POLITECNICO DI MILANO
DEPARTMENT OF ELECTRONIC, INFORMATION AND BIOENGINEERING
(DEIB)
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

ONLINE POWER MODELING, MONITORING AND OPTIMIZATION FOR MOBILE COMPUTING PLATFORMS

Doctoral Dissertation of:
Luca Cremona

Supervisor:
Prof. William Fornaciari

Co-Supervisor:
Prof. Davide Zoni

Tutor:
Prof. Francesco Amigoni

The Chair of the Doctoral Program:
Prof. Barbara Pernici

Ph.D Cycle XXXIII

Acknowledgements

To everyone that supported me during this PhD, especially to my family,
my girlfriend and all the HeapLab people.

Abstract

THE Internet-of-Things (IoT) revolution fueled new challenges and opportunities to achieve computational efficiency goals. Embedded devices are required to execute multiple applications for which a suitable distribution of the computing power must be adapted at run-time. Such complex hardware platforms have to sustain the continuous acquisition and processing of data under severe energy budget constraints, since most of them are battery powered. The state-of-the-art offers several ad-hoc contributions to selectively optimize the performance considering aspects like energy, power, thermal or reliability. In this scenario, the use of hardware-level online power monitors is crucial to support the run-time power optimizations required to meet the ever increasing demand for energy efficiency. To be effective and to deal with the time-to-market pressure, the presence of such requirements must be considered even during the design of the power monitoring infrastructure. This thesis presents a power model identification and implementation strategy with two main advantages over the state-of-the-art. First, the proposed solution trades the accuracy of the power model with the amount of resources allocated to the power monitoring infrastructure. Second, the use of an automatic power model instrumentation strategy ensures a timely implementation of the power monitor regardless the complexity of the target computing platforms. To assess the effectiveness of the proposed solution the identified power monitor has been adopted to feed a power optimization scheme, based on a control theory based PID controller. Both the single-core and multi-core scenarios have been taken into consideration. The online power monitor has been validated against 8 accelerators generated through a High-Level-Synthesis

flow and by considering a more complex RISC-V embedded computing platform. The all-digital power optimization scheme has been validated against the *nu+* processor, a 16-ways SIMD processor with a configurable number of cores. For the assessment of the proposed control scheme, this thesis considers the four core configuration, running 20 applications from the WCET benchmark suite. For what concerns the power monitor, depending on the imposed user-defined constraints and with respect to the unconstrained power monitoring state-of-the-art solutions, the proposed methodology shows a resource saving between 37.3% and 81% while the maximum average accuracy loss stays within 5%, i.e., using the aggressive 20us temporal resolution. However, by varying the temporal resolution closer to the value proposed in the state of the art, i.e. in the range of hundreds of microseconds, the average accuracy loss of the power monitors is lower than 1% with almost the same overheads. In addition, the presented solution demonstrated the possibility of delivering a resource constrained power monitor employing a 20us temporal resolution, i.e., far higher the one used by current state-of-the-art solutions. The power optimization scheme, instead, shows an overhead limited to 0.86%(FFs) and 5.3%(LUTs) of the FPGA chip. The performance results are analyzed considering three quality metrics. First, the efficiency in exploiting the imposed budget (EFF_g) that is on average 98.27%. Second, the overflow of the actual average power consumption with respect to the assigned budget (OVF_g), which is limited to 1.43 mW on average. Last, the performance utility loss due to the control scheme that is limited to 1.87% on average.

Abstract (in Italian)

La rivoluzione dell'Internet of Things (IoT) ha dato vita a nuove sfide e occasioni per raggiungere nuovi traguardi di efficienza computazionale. I dispositivi embedded sono spinti sempre oltre, per dar modo di affrontare applicazioni per le quali le alte prestazioni devono talvolta lasciar spazio a requisiti di risparmio energetico. Dato che tali dispositivi sono spesso alimentati a batteria, devono spesso svolgere compiti di acquisizione e elaborazione di dati sotto stringenti requisiti energetici. La letteratura offre molteplici soluzioni per ottimizare il consumo energetico di tali dispositivi, pur mantenendo un alto profilo in termini di prestazioni. In questo scenario, l'utilizzo di hardware power meters si rivela cruciale, al fine di supportare efficaci tecniche di gestione del consumo di potenza a run time. Per essere incisivi e pronti a ridurre i tempi di ingresso nel mercato, si deve condiserare la presenza di tali requisiti fin dalle prime fasi di design. Questa tesi presenta una metodologia di identificazione e implementazione di power monitors con due principali vantaggi rispetto a quanto presente in letteratura. Primo di tutti questo approccio gestisce il tradeoff tra una metrica di accuratezza del modello identificato e il corrispettivo overhead introdotto. Secondo, utilizzando un approccio completamente automatico, questa metodologia diminuisce sensibilmente i tempi di implementazione. Per verificare l'efficiacia di questa metodologia, sono stati considerati scenari con processori sia single che multi core, utilizzando attuatori costruiti secondo la teoria del controllo. L'efficacia dei power metes viene in questo lavoro verificata utilizzando otto acceleratori hardware e un System on Chip che implementa un processore RISC-V. Per quanto riguarda invece l'accuratezza degli

attuatori viene preso in considerazione un processore SIMD a 16 linee, con un numero configurabile di cores attivi, su cui vengono eseguiti venti benchmarks provenienti dalla suite WCET. Per quanto concerne i power monitors, a seconda dei constraints imposti dall'utente, rispetto alla versione senza constraints, questa metodologia presenta un risparmio in termini di risorse che va dal 37.3% al 81%, mentre la perdita di accuratezza rimane sotto il 5%., usando la maggiore risoluzione temporale (20us). Variando tale risoluzione temporale, e portandola ai valori presenti in letteratura (centinaia di microsecondi) la perdita di accuratezza rimane sotto il 1%. Questa metodologia presenta inoltre il vantaggio di poter raggiungere risoluzioni temporali molto fitte, utili in casi in cui la dinamica del consumo di potenza diventa molto alta. Lo schema di controllo del consumo di potenza, invece, presenta un overhead dello 0.86% (FFs) e 5.3%(LUTs). In questo lavoro sono state definite tre metriche per valutare l'efficacia dello schema presentato: l'efficienza risulta del 98.27%, l'overflow di 1.43mW e l'utility loss del 1.87%.

Contents

Contents

Contents

CHAPTER 1

Introduction

With the incoming of the IoT world, an ever growing number of digital devices are connected to the Internet. The key idea behind the IoT is the possibility for different types of physical devices to collect, elaborate and transmit data without the interference of humans. IoT devices rapidly became part of human life, thanks to their possibility to provide useful services at an affordable cost. Modern applications push higher and higher requirements for the computational power of these IoT devices that, due to their small size, sometimes cannot satisfy some user requirements, such as execution time, throughput, power consumption. Cloud computing comes into play here: data collected by the edge devices is sent to a server located into a cloud data center, where the most computationally hard calculations are performed. In many applications, IoT devices and sensors collect data and perform actions in a local network; also in such situation a central computing unit could be required to manage the interactions between the IoT devices and to help them with the computations. In this scenario, a new platform called *Fog/Edge Computing* has been introduced to help IoT applications that either are not executable on the cloud or the latency due to the communications with the cloud is not affordable. For example, applications constrained by a fast response time could find difficult the data

transmission to the cloud layer, since the delay of the communication could make these applications miss the imposed requirements. Fog server are not installed in the Cloud but they are placed near to the scenario where they are employed; however, the communications between the Fog and the Cloud are always guaranteed. In these last years, a huge effort has been spent on the upgrade of these Fog platforms, in order to improve security, privacy and, especially, energy efficiency aspects. Figure 1.1 shows an example of a cloud/fog computing environment. From Figure 1.1, it can be noticed how the different IoT devices can either be directly connected to a cloud computing infrastructure, or be part of an intermediate computing unit (the fog environment).

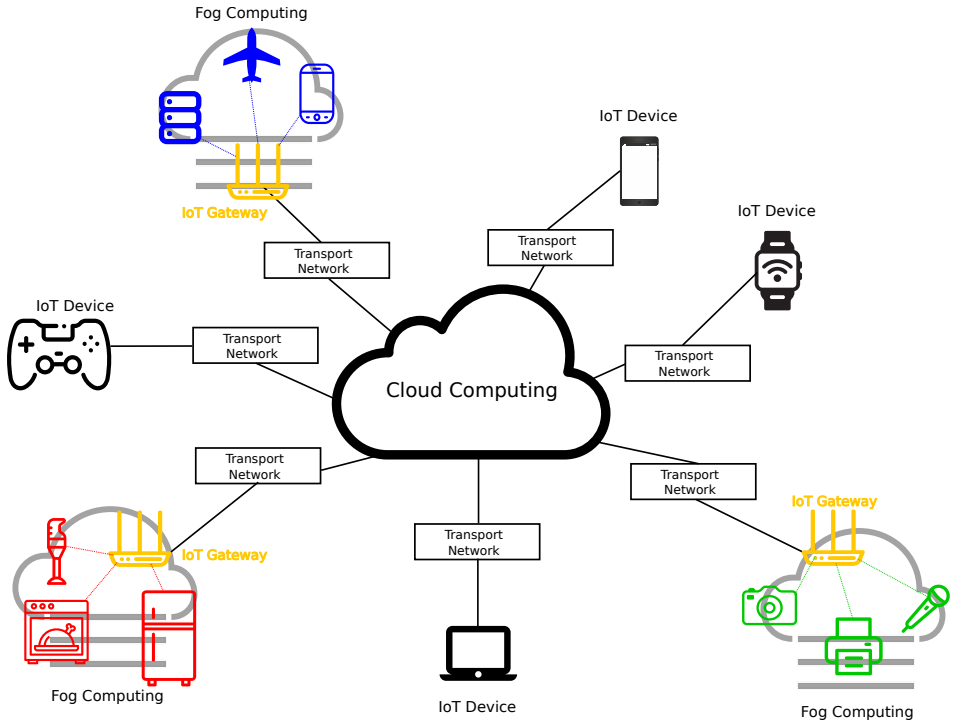


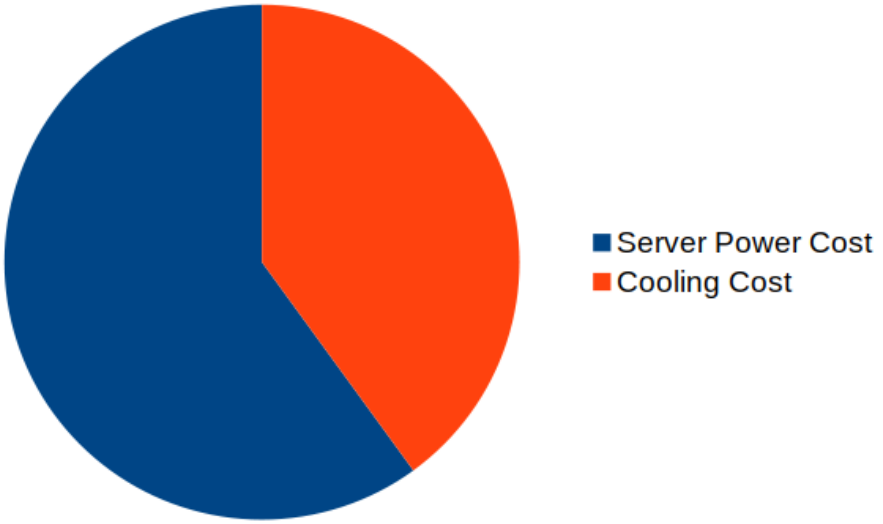
Figure 1.1: *An example of Cloud/Fog computing environment. IoT devices can either be directly connected to the cloud computing servers or being part of a fog infrastructure.*

The power/energy consumption is one of the most limiting factors for the improvement of the computational performance for nowadays microprocessors targeting both the embedded and the High Performance Computing (HPC) world. At the low end, the power consumption have always dominated over performance as the most important design constraint; however, while the battery capacity show a modest increase over these last

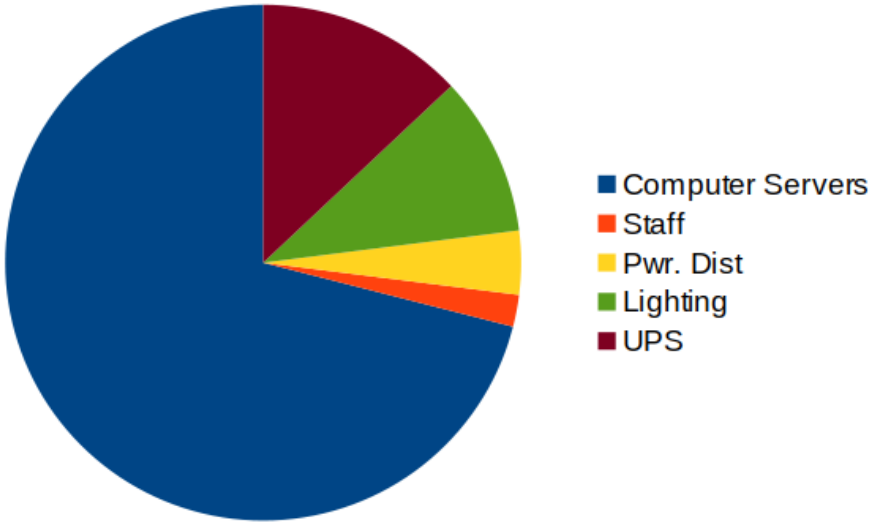
years, the ever increasing computational power demand critically affects the severity of the power constraint in the world of handheld and mobile devices. At high end, where performance was always the most important metric, the CMOS technology scaling constraints push the power and energy consumption aspects to a higher importance level, with respect to performance. Thus, regardless the application domain, the power consumption represents the most important "wall" for the growth of cost-effective performance in both HPC and embedded computing devices. This situation is known as the *Power Wall* problem. The power and energy consumption is not only a problem for the design of the microprocessors or a limiting factor for the growth of the computational capacity of nowadays digital computing devices. Power consumption is also one of the highest cost for most of the data and supercomputing centers. In fact, the impressive computational capacity of a supercomputer, makes it also very hungry for power. For example, the Cray XT5 Jaguar supercomputer at Oak Ridge National Laboratory, consumes up to seven megawatts, enough to power a town or a small city, and about a half of the total power is used to power the supercomputer, while the other half is used to cool it. Nowadays, with the incoming of the exascale era, the power bills for supercomputing centers spreaded in the world are around hundreds of million dollars annually. Sumit Gupta, senior manager of the Tesla GPU computing business unit at NVIDIA reports: "Power consumption and the need for more energy efficient computing systems is the top of mind for most of our customers."

As it can be noticed from Figure 1.2a, Gupta estimates that between 40% and 60% of the energy costs in these supercomputing centers are spent for cooling, while the remaining portion of the energy costs are for powering the supercomputers. Furthermore, the 71% of the cooling costs are spent for cooling the servers (see Figure 1.2b); this number underlines the importance of the energy-related cost in the economy of a data center.

The risk of a substantial increase of the energy consumption due to the spreading of IoT devices and the related services has been analyzed in [19]. In this work, provided by the International Energy Agency (IEA), based on the Energy Efficient End-Use Equipment (4E) Agreement, authors estimated the annual standby energy consumption for off-the-shelf mains-powered IoT devices and their respective gateways, using market research projections (2015-2025) of future IoT device shipments and power measurements of the IoT devices. Authors estimate that the annual global standby energy consumption of five selected IoT applications use-cases (Home Automation, Smart Lighting, Smart Appliance, Smart Roads and



(a) *Server costs*



(b) *Cooling costs*

Figure 1.2: *Distribution of the energy costs for a datacenter. Data from Mission Possible, Greening the HPC Data Center By Nicole Hemsoth.*

Smart Street Lights) could reach 46 TWh by 2025, with the share of Home Automation and Smart Appliances being 78% (36 TWh) and 15% (7 TWh) respectively. Figure 1.3 depicts this trend. Power/Energy consumption optimization is not a challenge for HPC architectures only, but for all the hierarchical infrastructure of the new introduced edge/fog computing. In fact,

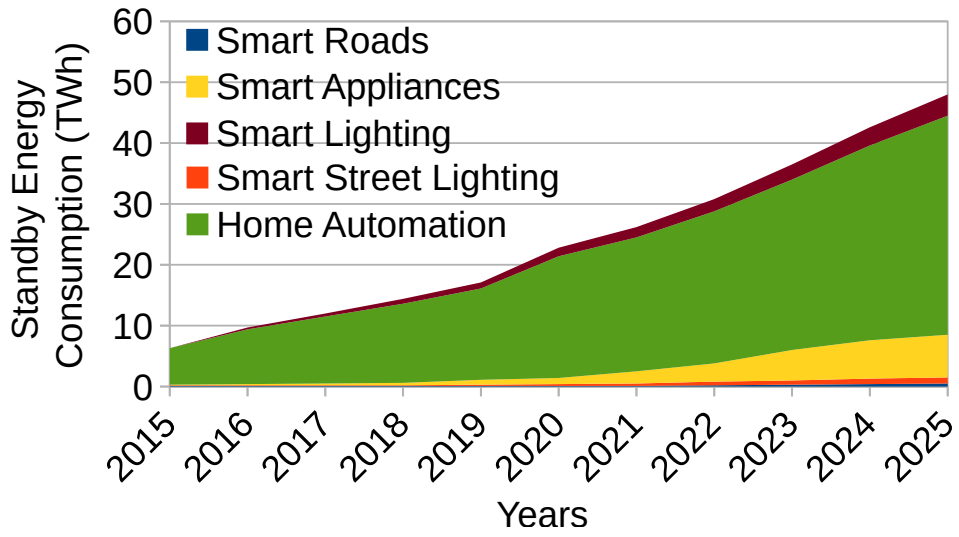


Figure 1.3: *The worldwide network related standby energy consumption of devices used for Smart Lighting, Home Automation, Smart Appliances, Smart Street Lighting and Smart Roads is predicted to reach 46 TWh in the year 2025. Data from "Energy Efficiency of the Internet of Things Technology and Energy Assessment"*

one of the most critical problems for the edge devices is the fact that they are mostly battery powered, thus having a limited power budget. On of the major challenges for the design of embedded devices is the optimization of the trade-off between energy consumption end performance: the most important final user requirement for these devices is a good user experience, characterized by a fast response time, good services, and the opportunity to use the devices for a long time without the risk of going out of battery. With the evolution of the science and technology, IoT devices are running more and more complex applications, covering a large set of human needs, and objects such as the smartphones are becoming essential in nowadays life.

Figure 1.4 shows the trend for the number of smartphone users from 2012 to 2020, with an estimate for 2023. As it can be noticed from Figure 1.4, the number of smartphones user in 2020 is four times higher with respect to 2012. This rapid growth is due to mainly two factors: first of all, the price of smartphones (especially the low-middle sector) decreases a lot: the first Apple iPhone was launched on the market in 2007 and its price ranged from 500 to 600 dollars. Nowadays a smartphone with the same technical characteristics is sold with a far less price, around 100 or 200 dollars. The second factor is the vastity of applications available for

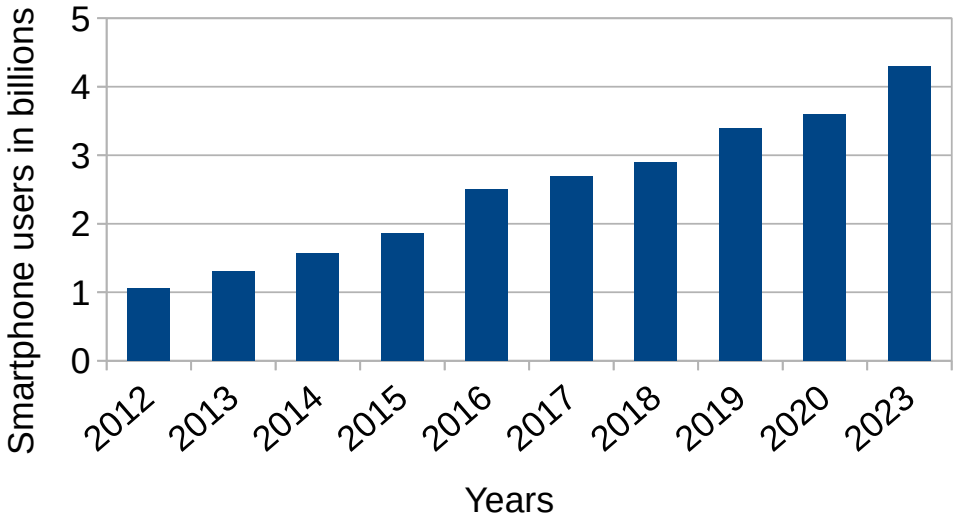


Figure 1.4: *The number of smartphone users worldwide today surpasses three billion and is forecast to further grow by several hundred million in the next few years. China, India, and the United States are the countries with the highest number of smartphone users, with a combined 1.46 billion users. Data from "Newzoo"*

the smartphones, able to satisfy a lot of human needs. Nowadays, with a smartphone people can turn on and off home lights, the home alarm, the oven, the washing machines and many other smart home devices. Furthermore, it is possible to pay contactless, take very high quality photos, play games, use social networks a many other useful things. It is important noticing that, nowadays, people rely on this digital devices, and the availability of their services has become of paramount importance in the life of most people.

The most important cause that can threaten the availability of the smartphones, is the battery status, that, especially at the end of the day could be low. This fact is underlined and confirmed by an analysis conduced by Grand View Research, where authors shows very interesting numbers about the North America market size of power bank devices. From this analysis it emerges that the global power bank market size was valued at USD 6.8 billion in 2019. and was anticipated to register a Compounded Average Growth Rate (CAGR) of 18.4% from 2020 to 2027. The authors of this report underline that the growth of this market can be attributed to the increase in the adoption of smartphones and other electronic tools and the rising power consumption of electronic devices due to advancements in mobile technologies. Figure 1.5 shows the trend of the North America

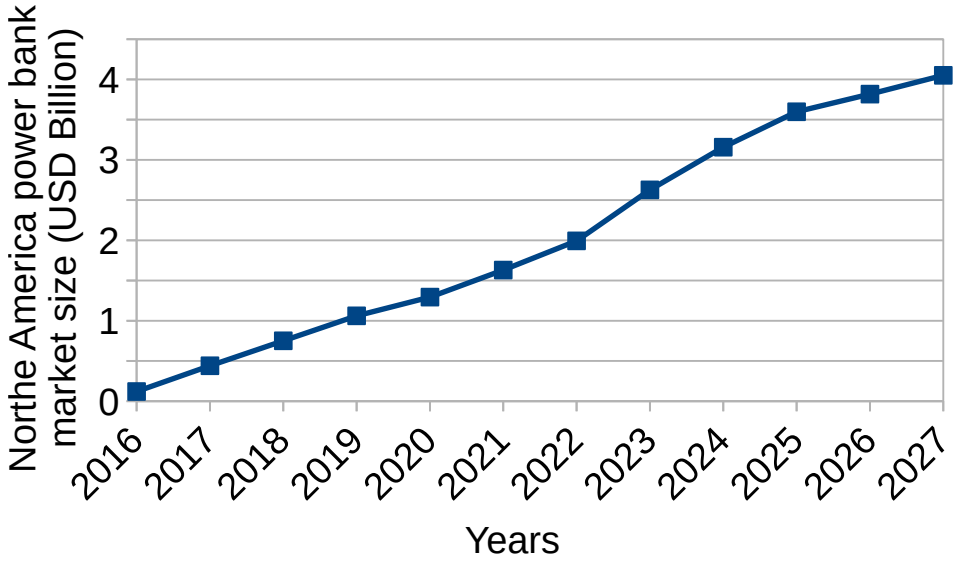


Figure 1.5: *The North America power bank market size grown from 0.12 billions in 2016 to 1,29 billion dollars in 2020. It is estimated to grow up to 4 billions in 2027. Data from: Grand View Research*

power bank market size from 2016 to 2027; as it can be noticed, the market size in 2016 was 0.12 billion, while in 2020 was 1,29 billion dollars. The North America market power bank size is estimated to grow up to 4 billion in 2027. These data show the need for the users to more energy to power on their devices and the fear to go off with the smartphones and other IoT devices that, as mentioned before, cover a key role in the daily life.

Given these scenario, the need for power optimization techniques emerges as a need of paramount importance, both for the HPC and embedded device world. Most common power optimization techniques adopts software routines, usually implemented within the operating system, that leverage power estimates to produce an action signal feeding the power actuators. The most used power actuators are the Dynamic Voltage and Frequency Scaling (DVFS) and clock gating. However, the key enabling factor to make these power optimization policies effective is the adoption of a fast and accurate power monitoring mechanism, able to provide power estimates at a fine grained time resolution. In fact, modern applications show different phases during their execution, alternating moments where the computational capacity required is high to moments where the application requires far less computational resources. This alternation of computational phases also implies an alternation of the power consumption, that

can increase or decrease its value by a 50% in some microseconds, thus increasing the required actuation rate of the power optimization schemes. The state of the art proposes different solutions to face the power monitoring problem, both hardware or software implemented. For what concerns the software implemented online power monitoring infrastructures, the most common procedure leverages the information coming from the performance counters as the power model predictors. After collecting the data of the performance counters and the power traces for a set of benchmarks, a mathematical representation of power model is extracted offline and then, a software routine implementing the corresponding power monitor is implemented. This software routine periodically runs, producing the power estimates. This approach shows both advantages and disadvantages: the main advantages are that software power monitors can be designed and implemented also after the production of the chip and that there it is no necessary to have and modify the original RTL design of the target device. Furthermore, this solution often shows a very high level of accuracy. However, this approach shows a main disadvantages that limits its application to a non negligible set of devices, especially in the embedded world: the performance counters are an optional feature, and they are not implemented in all the processors. Since they require additional hardware resources to be implemented, in small embedded chipset they are not implemented. Furthermore, due to the need of a software routine, this approach show a low power estimate rate. For what concerns, instead, hardware implemented power monitoring infrastructure, the state of the art proposes the adoption of the switching activity of the architectural wires as model predictors. Once the toggle counts and the power traces have been collected for a set of benchmarks, as before, a mathematical formulation of the power model is identified, and the corresponding power monitor is implemented directly into the RTL description of the target design. Also this approach shows advantages and disadvantages: the main advantage is the very high estimate rates, that can reach up to an estimates per microsecond. Furthermore, there is no need for pre-implemented resources (such as the performance counters): all the power monitoring infrastructure elements are designed and implemented within the methodology. One of the most important disadvantages of this solution is that it modifies the RTL description of the target devices, so it has to be applied during the design process of the chip. However, the main drawback of this methodology is the possible high resource overhead introduced by the hardware implementation of the power monitor; in fact, especially in small processors, the implementation of an hardware power monitor can introduce an area and power overhead of up to 30%.

From an analysis of the state of the art, it can be noticed that all the works targeting RTL online power monitoring have, as goal, the highest accuracy as possible. This fact makes the power monitor huge, while sometimes it is not necessary to have a too high accuracy of the power traces.

This thesis is focused on hardware solutions for switching activity based power monitoring infrastructures, in particular on the resource constrained identification of power models. The goal of this thesis is the identification of a methodology able to produce power monitors that optimize the trade-off between the estimates accuracy and the introduced area overhead. To this extent, the proposed flow accepts as an additional input the maximum acceptable overhead that the power monitor can introduce; the methodology tries to identify and implement the most accurate power monitor with the available resource. To assess the effectiveness of the proposed methodology, the identified power monitors have been used to feed a fully digital control theory based power optimization scheme.

The rest of the thesis is organized as follow: Chapter 2 explain the background for online power monitoring solutions, Chapter 3 presents the state of the art about power meters, while Chapter 4 presents the proposed methodology for resource constrained power model identification. Chapters 5 and 6 show the experimental results for the power monitoring infrastructure and for the power optimization scheme, respectively. Chapter 7 describes the conclusions, underlining the main contributions of this thesis.

CHAPTER 2

Background

With the end of the *Moore's Law*, power consumption has emerged as a major obstacle to any advancement in computing technologies, limiting the performance of both embedded and high performance computing (HPC) platforms. On one hand, embedded and portable devices operate within tight power budget constraints to prolong their battery lifetime. On the other hand, HPC platforms, that aim to maximize the performance, are becoming hot-spot limited since the performance increase is restricted by both the maximum junction temperature and the cost of the required cooling systems. The power consumed by the compute unit cores, i.e. microprocessors and accelerators like GPUs, represents a major component of the power budget in such systems, particularly in embedded and mobile platforms. As a consequence, the research community has explored an increasing number of online power monitoring techniques aimed at optimizing the trade-off between power and performance [6, 17, 27, 52]. Unlike special-purpose hardware, general-purpose units like microprocessors and GPUs pose significant challenges both because they are inherently less power efficient and more difficult to characterize by means of closed power models, due to the strong dependence on the software workloads. Furthermore, because general-purpose units are meant to provide the largest degree of flexibil-

ity to software applications, they are usually overprovisioned in terms of hardware resources and a significant portion of their subcomponents stay idle, depending on the requirements of the specific application (or application phase) being run [43]. On the other hand, the well-known *dark silicon* problem makes it impossible to concurrently power all the parts of the computing device due to the impossibility of dissipating the full amount of generated heat [9]. In this scenario, online power-aware optimization techniques may play a key role in that they allow the dynamic tuning of the available computing capacity aimed at maximizing the energy efficiency under given thermal constraints.

However, the effectiveness of such optimization techniques is critically subject to the employed power monitoring method as the incorrect assessment of the power state of the system strongly affects the quality of the actuation with a negative impact on the power efficiency on the platform. At run-time, the power consumption can be read out as either a *direct measurement* or an *indirect estimate*. The *direct measurement* is achieved by means of analog sensors providing highly accurate power values at high temporal resolution. However, such solution suffers from a severe scalability issue that limits the deployment of more than few sensors even in complex designs and the use of complex mixed analog-digital design methodologies to implement them. This fact also prevents the identification of the thermal hot-spots at run-time, thus negatively impacting the reliability of the computing platform [22]. In contrast, the *indirect estimate* is achieved by means of a power model of the target architecture that is fed with the platform statistics at run-time. Such solutions are usually scalable and cheaper due to the possibility of implementing an all-digital power monitoring infrastructure that can monitor any part of the target. However, in general, they provide a less accurate power estimate compared to direct measurement schemes, thus motivating a huge research effort to bridge such accuracy gap.

All indirect power estimate schemes leverage the relationship between the power consumption and the internal switching activity of the target architecture to build a power model that is later used as part of the online power monitoring infrastructure. The possibility of using such relationship at different abstraction levels highlights a trade-off between the accuracy of the power estimate and the effort required to extract the required information from the target platform. *Power Monitoring Counter* solutions leverage the information obtained from the performance counters that are used as the proxy for the switching activity of the target platform [3, 37]. Such solutions preserve good scalability properties and guarantee a good accu-

racy of the power estimates and deliver significant flexibility, since they can be implemented *after-shipping*. However, the identified power model is software-implemented, thus the induced system-wide performance overhead, at run-time, is proportional to the temporal resolution of the power estimate time series. Moreover, the performance counter infrastructure represents an optional subsystem that can be either not implemented or limited in the number of the counters that can be concurrently read out. For example, commercial GPUs expose a variety of performance counters to software applications [6, 21], but they often limit the number of different values that can be read out simultaneously. Moreover, counters are normally coarse-grained, e.g. working at the *streaming multiprocessor* level in a GPU. In situations where not all cores within a streaming multiprocessor are active, e.g. because of nonuniform partitioning of the workload, the activity at the streaming multiprocessor level, as measured by the available performance counters, is not sufficient to capture the physical power behavior [6]. Fig-

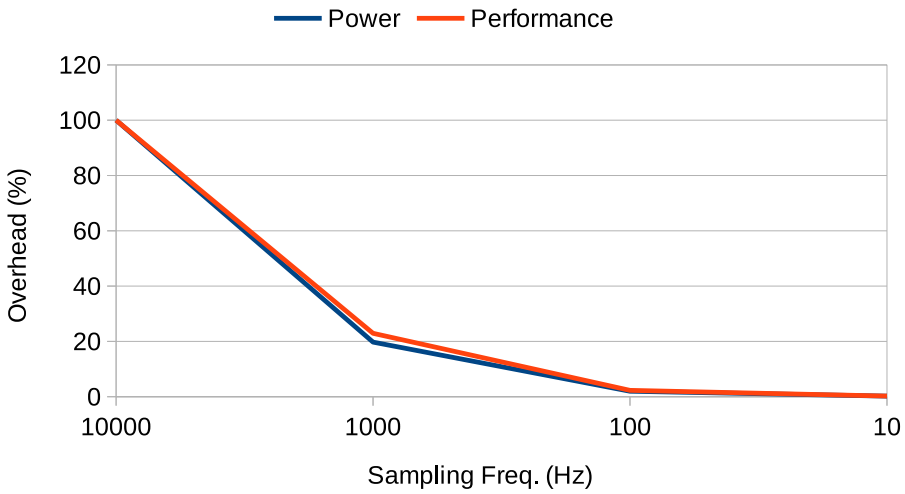


Figure 2.1: *The overhead introduced by a software power model based on performance counters is not negligible. In a quite small architecture, like the OpenRISC, the overhead on the execution time is about 2,29% @ 100Hz and 22,96% @ 1KHz, while the energy overhead is about 1,97% @ 100Hz and 19,73% @ 1KHz.*

ure 2.1 shows an example of how the execution of a software power model can impact on the system performances. The data used to draw Figure 2.1 have been obtained simulating a software power model running on a small RISC-based based processor ([23,28]), as a representative target device for the embedded processor family. As it can be noticed from Figure 2.1, the

overhead introduced by the execution of the software power model is about 2,29% @ 100Hz and 22,96% @ 1KHz, while the energy overhead is about 1,97% @ 100Hz and 19,73% @ 1KHz. In contrast, the most precise switching information correlating with the power consumption can be extracted by monitoring the toggle activity of each signal in the target platform at Register Transfer Level (RTL) or the corresponding activity of the driving logic cells. While the high number of signals to be monitored makes such solution infeasible, the possibility to remove the drawbacks of *Power Monitoring Counter* schemes motivates the proposal of a number of *all-digital power monitoring* solutions which try to optimally balance the number of monitored RTL signals and the accuracy of the power estimate [22, 24, 44].

The rest of this chapter is organized in two sections. Section 2.2 describes the performance counter based power monitoring methodologies, while Section 2.1 shows the switching activity based ones. Direct power measurements methodologies are not considered in this thesis for two reasons: first of all it is employed in a very limited number of devices with respect to the other two methodologies. Second, it is an analog electronic approach to the power monitoring problem, which is out of the field of my research.

2.1 Switching activity based power monitors

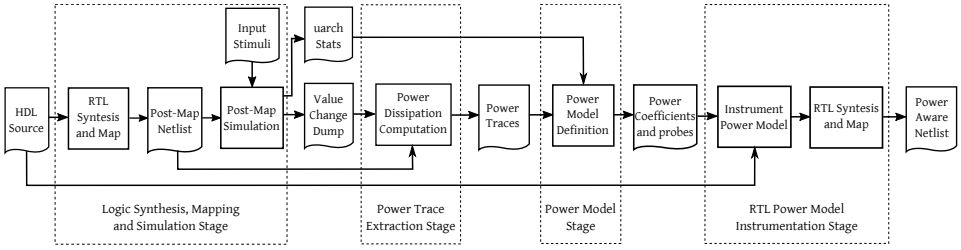


Figure 2.2: *Overview of a generic switching activity based power model identification flow.*

Figure 2.2 depicts a generic flow for a switching activity-based power model identification and RTL implementation of the corresponding power monitor. Starting from the HDL source code of the target device, the workflow produce another HDL description, semantically equivalent to the first one, but embedded with an online power monitoring infrastructure. This chapter analyzes the main features of each stage underlined in Figure 2.2.

2.1.1 Logic synthesis, mapping and simulation

The first stage that every switching activity-based power monitoring methodology has to perform is the implementation and RTL simulation of the target design. This stage takes as input the original RTL design (*HDL Source*) of the design under test, and produces as outputs the switching activity values (*uarch Stats*) for a selected set of wires of the design.

The *uarch Stats* file is organized as a matrix, containing the toggle count of each wire, for each sampling window. The toggle count of a *n-bit* wide signal can be computed in three different ways: Single Toggle Count (STC), Hamming Weight Count (HWC) and Single Bit Count (SBC). The first counting mode increments the statistic by one unit every time one or more bits of the signal toggle. The HWC counting mode increments the statistic by *m*, where *m* is the number of flipped bits of the considered signal. The last counting mode considers every *n-bit* signal as *n* single bit signals; for each signal the STC counting mode is applied.

The *Value Change Dump*(VCD) file also contains the toggles of every signals of the target device; it can be seen as a sort of summary of the values taken by each wire of the RTL description. This kind of file is widely used to obtain the power traces of an RTL simulation, and it is a common format for both ASIC and FPGA design flow tools, since, for every half-clock cycle, it can provide the value assumed by each wire of the design and the produced power value can have the temporal resolution of an half clock cycle.

The *HDL Source* file is fed to the Logic synthesis and mapping step, which is of paramount importance for the rest of the flow; in fact, it produces the post map netlist considering the technology library adopted, the pinout of the design and the timing and clock information. In order to obtain accurate power values from the hardware design flow tools, the design fed to these tools has to contain all the implementation details of the real devices, otherwise, the final embedded power monitor will produce inaccurate power estimates.

The netlist produced by the RTL synthesis and map step is then fed to an RTL simulator, together with some benchmarks (see *Input Stimuli* in Figure 2.2). Depending on the target device, the traffic provided to the RTL simulator can be different; for example, if the target is a CPU, the *Input Stimuli* set can be represented by the binary of a benchmark application, while in other cases the *Input Stimuli* can be a synthetic traffic generated by a testbench. It is worth noticing that the choice of the *Input Stimuli* is very important: non appropriate benchmarks of synthetic traffic can stim-

ulate only a part of the netlist, thus the switching activity information produced by the *Logic Synthesis, Mapping and Simulation Stage* can be incomplete. Generating synthetic traffic is a suitable way to stress each wire of the design, even if, especially in complex designs, generating testbench able to produce the desired traffic is very challenging. For what concerns the benchmarks, instead, it is important to chose a set of application able to stress the whole architecture. The problem here is the simulation time: RTL simulation are very complex to be executed and they require many hours, especially for post map simulations. Most of the times, running complete applications from the most famous benchmark suites like SPEC [20], SPLASH [32] or PARSEC [2] is unaffordable, thus many works in the state of the art [22, 24, 47] adopt microbenchmarks tailored to stress a specific component of the architecture. In this way, with a small set of microbenchmarks, it is possible to get the switching activity of the entire architecture, employing a limited amount of time for the simulations.

2.1.2 Power traces extraction stage

The *Value Change Dump* file produced by the *Logic Synthesis, Mapping and Simulation Stage* is fed to the *Power Trace Execution Stage* to compute the power dissipation. Starting from the VCD file, the power estimator of each hardware design toolchain computes the power consumption of the device, simulating the execution of the provided benchmark. Usually, the power estimation tools expose a parameter that allows the tuning of the temporal resolution. As mentioned before, the VCD format reports the value of each wire of the architecture every half clock cycle, so the maximum allowed temporal resolution is an half clock cycle. Changing the value of the temporal resolution parameter it is possible to augment the sampling window; from an analysis of the state of the art, it emerges that the most adopted temporal resolutions range from 50 to 500 microseconds. However, not every power computation tool provided within an hardware design tool offers the possibility to get a power trace, given a VCD as input. For example, Vivado [42], an FPGA hardware design flow toolchain from Xilinx, proposes as power analysis tool *report_power*, which only provides the average power consumption of the design under test, for the entire simulation. To overcome such limitation, authors in [44], propose a methodology to sample the VCD file and split it into several chunks, as if there were several VCD files. Then, the power trace is computed by feeding the power analysis tool with the sampled VCD files. This approach is not bounded to a specific toolchain and it can be used for every power analysis tool that does not natively provide the computation of a complete power trace. It is

important noticing that all the tools for power analysis within an hardware design flow toolset provide estimates for both the static and dynamic power consumption. Switching activity based power monitoring solutions have to take into account both the power contributions: the dynamic power is modeled leveraging the switching activity data, while the static power is usually modeled with a known term in the mathematical model.

2.1.3 Power model identification stage

The power model identification stage takes as inputs the power traces and the toggle count statistics(see emphPower Traces in Figure 2.2) and produces as output the mathematical formulation of the identified power model *Power Coefficients and probes* in Figure 2.2). The goal of this stage is to find the mathematical formulation that best models the power consumption of the target device, given the switching activity of the architectural wires. In order to measure the accuracy of the identified models, the state of the art proposes several metrics, such as the Root Mean Square Error (RMSE), the Mean Average Error (MAE) and many others. Chapter 4 will describe these metrics in an exhaustive way.

The state of the art proposes basically two different procedure for the power model identification algorithm, depending on how the target device has been designed.

The first one is an hierarchical approach, where the power model identification algorithm starts from the top module of the design hierarchy and tries to identify each level of the hierarchy tree, finding the one that produce the smaller prediction error. The final power model will be built as the sum of the power estimates of all the modules of that level plus an estimate of the power consumption of the glue logic, i.e., the digital logic that connect the modules together and manage the relations with the higher hierarchical level modules.

The second approach considers the RTL design as a single hierarchical level. Even if the design is organized more levels, this approach considers them as flattened in a single one, and the power model identification algorithm will select the wires, independently from the original hierarchy level.

The two aforementioned approaches present both strong and weak points. The hierarchical approach is more flexible, as it allows the possibility of monitoring the power consumption of single or a set of submodules within the design hierarchy. However, most of times, this approach presents higher implementation overheads, since, with respect to the second approach, it uses more wires to build the power model. The flattened hierarchical ap-

proach is more resource saving, since it is not forced to identify the single submodule, but it can consider the whole design with all the available wires. Nevertheless, it is less flexible, and no power estimates of single submodules can be provided.

$$P_{Static} = V_{dd} \times I_{supply} \tag{2.1}$$

$$P_{Dynamic} = \alpha \times C_L \times V_{dd}^2 \times f \tag{2.2}$$

The power consumption of logic gates can be described with two models, static and dynamic. Equation 2.1 models the power consumption of a circuit under static conditions, i.e., when the logical values of the inputs are fixed to 0 or 1. Under dynamic conditions, the inputs changes their logical state, so the transistors between the power supplies will be in an active state, or they will continuously change values, requiring power to charge and discharge the output capacitance. Thus, the dynamic power consumption will depend on the switching activity of the architectural wires. Equation 2.2 presents the mathematical model that describes the dynamic power consumption. From this equation it emerges that there is a quadratic dependence between the dynamic power consumption and the *Voltage Drain Drain*(V_{dd}), while the total capacitance (C_L), the operating frequency (f) and the switching probability (α) contribute linearly. It is important noticing that, even if Equation 2.2 presents a quadratic term, most of the state of the art articles about switching activity based power monitors use linear mathematical formulations to model the power consumption of a digital CMOS based device. This choice is due to mainly two factors: first of all, as it can be noticed in Equation 2.2, α contributes to the dynamic power consumption linearly, so even the switching activity data collected from the RTL simulation stage should contribute linearly to the power estimates. The second reason that push many authors of RTL online power monitoring papers to adopt a linear mathematical model is the easiness of implementing the corresponding power model into the RTL description. It is important to remember that the implementation of the power model introduces a resource and power overhead with respect to the original device; the more the mathematical model is complex, the more this overhead is high.

One of the most frequent problems in the mathematical model identification field is the choice of the model predictors. Especially in switching activity based power models, the final formulation should use the minimum set of predictors, as the more predictors are considered, the higher the overhead due to the power monitor implementation is. For this reason the set of statistics produced by the RTL simulation stage has to be filtered out from

those statistics showing a low correlation with the power consumption or those showing an high multicollinearity. In statistic, multicollinearity is the property of a predictor to be linearly estimated by a set of other predictors. The more the accuracy of this estimate is, the more the multicollinearity value is; a model built with predictors that show an high multicollinearity value will be very accurate with training set data, while it could show a low accuracy with test and validation set data.

2.1.4 RTL Power Model Instrumentation Stage

The last stage of a generic flow for the design of a switching activity based power monitoring architecture is the implementation of the identified power monitor into the original RTL description of the target device. As it can be observed from Figure 2.2, the *RTL Power Model Instrumentation Stage* takes as inputs the mathematical formulation of the power model, identified in the *Power Model Stage*, and the original RTL source code. The output of the *RTL Power Model Instrumentation Stage* is always an RTL description, semantically equivalent to the original one, but augmented with online power monitoring capabilities (*Power Aware Netlist* in figure 2.2). The RTL switching activity based power monitors are usually built adopting three basic blocks: counters, multipliers and adders.

Toggle counters: The *Toggle counter* is a synchronous module within the power monitoring infrastructure that measures the switching activity of a specific wire, i.e., it counts the number of toggles of the wire in a certain time window. It can both measure the switching activity with single counter or hamming weight counting mode, depending on the model identified in the *Power Model Stage*.

Power Multipliers: The mathematical formulation of the identified model can be summarized as a sum of multiplications between the toggles counts and coefficients; depending on the order of the identified model, this multiplications can be more complex, but, given that the majority of the power models are linear, it is most of the times a simple multiplication between a number of toggles and a coefficient. The *Power Multipliers* is the component of the power monitor in charge of performing these multiplications. It has at least two inputs which are the toggle count and the model coefficient, and one output represented by the result of the multiplication. Depending on its microarchitectural structure, the multiplier can have as input also the clock and the reset signal. It is important noticing that, especially in small target designs, a combinatorial multiplier can impact a lot on the critical path of the circuit. The internal structure of the *Power multiplier* greatly

depends on the format of the coefficient provided as input: since it is, usually, a decimal number, it can be represented both as a floating point or a fixed point value. Depending on this choice, the multiplier has to be designed in order to accept one between the two formats; please note that the implementation of a floating point unit within a power monitor will increase the implementation overhead significantly. For this reason, the adoption of a fixed point format to represent the model coefficients is preferred.

Power adders: Considering polynomial models, the formulation of a power model can be summarized as a sum of products. Thus, every multiplier provides a power contribution term that has to be logically summed with the others in order to get the final power estimates; for this reason a *Power adder* is added into the design of the power monitor, to sum all the results of the instantiated multipliers. The interface of a power adder is usually made by n inputs, depending on the number of monitored signals, and one output providing the final result. Given that this adder has only to perform additions (and subtractions), it is usually implemented as a combinatorial module.

The instrumentation stage first instantiates a power counter and a power multiplier for each signal considered in the power model, then introduces a power adder to collect all the power contributions and to produce the final power estimate.

Once the power adder is instantiated, the last implementation choice is the mechanism to expose the power estimates to the user. From analysis of the state of the art, two main solution can be found. The first solution leverages a input/output mechanism, e.g., a serial port, to provide the power estimates out of the device. The second mechanism leverages a memory-mapped register to store the power estimates; in this way, from the software it is possible to read the power consumption of the chip and use it to, for example, feed a power/energy optimization mechanism.

2.2 Performance counter based power monitors

Figure 2.3 shows a generic flow for performance counter based power monitoring design flow. It takes as input a set of benchmark applications *Input stimuli* as produce as output a binary file implementing the software power monitor. This type of methodology, instead of using the switching activity of the architectural wires, as described in the previous chapter, adopts the performance counter data as model predictors. The performance counters are special memory mapped registers implemented within the CPU, containing information about a set of selected architectural events. For ex-

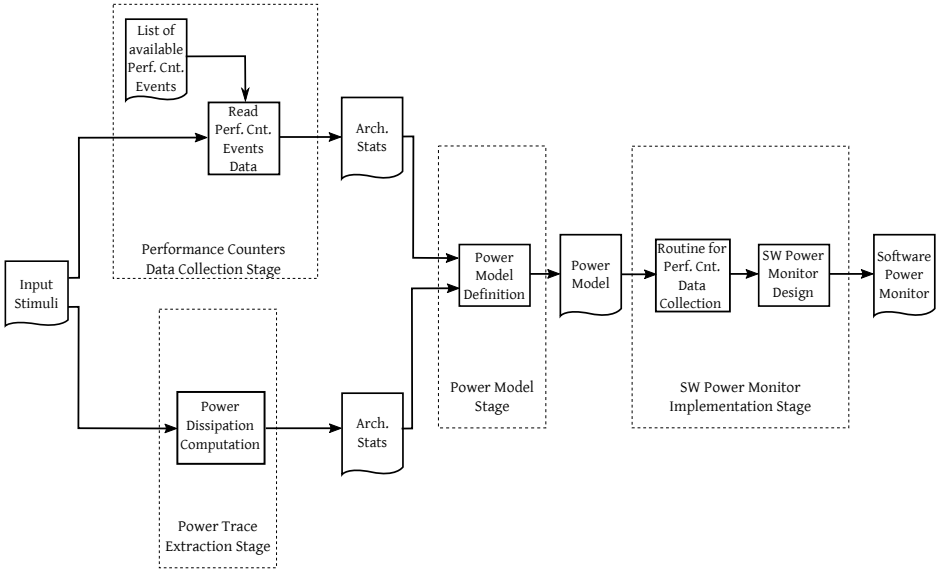


Figure 2.3: *Overview of a generic performance counter based power model identification flow.*

ample, a performance counter can measure the number of committed instructions, the number of cache hits or misses or the number of accesses to the central memory. Typically, they are used to check the performance status of an architecture, as well as to feed run-time performance optimization schemes. The state of the art, present a great number of works that use the performance counters to predict the power consumption of a computing architecture. However, it is important noticing that, as the main limitation of switching activity power monitor was the availability of the RTL description of the target device, the main limitation of performance counter based power monitors is the availability of the performance counters. In fact, it is a feature not implemented in all the computing architectures and, especially in small embedded devices, where the processors are small and the computational resources are limited, the performance counters are not implemented, since they will represent an additional resource overhead.

This chapter summarizes the main features of each stage represented in Figure 2.3.

2.2.1 Perf. Cnt. Data Collection Stage

The first step of a generic performance counter based power monitor design consists in the acquisition of the required data for the power model

identification. These data are the power traces of the execution of a set of benchmarks, and the performance counter data, used as model predictors. Especially when dealing with a target device running an operating system, it is important that the power traces and the performance counters data have to be collected simultaneously, as the operating system can change the temporal allocation of the running threads, thus introducing discrepancies from tho consecutive runs of the same applications. The collection of the power traces and the performance counters data are organized in Figure 2.3 in two different stages, only for seek of clarity. This subsection describes the performance counters data collection stage, while Subsection 2.2.2 will describe the power traces acquisition.

Differently from the switching activity based power monitor design flow, presented in Section 2.1, the performance counter based one works after production. It means that the RTL design is no more needed, as it works directly on the target device. For this reason, instead of using an RTL simulation to collect the predictors data, it reads the performance counters data through a software routine that periodically performs a set of system calls to read the performance counters. It is important in this stage to have a list of the performance counters implemented in the target architecture, together with the scheme of the hardware mechanism that provide the performance counters data to the software. In fact, in most of the architecture, especially those exposing a great number of performance counters, with a single system calls only a subset of the available performance counters can be read; thus, many system calls have to be performed to get the required data. This limitation have to be considered also in the *Power Model Stage* as the final power monitor should perform the minimum number of system calls to get the data of all the model predictors. The *Arch. Stats* file reported in Figure 2.3 is the output of this stage and it contains, for each executed benchmark, the data of all the available performance counters.

2.2.2 Power Trace Extraction Stage

The *Power Trace Extraction Stage* is the step of the flow in charge of collecting the power consumption of the target device for the executions of each benchmark. As mentioned before, the performance counter based power monitor design methodologies work on the physical devices, instead of working on their RTL descriptions. For this reason, to extract the power traces, instead of using a post-map simulation, a direct measure on the target device has to be performed. The measurement of the power consumption of a digital device can be performed in two different ways: direct current measure or Electromagnetic Emission (EM) measure. The

first approach leverages a shunt resistor plugged along the power line of the device and measures the voltage drop on it through a multimeter or an oscilloscope. Knowing the value of the resistor, the current can be computed through the ratio between the measured voltage drop and the resistor value. The other approach leverages the electronic emissions produced by the chip to extract the power consumption. To this extent, an electronic emission probe is placed close to the chip and through an oscilloscope the values coming from the probe can be collected. It is important noticing that, since the signal amplitude is tight, a signal amplifier is often required to ease the sample process and make it more accurate. It is also important to design the amplifier in order to be able to have the precise value of the gain, otherwise the power measures cold be scaled by a proportional factor.

The sample frequency of the power traces should be the same as that of the predictors (performance counter data). Unfortunately, especially in the performance counter based power monitor design, the predictors are sampled much slower with respect to the power traces, due to the delay introduced by the system calls to read the performance counter values. For this reason the state of the art proposes a solution that allows the sampling of the power traces at a frequency higher than the performance counter collection and then computes the average of the power samples to reach the same number of samples as the performance counters data.

2.2.3 Power Model Stage

The *Power Model Stage* for a performance counter based power monitor design is very similar to the one presented in the previous chapter for switching activity based power monitors. In fact, in both situations, the inputs of this stage are the power traces and the corresponding architectural statistics, and the goal of the *Power Model Stage* is always the identification of a mathematical model able to predict the power consumption of the computing devices with an high accuracy, while using the minimum number of model predictors. There are also similar constraints for switching activity and performance counter based methodologies: both them have to select with care the number of predictor since the add of a statistic in the model cause an increase of the resource overhead from one side, and of the performance overhead from the other.

The main difference between performance counter based and switching activity based power monitor design methodologies is the choice of the adopted mathematical model. Differently from what has been described in section 2.1, where the liner model family was the common choice due to a low implementation overhead, here the power model is implemented in

software, so, the impact of a more complex mathematical model formulations on the overall system performance has to be minimized. This means that the adopted model family should be the one showing the best trade-off between accuracy, and performance overhead. For example, in [22], authors propose a comparison between different model identification algorithms, such as linear regression, Multivariate Adaptive Regression Splines(MARS) and Neural Network, to find the algorithm that best predicts the power consumption of the target device, with the corresponding performance overhead, measure ad the number of clock cycles use to compute a single power estimate. For that particular case, experimental results show that the model obtained through the MARS approach is the best one in terms of accuracy, but it shows up to 4x of clock cycles adopted to compute the power estimates.

2.2.4 SW Power Monitor Implementation Stage

As mentioned before, two of the main differences between switching activity based and performance counter based power monitoring infrastructures, are the implementation and the moment when the power monitor is added into the target device. For what concerning the implementation the different between the two approaches is that a switching activity power monitor is an hardware component, while a performance counter based power monitor is a software component. Furthermore, for switching activity based methodologies, the power monitor is added directly into the RTL description of the target just after the end of the design stage, before the production of the chip, while for performance counter based solutions, the power monitor is a software routines that can be added in any moment after the production of the chip. The software application that implements the power monitor can be organized in two parts: data collection and power estimates computation.

Data collection: the power model identified in the *Power Model Stage* is made by a list of performance counters, selected as model predictors, a list of coefficients bounded to each predictor and a function representing the mathematical model. The first thing that the power monitor should do is the collection of the data from the performance counters. To this extent, the power monitoring routine performs some system calls to read the performance counter registers; as mentioned before, not all the performance counter registers can be read simultaneously, so, multiple system calls have to be performed to get all the required data.

Power estimate computation: once the predictors data have been ob-

tained, the next step is the software implementation of the mathematical algorithm to compute the power estimates. To this extent, the algorithm have to be written in any programming or scripting language and the identified model parameters have to be associated to the corresponding predictors.

The obtained software code has to be compiled and run periodically, to achieve a continuous online power monitoring of the target platform.

CHAPTER 3

State of the Art

The continuous increase in performance requirements which has been imposed by current applications motivated the widespread adoption of multi-core architectures ranging from embedded to High Performance Computing (HPC) solutions. In this scenario, power consumption represents a major challenge for the entire *computer continuum*. The performance of both embedded and high-end computing architectures is in fact limited by both the available energy budget and the power related technology constraints. Servers' processors are typically constrained by both the max junction temperature and the cooling systems, thus imposing energy-aware server consolidation techniques [33] to limit the average power consumption while optimizing the energy-performance trade-off. Conversely, the limited energy budget of battery powered embedded systems imposes the design of complex energy-performance resource allocation schemes [18] and aggressive power saving infrastructures [45, 46].

To this extent, providing accurate power estimates to both the software and the hardware architects is of paramount importance to enable *i)* power-aware architectural optimizations at design time and *ii)* a continuous monitoring of the power profile of the architecture, thus enabling run-time schemes for power-performance resource allocation. The majority of available Computer-

Aided Design (CAD) tools already support some sort of power modeling feature to fulfill *i)*, while *ii)*, which is less standard, is still evolving to cope with the ever increasing constraints in terms of accuracy and low latency imposed by the applications.

The online power monitoring emerged as a viable solution to fulfill requirement *ii)*. It leverages either the *performance counters* [37] or the switching activity of the architectural wires to provide a run-time power estimate of the target architecture. The use of performance counters demonstrated a good fit between the predicted and the real power consumption of the target. It also provides a low cost and flexible solution to the run-time power monitoring problem while introducing two drawbacks. First, performance counters are not primarily intended for power modeling purposes and, even worse, they cannot be always available, thus limiting the feasibility of the related power monitoring solutions. Second, the update of the power estimate is achieved through a software routine that consumes CPU time and energy. The overheads become more severe with both the increase in the update frequency and the decrease in the computational power of the target architecture.

To solve the limitation imposed by performance counter based methodologies, switching activity based power monitors emerged as a viable solution to provide fast and accurate power estimates for those digital devices not embedding the performance counters or constrained by tight performance requirements.

This chapter shows the main methodologies to identify and implement a power monitoring infrastructure for different class of digital computing systems.

3.1 Direct measurements

A first approach to the power monitoring problem is represented by the real measures solutions. Leveraging analog power meters, direct measurements are performed on the target platform. These power meters can be external [15] or internal [1] with respect to the target device to be monitored. In [15] authors present a scalable power observation tool that enables power and energy measurement for any kind of nodes adopted in a sensor network. With this monitoring system, every node can communicate to the main node their power and energy consumption, enabling the possibility to apply energy saving policies. The power measurements are performed using shunt resistors plugged on the power supply lines of the node. A voltage meter measures the voltage drop on these resistors to compute the

power consumption. This approach results very accurate, since the power consumption is measured and not estimated; in addition to that, this solution is clock independent, so it can normally work even if power saving mechanisms are applied to the target devices (i.e. DVFS). Furthermore this approach can reach a very tight temporal resolution, enabling the possibility to apply also thermal controls schemes. Unfortunately, this approach presents mainly two drawbacks: first, they return only the total power consumption, thus making impossible to measure both thermal hot-spots or the power consumption of specific subsystem. Second the shunt resistors dissipates power itself, so possible power overhead can be introduced.

[1] proposes a real-time, on-chip power sensor that estimates load currents and on-chip temperatures concurrently. The power monitoring infrastructure is implemented in CMOS technology directly into the processor package. The main idea behind this work is to implement a on-chip current meter, that monitors the power delivery network of the chip, so it can be seen as a digital implementation of a shunt resistor. Experimental results show a good sampling rates (nearly every ten clock cycles) both for power and thermal measurements. The main drawbacks of this approach is represented by the scalability issue, that prevents deploying more than few of them in a complex architecture.

3.2 Performance counter based methodologies

Another approach to the power monitoring problem is based on the tight correlation between some hardware events and the power consumption of the target device. For this reason, a great portion of the literature about power modeling and monitoring is represented by software solution that leverages the performance counters to collect information useful for generating power estimates. Performance counters are special registers instantiated within modern SoCs that collect information about the hardware activity. Most popular performance counters collect, for example, the number of hits and misses in the cache memory, the Clock per Instruction (CPI), the number of floating point operations and other architectural information about the execution of certain applications. These counters can be read from the operating system (or a supervisor software in case of bare metal applications) through a dedicated hardware support. Generally, a power model is computed in software, taking as reference a measured power trace of the same benchmark execution. A software routine, implementing the power model, continuously collects data from the selected performance counters and compute the power estimates. Starting from the 1990s, many

scientists worked on this topic; one of the first structured approaches is presented in [13]. Here, authors propose a per-unit power estimation techniques, leveraging the information coming from the performance counters as model predictors. The resulting tool offers a run-time overall power estimates for Intel Pentium 4 processors, and also provides power breakdowns for 22 of the major CPU sub-units. The resulting power model accuracy is validated on the SPEC2000 benchmark suite, together with normal desktop workloads. Despite, this work has a main drawback, represented by the very slow prediction rates (per minute), it set-up the base for many further research on this field.

Further investigations on this approach were proposed in [4]. Here authors propose the use of microprocessor performance counters for online measurement of complete system power consumption. While past studies have demonstrated the use of performance counters for microprocessor power, this work is the first that creates power models for the entire system based on processor performance events. The proposed methodology leverages the "trickle-down" effect of the architectural events to estimate not only the power consumption for the SoC, but also for other elements of the entire computing system, such as memory, disk, chipset and I/O. In this article authors show a very interesting analysis on how on-chip events (i.e. DMA transactions, cache miss) can provide information on the activity and, consequently, the power consumption of off-chip components. For example, a Last Level Cache (LLC) miss means an access to the main memory. The good experimental results of this work open a new perspective in this research area, since they assess the possibility to estimate system power consumption of an entire computing system, without the need for additional power sensing hardware.

Asymmetric multi-core architectures have recently emerged as a promising alternative to provide efficient computing platform, optimizing the energy-performance trade-off. Typically, this kind of architectures, are equipped with two or more cores, some targeting performance, others targeting low power computation. Tasks are dynamically allocated to each core, depending on the adopted energy-aware policy. The state of the art is largely populated by power monitoring methodologies, targeting this family of processors. One of the first works targeting this kind of processors is [27]. Here authors propose a power model based on performance counters data, able to provide accurate power estimates for both the high-performance and the low-power cores of an ARM big.LITTLE [14] Multi-Processor System on Chip (MPSoC). Experimental results show an average estimation error of about 9%.

Further investigation on the possibility to identify an accurate power model for asymmetric processors are presented in [37]. Here, authors propose a mathematically rigorous novel methodology for identifying accurate run-time power models for mobile and embedded devices, leveraging performance counters. In particular, they put particular attention to the stability of the identified model, showing how this aspect is of paramount importance and how a stable power model can reduce the number of monitored counters without losing accuracy. The main contribution proposed in this work is the analysis of the multicollinearity among the model predictors. Previous approaches, in fact, used to consider a lot of performance counters to build the power model. This implies two main drawbacks: first, only few counters can be monitored at the same time; thus, considering too many counters implies a reduction of the prediction rate. Second, many counters can be auto-correlated, so they can provide the same information to the power model; this could introduce a model instability, with a huge accuracy loss between the training and the test set. In this article, authors provide a formal method to reduce the multicollinearity; this allows also a 100% reduction in experiment time, while losing only 0.6% in accuracy. Furthermore, this work highlights and addresses the problem of heteroscedasticity in power modeling. The identified model is then implemented and validated on an ARM big.LITTLE architecture.

[36], instead, is focused on a little bit different problem. Instead of using the performance counters to predict the power/energy consumption, in this work authors use them to predict the temperature. The estimated temperature feeds a Dynamic Thermal and Power Management (DTPM) algorithm, to compute the power budgets for the eight cores of the considered big.LITTLE processor. The power management infrastructure of the device will adjust the values for voltages and frequencies to maintain the power budget. Experimental results show an average accuracy for the temperature estimates of 3%, while the DTPM algorithm provides around a 6% reduction in temperature variance and a 16% reduction of the total platform power. A similar approach is adopted in [35]; here, authors adopt a set of microbenchmarks to identify a performance counter based power model and use the monitor estimates to feed a power-aware thread scheduler. A slightly different approach is presented in [30]. With this work, authors try to identify a set of performance counter able to predict the power consumption not only for a single architecture, but for a large set of CPUs. After a preliminary analysis, intended for identifying a set of common performance counters that can be implemented on both HPC and embedded CPUs, they try to use them to identify a power model that can be used

to estimate, with sufficient accuracy, the dynamic power consumption of processors with varying microarchitecture. The validation of the proposed methodology was performed against two CPUs, an Intel Atom and a Intel Nehalem, representing low-power and HPC architecture families, respectively. Experimental results show that the identified counters are shown to be effective in predicting the dynamic power consumption across processors of varying resource sizes, achieving a prediction accuracy of 95%.

3.3 Switching activity based methodologies

Another approach to the power monitoring problem is represented by those methodologies that leverage the tight relationship between the power consumption of a digital device and the switching activity of the architectural wires.Starting from the beginning of 2000, several work addressing this problem can be found in the literature. One of the first work on all-digital power meters is presented in [5]. Here authors try to use the switching activity of the architectural inputs and outputs wires to predict the power consumption of hard and soft hardware macros. The identified model, obtained through a linear regression algorithm, is used only for an offline power characterization at design time.

One of the first online implementation of an all-digital power monitor is proposed in [25]. In this article, authors propose a novel processor, able to auto monitor its power consumption. To achieve this goal, authors adopt a mixed approach. In fact, they use as model predictors, both the switching activity of some architectural wires and the information coming from ad-hoc hardware counters that are similar to the performance counters. In other words, they collect the performance counters data directly in hardware, with ad-hoc counters. A linear regression algorithm has been adopted to extract the power model and the estimates are provided to the software through a dedicated serial port. Experimental results show a very good accuracy of the power estimates, with a prediction error below 2% and a area overhead below 5%.

A more general approach is described in [31]. In this work authors propose a platform-independent power model identification methodology. The key idea of this work is to identify set of wires able to provide accurate power estimates over different FPGA model. A set of FPGA-specific parameters has been identified, to adapt the model to different FPGAs. In other words, the model predictors remain the same across different platforms but the model coefficients change. Although this solution speeds-up the model identification process when dealing with different FPGAs, a non-

negligible estimation error (around 15%) is introduced.

A first formal definition for all-digital power models can be found in [16]. This work, like [37] for performance counter based methodologies, proposes a mathematical approach to the power model identification problem for all-digital power monitoring infrastructure. The approach adopted here can be considered as an all-digital version of the performance counter based methodologies. In fact, instead of monitoring the toggle activity of architectural input and output wires, authors decided to monitor the switching activity of the performance counters, thus implementing a hardware collection method for performance counters data. This allows to collect data that precisely describe the actual workload of the architecture. In order to reduce the multicollinearity and the area overhead due to the power monitors, authors apply a Singular Value Decomposition (SVD) algorithm to abstract the principle components of relationship between register toggling profile and accurate power waveform. The model is then identified using machine learning algorithms and automatically instrumented into the target RTL design, allowing a cycle-by-cycle power dissipation estimation. Despite a good accuracy of the identified model (error below 3%), the main limitation of this work is the use of performance counters, not present in all architectures, especially in small embedded CPUs. This mathematical approach revealed very interesting, such that many other works adopt the same methodology. For example, [22] presents a deeper analysis on this argument. Starting from the work proposed in [16], it perform a detailed evaluation on how to measure the switching activity of the considered wires. Authors identified three ways do measure the toggle of a signal: single count, hamming weight and single bit. The first one increments the count by one unit every time at least one bit of the signal change. The second counting mode increments the count by the number of bits that change value. Lastly, the third considers every bit of a signal separately. Thanks to this differentiation, they succeed in extracting the different contribution to the power consumption for each considered signal. Experimental result demonstrate that the proposed methodology can reach high level of accuracy (about 1% of accuracy error), with a minimal impact on the area overhead. A similar approach has been adopted in [44]. Instead of targeting complex architecture like those considered in [22], here authors considered a small RISC architecture and show how too complex power model introduced large area and power overhead. In this work a linear regression based power model is extracted considering only the single count mode, since hamming weight counting mode requires more LUTs. Furthermore, a novel accuracy measure has been introduced. In the state of

the art the most adopted metric to measure the model accuracy is the Root Mean Squared Error (RMSE); however, this metric only consider the absolute difference between the estimates and the reference. In this work authors introduced the Mean Relative Error (MRE) which consider the relative distance between the estimates and the reference, giving a more accurate metric of the model error. The analysis about the different counting modes has been deepened in [24]. Here, authors show that the best way to measure the signal switching activity is to consider each bit separately. The reason why this choice can be considered twofold: from one side different bits in the signal can influence different data- or control-path of the design and thus, can introduce different contribution to the overall power consumption. From the other side, considering only a subset of the signal bits can highly reduce the area overhead introduced by the implementation of the power monitoring infrastructure. Experimental results show that the overall model accuracy hasn't been affected too much (it remains below 3%). A deeper and formal analysis about the different methods to measure the switching activity is presented in [47]. Starting from the considerations proposed in [22] and [24], this work proceeds in two directions: from one side it proposes a deeper analysis about the three counting modes, while, from the other side, it try to match each counting mode to a different type of signal (data or control). From the first analysis it emerges that the single bit counting mode can provide wrong results. For example, considering a subset of a 32 bit memory address signal of a CPU, it is sufficient to change the linker script to invalidate the power model. The second analysis, instead, assess the effectiveness to consider the control signals with a single count mode, while the data signal with a hamming weight mode. Authors show the efficacy of the proposed methodology targeting a multicore multithreaded CPU; the experimental results show the same accuracy as the one obtained in [22] and [24], without introducing possible errors like the one mentioned before.

CHAPTER 4

Methodology

With the incoming of IoT world for the first time we have much more data than the computational capacity to process them. In fact, huge HPC centers are no more able to manage the huge amount of rough data provided by sensor spreaded among the IoT environments. In this scenario, edge computing emerges as the most promising solution to face the challenges proposed by this computing revolution. With this new computing paradigm, data elaboration is no more performed in a single step, but it is distributed among the different layers of a huge computing environment. Starting from the edge, the IoT sensors are no more only sensors but they are complex computing devices able to collect, process and transmit data, at least. Even if edge devices are often equipped with small processors with limited computational capacity, they run complex applications, implementing algorithms ranging from data mining to artificial intelligence field. Furthermore, since they are mostly battery powered, energy efficiency emerges as one of the most important metric to be considered, from the hardware design step of the device, to the implementation of run-time software optimization policies. In this scenario, online power monitoring emerged as the key enabling factor that allows run-time power optimization techniques to be effective. As described in Chapter 2, power monitors can be implemented both in

software or in hardware. This thesis is focused on hardware solutions for online power monitoring infrastructures. From the state of the art analysis described in chapter 3, it emerges that all the proposed solutions for hardware power monitors aim to the development of methodologies offering the highest accuracy as possible, without considering the resource overhead introduced by the implementation of such monitors. It is important noticing that, in the world of IoT devices, characterized, as mentioned before, by small computing units, the overhead introduced by the additional resource used for implementing the power monitors becomes non-negligible. For example the power overhead due to the hardware monitors can overcome the amount of power saved thanks to the run-time optimization techniques. For this reason, during the power model identification, it is necessary to take into account not only the accuracy of the power estimates, but also the resource overhead introduced by the implementation of the power monitor. This thesis proposes a methodology for the identification of resource-constrained power models and automatic implementation of the corresponding power monitoring infrastructures for generic RTL designs.

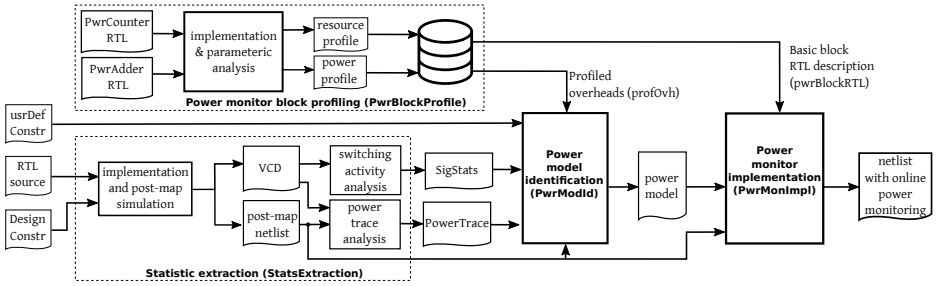


Figure 4.1: *Overview of the proposed resource-constrained automatic power model identification and instrumentation flow.*

Figure 4.1 depicts the proposed toolchain to automatically generate a hardware-level resource-constrained power monitor for generic computing platforms. Starting from the hardware description of the target device (`RTL-source`), its corresponding set of design constraints (`designConstr`) and the user-defined constraints (`usrDefConstr`), the flow outputs an hardware description file containing the target design augmented with the power monitor (`netlist with online power monitoring`). It is worth noticing that the design constraints are expressed in terms of the standard timing, e.g., required operating frequency, and physical, e.g. pinout, requirements for the implemented computing platform. In contrast, the

user-defined constraints allow the user to specify the number of allowed resources to implement the power monitoring infrastructure.

The design and implementation of the power monitor is organized in two separate steps: *i)* power model identification (`PwrModId`), and *ii)* power monitor implementation (`PwrMonImpl`). The `PwrModId` stage identifies a power model that ensures the smallest accuracy error within the resource budget, by leveraging: *i)* the switching activity of the signals, *ii)* the power consumption of the computing platform, *iii)* the user imposed resource constraints, and *iv)* the profiled overhead of the basic blocks used to realize the power model. Finally, the `PwrMonImpl` stage augments the RTL description of the computing platform with a power monitoring infrastructure which implements the identified power model.

4.1 Data collection

The first step for every power model identification flow, is the collection of the required data, i.e., the power consumption of the target device and the model predictors. Section 4.1.1 describes the power monitor block profile stage, that produce the components of the power monitor, together with the corresponding resource information, while Section 4.1.2 presents the solutions adopted in this thesis to obtain the toggle counts and the power consumption of the considered architecture leveraging the power estimation tools provided by Xilinx.

4.1.1 Power monitor block profiling

The *PwrBlockProfile* is the stage of the power model identification flow in charge of producing the RTL description of the basic blocks of the power monitor (see *pwrBlockRTL* in Figure 4.1), and the corresponding profiled overheads (*profOvh*). The RTL descriptions will be used in the power monitor implementation stage (*pwrMonImpl*), while the resource and power profiles will be considered during the power model identification stage (*PwrModId*). The power monitor block profile stage starts with the implementation and the parametric analysis of the power counter and the power adder basic blocks. In fact, the power counter can be parameterized with the dimension of the signal to be analyzed, the Switching Activity Counting Mode (SACM) adopted, the sampling windows (measured as a number of clock cycles) and the output signal width, while the power adder can be parameterized with the number of signals to be summed and the width of the input and output signals. For each configuration of both the power counter and adder an out of context synthesis and map process is performed, and the

resource occupations and power consumption are reported in the *resource profile* and *power profile* files, respectively. During the power model identification stage, these files are considered, in order to check that the identified power monitor respects the user constraints.

4.1.2 Statistic extraction

The *StatsExtraction* stage is in charge of collecting the power traces and the switching activity information of the target device. In fact, this stage takes as input the *RTL source* and the *DesignConstr* file and produces as output the *SigStats* and the *PowerTrace* files containing the toggle counts of all the architectural wires and the power trace of the executed benchmarks, respectively. This stage starts with the implementation of the target design, accordingly to the design constraints (physical and timing). The obtained post-map netlist is used for a set of RTL simulations, using different testbench, in order to stress all the architectural wires. From the *implementation and post-map simulation* step, this stage extracts a Value Change Dump (VCD) file and a post-map netlist; the VCD file is used to compute the toggle counts of the considered architectural wires, while the post-map netlist is fed to the power estimation tool that computes the power traces. As mentioned in Section 2.1.2, rarely an FPGA design tool suite provides the possibility to obtain a temporal power trace of the simulation of an RTL design; usually, these tools provide only the average power consumption. For this reason, the proposed methodology adopts a custom tool, designed specifically for this purpose, that samples the VCD file obtained from the post-map simulation creating a set of VCD files, one for each considered time epoch. For example, if the desired sampling window of the power monitor is an hundred clock cycles, this tool will sample the VCD file every hundred clock cycles. All these obtained sampled VCD files are provided one by one to the power estimation tool, to obtain a complete power trace.

4.2 Power model

The power model identification is the second stage to be performed after the acquisition of the required data. Before proceeding with the real identification stage, it is necessary to define the model predictors and how to measure them, analyze the multicollinearity of the model statistics, define a set of evaluation metrics and, finally, identify the mathematical formulation of the power model.

4.2.1 Model predictors

In general, the switching activity of a signal is defined as the number of changes in the logic state of the signal itself over a finite amount of time. In the state of the art, different works extend such definition for multibit signals. In particular, three different counting modes have been identified: The *Single Toggle Count* (STC) of a multibit signal is defined as the transition of the logic state of one of more bits of the signal itself. The *Hamming Weight Count* (HWC) of a multibit signal is the number of bits that flip their logic state during the multibit signal transition. The *Single Bit Count* (SBC) of a n-bit signal produces n different values, each one corresponding to the STC of each bit of the signal. For example, considering a 5-bit signal *S* whose value at time t_0 is 5'b10100 and at time t_1 is 5'b11001, the STC value for this transition is 1, the HWC value is 3 and the SBC value is {0,1,1,0,1}. It is important noticing that the relationship between the power consumption and the switching activity of a data signal depends on the actual number of bits of the signals that switches their state, i.e., hamming weight, since, in general, the higher the hamming weight, the higher the power consumption. For this reason, the HWC counting mode is suitable for measuring the switching activity of data signals. On the contrary, a change in a control signal of the design enforces the execution of a different hardware operation regardless of the hamming weight of the control signal itself. Thus, the STC counting mode is suitable for measuring the switching activity of control signals. Authors in [22] propose the adoption of the single bit counting mode, in order to reduce the power monitor implementation overhead. In fact, instead of considering the entire multi-bit signal, this counting mode allows the selection of a subset of bits of the considered signal and use only those bits to build the power model. However, this counting mode can cause non negligible estimation errors. Considering, for example, a power model built with two bits of the 32-bit *Operand_A* of a RISC ALU is considered, the power estimates would depend on the value of the operand. Every values of *Operand_A* showing a zero in the two positions considered by the power model, will produce zero as power estimate, which is, of course, a huge estimation error.

4.2.2 Multicollinearity analysis

One of the main problems for mathematical models identification methodologies is the multicollinearity. It occurs when different variables in a regression model show a high value of correlation. This is a problem because, in a mathematical model, the variables should be independent and, if the de-

gree of correlation between them is high, the identified model can show a low accuracy in fitting data of the test or validation set. This problem is very relevant when identifying power models: in fact, often the wires of an hardware module are bounded by a tight relationship. In order to avoid the multicollinearity problem, the proposed methodology implements a signal filtering stage, that, for each signal, excludes the most correlated ones from the final set of wires to be considered as possible model predictors.

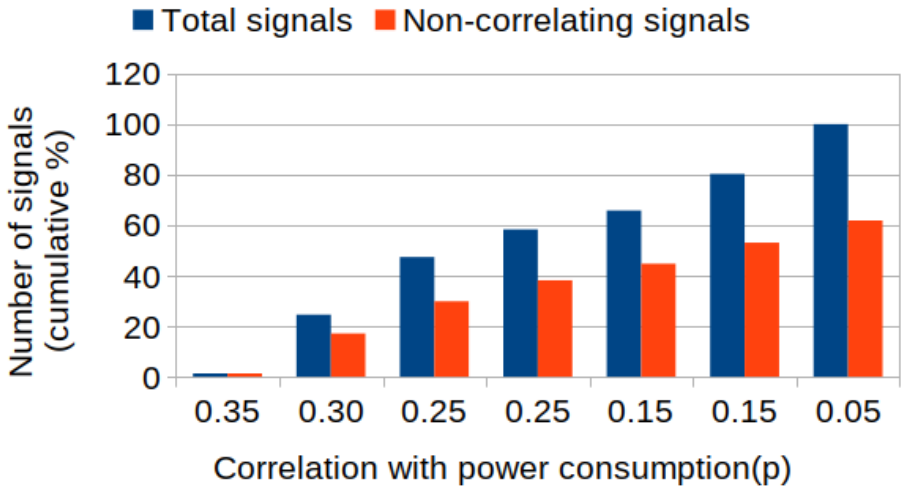


Figure 4.2: *Number of wires before and after the correlation filter.*

Figure 4.2 shows the cumulative percentage of the available statistics before and after the correlation filter. The X axis shows the ranges of correlation with the power consumption, while the Y axis shows, for each chunk, the cumulative percentage of wires that correlates with the power consumption of the target devices. In this work the value of the correlation threshold is set to 20%; it means that is the statistic produced by a wire correlates with another one more than 20%, this statistic is not considered for the model identification step. It can be noticed that a non-negligible number of wires produces statistics that correlate with other ones for more than the threshold. In particular the 38% of the total wires are not considered in the model identification step, due to a too high multicollinearity value.

4.2.3 Evaluation metrics

To measure the accuracy of the proposed power monitor, the methodology considers the distance between each of the n samples of the power estimates

trace ($\hat{p}_i$) and the corresponding power consumption sample (p_i), provided by Xilinx technology libraries (see Equation (4.3)). The analysis of the state of the art proposed in chapter 3 underlines four different metrics to evaluate the accuracy of a power model:

- Mean Average Error (MAE):

$$MAE = \frac{\Sigma_{i=1}^{n}\left(p_i - \hat{p}_i\right)}{n} \tag{4.1}$$

  The Mean Average Error (MAE) metric considers the average distance between the reference power measure p_i and the estimate produced by the power monitor $\hat{p}_i$. However, this metric does not consider the magnitude of the power values so, two different data series with the same ratio between p_i and $\hat{p}_i$ but with different magnitudes, will have very different values of the MAE metric.

- Mean Relative Error (MRE): In order to solve the problem underlined for the MAE metric, author of [48] propose a slightly different metric, the Mean Relative Error.

$$MRE = \frac{\Sigma_{i=1}^{n}\frac{\left(p_i - \hat{p}_i\right)}{p_i}}{n} \tag{4.2}$$

  This metric introduces a ratio between the distance (p_i - $\hat{p}_i$) and the value of p_i. In this way, the value of MRE considers the magnitude of the power values.

- Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{n}\left(p_i - \hat{p}_i\right)^2}{n}} = \sqrt{\frac{\Sigma_{i=1}^{n}\left(e_i\right)^2}{n}} \tag{4.3}$$

  The RMSE measures the standard deviation of the residuals, i.e., the distance between p_i and $\hat{p}_i$. In other words, it measures the spread of these residuals, showing how concentrated the data is around the line of best fit. This metric suffers from the same problem underlined for the MAE metric; an improved definition of the RMSE can be found in the state of the art under the formulation of RMSE$_{norm}$.

- Normalized Root Mean Square Error (RMSE$_{norm}$):

By employing the formulation in Equation (4.4), the point-wise real power consumption (p_i) is defined as the addition between the power estimate ($\hat{p}_i$) and an error (e_i). Considering the error as a random variable with Gaussian distribution, zero-mean and with a σ standard-deviation, the RMSE actually defines the σ quantity, i.e., the variability of the power estimates over the actual power consumption. To this end, Equation (4.5) defines the percentage RMSE normalized to the average power consumption ($RMSE_{norm}$)i.

$$p_i = \hat{p}_i + e_i \tag{4.4}$$

$$RMSE_{norm}[\%] = \frac{RMSE}{\mathbb{E}(p)} \times 100 \tag{4.5}$$

4.2.4 Constrained power model identification

The power model identification algorithm employs a recursive approach to implement the top-down hierarchical visit of the target design (see Algorithm 1). The algorithm takes five inputs: *i)* the top module of the design, *ii)* the user-defined constraints, where each of them is specified as a fraction of the same resource type used by the target design also including an upper bound that specifies the maximum acceptable accuracy error for the identified model, *iii)* profiled information from the `PwrBlockProfile` module, as well as the *iv)* the switching activity and the *v)* power traces of the target computing platform. The mathematical formulation of the identified power model represents the output of the algorithm. It consists in a list of triples, where each triple is defined as the name of a signals of the design to probe, the associated coefficient of the power model, and the employed switching activity that is selected as either the Single Variation Count (SVC) or the Hamming Weight Count (HWC).

Starting from the top module of the design, Algorithm 1 performs a top-down visit of the target design hierarchy to find the best power model within both the imposed resource constraint R and the accuracy upper bound limit $\mathbf{e}_{Th}$. The power model of module M_0 (mId_{M_0}) is accepted if the accuracy is within the allowed error $\mathbf{e}_{Th}$ (see lines 2-5 of Algorithm 1). Otherwise, the children modules of M_0 are sorted in a descending order according to their power consumption and an iterative power model identification exploration starts (see lines 8-20 of Algorithm 1). At each iteration in the `for-loop`, the first sub-module in the sorted `container` list is popped out and its power model is identified (see mId at line 11 of Algorithm 1).

Algorithm 1 Top-down hierarchical visiting algorithm.

```
1: function [model, e] VISIT(M_0, R, e_Th)
2:     [mId, e_M0] = ComputePwrModel(M_0, R);
3:     if e_M0 < e_Th then
4:         model = mId; e = e_M0;
5:     else
6:         container = sortByPower(M_0.m_0 ... M_0.m_N);
7:         mIdlList = [ ]; rIdList = [ ];
8:         for i = 1 : container.size do
9:             m_Tmp = container.pop(i);
10:            R_Tmp = R * mList(i).pwrRel;
11:            [mId, e] = VISIT(m_Tmp, R_Tmp, e_Th);
12:            [mCont, eCont] = VISIT(container, R − rIdList, e_Th);
13:            mIdList.add(mId);
14:            rIdList.add(R_Tmp);
15:            if compErr([mIdList mCont]) < e_Th then
16:                model = [mIdList mCont];
17:                e = compErr(model);
18:                break;
19:            end if
20:        end for
21:    end if
22: end function
```

Moreover, the remaining modules in the `container` list are identified within a single power model. To this extent, a bi-partition of the modules is created. It is worth noticing that the available resources are split proportionally to the two partitions. The module identified in isolation is added to the $mIdList$ list, i.e., the list containing the power models identified on the modules consuming the majority of the power in the target design. At line 15, Algorithm 1 checks if the aggregate error of the power models in the $mIdList$ plus the power model identified for the $container$ modules is lower than the $\mathbf{e}_{Th}$ threshold. This approach allows to optimize the number of the implemented power modules, since the iterative algorithm tries to identify a dedicated power model for the modules than contribute the most to the power consumption, while a single aggregate power model is identified for the remaining ones. The recursive visit of the target design terminates either with a set of identified power models or when the container list is empty and the error is bigger than the imposed $\mathbf{e}_{Th}$ threshold.

Algorithm 2 describes the power model identification procedure, i.e., the `ComputePwrModel` function, employed in lines 2 of Algorithm 1. Algorithm 2 takes as input a module of the design (`m`) and the maximum amount of resources R that can be employed for its power monitor implementation and outputs the identified power model ($\texttt{model}_{cur}$) and the associated accuracy error ($\mathbf{e}$). It is important to note that both HWC and SVC switching activity measurements are provided to Algorithm 2. It will

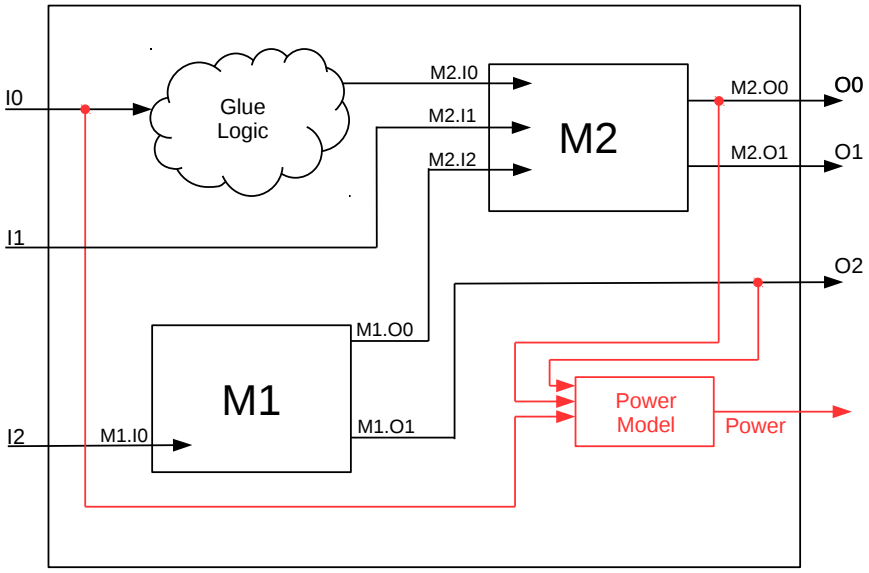
Algorithm 2 Power model computation for module m.

```
1: function [model_cur, e] ComputePwrModel(m, R)
2:     e = MAXERR; model_cur = [ ]; res_cur = MAXRES;
3:     for i = 1 : size[m.I⃗ m.O⃗] do
4:         C = (^[m.I⃗ m.O⃗]_i);
5:         for j ∈ C do
6:             res_new = computePwrMonRes(j);
7:             if res_new < R then
8:                 [model_new, e_new] = linReg(j, m.pwr);
9:                 e_tmp = k1 * (e_new − e)/e;
10:                res_tmp = k2 * (res_cur − res_new)/res_cur;
11:                if res_tmp − e_tmp > 0 then
12:                    e = e_new; res_cur = res_new;
13:                    model_cur = model_new;
14:                end if
15:            end if
16:        end for
17:    end for
18: end function
```

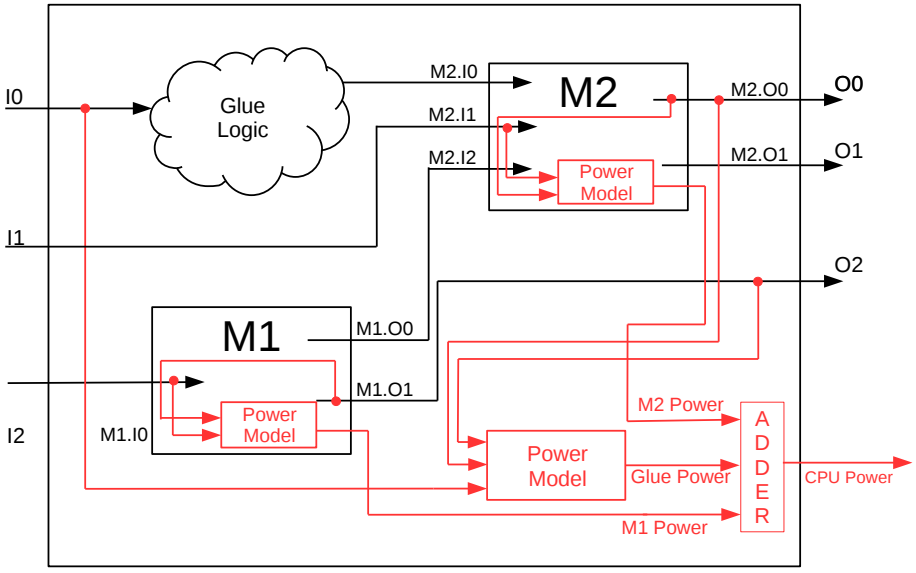
be the algorithm itself choosing the one which maximizes the accuracy of the identified model.

To obtain a simple formulation, the algorithm identifies the power model by leveraging the primary inputs and outputs of the target module. Moreover, the two nested for-loops (lines 3 and 5) drives the exploration to favor the power models that require a small number of probed signals. For each iteration of the outer for-loop, the statement at line 4 determines the set of all the combinations of i signals. For each combination j, the inner for-loop (line 5) computes the power model ($model_{new}$) and the accuracy error (e_{new}) by means of a linear regression procedure (see line 8 in Algorithm 2). Moreover, the estimated resource usage (res_{new}) for the identified power model is computed at line 6.

The identified power model ($model_{new}$) becomes the new candidate if the weighted sum of its marginal increment with respect to the current candidate model, i.e. $model_{cur}$, in terms of accuracy (line 9) and resource saving (line 10) is positive. In such a case, the identified power model ($model_{new}$) becomes the current candidate, i.e., $model_{cur}$ (see lines 11-14 in Algorithm 2).
It is worth noticing that the algorithm allows tuning the weight of the marginal increments in favor of either a more resource-optimized or a more accuracy-optimized power model by means of two parameters, i.e., k_1 and k_2. Without lack of generality, this work used $k_1 = 0.9$ and $k_2 = 0.1$. It is important noticing that, the target architectural module can be identified directly, using only its inputs and outputs (*direct power model*), or



(a) *Direct power monitor*



(b) *Indirect power monitor*

Figure 4.3: *Depending on the result of the power model identification algorithm, the power monitor can be implementing using only the inputs and outputs of the identified module (*direct power model*), or a mix of signals made by the inputs and outputs of the identified module and of some of its submodules (*indirect power model*).*

indirectly, as the sum of the power contributions of some of its submodules, plus the contribution of the glue logic power (*indirect power model*). Figure 4.3 shows an example of a direct and an indirect power monitor: Figure 4.3a reports an example of a direct power monitor, where the power consumption of the target module is estimated directly with its interface signals (*I0, O0, O2*). Figure 4.3b shows an example of an indirect power monitor, where the power estimates of the target are computed as the sum of the power estimates of *M1*, *M2* and the glue logic, represented as the power contribution of the input *I0*.

4.3 Power monitor

The power model identification stage of the proposed methodology, described in section 4.2, produces as output a list of tuples, representing the identified power model. Each entry of the list contains:

- The hierarchical name of each monitored signal
- The model coefficient related to that signal
- the Switching Activity Counting Mode (SACM)

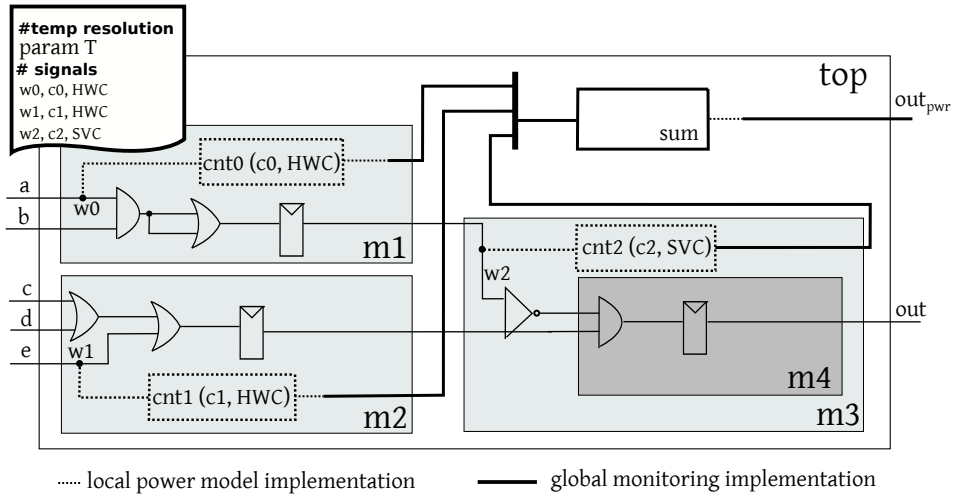


Figure 4.4: *Example of a power monitor implementation, accordingly to the proposed methodology.*

Starting from the mathematical formulation of the identified power model, the `PwrMonImpl` stage delivers the final netlist of the target design augmented with the power monitor. In particular, the RTL description of the

power monitor, that is target independent, is added to the netlist of the computing platform and then a last incremental implementation pass is required to deliver the final implementation netlist. Such aspect of technology independence is crucial and it is enabling the use of the proposed methodology in any hardware design flow. The power monitor implementation stage realizes the identified power model in two steps, moving from the local to the global power monitor. For each triple of values in the mathematical formulation of the identified power model, the local power monitor stage implements a customized power counter. The power counter is customized in terms of the width of the monitored signal, the coefficient associated with the signal, the type of switching activity and corresponding counter selected for the signal. In contrast, the global power monitor stage implements a single power adder in the top module of the target design and connects its inputs with the output of each implemented power counter. Figure 4.4 details the instrumentation of a three-counter power model into a target design where the top module (`top`) implements three sub-modules (`m1`, `m2` and `m3`). The three power counters specified in the power model are implemented to monitor the `w1`, `w2`, and `w3` signals in the target. For each power counter the associated coefficient (`c0`, `c1`, and `c2`) and the type of switching activities are specified in the power model. Last, the power adder instance (`sum`) delivers the periodic power estimate. We note that, in addition to the width of each probed signal and the type between either HWC or SVC to account for the switching activity, the power monitor implementation stage makes use of the temporal resolution (see `param` T in Figure 4.4) to correctly size each power counter. For each signal that is part of the identified power model, a power counter module is instantiated. It collects the switching activity of the corresponding physical signal and, periodically, outputs the power contribution as the product between the switching activity and the coefficient associated with the signal in the identified power model. Figure 4.5 depicts the architectures of the proposed power counter templates tailored to monitor the switching activities of the signal in terms of either Hamming Weight Count (`HWC`) or Single Variation Count (`SVC`). The two power counters share a similar structure to store and to measure the switching activity. In particular, the accumulated switching activity of the monitored signal is stored in the FF_{sa} memory, and another dedicated memory element is used to store the corresponding power model coefficient (see `coeff` in Figure 4.5). In particular, the rst_{sa} signal is used to reset the accumulated switching activity at the beginning of each time period. Moreover, the proposed power counter architectures sample the monitored signal once per clock cycle to measure the switching

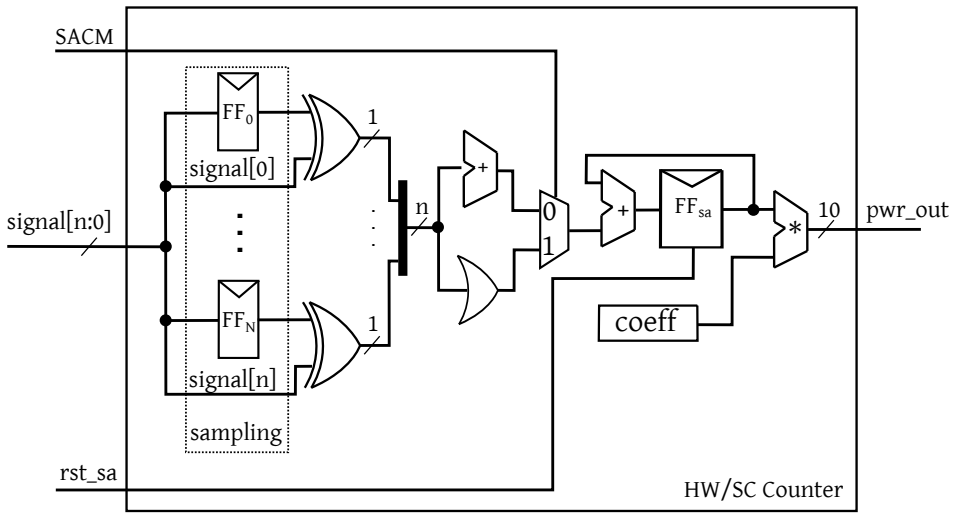


Figure 4.5: *Details of the implementation of the HWC and SVC counters.*

activity in terms of the signal variations in two consecutive sampled values (see `Sampling` block in Figure 4.5 and Figure 4.5). Such monitoring strategy avoids measuring the glitching activity to provide a strong upper bound to the number of switches for each single-bit signal within a single clock cycle, i.e., 1 at the most. However, the power monitor can still capture the non-negligible power contribution due to the glitching activity. In particular, the glitches are due to the particular microarchitecture of the implemented design. To this end, for each toggle in the monitored signal, the associated power observed by the power model identification strategy is the one of the actual signal toggle plus the power due to the related glitching activity, if any.

The most evident difference between the two power counter architectures is in the way the switching activity is accumulated, since an arithmetic adder is used in the HWC power counter, while an OR gate is used in the SVC counter. The power adder is a combinatorial block that is meant to sum up all the power contributions from the instantiated power counters to deliver the periodic power estimate. We note that the impact of the power adder in terms of resource and power overheads is very limited with respect to the rest of the power monitor architecture. To minimize the number of dimensions of the design-time exploration without any flexibility loss, each input to the power adder, i.e., the power counter output, is 10-bit signal while a 12-bit signal is used to size the output of the power adder. By trading the precision and the dynamic range of power measures, this solu-

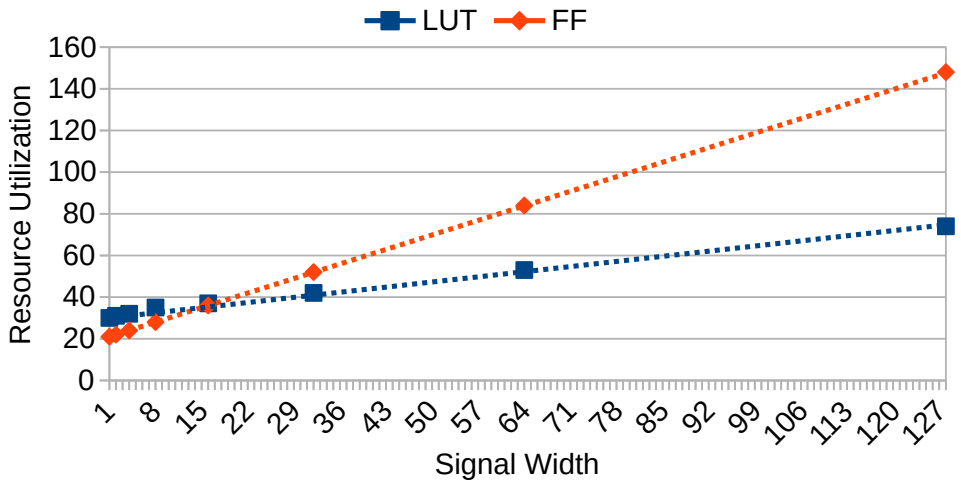
tion can address computing platforms ranging from embedded systems up to High-Performance Computing (HPC) platforms. Such configuration can be achieved by changing the unit employing to report the power consumption in the power traces (see `PowerTrace` in Figure 4.1).

When targeting the embedded system domain, the use of a mW precision to report the power consumption in the power traces allows each power counter to deliver a probed power consumption between 0 and 1024mW while the output of the power adder ranges between 0 and 4096mW. In contrast, the use of a Watt precision to report the power consumption in the power traces allows each power counter to report a probed power between 1 and 1024W while the output of the power adder can top up to 4096W.

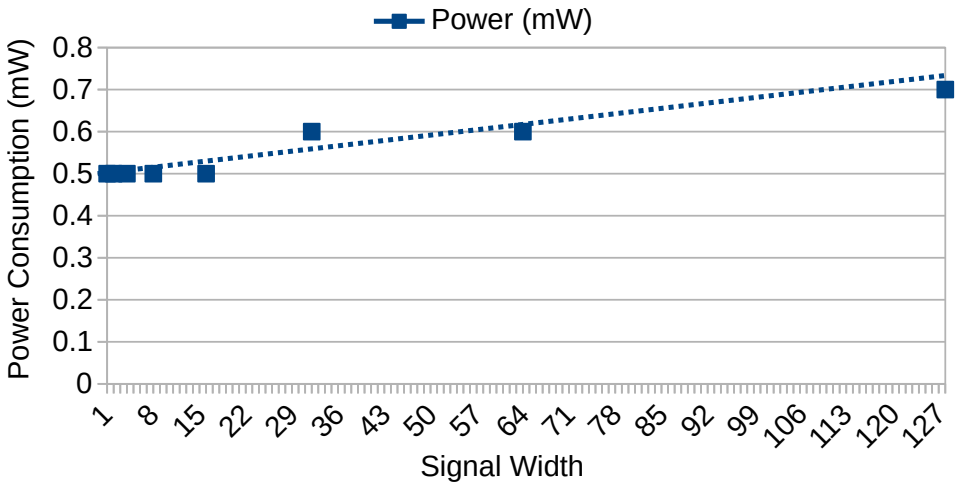
For each power counter architecture, Figure 4.7a and Figure 4.6a) report the resource utilization, i.e., flip-flops (FF) and Look-Up-Tables (LUT), with respect to the width of the probed signal. Considering the SVC power counter architecture in Figure 4.5, both the number of required FFs and LUTs is linear with the width of the monitored signal even if the number of FFs grows faster (see Figure 4.6a). It is important noticing that while the number of FFs is dominated by the width of the signal showing a linear coefficient equal to 1, the number of LUTs required to perform the SVC computation, i.e., the boolean-OR, grows slower. In contrast, the number of LUTs dominates the resources required to implement the HWC power counter due to increasing complexity of performing the arithmetic addition as the width of the probed signal increases. In particular, the number of FFs still grows linearly with the width of the probed signal and it is comparable with the one required for the SVC power counter. In particular, given the width of the probed signal, the marginal difference in the number of required FFs between the two power counter architectures is due to the memory where the switching activity is accumulated during the time epoch. As expected, the power consumption of the two power architectures slightly grows with the increase of the width of the probed signal, while the HWC power counter shows an higher power consumption than the SVC one due to the additional design complexity of the arithmetic addition.

4.3.1 Automatic implementation

In order to make the proposed methodology completely automated, this work proposes, as the final step, an automatic implementation of the identified power monitor. From figure 4.1, it can be noticed that the power monitor implementation step (*PwrMonImpl*), takes as input the mathematical formulation of the power model and the post-map netlist, and gives as output a new netlist augmented with the identified online power monitoring



(a) *Resource utilization*

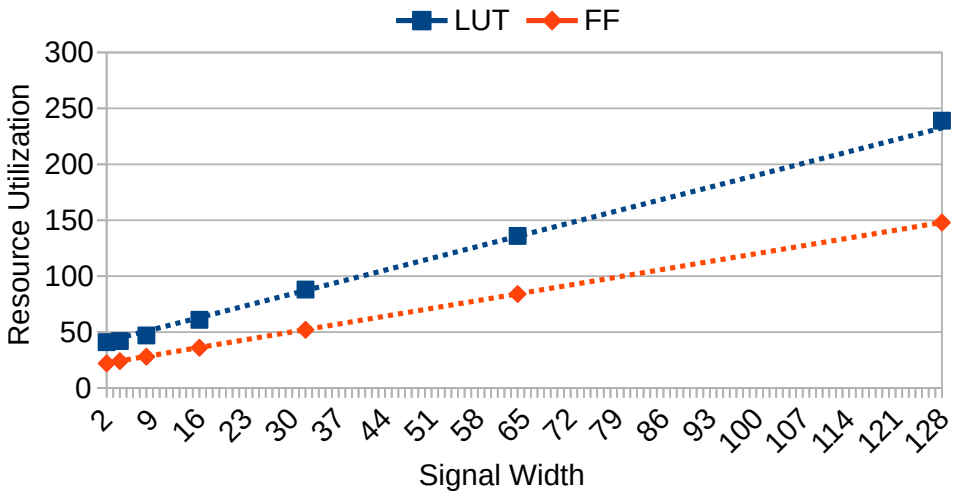


(b) *Power consumption*

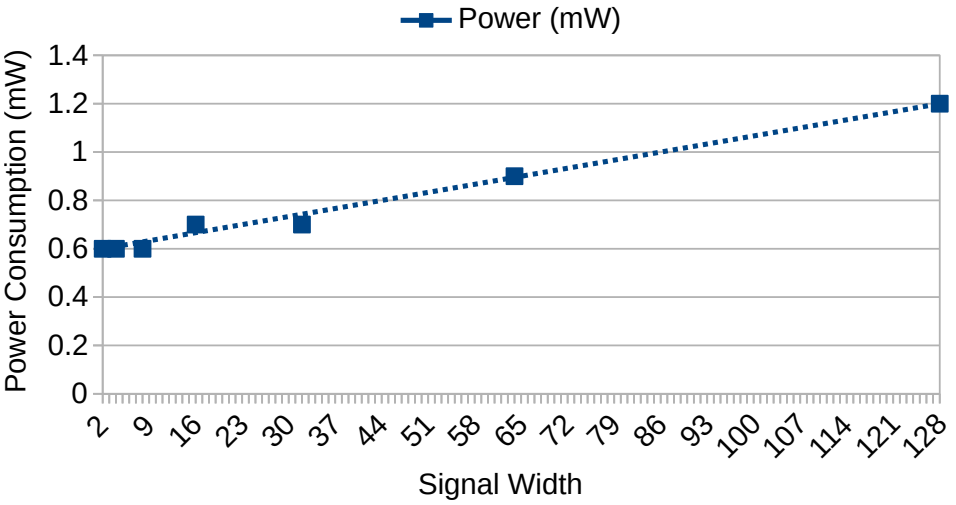
Figure 4.6: *Single Variation Count (SVC) counters architecture.*

infrastructure. To this extent, this work adopts Yosys [38], an open source framework for Verilog RTL synthesis. In particular, the internal intermediate representation of the post-map design has been modified, in order to implement all the components of the power monitor. The automatic implementation algorithm is organized in four steps:

- **Signal analysis**: The first step consists in an analysis of the selected signals to be monitored; in particular the proposed methodology checks whether there are signals that are actually the same in the design or



(a) *Resource utilization*



(b) *Power consumption*

Figure 4.7: *Hamming Weight Count (HWC) counters architecture.*

not. For example, the model can consider two signals a and b, where a is the output of a module, b is the input of another module, and a and b are connected each other. If this occurs, the second signal is not considered and its coefficient is summed with the first one.

- **Model reset**: In this step, a model reset (*model_rst* in figure 4.5) is instantiated. This signal becomes an input of the top module of the design, so it is available to the user for aligning the power traces to a

defined temporal instant.

- **Building blocks instantiation**: In the third step, the power monitor building blocks are instantiated. For each entry of the power model specification, an hardware counter is instantiated within the RTL module where the wire is located. The counter is parameterized with the dimensions of the input and output wires, the model coefficient and the Switching Activity Counting Mode (HWC or SVC). The wire to be monitored is connected to the counter, together with the clock and the reset signals. The output of each counter is propagated up to the top module of the design hierarchy.
- **Power adder instantiation**: In the last step, the power adder is instantiated in the top module of the design. Considering a power model made of n monitored signals, the adder is parameterized with the number of instantiated counters (n), the dimension of the input signals, the dimension of the final result signal and the constant term identified in the model. The adder inputs are the clock and reset wires, the counters results, obtained by concatenating all the results of the power counters, and a signal specifying the sign (+ or -) of each power contribution. The result of the power adder is propagated as an output of the top module.

CHAPTER 5

Experimental Results: Power Monitoring

This section reports the assessment of the proposed methodology focusing on the accuracy and the resource utilization of the hardware-level power monitoring infrastructure. The experimental setup is discussed in Section 5.1, while the accuracy of the power model estimates and the resource utilization results are detailed in Section 5.2. Section 5.3 will analyze the trend of the design metrics, by varying the temporal resolution of the power estimates. This work has been published to the Sustainable Computing: Informatics and Systems [8]

5.1 Experimental setup

To demonstrate the value and the feasibility of this solution a set of HLS-generated hardware accelerators and a complete embedded RISC-V-based system-on-chip computing platform have been employed.

Power monitor generation for the use-case scenarios - The Bambu open-source HLS tool from the Panda project [26] has been selected to translate eight WCET kernels [11] into their corresponding Verilog RTL descriptions. Leveraging the common interface provided by the Panda tool, a hardware wrapper has been designed to allow the benchmarks to communicate with a host computer. A description of this wrapper is provided

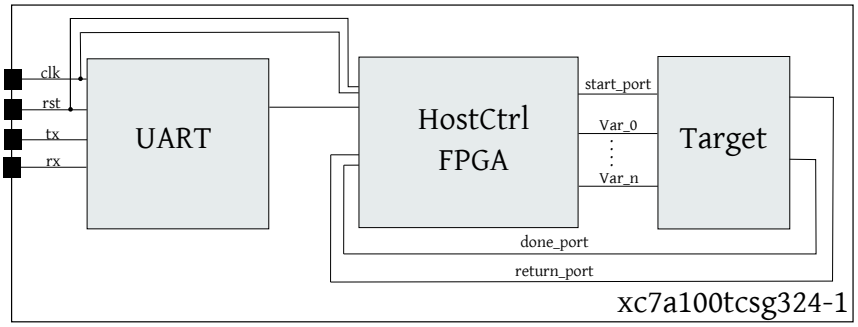


Figure 5.1: *The system used to test the HWC accelerators after the power monitor instrumentation. The HostCtrlFPGA controller drives the target accelerator and communicates with an host computer. A request-response UART-based protocol is used by the host computer to send the inputs and to collect the outputs of the computation.*

in Figure 5.1. Through the UART interface it is possible to set the benchmarks operands, start and stop the execution and read the power estimates provided by the implemented power monitoring infrastructure.

Moreover, the System-on-Chip (SoC) from [34] has been employed, to further assess the effectiveness of this solution against complex hand-coded hardware designs. The SoC features a 32-bit bus-based architectures and an in-order, five-stage RISC-V CPU using an Harvard memory architecture. The CPU offers the hardware support to single-precision floating point as well as to integer divisions and multiplications. A SoC debugger completes the SoC architecture.

Table 5.1 reports the resource utilization of each considered design in terms of Digital Signal Processing (DSP) tiles, LUTs, and flip-flops (FFs) before the power monitor instrumentation. For each hardware design, the methodology presented in Chapter 4 has been implemented in five steps:

1. Vivado 2018.2 has been employed to generate both the post-synthesis and the post-implementation netlist of each considered hardware design.

2. The switching activities and the power traces to identify the power models have been collected from the post-implementation simulations.

3. Starting from the switching activities, the netlist, the power traces and the user-imposed constraints, Matlab-2019a has been employed to implement the power model identification algorithms.

4. The identified power model as well as the post-synthesis netlist of the corresponding hardware design, have been fed to the Yosys open-

source synthesizer which delivered the netlist augmented with the power monitor.

5. Vivado 2018.2 post-implementation has been executed on the netlist generated by Yosys, to deliver the final design implementation.

The functional validation has been assessed by prototyping each design on the Digilent Nexys4-DDR board featuring a Xilinx Artix7 XC7A100TCSG324-1 FPGA targeting a 100MHz implementation frequency. It is worth noticing that the operating frequency of the power monitor exceeds 200MHz, hence the maximum clock speed is limited by the rest of the computing platform. In fact, the power monitor is working in parallel to the monitored design, thus leaving unchanged the original critical path.

Quality metrics - In Section 4.2.3 different evaluation metrics found in the state of the art have been analyzed; this work employs the normalized Root Mean Square Error ($RMSE_{norm}$) to discuss about the accuracy of the power estimates. Given the limitations underlined in Section 4.2.3, the $RMSE_{norm}$ is the metric that best describes the goodness of a power estimate.

To demonstrate the effectiveness of the resource constraints, different power monitors are discussed, each one implemented by fixing the maximum Look-Up-Tables (LUT) budget. For each design, the LUT boundary is specified as a percentage of the total number of LUTs. It is worth noticing that the proposed methodology allows to constrain any type of resource. This work only addresses the LUT constraint, since the LUTs are the most critical resources in FPGA designs.

Table 5.1: *Resource utilization for the considered benchmarks, without the implementation of any power monitor.*

Benchmark	DSP	LUT	FF
fibonacci	0	185	101
crc	0	438	297
aes-Enc	0	491	233
aes-Dec	0	1037	137
expint	2	1361	1334
sqrt	2	2299	1682
qsort	0	2775	1323
fft	78	14974	10776
RISC-V	10	7868	5606
average	10.2	3492	2378.7

5.2 Accuracy and overheads

Table 5.2: *Experimental results for the benchmarks considering three constraints on the use of resources (5%, 10%, 20%) for the power monitoring infrastructure and without any boundary as in the state-of-the-art. A dashed cell means that the cost of the power monitor exceeds the boundary and it is discarded. The overheads on the use of resources and power are relative % values expressed over the size of the original unmonitored designs reported in Table 5.1.*

Name	Unconstrained					Constraint 20%					Constraint 10%					Constraint 5%				
	# HWC	# SVC	% of used LUT	% of used FF	Pwr Ovh	# HWC	# SVC	% of used LUT	% of used FF	Pwr Ovh	# HWC	# SVC	% of used LUT	% of used FF	Pwr Ovh	# HWC	# SVC	% of used LUT	% of used FF	Pwr Ovh
fibonacci	1	2	38.2	20.4	33.5	0	1	16.2	8.3	13.6	-	-	-	-	-	-	-	-	-	-
crc	2	3	33.3	12.8	28.4	0	2	15.3	7.9	13.6	0	1	6.8	3.1	6.1	-	-	-	-	-
aes-Enc	1	2	20.2	10.3	17.1	0	2	12.2	7.1	9.6	0	1	6.1	2.9	5.4	-	-	-	-	-
aes-Dec	1	3	15.9	9.8	12.7	0	3	12.9	7.1	9.7	0	3	8.7	3.8	7.7	0	1	2.9	1.1	2.3
expint	1	2	8.9	4.8	6.9	1	2	8.9	4.8	6.9	1	2	8.9	4.8	6.9	0	1	3.1	1.7	2.8
sqrt	2	1	8.9	4.9	6.6	2	1	8.9	4.9	6.6	2	1	8.9	4.9	6.6	0	1	1.3	0.4	1.1
qsort	0	1	1.1	0.6	0.8	0	1	1.1	0.6	0.8	0	1	1.1	0.6	0.8	0	1	1.1	0.6	0.8
fft	1	2	0.7	0.5	0.7	1	2	0.7	0.5	0.7	1	2	0.7	0.5	0.7	1	2	0.7	0.5	0.7
RISC-V	19	10	18.4	10.2	15.9	18	10	17.9	9.9	15.2	10	4	9.1	4.1	8.2	4	4	4.6	2.2	4.1
average	3.11	2.9	16.2	8.2	13.6	2.4	2.7	10.1	6.0	7.4	2.1	2.6	5.7	3.6	4.3	1	2	2	1.3	1.9

For each benchmark design, Table 5.2 reports the occupation of the implemented power monitors considering different constraints for the LUTs and by assuming a temporal resolution of 20us, i.e., the highest this methodology can provide. In particular, we report results for the unconstrained power monitor (`Unconstrained`) and for three LUT-constrained power monitors (`Constr-20%`, `-10%`, `-5%`). The area and power overheads of the power adder of the different module are negligible and thus they are not explicitly reported in Table 5.2, although such contributions are included and melted in the data reported for each scenario. Results demonstrate the effectiveness of the proposed resource-constrained methodology where all the implemented power monitors respect the specified resource constraints. As expected, the use of an aggressive resource constraint on small designs, i.e., using less than 500 LUTs, prevents the implementation of the power monitor (it is still feasible but exceeding the user-defined constraints). For example, the methodology fails to implement the power monitor for some targets, such as `fibonacci`, `crc` and `aes-Enc` designs when the power monitor overhead is limited to 5% (see `Constraint-5%` results in Table 5.2). This is not a drawback of the proposed methodology, but, on the contrary, it is a proof that this solution ensures a robust control over the resource overheads due to the power monitor. The power monitor for small designs can anyway be implemented simply by relaxing the resource constraint, as it is evident moving from 5% to 10% and then 20% overhead, where all the power monitors are implemented for all the designs. In particular, this investigation confirms that the smallest single-counter power

monitor implementation requires a minimum of 30 LUTs, thus the target computing platform must use 600 LUTs at least to allow a power monitor constrained to 5%. It is worth noticing that the smallest Xilinx Artix7 FPGA, i.e., Artix7-12, features 8000 LUTs thus confirming that a 600 LUT design is actually a tiny one.

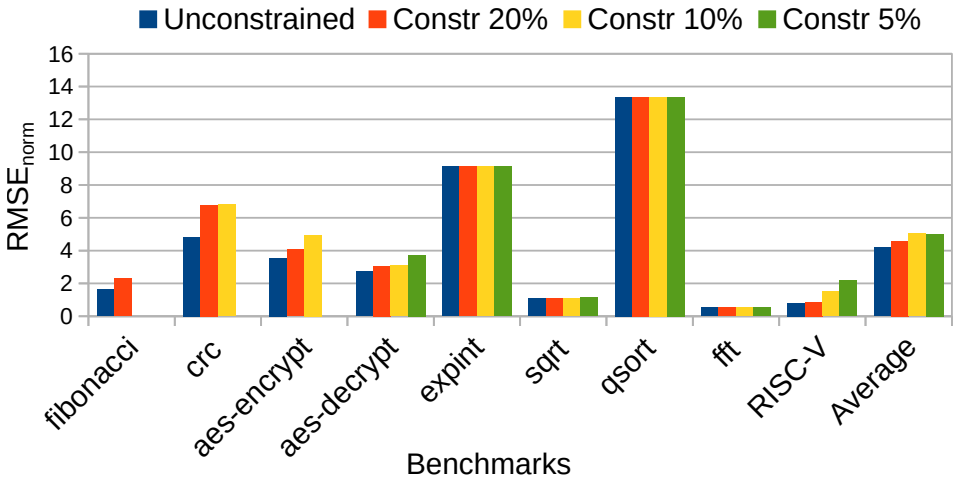


Figure 5.2: *Accuracy loss of the power estimates with respect to the power traces extracted in clean room (Vivado 2018.2). The policy considers acceptable only the solutions with a $RMSE_{norm}$ error below 15.*

5.3 Exploring different time resolutions

As expected, the power overhead has a trend aligned to the resource overhead, hence it is decreasing moving from the unconstrained implementation to our constrained power models. The average power overhead is lower than 5% for a resource constraint of 10% showing an average of 1.9% when LUTs are constrained to 5%.

For each evaluated design, Figure 5.2 reports the accuracy of the implemented power monitors employing the 4 LUT-constraints, i.e., unconstrained, 5%, 10%, and 20%. The RMSE degrades, i.e., higher values, with the resource constraint, although it is always lower than 5% on average regardless of the imposed resource constraint. It is important to notice that the obtained accuracy is aligned with other state of the art solutions [22,24] for which it is not possible however to constraint the resources, and to select the temporal resolution during the automatic instrumentation process.

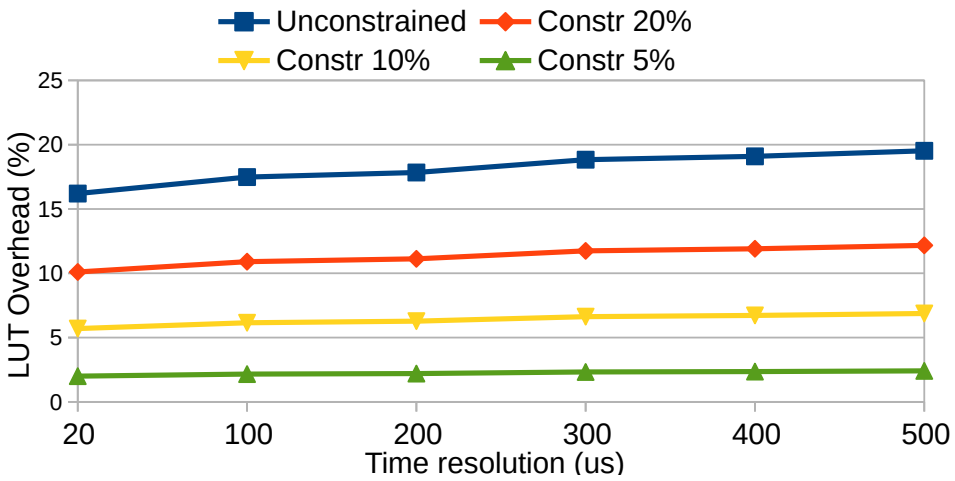
For example, by comparing the unconstrained solution to the one using a `Constr-20%` constraint, the average resource saving is 37.3% while the

average RMSE_{norm} degradation is limited to 0.4%. As expected, the benefit increases with the severity of the imposed LUT-constraint, i.e. a negligible accuracy loss allows to sensibly drop down the used LUTs.

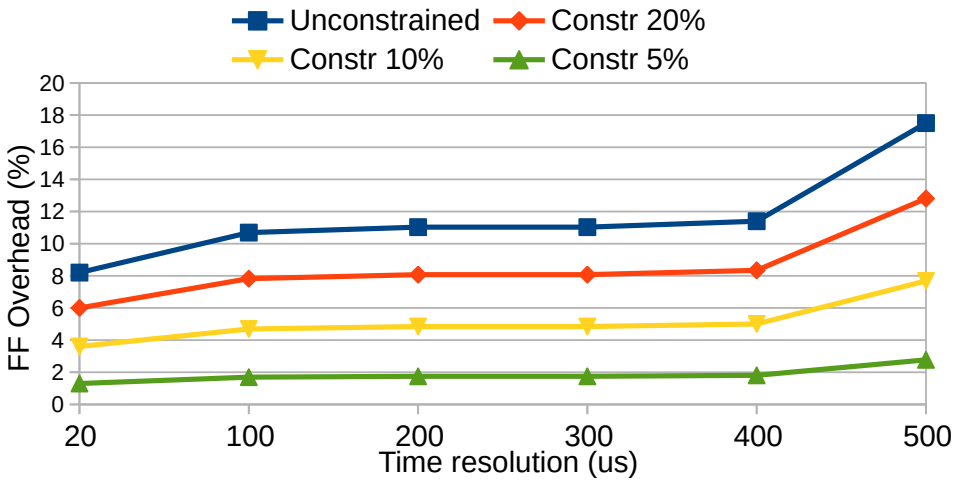
To provide a wider evaluation of the proposed methodology, this section discusses the trend of the considered design metrics by varying the temporal resolution of the power estimates. In particular, this work analyzes area, i.e., LUTs and FFs (see Figure 5.3a and 5.3b), and power overheads (see Figure 5.2c) as well as the accuracy loss by considering different temporal resolutions, i.e., 20us, 100us, 200us, 300us, 400us, and 500us (see Figure 5.2d). It is important to note that the investigated temporal resolutions are aligned with the ones employed in state-of-the-art hardware solutions to optimize the energy-performance trade-off in single- [49] and multi- [50] cores.

By lowering the temporal resolution, a marginal increase for both the resource utilization and the power overhead can be observed. Such increase is due to the need to store more statistics, i.e., for a longer time period, related to the switching activity of the probed signals in the circuit. In particular, more FFs and LUTs are required to perform larger multiplications and additions to deliver the power estimates. However, considering each implemented power monitor, the area and power overhead increase is always lower than 5% across the entire range of the analyzed temporal resolutions.

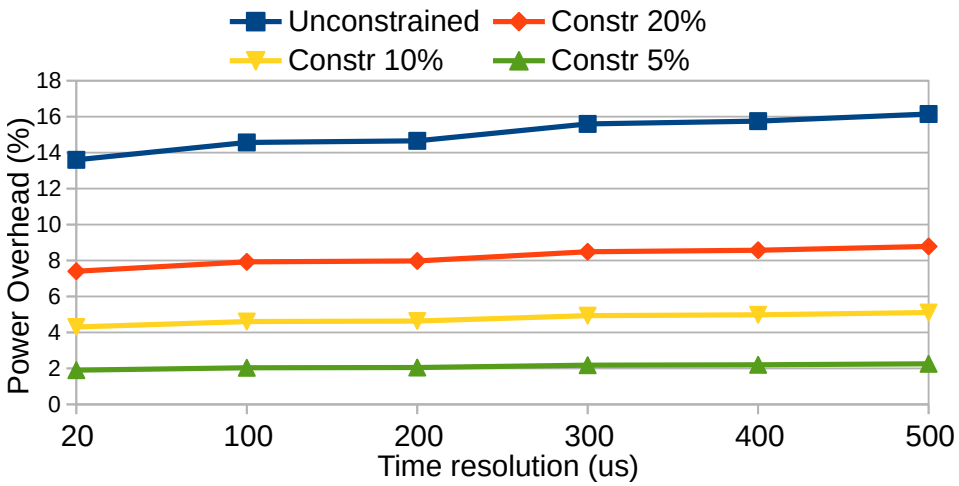
In contrast, the accuracy of the power estimates improves by lowering the temporal resolution of the power monitor (see Figure 5.2d). Lower temporal resolutions act as smoothing factors for the power spikes that are the primary cause of the accuracy loss. In particular, the power consumption trace appears more regular at lower temporal resolutions, thus allowing the power monitor to better track it.



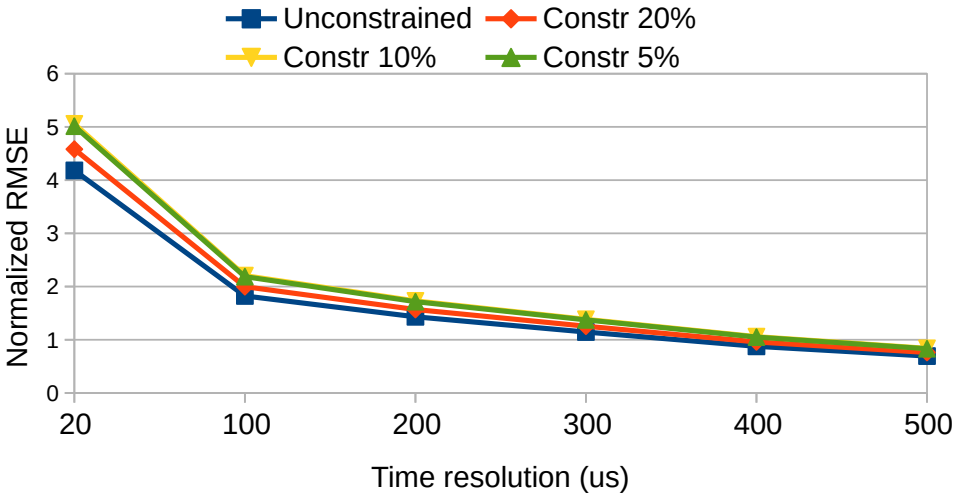
(a) *LUT Overhead*



(b) *FF Overhead*



(c) *Power Overhead*



(d) *Accuracy Loss*

Figure 5.2: *Design Space Exploration by varying the time resolution of the implemented power monitor. The picture reports the trend of the average values.*

CHAPTER 6

Experimental Results: Power Control

Energy efficiency represents a standing obstacle for the evolution of any computing platform, limiting the performance of both embedded and high-performance computing (HPC) applications. Unfortunately, the practice of minimizing the energy consumption under performance constraints, commonly used in HPC scenarios, cannot be readily applied to embedded platforms. In fact, the latter, which are battery-powered, exhibit an opposite optimization problem that imposes to maximize the performance under energy-budget constraints. Moreover, the traditional embedded platforms, that were in charge of a single task, have been replaced by complex multi-cores executing multiple concurrent applications where specialized hardware accelerators are used to offload the most intensive part of the computation. The complexity of such architectures, coupled with the required computational efficiency, highlights a coordinated management problem made of two tightly linked aspects. First, an energy-budget management strategy is required to ensure that the overall device remains operational for a given amount of time. Second, an allocation scheme is required to split the energy budget inside of the computing platform, also accounting for the application-level requirements. However, a single policy cannot easily encompass both the energy-budget and the application-specific requirements.

In fact, the state-of-the-art proposals rely on ad hoc-solutions to solve specific energy-performance optimization problems. There exists some control-theoretic PID schemes used to trade performance with energy [49, 50], thermal [7] or reliability [29] aspects. Such schemes ensure theoretical stability for the controller that, however, is too simple to manage either application constraints or multiple system-wide objectives. In contrast, the majority of the investigations in the state-of-the-art deliver heuristic, or algorithmic implementations, for which even no theoretical guarantee on stability or optimality can be ensured. In general, the use of ad-hoc schemes prevents any comparative analysis and any reuse in application scenarios that are even slightly different from the original one. The majority of the proposals leverages Dynamic Voltage and Frequency Scaling (DVFS) as the sole actuator to achieve both the energy-budget and the performance goals. It is important noticing that the effectiveness of the DVFS mechanism is mitigated by its complex mixed analog-digital design that imposes the use of voltage and frequency islands and the use of a proper resynchronization logic infrastructure [51]. To this extent, faster and simpler actuators are emerging to support the run-time optimizations in the embedded platforms domain. To limit the design complexity of the power rails to support hardware reconfigurability, Xilinx FPGAs only offers Dynamic Frequency Scaling (DFS) only.

One of the main problems emerged by a review of the state of the art about run-time power control schemes, is the low temporal resolution of the online power monitoring infrastructures. In fact, all the software implemented mechanisms for power monitoring and optimization work with actuation intervals that range from one to hundreds milliseconds; most of nowadays computing platforms run applications that show power consumption trends changing faster than one millisecond. Furthermore, some application fields, i.e., thermal managements, requires control steps running in the order of some microseconds.

For this reason, the power monitoring methodology, described in the previous chapters, has been adopted to feed an all-digital run-time power optimization scheme. This section presents the design of a control-based, all-digital, energy-constrained management scheme for general purpose processors and accelerators, leveraging all-digital actuators and monitors.

6.1 Power Controllers

This section describes the control theory based power controllers adopted in this thesis. In particular, Section 6.1.1 explains the hierarchy of the pro-

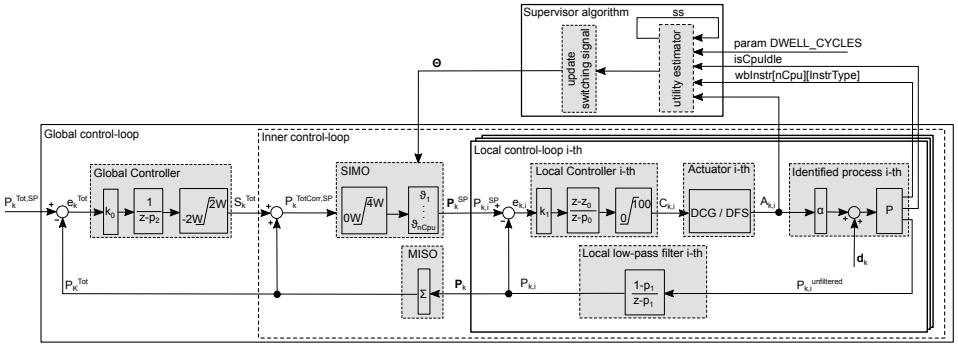


Figure 6.1: *Closed-loop view of the proposed all-digital coordinated energy-budget and energy-allocation system. Local control-loop parameters:* z_0=0.32, p_0=1, p_1=0.5. *Global control-loop parameters:* p_2=1, k_0=0.01. *The identified* α *is equal to* 0.8.

posed power control infrastructure, while Section 6.1.2 analyzes the design of each element of the hierarchical structure.

6.1.1 Hierarchical control scheme

Figure 6.1 depicts the hierarchical structure of the proposed control scheme, which has three parts: local controller, global controller and supervisor controller.

Local controller. At the innermost level, a local control loop is implemented for each computing unit (CPU_i) (see *Local control-loop i-th* in Figure 6.1). The number of implemented CPUs is $nCpu$. The *i-th* local controller of such innermost loop regulates the control action to ensure the power consumption of the corresponding *i-th* core ($P_{k,i}$) to follow the local set point value (see $P^{SP}_{k,i}$ in Figure 6.1).

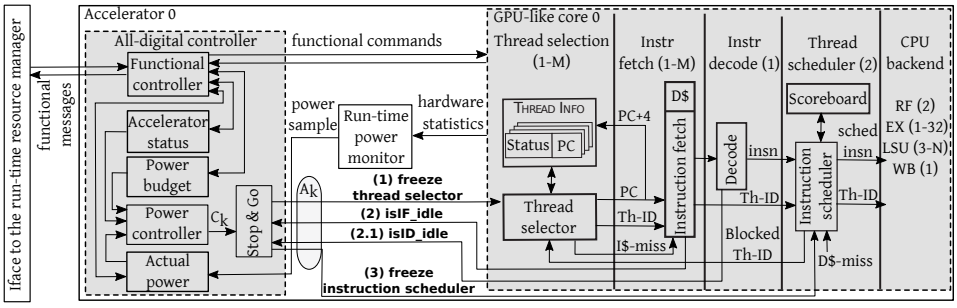


Figure 6.2: *Architectural view of the energy controller and the front-end of the considered accelerator*

Figure 6.2 details the proposed local controller. Please note that the

proposed energy-cap methodology actually constrains the average power consumption that is sampled at a fixed time interval, i.e., once for each time window (k), by the online power monitor. The *Functional controller* interacts with the global resource manager or the Operating System (OS) by means of *functional messages*. Such messages are parsed and actuated via the *functional commands* interface. The global resource manager and the OS can neither directly observe nor control the accelerator. They can only read out its status and dynamically assign a power budget through the memory mapped *Accelerator status*, *Actual power* and *Power budget* registers. The *Accelerator status* register is also updated when a new application is kicked off or terminated. The *Functional controller* delivers the same interface to communicate with the controller regardless of whether the latter control either the host CPU or the accelerators.

The *Power controller* takes three inputs, *i)* the status of the computing resources, *ii)* the power budget, and *iii)* the actual power consumption, to produce the actuation signal (C_k), for each time window k, used as the input to the power actuators.

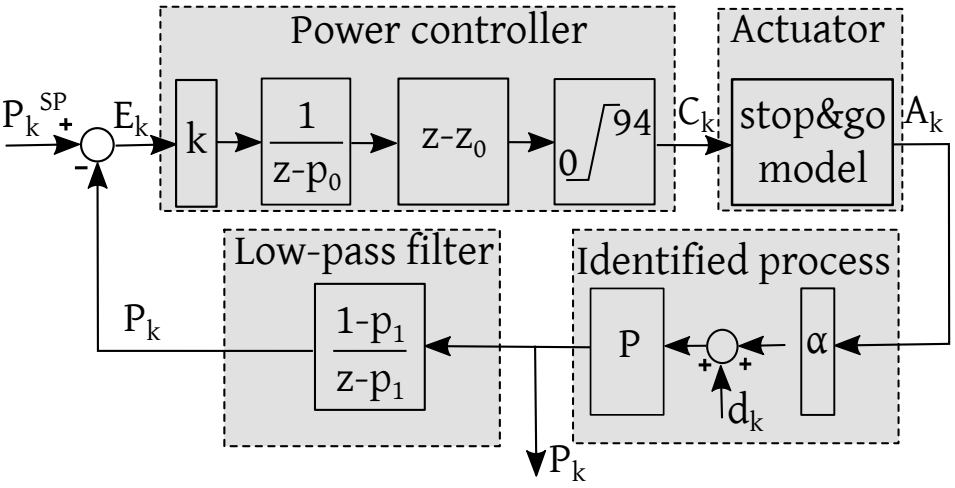


Figure 6.3: *Block diagram of the closed-loop controller.*

Figure 6.3 depicts the discrete time-domain closed-loop of the proposed power controller. It implements a programmable control-based PID and a low pass filter, and it actuates on the process (P) that models the power consumption of the system. The set point (P_k^{SP}) represents the energy-cap dynamically imposed by the OS or the resource manager for each time window k. The actual power consumption (P_k) is backward propagated via a low pass filter to generate the error signal (E_k) which is then fed to the

controller. The control signal C_k drives the *Stop&Go* actuator by directly impacting the controlled variable, i.e., power state of the system, via the actuation variable (A_k). The d_k quantity models the non-controllable disturb on power consumption.

Global controller. The outer layer (see *Global control-loop* in Figure 6.1) is made of a single-input single-output (SISO) *Global controller* that generates a correction factor to the power budget (S_k^{Tot}) starting from the difference between the global set-point $P_k^{Tot,SP}$ and the total consumed power in the considered time epoch k (P_k^{Tot}). It is worth noticing that the global power set-point ($P_k^{Tot,SP}$) is imposed by either the operating system or the resource manager and it represents the average power consumption to enforce the required energy-budget. Moreover, it can change overtime to honor any higher level policy and requirement. The correction factor S_k^{Tot} is generated by the global controller and it is added to the P_k^{Tot} quantity to generate the global available power budget ($P_k^{TotCorr,SP}$) for the time epoch k. Such budget is split between the local controllers in the form of their local set-points ($P_{k,i}^{SP}$) (see *Inner control-loop* in Figure 6.1). Moreover, the saturation blocks in the *Global control-loop* are used for implementability reasons and their saturating values have been selected according to the employed reference platform (see Section 6.1.2 for further details).

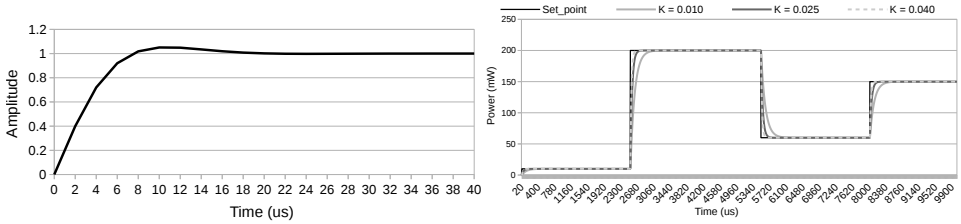
The global controller implements a single-input single-output (SISO) scheme that, for each time epoch k, takes the global power set point ($P_k^{Tot,SP}$) and outputs the available power budget ($P_k^{TotCorr,SP}$). Such global power set point ($P_k^{TotCorr,SP}$) is obtained by adding the total consumed power (P_k^{Tot}) and the correction factor S_k^{Tot} generated by the global controller. The correction factor S_k^{Tot} is related to the difference between the global set point $P_k^{Tot,SP}$ and the total consumed power, while the $P_k^{Tot,SP}$ quantity represents the average power consumption corresponding to the assigned energy budget imposed by the operating system (OS) or by the resource manager. The strategy adopted by an OS or the strategy used by a resource manager to decide the energy budget, falls outside the scope of this work. Last, the overall budget ($P_k^{TotCorr,SP}$) is split into the local set-points (see the *SIMO* in Figure 6.1).

6.1.2 Controller design

The global controller structure is meant to apply a correction factor to the total budget, i.e., $P_k^{Tot,SP}$. In particular the global controller consists in an integral component ($p_3 = 1$) that is fed with the error signal (e_k^{Tot}), i.e., the difference between the total power budget ($P_k^{Tot,SP}$) and the actual total

consumed power (P_k^{Tot}). The controller works as an energy buffer to make available the power that hasn't been used so far in the next epochs. Similarly, the integral controller operates in the opposite direction if the total consumed power is higher than the set point. The correction factor S_k^{Tot} is algebraically added to the total budget $P_k^{Tot,SP}$ to get the total corrected power budget, i.e., $P_k^{TotCorr,SP}$. Such scheme allows to maximize the use of the energy budget in the long run by increasing or by reducing the total energy budget available to the computing units at each epoch k, depending on the energy spent in the past. Without lack of generality, the correction factor S_k^{Tot} is limited between $+/-2$ watts, while the overall budget $P_k^{TotCorr,SP}$ stays between 0 and 4 watts. Such numbers have been selected starting from the observed maximum power consumption of the reference platform, i.e., 1.6 W, to maximize the benefit of the proposed scheme and to size the width of buses and wires in the implemented microarchitecture.

k_0 has been set equal to 0.01 to enforce a slow dynamics of the integral part of the controller. In this perspective, the pure integrator ensures that the global controller matches the imposed energy budget in the long run. The controller has been designed to ensures the correction factor has a "slow" dynamics, namely milliseconds (see Figure 6.4b), compared to the response time of the local controllers, 30μs (see Figure 6.4a). The reason for this is twofold. First, the dynamics of the set-point in the global controller is expected to change in the order of seconds and above, thus a fast controller is not required. Second and more important, the difference between the dynamics of the global and of the local controllers, allows the latter to converge between two consecutive changes in the local set-points operated by the former.



(a) *Simulation of the step response of the local control-loop*

(b) *Simulation of difference step responses of the closed-loop transfer function*

Figure 6.4: *Figure 6.4a shows the simulation of the step response of the local control-loop, while Figure 6.4b analyzes the simulation of different step responses, considering the global closed-loop transfer function and different k_0 values.*

The budget-split module ($SIMO$) distributes the total power budget to

each computing element of the multi-core, by modifying the set point of each local controller. Such distribution of the budget is achieved by means of a vector of θ values defined as follows:

$$\theta_i \in \mathbf{R}, \qquad \theta_i \geq 0, \qquad \sum_{i=1}^{nCpu} \theta_i = 1 \tag{6.1}$$

where $nCpu$ is the already defined number of computing units in the multi-core. The vector of θ values allows to distribute the total budget to the local controllers and it represents a flexible point to design the energy-allocation policies. The design of such policies has been constrained to the supervisor algorithm.

Supervisor controller. On top of the control loops, the supervisor algorithm actually implements the energy-allocation policy by modifying the θ coefficients of the SIMO box. Such controller regulates the quota of the energy-budget assigned to each local controller in the form of its set-point value. As one of the possible strategies, this manuscript presents a fair energy-allocation strategy for performance balancing. However, the supervisor algorithm can implement any scheme or heuristic to shape the performance metric, thus ensuring a great flexibility in the customization of the system. From a mathematical perspective, the possibility of changing the θ values at run-time turns the proposed control scheme into a *switched system* that is made of a set of asymptotically stable linear systems. In particular, each linear system of such switched system is the hierarchical control scheme made of the global and all the local controllers with a specific instance of the θ vector of weights. It is worth noticing that the obtained switched system is a particular non-linear system made of a set of time-invariant asymptotically stable linear systems for which the stability condition has been proven to be fairly easy to obtain. In particular, this work is interested in the exponential stability of the switched system [12] that is the specific asymptotic stability property required for non-linear systems. The proposed methodology leverages the work in [12] which demonstrates that any switched system made of a set of time invariant asymptotically stable i linear systems, always has a minimum *dwell time* that ensures the exponential stability of the entire control scheme. The *dwell time* represents the minimum time interval between two consecutive changes of the θ values, and it guarantees the exponential stability of the switched system. Such time interval depends on the characteristics of the set of closed-loop systems making the switched system. It represents a single and easy to achieve condition to preserve the stability of the proposed control scheme. It is important to notice that such result represents a critical advance in the

design of the coordinated energy-budget and energy allocation schemes, since this work provides a framework to virtually design any possible policy. The supervisor controller implements an energy-allocation policy to balance the performance of the CPUs. Such policy is mainly intended to show the wide range of design possibilities offered by the proposed control scheme, although it also achieves remarkable results with respect to the imposed performance goal. A core-wise utility metric has been defined as the proxy for the performance. In particular, Eq. 6.2 defines the calculated utility ($uCalc_{i,k}$) of core i-th at time epoch k as the number of committed instructions (*wbInstr*) within a time epoch of 2μs corresponding to 100 clock cycles in the proposed implementation running at 50MHz.

$$uCalc_{i,k} = \sum_{j=1}^{instrType} wbInstr_{i,k} * numCcInstrType_j$$
$$i \in \{\ 1,\ ..,\ nCpu\ \} \tag{6.2}$$

Each instruction have been weighted according to its theoretical latency expressed in terms of clock cycles to complete its execution (*numCcInstrType*). This allows to acknowledge that executing fewer multi-cycle instructions rather than multiple single-cycle leads to the same utility. In particular, three instruction classes, have been considered i.e., *instrType*. Load/store instructions have a weight of 8, FPU ones have a weight of 16 while all the others, including the ALU ones, have a weight of 1. It is worth considering that, by construction, it is impossible to define an accurate utility model since some types of instructions, e.g., loads, stores, are affected by a non-deterministic latency dependent on the shared resource contention. Even single clock latency instructions may take multiple cycles to complete, due to data and/or structural hazards in the CPU. However, the experimental results show that the control-inspired nature of the supervisor controller makes it adaptive with respect to such possible model inaccuracies. Please note that the specific instance of the policy does not represent the key contribution of this work and thus any other utility metric and policy can be implemented to optimize different run-time goals. The calculated utility $uCalc_{i,k}$ is employed to define the CPU utility that represents the performance proxy for each core (see Eq 6.3). In particular, the supervisor algorithm updates the utility of the core i at the time epoch k ($u_{i,k}$) considering a weighted sum of the current utility and the calculated utility $uCalc_{i,k}$ as defined in Eq. 6.2 .

$$u_{i,k} = 0.5 \times u_{i,k-1} + 0.5 \times uCalc_{i,k} \tag{6.3}$$

Starting from the defined utility ($u_{i,k}$), the supervisor implements a finite state machine (FSM) that aims at optimizing the performance fairness between the cores, i.e., to maximize the balanced utility ($u_{i,k}$), by acting on the θ values to modify the energy-budget allocation to each local controller. Note that a CPU can have its utility lower than the average one due to either its energy-budget or the application behavior. In the former scenario, the supervisor controller can increase the application utility by increasing its local budget, i.e., by rising the set point of the corresponding local controller. In the latter scenario, the application is self-constraining its own utility and no increase of the energy budget can improve it. For example, an idle core always shows zero utility and a core executing a self-suspended application reports a close to zero utility.

To this extent, the supervisor algorithm identifies three sets to classify each core: *idle*, *balanced* and *unbalanced*. The *idle* set contains all the idle cores. The *balanced* set contains all the cores running a self-limiting application for which no further action of the controller can increase its utility. The *unbalanced* set contains all the other cores.

The supervisor algorithm aims at maximizing the average utility of the *unbalanced* set with minimum standard deviation. This set is in fact the only set of cores that can benefit from a reshape of the θ values. Such goal allows to balance the utility between all the cores when the energy budget is low enough to actually constrain the execution of the applications and no core is idle.

Actuators. To argue the feasibility of the proposed solution, this work is constrained by the characteristics of existing FPGA platforms, that offer DFS support only. In fact, due to the complexity of designing a dynamically scalable power rail for the reconfigurable logic, no commercial Xilinx FPGA integrates DVFS actuators. However, our controller can be coupled with any actuator and, to this purpose, this work shows the use of DFS and DCG actuators which have been both implemented on the considered Digilent board. The Dynamic Clock Gating (DCG) has been integrated to control each *nu+* core of the considered platform. It acts within each time epoch k by disabling the clock signal for a fraction of the time epoch to reduce the activity, and thus the power consumption, of the corresponding core. At the chosen time resolution, i.e., 2μs, the operating frequency of 50MHz shapes time epochs made of 100 clock cycles each. In particular, the designed DCG actuator leverages the programmable FPGA resource only, and takes up to 4 cycles to stop the clock cycles of the controlled CPU. Differently, the Dynamic Frequency Scaling (DFS) actuator leverages the Xilinx Mixed Mode Clock Manager (MMCM) resources of the

Artix-7 FPGA family [39]. The MMCM generates the clock frequency and it offers a reconfiguration port to change the operating frequency at run-time.

6.2 Quality metrics

This section presents the quality metrics designed to assess the proposed methodology. These metrics can be organized in two categories: the *Local quality metrics* have been designed to measure the effectiveness of each local controller, while the *Global quality metrics* measure the effectiveness of the overall control system.

6.2.1 Local quality metrics

Two quality metrics have been defined to capture the performance loss and the energy cap violations for energy-constrained optimizations. Both metrics are piece-wise defined for constant set point values and are limited between 0 and P^{SP}. The *overflow metric* (OVF) sums up the magnitudes of the energy budget violations (see Equation 6.4). The *efficiency metric* (EFF) measures the performance loss induced by a non-zero control action when the power consumption is below the assigned energy budget (see Equation 6.6). For each epoch k, Gap_k (see Equation 6.5) measures the minimum positive value between two quantities: *i)* the difference between the power cap and the actual power ($P_k^{SP} - P_k$), and *ii)* the proportional increase in the actual power in case the control action is equal to zero, i.e. $(\frac{TP*P_k}{TP-C_k} - P_k)$. A temporal resolution (TP) of 100 clock cycles has been set for the entire system.

$$OVF = \frac{1}{P^{SP}} * \frac{\sum_{k=1}^{\#samples} max(0, P_k - P^{SP})}{\#samples} \tag{6.4}$$

$$Gap_k = min\Big((P_k^{SP} - P_k), (\frac{TP * P_k}{TP - C_k} - P_k)\Big) \tag{6.5}$$

$$EFF = \frac{1}{P^{SP}} * \frac{\sum_{k=1}^{\#samples} max(0, Gap_k)}{\#samples} \tag{6.6}$$

6.2.2 Global quality metrics

Three metrics have been defined to measure the quality of the collected results. The global efficiency (EFF_g) expresses the efficiency of the assigned global power budget. The global overflow (OVF_g) measures the average overflow with respect to the imposed global set-point ($P_k^{Tot,SP}$). Last, the global utility (U_g) measures the obtained performance as intended in the supervisor algorithm, i.e., a fair balance of the utility within the set of *unbalanced* cores.

Global efficiency - The global efficiency (EFF_g), which is a percentage value between 0 and 100, measures the efficacy of the assigned global set-point to execute the computation. The EFF_g figure of merit has been developed starting from the concept of the maximum consumed power at time epoch k for core i ($P_{k,i}^{max}$), which is defined as follows:

$$P_{k,i}^{max} = \frac{100}{100 - A_{k,i}} \times P_{k,i}, \quad 0 \leq A_{k,i} < 100 \tag{6.7}$$

In particular, Eq. 6.7 assumes 100 as the maximum number of uncontrolled clock cycles within any time-epoch k, while $A_{k,i}$ ($0 \leq A_{k,i} < 100$) is the control action that limits such value. Please note that the control action is upper limited to 99 to prevent Eq. 6.7 from getting infinite values. In particular $A_{k,i}$ either stretches the clock cycle period (DFS) or selectively masks some positive edges of the clock to the computing logic (DCG).

Eq. 6.8 leverages the current power ($P_{k,i}$) and actuation ($A_{k,i}$) on the i-th core as well as its maximum power consumption ($P_{k,i}^{max}$) to define the power gap between the actual power and either the power max ($P_{k,i}^{max}$) or the current local set-point ($P_{k,i}^{SP}$). The $P_{k,i}^{cap}$ quantity accounts for the running applications that consume far less of the assigned power budget. For this reason, such applications must not contribute to reduce the effectiveness of the energy-budget scheme. In fact, the control action for these applications is null and it is not possible to increase neither the power consumption nor the performance.

$$P_{k,i}^{cap} = min\Big((P_{k,i}^{SP} - P_{k,i}), (P_{k,i}^{max} - P_{k,i})\Big) \tag{6.8}$$

Eq. 6.9 defines the energy-budget efficiency for the i-th CPU as the weighted average of the efficiency at each time epoch k. Please note once more that the global set-point is equal to the sum of the local set-points. Thus, the total energy-budget efficiency (EFF_g) has been defined as the

average of the local efficiencies, i.e., $EFF_i,\ \forall i \in nCpu$, (see Eq. 6.10).

$$EFF_i = \frac{1}{P_i^{SP}} \times \frac{\sum_{k=1}^{\#samples} max(0, P_{k,i}^{cap})}{\#samples} \tag{6.9}$$

$$EFF_g = \frac{\sum_{i=1}^{|nCpu|} EFF_i}{|nCpu|} \tag{6.10}$$

Please note that the proposed definition of the global efficiency takes into account those scenarios for which the imposed global set-point is far bigger than the cumulative power consumption of all the applications running on the multi-core, e.g., when some cores are idle or when some applications are self-suspending. In particular, the efficiency is preserved in these scenarios.

Global overflow - While the global efficiency measures the quality of the assigned budget, the global overflow OVF_g measures the quality in fulfilling the total budget. In a nutshell, OVF_g measures the difference between the actual total power consumption and the set-point. Its value is limited to positive numbers and it is reported in terms of milliwatts.

The definition of such metric starts by noting that the global controller implements an integral component that allows to employ an unused part of the budget at epoch k, in the following epochs. In the same way, an overuse of the budget at epoch k determines a negative correction (S_k^{Tot}) on the budget pertaining the subsequent epochs. Such implementation allows to constrain the consumed energy to the allocated energy budget in the long run. In this scenario it is worthless to measure the epoch-wise overflow, being the global setpoint dynamically corrected to optimize the use of the imposed energy budget. To acknowledge such observation, the segmented overflow (OVF^{seg}) has been defined considering a set K of consecutive time-epochs, where the global set-point ($P_k^{Tot,SP}$) is fixed.

$$\begin{aligned} OVF^{seg} &= \frac{\sum_{i=1}^{|K|} P_i^{Tot}}{|K|} - P_i^{Tot,SP} \\ subject\ to \quad & P_i^{Tot,SP} = P_j^{Tot,SP} \quad \forall i,j \in K \end{aligned} \tag{6.11}$$

In particular, the OVF^{seg} quantity is defined as the algebraic difference between the averaged power consumption in the K set of time-epochs and the

global set-point. Since the global set-point is fixed and the period of time within the set of epochs is expected to be sufficiently long, such quantity measures the overflow with respect to the imposed budget in the long run.

Starting from the OVF^{seg}, Eq. 6.12 defines the global overflow OVF_g as the average of the segmented overflow values to account for the changes of the global set-point. The OVF_g has been limited to positive values since this work is interested in the overflow to the imposed set-point. The underflow would instead indicate that the set-point is too relaxed with respect to the actual power consumption.

$$
\begin{aligned}
OVF_g = & \frac{\sum_{i=1}^{|T|} OVF_i^{seg}}{|T|}\Big\rfloor_0 \\
subject\ to \quad & T = \{K_1, .., K_n\} \\
& K_z = \{k_1, .., k_m\}, \quad z \in \{1, .., n\} \\
& k_t\ is\ a\ time\ epoch, \quad t \in \{1, .., m\} \\
& P_{k_r}^{Tot,SP} = P_{k_q}^{Tot,SP} \quad \forall k_r, k_q \in K_z
\end{aligned} \tag{6.12}
$$

Global utility loss - The global utility loss ($ULoss_g$) measures the distance between the average utility and the utility of each core as the percentage relative error. To this extent, the smaller the better.

To define the global utility loss, the concept of utility has been leveraged as the performance proxy to implement the fairness policy of the proposed control scheme. In general, two applications showing the same power consumption can have different utility values. The utility is in fact related to the mix of instructions executed by each application and to the structural and data hazards.

As already discussed before, the fairness has been enforced by minimizing the utility loss of the cores in the so-called unbalanced set, namely the cores that are neither idle nor have a utility lower than the average, and zero control action. In fact, idle CPUs show zero utility. Moreover, those CPUs for which the control action is zero cannot experience a utility improvement by increasing their set point, since their utility is limited by the application behavior itself. Thus, the unbalanced cores are those for which a change in the power set-point and the relative control-action affects the utility.

The utility definition of Eq. 6.13 has been leveraged; it, for each time-epoch k, updates such quantity depending on the number of committed instructions each of them weighted according to its instruction type. For

each time epoch k, the utility loss of the cores in the unbalanced set (U^{epoch}) is defined as follows:

$$ULoss^{epoch} = \frac{\sum_{i=1}^{|set_{unbal}|} \left| \frac{u_i - U_{avg}^{set_{unbal}}}{U_{avg}^{set_{unbal}}} \right|}{|set_{unbal}|} \times 100 \tag{6.13}$$

The quantity u_i is the utility of the i-th core at time k as defined in Eq. 6.13, while $U_{avg}^{set_{unbal}}$ is the average utility of the cores in the unbalanced set at time-epoch. To this extent, the utility loss at time-epoch k measures the distance, as the relative error, between the average utility and the utility of each core in the unbalanced set. This work leverages the quantity $ULoss^{epoch}$ to define the global utility loss (U_g) as the $ULoss^{epoch}$ averaged on the set of the considered time-epochs (see Eq. 6.14).

$$\begin{aligned} ULoss_g &= \frac{\sum_{i=1}^{|K|} U_i^{epoch}}{|K|} \\ subject\ to \quad & K = \{k_1, ..k_n\} \\ & k_i\ is\ a\ time\ epoch,\ i \in \{1, .., n\} \end{aligned} \tag{6.14}$$

6.3 Results

This work employs the *nu+* SIMD processor as the reference computing platform for which the complete description is available in [48]. To show the scalability, this work employs both the quad- and the eight-core versions of the processor, for which each *nu+* core supports 16-way SIMD instructions but it is limited to a single-thread of execution. Multi-threaded applications run by using one core for each running thread. Such architecture can stress each part of the proposed control scheme. 20 programs from the WCET benchmark suite have been used as representative applications [48]. The applications are running as bare-metal programs and the required run-time software has been implemented to support multi-threaded execution, hence no OS support is offered and, thus, no GNU Linux libraries can be used. Such choice is meant to favor the implementation of the complete prototype of the reference computing platform, with emphasis to the hardware implementation of the proposed controller, i.e., the relevant contribution of this work. WCET can run on bare metal, but since they are indeed complete applications, they can stress each part of the computing

platform. The entire design is synthesized, implemented and simulated at 50MHz targeting the Digilent Nexys4-DDR board [41] equipped with a Xilinx Artix-7 100t FPGA chip [40].

6.3.1 Static scenario

Table 6.1: *Results considering the 4-core processor using the Dynamic Clock Gating (DCG) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.*

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	99.78	97.53	95.69	93.94
200	100	99.92	98.81	94.14
300	100	100	99.99	96.94
400	100	100	100	99.85

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	0.43	2.09	4.01	5.50
200	0	0.03	0.39	5.11
300	0	0	0	1.42
400	0	0	0	0.21

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	3	4
100	0	4.54	6.78	8.88
200	0	0.37	0.43	5.27
300	0	0	0.01	3.7
400	0	0	0	0.41

This section reports the results in terms of the global efficiency (EFF_g), global overflow (OVF_g) and global utility loss ($ULoss_g$) considering the quad- and the eight-core reference processors, with a two-fold objective. First, this work assesses the performance of the proposed control scheme, i.e., the effective use of the allocated global budget without overflowing it, while considering a performance constraint (balanced utility for the running applications in the considered experiments). Second, it assess the scalability of the proposed solution by considering two processors, i.e. up to 8 cores. To ensure a statistical significance, each reported result is obtained as the average of 30 simulations of the same usecase, for which different randomly chosen benchmarks have been selected. Table 6.1 and Table 6.2 report the obtained results for the quad-core processor considering the use of the Dynamic Clock Gating (DCG) or the Dynamic Frequency Scaling (DFS) actuator, respectively. Results for the eight-core processor considering DCG and DFS are reported in Table 6.3 and Table 6.4, respectively. For

Table 6.2: *Results considering the 4-core processor using the Dynamic Frequency Scaling (DFS) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.*

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	100	98.04	92.25	94.4
200	100	99.99	96	93.54
300	100	100	99.99	98.85
400	100	100	100	99.23

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	0.05	0.49	6.33	13.42
200	0	0.03	0.38	1.71
300	0	0.09	0.09	0.1
400	0	0.04	0.04	0.05

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	3	4
100	0.03	0.41	7.81	9.22
200	0	0.16	6.14	4.33
300	0	0.11	0.3	0.72
400	0	0.04	0.04	0.08

Table 6.3: *Results considering the 8-core processor using the Dynamic Clock Gating (DCG) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.*

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	100	99.92	94.14	91.11
400	100	100	99.85	92,59
600	100	100	100	97.08
800	100	100	100	99.58

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.03	5.11	7.77
400	0	0	0.21	6.09
600	0	0	0	2.24
800	0	0	0	0.18

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.37	5.27	8.51
400	0	0	0.41	6.09
600	0	0	0.17	4.16
800	0	0	0	0.77

each actuator and processor the results have been collected exploring two different design space directions: *i)* global set-point values and *ii)* number of executing applications. The use of different global set-points stresses the quality of the control scheme with a limited or a severely constrained budget. Differently, changing the number of executing applications stresses the quality of the control scheme for realistic scenarios, where the platform is not required to show its full computing power. In particular, an average power consumption of 500 mW and of 1000 mW have been observed when 4 and 8 applications are executing. To this extent, this work analyzed the quality of the control scheme considering four set-points for each processor that roughly correspond to 100%, 75%, 50% and 25% of the total required power. In the same way, this work considered scenarios where the platform is used at different fractions of its total computing capacity. The quad-core processor has been stressed at 25% (1 application), at 50% (2 applications), at 75% (3 applications) and at 100% (4 applications). Similarly the eight-core has been exercised considering scenarios where 1 (12.5%), 2 (25%), 4 (50%), or 8 (100%) applications have been executed.

Regardless of the employed actuator and the processor (quad- or eight-core architecture), all the three considered metrics follow the same trend. In particular, they degrade with the reduction of the budget and with the

Table 6.4: *Results considering the 8-core processor using the Dynamic Frequency Scaling (DFS) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.*

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	100	99.99	93.54	95.91
400	100	100	99.23	95.85
600	100	100	99.71	98.02
800	100	100	100	99.12

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.03	6.33	1.71
400	0	0.04	0.38	0.05
600	0	0.04	0.09	0.03
800	0	0.09	0.04	0.03

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.16	4.33	9.49
400	0	0.04	0.08	5.88
600	0	0.03	0.05	3.17
800	0	0.03	0.04	1.01

increase in the executing applications. This depends on the severity of the scenario. If the controller does not drive the actuation, instead, there is no penalty from a metrics' perspective. Moreover, the use of 4- and 8-core processors show the scalability of the proposed controller since the increase in the number of cores does not degrade the considered quality metrics.

Global efficiency - Considering the 4-core processor, the average efficiency is 98.27% with a minimum value of 92% and 93.94% using DFS and DCG, respectively. The worst case values are obtained when the set-point is limited to 100 mW and the required platform computing capacity is above 75%, i.e., 3 or 4 applications are running. Such scenario occurs because each application is most likely blocked by the control scheme that is limiting its performance due to the imposed set-point. The high scalability of the proposed scheme is testified by the performance of the 8-core processor. In particular, the average global efficiency is 95.09% and 97.23% using the DCG and DFS, respectively.

Global overflow - Considering the global overflow on the 4-core processor, i.e., OVF_g, the proposed control scheme is meant to align the total power consumption to the global set-point rather than keeping the former below the latter. In fact enforcing the latter condition penalizes the efficiency, although with the proposed design this work is trading efficiency

and overflow. The global overflow degrades when the control scheme operates under severe energy budget constraints, i.e., multiple running applications and low set-point value (100mW for 4-cores and 200mW for 8-cores). Such degradation is independent from the type of employed actuator and from the number of employed cores. For example, the worst case OVG_g are 5.5 mW and 13.42 mW using DCG and DFS for the 4-core CPU and 7.77 mW and 1.71 mW considering 8-cores with DCG and DFS actuators.
Global utility loss - The average global utility loss ($ULoss_g$) for the quad-core processor is 1.84% with a worst value of 8.88% and 9.22% using the DCG and DFS, respectively. Such metric shows a trend that is similar to the one observed for which the utility loss increases with lower set-points and an higher number of applications. In fact, such scenario is more likely constraining the applications that, thus, fall into the unbalanced set. As expected, an higher number of elements in the unbalanced set, makes the balancing process more difficult, thus lowering slightly the utility metric. A slightly increase in the utility loss, i.e., 1% must be acknowledged when the 8-core processor is considered. Such loss is due to the need of balancing a larger number of cores. As a second critical observation, the optimal value of the utility loss metric ($ULoss_g$) is determined by the supervisor algorithm that enforces the θ values. In particular, the θ values change in the order of 64 μs, thus showing a dynamic that is far slower than the latency of both the employed actuators. To this extent, even the DFS has the time to converge before the subsequent change of the θ values, thus showing results aligned to the one reported when the the DCG is used.

6.3.2 Dynamic scenario

This section discusses the results obtained from the execution of 4 applications on the 4-core processor employing the DCG actuator. It then considers the interaction between the controller, the OS and the applications. The final goal is to demonstrate the possibility to seamlessly integrate standard OS-level and application-level policies to be effectively actuated by the proposed controller. For example, the OS can impose a different global energy-budget at run-time or limit the energy quota on specific applications [10]. Differently, an application that is self-monitoring through emerging self-adaptive distributed management schemes, e.g., MARGOT [10], might need to change its energy quota to optimize its own quality-of-service.

It is worth considering that the definition of the QoS for an application or that of the policy needed to specify the global energy-budget, are out of scope here. Our work aims instead at making it possible to account for such decisions and constraints into the proposed control scheme. In

particular, a self-monitoring application can regularly ask to change its θ value depending on its application-level policy. The critical aspect to note is that such application behavior, seamlessly integrates with the proposed controller that fixes the θ required by the application. At the same time it is ensured that the energy-allocation policy implemented in the supervisor continues to manage the remaining running cores.

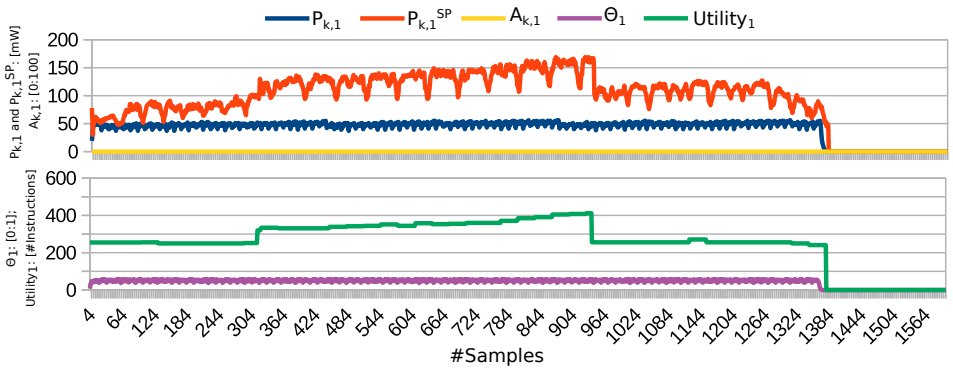
Results are reported in Figure 6.3 considering the signals of the 4 local controllers (see Figure 6.5a- 6.4d) and the single global one (see Figure 6.3e). According to the notation defined in Figure 6.1 for the control scheme, for each local controller i-th this work reported the power consumption ($P_{k,i}$), the power consumption set-point ($P_{k,i}^{SP}$), the actuation signal ($A_{k,i}$) as well as the utility ($Utility_i$) and the assigned θ value (Θ_i). Moreover, the total budget with slack ($P_k^{TotCorr,SP}$), the total consumed power (P_k^{Tot}), the slack (S_k^{Tot}) and the total power set-point ($P_k^{Tot,SP}$) are displayed for the global controller.

The proposed scenario allows to discuss three aspects. First, the OS is forcing a low θ_4 value between sample 310 and 364. Such action has two consequences. Local controller 4 starts actuating due to the reduced power set-point ($P_{k,i}$) for application 4 and thus a reduction in the utility for the same application can be observed. In contrast, the remaining applications experience an increase in their utility, due to the extra θ fraction removed from application 4 that has been redistributed among the other running applications.

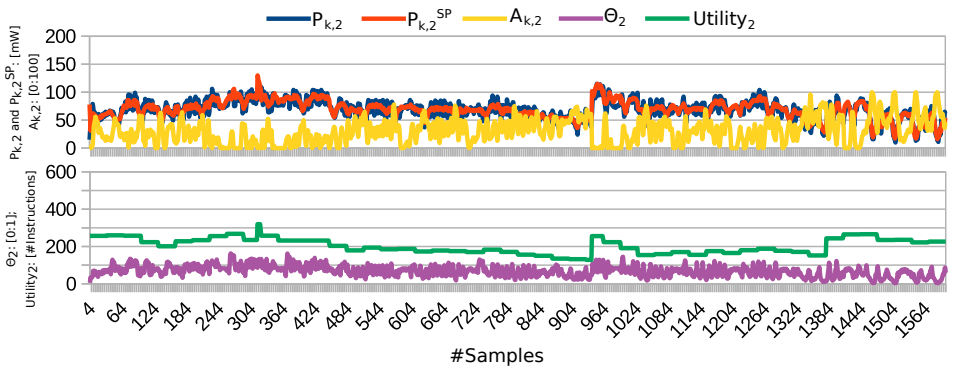
Second, the OS is reducing the global power budget at time sample 1264 with two distinct consequences. The total power set-point ($P_k^{TotCorr,SP}$) is gracefully lowering as a consequence of the energy budget buffer implemented in the global controller; in other words, $p_2 = 1$ generates an integrator in the global controller (see Figure 6.1). Such buffer ensures a graceful degradation of the performance of the running applications, while maintaining stable the difference between the sum of the global set-points and the consumed energy, i.e., the error (e_k^{Tot}) is 0 at steady state. In particular, the system is using the energy accumulated in the buffer until the reduction of the global set-point at time sample 1264. The energy in the buffer increases because the applications were not using all the available energy budget (see the increase of the S_k^{Tot} until sample 1264).

The last scenario is devoted to the evaluation of the system when application 1 terminates around sample 1384. The controller acknowledges the end of the application by removing its core from the set of unbalanced ones. To this extent, the energy budget allocated to the terminated application is distributed to the other three running applications, with a net increase in

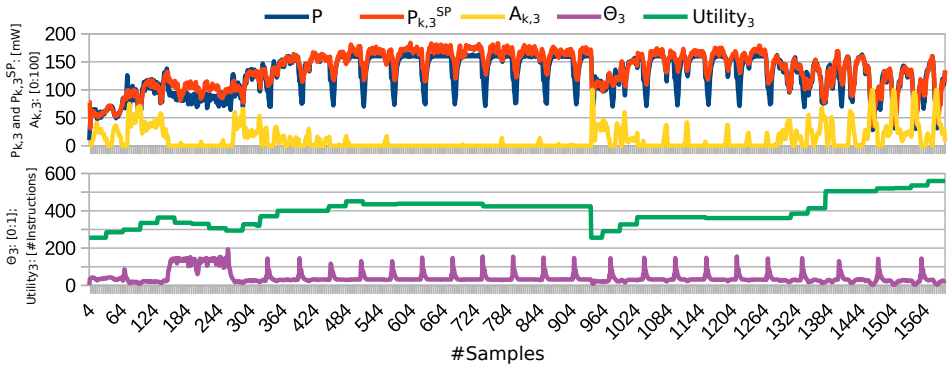
their utility. Such results demonstrate the adaptivity of the system to an external event.



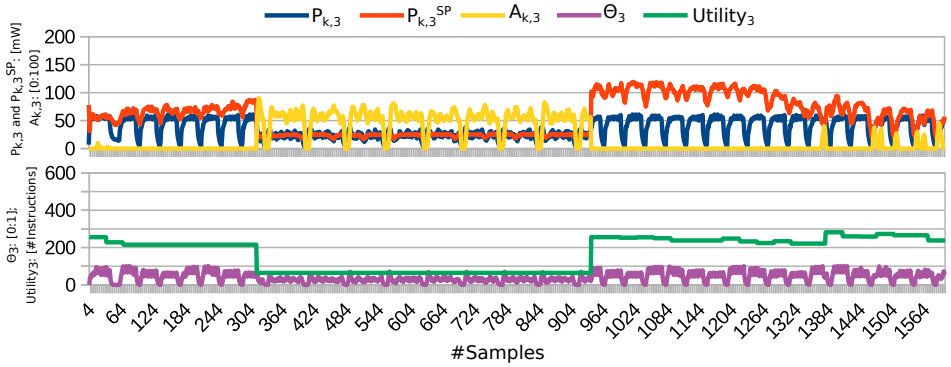
(a) *Local controller 1*



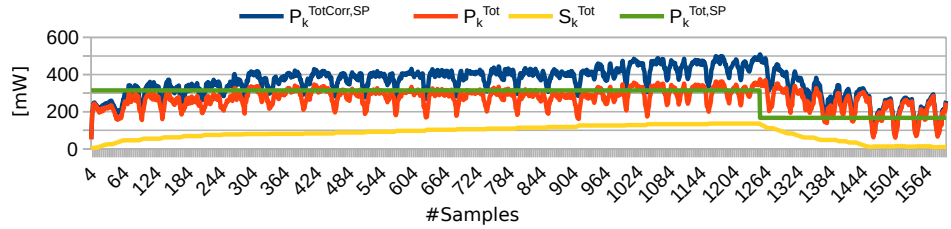
(b) *Local controller 2*



(c) *Local controller 3*



(d) *Local controller 4*



(e) *Global Controller*

Figure 6.3: *Evaluation over the time of the actuation and monitored signals for the global and the 4-local controllers considering a quad-core processor employing the DCG actuator. Application 4 imposes its own θ_4 value between sample 320 and 950 (see Figure 6.4d), application 1 terminates at sample 1390 (see Figure 6.5a) and OS imposes a new energy budget at sample 1264 (see Figure 6.3e).*

CHAPTER 7

Conclusions

This thesis presented a methodology to automatically instrument a resource-constrained power monitor into generic hardware designs. Furthermore, the identified power monitors have been adopted to feed a coordinated control scheme to optimize the energy-budget and energy-allocation for multi-cores, also ensuring the exponential stability of the overall system. The proposed power control scheme is neither an ad-hoc heuristic nor an energy-budgeting algorithm, but a complete framework to design any energy allocation policy. The online power monitors results have been validated considering both HLS-generated hardware accelerators as well as a RISC-V based SoC across a wide set of temporal resolutions ranging from 20us to 500us.

Depending on the imposed user-defined constraints and with respect to the unconstrained power monitoring state-of-the-art solutions, the proposed methodology shows a resource saving between 37.3% and 81% while the maximum average accuracy loss stays within 5%, i.e., using the aggressive 20us temporal resolution. However, by varying the temporal resolution closer to the value proposed in the state of the art, i.e. in the range of hundreds of microseconds, the average accuracy loss of our power monitors is lower than 1% with almost the same overheads.

In summary, the proposed constrained power monitoring infrastructure identification methodology allows to optimize the time-to-market by allowing the automatic integration of a power monitor into a target design, while allowing to flexibly trade between the accuracy of the estimate and the overheads.

The assessment of the identified power monitors have been performed by employing them within an all-digital control theory based power optimization scheme, which advances the state-of-the-art in three ways. First, it allows to cast any energy allocation policy as an algorithm of the supervisor controller while maintaining the global control scheme exponentially stable, regardless of the implemented policy. Second, the proposed controller can seamlessly integrate constraints from the OS and application level, also working with different actuators, still preserving the stability property. Third, the scalability of the proposed solution has been demonstrated through the use of a quad- and an eight-core processors as computing platforms. Moreover, the low-area overhead in the standard-logic implementation of the proposed controller, highlighted its flexibility and wide applicability.

The proposed scheme has been validated on a real prototype considering a quad- and an eight-core processor as computing platforms as well as two actuators, i.e., DFS and DCG. In particular, the complete design has been synthesized, placed and routed on a Nexys4-DDR board featuring a Xilinx Artix-7 100t FPGA chip and considering a 50 MHz clock since it is the maximum operating frequency supported by the quad-core. The obtained area occupation is limited to 0.86% (FFs) and 5.3% (LUTs) of the FPGA chip considering the reference quad-core. We use three metrics to assess the quality of the proposed control scheme: the respect of the imposed energy-budget (OVF_g), the performance loss due to control scheme ($ULoss_g$) and the quality in terms of how efficiently the granted energy budget (EFF_g) is exploited. In particular, results have been collected for a huge variety of realistic scenarios and the statistical significance has been considered by executing each scenario 30 times. The obtained results show valuable achievements: the average EFF_g is 98.27% (worst case 92.25%), the average OVF_g is 1.43 mW (worst case 13.42 mW) and the average $ULoss_g$ is 1.87% (worst case 9.22%).

APPENDIX $\mathcal{A}$

List of Publications

This appendix contains the list of scientific publications resulting from the research conducted during the PhD and presented in this thesis. Each paper is summarized with the following structure: **Title:** Title of the publication

Authors: *List of authors*

Publication venue: Journal, Proceedings, etc...

Year: Publication year or Conference year

Bibliographic info: Pages of the journal where the article ban be found

DOI: Digital Object Identifier

Thesis citation number: Reference to the citation ID of the thesis

References to thesis chapters: Section or chapter where the reference is used

Section A.1 lists the main papers related to this thesis and presented in the previous chapters. The author of this thesis contributed to the papers listed in Section A.1 both for the theoretical and for the experimental parts, even when he is not the first author. Section A.2 contains the papers not very related to this thesis, but representing the result of collaborations with other researchers or European Projects. Finally, Section **??** lists the papers currently under review.

A.1 Main Papers

- **Title:** PowerProbe: Run-time power modeling through automatic RTL instrumentation

  Authors: *D. Zoni, L. Cremona and W. Fornaciari*

  Publication venue: Design, Automation & Test in Europe Conference & Exhibition (DATE)

  Year: 2018

  Bibliographic info: pp. 749-754

  DOI: 10.23919/DATE.2018.8342106

  Thesis citation number: [44]

  References to thesis chapters: Section 2.1.2 and 3.3

- **Title:** PowerTap: All-digital power meter modeling for run-time power monitoring

  Authors: *Davide Zoni, Luca Cremona, Alessandro Cilardo, Mirko Gagliardi, William Fornaciari*

  Publication venue: Microprocessors and Microsystems

  Year: 2018

  Bibliographic info: Volume 63, Pages 128-139

  DOI: 10.1016/j.micpro.2018.07.007

  Thesis citation number: [47]

  References to thesis chapters: Section 3.3

- **Title:** "All-Digital Energy-Constrained Controller for General-Purpose Accelerators and CPUs

  Authors: *D. Zoni, L. Cremona and W. Fornaciari*

  Publication venue: IEEE Embedded Systems Letters

  Year: 2020

  Bibliographic info: vol. 12, no. 1, pp. 17-20

  DOI: 10.1109/TC.2019.2963859

  Thesis citation number: [49]

  References to thesis chapters: Section 5

- **Title:** All-Digital Control-Theoretic Scheme to Optimize Energy Budget and Allocation in Multi-Cores

  Authors: *D. Zoni, L. Cremona and W. Fornaciari*

  Publication venue: IEEE Transactions on Computers

  Year: 2020

  Bibliographic info: vol. 69, no. 5, pp. 706-721

  DOI: 10.1109/LES.2019.2914136

  Thesis citation number: [50]

  References to thesis chapters: Section 5

- **Title:** Automatic identification and hardware implementation of a resource-constrained power model for embedded systems

  Authors: *Luca Cremona, William Fornaciari, Davide Zoni*

  Publication venue: Sustainable Computing: Informatics and Systems

  Year: 2020

  Bibliographic info: ISSN 2210-5379

  DOI: 10.1016/j.suscom.2020.100467

  Thesis citation number: [8]

  References to thesis chapters: Section 4

- **Title:** Design of side-channel resistant power monitors

  Authors: *D. Zoni, L. Cremona and W. Fornaciari*

  Publication venue: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

  Year: Under review

  Bibliographic info: Under review

  DOI: Under review

  Thesis citation number: []

  References to thesis chapters: No references

A.2 Secondary Papers

- **Title:** DENA: A DVFS-Capable Heterogeneous NoC Architecture

Authors: *L. Cremona, W. Fornaciari, A. Marchese, M. Zanella and D. Zoni*

Publication venue: IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

Year: 2017

Bibliographic info: pp. 489-494

DOI: 10.1109/ISVLSI.2017.91

Thesis citation number: []

References to thesis chapters: No references

- **Title:** Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems

  Authors: *William Fornaciari, Giovanni Agosta, David Atienza, Carlo Brandolese, Leila Cammoun, Luca Cremona, Alessandro Cilardo, Albert Farres, JosÃ© Flich, Carles Hernandez, Michal Kulchewski, Simone Libutti, JosÃ© Maria MartÃnez, Giuseppe Massari, Ariel Oleksiak, Anna Pupykina, Federico Reghenzani, Rafael Tornero, Michele Zanella, Marina Zapater, Davide Zoni*

  Publication venue: 18th International conference on Embedded Computer Systems: Architectures, Modeling and Simulations

  Year: 2019

  Bibliographic info: Proceedings

  DOI: 10.1145/3229631.3239368

  Thesis citation number: []

  References to thesis chapters: No references

- **Title:** VGM-Bench: FPU Benchmark suite for Computer Vision, Computer Graphics, and Machine Learning applications

  Authors: *L. Cremona, W. Fornaciari, A. Galimberti, A. Romanoni, D. Zoni*

  Publication venue: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation

  Year: 2020

  Bibliographic info: pp 323-335

  DOI: Publishing

  Thesis citation number: []

References to thesis chapters: No reference

Bibliography

[1] S. Bhagavatula and B. Jung. A power sensor with 80ns response time for power management in microprocessors. In *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*, pages 1–4, Sept 2013.

[2] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[3] W. L. Bircher and L. K. John. Complete system power estimation using processor performance events. *IEEE Transactions on Computers*, 61(4):563–577, April 2012.

[4] W. L. Bircher and L. K. John. Complete system power estimation using processor performance events. *IEEE Transactions on Computers*, 61(4):563–577, April 2012.

[5] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. Regression-based rtl power modeling. *ACM Trans. Des. Autom. Electron. Syst.*, 5(3):337–372, July 2000.

[6] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. Understanding GPU power: A survey of profiling, modeling, and simulation methods. *ACM Comput. Surv.*, 49(3):41:1–41:27, September 2016.

[7] P. Chaparro, J. GonzÃ¡les, G. Magklis, Q. Cai, and A. GonzÃ¡lez. Understanding the thermal implications of multi-core architectures. *IEEE Transactions on Parallel and Distributed Systems*, 18(8):1055–1065, Aug 2007.

[8] Luca Cremona, William Fornaciari, and Davide Zoni. Automatic identification and hardware implementation of a resource-constrained power model for embedded systems. *Sustainable Computing: Informatics and Systems*, 29:100467, 2021.

[9] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[10] D. Gadioli, E. Vitali, G. Palermo, and C. Silvano. margot: A dynamic autotuning framework for self-aware approximate computing. *IEEE Transactions on Computers*, 68(5):713–728, May 2019.

[11] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and BjÃ¶rn Lisper. The malardalen wcet benchmarks: Past, present and future. In BjÃ¶rn Lisper, editor, *WCET*, volume 15 of *OASICS*, pages 136–146. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

Bibliography

[12] Joao P Hespanha and A Stephen Morse. Stability of switched systems with average dwell-time. In *Proceedings of the 38th IEEE conference on decision and control (Cat. No. 99CH36304)*, volume 3, pages 2655–2660. IEEE, 1999.

[13] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.

[14] Brian Jeff. Big.little system architecture from arm: saving power through heterogeneous multiprocessing and task context migration. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *DAC*, pages 1143–1146. ACM, 2012.

[15] Xiaofan Jiang, Prabal Dutta, David Culler, and Ion Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, IPSN '07, pages 186–195, New York, NY, USA, 2007. ACM.

[16] Jianlei Yang, Liwei Ma, Kang Zhao, Yici Cai, and Tin-Fook Ngai. Early stage real-time soc power estimation using rtl instrumentation. In *The 20th ASPDAC*, pages 779–784, Jan 2015.

[17] Jakub Krzywda, Ahmed Ali-Eldin, Trevor E. Carlson, Per-Olov Ostberg, and Erik Elmroth. Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling. *Future Generation Computer Systems*, 81:114 – 128, 2018.

[18] Simone Libutti, Giuseppe Massari, Patrick Bellasi, and William Fornaciari. Exploiting performance counters for energy efficient co-scheduling of mixed workloads on multi-core platforms. PARMA-DITAM '14, pages 27:27–27:32, New York, NY, USA, 2014. ACM.

[19] Rainer Kyburz M. Friedli, Francesco Paganini. Energy efficiency of the internet of things - technology and energy assessment report, 2016.

[20] Matthias Müller, Brian Whitney, Robert Henschel, and Kalyan Kumaran. *SPEC Benchmarks*, pages 1886–1893. Springer US, Boston, MA, 2011.

[21] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka. Statistical power modeling of GPU kernels using performance counters. In *International Conference on Green Computing*, pages 115–122, Aug 2010.

[22] M. Najem, P. Benoit, M. El Ahmad, G. Sassatelli, and L. Torres. A design-time method for building cost-effective run-time power monitoring. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(7):1153–1166, July 2017.

[23] OpenRISC Project. OpenRISC 1000 Architectural Manual. Technical report, OPENCORES.ORG, 2014.

[24] D. J. Pagliari, V. Peluso, Y. Chen, A. Calimera, E. Macii, and M. Poncino. All-digital embedded meters for on-line power estimation. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2018*, pages 743–748, March 2018.

[25] J. Peddersen and S. Parameswaran. CLIPPER: Counter-based low impact processor power estimation at run-time. In *2007 Asia and South Pacific Design Automation Conference*, pages 890–895, Jan 2007.

[26] C. Pilato and F. Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In *23rd Int. Conf. on Field programmable Logic and Applications*, pages 1–4, Sep. 2013.

[27] Mihai Pricopi, Thannirmalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Power-performance modeling on asymmetric multi-cores. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '13, pages 15:1–15:10, Piscataway, NJ, USA, 2013. IEEE Press.

[28] OpenRisc Project. Mor1kx Cappuccino: OpenRISC compliant SoC, 2016.

[29] D. Rodopoulos, F. Catthoor, and D. Soudris. Tackling performance variability due to ras mechanisms with pid-controlled dvfs. *IEEE Computer Architecture Letters*, 14(2):156–159, July 2015.

[30] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(12):882–886, 2013.

[31] M. Rogers-VallÃ©e, M. A. Cantin, L. Moss, and G. Bois. Ip characterization methodology for fast and accurate power consumption estimation at transactional level model. In *2010 IEEE International Conference on Computer Design*, pages 534–541, Oct 2010.

[32] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros. Splash-3: A properly synchronized benchmark suite for contemporary research. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 101–111, 2016.

[33] A. Sansottera, D. Zoni, P. Cremonesi, and W. Fornaciari. Consolidation of multi-tier workloads with performance and reliability constraints. In *2012 International Conference on High Performance Computing Simulation (HPCS)*, pages 74–83, July 2012.

[34] Giovanni Scotti and Davide Zoni. A fresh view on the microarchitectural design of fpga-based risc cpus in the iot era. *Journal of Low Power Electronics and Applications*, 9:19, 02 2019.

[35] Karan Singh, Major Bhadauria, and Sally A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009.

[36] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 960–965, March 2015.

[37] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett. Accurate and stable run-time power modeling for mobile and embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(1):106–119, Jan 2017.

[38] Clifford Wolf. Yosys open synthesis suite. `http://www.clifford.at/yosys/`.

[39] Xilinx. 7 Series FPGAs Clocking Resources. `https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf`.

[40] Xilinx. 7 Series FPGAs Data Sheet: Overview. `https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`.

[41] Xilinx. Digilent Nexys4-DDR. `https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start`.

[42] Xilinx. Vivado design suite.

[43] D. Zoni, L. Colombo, and W. Fornaciari. DarkCache: Energy-performance optimization of tiled multi-cores by adaptively power gating LLC banks. *ACM Trans. Archit. Code Optim.*, page 1:25, 2018.

[44] D. Zoni, L. Cremona, and W. Fornaciari. PowerProbe: Run-time power modeling through automatic RTL instrumentation. In *DATE 2018*, pages 749–754, March 2018.

[45] D. Zoni, J. Flich, and W. Fornaciari. Cutbuf: Buffer management and router design for traffic mixing in vnet-based nocs. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1603–1616, June 2016.

Bibliography

[46] Davide Zoni, Andrea Canidio, William Fornaciari, Panayiotis Englezakis, Chrysostomos Nicopoulos, and Yiannakis Sazeides. Blackout. *J. Parallel Distrib. Comput.*, 104(C):130–145, June 2017.

[47] Davide Zoni, Luca Cremona, Alessandro Cilardo, Mirko Gagliardi, and William Fornaciari. Powertap: All-digital power meter modeling for run-time power monitoring. *Microprocessors and Microsystems*, 63:12, 07 2018.

[48] Davide Zoni, Luca Cremona, Alessandro Cilardo, Mirko Gagliardi, and William Fornaciari. Powertap: All-digital power meter modeling for run-time power monitoring. *MICPRO*, 63:128 – 139, 2018.

[49] Davide Zoni, Luca Cremona, and William Fornaciari. All-digital energy-constrained controller for general-purpose accelerators and cpus. *IEEE Embedded Systems Letters*, PP:1–1, 04 2019.

[50] Davide Zoni, Luca Cremona, and William Fornaciari. All-digital control-theoretic scheme to optimize energy budget and allocation in multi-cores. *IEEE Transactions on Computers*, PP:1–1, 01 2020.

[51] Davide Zoni and William Fornaciari. Modeling dvfs and power-gating actuators for cycle-accurate noc-based simulators. *J. Emerg. Technol. Comput. Syst.*, 12(3):27:1–27:24, September 2015.

[52] Davide Zoni, Federico Terraneo, and William Fornaciari. A control-based methodology for power-performance optimization in NoCs exploiting dvfs. *Journal of Systems Architecture*, 61(5):197 – 209, 2015.

Acronyms

ALU: Arithmetic and Logic Unit
ASIC: Application Specific Integrated Circuit
CAGR: Compounded Average Growth Rate
CMOS: Complementary Metal-Oxide Semiconductor
CPI: Clock Per Instruction
CPU: Central Processing Unit
DCG: Dynamic Clock Gating
DFS: Dynamic Frequency Scaling
DMA: Direct Memory Access
DSP: Digital Signal Processing
DTPM: Dynamic Thermal and Power Management
DVFS: Dynamic Voltage and Frequency Scaling
EFF: Efficiency
FF: Flip Flop
FPGA: Field Programmable Gate Array
FPU: Floating Point Unit
FSM: Finite State Machine
GPU: Graphic Processing Unit
HDL: Hardware Description Language
HLS: High Level Synthesis
HPC: High Performance Computing
HWC: Hamming Weight Count
IEA: International Energy Agency
I/O: Input/Output
IoT: Internet of Things
LLC: Last Level Cache
LUT: Look Up Table
MAE: Mean Average Error
MARS: Multivariate Adaptive Regression Splines
MMCM: Mixed Mode Clock Manager
MPSoC: Multi Processor System on Chip
MRE: Mean Relative Error

Bibliography

NN: Neural Networks
OVF: Overflow
PID: Proportional Integral Derivative
RISC: Reduced Instruction Set Computing
RMSE: Root Mean Square Error
RTL: Register Transfer Level
SACM: Switching Activity Counting Mode
SIMD: Single Instruction Multiple Data
SIMO: Single Input Multiple Outputs
SISO: Single Input Single Output
STC: Single Toggle Count
SP: Set Point
TWh: TeraWatt Hour(s)
SBC: Single Bit Count
SoC: System on Chip
SVD: Singular Value Decomposition
USD: United States Dollar
VCD: Value Change Dump
VDD: Voltage Drain Drain
WCET: Worst Case Execution Time