5.6) and (5.10), both eFPGA and ASIC, and PULPino and STM32 have the same computational model. The first couple is representative of hardware implementation devices whereas the second one is for software-programmable devices. The difference is the power density multiply factor that in hardware-programmable devices is the number of clock cycles (which is typically one) while in a processor implementations is the number of instructions to be executed. Hardware implementations directly map in hardware the digital block with no need to execute a set of sub-instructions to obtain the same results, as happened for the software-programmable solutions. Therefore, considering the energy per task and not the power consumption it is possible to compare the performances of different solutions to carry out different applications. In order to qualify the performance of the different implementations, especially to find out which one is the most efficient solution in our proposed system-on-chip to manage a specific task, we define the energy gain between PULPino and the eFPGA as follows:

$$E_{GAIN} = \frac{E_{PULP}}{E_{eFPGA}} \tag{5.11}$$

This model is clearly optimistic for PULPino because, in our estimation E_{PULP} does not take into account processor pipeline stalls, and it assumes that the microprocessor executes one instruction per cycle. This energy gain evaluates whether a specific task is better handled in the eFPGA (used as a processor programmable peripheral) when $E_{GAIN} > 1$ or in the microprocessor $E_{GAIN} < 1$. This energy ration can guide the system application designer to an efficient task partitioning between the processor and its configurable peripheral. In addition, for a SoC designer, the possible energy gain can justify the non-negligible area overhead.

For a latency point of view, considering PULPino and the eFPGA working at the same frequency, for a specific task the responsiveness gain is expressed with the processor number of instructions n_{insn} and eFPGA clock cycles n_{tick} ratio. Hence, to aim at a desired latency with PULPino, one can both:

- optimize as much as possible the execution code, even if it is not always possible;
- increase the operating clock frequency, keeping in mind that there is the physical upper limit of the maximum implementation frequency.

Regarding the eFPGA, one can just set the operating clock frequency since the number of clock cycles necessary for the task is usually one. eFPGA clock frequency, as well as PULPino clock frequency, has the same upper limit which is the maximum implementation frequency. Latency consideration too can address the system application designer to the right task partitioning between PULPino and the eFPGA, and justify the area overhead introduced by the eFPGA.

5.3 Application Results

In this section, we analyze different application domains starting from control applications, then streaming tasks and in the end ultra low-power application. Each application kind has different peculiarities, for instance, control applications generate signal pattern at the outputs based on inputs. Hence, in this case, the computational base model is described through a Finite State Machine (FSM). Concerning bitwise streaming applications, the key point is the continuous data stream which is processed at a bit level. In ultra low-power tasks, the elaboration unit has to simple manage data, reducing as much as possible the power consumption. The different kinds of applications should fit with the different elaboration unit architectures. Through the energy comparison, we evaluate how the proposed SoC is efficient in managing different kinds of applications. In Table 5.2 are summarized all the data of the different four considered devices (eFPGA, PULPino, ASIC and STM32) to manage different applications.

5.3.1 Control Applications

In control applications, the computational base model of the elaboration unit is described through an FSM, since the elaboration unit has to generate specific patterns at outputs based on the inlets. The required data processing is "simple" and it is not needed high-density computing because the controller behavior is basically an FSM, which updates the outputs (and the internal state) based on the inputs (and the internal state). This kind of application typically interacts with the real physical world since both inputs and outputs are from/to some physical quantities. Usually one wants to manage electrical quantities, such as average voltage or current based on some kinds of input such as both user inputs and feedback sensor outputs

Table 5.2: Applications Results. Reused from [1].

		Applications						
		PWM	RGB LED	BRUSHED	STEPPER	CRC16	LFSR	WUR
eFPGA	f_{eFPGA} [MHz]	1.25	1.25	1.25	1.25	1.25	1.25	1.25
	CLBs	7	11	7	4	4	15	5
	P_{eFPGA} [μW]	20.86	28.45	22.75	21.86	64.85	110.95	34.05
	$P_{d_{eFPGA}}$ [μW/MHz]	16.69	22.76	18.2	17.49	51.88	88.78	27.24
	n_{tick}[c]	1	1	1	1	1	1	1
	E_{eFPGA} [pJ]	16.69	22.76	18.2	17.49	51.88	88.78	27.24
PULPino	f_{PULP} [MHz]	10	n.a.[a]	10	10	10	10	n.a.[b]
	P_{PULP} [μW]	984	n.a.	1082.5	1058.4	1050	886	n.a.
	$P_{d_{PULP}}$ [μW/MHz]	98.4	n.a.	108.25	105.84	105	88.6	n.a.
	n_{insn}[c]	77	n.a.	110	145	8	42	n.a.
	E_{PULP} [pJ]	7576.8	n.a.	11907.5	15346.8	840	3721.2	n.a.
ASIC	*Area* [μm^2]	442.3	732.1	512.6	340.26	554.3	1174.4	367.7
	Eq. gates	156	260	179	125	201	442	134
	f_{ASIC} [MHz]	10	10	10	10	10	10	0.1
	P_{ASIC} [μW]	2.67	3.73	2.41	4.19	11.3	29.3	0.0398
	$P_{d_{ASIC}}$ [μW/MHz]	0.267	0.373	0.241	0.419	1.13	2.93	0.398
	n_{tick}[c]	1	1	1	1	1	1	1
	E_{ASIC} [pJ]	0.267	0.373	0.241	0.419	1.13	2.93	0.398
STM32	f_{STM32} [MHz]	0.524288	0.524288	0.524288	0.524288	0.524288	0.524288	n.a.[b]
	P_{STM32} [μW]	1.67	2.769	69.785	70.892	82.523	75.877	n.a.
	$P_{d_{STM32}}$ [μW/MHz]	3.185	5.282	133.1	135.22	157.4	144.724	n.a.
	n_{insn}[c]	sleep	sleep	79	111	10	15	n.a.
	E_{STM32} [pJ]	3.185	5.282	10515.24	15009	1574	2170.86	n.a.

[a] PULPino implementation is too inefficient
[b] μC implementations are not used for ultra-low-power applications
[c] it corresponds to latency in an iso-frequency case

and hence, control applications are event-driven applications. As already said, control applications are related to the physical world, for instance, motor controllers, smart-switches, switching regulators, etc., and therefore are smart power control applications.

As it will be explained, PULPino is less efficient than the eFPGA in managing simple finite state machines with few inlets/outlets and internal states since a microprocessor is oversized for a simple FSM. While eFPGA maps the FSM in hardware, the processor has to implement an FSM in software and hence, has to execute many instructions just to update both the outputs and the internal state. In the processor implementation, the whole processor pipeline has to work to manage the FSM and thus, processor architecture plays an important role in overall efficiency. However, processor at every clock cycle accesses to instruction memory to fetch instructions, decodes and executes them, and potentially it might have to read and write-back data with bigger parallelism than the required by the finite state machine. Since control applications are event-driven ones, the most efficient way to implement them with a microprocessor is based on the interrupt paradigm, coupling peripherals and when it is necessary the processor. Using an interrupt paradigm one optimizes the power consumption, but on the other hand, inefficient factors remain. Interrupt Service Routines (ISRs), which are activated when necessary, manage the task but, to preserve the computation consistency they have both prologue and epilogue substantially affecting the power consumption [36].

The considerations regarding the latency are still valid. The latency is related to both the clock period and the number of required clock cycles to execute the task, hence, hardware-based solutions are more reactive than software-programmable solutions, since they respond in just 1 cycle. Therefore, in an iso-frequency case, the processor has to increase its operating clock frequency by an $\simeq n_{insn}$ (number of executed instructions) factor (assuming the execution of one instruction per cycle), which results in a straightforward drawback in terms of power consumption.

As control application testbenches are considered a Pulse-Width Modulation controller, an RGB LED controller and both brushed and stepper motor controllers.

Pulse-Width Modulation

Pulse-Width Modulation (PWM) is widespread (due to its simplicity) technique to modulate physical quantity. It consists of the generation of rectangular wave and through the duty cycle, one can modulate the average value of the physical quantity as schematized in Figure 5.4. In general, one can define the period and the duty cycle of the signal.

As testbench, we implement a PWM controller with 8-bit programmable period and duty cycle. This controller is described in HDL code and synthesized in an ASIC solution. As reported in Table 5.2, the ASIC version of the PWM controller has a dimension of 442.3 μm^2 which corresponds to 156 equivalent gates and an operating clock frequency of 10 MHz. The same controller HDL code is synthesized in the

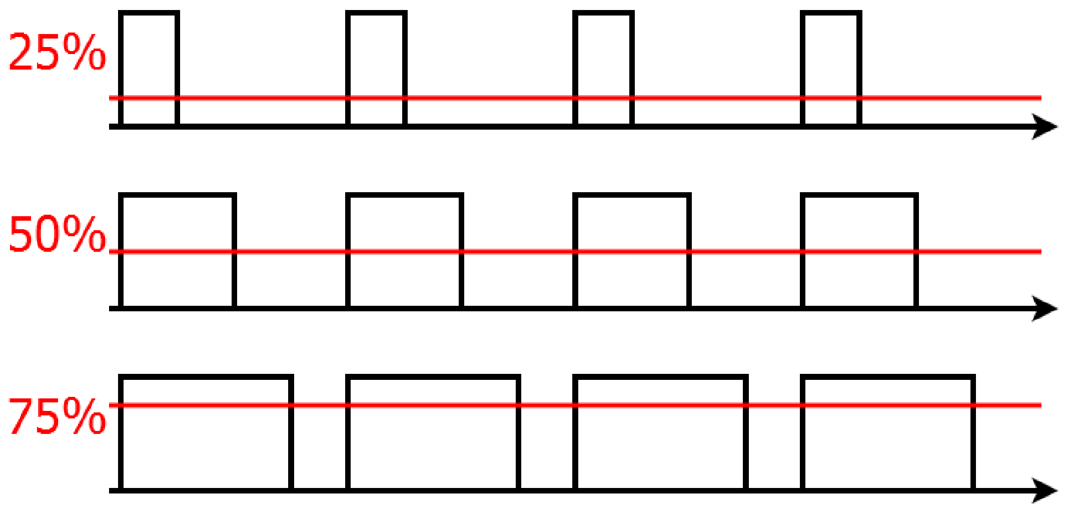


Figure 5.4: PWM signals and corresponding average value with different duty cycles: 25%, 50% and 75%.

eFPGA using 7 of the 16 available CLBs. In PULPino implementation the PWM controller is implemented using two timers to generate the timing and the interrupt controller since PULPino does not have a dedicated PWM peripheral. When the timers finish the count, the interrupt controller generates an interrupt and then, the processor in the Interrupt Service Routines (ISRs) manages the outputs. Microprocessor, when it is not executing the ISRs, is in sleep mode to reduce power consumption. PULPino is designed targeting a clock frequency of 50 MHz, but now, the operating clock frequency is 10 MHz. The eFPGA operating clock frequency f_{eFPGA} is scaled from the PULPino clock frequency by a factor n_{div} of 8 and hence, it is 1.25 MHz. STM32 is an example of a commercial microcontroller equipped with different peripherals including the PWM ones, useful to perform a comparison with devices available on the market. We measure the power consumption of the STM32L152RE which is an ultra low-power microcontroller as reported in Figure 5.3. Initially, we perform the measurements with processor in low-power sleep mode and the GPIOs in analog mode and then activating only time 3 (TIM3) used for timing generation and GPIO port B to generate the output signal. The power consumption change is due to the peripheral operation to generate the PWM signal. Naturally, the measured power consumption is scaled in BCD through equation (5.8).

Therefore, the resulting energies per task for the different solutions are 0.267 pJ for ASIC, 16.69 pJ for eFPGA, 7576.8 pJ for PULPino and 3.185 pJ for STM32L152RE as summarized in Table 5.2. Clearly, ASIC implementation has the best achievable performance but, on the other hand, it is not reconfigurable. The PWM peripheral lack in PULPino has really strong impact on the inefficiency of the system, since processor has to wake up from sleep mode to execute the interrupt service routines, and hence, it has to fetch, decode, execute, read/write data just only to update few bits of the FSM that manage the PWM controller. As expected the eFPGA fits better than

PULPino the FSM achieving an energy gain E_{GAIN} (equation (5.11)) of 454. The usage of a commercial solution, in this case, yields good energy performance since the employed peripheral tends to an ASIC implementation enhanced with some configuration parameters. Typically these kinds of microcontroller peripherals can be configured but just only for a specific function and not fully-configured as an eFPGA.

RGB LED Controller

This application testbench is an extension of the previous one since an RGB LED controller has to manage the three components of the color, Red, Green, and Blue. Hence, the easiest way to modulate the color contribution is through a PWM signal [64]. Therefore, the RGB LED controller has three PWM controllers one for each color component and thus, the application analysis is similar to the previous one.

The designed controller has $3\times$ PWM controllers with an 8-bit programmable duty cycles each and 8-bit programmable period common to every color. The controller described in the HDL code implemented in ASIC has an area occupation of 732.1 μm^2 resulting in 260 equivalent gates. The same HDL code synthesized in the eFPGA occupies 11 CLBs. The RGB LED controller is not implemented in PULPino since the lack of the required peripherals results in a too inefficient implementation, which implies the processor busy all the time in managing the three PWM signals. The usage of a processor just to realize three PWM signals produces a straightforward energy waste. On the other hand, the STM32 implementation is very efficient thanks to the exploitation of the hardware peripherals. For this testbench, we follow the approach of the previous application to compute the STM32 energy per task, keeping the microprocessor in low-power sleep mode and using a timer peripheral (TIM4) to generate the timing for the three channels, and the GPIO port B to manage the outputs and inputs.

The ASIC, first in class, has an energy per task E_{ASIC} of 0.373 pJ, 22.76 pJ for the eFPGA E_{eFPGA} and 5.282 pJ for the STM32 E_{STM32}, as reported in Table 5.2.

Brushed Motor Controller

Brushed motors are DC motors extensively used in different application domains. Since they are DC motors, they are usually driven through half or full H-bridge circuits [65]. As well known, this kind of circuit allows the motor to run forwards or backwards inverting the current direction. The H-bridge is historically realized in discrete components but now, several semiconductor companies are developing dedicated integrated circuits with both logic controller and power devices and not just only the digital controller. In this testbench, we consider as actuation circuit the full H-bridge since for the half H-bridge are valid the same considerations. The digital controller has to generate the signal patterns for the four transistors composing the full H-bridge. In Figure 5.5(a) are reported the required signals for running mode

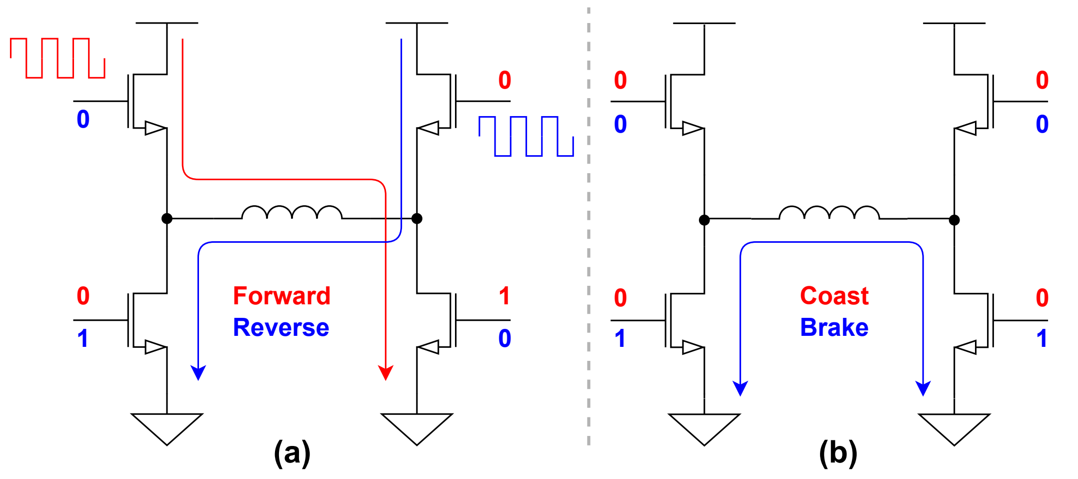


Figure 5.5: Brushed controller H-bridge operating modes. (a) forward - red - and reverse - blue - mode. (b) coast - red - and brake - blue - mode. Picture reused from [1].

in forward and reverse directions (chosen through an input). Applying a PWM signal one can modulate the motor speed acting on the duty cycle. Figure 5.5(b) describes two stopping techniques, in brake mode, the motor is abruptly stopped whereas in coast mode it is free to relax. This controller has a PWM controller extended with a more complex finite state machine that has to handle the full H-bridge transistors based on the desired running and stopping mode. The designed brushed motor controller has a PWM generator with both an 8-bit programmable both period and duty cycle and the finite state machine to manage all the full-bridge signals.

The digital controller designed in HDL implemented in an ASIC solution has an area occupation of 512.6 μm^2 which corresponds to 179 equivalent gates with an operating frequency of 10 MHz. The same HDL code synthesized in the eFPGA occupies 7 CLBs and the operating clock frequency of the eFPGA is set at 1.25 MHz. The PULPino implementation is based on an interrupt paradigm and it uses timers to generate the timing for the PWM signal, GPIOs to handle both inputs and outputs and interrupt controller for the interrupt management. PULPino microprocessor has to execute 110 assembly instructions just to update the finite state machine states. The same approach is used for the STM32 implementation, using GPIO, interrupt controller and a timer peripheral. In this implementation, the processor has to execute 79 assembly instructions to update the FSM.

As reported in Table 5.2, the energy per task of the ASIC implementation E_{ASIC} is 0.241 pJ, while eFPGA energy per task E_{eFPGA} is 18.2 pJ. Both microcontroller solutions are comparable since they have quite similar architecture, both are based on Harvard architecture and the RI5CY core ha 4-stage RISC pipeline while the ARM Cortex-M3 has 3-stage RISC pipeline [66]. Therefore, they have a comparable number of assembly instructions to handle the FSM and a similar energy per task which is 11907.5 pJ for PULPino and 10515.24 pJ for the STM32. In this testbench, the achieved eFPGA energy gain over PULPino is 654.

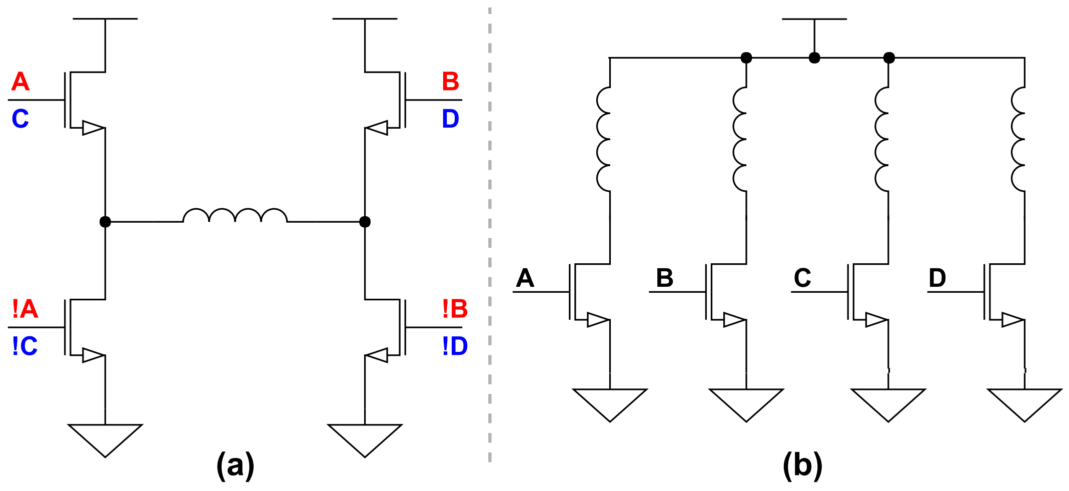


Figure 5.6: Driver circuits for stepper motors. Two full H-bridges - respectively red and blue inputs - for bipolar stepper motors (a). Transistor scheme for unipolar stepper motors (b). Picture reused from [1].

Stepper Motor Controller

Stepper motors are brushless DC electric motors [67] and their full rotation is divided into equal steps, and the controller has to drive every single step and therefore, they are usually driven in open-loop configuration without negative feedback. Typically there exist two-phase stepper motors, the bipolar one and the unipolar one. Both bipolar and unipolar stepper motors have the same digital controller but different actuation circuit. In Figure 5.6(a) are reported two (the first one has red inputs while the second one has blue inputs and !x is the Boolean operation not(x)) full H-bridge circuits for bipolar stepper motors, since they have two coils. Figure 5.6(b) shows an actuation circuit for unipolar stepper motors which have 4 coils. Through different control techniques, one can modulate the torque and the angular resolution while the frequency impacts the motor speed. In Figure 5.7 are reported the considered control modes where the blue letters are for bipolar stepper motor and the red are for unipolar motors.

- In full-step mode (Figure 5.7(a)) one has always two phases on, and hence the motor provides the maximum rated torque.
- In half-step mode (Figure 5.7(b)) the actuation circuit alternates between two phases on and a single phase on increasing the angular resolution but reducing the available torque.
- In wave-drive mode (Figure 5.7(c)) the motor has always just a phase on, involving a torque reduction.

When a stop input is activated, the controller generates no more signals even though one or two remain active, depending on the selected control policy. Thus, in order to

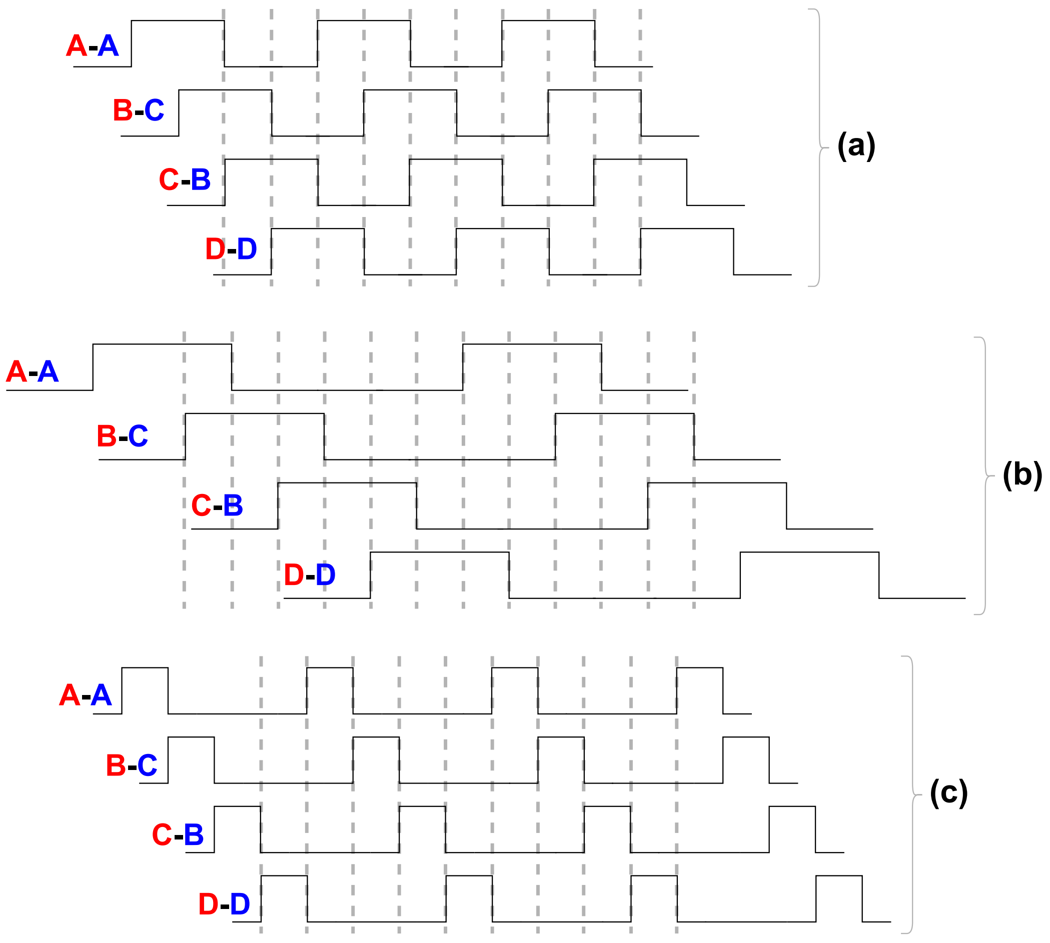


Figure 5.7: Signals patterns for unipolar - red - and bipolar - blue - for: full-step (a), half-step (b) and wave-drive control modes. Picture reused from [1].

avoid motor short circuit current while keeping the torque, a PWM signal is applied to the active phases.

The designed controller works for the mentioned control policies and it has an 8-bit programmable PWM both period and duty cycle. The ASIC implementation of the stepper motor controller has an area occupation of 340.26 μm^2, which corresponds to 125 equivalent gates. The same HDL code synthesized in the eFPGA occupies 4 CLBs of the 16 available. Since time unit is important, one can set the controller time resolution (vertical grey dashed line of Figure 5.7) acting on the eFPGA prescaler. In the PULPino solution, it is used the previous methodology, timers are used to generate the time unit and the processor manages the finite state machine. GPIOs are necessary for input and output ports, and the interrupt controller handles the interrupts generated by the timers and the GPIOs. PULPino core to handle the FSM state executes 145 assembly instructions. In the STM32 implementation, the approach is the same, the timer generates time resolution, the GPIOs handle inlets and outlets and the interrupt controller allows the core to be in sleep mode. In this case, the core has to execute 111 assembly instructions to manage the finite state machine.

As reported in Table 5.2, the equivalent ASIC energy per task E_{ASIC} is 0.419 pJ, while for the eFPGA is 17.49 pJ. As already mentioned, both microcontroller implementations have comparable architecture, and sure enough, they have similar energy performance. PULPino implementation has an energy per task E_{PULP} of 15346.8 pJ, whereas STM32 has 15009 pJ of energy per task. For this testbench, the energy gain E_{GAIN} between eFPGA and PULPino is 877.

5.3.2 Bitwise Streaming Applications

Bitwise streaming applications have to elaborate data applying some kind of operation. In this case, the system has a structure of data flow, unlike the previous case where the computational base model is a finite state machine. A microprocessor has an intrinsic data flow structure in its pipeline where data are elaborated. On the other hand, hardware-paradigm devices must be configured to execute specific operations. In this case, microprocessors exploit their structure better than in the case of control applications, especially if the required operations are available in the processor instruction set architecture. As bitwise streaming application testbenchs, we consider both an error-detecting code generator (Cyclic Redundancy Check) and pseudo-random number generator. Both of them are typically used in cryptography, memories and storage devices, and communication protocols.

Cyclic Redundancy Check

Cyclic Redundancy Check (CRC) is an error-detecting code widespread used in different applications. In this testbench, we consider the CRC16 with a programmable polynomial. For the performance estimation, we use the polynomial $x^{16} + x^{15} + x^2 +$

x^0 used in both industrial communication protocols such as Modbus and the USB. In order to perform CRC computation, the elaboration unit has to execute both xor and shift operations on the data.

The ASIC implementation of the CRC16 has an area occupation of 554.3 μm^2 which corresponds to 201 equivalent gates. The same HDL code synthesized in the eFPGA uses 4 CLBs of the 16 available, generating CRC at every clock cycle. In the case of PULPino implementation is used a fast-CRC algorithm based on hash tables [68] which uses 8 assembly instructions n_{insn}. The same approach of a hash table is used for the STM32 implementation which corresponds to 10 required assembly instructions n_{insn}.

As reported in Table 5.2, the ASIC implementation energy per task E_{ASIC} is 1.13 pJ, which is slightly higher than the previous cases. Since the ASIC area occupation is almost the same (and hence, the power consumption due to the leakage current), the energy increase is due to the increase in switching activity. Obviously, the ASIC implementation is still the best in terms of performances, but if one wants to change the data-width or the CRC polynomial, it has to redesign and refabricate the integrated circuit. The same thing happens to the eFPGA. In this case, the eFPGA uses the same number of CLBs of the stepper motor controller but consuming more power because is increased the switching activity. In the stepper motor case just a few bits of the FSM switch at the same time while in this testbench more bits are changing. The resulting eFPGA energy per task E_{eFPGA} is 51.88 pJ. Processor implementations have almost constant power consumption when the pipeline is working, and then, the assembly instruction number impacts on energy efficiency. PULPino energy per task E_{PULP} is 840 pJ while for the STM32 solution E_{STM32} is 1574 pJ. The achieved energy gain E_{GAIN} between the eFPGA and PULPino is 16, which is smaller than that reached in the previous testbenches of control applications. For the processor implementations, supposing the execution of one instruction per clock cycle, in order to obtain the same throughput of the eFPGA, the processor clock frequency f_{CPU} should be:

$$f_{CPU} = \frac{f_{eFPGA}}{n_{tick}} \cdot n_{insn} \tag{5.12}$$

where n_{tick} is the number of required clock cycles to execute the task (which is typically 1), and n_{insn} is the number of the required Central Processing Unit (CPU) assembly instructions to execute the same task.

Pseudo-Random Number Generation

The pseudo-random number generation is a methodology to generate a sequence of numbers using mathematical algorithms. One of the easiest algorithm to generate pseudo-random numbers is based on Linear Feedback Shift Registers (LFSRs). The pseudo-random number generator designed is a 16-bit LFSR.

The ASIC implementation has an area occupation of 1174.4 μm^2 which corresponds to 442 equivalent gates. In this case, the area occupation is bigger than the

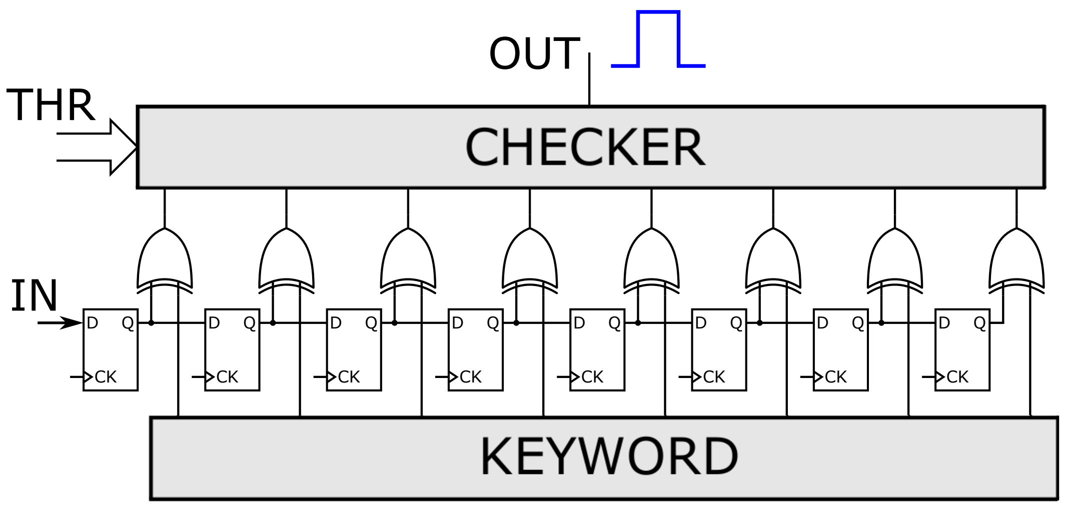


Figure 5.8: Wake up radio correlator architecture. Picture reused from [1].

previous cases since the task complexity is higher. The same ASIC HDL code is synthesized in the eFPGA and it uses 15 CLBs of the 16 available CLBs. The micro-controller approach is the same followed in the previous testbench, which is based on the fast-LFSR algorithm using hash tables. PULPino solution needs of 42 assembly instructions to generate a pseudo-random number, while the STM32 uses just 15 assembly instructions.

As reported in Table 5.2 the ASIC power consumption P_{ASIC} is increased becoming 29.3 μW compared to the previous testbenches. The ASIC area occupation is increased too and hence, also the contribution to the power consumption due to the leakage power should be increased. However, in this technology (90 nm BCD technology) the power consumption due to the leakage is negligible, for instance, in this case of the overall power consumption (of 29.3 μW) just 10.3 nW is due to power leakage which is almost the 0.035%. The corresponding ASIC energy per task E_{ASIC} is 2.93 pJ. The eFPGA implementation has an energy per task E_{eFPGA} of 88.78 pJ, while the microcontroller implementations have 3721.2 pJ PULPino energy per task E_{PULP} and 2170.86 pJ STM32 energy per task E_{STM32}. In this case, the energy gain between the eFPGA and the PULPino E_{GAIN} is 42 which is again smaller than control applications. For a throughput point of view, it is still valid the equation (5.12), and hence, to reach the same eFPGA throughput, PULPino should have a clock frequency of the eFPGA multiplied by an n_{insn} factor.

5.3.3 Ultra Low-Power Applications

In ultra low-power applications the main goal is to have the smallest reachable power consumption. For this reason, the most used devices are ASICs designed for a specific application avoiding the usage of software-programmable devices that are not appealing due to their energy inefficiency. As analyzed in the next chapter, nowadays, the aim is to connect everything to each other, and hence, is required excellent energy performance. In order to optimize the power consumption of each node, the communication interface can be switched off when is not required since it strongly affects the overall system power consumption. As shown in [69], a wake up radio (WUR) which is an always-on minimalist radio interface capable of recognize a simple message and wake up the sleeping system. For instance, the designed controller receives the bitstream from the analog front-end (presented in the next chapter and in [69]) and it is able to detect an 8-bit word. In Figure 5.8 is reported the architecture of the proposed digital correlator. The correlator is capable to compare 8-bit of the serial received bitstream with a predefined "keyword" through xor gates. The xor outputs are connected to a "checker" block which is a combinational logic circuit and it counts how many xor gates have "1" at the output. The count of how many "1"s are at xor outputs means how many received bits are equal to the keyword. Then, if the number of "1"s at xor outputs is greater than a defined threshold (programmed with THR of Figure 5.8), the correlator generates a wake up signal (blue line of Figure 5.8) for the node elaboration unit. Since the bit rate of the analog front-end presented in the next chapter and in [69] is 1 kHz (chosen to reduce the dynamic power consumption), the designed ASIC targets a clock frequency of 100 kHz. As reported in Table 5.2, the ASIC area occupation is 367.7 μm^2 resulting in 134 equivalent gates. The same controller HDL code synthesized in the eFPGA uses 5 CLBs of the available 16.

The ASIC energy per task E_{ASIC}, which is an 8-bit comparison is 0.398 pJ. Obviously, this implementation is the best from both energy performance and area occupation points of view, even though it has not any reconfigurable features (useful to upgrade the node during its operating life especially in a remotely way). The same controller represented in Figure 5.8 synthesized in the eFPGA requires an energy E_{eFPGA} 27.24 pJ to compare 8 bits.

5.4 Results and Discussion

In this section we discuss the data obtained in the previous section, focusing on both energy and latency performances.

5.4.1 Energy Efficiency Consideration

The overall energy data obtained in the previous section (and summarized in Table 5.2), for each testbench and each elaboration unit implementation, are reported in Figure 5.9. Figure 5.9 shows the energy per task of ASIC, eFPGA, PULPino and STM32 for the considered applications. The chart highlights how different architectures impact on energy performance.

As well known, ASIC implementation has the best performance in terms of energy efficiency since the specific circuit is designed and optimized for the specific

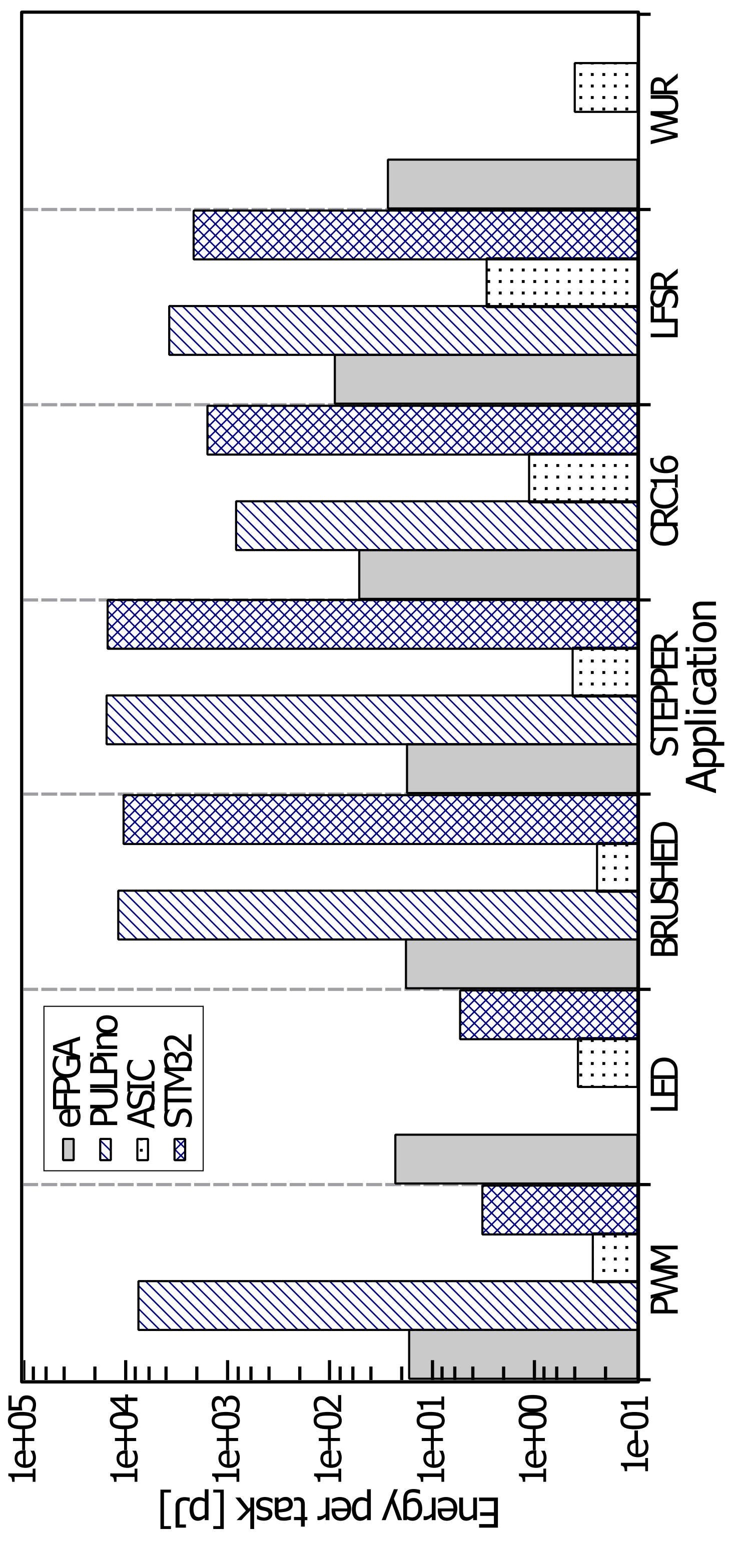


Figure 5.9: The energy efficiency derived from equations (5.2), (5.4), (5.6) and (5.10) - in log scale - and related by number of both CLBs and instructions to carry out the required functionalities. Picture resued from [1].

task. Because of this reason, the energy per task is increasing with the required computational complexity, as visible in the chart, impacted by the switching activity and not by leakage power (for the considered 90 nm BCD technology). Indeed, as reported in Figure 5.9 (dotted bars) and Table 5.2, from control applications to bitwise streaming applications, the energy per task increases, due to an increase in the switching activity. Since the computational base model of control applications is a simple finite state machine with few bits changing at the same time, whereas is a data flow for bitwise streaming applications. ASIC is the best in class and hence, it is impossible to exceed its energy performance.

The eFPGA has the same behavior of the ASIC, its average power consumption P_{eFPGA} is related to both application complexity and its switching activity, as shown in Figure 5.9 (solid bars) and Table 5.2. Therefore, the eFPGA power consumption P_{eFPGA} goes from 20.86 μW (for simple PWM application) to 110.95 μW (for the more dynamic and larger LFSR application) which corresponds to around a $5\times$ increase. The power consumption and hence, the energy per task of both the hardware-based devices, ASIC and eFPGA, follow the same trend as visible in Table 5.2. The eFPGA due to its reconfigurability has a power and energy overhead, which is the offset between the two hardware implementation.

On the other hand, the microprocessor has a fixed structure designed to execute instructions. At every clock cycle, a processor fetches an instruction from the instruction memory, decodes the instruction, executes it and eventually saves data. Therefore the microprocessor power consumption is almost constant especially for single-datapath RISC processor since all the pipeline stages are working, however, the power consumption is not far from to be dominated by the instruction memory access. As shown in Table 5.2, the power consumption of both microcontroller implementations P_{PULP} and P_{STM32}, is almost constant when the processor is working and it is in average 1012.18 μW for PULPino (considering all the available applications) and 74.77 μW for STM32 (without taking into account both PWM controller and RGB LED controller since the processor is in sleep mode). Evidently, the difference in power consumption between the two microcontroller implementations is due to the difference in the operating frequency, which is 10 MHz for PULPino f_{PULP}, and 524.288 kHz for the STM32 f_{STM32}. Therefore, if the average power consumption is constant, the energy per task is not constant and it strongly depends on the number of executed assembly instructions, as shown in Table 5.2, which defines how many clock cycles the pipeline has to work, and hence, using energy. For this reason, it is important how the instruction set architecture tailors the applications. As highlighted before, the energy per task of the eFPGA increases going from control applications (based on finite state machine) to bitwise streaming applications (based on a data flow model). On the contrary, microprocessors decrease their energy per task from control applications to streaming applications as visible in Figure 5.9 (diagonal bars for PULPino). In control applications, microprocessors have to execute

Table 5.3: eFPGA - PULPino Energy Gain. Reused from [1].

	Applications				
	PWM	Brushed	Stepper	CRC16	LFSR
E_{GAIN}	454×	654×	877×	16×	42×

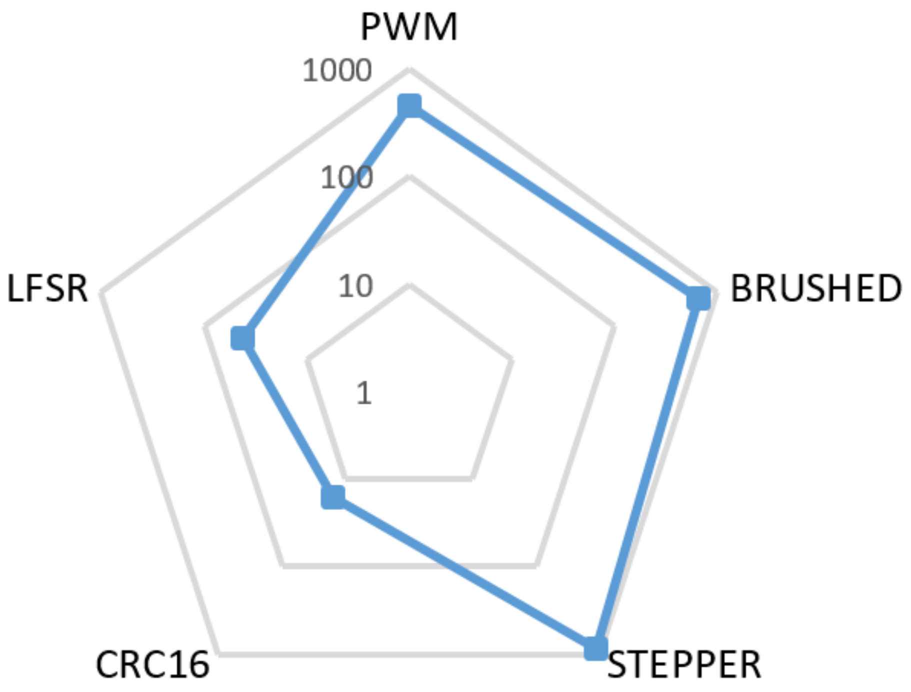


Figure 5.10: Energy gain - in log scale - defined by equation (5.11) for applications implemented in both eFPGA and PULPino. Picture reused from [1].

more instructions than necessary due to the prologue and epilogue of the processor routines as introduced in [36]. This instruction overhead is inevitable since the processor has to preserve the computation consistency.

The energy per task gap between eFPGA and PULPino is hence bigger in control applications and it reduces in bitwise streaming applications as shown in Figure 5.9 and Table 5.2. This energy gap is defined as an energy gain through the equation (5.11). The energy gain for each application is summarized in Table 5.3 and reported in Figure 5.10. Data highlight that for control applications the processor is certainly oversized, developing in time what it can not do in space. That means the processor has to execute many instructions if it has not available a specific instruction for the task. The processor time exploitation results in a decrease in energy performance. Whereas, on streaming applications processor can needless instructions to perform a more complex computation than just a check and a bit update. On the other hand, embedded FPGA develops in space the task, and hence, in this case, the limitation is the device size. In the right side of Figure 5.10 is well shown how the small eFPGA dimension fits better control application, reaching high energy gain. Therefore, for a system-on-chip designed for smart power applications, this energy gain justify the area overhead introduced by the eFPGA peripheral. Besides, Figure 5.10 reveals how should be the task partitioning in the proposed heterogeneous system-on-chip composed of PULPino and its reconfigurable peripheral eFPGA. The control application should be managed by the eFPGA while all other computations by the microcontroller.

The STM32 is a processor implementation and hence, its efficiency is related to the number of required assembly instructions, since its power consumption is almost constant when the processor is working. However, STM32 implementation highlights, as shown in both Table 5.2 and Figure 5.9 (crossbars), how the exploitation of a dedicated peripheral allows the system to strongly increase the performance. This happens because dedicated peripherals tend to be almost ASIC implementation. For instance, in both PWM and RGB LED testbench, the STM32 uses entirely dedicated peripherals achieving good performance worse than real ASIC but better than eFPGA, since the processor is in sleep mode. On the other hand, when is required the processor activity to handle the finite state machine, the performance is deteriorated. However, one may conceivably replace some dedicated peripherals (such as timers, PWM controllers, small pre/post-processing accelerators) with a more reconfigurable peripheral like the proposed embedded FPGA, extending the usage of peripherals instead of a processor for different kinds of application, which proves more efficient in some cases [1].

5.4.2 Latency Consideration

From a latency point of view, we consider the time required to update the outputs of the digital controller. As already introduced, hardware-based devices respond in typically one clock cycle ($n_{tick} = 1$) since their structure directly maps digital controller in hardware.

Software-programmable device latency, supposing they execute one instruction per clock cycle, which is slightly optimistic, is related to the number of executed assembly instructions (n_{insn}). Hence, the microprocessor implementation latency, as well as energy performance, is related to the instruction set architecture. Therefore, the usage of dedicated hardware peripheral allows the system to achieve just one clock cycle latency. For that reason, the eFPGA addition as a reconfigurable peripheral could enhance the latency performance of the system, using just only the eFPGA without any cooperation with the processor as schematized in Figure 5.11.

In Figure 5.11 using the values reported in Table 5.2 for all applications, we consider the eFPGA latency as reference (black trace with triangles). PULPino with the same eFPGA operating clock frequency (iso-frequency) has a latency, in terms of clock cycle referred to the eFPGA, which is equal to the number of executed assembly instructions n_{insn} and it is reported in the chart in red trace with circles. In

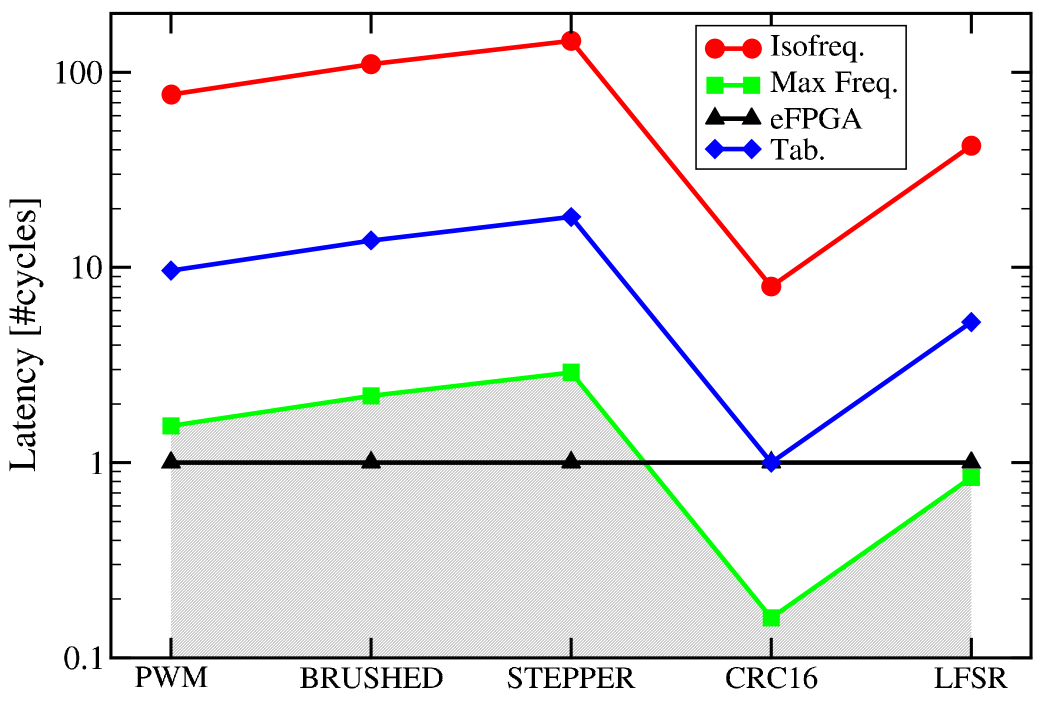


Figure 5.11: PULPino and eFPGA latency (in clock cycles) for the applications under analysis. Picture reused from [1].

this case, the PULPino latency is n_{insn} times the eFPGA latency. The corresponding PULPino latency of Table 5.2 (where the PULPino operating clock frequency is 8 times the eFPGA operating clock frequency) is reported in Figure 5.11 with blue trace with diamond symbols. Therefore, in order to improve the PULPino latency performance, one has to increase the PULPino operating clock frequency with a straightforward drawback in terms of power consumption. To reach the same eFPGA latency performance, PULPino operating clock frequency should be the eFPGA operating clock frequency multiplied by n_{insn} factor, and more in general should be equal to f_{CPU} reported in equation (5.12). However, it is possible to increase the PULPino operating clock frequency just until its maximum frequency which is the green trace with square symbols in Figure 5.11. The filled region of Figure 5.11 represents the "unreachable latency" zone. This area is out of the PULPino domain since the required clock frequency upscaling is over the implementation frequency.

5.4.3 Implementation Solutions Under Analysis

For a qualitative analysis of the considered implementation solution performances, we evaluate:

- the *configurability* allows one to reuse the device for different applications. In this way, a microelectronics company can produce one component covering different application fields. As well known, configurability impacts directly on the area occupation of the device. Besides, configurability affects the performance of the device in terms of both power consumption and latency.

- The *area* occupation mainly depends on the configurability features of the device. Certainly, it affects the device cost and hence, the possible market.
- The *efficiency* depends primarily on both the computation base model and the application requirements.
- The *latency* depends directly on the computational base model too. As shown in the previous sections and chapters, hardware-paradigm devices allow the system to reach the best performance in terms of latency.

As explained in the following, all these features are summarized in charts for each implementation solution in order to give a qualitative idea of the specific device capabilities. The chart scales go from 1 to 10, and as bigger are the numbers as the device under analysis has better performances. It does not exist a unique choice, and one should decide its strategy based on both application needs and its available resources. We summarize all these four features for all the considered elaboration unit architectures.

Microcontroller Implementation

Microcontrollers as already shown, combine software-programmable features with some dedicated hardware peripherals. In Figure 5.12 are summarized the main four features of a device. Both microcontrollers PULPino and STM32 have very good performance in terms of configurability. They have software-programmable features and hence, they provide the easiest programmable features. Therefore, the main microcontroller feature, as reported in Figure 5.12, is the ease of use and then, its general-purpose characteristic. However, it has not all the programmable features since it can not implement an elaboration unit based on a hardware-paradigm. In terms of area occupation, microcontrollers have a medium impact and it can be reduced designing easier microprocessor architectures. As shown in the previous section, microcontroller latency and efficiency performances are not good, especially in the smart power arena. As already shown both efficiency and latency are related to the number of assembly instructions n_{insn}. In order to improve the microcontroller area occupation, researchers have been developing easier microprocessor architectures, reducing, for instance, the number of pipeline stages. In this way, the area occupation is strongly reduced with a straightforward negative impact on both latency and energy efficiency, since the processor needs both more clock cycles and perhaps more assembly instructions to execute the task.

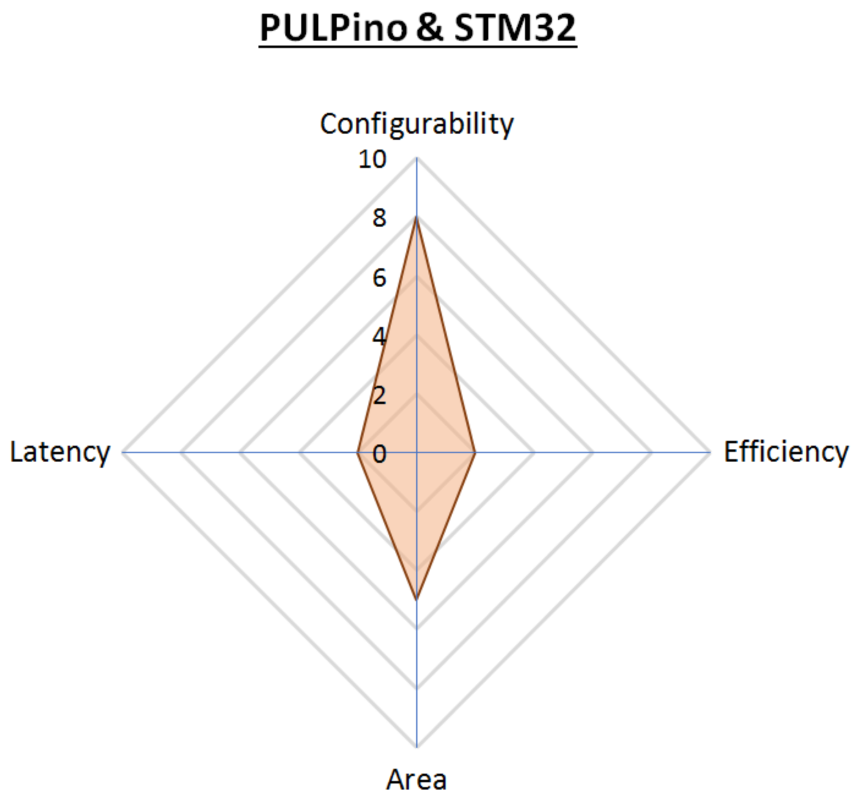


Figure 5.12: Microcontroller PULPino and STM32 implementation features.

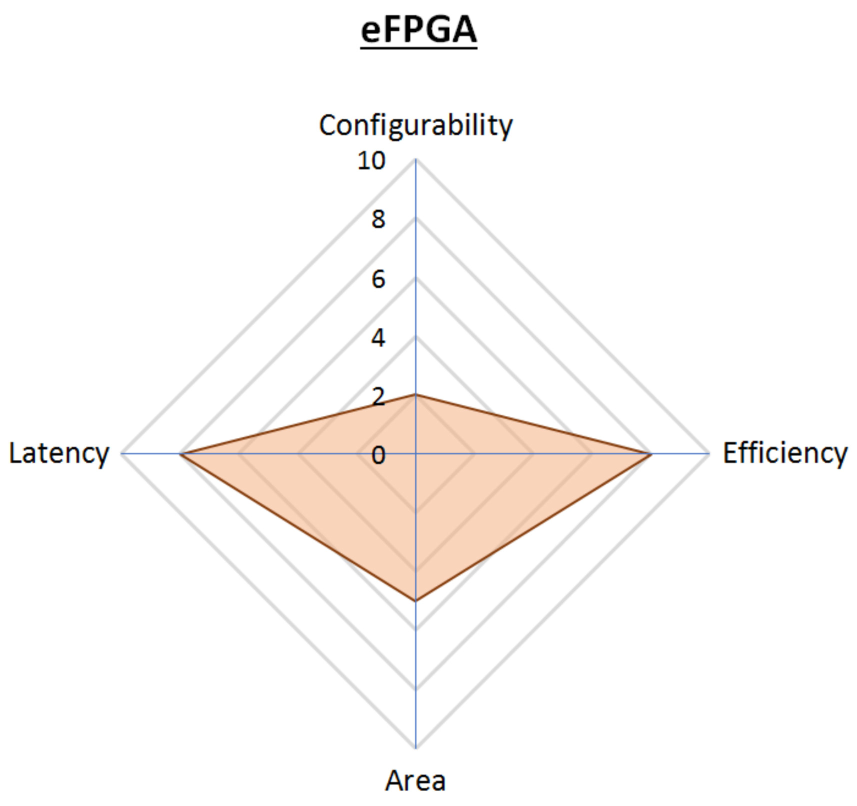


Figure 5.13: Proposed eFPGA implementation features.

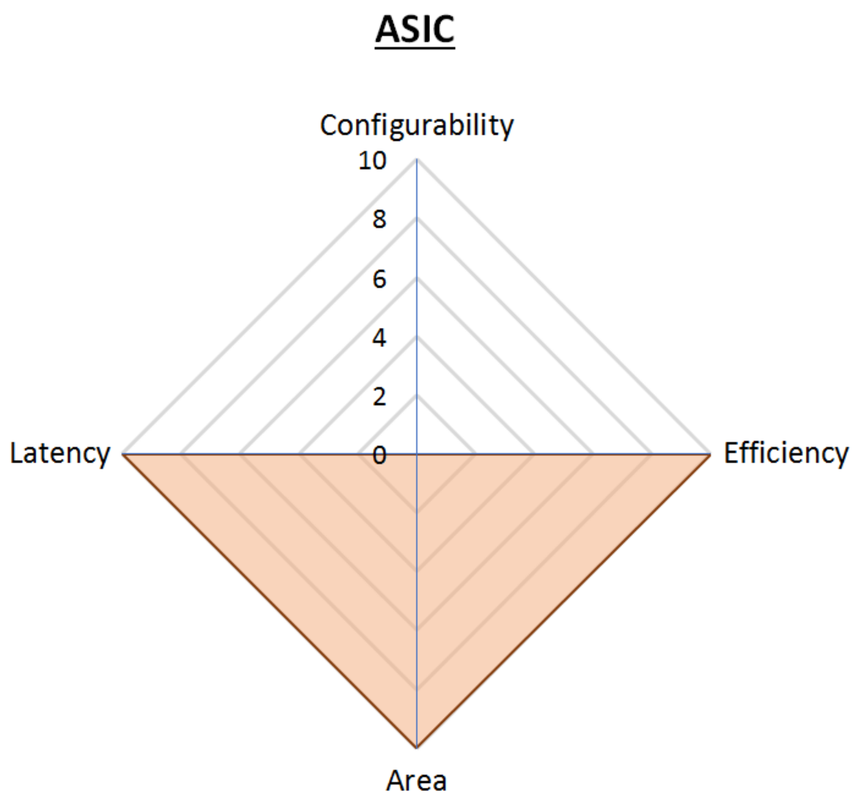


Figure 5.14: ASIC implementation features.

eFPGA Implementation

The proposed eFPGA has a very small dimension, around 1-kilo equivalent gates and its features are reported in Figure 5.13. For this reason, its programmable capability is very limited, especially compared to both a general-purpose microprocessor solution and a standalone or embedded FPGA tailored for high-density computing. Due to its synthesizability characteristic, the area occupation is not negligible and it is almost comparable to the PULPino core area occupation. Since it is a hardware-programmable device, it can reach excellent performances in terms of both latency and energy efficiency better than processor, particularly in smart power applications. The reconfigurability slightly affects the efficiency, since the device has a circuit overhead in order to guarantee the configurability. The configurability affects also the device reachable latency in terms of time since the eFPGA critical path is slower than an ASIC implementation due to the presence of additional circuits for the reconfigurability.

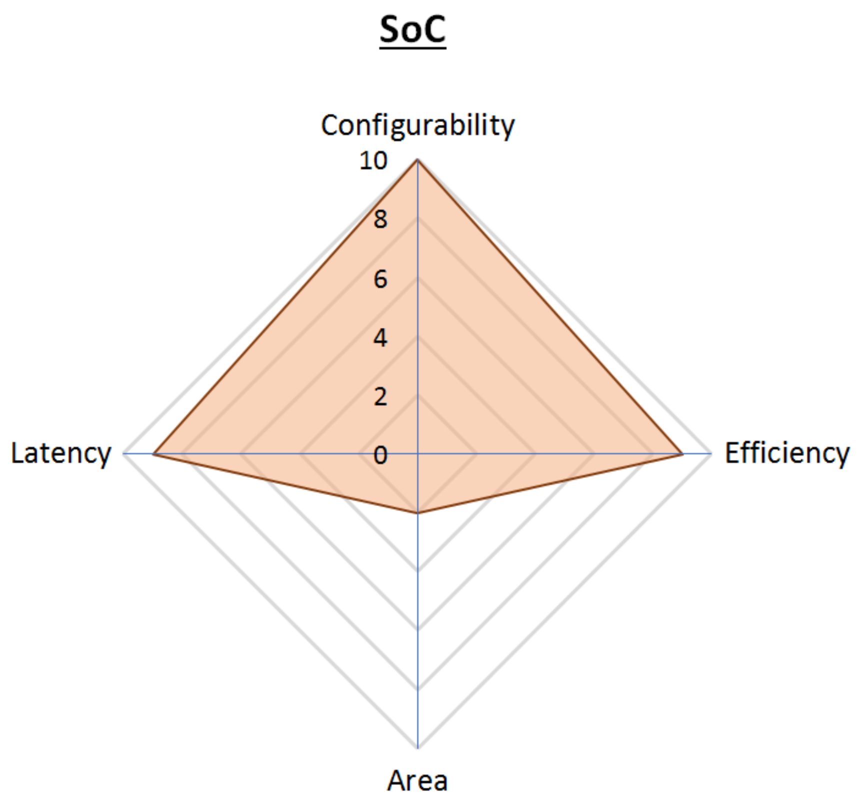


Figure 5.15: Proposed SoC implementation features.

ASIC Implementation

As well known, ASIC is the best solution in terms of area occupation, energy efficiency and latency as reported in Figure 5.14. The reason is that ASICs are realized optimizing the circuits required to implement a specific digital circuit. In this way, ASICs totally lose any reconfigurability features and this makes ASICs appealing just for certain kinds of markets.

Proposed SoC Implementation

The proposed system-on-chip composed of PULPino microcontroller and our proposed eFPGA as a reconfigurable peripheral combines the features of the single solutions at the expense of the area occupation, as reported on the chart of Figure 5.15. The combination of both devices increase the reconfigurability of the system-on-chip, and therefore, its reusability in different application scenarios. In this way, the system is provided with both software-programmable and hardware-programmable features and hence, the system application designer can partition the task using the most efficient solution. PULPino and eFPGA union has a straightforward benefit in terms of energy efficiency thanks to a careful task partitioning, which can impact directly the latency performance of the system-on-chip. As summarized in Table 5.2 and Figure 5.10, one can use the eFPGA to manage more efficiently smart power applications and guaranteeing better latency performance. In the meanwhile, the processor can handle other kinds of computations that fit better its structure such as operations mapped in its instruction set architecture. Somehow, the proposed system-on-chip represents the opposite of an ASIC implementation as visible in Figure 5.15 and in Figure 5.14. Both of them achieve excellent performances in terms of latency and energy efficiency targeting respectively configurability (the proposed SoC) and the best area occupation (ASIC implementation).

Chapter 6

Wake Up Radio

Most of the material reported in this chapter is reused from [69] (©2018 IEEE), in agreement with IEEE copyright policy on theses and dissertations.

This chapter presents a technique to reduce the power consumption of the communication interface of Figure 2.4. In the previous chapters we have analyzed techniques, in terms of elaboration unit architecture to optimize the computational unit performance. We consider now the communication interface in order to reduce its power consumption making the proposed SoC of Figure 2.4 appealing for the IoT scenario. On top of the chosen radio communication protocol, we have developed a dedicated circuit that reduces the power consumption of the communication interface during the node listening phase. Hence, it is presented a wake up radio integrated circuit developed in 90-nm BCD technology by STMicroelectronics. The reason which induces the wake up radio usage and the proposed wake up radio architecture are introduced . This chapter is the result of the collaboration with other researchers and Ph.D. students in our laboratory. The contribution of this thesis is the design of some wake up radio blocks through a full custom design methodology.

6.1 ULP Communication Techniques

In the IoT scenario, the ultra low-power design techniques are the key enabling technologies for its diffusion. Coming back to Figure 2.1, ULP technologies should be applied to every end-node blocks. In this chapter is proposed a ULP technique for the communication side of the IoT node in an integrated circuit solution. Since the wireless communication unit is one of the subsystems with the highest power consumption, the main idea is to reduce the communication activities [70]. A well-known communication activity reduction technique is Duty Cycling (DC) while wake up radio is one of the emerging technologies.

- Duty Cycling consists in periodically turning off the wireless transceiver based on either fixed or variable timing schedule. In this way, the energy saving is considerable because the listening phase power consumption (without transmitting any data) is substantially reduced, but on the other hand, the communication latency is notable. In addition to the latency increase, DC does

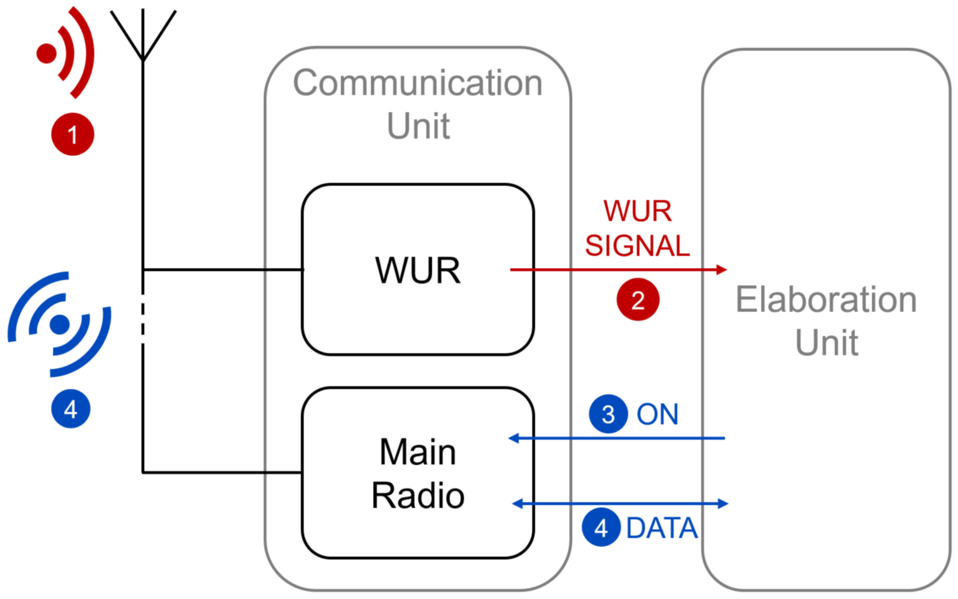


Figure 6.1: System architecture with WUR.

not eliminate potential useless listening procedures resulting in wasted power consumption [71].

- Wake up radio change the communication unit architecture in terms of both structure and protocol. WUR is an additional always-on radio which duty is to always listen to the communication channel, while the main radio is switched off and the elaboration unit can be switched in sleep-mode in order to minimize the power consumption. In this way, wake up radio must be an ultra low power radio capable only to detect the wake up signal (using eventually some kind of addressing technique) on the communication channel and then generate the wake up signal to the elaboration unit. This approach enables asynchronous communication architectures improving system energy efficiency since both the main radio and elaboration unit work only when it is really necessary. WUR can share the physical channel with the main radio and it typically uses On-Off Keying (OOK) modulation since the demodulator (inside the WUR) requires low power consumption [72]. Since long-range communication protocols are becoming popular and they use a carrier frequency of 868 MHz, WURs use this frequency too. Wake up radio are realized in both discrete component and integrated circuit solutions [69]. Based on the power consumption there exist different kind of WUR for different applications, such as:

  – fully-passive WURs could work without any supply reaching sensitivities in the order of -25 dBm. The limited sensitivities restrict applications to short-range communications (few centimeters) such as implantable devices.

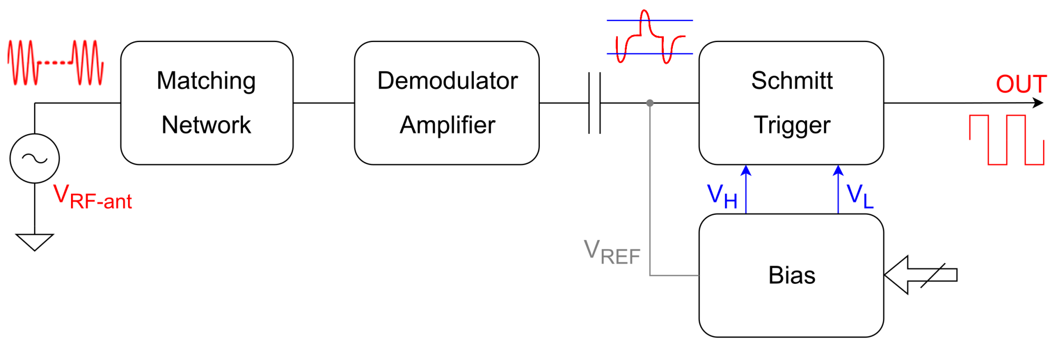


Figure 6.2: Wake up radio architecture.

- Nano-watts WURs are usually suitable for medium-range communications (at most 100 m) for instance in indoor applications.
- Micro-watts WURs are useful for long-range communications (km) for outdoor applications since they have better sensitivity in order of -100 dBm.

The new system architecture provided a wake up radio is reported in Figure 6.1. In this case, in the communication unit in addition to the main radio, there is a further block which is the WUR. Starting from a sleep-mode condition of the end-node where only the WUR is active, the working procedure is the following reported in Figure 6.1:

1. WUR is always listening to the channel and it receives a wake up RF signal.
2. If the address in the message detected from the WUR is correct, WUR generates a wake up signal for the elaboration unit.
3. Then the elaboration unit switches on the main radio for the communication.
4. The elaboration unit handles all the data transfers.

After the communication procedure, the system can come in sleep mode, turning off both the main radio and the elaboration unit.

6.2 Wake Up Radio Architecture

The proposed solution is an active integrated circuit nano-watts wake up radio, working with OOK modulation at 868 MHz, with a bitrate of 1 kbit/s, and it is implemented in 90 nm BCD technology. The wake up radio architecture is reported in Figure 6.2 and it is composed of a matching network, a demodulator and amplifier circuit, a Schmitt trigger and a bias circuit to generate the reference voltages. The

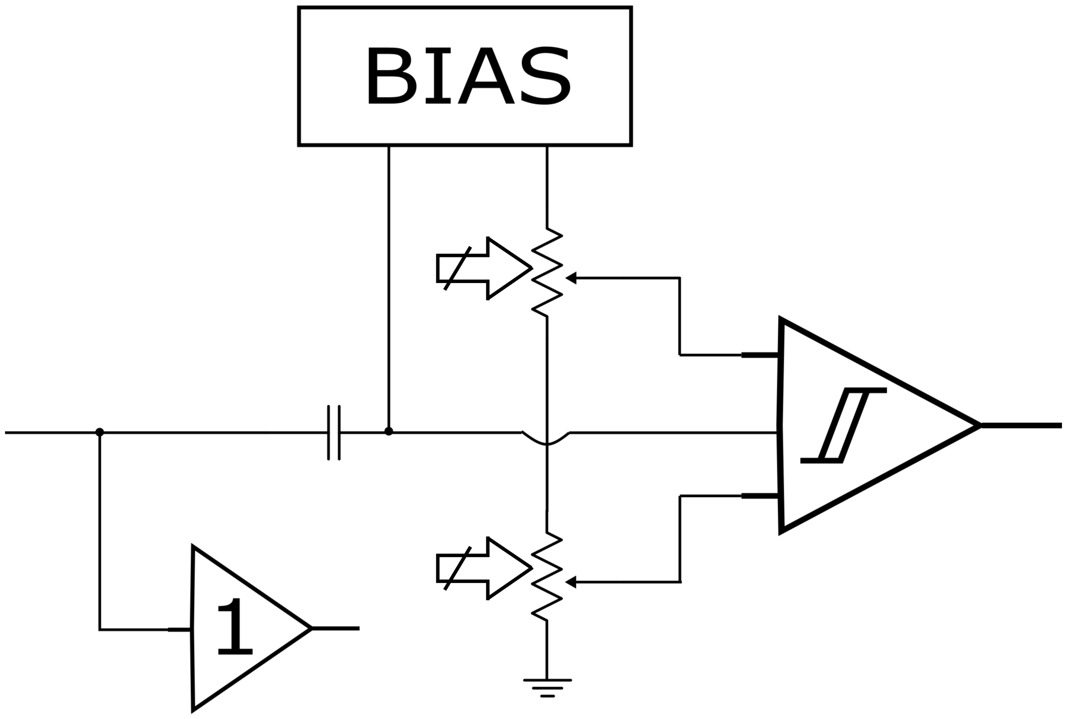


Figure 6.3: Bias circuit function.

high-Q off-chip matching network is used to match the antenna impedance and provide some additional amplification. The demodulator and amplifier detects the envelope of the OOK modulated signal and amplifies it. The demodulator and amplifier, with a band-pass response, generates a pulse at each OOK bit transition. Then, the generated pulses through a capacitive coupling are digitalized via a Schmitt trigger as reported in Figure 6.2. A bias circuit provides both the DC biasing voltage V_{REF} for the Schmitt trigger input and the thresholds V_H and V_L for the Schmitt trigger hysteresis. The DC voltage V_{REF} is between the two threshold voltage. The hysteresis thresholds are digitally programmable by the user as schematized in Figure 6.3 where it is also present a standard analog buffer in order to monitor the amplifier output during the testing phase. The thresholds are generated from a kind of voltage divider based on resistors. These resistors are on-chip digital potentiometers with 5-bit inputs, which are decoded in 32 resistance steps. Figure 6.4 shows the structure of both digital potentiometers to generate the two threshold voltage V_H and V_L. In Figure 6.4 the voltage V_x is a copy of the DC bias voltage V_{REF} applied to the Schmitt trigger input. When the Schmitt trigger input crosses a threshold (based on the Schmitt trigger internal state), the output is thus inverted. In the following section is presented the architecture of the demodulator-amplifier designed by a colleague. Then, is explained the actual contribution of the thesis which corresponds to both Schmitt trigger design and threshold generation dedicated circuit design.

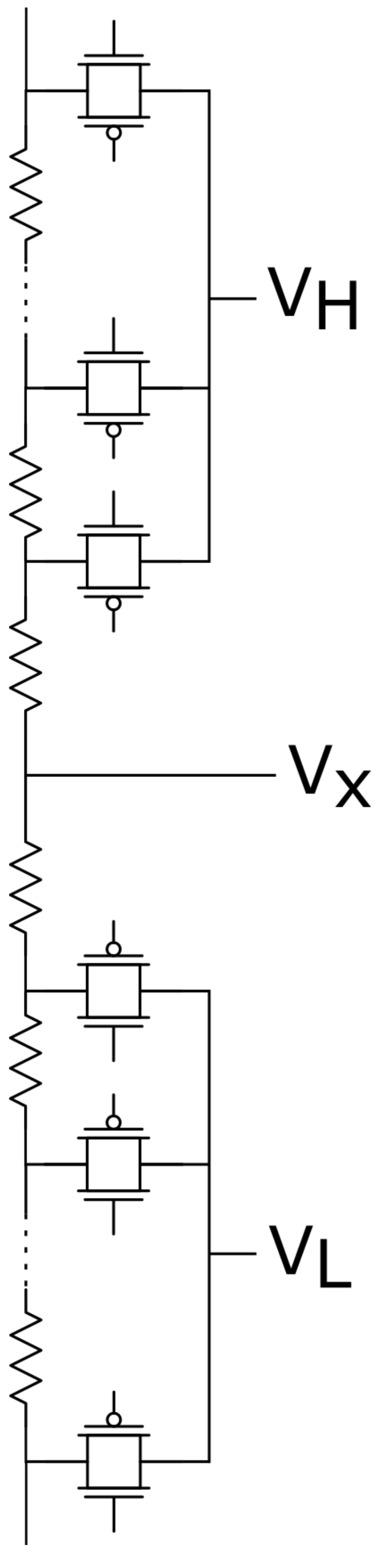


Figure 6.4: Digital potentiometer structures used to generate the Schmitt Trigger voltage references.

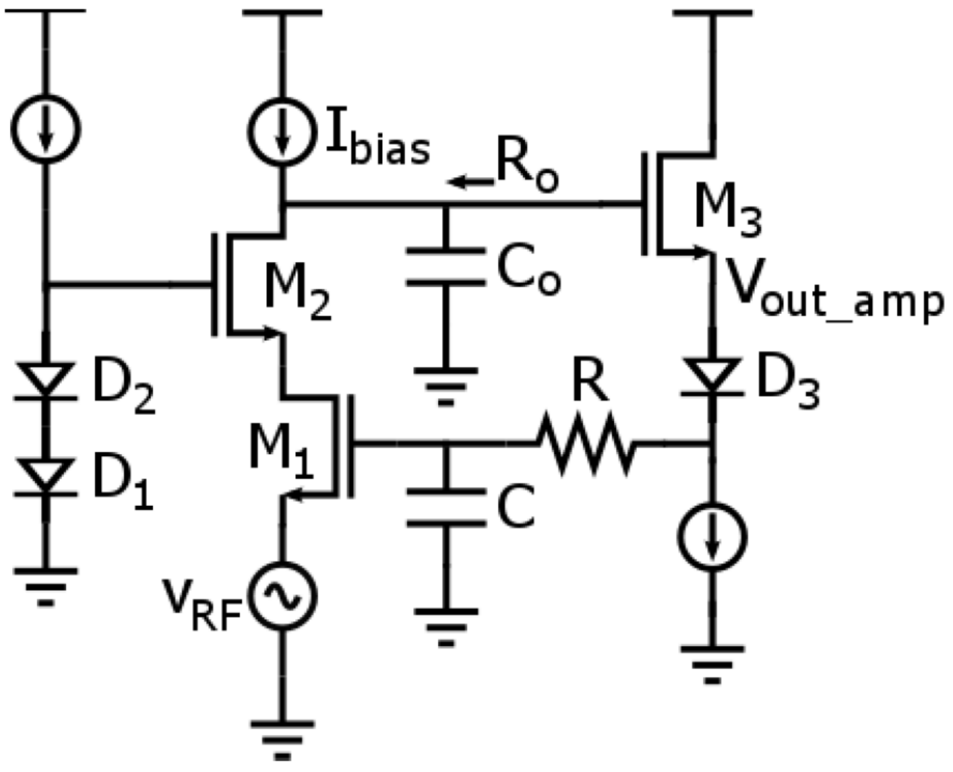


Figure 6.5: Demodulator-amplifier architecture. Picture reused from [69].

6.2.1 Demodulator-Amplifier Architecture

Since the ultra low-power feature is the main aim, all transistors in the signal path work in subthreshold region and hence, the MOSFET current equation is:

$$I_D = \frac{W}{L} I_0 \cdot e^{\frac{V_{GS}-V_{th}}{nV_T}} \left(1 - e^{-\frac{V_{DS}}{V_T}}\right) \tag{6.1}$$

which can be approximated if V_{DS} is enough greater than the thermal voltage V_T as:

$$I_D \simeq \frac{W}{L} I_0 \cdot e^{\frac{V_{GS}-V_{th}}{nV_T}} \tag{6.2}$$

where V_{th} is the MOSFET threshold voltage, n is a non-ideality factor and I_0 is a process parameter. The OOK modulated signal is stated by:

$$v_{RF}(t) = V_{OOK}(t) \cdot \cos(\omega t) \tag{6.3}$$

where $V_{OOK}(t)$ is the modulating signal while $\cos(\omega t)$ is the RF carrier.

The demodulator-amplifier architecture presented in [69] is reported in Figure 6.5 and it consists of a non-inverting cascode stage M1 and M2, and a voltage follower M3. The voltage follower provides negative feedback to input transistor M1 gate resulting in a self-biasing architecture. The circuit input port is on the M1 source reaching very high input impedance (much more than the antenna impedance). The capacitor C of Figure 6.5 is an off-chip high-quality capacitor due to its big value (magnitude order of pF) which make it unfeasible in integrated circuit solution especially in BCD technology. The envelope detection process is based on the non-linear relation between current I_D and V_{GS} of a MOSFET as stated in equation (6.2). As shown in Figure 6.5, the RF input is provided to the MOSFET source terminal, and considering the RF signal grounded by the capacitance C, the M1 current can be written as:

$$I_{D1}(t) = I_{BIAS} \cdot e^{-\frac{v_{RF}(t)}{nV_T}} \tag{6.4}$$

Through the Taylor approximation of the exponential function, the current of M1 can be expressed as:

$$I_{D1}(t) = I_{BIAS} \cdot e^{-\frac{v_{RF}(t)}{nV_T}} \simeq I_{BIAS}\left[1 - \frac{v_{RF}(t)}{nV_T} + \frac{1}{2}\left(\frac{v_{RF}(t)}{nV_T}\right)^2\right] \tag{6.5}$$

Substituting equation (6.3) into the previous equation, and reminding trigonometric identity

$$\cos^2(x) = \frac{1 + \cos(2x)}{2} \tag{6.6}$$

the M2 current becomes:

$$I_{D2}(t) = I_{BIAS}\left[1 + \frac{V_{OOK}^2(t)}{4n^2V_T^2}\right] \tag{6.7}$$

considering the high-frequency components grounded by the parasitic capacitances at the M2 source. Therefore, M2 current contains only low-frequency components. The M2 current can also be expressed highlighting both its DC component (I_{BIAS}) and its signal component which is envelope-dependent:

$$I_{D2}(t) = I_{BIAS}\left[1 + \frac{V_{OOK}^2(t)}{4n^2V_T^2}\right] = I_{BIAS} + \Delta I(t) \tag{6.8}$$

The I_{D2} envelope-related part $\Delta I(t)$ can be considered as resulted of a low-frequency input voltage source $v_{IN}(t)$

$$\Delta I(t) = v_{IN}(t) \cdot g_{m1} = I_{BIAS}\frac{V_{OOK}^2(t)}{4n^2V_T^2} \tag{6.9}$$

Since M1 is working in weak inversion, its transconductance is $g_{m1} = I_{BIAS}/nV_T$, and hence, the equivalent low-frequency input voltage source is:

$$v_{IN}(t) = \frac{\Delta I(t)}{g_{m1}} = \frac{V_{OOK}^2(t)}{4nV_T} \tag{6.10}$$

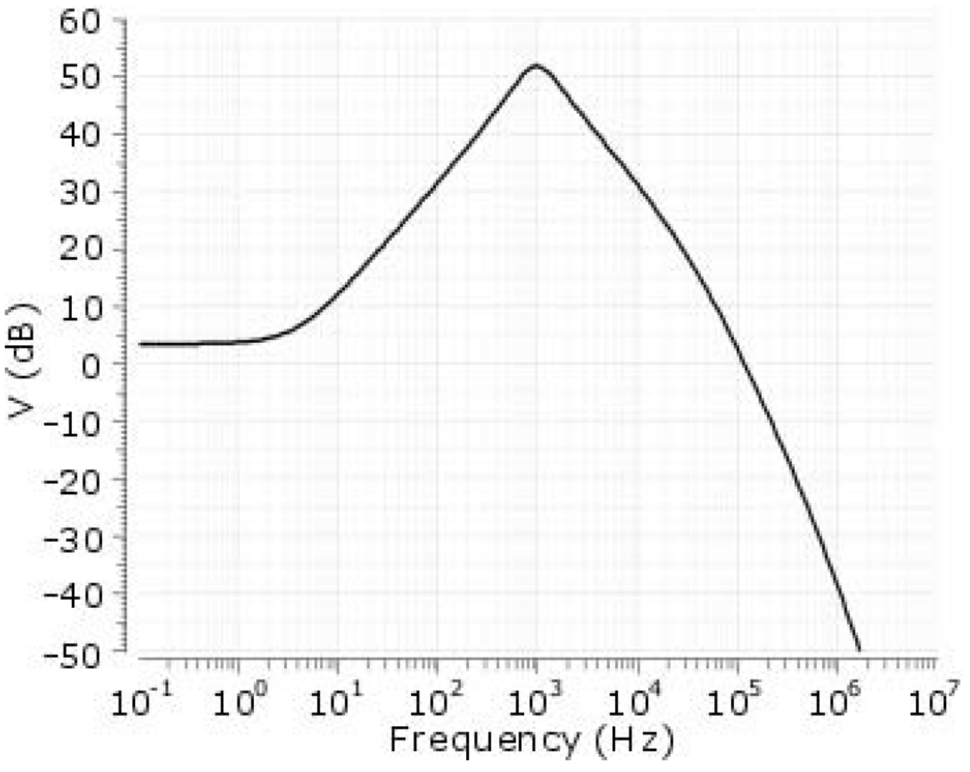


Figure 6.6: Frequency response of the proposed circuit. Picture reused from [69].

Assuming unity gain for the source follower, the circuit response to $v_{IN}(t)$ is approximately:

$$A(s) = \frac{v_{OUT\_AMP}}{v_{IN}}(s) = \frac{1 + sR^*C}{\left(1 + s\frac{R^*C}{g_{m1}R_o}\right)(1 + sR_oC_o)} \tag{6.11}$$

where $R^* = R + R_{OUT}^{FOLL}$, with R_{OUT}^{FOLL} the output resistance of the source follower. Both R_o and C_o are respectively the parasitic resistance and capacitance seen at the M2 drain. The circuit response of equation (6.11) has a band-pass response reported in Figure 6.6. From equation (6.11) one can see that the low-frequency gain is limited due to the negative feedback. Increasing the frequency above $(2\pi R^*C)^{-1}$, the capacitance C progressively grounds the feedback increasing the gain to the peak value $g_{m1}R_o$. While the second pole is due to the parasitic capacitance seen at M2 drain.

The band-pass frequency response allows suppressing the flicker noise effect using suitable transistor sizes especially for M1 and the current generator I_{BIAS} which are the main noise contributors.

Considering a signal-to-noise voltage ration (SNR) of 5, with a current power of 8 nA, the amplifier can reach gain between the output voltage and the envelope V_{OOK} of 5, resulting in an estimated sensitivity of 5 mV. The achieved gain allows the Schmitt trigger offset to not strongly impact the overall system performance.

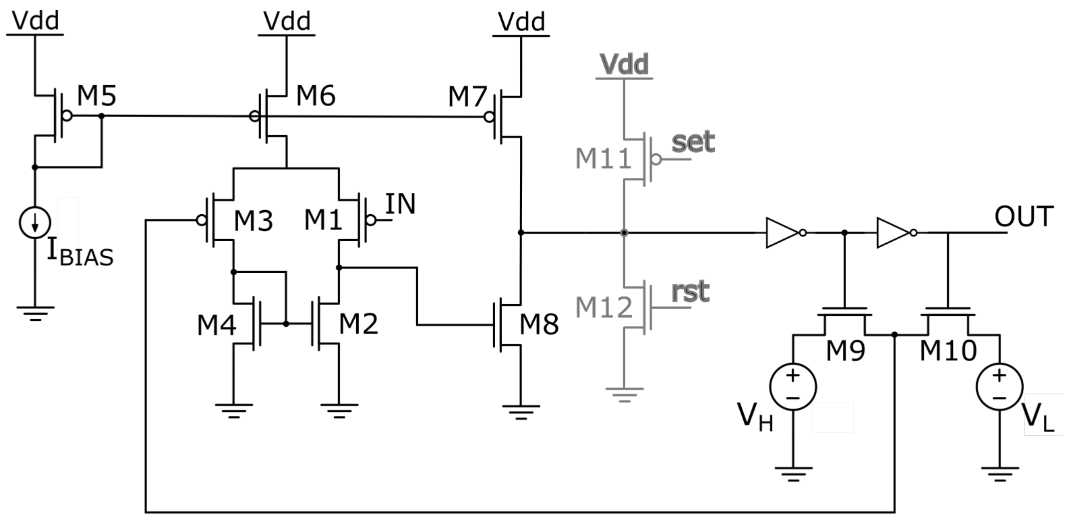


Figure 6.7: Schmitt trigger schematic.

6.3 Schmitt Trigger Design

The Schmitt trigger is based on a two-stage Operational Transconductance Amplifier (OTA) with positive feedback as reported in Figure 6.7. The trigger aims to detect the pulses generated by the demodulator-amplifier and generate a digital signal when the pulses cross the trigger threshold. The right hysteresis threshold is provided by the digital potentiometers, and selected, as shown in Figure 6.7, by either M9 or M10 based on the internal state of the trigger. This link is the positive feedback of the trigger. Both M11 and M12 are used to respectively set or reset the trigger internal state, in order to define the initial operating condition of the circuit. The main constraints for the Schmitt trigger design are:

- ultra low-power consumption to avoid a degradation on the power performance;
- limited equivalent input offset in order to not deteriorate the overall system performance.

The first constraint is achieved defining the current bias I_{BIAS} of 1 nA and sizing M5, M6 and M7 with the same aspect ration in order to have the same current of 1 nA in each branch, and hence, the power consumption of the Schmitt trigger is 3.6 nW. Regarding the input offset, we define the constraint as $3 \cdot \sigma < 10$ mV. Following the theory proposed in [73], where a fundamental analysis of the statistical variation of device parameter P on a wafer predicts a Gaussian distribution with variance

$$\sigma^2(\Delta P) = \frac{A_P^2}{WL} + S_P^2 \cdot D_x^2 \tag{6.12}$$

where A_P is an area proportionality constant for the parameter P, S_P describes variation of parameter P with the spacing D_x and both W and L are the device dimensions.

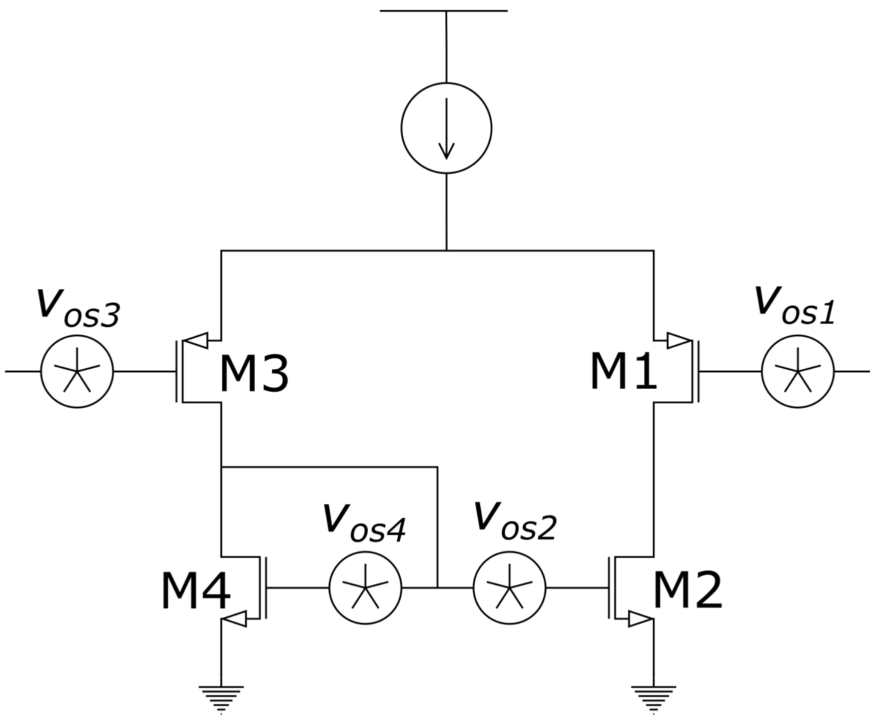


Figure 6.8: Equivalent offset model of the Schmitt trigger differential pair.

In the modern microelectronic technology, the offset is almost due to non-idealities on the transistor threshold mainly dominated by the area-dependant coefficient

$$\overline{v}_{OS}^{2} = \sigma^{2}\left(V_{th}\right) \simeq \frac{A_{V_{th}}^{2}}{WL} \tag{6.13}$$

The aim is to evaluate the transistor effect to the trigger input. Considering the trigger differential pair, reported in Figure 6.8, and thanks to its symmetry we evaluate only M1 and M2 through their equivalent offset voltage v_{OSeq1} and v_{OSeq2}:

$$\overline{v}_{OSeq}^{2} = \overline{v}_{OS1eq}^{2} + \overline{v}_{OS2eq}^{2} \tag{6.14}$$

Since M1 effect is already at the input $v_{OSeq1} = v_{OS1}$, we have to find out the expression of M2 equivalent offset

$$\overline{v}_{OSeq}^{2} = \overline{v}_{OS1}^{2} + \overline{v}_{OS2eq}^{2} \tag{6.15}$$

Therefore, one has to evaluate the effect of v_{OS2} at M1 gate

$$\overline{v}_{OS2eq}^{2} = \overline{v}_{gs1}^{2} = \frac{\overline{i}_{D1}^{2}}{g_{m1}^{2}} \tag{6.16}$$

and since $i_{D1} = i_{D2}$

$$\overline{i}_{D1}^{2} = \overline{i}_{D2}^{2} = \overline{i}_{D}^{2} = g_{m2}^{2} \cdot \overline{v}_{OS2}^{2} \tag{6.17}$$

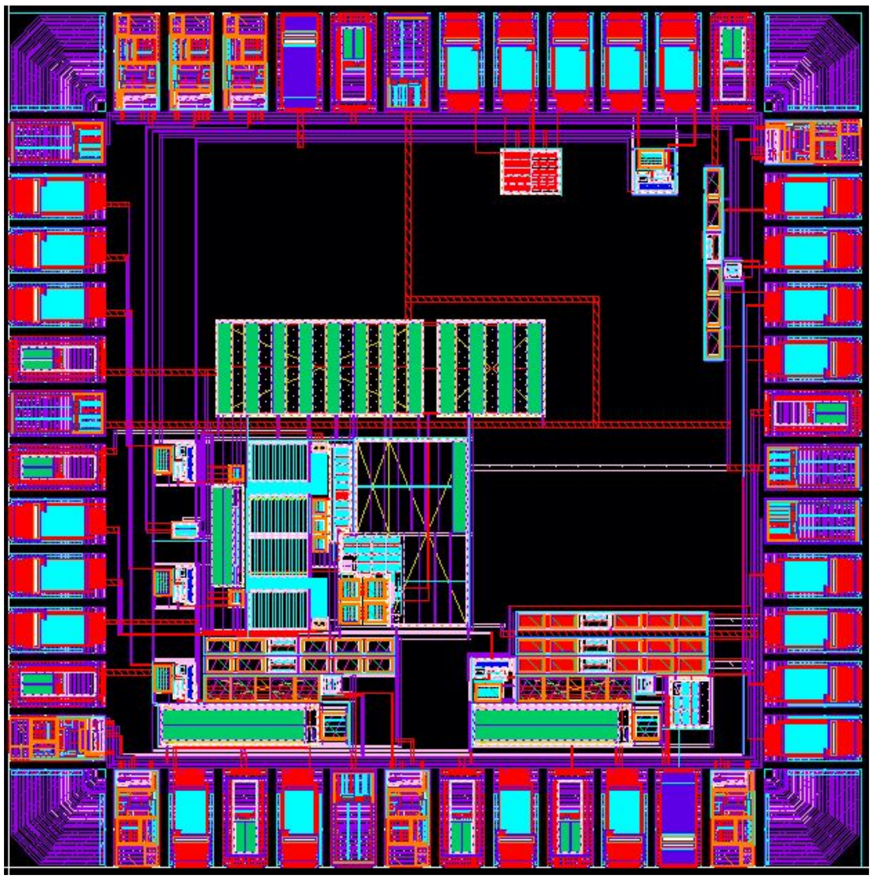


Figure 6.9: Wake up radio test chip layout.

v_{OS2eq} becomes:

$$\overline{v}^2_{OS2eq} = \overline{v}^2_{gs1} = \frac{\overline{i}^2_D}{g^2_{m1}} = \left(\frac{g_{m2}}{g_{m1}}\right)^2 \cdot \overline{v}^2_{OS2} \tag{6.18}$$

reminding the transistor transconductance in weak inversion

$$g_m = \frac{I_D}{nV_T} \tag{6.19}$$

and since $i_{D1} = i_{D2}$ the corresponding transconductances are equal $g_{m1} = g_{m2} = g_m$, and hence, $\overline{v}^2_{OS2eq} = \overline{v}^2_{OS2}$. Finally, the equivalent offset is simply:

$$\overline{v}^2_{OSeq} = \overline{v}^2_{OS1} + \overline{v}^2_{OS2} \tag{6.20}$$

Considering equal the offset contribution of both M1 and M2, which should be $3 \cdot \sigma_{TOT} < 10$ mV, it is possible to size M1, M2 and M4 through equation (6.12). Choosing all transistors equal with $W = L = 2$ μm the resulting estimated equivalent offset is $3 \cdot \sigma_{TOT} = 7.2$ mV.

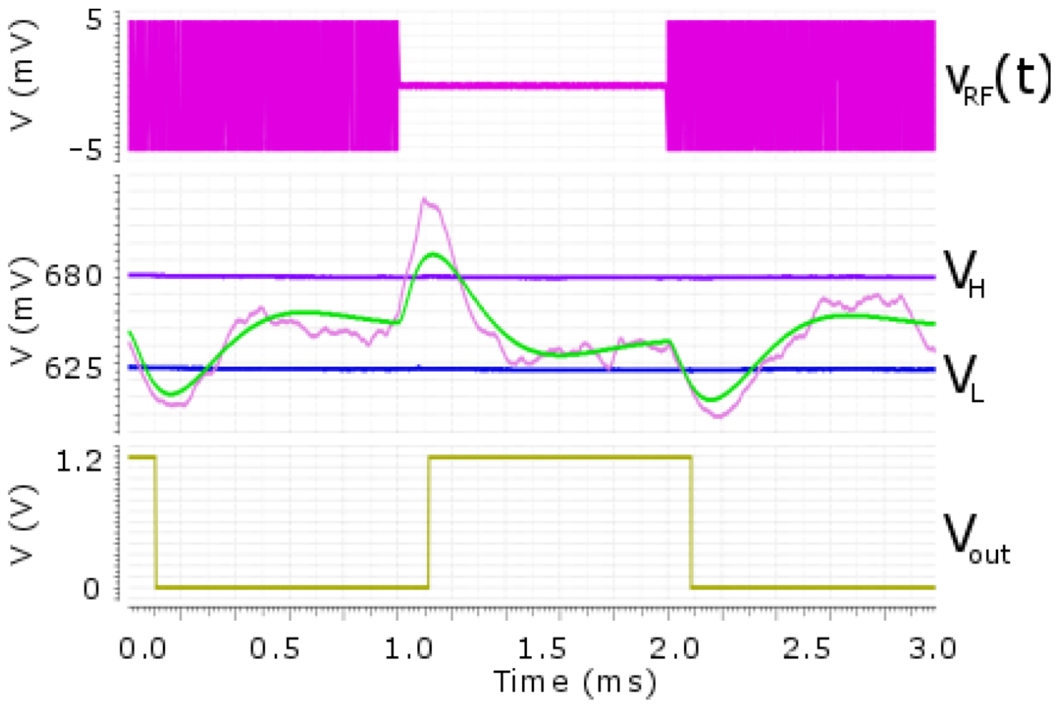


Figure 6.10: WUR system transient noise simulation. Picture reused from [69].

6.4 Results and Comparison

Figure 6.9 reports the layout of a designed wake up radio test chip in 90 nm BCD technology of STMicroelectronics. The area is strongly dominated by passive components. The wake up radio works with OOK modulation at 868 MHz, with a bitrate of 1 kbit/s, consuming only 11 nA (8 nA the demodulator-amplifier and 3 nA the Schmitt trigger).

According to the post-layout simulation, the demodulator-amplifier can provide amplification of 7 of the modulating signal $V_{OOK}(t)$. The estimate demodulator-amplifier input sensitivity at room temperature is 5 mV, assuming a signal-to-noise voltage ratio of 5, and hence, adding the gain of the matching network (that can reach 18 dB), the overall sensitivity becomes -54 dBm. Figure 6.10 depicts the results for transient noise simulation of the overall wake up radio system. The first signal of the picture is the modulated RF signal $v_{RF}(t)$ with a 1-0-1 commutation of the modulating signal $V_{OOK}(t)$ (reminding the equation (6.3)) with an amplitude of 5 mV. In the second chart from the top of Figure 6.10, is reported the demodulator-amplifier output without (green line) and with noise (violet line), and both the trigger threshold V_H and V_L. The last trace of Figure 6.10 is the output of the Schmitt trigger. At each crossing of the voltage threshold of the demodulator-amplifier output, trigger inverts its output. Therefore, it is verified the correct behavior of the overall wake up radio system.

Table 6.1: Wake up radio performance comparison. Reused from [69].

	Proposed WUR	IC [74]	IC [75]	Discrete [72]
Carrier freq.	868 MHz	114 MHz	915 MHz	868 MHz
Bitrate	1 kbit/s	0.3 kbit/s	100 kbit/s	1–20 kbit/s
Sensitivity at 27 °C	-54 dBm	-69 dBm	-41 dBm	-55 dBm
Supply voltage	1.2 V	0.4 V	1.2 V	1.2 V
Power consumption	13 nW	4 nW	98 nW	600 nW
Addressing	✗	✓	✗	✓
Technology	90 nm BCD	180 nm SOI-CMOS	130 nm CMOS	n.a.

In Table 6.1 is reported a performance comparison between our proposed solution and the integrated circuits presented in both [74] and [75], and the discrete solution presented in [72]. The IC solution proposed in [74] has the best performance in terms of power consumption, however, it is realized in a Silicon-On-Insulator CMOS technology, which allows one to efficiently control the transistor body in order to exploit dynamic threshold transistors. The integrated circuit presented in [74] has also the best sensitivity as shown in Table 6.1, but it works with a bitrate of 0.3 kbit/s, which corresponds to a 2.5 dBm sensitivity deterioration at 1 kbit/s due to a larger noise bandwidth. Besides, in [74] the carrier frequency is 114 MHz which allows one to design a matching network reaching around 25 dB gain instead of 18 dB working at 868 MHz or 12 dB in [75]. On the other hand, the solution presented in [75] works with 100 kbit/s of bitrate, which corresponds to a 10 dBm sensitivity improvement at 1 kbit/s. As well as [74], [75] exploits technology available features to increase transistor transconductance through dynamic threshold configurations.

Our proposed solution has performance comparable to both state-of-the-art integrated circuits and discrete component solutions. Our IC reaches a sensitivity similar to the one offered by the discrete circuit solution, but drastically reducing the power consumption.

Chapter 7

Conclusions

The Ph.D. research work tries to address techniques to improve performances of the modern internet-of-things end-nodes acting on both, the elaboration unit and the communication interface of the system-on-chip depicted in Figure 2.4.

The first thesis part proposes a combination of both microcontroller and soft core of embedded FPGA that increases the reconfigurability of the elaboration unit, providing both software and hardware programmable feature, useful in an always increasing resource demand scenario. The proposed heterogeneous system is fully-synthesizable, and hence, it is portable to different technology. The soft-core eFPGA has a small computational capability, and the system is designed to exploit the eFPGA to carry out smart power tasks while the microcontroller can handle other kinds of computations or can be switched in sleep mode. The proposed elaboration unit allows the system to increase its energy performance through an efficient task partitioning. As highlighted in Chapter 5, one can use the small embedded FPGA to perform smart power tasks in a very efficient way, with a straightforward drawback in terms of area occupation. The eFPGA usage allows the system-on-chip to improve also its latency performance since in this way, it can exploit the latency related to an hardware-programmable device, which is typically related just to the operating clock period.

The embedded FPGA exploitation just to implement simple finite state machines is an unconventional point of view to this kind of devices, usually employed for high computational load applications.

The proposed elaboration unit proves to be an interesting choice for a smart power system-on-chip, combining both the vast microcontroller capabilities and a hardware-programmable device which fits better smart power application requirements.

To the best of our knowledge the proposed heterogeneous system, which integrates a microcontroller and an embedded FPGA, is the first reconfigurable system-on-chip suitable for smart power internet-of-things applications.

On the communication interface side, wake up radio proves to be a promising and essential technology for the future internet-of-things end-nodes, changing both the communication paradigm and the electronic system architecture.

The proposed monolithic nano-watt wake up radio integrated circuit has performances comparable to the other integrated circuit solutions, and superior performances considering discrete component solutions. However, this encourages researchers to improve wake up radio performances in terms of power consumption, sensitivity and area occupation.

Appendix A

eFPGA Interconnection Network Details

Most of the material reported in this chapter is reused from [40].

Proprieties and results related to the proposed eFPGA interconnection network are reported in this appendix.

A.1 Multicast Radix-2 MSSN Proprierties

Banyan networks, as mentioned, are blocking also for unicast connections. Therefore a method to make the network RNB is the *Horizontal Cascading* (HC) of two baseline $\log_2 N$-networks as depicted in Figure A.1(a). Benes networks [76], [77] are an example of this approach. They are made connecting two baseline networks back-to-back. However, the *Horizontal Cascading* technique results in complex routing algorithms due to a large amount of reconfiguration activity [59].

An alternative to *Horizontal Cascading* is the *Vertical Stacking* (VS) technique introduced in [58] and shown in Figure A.1(b). According to the *Vertical Stacking* technique, the overall network consists of various copies (called planes) of a baseline $\log_2 N$ network. In this way, both a demultiplexer input stage and a multiplexer output stage (Figure A.1(b)) are necessary. Regarding multicast connectivity of *Vertically-Stacked* MSSN (VS-MSSN), the condition on the number of planes p necessary to obtain RNB features is:

$$p \geq 2^{\lfloor \frac{n}{2} \rfloor} \tag{A.1}$$

where n is the number of stages of the $N \times N$ baseline $\log_2 N$ network, and thus:

$$n = \log_2 N. \tag{A.2}$$

[78] presents a combination of HC and VS obtained by adding x extra stages to each baseline plane and the RNB condition for unicast traffic is:

$$p \geq 2^{\lfloor \frac{n-x}{2} \rfloor}. \tag{A.3}$$

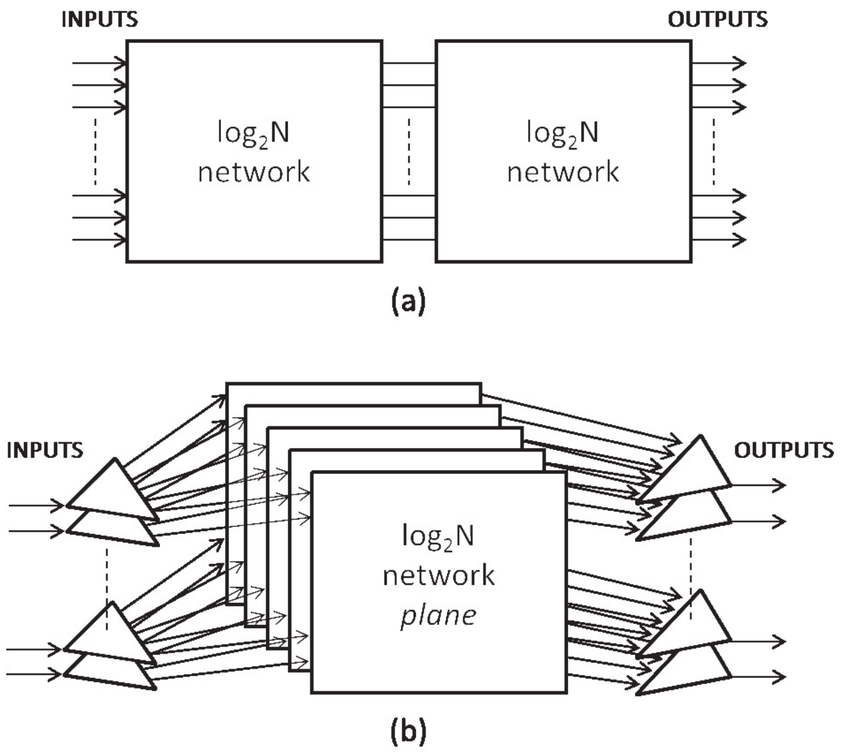


Figure A.1: Example of construction of Multi-$\log_2 N$ networks: with horizontal cascading (a) and vertical stacking (b) technique [40].

This technique can improve network area occupation, since the addition of x horizontal stages (lower than n) allows one to save one (or more) whole planes.

[78] demonstrates equation (A.3) for unicast connection, while [79] generalized the properties defined by [59] for vertical stacking also for multicast connection. We have intensely tested the proposed interconnection network in one-to-many applications, without finding any unresolved conditions [40].

From a qualitative point of view, the equivalence between horizontal and vertical decomposition finds its foundation in the fact that they are different strategies to provide the same paths multiplicity in connecting a generic input-output pair under the condition defined by (A.3). Simultaneously, the combination of HC and VS open the way for exploring various trade-off between hardware complexity and latency [40].

The interconnection network architecture proposed in both [3] and [4], is an MSSN based on 2×2 switches and it features a Benes-like topology with butterfly connectivity as shown in Figure 4.8(a). Considering its topologically equivalent depicted in Figure 4.8(b), which is a $N \times N$ I/Os MSSN with:

- $N_S = N$ switch blocks per stage,
- $n_S = 1_{INstage} + 1_{OUTstage} + (2\log_2(N) - 1)_{MIDstage} = 1_{INstage} + 1_{OUTstage} + n + x$, where $n = \log_2 N$ and $x = \log_2(N) - 1$.

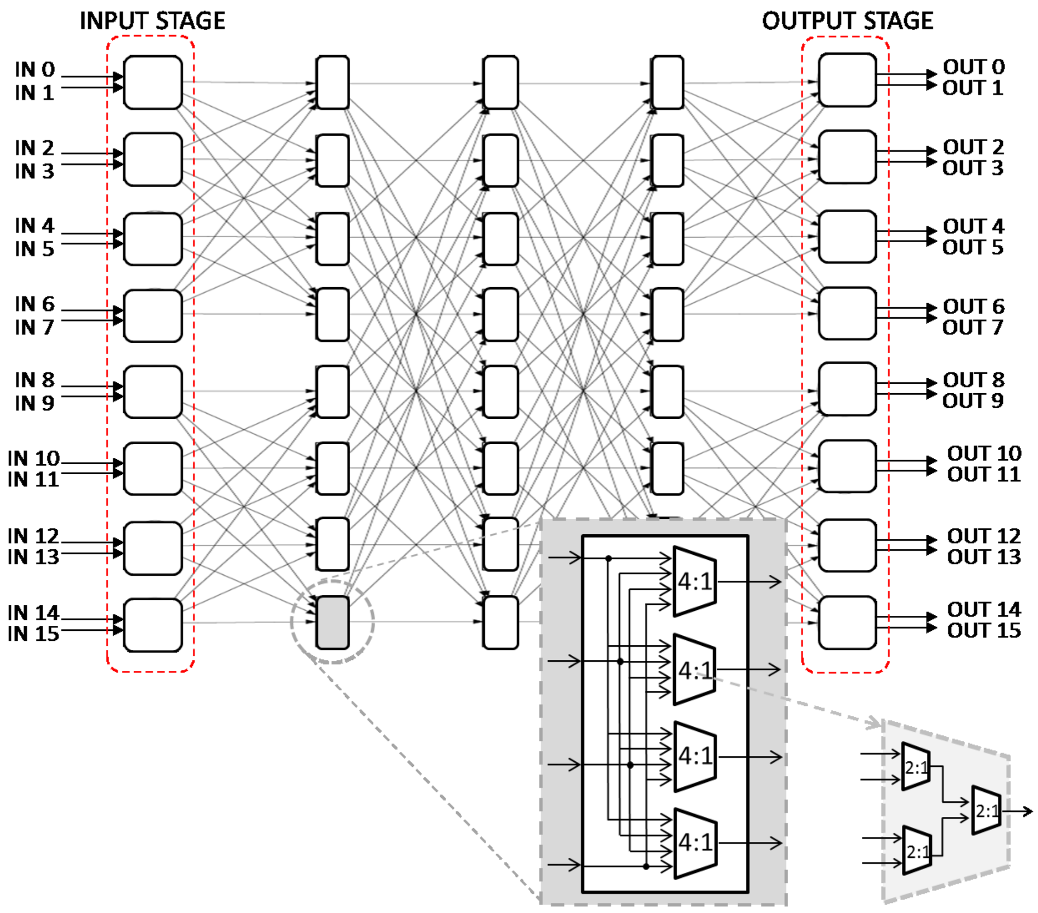


Figure A.2: An example of *radix-4* 16 × 16 *flat* MSSN. Picture reused from [40].

The network in Figure 4.8(b) has two planes p, p_1 and p_2 each made of a Banyan network of $n + x$ stages. Therefore, the equation (A.3) for rearrangeable non-blocking property becomes:

$$2 \geq 2^{\lfloor \frac{\log_2 N - (\log_2 N - 1)}{2} \rfloor} = 2^{\lfloor \frac{1}{2} \rfloor} = 1 \tag{A.4}$$

and is satisfied for any value of N with $p = 2$.

For the proposed interconnection network, it should be noticed that a lower number of extra stages x is necessary. With $p = 2$ and $n = \log_2 N$, from equation (A.3), only $x = \log_2(N) - 2$ extra stages are strictly required. The area-penalizing choice of $x = \log_2(N) - 1$ is made in the perspective to increase the symmetry which enables the introduction of enhanced switch elements.

A.2 Multicast Radix-k MSSN Proprierties

This appendix generalizes the results obtained in the Appendix A.1 to a generic *radix-k* of the network. Therefore, the same proposed MSSN architecture with a generic *radix-k* has now $k \times k$ switch elements in the middle stages, and again the *radix-k* Multi-Stage Switching Network is topologically equivalent to an MSSN with two planes, each made of an $N \times N$ *radix-k* Banyan network with additional stages. For instance Figure A.2 shows an $N \times N$ *radix-4* interconnection network with $N = 16$.

The number of stages n of the baseline Banyan interconnect is obtained generalizing the equation (A.2) for a generic *radix-k*, and hence:

$$n = \log_k N. \tag{A.5}$$

The additional extra stages necessary for the *Horizontal Cascading* x follows the definition of the *radix-2* case, and thus, $x = n - 1 = \log_k (N) - 1$. Consequently, the total number of stages n_S of the $N \times N$ $_{\text{radix-k}}$ MSSN is:

$$n_S = 1_{INstg} + 1_{OUTstg} + n + x = 1_{INstg} + 1_{OUTstg} + (2\log_k (N) - 1)_{MIDstg} \tag{A.6}$$

which has the form of the case of $k = 2$.

In [80] is generalized the equation (A.3) for a generic *radix-k* for one-to-one traffic, demonstrating that the number of vertical planes p necessary to obtain an RNB network is:

$$p \geq k^{\lfloor \frac{n-x}{2} \rfloor}. \tag{A.7}$$

From the previous definitions of the number of stages n and extra stages x, condition (A.7) becomes:

$$p \geq k^{\lfloor \frac{\log_k N - (\log_k N - 1)}{2} \rfloor} = k^{\lfloor \frac{1}{2} \rfloor} = 1. \tag{A.8}$$

Considering $p = 2$, equation (A.8) is always satisfied for any value of both number N of inputs/outputs and *radix-k*. In a generic *radix-k* network, all the stages have N_S switch elements, with $N_S = \frac{N}{k} \cdot p$. For instance, considering $p = 2$, a *radix-2* MSSN has $N_S = N$, while in a *radix-4* interconnection network $N_S = N/2$ as shown in Figure A.2. Regarding the multicast connectivity, we have carried out numerous routing tests without finding any critical situation for the routing scheme [40].

A.3 Hierarchical MSSN Performance

This section presents an evaluation of the improved MSSN performance through the *U-turn* switch elements. The analysis focuses on both the frequency improvement and the straightforward area overhead drawback due to the additional logic required by the *U-turn* switches. The following analysis compares *radix-2* and *radix-4* MSSN implemented in 65 nm CMOS LP technology of STMicroelectronics considering two different mixes of standard cell libraries, SVT-only for area optimization and full mix with H+S+LVT for high-speed optimization.

Table A.1: CMOS 65 nm 1024 × 1024 radix-2 MSSN post-synthesis results [40].

Implementation Results	Min Area			Max Speed		
	Flat	Half	Fully	Flat	Half	Fully
Area [mm^2]	0.20	0.34	0.40	0.80	0.89	1.01
Impl. Freq. [MHz]	200	200	180	480	445	395
Cells Mix		SVT			H+S+LVT	

Table A.2: CMOS 65 nm 1024 × 1024 radix-4 MSSN post-synthesis results [40].

Implementation Results	Min Area			Max Speed		
	Flat	Half	Fully	Flat	Half	Fully
Area [mm^2]	0.27	0.33	0.49	0.89	0.90	1.1
Impl. Freq. [MHz]	200	200	200	400	365	367
Cells Mix		SVT			H+S+LVT	

The *radix-2* and *radix-4* Multi-Stage Switching Networks considered have 1024 × 1024 inputs/outputs in three different organization:

- in a *flat* structure, the MSSN does not have any *U-turn* enhanced switch elements.
- In *fully-bypassed* architecture the interconnect has *U-turn* bypasses at each network stage. Referring to Figure 4.9, each switch at stage S, for S from 1 to $\log_2 N - 1$, is composed of source and target *U-switches*, while the switch in the central stage ($S = \log_2 N$) is a simple basic switch.
- In *half-bypassed* structure, the MSSN has *U-turn* switches only at the odd stages, whereas the switch in the central stage is again a simple basic switch.

Table A.1 and Table A.2 report post-synthesis results for all three MSSN architecture *flat*, *half*, and *fully*, respectively for *radix-2* and *radix-4*. The results are obtained considering two different implementations, optimized for a minimum area occupation or maximum speed. As clearly visible in both tables, the introduction of *U-turn* switches results in a straightforward area overhead. All the three interconnection network versions are implemented varying the target implementation frequency, where the implementation frequency is the *time-to-fly* between interconnect input and output, which is the delay necessary to cross all the MSSN switches.

A.3.1 MSSN Delay Model and Validation

In order to perform a trade-off area vs. speed analysis, it is useful to define an interconnection network delay model. In [3] is presented a post-synthesis application-aware analysis to evaluate the eFPGA performance exploiting the bypassed SEs. [3]

analyzed different benchmarks mapped in both *radix-2 flat* MSSN architecture and *radix-2 fully-bypassed* MSSN. The results show that the exploitation of bypassed SEs allows the eFPGA to increase its operating frequency by a factor (called *frequency gain*) from $\simeq 20\%$ to $\simeq 60\%$ depending on the capability to exploit bypass, and hence, on the considered application. In that analysis, it is supposed that each internal MSSN stage has the same delay, in both *flat* and *bypassed* architecture. Therefore, the delay associated to a single MSSN stage with n_S stages is approximated as:

$$STG_{dly} = \frac{T_{MSSN}}{n_S} \tag{A.9}$$

where T_{MSSN} is the delay of the whole MSSN, and thus, it is the implementation period in Tables A.1 and A.2.

Therefore, according to equation (A.9), the interconnection network input-to-output (IN2OUT) delay is proportional to the stage S, from 1 to $\log_2 N$, in which the *U-turn* connection is exploited. To better clarify, as reported in Figure 4.9, the stage S which exploits the bypass has the H_S group of switches. Hence, the IN2OUT delay is:

$$IN2OUT_{DELAY}(S) = \begin{cases} STG_{dly}[(2S-1)+2], & S = \log_2 N \\ STG_{dly}(2S+2), & \text{otherwise} \end{cases} \tag{A.10}$$

where the $2S$ factor is because the network has to be crossed back and forth, for instance, crossing the source and target switches in Figure 4.10. The $+2$ is due to both input and output stages. When $S = \log_2 N$, the interconnect does not exploit the *U-switches* and the overall network stages are crossed to connect an input to outputs. In this case, the IN2OUT delay degenerates into the implementation period T_{MSSN}, since $n_S = 2n+1$.

Figure A.3 shows the piecewise linear model (A.10) and a set of post-synthesis IN2OUT delays for the *radix-2 fully-bypassed* MSSN architecture, in order to prove the validity of the delay model proposed in equation (A.9) and (A.10). The considered post-synthesis delays are obtained for both maximum speed and minimum area implementations of the 1024×1024 *radix-2 fully-bypassed* MSSN of Table A.1, forcing the timing analysis tool to exploit the *U-turn* connections at different interconnect stages Sn (abscissa of Figure A.3). In this case, Sn goes from 1 to 10, and again, when $Sn = 10$ means that the network does not exploit the *U-turn* connections. Since for each input/output pair the 1024×1024 I/Os MSSN provides 1024 possible paths, the results are represented as clusters of points.

As visible in Figure A.3, the piecewise linear model of equation (A.10) is a "worst-case" compared to both the mean values depicted through lines with circle symbols and the ninetieth percentile values, triangular markers. There is only a case in Figure A.3 where the delay model under-estimates the real interconnection network delay of the max speed implementation and happens when Sn is equal to $S1, S2$, and $S3$. However, considering the overall delay related to the full embedded FPGA,

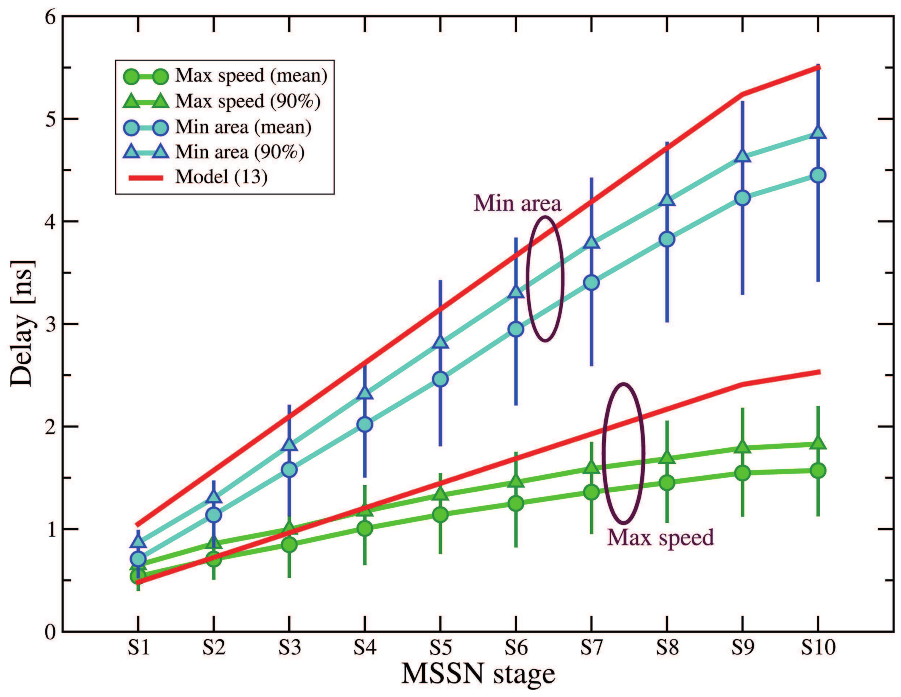


Figure A.3: 1024 × 1024 I/Os *radix-2* MSSN in-to-out delays (with bypass exploitation at different stages) compared to model (A.10) varying the MSSN stage *S* in which the *U-turn* connection is exploited. Picture reused from [40].

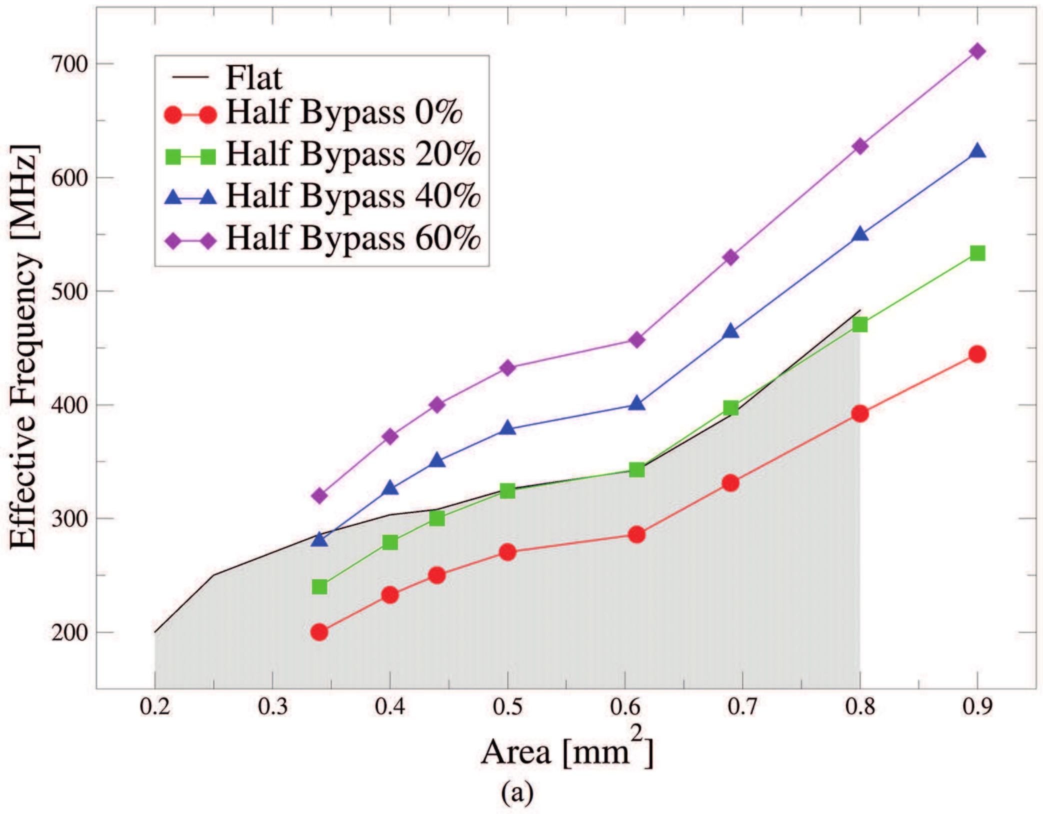
the under-estimation due to the proposed model would hardly affect the computation of the critical path, since it is dominated by the computational logic blocks contribution.

The previous considerations are still valid for *radix-4* MSSN architecture of Table A.2, differentiating for the fact that in this case, the stage delay is larger than that in the *radix-2* architecture since the *radix-4* switch elements have higher complexity as shown in Figure A.2.

The results depicted in Figure A.3 demonstrate the validity of the delay model of equation (A.10) and allow us to use it to compare the different MSSN architectures as explained in the following sections.

A.3.2 MSSN Effective Frequency Versus Bypass Exploitation

In order to evaluate the area/speed trade-off of the different MSSN architecture *flat*, *fully-bypassed* and *half-bypassed* it is performed an analysis taking into consideration the different interconnect architecture with the same area, but equipped with a different number of stages with enhanced via *U-turn* switches. For the *flat* architecture, we considered the implementation frequency as a figure of merit, since the interconnect structure delay corresponds to the MSSN full latency, while for the *bypassed* network the real working frequency has to be taken into account. A *bypassed* interconnection network provides shorter, and hence faster, links for inputs/outputs



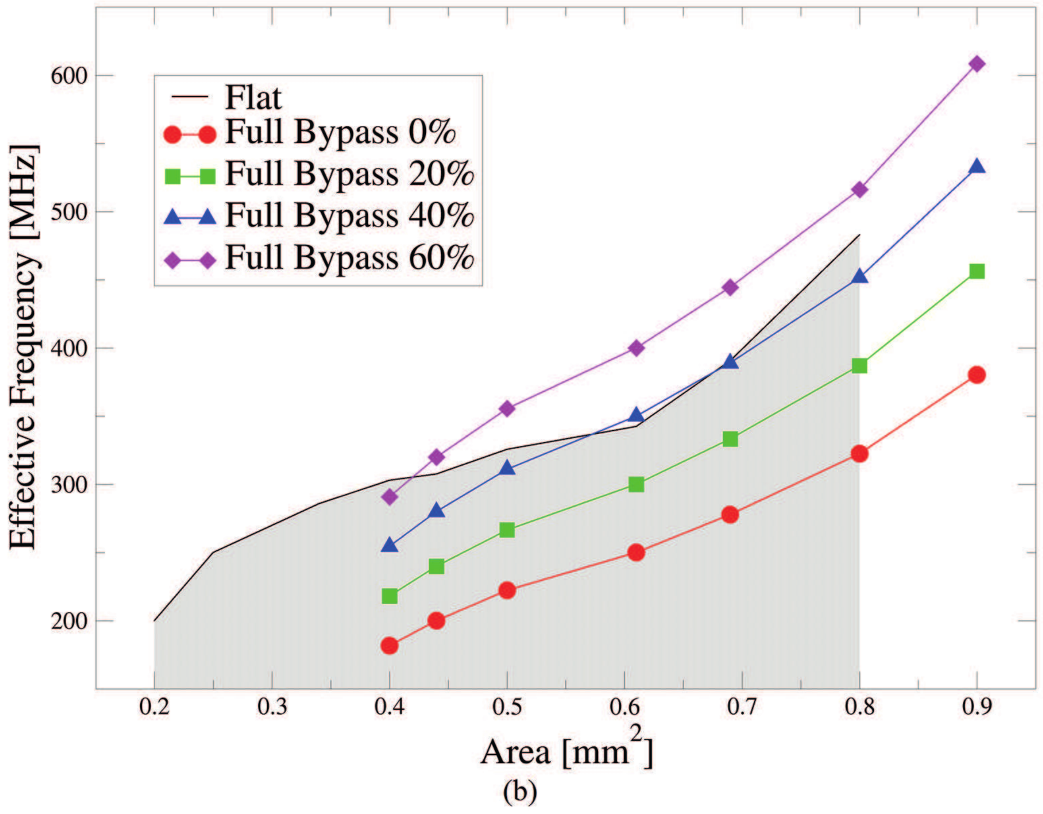


Figure A.4: 1024 × 1024 I/Os *radix*-2 MSSN effective frequency versus area for different *frequency gain* thanks to bypass exploitation: *flat* vs. *half-bypassed* (**a**) and *fully-bypassed* (**b**). Picture reused from [40].

logically nearby as shown in Figure 4.9. For this reason, the operating frequency depends on the locality of the application required connections. As illustrated in section 4.2.2, the locality optimization of the connection is mainly related to the routing strategy capability to best exploit bypasses.

Figure A.4 [4] shows the actual frequency of the *radix-2 flat* 1024×1024 interconnect architecture, and the achieved working frequency of a *radix-2 bypassed* 1024×1024 interconnection network. In this way, are specified four ranges of bypass exploitation (0% which corresponds to non-exploitation of the bypasses, 20%, 40%, 60%) resulting in four ranges of *frequency gain*.

As mentioned in Table A.1, the minimum area necessary to realize a *flat* interconnect architecture is 0.2 mm^2, 0.34 mm^2 for *half-bypassed* and 0.4 mm^2 for *fully-bypassed*. Hence, targeting small area devices less than 0.34 mm^2, the *flat* MSSN version is the unique solution among those listed. The *half-bypassed* interconnection network is advantageous for an area greater than 0.44 mm^2 as shown in Figure A.4(a), since it allows us to achieve higher frequencies even if the *frequency gain* is around 20%, while reaching *time-to-fly* frequency values beyond 700 MHz when the area is not a tight constraint. The same concept is applied to the *fully-bypassed* interconnection network where, if a small area (less than 0.8 mm^2) is required, it is achievable just only through a *flat* MSSN as depicted in Figure A.4(b). If one has a bigger area budget, is possible to reach higher operating frequencies. Comparing both *half-bypassed* and *fully-bypassed* is possible to notice that the *fully-bypassed* version reaches lower frequency due to the increased interconnection network density. On the other hand, the *fully-bypassed* interconnect provides a higher probability to exploit *U-turn* switches. For this reason, we decided to not consider structures with lower bypass levels such as *quarter-bypassed* network because with a small bypass exploitation probability the *frequency gain* might no longer be significant.

Applying the same analysis to a *radix-4* MSSN, we achieved similar results: advantages provided by introducing *U-turn* bypasses, either in half stages or in the full network, only become evident targeting high frequencies, with the drawback of additional area requirements [40].

A.3.3 Hierarchical MSSN Radix Comparison

In this section is performed a comparison in terms of area/speed trade-off in function of different *radix* of the interconnection network using the delay model presented in equation (A.10) and validated in the previous sections. In the previous section, we showed how the usage of enhanced switches improve the real operating frequency with respect to a *flat* interconnect structure. Now it is explained how the *radix* impacts on the interconnection network performance.

Changing the interconnect *radix* has a straightforward impact on the capability to exploit local connections as shown in Figure A.5. Indeed, in order to connect two MSSN input/output exploiting the bypass, the number of stages to cross change based on the chosen *radix*, as depicted in Figure A.5 [4]. For instance, in a 16×16

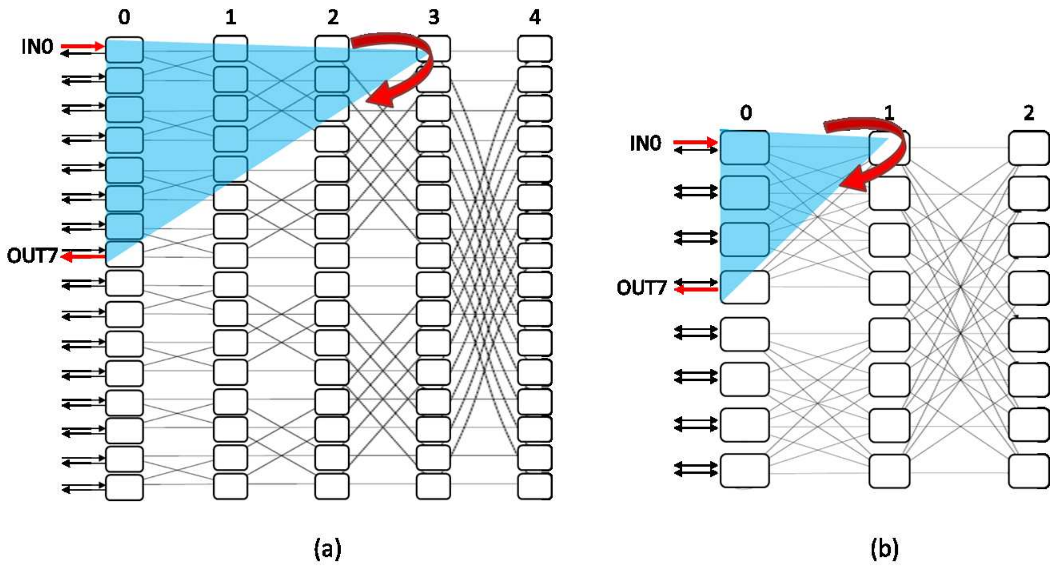


Figure A.5: Example of fully-bypassed MSSN connectivity with 16 × 16 I/Os: *radix-2* (**a**) and *radix-4* (**b**). Picture reused from [40].

radix-2 fully-bypassed MSSN of Figure A.5(a), in order to connect IN0 to OUT7 the minimum common stage S is $S = 3$, whereas, in a *radix-4* architecture the minimum common stage to be crossed is $S = 1$, as shown in Figure A.5(b). Therefore, in a *radix-2 fully-bypassed* MSSN the number of points (both inputs and outputs) subtended from a cone of height S is:

$$POINTS_2(S) = 2^S \tag{A.11}$$

while in the *radix-4* structure is:

$$POINTS_4(S) = \begin{cases} 2 \cdot 4^S, & S < \log_4 N \\ 4^S, & S = \log_4 N \end{cases} \tag{A.12}$$

The 2 factor is caused by the fact that each input and output stage is connected to two I/Os as visible in both Figure A.2 and Figure A.5(b). Generalizing to a generic *radix-k*, the number of subtended points becomes:

$$POINTS_k(S) = \begin{cases} \alpha \cdot k^S, & S < \log_k N \\ k^S, & S = \log_k N \end{cases} \tag{A.13}$$

for S from 1 to $\log_k N$. In the previous equation α (>0) is a factor that takes into account the number of inlets (outlets) connected to each input/output stage and it is $\alpha = \frac{N}{N_S}$. Since N_S is defined as $N_S = \frac{N}{k} \cdot p$, where p is the number of baseline planes. Hence, the α factor becomes:

$$\alpha = \frac{N}{N_S} = \frac{N}{N} \cdot \frac{k}{p} = \frac{k}{p}. \tag{A.14}$$

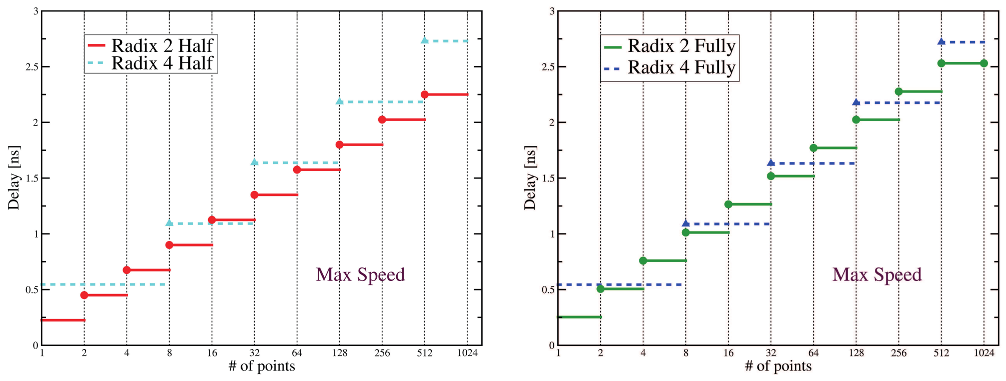


Figure A.6: Max-speed implementations: delay vs. topological distance between two I/Os for a 1024 × 1024 *fully-bypassed* and *half-bypassed* MSSN. Picture reused from [40].

In order to better understand equation (A.13), it can be re-write as:

$$POINTS_k(S) = \begin{cases} \alpha \cdot k^S, & S < \log_k N \\ \frac{\alpha}{\alpha} k^S = \alpha \cdot k^S \cdot \frac{p}{k}, & S = \log_k N \end{cases} \tag{A.15}$$

For a generic stage S, the number of points subtended from a cone of the previous stage $(S-1)$ is:

$$POINTS_k(S-1) = \alpha \cdot k^{S-1}. \tag{A.16}$$

As presented in [40], at the "*current*" stage S, the points subtended are those of the previous stage $(S-1)$ multiplied by k if S is not the central stage of the network, and thus, $S < \log_k N$. In the central stage case $S = \log_k N$, the different planes of Figure A.1(b) are merged in the central stage, as depicted in Figure 4.9, hence, the total number of points is the sum of the points of the previous stage of each plane p, and therefore multiplied by a p factor. Through equation (A.16), equation (A.15) can be expressed as:

$$POINTS_k(S) = \begin{cases} POINTS_k(S-1) \cdot k = \alpha k^{S-1} \cdot k = \alpha \cdot k^S, & S < \log_k N \\ POINTS_k(S-1) \cdot p = \alpha k^{S-1} \cdot p = k^S, & S = \log_k N \end{cases} \tag{A.17}$$

which corresponds to (A.13).

A comparison of *half-bypassed* and *fully-bypassed* architecture of a 1024 × 1024 MSSN with both *radix-2* and *radix-4* is performed. It is carried out a delay analysis, and thus the number of network stages to be crossed is considered, in order to connect two inputs/outputs placed at different distances. The analysis is based on both the stage delay expressed by the equation (A.9) and the $IN2OUT_{DELAY}$ delay model of equation (A.10). The analysis is performed assuming that the CAD tool, presented in section 4.2.2 places the MSSN I/Os minimizing their distance, and hence, forcing $S = S_{min}$.

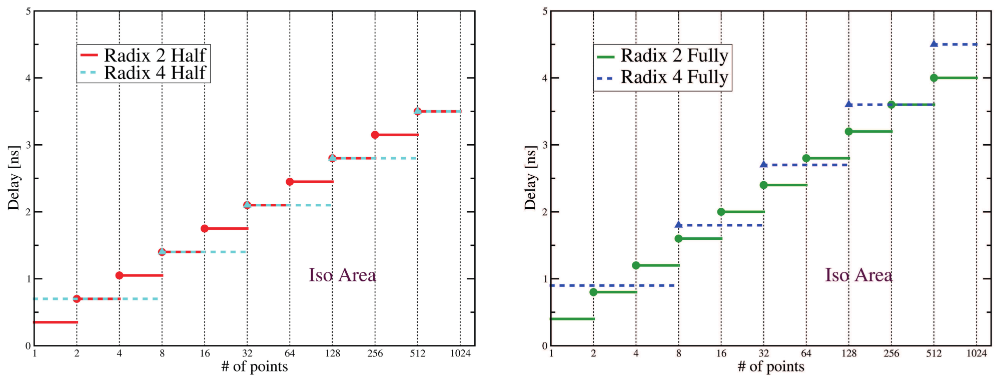


Figure A.7: Iso-area implementations: delay vs. topological distance between two I/Os for a 1024 × 1024 *fully-bypassed* and *half-bypassed* MSSN. Picture reused from [40].

Both Figures A.6 and Figure A.7 show the delays as a function of different distances between two MSSN I/Os that are to be connected. The data, obtained through post-synthesis simulation show that there is not a unique solution for the MSSN architecture and it is strongly related to the CAD capability to exploit bypasses. In the maximum speed implementations of Figure A.6, *radix-4* interconnect (dashed lines) has better performance, and thus, less average delay, only in the case of not too large input/output distance. On the other side, for iso-area implementations reported in Figure A.7, *radix-4* interconnection network (again dashed lines) prove to be advantageous reaching lower interconnect delay for any input/output distance in *half-bypassed* architecture whereas in the *fully-bypassed* structure is still advantageous in average for small I/Os distance, and hence, when it is possible to exploit its local connectivity.

The analysis can be extended to larger values of *radix-k*. It is worth noting that the area occupation of $k \times k$ switch element increases with $O(k^2)$ while the granularity of bypass exploitation decreases, hence reducing the probability of performance gain [40].

Bibliography

[1] F. Renzini, C. Mucci, D. Rossi, E. F. Scarselli, and R. Canegallo, "A fully programmable efpga-augmented soc for smart power applications", *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–13, 2019, ISSN: 1549-8328. DOI: `10.1109/TCSI.2019.2930412`.

[2] PULP-Platform. (Jan. 2019). PULPino datasheet, [Online]. Available: `https://www.pulp-platform.org/implementation.html`.

[3] M. Cuppini, C. Mucci, and E. F. Scarselli, "Soft-core embedded-fpga based on multistage switching networks: A quantitative analysis", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 12, pp. 3043–3052, 2015.

[4] M. Cuppini, "Methodologies for synthesizable programmable devices based on multi-stage switching networks", PhD thesis, University of Bologna, Via Zamboni 33, 40126 Bologna BO, Italy, May 2015. [Online]. Available: `http://amsdottorato.unibo.it/7013/1/Cuppini_Matteo_Tesi.pdf`.

[5] B. Murari, F. Bertotti, and G. A. Vignola, *Smart power ICs: technologies and applications*. Springer Science & Business Media, 2002, vol. 6.

[6] STMicroelectronics. (Mar. 2019). STSPIN datasheet, [Online]. Available: `http://www.st.com/en/motor-drivers.html`.

[7] ——, "STLUX datasheet", Tech. Rep. DocID027870 Rev 1, May 2015.

[8] A. Andreini, C. Contiero, and P. Galbiati, "A new integrated silicon gate technology combining bipolar linear, cmos logic, and dmos power parts", *IEEE Transactions on electron devices*, vol. 33, no. 12, pp. 2025–2030, 1986.

[9] STMicroelectronics, *BCD technology*. [Online]. Available: `https://www.st.com/content/st_com/en/about/innovation---technology/BCD.html`.

[10] M. Sanzaro, P. Gattari, F. Villa, A. Tosi, G. Croce, and F. Zappa, "Single-photon avalanche diodes in a 0.16 μm bcd technology with sharp timing response and red-enhanced sensitivity", *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 24, no. 2, pp. 1–9, 2018.

[11] Z. Dong, F. Lu, R. Ma, L. Wang, C. Zhang, G. Chen, A. Wang, and B. Zhao, "An integrated transmitter for led-based visible light communication and positioning system in a 180nm bcd technology", in *2014 IEEE Bipolar/BiCMOS Circuits and Technology Meeting (BCTM)*, IEEE, 2014, pp. 84–87.

[12] M. Rose and H. J. Bergveld, "Integration trends in monolithic power ics: Application and technology challenges", *IEEE Journal of Solid-State Circuits*, vol. 51, no. 9, pp. 1965–1974, 2016.

[13] D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low power methodology manual: for system-on-chip design*. Springer Science & Business Media, 2007.

[14] STMicroelectronics, "STM32L151xE STM32L152xE datasheet", Tech. Rep. DocID025433 Rev 9, Aug. 2017.

[15] Xilinx. (Aug. 2019). Xilinx system-on-chips, [Online]. Available: `https://www.xilinx.com/products/silicon-devices/soc.html`.

[16] Intel. (Aug. 2019). Intel system-on-chips, [Online]. Available: `https://www.intel.com/content/www/us/en/products/programmable/soc.html`.

[17] QuickLogic. (Aug. 2019). Embedded fpga, [Online]. Available: `https://www.quicklogic.com/`.

[18] Lattice Semiconductor. (Mar. 2019). Lattice products, [Online]. Available: `http://www.latticesemi.com/`.

[19] Microsemi. (Aug. 2019). Microsemi products, [Online]. Available: `https://www.microsemi.com/`.

[20] Menta. (Aug. 2019). Embedded programmable logic, [Online]. Available: `http://www.menta-efpga.com/`.

[21] Flex Logix. (Aug. 2019). Embedded fpga basics, [Online]. Available: `http://www.flex-logix.com/fpga-tutorial`.

[22] Achronix. (Aug. 2019). The Speedcore embedded fpga, [Online]. Available: `https://www.achronix.com/`.

[23] J. H. Kim and J. H. Anderson, "Synthesizable standard cell fpga fabrics targetable by the verilog-to-routing cad flow", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 2, p. 11, 2017.

[24] M. Borgatti, F. Lertora, B. Forêt, and L. Calí, "A reconfigurable system featuring dynamically extensible embedded microprocessor, fpga, and customizable i/o", *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 521–529, 2003.

[25] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, *et al.*, "Xisystem: A xirisc-based soc with reconfigurable io module", *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 85–96, 2006.

[26] D. Rossi, F. Campi, S. Spolzino, S. Pucillo, and R. Guerrieri, "A heterogeneous digital signal processor for dynamically reconfigurable computing", *IEEE Journal of Solid-State Circuits*, vol. 45, no. 8, pp. 1615–1626, 2010.

[27] NXP Semiconductors, "SAC57D54H Data Sheet: Technical Data", Tech. Rep. SAC57D54H, Rev. 7, May 2017.

[28] STMicroelectronics, "STM32™32-bit mcu family", Tech. Rep. BRSTM320218, Feb. 2018.

[29] Texas Instruments, "MSP430FR604x, MSP430FR504x ultrasonic sensing MSP430™ microcontrollers for gas and water flow metering applications datasheet", Tech. Rep. SLASEF5, Jan. 2019.

[30] STMicroelectronics, "STSPIN32F0 Advanced BLDC controller with embedded STM32 MCU", Tech. Rep. DocID029806 Rev 2, Mar. 2017.

[31] Nordic Semiconductor, "nRF52 Series SoC", Tech. Rep. nRF52_Series_PB_v3.0, Mar. 2019.

[32] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review", *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[33] E. Lupon, S. Busquets-Monge, and J. Nicolas-Apruzzese, "Fpga implementation of a pwm for a three-phase dc–ac multilevel active-clamped converter", *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1296–1306, 2014.

[34] S. J. Wilton, C. H. Ho, B. Quinton, P. H. Leong, and W. Luk, "A synthesizable datapath-oriented embedded fpga fabric for silicon debug applications", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 1, no. 1, p. 7, 2008.

[35] D. Rossi, C. Mucci, F. Campi, S. Spolzino, L. Vanzolini, H. Sahlbach, S. Whitty, R. Ernst, W. Putzke-Roming, and R. Guerrieri, "Application space exploration of a heterogeneous run-time configurable digital signal processor", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, pp. 193–205, 2013.

[36] F Renzini, D Rossi, E. F. Scarselli, C Mucci, and R Canegallo, "A fully programmable efpga-augmented soc for smart-power applications", in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, IEEE, 2018, pp. 241–244.

[37] ARM. (Sep. 2019). ARM Cortex-M, [Online]. Available: `https://developer.arm.com/ip-products/processors/cortex-m`.

[38] Andreas Traber, Michael Gautschi. (Jun. 2017). PULPino: Datasheet, [Online]. Available: `https://github.com/pulp-platform/pulpino/blob/master/doc/datasheet/datasheet.pdf`.

[39] Synopsys. (Mar. 2019). Design Compiler tool in graphical mode, [Online]. Available: `https://www.synopsys.com/`.

[40] F. Renzini, M. Cuppini, C. Mucci, E. Franchi Scarselli, and R. Canegallo, "Quantitative analysis of multistage switching networks for embedded programmable devices", *Electronics*, vol. 8, no. 3, p. 272, 2019.

[41] I. Kuon, R. Tessier, J. Rose, *et al.*, "Fpga architecture: Survey and challenges", *Foundations and Trends® in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[42] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays", *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1013–1029, 1993.

[43] F.-L. Yuan, C. C. Wang, T.-H. Yu, and D. Marković, "A multi-granularity fpga with hierarchical interconnects for efficient and flexible mobile computing", *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 137–149, 2014.

[44] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon, "Hsra: High-speed, hierarchical synchronous reconfigurable array", in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, ACM, 1999, pp. 125–134.

[45] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, and G. Snider, "Teramac-configurable custom computing", in *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, IEEE, 1995, pp. 32–38.

[46] R. Amerson, R. Carter, W Culbertson, P. Kuekes, G. Snider, and L. Albertson, "Plasma: An fpga for million gate systems", in *Fourth International ACM Symposium on Field-Programmable Gate Arrays*, IEEE, 1996, pp. 10–16.

[47] F. Reblewski and O. LePape, *Reconfigurable integrated circuit with a scalable architecture*, US Patent 6,594,810, Jul. 2003.

[48] D. Wong, *Interconnection network for a field programmable gate array*, US Patent 6,693,456, Feb. 2004.

[49] C. C. Wang, F.-L. Yuan, H. Chen, and D. Markovic, "A 1.1 gops/mw fpga chip with hierarchical interconnect fabric", in *2011 Symposium on VLSI Circuits-Digest of Technical Papers*, IEEE, 2011, pp. 136–137.

[50] C. Wang and D. Markovic, *Network architectures for boundary-less hierarchical interconnects*, US Patent App. 14/777,477, Feb. 2016.

[51] A. DeHon and R. Rubin, "Design of fpga interconnect for multilevel metallization", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10, pp. 1038–1050, Oct. 2004, ISSN: 1063-8210. DOI: `10.1109/TVLSI.2004.827562`.

[52] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks.* Elsevier, 2004.

[53] C. Clos, "A study of non-blocking switching networks", *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[54] Y. Yang and J. Wang, "Nonblocking k-fold multicast networks", *IEEE Transactions on Parallel and Distributed systems*, vol. 14, no. 2, pp. 131–141, 2003.

[55] ——, “On blocking probability of multicast networks”, *IEEE Transactions on Communications*, vol. 46, no. 7, pp. 957–968, 1998.

[56] D. P. Agrawal, “Graph theoretical analysis and design of multistage interconnection networks”, *IEEE Transactions on Computers*, no. 7, pp. 637–648, 1983.

[57] C.-L. Wu and T.-Y. Feng, “On a class of multistage interconnection networks”, *IEEE transactions on Computers*, vol. 100, no. 8, pp. 694–702, 1980.

[58] A. Pattavina, *Switching Theory: Architectures and Performances in Broadband ATM Networks*, J. Wiley and S. Ltd, Eds. Wiley, 1998.

[59] C. Lea, “Multi–log$_2$n networks and their applications in high-speed electronic and photonic switching systems”, *IEEE Trans. on Communications*, vol. 38, no. 10, pp. 1740–1749, 1990.

[60] M. Collier, “A systematic analysis of equivalence in multistage networks”, *Journal of Lightwave Technology*, vol. 20, no. 9, pp. 1664–1672, Sep. 2002.

[61] E. W. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische mathematik*, vol. 1.1, pp. 269–271, 1959.

[62] STMicroelectronics, “STM32L1xx ultralow power features overview datasheet (AN3193)”, Tech. Rep. DocID17369 Rev 2, Sep. 2013.

[63] G. Baccarani, M. R. Wordeman, and R. H. Dennard, “Generalized scaling theory and its application to a 1/4 micrometer mosfet design”, *IEEE Transactions on Electron Devices*, vol. 31, no. 4, pp. 452–462, 1984.

[64] Lattice Semiconductor, “ICE40 led driver usage guide”, Tech. Rep. Technical Note TN1288, Jun. 2016.

[65] Microchip, “Brushed DC motor fundamentals”, Tech. Rep. AN905, Jan. 2004.

[66] Shyam Sadasivan, “An Introduction to the ARM Cortex-M3 Processor”, ARM, Tech. Rep., Oct. 2006.

[67] C.-h. Liu, J.-s. Ji, and X.-p. Chen, “Control module for stepper motor based on fpga”, in *2010 International Conference on E-Product E-Service and E-Entertainment*, IEEE, 2010, pp. 1–3.

[68] C. Mucci, L. Vanzolini, I. Mirimin, D. Gazzola, A. Deledda, S. Goller, J. Knaeblein, A. Schneider, L. Ciccarelli, and F. Campi, “Implementation of parallel lfsr-based applications on an adaptive dsp featuring a pipelined configurable gate array”, in *Proceedings of the conference on Design, automation and test in Europe*, ACM, 2008, pp. 1444–1449.

[69] A. Elgani, M. Magno, F. Renzini, L. Perilli, E. F. Scarselli, A. Gnudi, R. Canegallo, G. Ricotti, and L. Benini, “Nanowatt wake-up radios: Discrete-components and integrated architectures”, in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, IEEE, 2018, pp. 793–796.

[70] E. Popovici, M. Magno, and S. Marinkovic, “Power management techniques for wireless sensor networks: A review”, in *5th IEEE International Workshop on Advances in Sensors and Interfaces IWASI*, IEEE, 2013, pp. 194–198.

[71] D. Spenza, M. Magno, S. Basagni, L. Benini, M. Paoli, and C. Petrioli, “Beyond duty cycling: Wake-up radio with selective awakenings for long-lived wireless sensing systems”, in *2015 IEEE Conference on Computer Communications (INFOCOM)*, IEEE, 2015, pp. 522–530.

[72] M. Magno, V. Jelicic, B. Srbinovski, V. Bilas, E. Popovici, and L. Benini, “Design, implementation, and performance evaluation of a flexible low-latency nanowatt wake-up radio receiver”, *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 633–644, 2016.

[73] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers, “Matching properties of mos transistors”, *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, 1989.

[74] P. P. Wang, H. Jiang, L. Gao, P. Sen, Y. Kim, G. M. Rebeiz, P. P. Mercier, and D. A. Hall, “A near-zero-power wake-up receiver achieving- 69-dbm sensitivity”, *IEEE Journal of Solid-State Circuits*, vol. 53, no. 6, pp. 1640–1652, 2018.

[75] N. E. Roberts and D. D. Wentzloff, “A 98nw wake-up radio for wireless body area networks”, in *2012 IEEE Radio Frequency Integrated Circuits Symposium*, IEEE, 2012, pp. 373–376.

[76] V. E. Benes, *Mathematical theory of connecting networks and telephone traffic*. New York: Academic press, 1965, vol. 17.

[77] ——, “Optimal rearrangeable multistage connecting networks”, *Bell System Technical Journal*, pp. 1641–1656, 1964.

[78] C. Lea and D. Shyy, “Tradeoff of horizontal decomposition versus vertical stacking in rearrangeable nonblocking networks”, *IEEE Trans. on Communications*, pp. 899–904, 1991.

[79] Y. Tscha and K. H. Lee, “Nonblocking conditions for multi-$\log_2 n$ multiconnection networks”, *Global Telecommunications Conference, GLOBECOM’92, Communication for Global Users, IEEE*, vol. 3, pp. 1600–1604, 1992.

[80] F. K. Hwang and W.-D. Lin., “Necessary and sufficient conditions for rearrangeable $\log_d(n, m, p)$”, *IEEE Trans. Commun.*, vol. 53, no. 12, pp. 2020–2023, Dec. 2005.