Specification Analysis for System-Level Power-Aware ASIC Design

Doctoral Thesis

David Ariel Lemma

Adviser: Prof. Dr. Rolf Drechsler

Universität Bremen

Specification Analysis for System-Level Power-Aware ASIC Design

David Ariel Lemma

A Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Engineering
- Dr.-Ing. -

Date of doctoral colloquium:
21.09.2021

Abstract

The field of *Electronic Design Automation* (EDA) is a growing field whose growth is fueled, among many factors, by the ever increasing number of *Application Specific Integrated Circuits* (ASIC) that are required by several digital systems. Amidst the current explosion in the design and production of *Internet of Things* (IoT) (typically battery powered devices), the need for power-aware design has come to reach critical importance. The need for a holistic view (power-aware) of the implications and challenges brought forth by the need to curb and properly manage power consumption power has revealed a need for the EDA field to explore system-level top-down based methodologies. As a response to the needs, this research document presents two frameworks based on specification analysis at the system-level in order to address the holistic view of power-aware design. One of the frameworks focuses on system-level specifications written in natural language (ie. English), whereas the other focuses on system-level specifications written in a technical language (ie. SystemC). The frameworks are shown to be able to analyze these types of system-level specifications as a means to aid designers in power-aware ASIC design at the system level. These frameworks encapsulate the contributions of this thesis, which are defined by the proper devising of analytical rules to parse the system-level specification so as to extract the basic underlying *Power Management Strategy* (PMS) laid out of by the specification under analysis. Use cases that show the effectiveness of the frameworks in aiding designers include typical ASIC elements such as a bus, a port processor, encoders and a processor centric programmable *System-on-Chip* (SoC), all of them also typical components of IoT devices. The research document ends with a conclusion both summarizing the work and pointing to possible future extensions to the frameworks and general research lines that are possibles avenues for future developments in the field of specification analysis for power-aware system-level ASIC design.

Acknowledgements

I would like to extend my gratitude to many people who have contributed to the completion of my doctoral journey with this doctoral thesis.

Firstly, I would like to express my thankfulness to Prof. Dr. Rolf Drechsler for the opportunity to do research work in AGRA, a group of very warm people and strong research output. It has been an experience that I shall never forget.

Secondly, my sincere appreciation goes to Prof. Dr-Ing Daniel Große and Dr. Mehran Goli. Simply put, without their assistance in research, co-authorship of papers and general advice, this thesis would not have been possible.

Thirdly, I would like to extend my greeting and well wishes to the entire membership of AGRA and to the members of CPS with whom I've shared more than 4 years of daily work.

Lastly, but certainly not least, I owe an immense debt of gratitude to my parents. Without their constant care, love and advice, this whole experience would have been duller and significantly less enjoyable.

Contents

List of Figures

List of Tables

1 Introduction and Motivation

In design, one of the most difficult activities is to get the specifications right [1]

— **Donald A. Norman**
(Professor. Usability engineering and design expert)

1.1 Introduction

Much has transpired since the origins of the *Integrated Circuit* (IC) design industry. 50 years after Gordon Moore first famously observed and proposed the "law" that bears his name, explaining how the number of transistors doubles every year, the IC industry has further developed following the upsurge in the demand for the highest processing power in the smallest possible package. What began in the late 1940s as a way to combine discrete electronic components to produce amplification devices, took a far more generic turn when Jack Kilby showed a working monolithic integrated circuit at the end of the 1950s [2]. While trying to find a way to improve the reliability of the interconnected modules made of discrete components (thereby offering a way to solve the so called "Tyranny of numbers"), Jack Kilby ushered an era in which the electronic circuits grew in complexity, reliability and performance and decreased in size and price[3].

Onto the 1960s and 1970s, the applications of the IC products grew, as their design and manufacturing costs diminished. The evolution of the IC design industry in these decades, also led to the emergence, development and consolidation of the *Electronic Design Automation* (EDA) field, whose techniques and tools supported the ever increasing rate of development. The automation, which began with tools for placement and routing (that is to say, mainly layout tasks), took a massive leap forward in the 1980s, when the number of integrated components in ICs boomed into the hundreds of thousands. Indeed, by the early 1980s, the relatively artisanal way in which IC design had been conducted for the previous two decades, had been transformed by systematization, spearheaded by the work of pioneers such as Carver Mead and Lynn Conway [4].

As the complexity of IC products was increasing, the EDA industry found itself needing to address an equally increasing number of recurring concerns in the design flow. Many of these concerns were no longer only associated with the layout tasks that had been the object of most of the automation techniques being developed thus far [5]. As the IC design industry moved away from focusing solely on the placement and routing phases of a design, ad-hoc design languages became the preferred way to deal with challenges brought up by the need for verification (assurance of compliance of the design with an specification) and validation (assurance of addressing the need that the design is supposed to meet) [6].

The main companies in the EDA industry became closely acquainted with the use of simulation methods as a viable approach to addressing the demands and requirements of an industry with ever expanding fields of application. What had first started as transistor and logic simulation, incorporated functional simulation, where the increasing need for architectural verification was more appropriately met [7]. Even before that time, the IC design industry had also begun to eye and behold tools and techniques of formal verification (such as model checking and temporal logics) as a more suitable way to address the stringent safety and security requirements of some industries in which ICs were being deployed (as become the case with the aerospace and defense industries).

By the 1980s, the IC industry had been the main driving force behind inherently programmable microprocessors and other general purpose highly configurable computing platforms, which the public had begun to realize were at the core of their digital electronic devices. However, in addition to enabling the design of highly complex comprehensive digital information processing circuits, the EDA community (now a blossoming industry of its own), also enabled the development of more specialized ICs, which became known under the moniker *Application Specific Integrated Circuit* (ASIC). The ASIC world benefited immensely from the creation of *Hardware Description Languages* (HDL), for this mirrored the flexibility of platform independent high-level software programming languages, thereby allowing designers to think of their circuits in increasingly more abstract terms [8] [9].

The creation of HDLs led to a shift away from the transistor and logical levels into the *Register Transfer Level* (RTL), which better suited the design flow from a top-down perspective. The Register Transfer Level not only allowed for a fairly common physical synthesis agnostic platform (as per abstracting away the design from the choice of the underlying transistor and gate technology), but also allowed for a way to design an IC from a more architectural point of view [10]. Such a point of view led to more system-wide approaches to the validation and verification challenges. Further expansion of the integration of ICs (many of them ASICs) to create full fledged digital *System on Chips* (SoCs) unveiled a pressing need for system-level approaches to address not only validation and verification, but also requirements that were themselves at a system level [11].

The emergence of SoCs (fueled by the need for self contained digital systems) meant that many ASICs turned into *Application Specific Standard Products* (ASSP) [12]. These products were and continue to be, ICs performing specialized standardized tasks typically present in many SoCs (such as buses or encoders/decoders). The specialized and standardized nature of these ICs is typically underlined by elements of their design such as an archetypal architecture. Furthermore, such a characteristic architecture of an ASSP is the natural byproduct of a detailed specification that the design has followed. The necessity of following and conforming to a specification (essentially a set of requirements to comply with) is at the core of the design of these ICs.

1.2 Motivation

Regardless of whether the IC is an ASSP or an ASIC, following and conforming to a specification is paramount. However, as crucial as this precept is in the design flow for any IC, usually the task associated with complying with a specification are quite demanding. The main difficulties are almost always associated with the ambiguities and the extent of the specification. Specifications are rarely of a formal nature (ie. described in a logical formulation), but are instead typically produced in a natural language (ie. English) or in a technical language (ie. a HDL), which captures requirements in more generic concepts. What is more, specifications are the starting point of almost all top down design flows, in which the architectural decisions are taken in early stages [13].

By beginning of the 21st century, technical standards (that is to say, standardized specifications) had grown to cover most, if not all, of the functional concerns related to any ASIC (or ASSP). Verification and validation of the functional requirements had escalated to system level, beyond RTL, in an effort to reduce the iteration cycle required by successive refinements to fulfill the requirements. In these system levels (such as for instance, the *Electronic System Level* (ESL), which is known for the presence of Virtual Prototypes, the *SystemC* technical language [14]) and the

Transaction Level Modeling (TLM) methodology [15], the EDA community has strived to ensure reliability in functionality by expanding the use of formal methods and by extending coverage metrics from lower levels of abstraction.

Also by the turn of the millennium, it became discernible that functional requirements were no longer the only sine qua non requirements stemming from specifications. Security, safety and power/energy consumption concerns had risen to rival the sense of priority accorded to functionality, to the point that it had become increasingly impossible to design an IC with only functionality in mind. Indeed, if the prototype for an IC did not meet the security provisions or the power or energy budget, the prototype was to be rejected. Such a heightened importance of non functional requirements, meant that top down design flows were then in need of some changes to reflect such an evolution [16].

While the importance of security and safety in ICs (especially so regarding data protection) is unquestionable, the evolution of the design flow to address power/energy consumption concerns has shown itself to be a top priority. From the transistor level up to a system-level, many of the innovations introduced to deal with the power/energy concerns, have been the result of wide ranging research leading to the concept of power-aware design. With the advent of the *Internet of Things* (IoT), in which ASICs play a central role, these innovations that have brought up power-aware design are shaping up a renovated ASIC design flow [17]. Not unlike the functionality focused design flows, the focus of power aware design techniques has shifted towards system-level architectural decision making. Owing to such a shift, and going beyond the reach of traditional HDLs designed to implement existing designs, specification analysis (often the first step in top down design flows) is becoming a focus area in the coming years [18].

1.3 Contributions

The contributions of this doctoral thesis are centered on addressing some of the challenges of system-level power-aware ASIC design via focus on specification analysis, for both specifications in natural and technical language [19] [20]. For most intents and purposes, the intention is to help usher a new stage in the revamping of top down ASIC design flows. In such a new stage, specifications written in natural/technical languages are considered the foundational design documents at the system-level. As such, specifications (whether technical standards or functional descriptions) are the main source of information for power -aware decision making, significantly integrating functionality and power/energy consumption concerns.

This document presents two contributions:

- A response framework to process (parse and analyze) any natural language specification (*in English*) in order to unveil the intrinsic power structure of a prototype that complies with said specification. The originality of the framework lies in its use of semantic analysis techniques and associated rules to be able to infer the innate power structure that is produced by any prototype following a functional description, typically found in the specification. This power structure (represented by a set of numerical parameters) allows for rapid power-aware exploration of a design, as well as working as a baseline comparison point for decision making regarding the power/energy consumption concerns. The core of the contribution is centered around the associated rules that enable the inference.

- A response framework to process (parse and analyze) any technical language specification (*in SystemC/TLM*) so as to reveal the underlying power architecture that a Virtual Prototype (acting as a functional description) produces by the mere fact of implementing its intended functionality. The originality of the framework lies in a set of algorithms that can extract numerical parameters describing the underlying power architecture. This power architecture comes with both benefits (in reduced power consumption) as well as costs (power and area overhead arising from the need to manage the modules of the IC with extra logic) and these are meant to be taken into consideration as part of the *Design Space Exploration* (DSE) at the system-level in order to decide on the most appropriate power-aware architecture for the design.

Both frameworks constitute a large step towards system-level power-aware specification analysis for ASIC design, as they are able to significantly automate (in as much as possible) tedious tasks that have made system-level power-aware design a laborious process highly dependent on the expertise of seasoned designers. The assistance provided by the frameworks to designers is to be thought of as a systematization of a portion of the designer's system-level expertise within the frameworks.

Part of the research work that leads to the framework has been previously presented in several publications that have been produced throughout the doctoral process. Any content of this document that is not attributed to a third party is either present verbatim in these publications or is a modified form of some content within said publications and is therefore consider own work. The publications are listed as:

1. David Lemma, Mehran Goli, Daniel Große, and Rolf Drechsler. “Power intent frominitial ESL prototypes: Extracting power management parameters”. *IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. 2018.

2. David Lemma, Daniel Große, and Rolf Drechsler. “Natural Language Based Power Domain Partitioning”. *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 2018.

3. David Lemma, Mehran Goli, Daniel Große, and Rolf Drechsler. “Towards Generation of a Programmable Power Management Unit at the Electronic System Level”. *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 2020.

1.4 Thesis Structure

After this introductory chapter (Chapter 1), the research work laid out in this document is structured as follows:

- Chapter 2 presents power-aware design as a concept. After a small prelude containing the rationale behind the concept, the implications and challenges in the power-aware design realm are put forward. Further into the chapter, the reader is introduced to the idea of a *Power Management Unit* (the most typical response to the management of the power/energy implications and challenges) with its main defining parameters: the *Power Domains* (PD), and the *Control Signals* (CS) and *Power Modes* (PM). An effective way to determine the PMU parameters from a system-level specification (in natural or technical language) is then shown to be crucial for a successful power-aware design.

- Part I concerns itself with the first contribution (*a framework for the processing of natural language specifications*). Part I consists of a theoretical chapter (Chapter 3) that explains the basics of specification analysis as performed for those documents when they are written in a natural language (English). In addition, there is a relevant works section and a presentation of the framework, based on *Information Extraction* (IE) and semantic analysis techniques (based on a recently developed grammar annotation scheme); and an application chapter (Chapter 4) that shows the applicability of the framework of the previous chapter to the unveiling of the inner power structure caused by the specification in terms of some or all PMU parameters (PD, PM and CS) for ASIC type use cases.

- Part II deals with the second contribution listed (*how to process a technical language specification in a power-aware fashion*). Part II is made of: a theoretical chapter (Chapter 5) that explains the basics of specification analysis as performed for those documents in a technical language at the system-level (Virtual Prototypes in SystemC/TLM). This chapter also contains a section on the relevant works in the field and a presentation of the theoretical framework, based on *Design Understanding* (DU) techniques and tools; and an application chapter (Chapter 6) that shows how the theoretical framework presented in the previous chapter allows for the unveiling of the intrinsic power structure produced by the Virtual Prototypes, a comparison between potential alternatives and an estimation of the effect of the unveiled power structure in the IC's power consumption.
- Chapter 7 brings forth a conclusion to the research work and offers some insights into possible future developments along several research lines branching from the research work.

2 Power-Aware Design: Implications, Challenges and Responses

> *Power has become the number one problem. All design targets are being constrained by power[21].*
>
> — **Gary Smith**
> (Electronic Design Automation analyst)

2.1 Preface

The proliferation of battery powered devices at the end of the 20th century, coupled with an increased awareness of the need to manage power/energy consumption, have been major causes spurring the development of power-aware design for ASICs.

While power consumption and energy consumption are not equivalent, they are both metrics that give information about the fulfilling of the requirements associated with the physical variables involved in the calculation of power/energy. Since energy is a variable derived from power (being the sum of power consumption over a given time period), the latter is the most usual target of many of the techniques developed by the IC design industry. However, power-aware methodologies also consist of techniques that help designers manage energy consumption, via the management of power throughout varying periods of operation, so as to meet any applicable energy budget. (In this document the word power will serve as the generalizing adjective that encompasses both power and energy as described above).

Power, as a magnitude for IC design, can be calculated or estimated by the use of a primary simplified equation. This equation, Eq. 2.1, contains two terms: **static power** and **dynamic power**, which are themselves defined by other simplified equations with their own terms in Eq. 2.2 and Eq. 2.3. **Static power** encompasses the dissipation of power produced by the IC by virtue of it being connected to an energy source. *Dynamic power* encompasses the dissipation of power by virtue of the IC switching its components according to operational need.

$$P_{total} = P_{static} + P_{dynamic} \tag{2.1}$$

$$P_{static} = V_{cc} * I_{leak} \tag{2.2}$$

$$P_{dynamic} = C * f * {V_{cc}}^2 \tag{2.3}$$

Inside Eq. 2.2 and Eq. 2.3 are several physical variables whose estimated or calculated values are related to the physical synthesis of the circuit (such as voltage (V_{cc}), capacitance (C), frequency (f) and current (I_{leak}). In the case of ASICs some of these variables are occasionally dictated and regulated in the specification (typically a technical standard). Regardless of whether or not the variables are directly set in some specification, the physical synthesis stage of the IC design process requires that they are known, as they are crucial in the evaluation of compliance with a power budget.

The terms in the equations are averages (given the simplified nature of the equations), but such equations could be used to obtain a value for power dissipation for every transistor in the IC. Such a value would be extremely accurate, regardless of the fact that is was obtained through the use of averages. This level of accuracy in power dissipation values is typically not feasible in the current state of IC design, since the number of transistors is well beyond the millions. However, provided that the time and computational resources are available, the accuracy of the power dissipation values can be guaranteed to be extremely high at physical synthesis stage.

As power/energy consumption became a crucial design requirement, the way to address the requisite was initially through directly targeting the physically significant variables that affected power dissipation. The techniques developed to manage the physically significant variables constituted the core of what became known as low-power design methodologies [22]. Indeed, by the 1990s and even before that time, power consumption management was characterized by the use of algorithms and heuristic rules that minimized power dissipation as much as it was possible. These algorithms and heuristic rules were focused on lower levels of abstraction (transistor and gate level) where the variables could be directly managed. Furthermore, the ways in which power dissipation was minimized were mostly orthogonal (that is, independent) of other implications and requirements in the design process.

However, with the growing complexity of ASICs (especially ASSPs), their specifications began to state requirements that greatly influenced each other. This turn of events led to low power design methodologies beginning to fall short of the global integrated approach which came to be required. The EDA industry responded to the challenge by putting forward a renewed set of methodologies, which became known as power-aware design. Closely linked to low power design methodologies, power-aware design and low-power design were not exactly addressing the same concerns. A power-aware system was (and is) not always a low-power system.

As stated by Unsal and Koren [23]:

> *It is important to note the conceptual difference between power-aware and low-power systems. In low-power design, the main goal is minimization of power. On the other hand, a power-aware system is one in which meeting power and energy goals is a significant design consideration[...]*

Low-power design methodologies are usually part of power-aware design methodologies, but do not constitute the totality of them. Furthermore, power-aware design may even preclude the use of some low power design techniques.

Power awareness in methodologies for IC design is meant to meet the power/energy requirements and concerns in a comprehensive fashion. In the words of Pedram and Rabaey [24]:

> *Just as with performance, power awareness requires careful design at several levels of abstraction. The design of a system starts from the specification of the system functionality and performance requirements and proceeds through a number of design levels spanning across architectural design, register transfer level design, and gate level design, finally resulting in a layout realization.*

As per the shift to power-aware design methodologies, many of the associated techniques ceased to be only concerned with the direct management of the physically relevant variables at lower levels of abstraction. In raising to gate level and even to the *Register Transfer Level* (RTL), both the detailed information of physical synthesis stage and the ability to directly control the physically relevant variables are reduced.

In higher levels in the abstraction ladder (for instance, a system level such as the ESL-*Electronic System Level*) the accuracy in the calculation/estimation of power dissipation is lowered as per the increased abstraction. The reduced accuracy is the result of lacking the detailed information that is available in physical synthesis stages. At the same time, the effect on the management of power consumption of whatever techniques used by the power-aware methodology at hand becomes increasingly noticeable. In many cases, techniques that are part of the power-aware design methodology at system-level are less focused on optimization of a given architecture than they are about being useful to architectural decision making. The impact of architectural decision making can be shown via a design level ladder featuring the different the different design stages.

2.2 Implications

As can be seen in Fig. 2.1, the nature of the design level ladder is well represented by pyramids depicting the power saving opportunity and the optimization effort involved for each level, from the system level to the physical level. Following a top-down perspective, it is becomes clear that on a system level, the impact of the techniques used by a power-aware methodology is at its highest, with the effect (power saving opportunity) not only decreasing significantly in lower levels of abstraction, but also requiring much more optimization effort. While the accuracy in the estimation of the effect at higher levels of abstraction is much lower than closer to implementation levels (eg. gate level), the dissimilar orders of magnitude of the estimated effects make it a compelling reason for the EDA industry to focus heavily on sound architectural decision making techniques.

Architectural decisions typically encompass a process of specification analysis that is done within a set of processes that the IC design industry labels *Design Space Exploration* (DSE) [26] [27]. DSE is a set of processes that attempt to compare and contrast different architectural alternatives that follow the specification for a given design. Typically, low abstraction level DSE is the realm of multi objective optimization algorithms with special focus on finding the optimal solution from within the different architectural alternatives [28]. In order to perform DSE at a high abstraction level it is paramount to be aware of the intended functionality of a system. Such an awareness typically entails knowing how the functional tasks of the design are implemented [29]. Both the originally devised architectural alternatives and the potential optimized solutions are presented in terms of the components at different level of abstraction, as components encapsulate and abstract away many (if not all) of the design factors present at the set of simplified power equations previously presented.

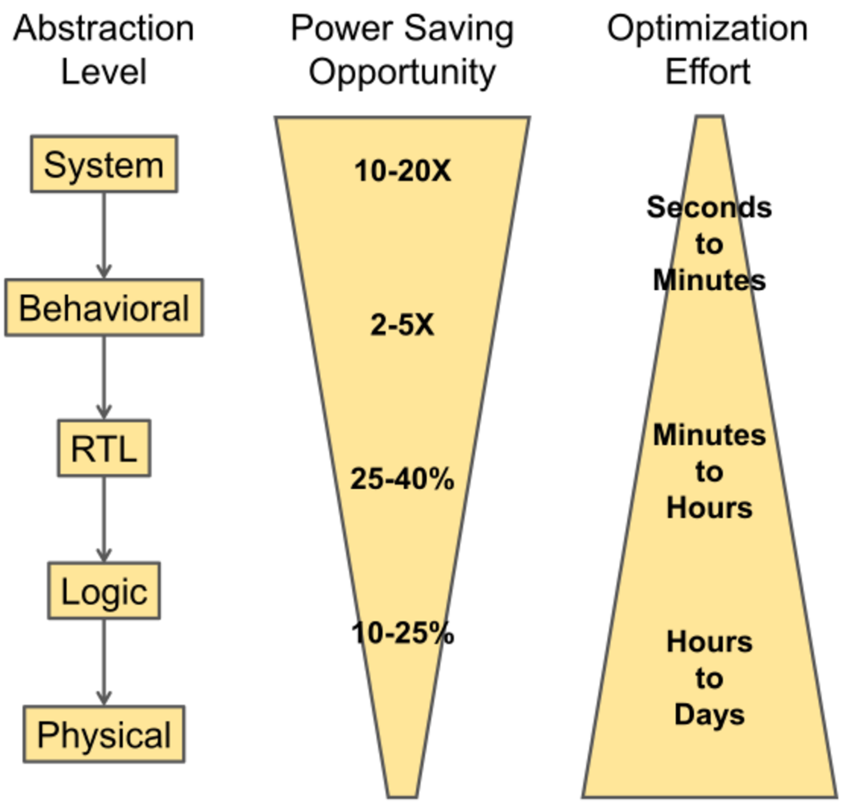


Fig. 2.1: Design Level Ladder and its Impact on Power from [25]

Based on the concept of component, the implications of a proper power-aware DSE process become easier to describe. Consequently, to conduct power-aware DSE on a given system provided by an IC it becomes necessary to analyze the impact of the components on the way power is consumed and on how it should be managed. The power structure, which is the byproduct of the intended functionality of a system, is defined by the boundaries and interactivity of the components and needs to be properly known. Whatever chosen arrangement (architecture) of the components, as well as the power structure they entail, needs to also be evaluated in light of both their fulfillment of the functional tasks required by the design and their impact on the way power is managed by the design (for its benefits and associated costs).

For the results of power-aware DSE to be of any practical use, the functional tasks that the system should be themselves be examined through a profiling process. The process of profiling the functional tasks can be summarized as the characterization of the active/inactive periods of the systems as dictated by the fulfillment of the tasks. The profiling process done on the functional tasks can be performed independently of any decision concerning the components, but the former affects the latter and

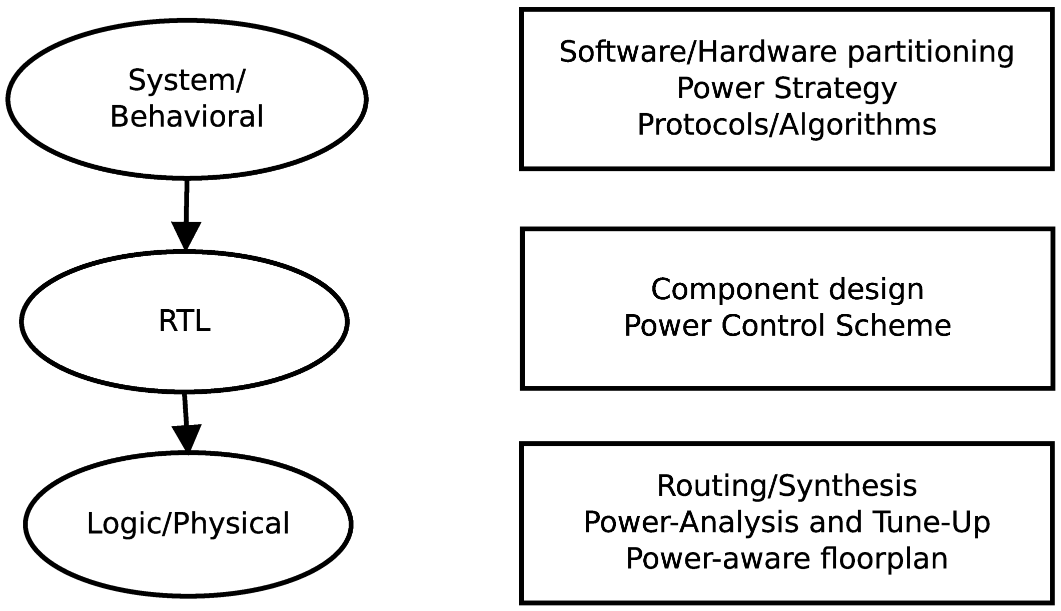


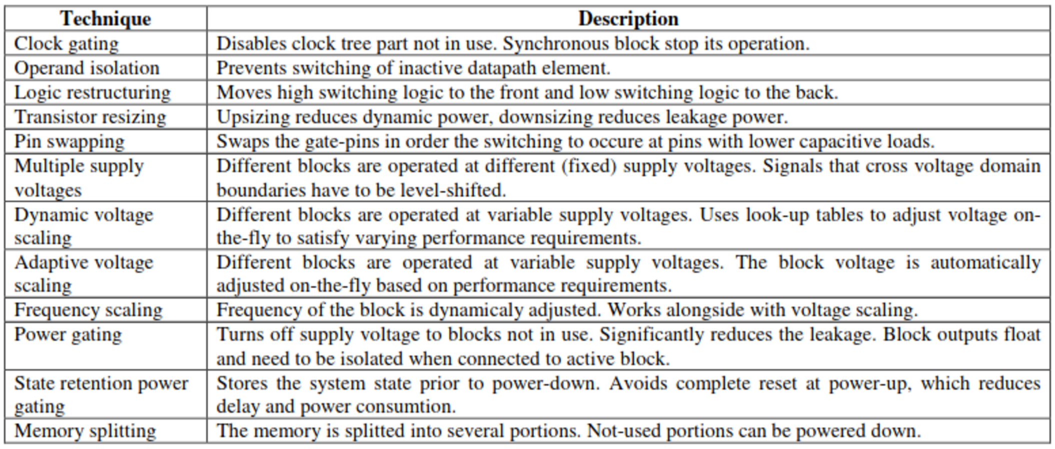
Fig. 2.2: Design Levels and Design Tasks Associated with Them based on [30]

should be thought of as the main starting point of power-aware DSE at higher levels of abstraction (such as the system levels).

2.3 Challenges

The series of steps that summarize power-aware DSE are summarized in Fig. 2.2. By looking at it, it becomes noticeable that the step at the top of the figure, under *System Design*, deals with tasks of planning nature that are run at system level, establishing the architecture of the IC. Among the tasks is deciding *Power Strategy*, which is obviously the most important factor when designing an architecture (a power structure) to back said strategy. Whatever functionality decision taken at the beginning stages of the IC design process is linked to a similar decision related to power at a similar stage. For instance, once the design reaches the RTL stage, the *Power Control Scheme* needs to be decided on and subsequently verified. Furthermore, in the implementation stage, which encompasses the logic and physical level in Fig. 2.1, *Power Analysis and Tune-up* are important power-aware tasks, alongside deciding on the *Power-aware floorplan*.

Power-aware design methodologies must enable sound architectural decisions that meet both functional and power related requirements. In addition to the need for judicious architectural decisions that fulfill both functional and power related requirements, whatever resulting byproduct of these decisions, should be analyzed in



Technique	Description
Clock gating	Disables clock tree part not in use. Synchronous block stop its operation.
Operand isolation	Prevents switching of inactive datapath element.
Logic restructuring	Moves high switching logic to the front and low switching logic to the back.
Transistor resizing	Upsizing reduces dynamic power, downsizing reduces leakage power.
Pin swapping	Swaps the gate-pins in order the switching to occure at pins with lower capacitive loads.
Multiple supply voltages	Different blocks are operated at different (fixed) supply voltages. Signals that cross voltage domain boundaries have to be level-shifted.
Dynamic voltage scaling	Different blocks are operated at variable supply voltages. Uses look-up tables to adjust voltage on-the-fly to satisfy varying performance requirements.
Adaptive voltage scaling	Different blocks are operated at variable supply voltages. The block voltage is automatically adjusted on-the-fly based on performance requirements.
Frequency scaling	Frequency of the block is dynamicaly adjusted. Works alongside with voltage scaling.
Power gating	Turns off supply voltage to blocks not in use. Significantly reduces the leakage. Block outputs float and need to be isolated when connected to active block.
State retention power gating	Stores the system state prior to power-down. Avoids complete reset at power-up, which reduces delay and power consumtion.
Memory splitting	The memory is splitted into several portions. Not-used portions can be powered down.

Fig. 2.3: Frequent Power management Techniques from [31]

comparison to potential alternatives. Typically, high level decisions of architectural nature reduce the number of potential alternatives in the design, which implies the use of a particular set of power management techniques. A set of power management techniques applied in a defined way constitute a power strategy, whose effectiveness and associated costs should be properly analyzed.

A *Power Management Strategy* (PMS) (also known as a Power Strategy) is pursued by the rational application of the adequate set of power management techniques. Some of these techniques (presented by D. Macko [31]) are listed in Fig. 2.3. Many such techniques are susceptible to classification under the abstraction level which they most prominently target. Typically, techniques related to direct effects on physical variables that are responsible for power dissipation (eg. voltage, capacitance and current) are targeted at transistor and gate levels. Techniques that indirectly affect the physical variables are typically used at higher levels of abstraction (eg. the RTL and system levels). In any case, techniques are to be applied throughout the entire design process [32] [33].

Any set of power management techniques may also be sorted into categories according to the term (dynamic or static power) of Eq.2.1, that they most directly impact. Such a simplified classification is neither unreasonable nor arbitrary, since the concepts of dynamic and static power remain extremely relevant in the general literature in the field. A key example of the relevancy of classifying power management techniques according to their influence in terms of the more generalized dynamic or static power can be seen in Table 2.1. In this table, besides their influence on dynamic or static power, the impact of some of these techniques in area and power overhead, as well as in the overall design are also shown to be different. It is then evident that power management techniques need to be used concurrently for most IC.

Tab. 2.1: Some Power Management Techniques and Their Impact

Power Management technique	Static Power influence	Dynamic Power influence	Area overhead	Power overhead	Impact on overall design
Clock Gating	0	Very high	Low	Low	Medium
Power Gating	Very high	0	Medium	Medium	High
Dynamic (Voltage) Frequency Scaling	Medium	High	High	Medium	Very high

While (re)producing an exhaustive list of power management techniques is not within the purview of this chapter, the techniques listed in Table 2.1 are further explained in the next paragraphs. These power management techniques are well known for their impact extending all throughout the abstraction ladder. It is because of the influence of these techniques on the effectiveness of the PMS that an intelligent application of them is of extreme importance. The techniques are: *Clock Gating* (CG) and *Power Gating* (PG) and *Dynamic (Voltage) Frequency Scaling* (DVFS):

Clock Gating (CG): is a technique based on switching off the clock signal for any given component whenever there is no need for said component to switch its internal state. The granularity with which the technique is applied can vary from gate level to RTL and beyond. As the signal under management is that of the clock, the dynamic power term of the total power equation is the one impacted, whereas the static term is not. Despite the absence of influence on static power consumption, the technique is widely used as typical ASICs have extremely large and complex clock trees for which pruning yields valuable results. CG is typically implemented at RTL or gate level, but the general principle operates on the behavioral level and the system level.

Power Gating (PG): is a technique based on switching off the voltage supply from any given component whenever there is no need for said component to be in operation. Typically, this technique is based on examining the active/inactive cycles of components so as to power off those in their inactive cycles. The fact that voltage supply is cut off from any given component (that is, it is made 0), essentially means that the impact on the static term constitutes the salient point of the technique. PG, like Clock Gating, is implemented at RTL or gate level, but its roots are at behavioral and system level. Unlike Clock Gating, PG greatly affects the static power term of the total power equation.

Dynamic (Voltage) Frequency Scaling (DFVS): is a technique by which any given component of the IC (typically a design element at the RTL, such as a buffer, a register, an adder, etc.) is run at a certain voltage and frequency value pair that maximizes its output or general performance while remaining under the power or energy budget. In general, low voltage-frequency value pairs reduce power consumption at the expense of performance degradation. However, given that performance requirements are neither static throughout the operation of the IC, nor homogeneous for every component, DVFS is a feasible attempt to reach a desirable tradeoff, typically being a technique in widespread use at the RTL, affecting both the dynamic and static power terms of the total power equation.

While the aforementioned techniques have a direct influence on physical variables (voltage, frequency), they also have an indirect effect on them via the architectural impact on ICs produced by the power management logic required for their application. In order to apply DVFS, CG or PG in an IC, the latter has to contain proper power management logic, which leads to associated overhead costs (such as the extra power dissipated by the required by the new logic or the increased area required by the physical synthesis of said new logic). The associated overhead costs can be thought of as one of the most evident and inevitable tradeoffs in the selection of a PMS.

2.4 Responses

It is a fairly common occurrence for the PMS to be carried out by a *Power Management Unit* (PMU). An illustrative PMU is made of a component with control over the power management logic. In ASICs, the PMU is frequently a very simple structure which drives signals controlling the execution of the chosen power management techniques as they are applied to control all other components. Frequently containing a type of Finite State Machine, a PMU has the PMS encoded in its own structure. By being the vessel than ensures the implementation of the power management techniques, the joint associated costs of these techniques can be summarized as the overhead costs of the PMU itself.

2.4.1 Power Management Unit (PMU)

The PMU itself (being a virtual aggregation of the power management logic of an IC) leads to an increased power consumption, even when one of the typical main goals behind the existence of such a unit is to reduce the overall power consumption of the IC. The overhead costs of the PMU are therefore of great importance following the corollary behind the statement "spend power to reduce power". Furthermore, the architectural impact of the PMS, which is represented by the structure of the PMU, acquires relevancy as it can lead to increased difficulty in validation and verification tasks both for the PMU and for the overall IC design.

Given the need for a Power Management Strategy for any properly designed ASIC, the associated costs of realizing that strategy (the implications of power-aware design) need to be handled carefully within the PMU. The PMU has to be designed in a power-aware manner, following successful DSE tasks that: effectively deal with the collateral effects of the PMU on any given ASIC (such as extra power consumption and endogenous verification effort already mentioned), minimize the expected extra management logic (reducing the area required for physical synthesis) and ensure that any concomitant power related decision is made in harmony with any non power related concern.

Decisions on the validity and proper implementation of the PMS in the PMU are dictated by how satisfactorily said strategy addresses power related requirements (for instance, the meeting of a power budget). In order for the decisions to lead to proper responses that fulfill a power related requirement, the tasks that provide the basis for the making of those decisions must be not only technically fitting, but also comprehensive. The rationale behind technically fitting, comprehensive power-aware DSE tasks leading to sensible decisions is in the global, architectural impact of the latter.

As a PMU represents the embodiment of the PMS, the architectural impact can be represented via the influence of the latter on two important notions: the functional block and the area overhead. These two notions showcase the variables that the PMS both affects and becomes affected by and underscore the comprehensive nature of the power-aware DSE tasks required to successfully implement the PMU. As such, the notions are elaborated as follows:

- A *functional block* is the label for the constitutive elements of components, a label that can be applied in different stages in a design flow. Being so widely applicable as a concept means that a functional block can be: a collection of gates and other similar elements working together in combinational or sequential logic (at the RTL), a set of algorithm-implementing objects representing an assortment of units at RTL (when considering the concept at the ESL, where algorithms are implemented) and even a series of self contained system level components seemingly working as blackboxes (at the specification based system level).

In a Gajski-Kuhn Y diagram [34] such as the one depicted in Fig. 2.4, the concept of functional block is reflected by the bisecting domain labeled "Structural". Along the arrow (from the center to the most external node) that states the elements of the structural domain, every element is matched to an abstraction level (Transistor Level, Gate Level, RTL, Algorithmic Level, System Level). The same type of matching is available for every element in both the arrows for Behavioral Domain and Physical Domain The Y diagram is a graphical way to understand the importance of the concept of functional block not only within the Structural Domain, but also within the Behavioral and Physical Domain.

The elements of the Structural Domain correlate with elements of the Behavioral and Physical Domain for each level. Such relations only underscore the relevance of the functional block for a comprehensive analysis of the design process. Typically, the highest level of abstraction in the design ladder (the System Level) ties in the functional block concept to the specifications of the design, as well as relating it to the global physical partitioning of the chip that the specification may require.

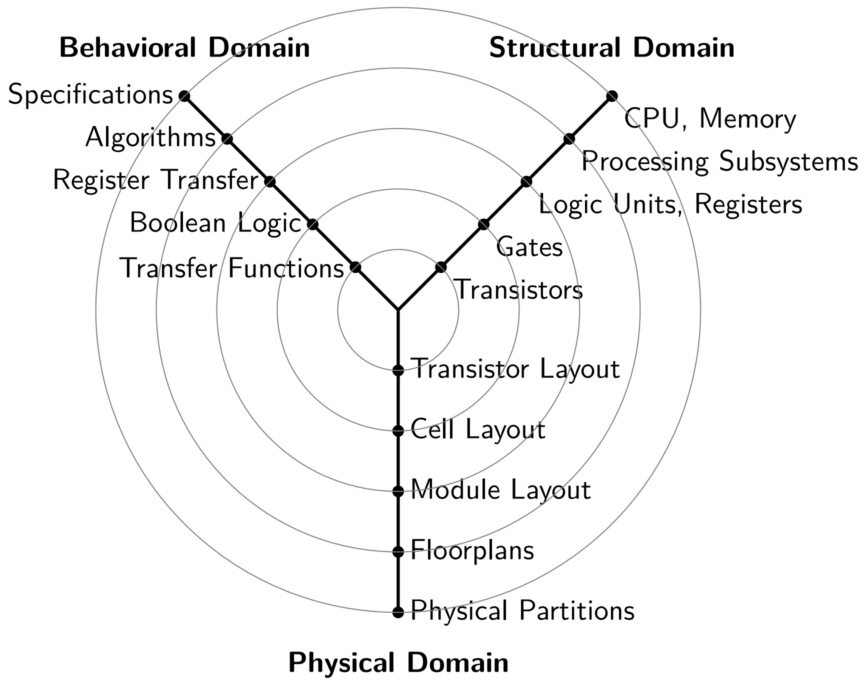


Fig. 2.4: Gajski-Kuhn Y Diagram

For ASICs or ASSPs, the specifications typically describe the functionality required of the circuit through listing the algorithms that define it, through naming the functional blocks that perform the functional tasks required of the design and also through the placement of relevant system wide constraints (eg. a power/energy budget, a security or safety check). In this document, the system wide constraints that become the focal point are those related to power management. However, a power budget is not the only system wide constraint that relates to power management.

There is a systemic influence of the power management as a concern in any design process. Such systemic influence means that the implementation of the power management strategy for the IC should be taken so as to comply with a power budget, security/safety checks, while also properly dealing with the inevitable extra circuit logic required. The additional logic is typically an undesired byproduct of the implementation of the PMS, one that should be minimized as much as possible.

When considering the impact of the extra circuit logic, a power-aware design flow usually leads to a trade-off. The more sophisticated (and potentially more power-aware) the PMS, the bigger the extra circuit logic required. This rule of thumb is not meant to be taken as a strict correlation, but nonetheless serves to illustrate the need for a balance between two potential factors: power efficiency and extra management logic. So as for a better understanding of the impact of the extra power management logic, such an impact is described by a second notion: area overhead.

- The *area overhead* of the implementation of a PMS is a concept that is representative of the amount of transistors, gates and other functional blocks that said implementation requires. This amount is, naturally, in addition to whatever was required for the implementation of the functionality of any intended design, which is consistent with the use of the word "overhead". Given the fact that area is a constraint in the synthesis of current (and foreseeable) ASICs, any overhead in this metric is to be carefully dealt with.

While the concept of area overhead is closely linked to lower levels in the abstraction ladder, the effects can be observed beyond the Gate Level or the RTL. Not unlike the notion of functional block, the area overhead resulting from extra power management logic can be considered throughout the levels of the Gajski-Kuhn Y diagram. Once again, the prototypical functional block for each level can be considered as the constituent element of the implementation of the power management logic.

Among the most important decisions that underpin the architectural impact of the power-aware DSE tasks that influence the notions above are: a) **Power Domain** (PD) partitioning and b) the setting of **Control Signals** (CS) and **Power Modes** (PM). The preceding decisions are naturally based on the power structure that is revealed by inspecting the specification, which yields PD, CS and PM. The frameworks that allow for this inspection to reveal the underlying power structure are the contributions of this thesis, laid out in the theoretical chapters of Part I and Part II.

So as to clarify what is meant by PD, CS and PM, the following are non exhaustive, but clear definitions:

2.4.1.1 Power Domain

A **Power Domain** is a collection of functional blocks that share the same voltage source and can be considered a single group whose power can be managed independently from other parts of the design. For example, functional blocks that operate whenever a device is powered are typically part of an Always-ON Power Domain, whereas other functional blocks that are activity dependent are typical part of other Power Domains.

2.4.1.2 Control Signal

A **Control Signal** is the way in which a Power Domain is driven ON/OFF (or into a different power state) as per dictated by the PMS. For many designs it is reasonable to assume that each Power Domain will have at least one Control Signal to drive it, but this Control Signal could potentially have different values leading to independent and concurrent control of several of those Power Domains.

2.4.1.3 Power Mode

A **Power Mode** is a given set of the power states (ON, OFF, other) of the Power Domains during the operation of a circuit. It is therefore, a way to identify the system wide operational status of a design. Examples of Power Modes are: ACTIVE, ON, IDLE, SLEEP, OFF, etc.

2.4.1.4 Power Domains, Control Signals and Power Modes and the PMU

The revelation of the implicit power structure is the basis for the handling of the PD, CS and PM that are associated with said power structure. Decisions relating to how the PMU is to properly address the management of power rely on the number of PD, CS and PM. Consequently, in the following paragraphs the reader is very succinctly introduced to the basics of both PD partitioning and the setting of CS and PM.

2.4.1.4.1 Power Domain Partitioning Power Domain (PD) partitioning is a process that deals with the organizational structure supporting the PMS. In essence, Power Domain Partitioning is the process by which such structure is decided upon. As a power-aware DSE task, it is typically the first critical decision making process. As such, Power Domain Partitioning is a key step in setting the basic architecture of the PMU.

The essence of Power Domain partitioning is, as would be expected, the concept of a Power Domain (PD). How to arrange the functional blocks in different PDs is the core of the Power Domain partitioning decision. As illustrated by Fig. 2.5, the process chiefly sorts the different functional blocks of a design into groups (the Power Domains). The power properties of these groups (such as, for instance power states-*on*,*off*,*standby*) can then be managed independently from each other. As evidenced in the figure, three functional blocks (Block C, Block D and Block E) can be part of the same power domain (Power Domain 3) if they share the same Power State Table (PST). However, this may not be desired if the designer wants to manage each of the blocks individually. A decision on the convenience of such an arrangement (the *Power Domain Partitioning scheme*) is the designers' responsibility.

Typically a fine grained Power Domain partitioning scheme leads to many PD (the maximum would be given by the number of functional blocks, with one PD per functional block) and an unavoidable area overhead. A more coarse grained approach to Power Domain Partitioning usually leads to a scheme with reduced area overhead and fewer PD, but at the cost of missing power consumption reduction opportunities. The opportunities for power consumption reduction are missed on account of the impossibility to control the power states of each functional block individually, since said blocks will be part of the same PD.

To group certain functional blocks together has a deep impact not only on power consumption, but also on further verification efforts, both for the overall design, and for the PMU executing the associated power management strategy, as shown by Agarwal et al. [36] and Wang et al. [37]. The deep impact is also felt in the overall

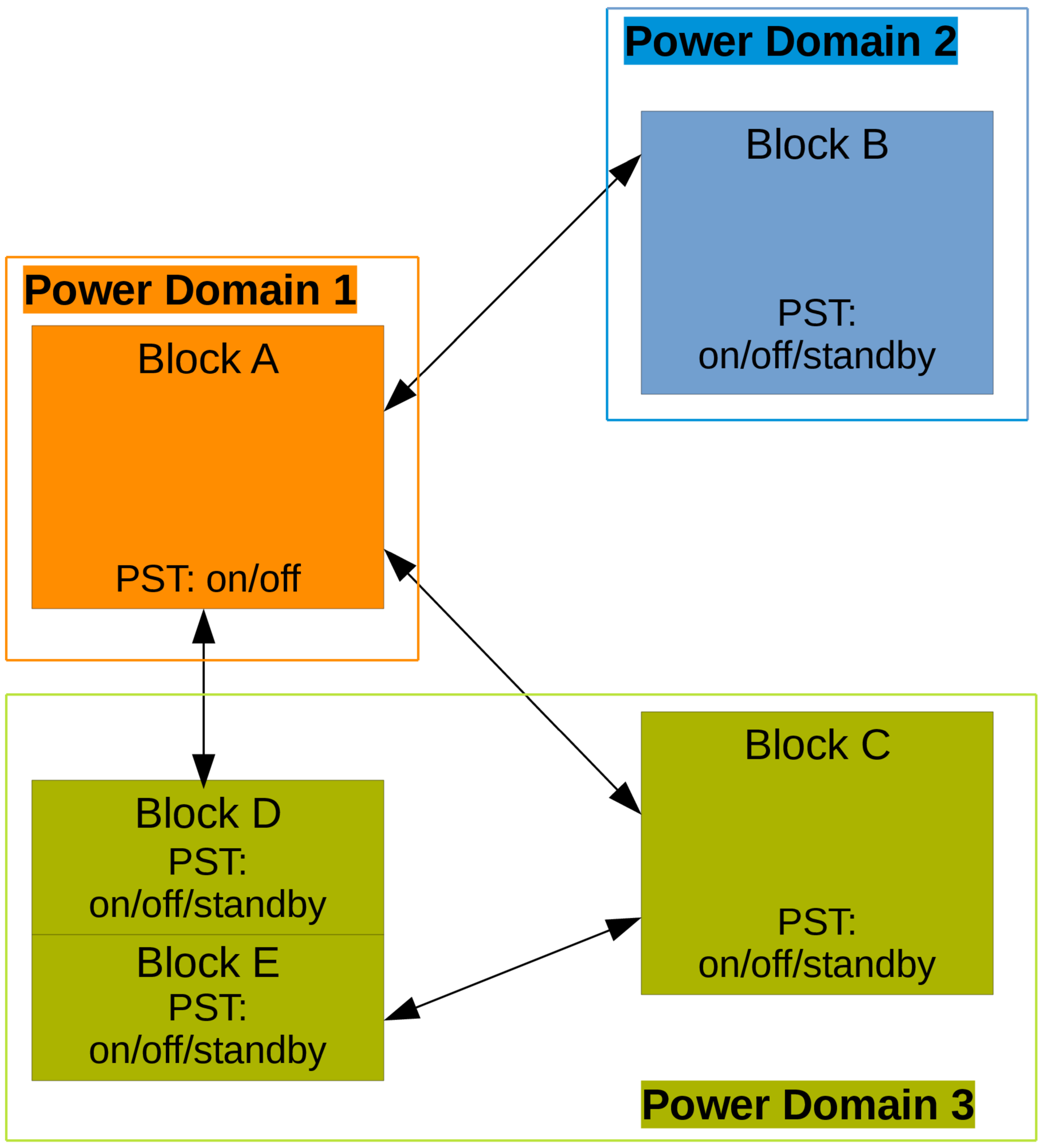


Fig. 2.5: Example of a Power Domain Partitioning Scheme from [35]

verification efforts for the design, which once again highlights the tremendous importance of deciding on a proper Power Domain partitioning scheme for power-aware design at system level.

2.4.1.4.2 Setting of Control Signals and Power Modes The Setting of Control Signals (CS) and Power Modes (PM) is a process that details the architecture of the PMU. Usually done after the PD partitioning process (it can also be done concurrently with the PD partitioning process), the setting of Control Signals and Power Modes is the cornerstone of the PMU, a process whose objective is to establish the way of operation of said PMU. In essence, CS and PM are numbers defining the behavior of the PMU, which, in itself, represents the way the PMS is put forward.

For the PMS to be properly deployed, the PMU additionally needs to have the system Power Modes (such as, for instance, IDLE, ACTIVE, INACTIVE modes) and Control Signals (which drive the power domains) set. Both PM and CS are widely influential PMU design parameters. Regardless of how the decision about the determination of the aforementioned parameters is reached, PM and CS will reflect the complexity of the PMS, which it itself signals the complexity of the PMU.

Just as is the case of a power domain partitioning scheme, the number of Power Modes and Control Signals are important because they are usual indicators of the power intent (that is to say, the power management strategy). The impact of both PM and CS on the overall costs (mainly amounting to verification time, power consumption and area overhead) of a PMU is as important as that of the Power Domain Partitioning process. This impact means that the PM and CS need to be determined as early as possible in the design flow [38].

A suitable set of assumptions are necessary to keep the approach to determining (and thus setting) both PM and CS as simple as possible. For instance, a simplified version of the process can have as premises: a single CS can power ON/OFF a Power Domain (no complex individual Power States, such as IDLE or SLEEP); and a Power Mode represents a system wide state (one where the state of each Power Domain is ascertained), which can be ACTIVE, IDLE, SLEEP or similar.

As a graphical example, please consider Fig. 2.6 and Fig. 2.7, which show a Power Domain Partitioning scheme and a table summarizing the Power Modes and Power States for a device. These figures are taken from the description of an IoT device which has a RF transceiver and an ARC EM processor, suitable for a well known 3GPP standard [39].

The figures show 6 PD (AON, PD1, PD2, PD3, PD4, PD5, PD6) in the bottom rows, 3 PM (ACTIVE, SLEEP and STANDBY) on the columns, and the Power State, controlled by a CS, for each of the PD (ON, OFF, "rentention"). This a typical Power Management scheme, consisting of enough PD, CS and PM, to presumably guarantee a low power consumption for a battery powered device without needlessly increasing the complexity of the PMU (which is on the AON-Always ON domain).

Research has been conducted for the complexity costs, concluding in a rule that basically leads to two statements:

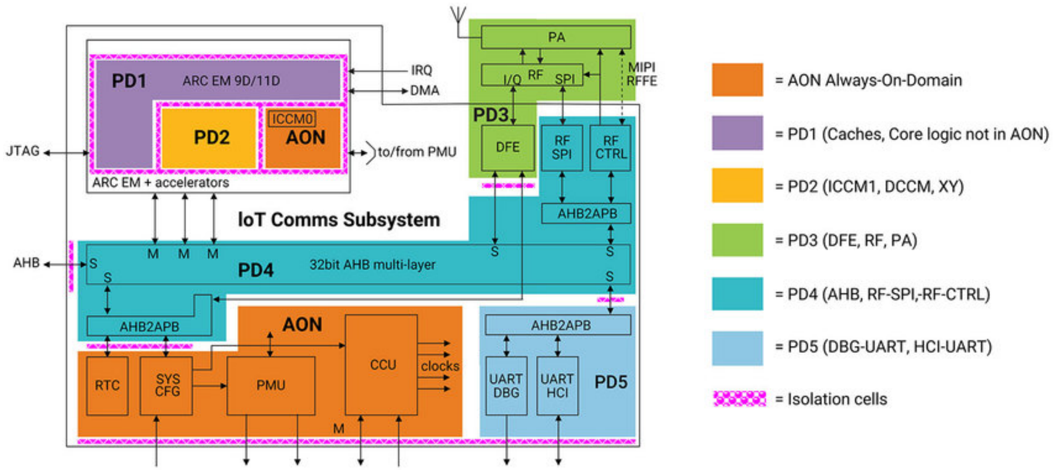
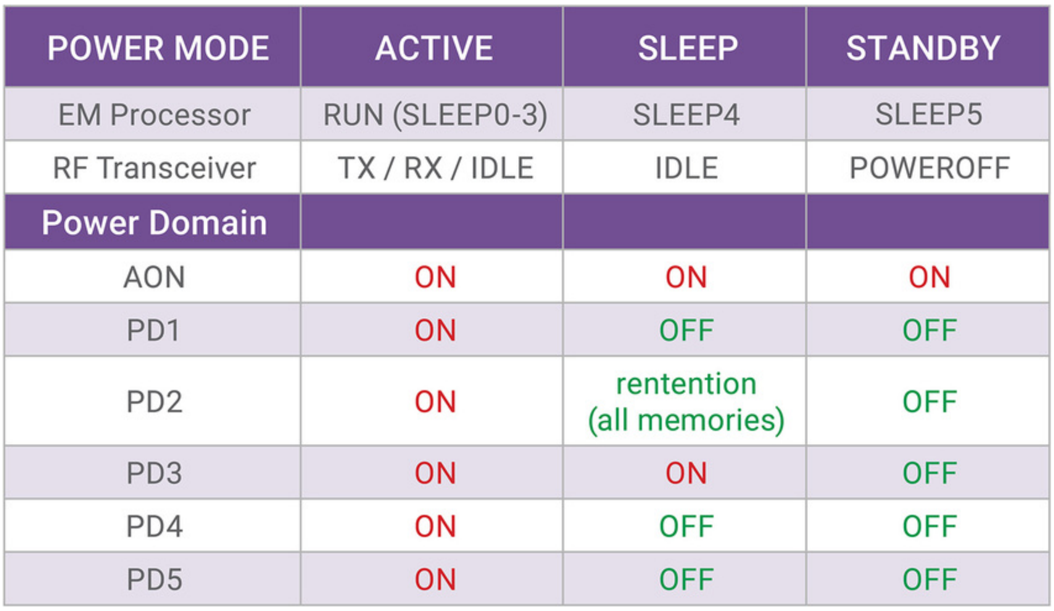


Fig. 2.6: An IoT Device's Power Domain Partitioning Scheme from [39]



POWER MODE	ACTIVE	SLEEP	STANDBY
EM Processor	RUN (SLEEP0-3)	SLEEP4	SLEEP5
RF Transceiver	TX / RX / IDLE	IDLE	POWEROFF
Power Domain			
AON	ON	ON	ON
PD1	ON	OFF	OFF
PD2	ON	rentention (all memories)	OFF
PD3	ON	ON	OFF
PD4	ON	OFF	OFF
PD5	ON	OFF	OFF

Fig. 2.7: An IoT Device's Power Modes and Power States from [39]

- *The number of PM and CS can be used to predict the power consumption of the PMU (with the CS having a greater impact than the PM, due to more CS requiring more area in silicon which will dissipate more energy)*

- *The number of PM and CS correlate greatly with the time needed to be spent on the verification of the PMU (with the PM having a great impact than the CS, because of the implications of PM in the number of transitions to validate)*

Because of those two statements, the designers need to be careful about choosing the most appropriate Power Partitioning scheme, as well as with choosing the amount of Control Signals and Power Modes that the PMU will implement. The fact that the PMU itself needs to be considered another functional block in an Always ON domain also highlights that the decisions need to be taken before the RTL, that is, before the microarchitecture of the system has been established.

The most natural way to decide the setting of PD, CS and PM is, therefore, to focus on a higher level of abstraction: the system level. In said level, the most typical design documents available are: a *natural language specification* (typically in English) or a *technical language specification* (typically in SystemC/TLM).

As stated in the Introduction chapter to this document, *this thesis presents two frameworks that allow for the power structure that is intrinsic to a design to become available to those responsible for implementing the PMS within the PMU.* The PD partitioning scheme and the setting of the CS and PM depend on the availability of the implicit power structure for the cases of a design following both a natural language specification (which is dealt with in Part I) or a technical language specification (dealt with in Part II).

Part I: Natural Language Specifications

3 Natural Language: Preliminaries, Relevant Work and Response Framework

3.1 Preliminaries

It is fairly common for designers to have specifications as their initial design documents when they are following a top-down design pattern. It is also common that, even before a specification is written, there are algorithms, flow or block diagrams that make intuitive sense to designers. Unfortunately, the very essential informal design documents that constitute the basic design idea are rather ambiguous and usually come as a response to a set of requirements that have to be met. When the way in which these early stage documents concerns themselves mainly proper elicitation of the requirements, the documents are best approached by requirement engineering [40].

If a specification is available, it is frequently one written in a natural language, such as English, either in an informal descriptive tone or a more formal tone, such as the case of a (technical) standard, very common for ASICs. In the latter case, it is also quite likely that there exists a reference implementation described within the standard or in a closely associated, yet separate document. Standard specifications (usually shortened to standards) are often described as documents expressing a series of characteristics provided by an implementation. For digital circuits and related Systems-on-Chips (SoCs) these characteristics pertain mostly (but not exclusively) to functionality and are typically covered in one or several chapters of the appropriate standards. The designers are expected to comply with and follow the standards, so as to produce validated implementations.

Traditionally, standards are long dense documents with tables, diagrams and text detailing the underlying several aspects, attributes and features of blocks/components/elements of a system. Because of the sheer volume of information contained in a standard, the examination of one is almost invariably done by experienced designers. This process includes the laborious elicitation of relevant data and knowledge from the contents of the standard, as well as any needed interpretation.

Given the opportunity, streamlining the laborious process of data and knowledge elicitation appears to be a natural step forward. To decrease the effort required for the manual elicitation of data and knowledge, Information Extraction (IE) is typically used [41]. As a well known way to push for the automation of the analysis of standards IE consists of a series of techniques to bring out relevant data. As a general purpose toolset, IE features two common approaches are: i) domain specific *Information Extraction* or ii) *Open Information Extraction* (OIE). As the name of the former suggests, the approach based on it, focuses on retrieving relevant information (facts) only on a very specific field (domain). This is usually aided by a robust concept corpus (typically, an ontology) and by the use of other additional knowledge sources [42] [43]. Contrary to this, OIE aims for general all purpose information extraction or retrieval from texts of varied topics, styles and sources. For OIE, there is a tradeoff between *precision* (retrieving all the useful facts) and *recall* (retrieving only the useful facts). For a succinct explanation of the concepts, please take into account that: i) precision may be defined as the number of true positives (the useful facts), divided by the number of selected elements (the total number of facts retrieved), ii) recall may be defined as the true positives (the useful facts) divided by the number of relevant elements (the total number of relevant facts). For both precision and recall the number defining them (the score) is between 0 and 1. Both these values are usually provided as a means of assessing the quality of the results provided by the OIE tool, but do not ensure that the results are of any use for further understanding or reasoning.

A useful fact is one germane to the understanding or description of a text based on its domain. For instance, a fact such as "*The circuit is internally connected through wires*" is typically too evident to constitute a useful piece of information, whereas "*The circuit has a output serial port*" is a fact that provides significant information that can be used to understand the text. Analogously, relevant facts are considered facts that are not tautologies (redundancies) or grammar based constructs devoid of significance in the context of the domain. For instance, a fact such as "*The digital circuit consists of digital components*" is not a relevant fact, since it is redundant, whereas "*The digital circuit consists of an encoder and a UART*" is a relevant fact since the added information leads to further understanding.

If OIE is used for domain specific information extraction, the desired recall is usually of higher importance than the desired precision. This is so because a high number of relevant but not useful facts can be discarded if needed further down the processing tasks, whereas non relevant facts can be harder to discard if they are firstly grouped together with relevant ones. However, a high number of both precision and recall (such as for instance, a high F1 score) are generally desirable, even if rather difficult to attain.

Approaches using OIE tools are more frequently researched than those using the domain specific counterparts. Two key reasons behind this fact are: a) the desire to make the approach as universal as possible and, b) the lack of an ontology or a similar domain specific knowledge source. Notwithstanding the preference for OIE tools, whatever approach to the analysis of standards can also work with domain dependent IE tools. If a domain dependent IE tool is available and is wisely used, it will most likely lead to more refined scrutiny than the OIE alternative.

OIE is routinely characterized as a task, in which the techniques used are grouped into diverse subtasks (Entity Extraction, Relationship Extraction, Word Sense Disambiguation, Coreference Resolution, Terminology Extraction and many more than tightly relate to *Natural Language Processing* (NLP)). Without entering into too much detail, two of the tasks listed above are extremely central to analyzing a text with semantic understanding as a goal: *Entity Extraction* and *Relationship Extraction*. For a short explanation it can be said that the former retrieves relevant (for whatever domain) entities (traditionally considered concepts-self standing notions), whereas the latter retrieves the association that binds the entities.

For a clearer explanation of both subtasks, please consider the following sentences from the README file (an informal specification in natural language) of an example FIR filter, a very basic design that comes with the official SystemC distribution suite from Accellera [44].

"The filter is a 16 tap FIR filter(fir.cc). The test bench feeds simply ascending values into the FIR(stimulus.cc) and the output is sampled (display.cc) and displayed with print statements."

Entity Extraction will yield a series of entities that can be pruned later, either by humans or by some automated procedure. A list of entities may be similar to that in Table 3.1. Here, anyone familiar with the field of digital circuit design will notice that *fir* and *fir.cc* refer to the some entity, not unlike the case for *output* and *display.cc* and *value* and *stimulus.cc* (which refer to the input of the circuit). Because of these equivalences, refined Entity Extraction leads to semantic entities, that is to say, entities which are not mere language artifacts, but meaningful concepts within the domain.

Tab. 3.1: Entity Extraction

display.cc
fir.cc
fir
output
print
statement
stimulus.cc
testbench
value

Tab. 3.2: Relationship Extraction

Semantic Entity 1	Type of relationship	Semantic Entity 2
print	associatedWith	statement
Fir	associatedWith	fir.cc
fir	associatedWith	stimulus.cc
Fir	hasDeterminer	the
statement	hasQuantifier	multiple
value	hasQuantifier	multiple
Fir	hasDataValue	16
fir	hasDeterminer	the
output	hasDataValue	the
Fir	hasQuality	Tap
tesbench	hasDeterminer	the

Relationship Extraction will reveal the associations between the semantic entities. This means, how the semantic entities relate to each other. A common format for this may be seen in Table 3.2. The most typical relationship can be seen as a simple association *associatedWith*, but there is also a numerical association *hasQuantifier, hasDataValue* and even more syntactic type of link *hasQuality, hasDeterminer*. This presentation format is also known as **triples** and constitutes the most typical presentation format for information facts output by OIE tools.

At this point it can be said that Entity Extraction is a subtask of OIE that is linked very closely to elucidating the functional blocks of a digital circuit design. This close link between Entity Extraction (essentially its semantic concepts) and obtaining the functional blocks of a circuit thus links the former to the setting of PD, as it has been previously explained how functional blocks are the constitutive elements of a Power Domain.

Similarly to the case of Entity Extraction and the setting of Power Domains, Relationship Extraction is associated with the setting of Control Signals and Power Modes. The reason behind this association lies in the fact that Relationship Extraction links semantic entities and these represent the functional blocks. Since functional blocks are the forerunners of the Power Domain, the links between functional blocks (which are characterized by the semantic entities) can be thought of as crucial information to properly notice the Control Signals and Power Modes for the digital circuit.

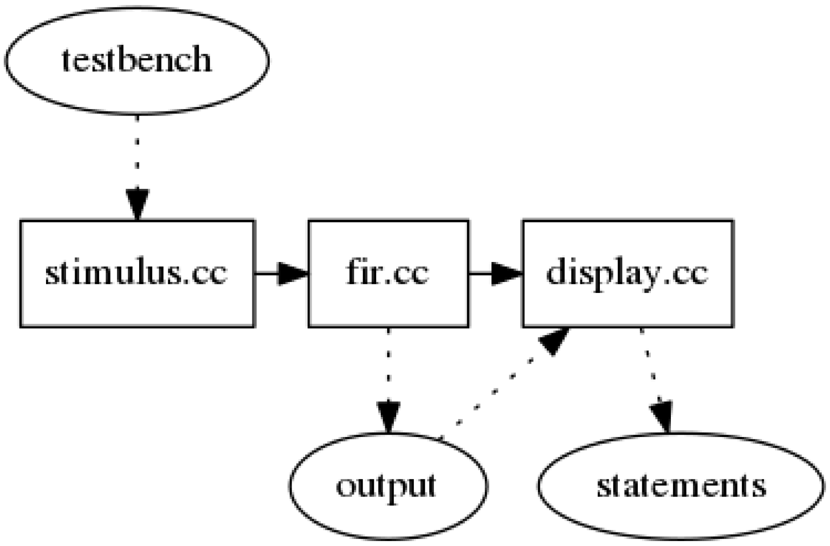


Fig. 3.1: Flow/Block diagram for the FIR Filter

Tab. 3.3: Condensed Tabular Format for the FIR Filter

Entity 1	**Type of relationship**	**Entity 2**
fir.cc	associatedWith	Fir
stimulus.cc	associatedWith	fir
display	comesFrom	output
display	isRelatedTo	statement

As a way to notice the power of Entity Extraction and Relationship Extraction to unveil the architecture of a design, a simplified flow/block diagram the FIR filter described above is depicted in Fig. 3.1. In it, the 3 functional blocks come from the extracted semantic entities: *stimulus.cc*, *fir.cc* and *display.cc*, which are depicted in rectangles, while the the other entities (*testbench*, *outputs* and *statements*) are depicted in ellipses and represents actions and information sources/outputs of the functional blocks. Fig. 3.1 follows the condensed tabular format summarizing both Entity Extraction and Relationship Extraction for the semantic entities, as depicted in Table 3.3.

In this example of Entity Extraction and Relationship Extraction, the values for *precision* and *recall* can be calculated by examining both Table 3.1 and Table 3.2 in light of the information condensed in Table 3.3. It is then possible to say that there are 11 extracted facts (the triples), of which 5 are semantically relevant and 9 self standing entities, of which 6 are relevant. We know the above thanks to the explanation of the diagram on Fig. 3.1. Out of the 6 relevant entities and the 5 semantically relevant relationships come 4 useful facts depicted in Table 3.3. As such the *precision* considered the average of 0.36 (4 out of 11) and 0.44 (4 out of

9), whereas the *recall* can be considered the average of 0.60 (4 out of 6) and 0.8 (4 out of 5). It is therefore noticeable that the recall is higher than the precision, which falls under the preferences stated above for OIE tools applied to domain specific scenarios.

3.1.1 Power-aware Interpretation

Once the semantic entities, as well as the relationship between them are known, not only a flow/block diagram can be built, but it can be interpreted in the context of power-aware design. For instance, following Fig. 3.1, a designer can confidently conclude the following:

- The FIR filter consists of 3 functional blocks that are prime candidates for **3 Power Domains**: *stimulus.cc*, *fir.cc* and *display.cc*.
- The aforementioned 3 functional blocks should be controlled independently of each other, considering the sequential nature of the information flow, leading to **3 Control Signals**, one per each Power Domain.
- The FIR filter consists of at least three stages: the *stimulus stage* (where the input is fed to the filter) the *filtering stage* (where the filtering takes place) and the *output stage* (where the results are shown in the output). These stages happen concurrently, in spite of the sequential nature of the information flow, leading to at least **4 Power Modes**: one in which the FIR filter is in full operation, one in which it is not operative at all, and two others in which either *stimulus.cc* or *display.cc* are inactive due to lack of input or lack of processed information to show, respectively.

The above interpretations are a result of the expertise possessed by a human designer. Later in this chapter, the proposed response framework showcases how similar interpretations can be achieved by the codification of the some of the expertise and the reasoning into a set of rules.

Based on how the power-aware interpretation above uses the information provided by OIE, it is reasonable to consider OIE as a more abstract way to gather information from a natural language specification for a variety of uses. This is why the next section looks at OIE and related NLP techniques in the broader context of specification analysis for both power-aware design and other related design tasks at system levels.

3.2 Relevant Work

The power of IE (both OIE and domain dependent) is why it is at the core of most automated (or semi-automated) attempts to address the issues of natural language specification analysis, especially those related to extracting and then addressing relevant functionality requirements or concerns. However, it has been shown that IE techniques can also be valid approaches to dealing with other matters within the analysis of standards. Some research has shown that IE techniques are appropriate, for instance, as aids in the verification process of digital circuit systems according to a standard, as shown by Harris et al. [45]. Moreover, some other works have showcased the usage of IE to deal with other concerns within broader and more generalized examinations of specifications, such as the work of Singh et al. [46] and Shankar et al. [47].

For the most part, the work on IE for specification analysis (of which the works cited above are shining examples) has a main goal: *to unveil the structure that the specification imposes on any design complying with it*. Whether the focus is on: how to use the unveiled information to verify, through assertions that a digital circuit design is compliant with a specification (as is the case of Harris), or on how to ultimately create a Knowledge Base from the information unveiled (the case of Singh and Shankar), the process is mostly concerned with the extraction of relevant facts with high precision, even at the cost of some recall.

The closest (so far) that research has got to the usage of IE techniques for power-aware DSE and for the extraction of the power structure innate to a specification seems to have been when the bases of the work of Singh were extended to deal with system level power estimation [48]. In said research work, the main concern was to estimate power from a given description, with no attempt to use the information extraction to uncover the innate power structure defined by the specification under analysis.

Regardless of how the analysis of the specification is performed, it is a commonplace scenario for designers in charge of system level power-aware design to conduct DSE after having scrutinized the natural language specification. Potential alternative design structures conforming to the specifications are then the result of the creative ingenuity of the designers. These alternative design structures are ordinarily considered early stage solutions in need of further evaluation. Such further evaluation of the alternatives is a task of DSE processes, which also allow designers to analyze latent optimizations for any selected alternative and even to ”discover” new variations of said alternative that may be suitable for varying scenarios. The byproduct of this evaluation task (the generated knowledge) is an extremely valuable asset that typically remains within the individual or group that produced it, generally in a visual representation (such as UML), as is described by Liehr et al. [49].

The holistic and expertise dependent nature of natural language specification analysis has led many to turn to ways to address the unsystematic and ambiguous nature of said specifications. One explored avenue has been the *Expert System* (ES). ES represent an attempt to mimic the way knowledgeable humans (in this case, designers with expertise) parse the specification and make sense (by reasoning) of the information contained therein. Robert Steele, in the late 1980s posed one of the first attempts at harnessing the power of an ES to help with digital circuit design [50]. His rationale was that designers' expertise typically shows itself in deep unsystematic knowledge whose storage, dissemination and further reuse were rather difficult for a *Computer Aided Design* (CAD) tool to apply on its own. For that to happen, the CAD tool needs to have understanding and command of information sources such as Knowledge Bases and Ontologies, as well as of concepts such as inference.

Unfortunately, while the previous concepts have been (and remain) subjects of research, the extent by which they are used by humans is still not matched by semi-automated machine based methods. The research line "inaugurated" by Steele was later explored (for example in [51]), but it never dealt with system-level design concerns, because at the time (early 1990s), the system levels were not a focus of research.

The use of information sources like the ones mentioned above within a semi-automated system (in many cases, *Intelligent Decision Support Systems*-IDSS) in the realm of EDA has been limited due their unavailability, general incompleteness and inadequate upkeep. As is the usual case with documentation of technical nature, the curating tasks associated to it are sadly relegated (whether consciously or unconsciously) in the list of priorities. Since properly curated information is the backbone of insight and expertise, *Knowledge Based Systems* (KBS) that try to mimic the reasoning, inference and matching abilities of humans are often limited by their inadequacies or limitations in particular knowledge domains.

Notwithstanding the apparently bleak outlook of ES for the IC design field, ES remain a very straightforward choice connected to IE. ES are the most natural choice for a KBS to yield useful and reproducible results in specification analysis done as part of the *Design Space Exploration* (DSE) process, as ES attempt to condense the available expertise into machine based reasoning, inference and decision making. Outside the realm of IC design, ES are reasonably common within diverse fields such as accounting [52], medicine [53] and engineering [54].

Other early attempts at introducing ES to the realm of IC design, such as the work of Subrahmanyam [55] and Wu et al. [56] well were received, but the interest in continuing the research ideas dwindled over the 1990s. There are several reasons for this decreasing interest, but arguably the most evident is that implementations of ES have not been able to materialize the level of automation and universality once envisioned. However, this does not detract from the powerful nature of ES based solutions for specification analysis and their potential application to power-aware design at the system level.

In an endeavor to prove the validity of the ES approach for specification analysis for power-aware design at the system level, in the next section, an ES based framework is presented as a response. The overall goal is to be able to unveil the power structure innate to the specification, thus having a baseline from which to aid in setting the Power Domains as well as in setting the Control Signals and Power Modes for the Power Management Unit to implement.

3.3 Response Framework

3.3.1 Architectural Overview

The response framework is ES based. The ES takes its input from IE tools applied to natural language specifications, and produces outputs that yield valid Power Domain Partitioning schemes, as well as validated sets of PM and CS. The system is a rule based tool that analyzes the text of a specification and seeks to automatically process the information obtained via the analysis in a way that resembles that of an expert designer. The reasoning of a seasoned designer (which is based on the expertise accumulated) is the way valid PD partitioning schemes and validated sets of PM and CS are obtained. *The main contribution of this framework is, then, the way the rules are encoded to mimic the type of analytical process with which an expert designer would approach the specification.*

The general architecture of the framework (depicted in Fig. 3.2) is summarily described through a block diagram. In such a block diagram, the main assumption of the system is clear: the Selected Input (sentences) are extracted from General Input (Specification) to be later fed to the Information Extraction process. This extraction task is not part of the ES (which is enclosed by a dash lined rectangle box in the figure) and is done by an OIE tool following the principles previously outlined. While such a thing seems to be in direct opposition to the intention of automated

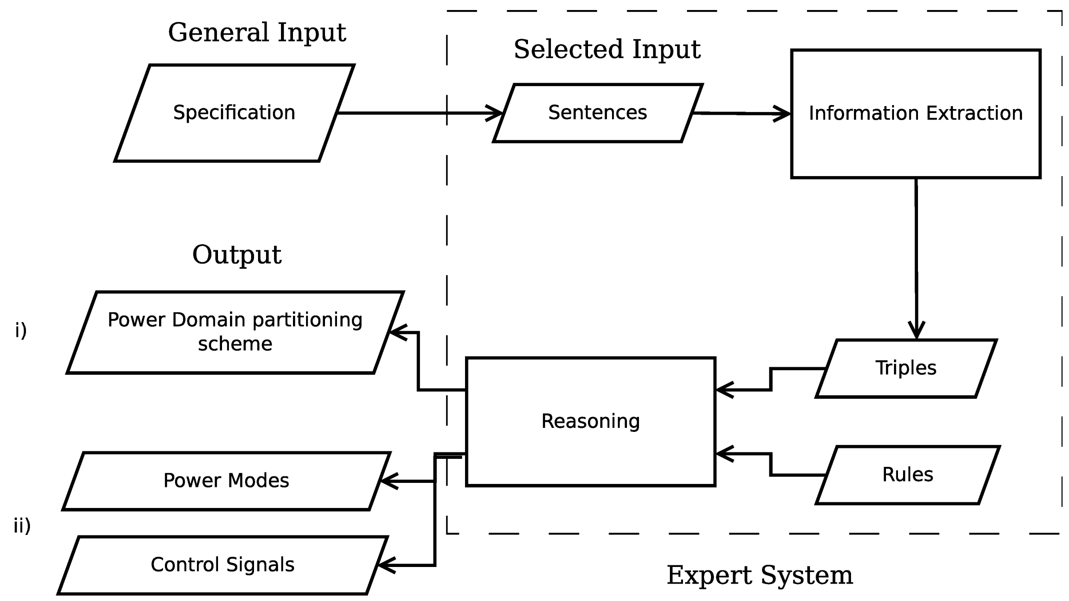


Fig. 3.2: Architecture of the Expert System

analysis for a specification, both the sheer volume of text in a typical standard and the lack of extremely advanced topic modeling tools, make this assumption quite reasonable. Moreover, the ES (and the response framework as a whole) is not intended to replace the designer, not even in its supervisory role, but more so to aid in the design process.

The set of rules that are the input needed for the Reasoning process are editable and may require adaptation for generalized use. This should not be looked at as hindrance, since an ES is typically developed ad-hoc, with its logic being generalizable only to a certain extent within its intended field of use. Furthermore, the rules (which currently either fire or do not) may require fuzzification in the future, so as to reach the kind of subtle analysis performed by a well versed designer. While fuzzification would a welcome addition, it is still not fully necessary, because a set of clear rules can still produce validated results that constitute a baseline output.

The output, as clearly depicted in the figure, are the Power Domain Partitioning scheme, the Control Signals and the Power Modes. For the latter two, the number is the important value, since the nature of the PM or what type of CS may be required are not needed for the basic decision making process targeted by the response framework.

3.3.2 The Stages of the Response Framework

Let the case of an MBus device be considered as the running case. MBus is a mid 2010s development of a bus architecture aimed at low power consumption and at the application of power-aware principles [57]. The MBus authors provide a standard specification [58], as well as a reference implementation described in natural language [59]. By going through the execution flow for the ES, how the PD Partitioning Scheme as well as the setting of the CS and PM are obtained will become readily apparent.

So as to familiarize the reader to the kind of sentences functioning as Selected Input, some are listed below:

Selected sentences from the standard specification:

- For the purposes of this document, each MBus node has two modules: (i) The block that interfaces with the bus itself—we define this as the Bus Controller—, and (ii) the block that comprises the rest of the node—we define this as the Layer.
- With MBus, a completely power-gated node can seamlessly awaken its Bus Controller with no special assistance from the sending node or the mediator node.
- A Bus Controller can filter addresses, only waking the Layer for a message destined for that node.
- MBus edges can also be harvested to return both the Layer and the Bus Controller to sleep mode.

Notice that the sentences above are quite descriptive and relatively verbose. They have been extracted from the Chapter "Power Design" of the standard. Owing to their verbosity, which is typical in standards, these selected sentences are relatively challenging to parse for relationship extraction. The verbosity tends to require more parsing prowess from the OIE tools, because each sentence contains words (such as *seamlessly*, *special* or *harvested*) whose relationship to the entities is not as straightforwardly unveiled as is the case for *Part of Speech* (POS) tokens, such as verbs or adjectives in close proximity to a noun defining those entities.

The challenges posed by the proper selection of the sentences leads to incorporating into the list some other selected sentences from the reference implementation (which is also a specification for the purposes of the response framework). Among those further selected sentences are:

- The M3 MBus implementation defines two major components: a Bus Controller and a Layer Controller.
- The bus controller understands the MBus protocol and presents a simple word-wide interface to higher layers.
- The generic layer controller provides a register file and a memory interface, sufficient for most simple devices.
- In addition, the M3 MBus implementation requires some support blocks: a Sleep Controller, a Wire Controller, and an Interrupt Controller.
- If the layer controller is powered off, however, the bus controller must wake the layer controller.

Notice that these other selected sentences exhibit a simpler pattern. Not as verbose as those selected from the Standard, the words in these sentences are mostly nouns, verbs and adjectives (in terms of POS tokens), within a mostly *Subject-Verb-Object* (SVO) order that reduces the parsing prowess require to parse them.

It is also important to notice that the selection of the sentences is a process than could potentially incur in automation at some point in the future. However, said process of selection would require considerable use of machine learning techniques that are still too onerous (in terms of computational complexity), which constitutes the main reason why this step is not automated.

Once the Selected Input has been defined, the way the response framework is implemented responds to the following stages:

3.3.2.1 The Information Extraction Stage

The *Information Extraction* (IE) stage is where the processing of the sentences fed into the ES takes place. The main goal of this stage is to identify the functional blocks and their connection from the entities and relationships extracted from the Selected Input.

The IE task mainly performs the subtasks of *Entity Extraction* and *Relationship* Extraction, leading to a list of relevant entities and their relationships. The list is represented by triples of the form [Entity 1]-[Relationship]-[Entity 2]. The triples are then an input to the Reasoning process.

The IE process is automated, although its output (the triples) is not meant to remain unsupervised. The aptness of the triples for their inclusion as input to the Reasoning process should be ascertained by a seasoned designer. Human intervention is only required to adjust the depth and inner workings of the IE tasks before usage and not on a continuous basis. The adjustments are intended as a way to produce a better set of triples for further analysis.

In order to explain how this IE task works in further detail, it becomes necessary to introduce the reader to more specialized concepts in the realm of NLP and computational linguistics. These concepts and how they shape this stage are explained next.

3.3.2.1.1 Natural Language Processing and Universal Dependencies

As very few ontologies have been developed in for ASIC design [60] [61], non ontology based approaches are to be used. OIE approaches, owing to their universality need to be tailored to better meet the needs of those performing specification analysis. This tailoring can be done through a suitable use of a *Dependency Grammar* (DG) scheme, one that can better capture the semantics of the sentences under analysis. One such DG scheme is that of *Universal Dependencies* (UD) [62].

UD are a novel way in which linguistic entities in a sentence (typically Parts of Speech tokens) are related to one another based on the subordination of functional entities (verbs and adjectives, for instance) to content entities (nouns, for instance). This dependency style is better suited for the type of semantic analysis used in OIE than traditionally purely syntactic DGs [63]. UD has been recently used for OIE, showing promising results [64], specifically through the implementation of a Predicate-Argument extraction tool that is usable as part of a NLP approach to specification analysis.

The salient point of the UD paradigm is that POS tokens are labeled in a way that gives content words the role of central nodes in the dependency hierarchy, as opposed to giving said role to function words. In UD, content words are considered relevant standing notions (for example, POS tokens such as nouns) are considered the master nodes to which the other nodes (for example, adjectives or verbs) refer to.

In order to better analyze the UD structure of a given sentence, it is useful to utilize two computational linguistic concepts: **Predicate** and **Argument**. In the context of linguistics, *predicates* are verbs and their auxiliaries and *arguments* are other words (tokens) that give meaning (semantic sense) to the predicate. These concepts can be extracted using UD as a basis.

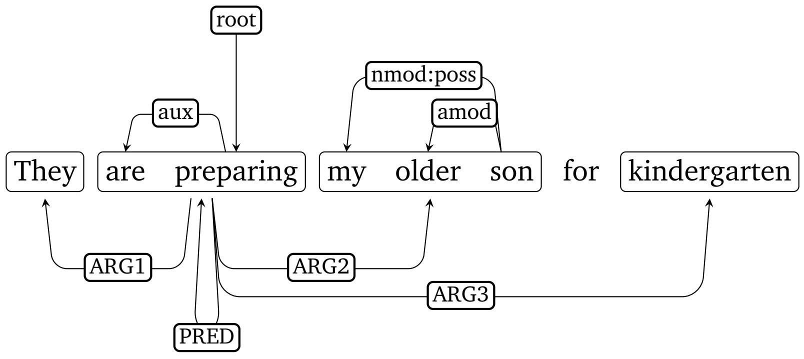


Fig. 3.3: Example of a Predicate-Argument Extraction based on UD

In Fig. 3.3 there is a simplified presentation of how predicates and arguments can be extracted with the UD scheme being used as the underlying structure in the extraction. In the figure (that appears in the documentation for the aforementioned Predicate-Argument tool), the following phrase is analyzed:

"*They are preparing my older son for kindergarten*"

The 3 argument phrases are extracted as "They", "my older son" and "kindergarten", with the predicate phrase being "are preparing". Within the "my older son" argument phrase the token "my" is suitably labeled with the UD label *nmod:poss* (for a nominal possessive modifier), whereas the token "older" is labeled as a *amod* (for a nominal adjective modifier). Additionally, within the predicate phrase "are preparing", "are" is labeled a *aux* (for auxiliary of the predicate root "preparing")

For the processing of the Selected Input, a Predicate-argument Extraction tool that uses UD [65] is the tool of choice. Regardless of the perception of the linguistic community over the usefulness of the UD paradigm, it has been proven useful for semantically oriented analysis, such as the one required for specification analysis [64]. The tool follows the following set of directives:

1st directive: Predicate root extraction

Here, UD labeled tokens of the type *nsubj*, *csubj*, *nsubjpass*, *csubjpass* (all of them denoting nouns, subjects or subject modifiers) or of the *advcl* or *acl* type (adverbial clauses or finite clauses modifying nouns) are prime candidates for predicate root. Those tokens of the *conj* type (denoting a conjunction of two nouns) are also prime candidates.

2nd directive: Argument root identification

Here, UB labeled tokens of the type *nsubj*, *csubj*, *nsubjpass*, *csubjpass* (which denote nominal nouns and modify the previously identified predicate root) or of the type *nmod* and *advmod* (which are nominal modifiers of the previously identify predicate root) are identified.

3rd directive: Argument resolution

In this step, the tool extract and manages (does resolution) of additional argument roots, based on the understanding that:
-a UD labeled *xcomp* (a clause that acts as a complement to a verb without its own subject) are not argument on their own right, but are instead subject to the root token they complement.
-a UD labeled *acl* is related to the argument root that it modifies.
-a UD labeled *conj* of an argument root is in itself an argument root.

4th directive: Predicate phrase extraction

Here, the dependency tree of the predicate is established. This means that the structure of the predicate phrase is established in relation to the predicate root extracted.

5th directive: Argument phrases extraction

Here, the dependency tree of the arguments are established. This means that the structure of the predicate phrase is established in relation to each argument root extracted.

Let one sentence from the Selected Input be taken for the same process outlined in Fig. 3.3:

The M3 MBus implementation defines two major components: a Bus Controller and a Layer Controller.

In seeing Fig. 3.4, it becomes clear that the power of UD for general POS token analysis can be properly harnessed for its usage in the ES. Yet, the above only shows how a sentence can be semantically parsed for appropriate content thanks to the Predicate-argument tool. An explanation of the internal rules that form the Relation Extraction subtask of this stage is next.

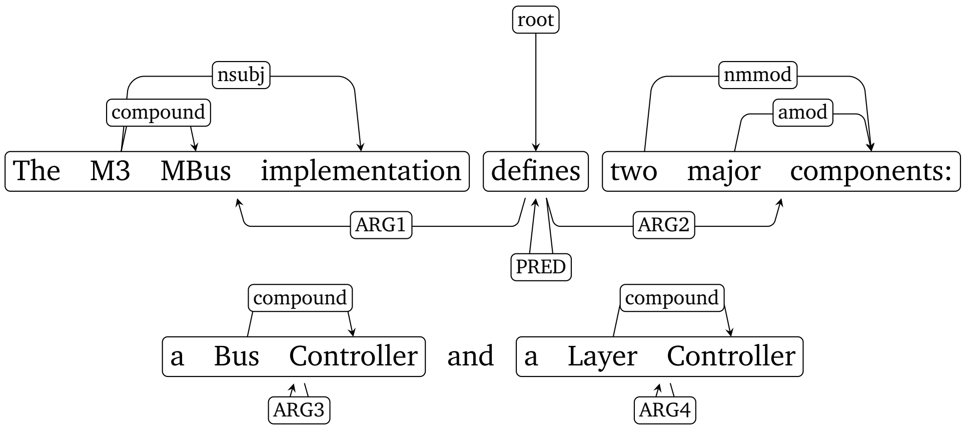


Fig. 3.4: Output of a Predicate-Argument Tool for Part of the Running Case

3.3.2.1.2 From Universal Dependencies to Triples To go from the Predicate-Argument tool output to the triples containing the knowledge in the Relation Extraction task, the output of the tool should be considered following a set of internal rules. The application of this set of internal rules highlights how the semantically oriented nature of the UD scheme and the Predicate-Argument extraction can be successfully used.

The internal rules have been devised from a heuristic approach that attempts to mimic the reasoning and conceptual understanding of a human designer. Given the well known difficulties in the modeling of expert reasoning (for instance, as part of an Expert System) [66], the internal rules are always subject to improvement efforts. As such, the internal rules should be considered as a validated approach to knowledge acquisition from NLP based specification analysis and not as an immutable set of information extraction criteria.

Furthermore, the internal rules have been devised so as to use make sense of UD labeled tokens within the ASIC domain and, as such, represent one of the fundamental cogs in the machinery of the framework. Without the internal rules, whose authorship belongs to the document writer, it would not be possible to use OIE (with UD based NLP) as a viable approach to specification analysis in the ASIC design field. The viability of the approach rests on the internal rules being able to elicit useful facts in the form of triples.

A non exhaustive set of internal rules are listed as follows:

- Predicate phrases are to be considered the semantic link between two concepts (a functional block and another functional block, a functional block and a component or a functional block and its properties, etc.). As such, predicate phrases constitute

$\boxed{\text{M3 MBus implementation}}$ **defines** $\boxed{\text{two components.}}$

$\boxed{\text{Component of MBus implementation}}$ **is** $\boxed{\text{Bus Controller.}}$

$\boxed{\text{Component of MBus implementation}}$ **is** $\boxed{\text{Layer Controller.}}$

Fig. 3.5: Domain Specific Triples from Part of the Selected Input

the relation between two entities. In the example sentence analyzed in Fig. 3.4 the predicate is the verb "*defines*".

- The root of a predicate phrase must be analyzed for relevant domain specific semantic value in the following stage of the framework. For instance, in Fig. 3.4 this root is just the verb "*define*", which strongly suggests that *Arg 2*, *Arg 3* and *Arg4* are features of *Arg1*.

- Argument phrases with both *nsubj* labeled tokens and no *dobj* labeled tokens are prime candidates for digital circuit master functional blocks, whereas argument phrases with *dobj* and *nsubj* labeled tokens are prime candidates for component blocks or properties of the master functional blocks. In Fig. 3.4, it becomes clear that "*implementation*" (the nsubj token), which is compounded with M3 MBus is a master functional block.

- A component block can be a functional block if it is possible to apply the rule above to itself (implying a hierarchy of blocks).

- *Compound* and *amod* and *nmod* labeled tokens of an argument phrase are to be taken as properties of a functional block or component block. In Fig. 3.4, "*Bus Controller*" of Arg 3, "*Layer Controller*" of Arg3 and "*two major components*" of Arg2, are either labeled *Compound* or *nmod* and constitute component blocks.

- *dep* labeled tokens identify a hierarchical link between a functional block and a property or a component block.

Applying the rules for processing, it is possible to arrive to the triples for said sentence as stated in Fig. 3.6:

The entity-relation aspect of the triples can be alternatively viewed in Fig. 3.6 (the entities are in a bubble, while the relation is in bold).

In the context of the end result of this stage, it is important to consider the precision and recall achieved by the application of the Predicate-argument tool and the internal rules. For the running case, and following the example set up in the Preliminaries section, these are presented in Table 3.4.

As can be seen in the table, the leftmost columns show the amount of information that is valuable for the understanding of the circuit. These numbers are validated by the expertise of the designer evaluating the approach.

The values for precision and recall, in the rightmost columns are not as high as desired, but the influence of these in the overall quality of the output produced by the framework is lessened greatly as the next stage of the framework comes to the forefront.

3.3.2.2 The Reasoning Stage

The *Reasoning* stage is in charge of producing the final output of the ES, that is to say i) a Power Domain Partitioning scheme and/or ii) Control Signals and Power Modes. Its inputs are two: the *triples* and the *rules*. The former are the output of the previous process, while the latter are an integral part of the ES, but independent of previous processes.

The rules are supposed to summarize and bring together the expertise that constitutes the basis for any power management decision. As is the case with facts, heuristics, guidelines and procedures, the set of rules should be refined and augmented as need be. The set of rules used will also vary depending on the output desired, but given the close link between PM, CS and PD, most of the rules will be used in producing both output i) or output ii).

Notice that the *rules* are different from the *internal rules* of the previous stage, as the *rules* are meant to be an input to the ES and concern themselves with how the triples are supposed to be interpreted in terms of the output of the ES. In summary, while *internal rules* concern themselves with the enabling of semantic understanding within the Information Extraction stage, *rules* concern themselves with how to unveil the power architecture intrinsically provided by the specification.

	Number of extracted facts	**Total number of relevant domain specific facts**	**Recall**	**Precision**
Predicate-argument tool based approach	15	22	0.6	0.3

Tab. 3.4: Scores Validating the Approach

In simplified and not exhaustive form, the rules can be grouped into 4 types, each of which is likely to be representing a portion of the heuristic method followed by a seasoned designer.

1. If any extracted entities are consistently (that is repeatedly) appearing as [entity 1] in the [entity 1] [relationship] [entity 2] triple structure, then they are singled out as presumed blocks and considered prime candidates for independent power domains.

2. Entities repeatedly appearing as [entity 2] in several triples are likely to be subordinate components of entities often appearing as [entity 1]. If these entities appearing as [entity 2] are components, they should be grouped in the same domain as those entities appearing as [entity 1], which they relate to.

3. When successful coreference resolution is present and entity 1 is a block, entity 2 probably has a hierarchical relationship with entity 1. Entity 2, even if a block in itself, is likely to signal an architectural or timing related property of entity 1.

4. If a condition is set for an entity to have some relationship with another, there is an assumption made about the entities having different activity (timing patterns).

These 4 type of rules are intended to be generic forms to make sense of triples with regards to their impact on the power management strategy and the general power architecture of the design described by the triples. In essence, the idea of each of the type of rules is to "make sense" of the triples.

Type 1 rules follow the heuristic that an entity that repeats itself as preeminent in several triples (by virtue of its placement as entity 1 in the triple format), points to an "independent" component of the design. For instance in the set of triples in Fig. 3.6, *MBus implementation* is taken to be the top level component of the design, pointing to a similarly top level Power Domain. This kind of rule bespeaks the fundamentals of the power structure of the design under analysis.

Type 2 rules follow the heuristic that an entity in a subordinate position within the triple (as entity 2), is highly likely to also be subordinate to the component defined by entity 1. Back to the set of triples in Fig. 3.6, this type of rule yields the highly probably "conclusion" that *Bus Controller* and *Layer Controller* are subordinate to *MBus implementation*. In this case, the subordination means that the *Bus Controller* and *Layer Controller* belong to *MBus implementation* and are therefore not the top level Power Domain.

Type 3 rules follow the heuristic that if there is a way to know all the relationships that signal how an entity is affected by other (which is essentially what coreference

$\boxed{\text{Bus Controller}}$ **wakes** $\boxed{\text{Layer Controller.}}$

Fig. 3.6: Another Domain Specific Triple from Part of the Selected Input

resolution enables), these relationships must be primarily analyzed as signaling possible characteristics of a component. As an example of how this type of rule operates, please notice how in Fig. 3.6, there is the case that *Bus Controller* wakes *Layer Controller*, implying two blocks with a controlling sequence. Such a useful fact leads to thinking of at least a control signal being present in the design.

Type 4 rules follow the heuristic that if there is a relationship of conditionality between entities, there are highly likely to have different activation patterns. Going back to Fig. 3.6, the application of this type of heuristic leads to considering the high probability that *Bus Controller* and *Layer Controller* should be controlled separately. This, in turn, strongly suggests one extra Control Signal, the likes of which is related to the control of at 2 Power Modes (since the conditional OFF property of *Layer Controller* supposes a corresponding ON property).

In this stage, the rules are applied within the context of a CLIPS inspired Python library for the building of Expert Systems called Experta [67]. Essentially, Experta is the software framework that allows for the Expert System to be built.

An excerpt of the code that implement the first rule can be seen in Fig. 3.7:

In Fig. 3.7 it is possible to see that the main Facts leading to the Main Rules in the class *Triple* are (in order of appearance):

- The declaration of the *setblock* fact as a resulting fact
- The declaration of the fact that once the number of instances of entity1 (a) is beyond 5, it means that the entity is prime candidate for being taken as a preliminary PD
- The declaration of the fact that entity1, relation and entity2 can be filled with the appropriate names

As it is not within the purview of the thesis to further explain the inner workings of the Experta library, this shall serve only as to familiarize the reader with the type of programming involved in the ES.

In this stage, it is possible to add another score into consideration as a quality metric for the response framework as in acts until said stage. This quality metric is the time taken for the framework to produce valid triples. Whereas said values are

```
import re

from experta import *

//...

class Triple(KnowledgeEngine):

 @DefFacts()
 def_ascertain(self):
  yieldFact(action="setblock")

 @Rule(Fact(a=P(lambda a: a >= 5))

 @Rule(Fact(action='setblock'),
   NOT(Fact(entity1=L())))
 def ask_entity1(self):
  self.declare(Fact(entity1=input("State the entity 1")))
  a += 1

 @Rule(Fact(action='setblock'),
  NOT(Fact(relation=W())))
 def ask_relation(self):
  self.declare(Fact(relation=input("State the relation type")))

 @Rule(Fact(action='setblock'),
  NOT(Fact(entity2=L())))
 def ask_entity2(self):
  self.declare(Fact(entity2=input("State the entity 2")))

 @Rule(Fact(action='setblock'),
  NOT(Fact(entity2=())))
 def ask_entity2(self):
  self.declare(Fact(entity2=input("State the entity 2")))

 @Rule(Fact(action='setblock'),
       Fact(entity1=MATCH.entity1),
       Fact(relation=MATCH.relation),
       Fact(entity2)=MATCH.entity2),
       Fact(a=Match.a))
 def setblock(self, entity1, relation, entity2, a):
     print("Introduce the triple" % (entity1, relation, entity2))
     print("Entity 1 has been instantiated enough times to be
         considered candidate for PD")


 //...
  engine.Triple()
  engine.reset()
  engine.run()
```

Fig. 3.7: Excerpt of the Rules

obviously highly dependent on the length of the sentences parsed, the amount of sentences parsed (in the previous stage), as well as on the firing of the appropriate rules programmed into the ES in the current stage, this is a number than rarely exceeds 2 seconds per sentence.

Given the amount of sentences selected as input and compensating for the need for more rules whenever the ES needs refinement, it is safe to conclude that the time spent by the ES is rather negligible when compared to the crafting of the Selected Input, a non automated procedure.

3.3.2.3 The Outputs

The ES will yield a PD partitioning scheme and a set of CS and PM.

From the sentences from the MBus implementation being taken as the specification and through the ES (both the Information Extraction and the Reasoning stages), the response framework yields a likely Power Domain partitioning scheme consisting of two Power Domains that can be switched on/off (power-gated). These two Power Domains are the Bus Controller and the Layer Controller. While the existence of the aforementioned controllers can be easily deduced from the first element in the list of selected sentences from the reference implementation, this is not enough to deduce whether or not the components belong in two different power domains. It is through the reasoning based on the rules that the existence of two Power Domains is surmised. Additionally, there is an Always ON domain that is to contain functional blocks.

Also from the selected sentences of the specification being considered, the response framework yields a set of Control Signals and Power Modes. Once the Information Extraction stage identifies the entities and their relationships, the Reasoning stage is where rules designed to set the Control Signals and Power Modes are applied.

The Reasoning stage is based on the ability to analyze the potential profile of an implementation, a description of which is taken to be the specification under analysis. Such a profile is a record of the pattern of activation/deactivation (powering on/powering off) of each functional block for each of the tasks performed by the implementation. Taking the profile into consideration, the gist of the Reasoning stage can be summarized by the following two statements:

1. A PM is a unique pattern of active/inactive states for all identified functional blocks.

2. A CS is taken as a unique way to switch one or several functional blocks into active/inactive state for each pattern.

In the case at hand, the number of PM is set to 4, while the number of CS is determined to be 3. A way to visualize these results is depicted in Table 3.5

	Sleep Controller	**Interrupt Controller**	**Wire Controller**	**Layer Controller**	**Bus Controller**
Power Mode 1	1	1	1	0	1
Power Mode 2	1	1	1	1	1
Power Mode 3	1	1	1	0	0
Power Mode 4	0	0	0	0	0
	Control Signal 1			*Control Signal 2*	*Control Signal 3*

Tab. 3.5: Control Signals and Power Modes for the MBus Device

A short explanation on how the response framework arrives at **3 CS** and **4 PM** can be offered through a validating manual analysis, as follows:

There is a a natural (to an experience designer) understanding that the Sleep Controller, the Interrupt Controller and the Wire Controller belong in the same Always ON domain, which means that they are controllable through a single Control Signal. in addition, it is clear that the Bus Controller and the Layer Controller belong in different Power Domains and that the former can be active or inactive while the latter is active. Given the fact that the Bus Controller is also capable of being Power Gated, it is clear that if such a component is inactive, the same would happen to the Layer Controller, which is dependent on it.

Following the premises outlined above, it is clear that at least 3 CS are needed for the implementation (one per each Power Domain), as well as 3 Power Modes (according to the combination of active/inactive states for the Layer Controller and the Bus Controller). The extra (fourth) PM is shown in Table 3.5 as a Power OFF Mode corresponding to the non operative status of the Always ON domain, which is a typical PM for any circuit.

3.3.2.4 Limitations of the Response Framework

Based on the description of the stages of the response framework, it becomes clear that said framework fulfills its purpose only under certain conditions of operation. The extent of these limitations does not invalidate the efficacy of the framework, but does reveal areas in which further work is necessary to improve its effectiveness.

The limitations can be grouped into 3 categories:

1. Working assumptions: these are limitations that relate to certain structures for the framework to properly work. For instance, the Selected Input of the framework needs to be both manually selected (which precludes full automation) as well as selected for clarity in its sentences. A specification with extremely verbose sentences is not apt for the framework.

2. Efficacy assumptions: these are limitations related to how the framework is designed to operate in order to conduct proper analysis. For instance, the internal rules in the Information Extraction stage need to be comprehensive enough so as to yield a recall over 0.4 (at the least) so that as many useful facts are properly turned into triples. Furthermore, this category of limitation also applies to the rules as input to the Reasoning stage, since these have to capture the heuristics of the expertise of the designer in order to turn the set of triples into the Power Domain partitioning scheme and/or the setting of Power Modes and Control Signals. If either the *internal rules* in the IE stage or the *rules* in the Reasoning stage are insufficient in number or adequacy, the end result will yield extremely simplified results, which render the output inefficacious.

3. Output assumptions: these are limitations that relate to how the output is manifested. For instance, in the running case at hand, the assumption that a Control Signals imply at least 2 Power Domains enables the determination of a Power Domain partitioning scheme. In the same vein, for the sake of simplicity, a Control Signal is considered responsible for only 2 internal power states (ON/OFF) of Power Domains, leading to a reduced number of Power Modes. This type of assumptions limit the framework to very basic outputs, which shun DVFS or other power management techniques, as well as forcing output i (the PD partitioning scheme) and output ii (the CS and PM) to be determined at the same time.

The limitations of the response frameworks have several origins, but they essentially relate to technical limitations of both Expert Systems, as well as Open Information Extraction tools and frameworks. In addition, there are limitations related to implementation issues stemming from the original nature of the approach presented by the framework. Irrespective of the origin, these limitations do not imply that the framework is not usable, but point to the what it requires to be be usable.

3.4 Concluding Remarks

Since the need to parse a natural language specification is becoming more ubiquitous, tools have been developed to aid the designer in this process. While the lack of ontologies is a roadblock to more efficient and fine tuned Information Extraction within the ASIC design field, Open Information Extraction (OIE) is still effective in providing valuable assistance for some specification analysis tasks such as extraction of verification statements and ASIC property recognition.

The main contribution presented here is an Expert System (ES) based on Universal Dependencies (UD), a Dependency Grammar scheme that is semantically oriented. The use of UD within the Expert System makes it possible to elicit useful and relevant facts from a Selected Input (sentences from a specification) from the triples yielded via the use of the internal rules governing the Dependency Grammar scheme. These triples working together with the ES rules are meant to mimic the reasoning of a designer with regards to extracting valuable knowledge to finally output a Power Domain partitioning scheme and/or the setting of Control Signals and Power Modes.

4 Application of the response framework

4.1 ASIC Designs from Natural Language

When the specification a designer has as its main design document is in natural language and the digital circuit being designed is an ASIC, it is very likely that said specification will be either: i) a technical standard filled with flowcharts, graphs and other text elements, together with prose, or ii) a reference implementation, filled with list of variables and configuration options, as well as specialized comparative graphs that purport to show how the implementation complies with a given standard. In addition the latter are more likely to contain textual descriptions of the intended operation flows in prose.

Typically, an ASIC design is under the need to comply with strict regulations involving physical variables up to the transistor level, or the gate level, as well as area or power budgets, closely tied to floorplanning, routing and other tasks that are performed at lower levels of abstraction. A technical standard normally lists numerical ranges for important variables that the circuit has to manage, sometimes presented in tables, sometimes presented in graphs. Said data constitute the majority of the content of a specification that is a technical standard. However, it is also traditional for a technical standard to have a very succinct explanation of the flows of operation that the ASIC design has to implement.

As was shown in the previous chapter, the response framework does not need to have a large text input to be able to select worthy sentences as input to the Expert System. When the natural language explanation (in prose) of the normal operation flow of the design is sufficiently "fleshed out" (detailed), the response framework can provide assistance with setting the *Power Domains* (PD) (by yielding a Power Domain partitioning scheme), as well as in setting the *Control Signals* (CS) and *Power Modes* (PM).

As a way to showcase the efficacy and applicability of the response framework with use cases featuring designs representatives of the ASIC world, two were selected. They are: a *LZW encoder* and an industrial grade *HDMI Port Processor*.

Encoders are archetypal circuits used in *Internet of Things* (IoT) devices, They transform data (encode) from one representation to another and need to be fast and consume very few resources or else it may become better for the designer to implement the encoding process in some embedded form in a general purpose processor. In addition, the typical sequential nature of the encoding processes lends itself to a set of functional blocks that operate passing each other information with very distinct timing patterns for the process to be successful.

The encoding processes implemented by encoders can be multiple or singular, but are well defined and extremely likely to be standardized in either a technical standard on in a reference implementation of some kind. Textual descriptions of the operation flow in terms are regularly found in reference implementations or in informal sources in which the encoding process is described.

In view of the above, a LZW encoder is a natural selection for a use case in which to test the response framework. The fact that the LZW algorithm is an algorithm requiring repetitive steps and non-trivial activation profiles further vouches for its inclusion as a use case.

With regards to the HDMI Port Processor, several salient points merit its selection as a use case. HDMI being a well known technology in use to interface High Definition Video and Audio (hence the M, standing for Multimedia), an HDMI port processor is a typical ASIC solution for the hardware world. HDMI ports are meant to be the connecting block that allows transmission and retransmission of HDMI compliant data from sources to processing or displaying devices (such as a Digital TV). HDMI is a well known and well supported standard, which requires implementations of port processors to support switching from sources. Said requirements imply that the Port Processor is bound to require memory blocks and some method of polling, as well as a control unit, further implying not only several modes of operation, leading to several PD, but several CS and PM, as well.

Furthermore, as a well known and well supported standard, the actual HDMI technical standard specification is, unfortunately, not readily accessible, and contains prose and tables and graphs in excess of 200 pages. While a proper understanding of the relevant sentences and textual data from this technical standard specification requires expertise, the specification that shall be considered in this chapter is that of the HDMI Port Processor itself, which, in turn, contain plain language descriptions of operation flows, which are sufficient for the response framework to be of use.

In view of the above, the HDMI Port Processor is also a natural selection for a use case to showcase the efficacy of the proposed response framework, as it combines

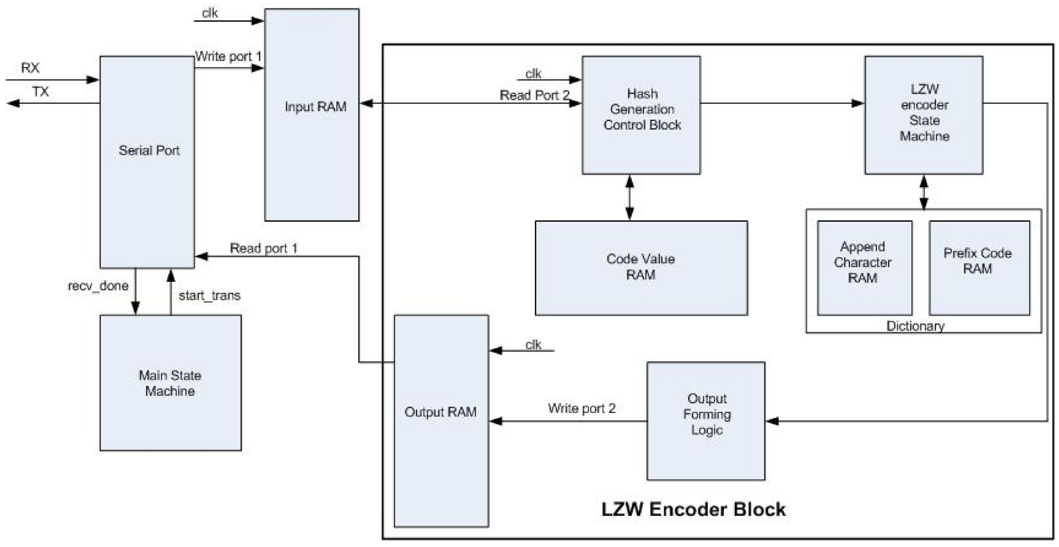


Fig. 4.1: Block Level Diagram of LZW Encoder based on[71]

the allure of being an industrial grade example with the simple nature of an ASIC that does the job its specification describes.

4.2 The LZW Encoder

LZW encoders have been implemented repeatedly in VLSI fashion [68] [69] [70], throughout the last 30 years. They are in widespread use within the industry on account of their application of a very flexible and highly reliable algorithm.

Unfortunately, the encoding algorithm is hardly ever explained in written form, so there is no real specification for an encoder. However, the popularity of the algorithm has led to several examples of implementations beyond the gate level. Some of these implementations provide some documentation explaining design aspects of the proposed LZW encoder. Among them, one was selected for its convenient access.

The selected LZW encoder is an open source design which is designed to be FPGA synthesizable, but has a matching SystemC implementation. This latter implementation is supplemented with a textual description of the proposed architecture of the encoder [71]. This architecture (the Block Level Diagram) is also featured in a figure, which is reproduced next.

As can be gleaned from Fig. 4.1, the proposed architecture contemplates the following functional blocks:

- *Serial Port*
- *Input RAM*
- *Main State Machine*
- *Hash Generation Control Block*
- *LZW encoder State Machine*
- *Code Value RAM*
- *Output Forming Logic*
- *Output RAM*
- *Append Character RAM*
- *Prefix Code RAM*

Not all of the listed functional blocks are in the same hierarchical level of "importance". As a mater of fact, from simple inspection it is noticeable that *Append Character RAM* and *Prefix Code RAM* are both part of a potential *Dictionary* block. In the same vein, with the exception of *Serial Port*, *Input RAM* and *Main State Machine*, the remaining functional block can all be grouped under a larger functional block called *LZW Encoder Block*.

In view of the above it is possible to foresee that *Serial Port Input RAM* and *Main State Machine* are prime candidates for Power Domain status, as is the "super" functional block *LZW Encoder Block* with all of its child blocks.

All of the information taken from Fig. 4.1 is very useful, but unfortunately cannot be extracted by the framework proposed herein, as the framework only deals with plain text. However, it is possible to extract very similar conclusions via the use of the framework. For this to happen, a cursory examination of the same specification from which the Fig. 4.1 comes from is conducted in order to find descriptive paragraphs. In order to do make use of the framework the following descriptive paragraph is chosen as the text source:

The main state machine controls the data movement and control flow for all the total design including the serial port and the LZW encoder block. The input RAM is used to store the data received from the serial port of the PC. The LZW encoder block implements the LZW algorithm, both the control and the data path. The code value RAM is the Hash Table implementation, while the dictionary block implements the LZW dictionary. The output forming logic breaks and merges the 13-bit data output from the LZW data path on correct 8-bit boundaries before writing it to the output RAM. The output RAM is used to store the compressed data which is than transmitted through the serial port to the host PC to be displayed on the terminal.

The above paragraph contains the Selected Input to the Expert System. These sentences are:

- The main state machine controls the data movement and control flow for all the total design including the serial port and the LZW encoder block.
- The input RAM is used to store the data received from the serial port of the PC.
- The LZW encoder block implements the LZW algorithm, both the control and the data path.
- The code value RAM is the Hash Table implementation, while the dictionary block implements the LZW dictionary.
- The output forming logic breaks and merges the 13-bit data output from the LZW data path on correct 8-bit boundaries before writing it to the output RAM.

Once the Selected Input has been established, the next stage of the ES comes involves the extraction of domain specific triples from it.

4.2.1 Extracting the Triples

Based on the sentences above and the resulting triples after the Information Extraction stage of the response framework can be seen in graphical form as a semantic map in the following figure:

The semantic map can be read in triples in following the formula Main/Secondary Block-Relationship-Main/Secondary Block. There are 8 such triples. Some of them are:

1. Main State Machine-controls-Serial Port.

2. Serial Port-uses(stores in)-Input RAM.

3. Output Block-writes-Serial Port.

The triples are obtained in the same fashion described in the preceding chapter. So as to showcase how Universal Dependencies (UD) are used in parsing the sentences for semantically validated concept, the next paragraphs offer a succinct explanation.

Let the sentence "*The input RAM is used to store the data received from the serial port of the PC.*" be analyzed following the use of UD, with the Predicate-argument tool. Doing this would yield the result shown in simplified form in Fig. 4.3. The form clearly identifies the phrase "is used to store" as a predicate (and therefore a potential relationship), while having "the input RAM" and "the data received from the serial port of the PC" as arguments (that is to say, potential entities). Is it therefore possible to apply the internal rules after the Predicate-argument tool has done its job (as presented in the previous chapter) to achieve the aforementioned number of 8 triples.

As has been explained before, the quality of the triples as units of information to be reasoned in the next stage of the framework can be measured by the precision and recall scores. These values are expressed in Table 4.1. Once again, while these numbers are not as high as desired, their impact in the overall purpose of the framework is diminished by the use of intelligent rules for reasoning.

The input RAM is used to store the data received from the serial port of the PC.

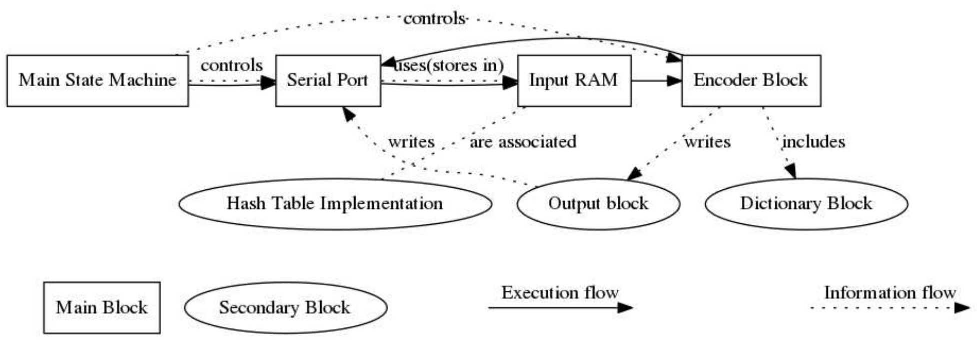


Fig. 4.2: LZW Encoder Semantic Map

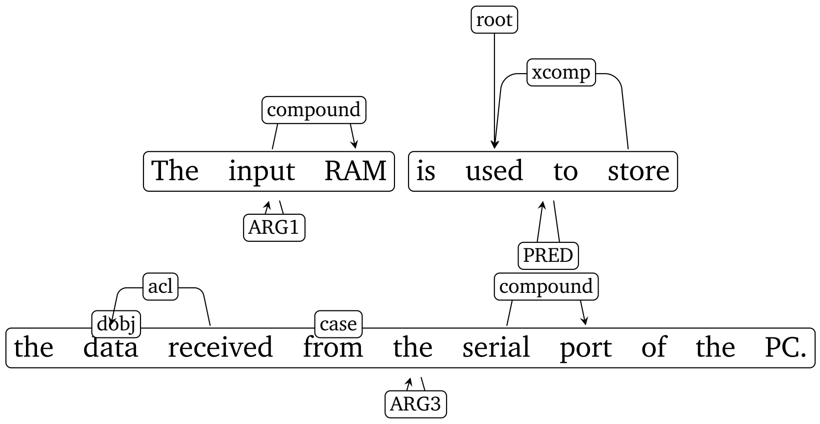


Fig. 4.3: Simplified Predicate-Argument Output for Part of the LZW Encoder Specification

	Number of extracted facts	**Total number of relevant domain specific fact**	**Recall**	**Precision**
Predicate-argument tool based approach	15	12	0.66	0.53

Tab. 4.1: Precision and Recall within the LZW Encoder Use Case

4.2.2 Rules and Reasoning

Fig. 4.2 contains distinctions between the type of functional blocks and the type of data flow (execution or information) between said functional blocks. This simplification is not within the purview of the Expert System implementing the response framework, but is only done in order to better illustrate the conceptual architecture for the setting of a Power Domain partitioning scheme.

Once the triples have been obtained, the triples have to be fed into the Reasoning stage along with the rules. As it is evident from Fig. 4.2, the subtask has to establish hierarchical links between the functional blocks (the semantic entities) so as to yield a Power Domain partitioning scheme. In order to do this, the Reasoning stage employs rules. Some rules can be shortly stated as follows:

R1: Blocks that are linked to at least 2 other blocks are considered candidates for their own Power Domains as they are likely to control others and, therefore, have a longer active time.

R2: Any two blocks not directly linked to each other through the execution flow belong to different domains.

R3: Blocks relating to each other through hierarchical dependencies are grouped together in a single power domain.

Rules R1 and R2 are Type 1 and Type 2 rules, respectively (following the classification expressed in the previous chapter). Let it be recalled that Type 1 rules and Type 2 rules are about unveiling hierarchy between potential blocks of the design (Main/Secondary Blocks), as well as understanding which blocks are prime candidates for Power Domains. Rule 3 is a Type 3 rule, because it concerns itself with strongly suggesting that the hierarchical structure unveiled by Rule 1 and 2 leads to Secondary blocks signaling an architectural property of Main blocks.

After the application of the rules, the framework can output a PD partitioning scheme and values for Power Modes and and Control Signals. This is explained in the following subsection.

4.2.3 Power Domain Partitioning Scheme and Power Modes

After the application of rules of the previous step, the response framework yields a Power Domain partitioning scheme that consists of **4 PD**: *Main State Machine*, *Serial Port*, *Input RAM* and *LZW Encoder Block*

Also after the application of a set of rules modeled after the previously mentioned fashion, the Expert System yields **4 CS**, one per each PD, as well as **8 PM** (considering an ON/OFF state for each PD other than the one of the Always ON PD).

This Power Domain Partitioning scheme and set of Power Modes is explained in more human oriented terms both by the LZW Encoder semantic map in Fig. 4.2 and by the following reasoning process:

"The *Main State Machine* controls the *Serial Port* and the *Encoder Block*. The *Serial Port* writes/stores in the *Input RAM* and interacts with the *Encoder Block*, following the Execution flow". As reflected in Fig. 4.2, the aforementioned elements of the design are Main Blocks and each of them is considered a distinct Power Domain.

The independent control of the Power Domains is based on the fact that 3 of the Blocks (*Serial Port*, *Input RAM* and *Encoder Block*) interact (communicate) with each other throughout the Secondary Blocks (*Hash Table Implementation*, *Output Block* and *Dictionary Block*) at presumably different times following the Information flow. The *Main State Machine* block controls both *Serial Port* and *Encoder Block* directly (and Input RAM indirectly), further cementing that there is a need for each Power Domain to have its own Control Signal.

With regards to the Power Modes, the reasoning process takes the conclusion of the setting of 4 CS (one per each PD) would at the very least 16 PM, given that each Power Domain may be ON/OFF (16 comes from a $2\hat{4}$ for a two state variable independently sorted in 4 different combinatorial elements). However, from inspection of the LZW Encoder semantic map, it becomes clear that the Main State Machine is in an Always ON PD. This halves the potential PM number to 8.

4.2.4 Validation of the Power Domain Partitioning Scheme and Power Modes

So as to assess how valid the suggested power partitioning scheme really is, the testbench of the LZW Encoder is run in order to simulate the circuit in SystemC. During the run, timestamps for the activation patterns of the different blocks and subblocks are logged for further inspection. This allows for the determination of when the *Serial Port* is active, when the *Input RAM* is being fed, when there is a change of states within the *Main State Machine* and when the *LZW Encoder Block* is being used.

Thanks to the log data and the control signals that drive the active/inactive state of a block, it is possible to construct a basic *Power State Table* (PST) for each of the power domains. In this scenario, the PST stores the times for which a power domain is ON or OFF, active or inactive, according to the value of the control signals. In the case of the *Main State Machine*, the PST consists of a single "always on" state, as this block controls the execution flow of the entire system.

So as to validate the impact of the Power Domain partitioning scheme, PKTool, a power estimation tool for SystemC [72] is used together with the PST. In order to simplify the validation mechanism and without delving into how precise the power estimation is, a very simple model from PKTool is chosen. The *fixed_power* model from PKTool, which calculates the energy following the equation $E = PT$, where E is the energy, P is a fixed power value (provided by the us) and T is the simulation time for each state (calculated by PKTool through the PST). A value of 0.005 mW is adopted as a static power consumption value for an inactive module, while 1 mW is adopted for the fixed power consumption of an active module. These values are mere approximations based on previous knowledge and are only meant to illustrate the differences in power consumption of a Power Domain when active and when inactive.

Tab. 4.2: Comparison Between Suggested PD Partitioning Scheme and Single Domain Scheme for the LZW Encoder

	Energy consumption [nJ]				
Partitioning scheme	**Main State Machine**	**Serial Port**	**Input RAM**	**Encoder Block**	**Total**
Single domain scheme	389.2360	389.2360	389.2360	389.2360	1556.9440
Four domain scheme	389.2360	388.0310	21.7799	26.5861	825.7577

So as to show that the scheme determined by our approach is a solid decision, let there be a comparison between the suggested Power Domain partitioning scheme (the Four Domain scheme) and a Single domain scheme. Further description of these schemes is as follows:

- *Single domain scheme*: it consists of only one power domain that includes all four of the main blocks and is on an active state throughout the entire time. Essentially, this is equivalent to performing no power domain partitioning.

- *Four domain scheme*: it consists of four power domains (one for each of the main blocks). This is the partitioning determined by the response framework.

By using PKTool as advertised, the results regarding energy consumption during the simulation run for both schemes is presented in Table 4.2. The rows in the table represent the schemes under comparison while the columns show the energy consumption for each of the modules and also the total energy consumption. As the single domain scheme simply neglects the activity profile of the modules, the total energy consumption for it is considerably higher than that of the four domain scheme in the second row (15556.9440 nJ vs 825.7577 nJ, respectively). The first row shows that the consumption of each module for the single domain scheme is the same, implying that all the modules belong in the same power domain (which follows the activity profile of the Main State Machine module). The second row shows that the four domain scheme considers the activity profile of the modules. Both the *Encoder Block* and the *Input RAM block* have activity profiles which show that they are inactive for longer periods of time, thereby consuming less energy than the *Main State Machine*.

It is then evident that the suggested 4 Power Domain partitioning scheme significantly reduces the power consumption, which means that the response framework has output a validated scheme.

4.3 The HDMI Port Processor

An HDMI Port Processor is usually considered a circuit very likely to be designed and manufactured as an ASIC. As HDMI itself is an interface (with its own protocols spanning from the physical layer to the link layer of the OSI model) there exists a great number of industrial grade implementations. Regardless of the myriad of implementations with different levels of support for the Physical and Link Layers, the salient point of any HDMI compliant device of this kind is its support for managing at least 4 HDMI sources for transmission and retransmission.

Without entering into a degree of detail that this document need not espouse, a HDMI Port Processor needs to be complaint with the HDMI standard [73]. The standard is a moderately lengthy document filled with tables and graphs and contains very little plain text from which to extract the required Selected Input for the response framework. However, by looking into the datasheet (a specification) of an actual implementation, the issue can be rapidly addressed.

The datasheet of an industrial grade implementation by Silicon Image [74] is therefore chosen as the specification from which to extract text information. The following (non exhaustive) list of sentences (or phrases) have been extracted from the aforementioned specification and serve as part of the Selected Input:

- The four HDMI/DVI receiver ports are defined as Port 0, Port 1, Port 2, and Port 3.
- Each of the ports is terminated separately and equalized under the control of the receiver digital block and is controlled by the local I2C bus.
- The transmitter block sends an HDMI content stream based on the content delivered from the selected source.
- The port processor has 256 bytes of NVRAM for storing common EDID data that can be used by each of the ports.
- An additional 64-byte block of NVRAM is used by the Auto-Boot feature, which initializes some of the registers used to enable the EDID for the respective port.
- The EDID block consists of 1280 bytes of SRAM.
- Each port has a block of 256 bytes of SRAM for EDID data,
- Both the NVRAM EDID data and NVRAM Auto-Boot data should be initialized by software using the local I2C bus.

While these selected sentences may look disjointed and verbose, it is still possible to infer a partial picture of the architecture of the HDMI Port Processor by parsing them appropriately. However, so as to guide the reader and a clearer understanding of the architecture that the sentences seemingly describe, please see Fig. 4.4 directly taken from the specification itself:

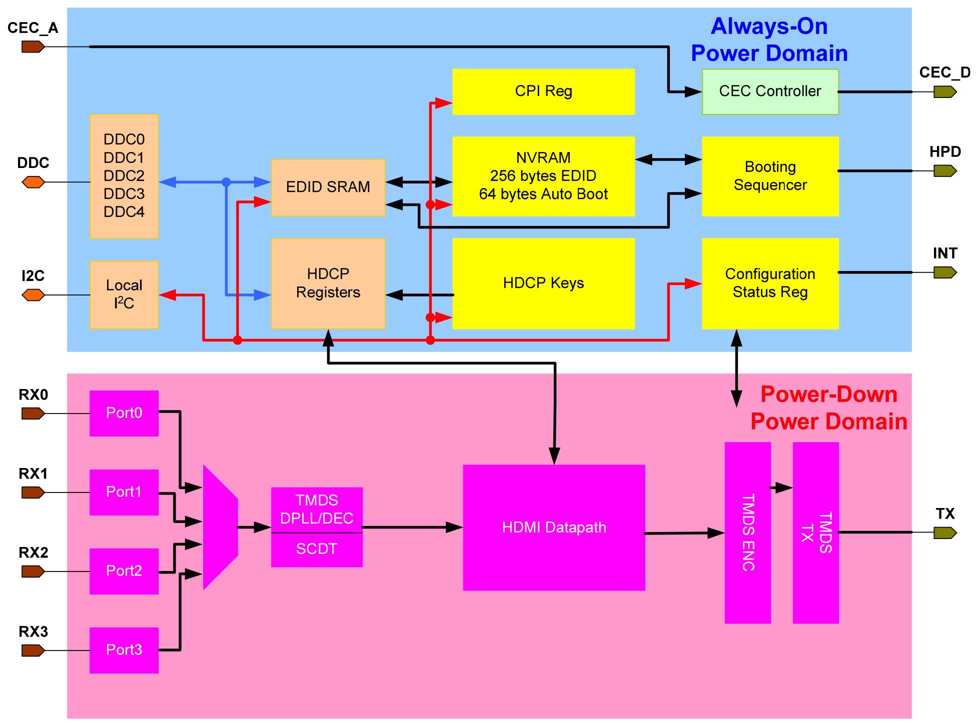


Fig. 4.4: The Basic Architecture of the HDMI Port Processor from [74]

Fig. 4.4 shows how the main blocks are connected and explicitly states 2 Power Domains (An Always On Power Domain and a Power Down Power Domain) for the design. Achieving a similar conclusion via the use of the response framework is the challenge at hand. To face the challenge, the response framework is fed the Selected Input. The next step is to extract the triples.

4.3.1 Extracting the Triples

Once again, thanks to the use of Universal Dependencies it is possible to parse the sentences and produce a set of useful triples containing information describing the structure of the design.

In order to briefly show how the UD based Predicate-argument tool that is the base of this stage works upon the selected sentences, let the following sentence be taken as an example:

The four HDMI/DVI receiver ports are defined as Port 0, Port 1, Port 2, and Port 3.

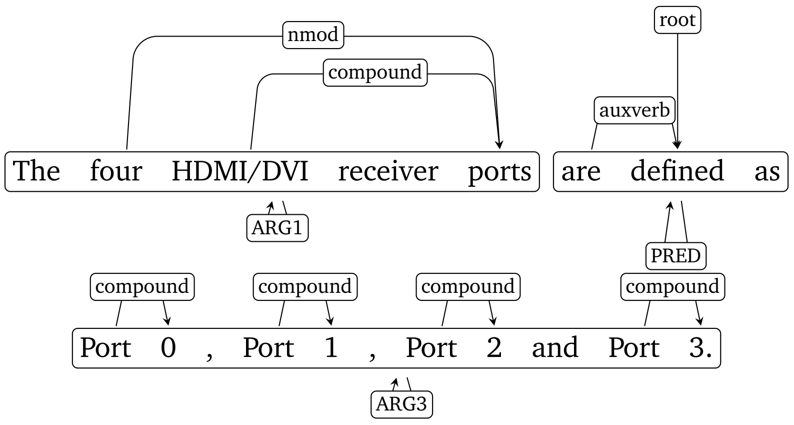


Fig. 4.5: Simplified Predicate-Argument Output for Part of the HDMI Port Processor Specification

By looking at Fig. 4.5 it is possible to notice its resemblance to previous such outputs. Owing to this, belaboring the fact that the internal rules of the Predicate-Argument tool work to produce the first candidate triples from the relevant facts is unnecessary. However, it is important to notice that the structure of the example sentence is rather simple, as is that of the other sentences constituting the Selected Input. While this makes validated triples easier to build, how useful these can be is still affected by the depth of the semantically oriented information that can be extracted from them.

By applying the technique detailed in the previous chapter, it is possible to arrive to a number of 12 valid triples that encapsulate the useful facts. These include, but are not limited to the following:

- The ports-are controlled by-the bus
- The Transmitter block-transmits-from the ports
- NVRAM-is used by-the ports
- The ports-use-the EDID
- NVRAM-is used by-AutoBoot Feature

	Number of extracted facts	**Total number of relevant domain specific fact**	**Recall**	**Precision**
Predicate-argument tool based approach	30	22	0.54	0.4

Tab. 4.3: Precision and Recall within the HDMI Port Processor Use Case

As this stage concludes, it is important to know how did the stage far with regards to recall and precision. The stage outputs 22 relevant facts and 30 facts total. The numbers are condensed in Table 4.3.

The values of 0.54 and 0.4 for recall and precision are within the "expected" range. Most notably it is worth mentioning that the simple grammatical structure of the selected input sentences and phrases compensates the extra elements of information that they also contain. The writing style of the specification thus plays a significant role in how clear the Information Extraction stages can become.

The listed valid triples seem as disjointed as the non exhaustive list of selected sentences (or phrases), which leads to a reduced knowledge of the structure of the design, thereby requiring more refined rules to produce a PD partitioning scheme. Some of these rules are presented in the following subsection.

4.3.2 Rules and Reasoning

As each of the valid triples may not contain deep semantically relevant knowledge in the form of useful facts, the rules of the Expert System need to be more exhaustive and detailed than in the previous use case. This need implies rules that are ad-hoc for use cases like that of a datasheet, which is a very particular type of specification typically less textually verbose than other types of specifications.

Among the rules that are used in this particular case, there is a higher number corresponding to type 2 and 4 of the list of general rules. So as to refresh the reader's memory, the description of type 2 and 4 of the general rules is given below.

2. Entities repeatedly appearing as [entity 2] in several triples are likely to be subordinate components of entities often appearing as [entity 1]. If these entities appearing as [entity 2] are components, they should be grouped in the same domain as those entities appearing as [entity 1], which they relate to.

4. When successful coreference resolution is present and entity 1 is a block, entity 2 probably has a hierarchical relationship with entity 1. Entity 2, even if a block in itself, is likely to signal an architectural or timing related property of entity 1.

Exemplifying both the ad-hoc nature of the rules and the belonging to type 2 or 4 are the following:

- If an entity controls a "bus" type entity, it is prime candidate for inclusion in an Always ON Power Domain
- If an entity is used by another entity which has an initialization feature (architectural property), the latter entity is prime candidate for an Always ON Power Domain.
- If an entity has a name including the word or parts of the word "control", it is prime candidate for inclusion in a Power Domain with other similar entities.
- The existence of a initialization feature presupposes at least 2 distinct Power Modes connected to the entity which possesses the feature.
- A "bus" type entity presupposes at least one control signal for each Power Domain.

The above rules are laid out in the most generalizable terms possible, yet are distinctively part of the need for refinement on the part of the designer being aided by the response framework. The latter 2 rules of the list, in particular, are heuristics based and lead to a better understanding of the relationship between a Power Domain partitioning scheme and the setting of Power Modes and Control Signals.

After the application of the rules, the framework can output a PD partitioning scheme and values for Power Modes and and Control Signals. This is explained in the following subsection.

4.3.3 Power Domain Partitioning Scheme and Power Modes

The response framework for this use case outputs a **2 PD** partitioning scheme:

1. An Always ON Power Domain with the NVRAM and a bus.
2. A second Power Domain with the Ports and the Transmitter block.

Regarding the setting of Control Signals and Power Modes, the response framework yields a result of:

1. **4 PM** (1 FULL ON Mode, 1 ALWAYS-ON only Mode, also know as STANDBY Mode, 1 OFF Mode).
2. At least **2 CS** (one per each Power Domain).

The values presented above are not atypical for a specialized ASIC such as a HDMI Port Processor. They are, unfortunately, rather basic and simple, in spite of the effort incurred on by increasing the ad-hoc rules. It must be said, however, that basic and simple may mean incomplete, but not wrong.

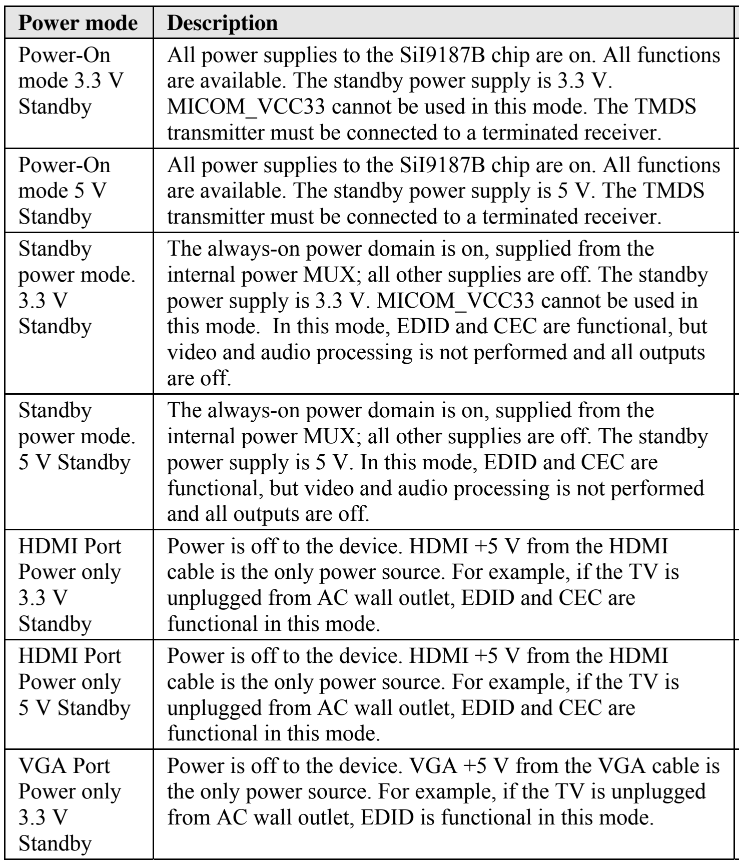
To validate the output (and hence the suitability) of the response framework a comparison to an already valid PD partitioning scheme and set of PM and CS is required. A way to do this is to compare to already validated settings provided by the specification on how it handles the PD, PM and CS.

4.3.4 Validation of the Power Domain Partitioning Scheme and Power Modes

As the case under analysis corresponds to a finished industrial grade product, it is not possible to simulate the design or to evaluate other alternatives since the architecture is fixed. This issue is countered by the fact that, as was previously mentioned, Fig. 4.4 contains sufficient information about the Power Domain partitioning scheme.

By inspecting Fig. 4.4 more clearly, it is possible to conclude that the Power Domain partitioning scheme output by the response framework is valid, albeit incomplete. NVRAM (*NVRAM*) is indeed a block situated within the Always ON Power Domain, as is a bus (*Local I^{2}C*), but there is a very important module that was not considered by the output: the CEC Controller. This omission is the byproduct of the "indirect" wording style of the sentences and phrases related to this entity and their lack of links to it from other blocks. Further inspection reveals that the Ports (*Port 0, Port 1, Port 2, Port 3*) and the Transmitter Block (*TMDS* named blocks) are indeed in the second Power Domain, named *Power-Down Power Domain*. A further figure (Fig. 4.7) extracted from the some specification (datasheet) confirms those findings, even adding extra information about voltage sources that have never been relevant data for the response framework to tend to.

As it pertains to the setting of Power Modes and Control Signals, the datasheet provides detailed information, as is required of a document of that nature. The information is neatly summarized by a table figure extracted from said document that is available as Fig. 4.6.



Power mode	Description
Power-On mode 3.3 V Standby	All power supplies to the SiI9187B chip are on. All functions are available. The standby power supply is 3.3 V. MICOM_VCC33 cannot be used in this mode. The TMDS transmitter must be connected to a terminated receiver.
Power-On mode 5 V Standby	All power supplies to the SiI9187B chip are on. All functions are available. The standby power supply is 5 V. The TMDS transmitter must be connected to a terminated receiver.
Standby power mode. 3.3 V Standby	The always-on power domain is on, supplied from the internal power MUX; all other supplies are off. The standby power supply is 3.3 V. MICOM_VCC33 cannot be used in this mode. In this mode, EDID and CEC are functional, but video and audio processing is not performed and all outputs are off.
Standby power mode. 5 V Standby	The always-on power domain is on, supplied from the internal power MUX; all other supplies are off. The standby power supply is 5 V. In this mode, EDID and CEC are functional, but video and audio processing is not performed and all outputs are off.
HDMI Port Power only 3.3 V Standby	Power is off to the device. HDMI +5 V from the HDMI cable is the only power source. For example, if the TV is unplugged from AC wall outlet, EDID and CEC are functional in this mode.
HDMI Port Power only 5 V Standby	Power is off to the device. HDMI +5 V from the HDMI cable is the only power source. For example, if the TV is unplugged from AC wall outlet, EDID and CEC are functional in this mode.
VGA Port Power only 3.3 V Standby	Power is off to the device. VGA +5 V from the VGA cable is the only power source. For example, if the TV is unplugged from AC wall outlet, EDID is functional in this mode.

Fig. 4.6: Power Modes of the HDMI Port Processor from [74]

The table shows that the design is characterized by 7 Power Modes (2 of which are Power ON Modes, while 5 are STANDBY modes). While it may be tempting to outright dismiss the Power Modes setting output by the response framework, as being wrong, a closer inspection reveals that the Power ON Modes differ in matter of which is their voltage source, with the same happening in similar forms to the remaining STANDBY modes. In light of this "revelation", it may be permissible to conclude there are indeed 2 main Power Modes: a Power ON Mode and a STANDBY Mode. An OFF Mode is not considered by the datasheet, but some of the Power Modes alluded to in the last rows on the table figure are similar in spirit to that OFF Mode.

Finally, as it comes to Control Signals, it is clear from the previous figures from the datasheet that there is a need for at least a Control Signal for each Power Domain, as per the output of the response framework. The fact that the datasheet includes voltage source information leads to more required Control Signals, as it becomes clear that each voltage source will lead to an extra Control Signal being required.

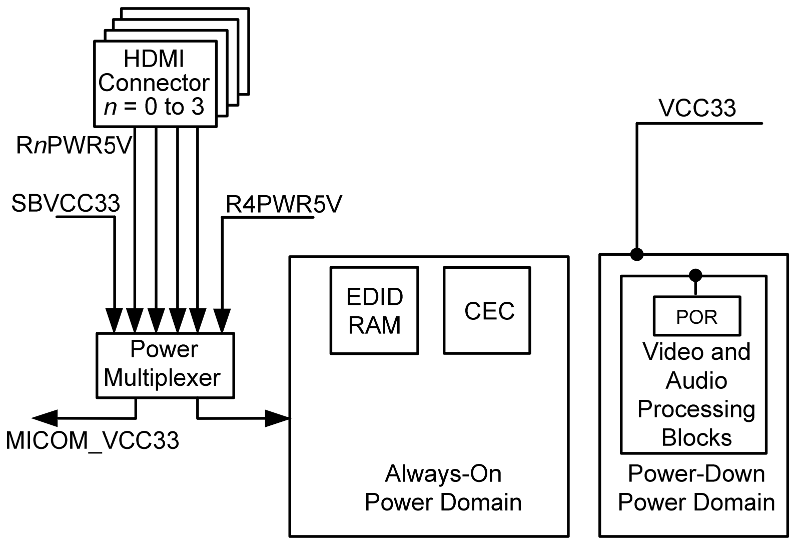


Fig. 4.7: Simplified Power Architecture of the HDMI Port Processor from [74]

Based on the rationale above it is reasonable to state that the output by the response framework has been validated by the comparison to the decisions already implemented for an industrial grade finished design.

4.4 Concluding Remarks

In this chapter, two use cases (*LZW Encoder* and *HDMI Port Processor*) were selected to demonstrate the efficacy of the response framework presented in the previous chapter. Guiding the reader through the stages of the framework it is possible to conclude that the response framework outputs valid Power Domain partitioning schemes, as well as similarly valid settings for Power Modes and Control Signals, in a short time, which is undoubtedly a very useful feature for early Design Space Exploration.

Through the use cases, it has also become apparent that the rules that govern the most important stage of the Expert System (which is at the core of the response framework), need to be refined for every situation. This step is still a manual task, but it greatly aided by the accessible and easy way in which the rules can be devised. It is important to note that, while the Information Extraction stages of the ES have recall and precision scores below 0.7 and 0.5 respectively, these are not as relevant to the end output of the framework as the pertinacity and wide scope desired of the ad-hoc rules. Some other difficulties encountered relate to the wording of the Selected Input (sentences and phrases), are lessened by the use of Universal Dependencies (UD), although this is an area in which Predicate-argument tools are still in their infancy.

Part II: Technical Language Specifications

5 Technical Language: Preliminaries, Relevant Work and Response Framework

5.1 Preliminaries

In the realm of IC development, especially so for ASIC, rapid prototyping is crucial in the context of regular *Design Space Exploration* (DSE). The reason behind the crucial nature of rapid prototyping is not only because of the need to accelerate the early stages in development as per the regular hurry to get the product ready as soon as possible, but also because it is hardly possible to test and verify compliance for an intended implementation when the implementation is nothing but a text document in natural language or in a block diagram.

Due to the ever-increasing optimization required of the modern ASIC, non-functional design aspects such as power consumption are now gearing rapid prototyping towards the use of power-aware DSE tools [75]. Additionally, it is important to mention how the top-down approach to designing ASIC has been gaining traction, as it becomes more and more difficult to design an ASIC likely to be used in a bigger *System on Chip* (SoC) without consideration to how it shall fit in the overall architecture of said SoC. Hence, the move is towards the *Electronic System Level* (ESL), a system level in which the main point is the use of technical languages such as SystemC, which make the necessary abstractions possible [76].

Traditionally up until the early 2000s, the level at which power-aware techniques were applied was the *Register Transfer Level* (RTL). As has been discussed previously in an earlier chapter of this document, the RTL is close enough to the physical implementation of an IC that it is possible not only to measure or estimate design variables that impact on power, but also to use many of the widely available optimization tools and algorithms. Unfortunately, it is not possible to simulate or estimate the architectural impact of power-aware decision and it additionally very time consuming to simulate certain scenarios common to DSE (as evaluating different architectures).

Furthermore, the opportunities to decrease power consumption at the RTL are more limited than those at a system level [77], such as the ESL.

At the ESL, the IC is developed as a SystemC-based *Virtual Prototype* (VP). A VP can be described summarily as a model of the design that can be simulated and "run" (in software fashion). The VP is an example of what modern rapid prototyping means. With a VP, an initial design can be tested as if it were ready for deployment, through the simulation kernel provided by SystemC, a C++ based framework that mimics the intrinsic non sequential and concurrent nature of an IC.

For the VP to meet a power budget, it has been shown that the setting of the architecture of the VP has to be done as early as possible in the initial design phases [78]. These early phase decisions that constitute the nucleus of proper power-aware DSE are, however, limited by the nature of the ESL technical language. The limitations comes in the form of the traditional workflow at the ESL being almost exclusively about the functionality of a design, tuned in and geared towards making it possible for the designer to rapidly implement and test said functionality. As such, there is no room at the traditional ESL workflow for power concerns.

It is not yet possible to take any VP in SystemC and, through a process of *Design understanding* (DU), be able to understand the impact of the already existing functionality based architecture of the VP has on the power related issues (power consumption, area overhead, complexity of a suitable Power Management Unit, etc.). It is nonetheless possible to compare and estimate the impact of existing functionally validate VP when the associated *Power Management Strategy* (PMS) has been decided beforehand [79] [79]. The latter is possible due to the "profiling ready" semantics of the SystemC framework.

Regrettably, due to the non univocal semantics of SystemC full power-aware understanding of the VP written in it is inhibited. Notwithstanding that limitation, a SystemC specification can still be parsed not only for functionality, but also for information about how it consumes power (something that is akin to a power profile). The intrinsic power profile is nothing but the way power is consumed through the operation flows of the design, as per its activity profile. In this activity profile, different functional blocks are active or inactive at various stages of execution and they connect to each other in a variety of forms, leading to a distinct imprint on the power concerns and to the need for a specific PMS.

In order to aid the understanding of the difficulties in parsing a SystemC description for power related information, please consider a portion of the SystemC specification for the Pipe example of the SystemC Accellera suite [44] like the one in Fig. 5.1:

```
struct stage1 : sc_module {
//...
void stage1::addsub() {
 double a, b;
 a = in1.read();
 b = in2.read();
 sum.write(a+b);
 diff.write(a-b); }
struct stage2 : sc_module {
//...
struct stage3 : sc_module {
//...
void stage3::power() {
 double a, b, c;
 a = prod.read();
 b = quot.read();
 c = (a>0 && b>0)? pow(a, b) : 0.0;
 powr.write(c); }
struct numgen : sc_module {
//...
void numgen::generate() {
 static double a = 134.56;
 static double b = 98.24;
 a -= 1.5;
 b -= 2.8;
 out1.write(a);
 out2.write(b); }
//...
```

Fig. 5.1: Part of Pipe Design Implemented in SystemC

The Pipe example is a simple pipelined SystemC VP in which successive algebraic operations (addition, subtraction, division, exponentiation) are applied to a self generated input. It contains four SystemC modules (`numgen`, `stage1`, `stage2`, `stage3`). (`stage1`, `stage2` and `stage3`) compute the algebraic operations on the input generated by the former `numgen`. This very simple structure already provides any designer acquainted with SystemC with information about the activity profile. By looking at the *write* it is possible to see the sequential nature of the design, as well as knowing which modules communicate with each other. For a graphic view of the pipe architecture, please see Fig. 5.2.

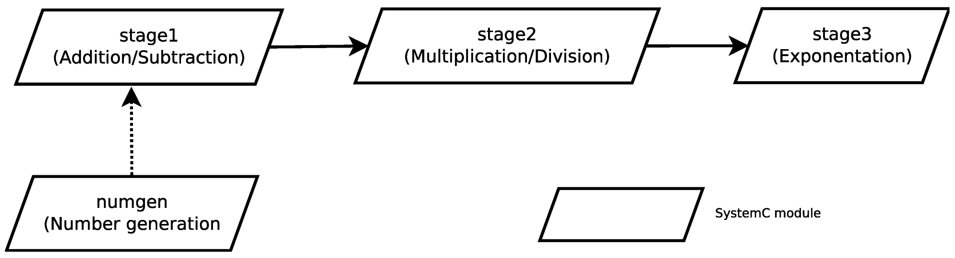


Fig. 5.2: Architecture of the Pipe Example

```
Seq-0: [sc_main, NULL, 0ns]
Seq-1: [numgen::generate, 0x7fffd0, 0ns]
Seq-2: [stage1::addsub, 0x7fffd10', 0ns]
...
Seq-4: [stage3::power, '0x7fffd62, 1000ns]
...
```

Fig. 5.3: Part the Pipe's Activity Profile

The activity profile of the Pipe example can be thought of as a series of sequences of communication between the modules, as obtained by doing a simulation of the execution run of the VP. A activity profile, therefore, has a structure and look as the one depicted in Fig. 5.3. The way to read this activity profile is as follows:

Each sequence shows first the module's instance that is activated during execution run followed by name of module and function, the instance number of the module and the simulation time stamp at which point it is active. As it can be seen, on time 0 several sequences fire concurrently.

The analysis of the activity profile leads to the need to analyze for the most suitable PMS to manage said activity in order to reduce power consumption. For the PMS to be realized, the setting of *Power Domains* (PD), *Power Modes* (PM) and *Control Signals* (CS) and has to be done accordingly. The way to do this is to appropriately parse the activity profile. In order to do this, it is important to understand the analytical method by which the activity profile is parsed. A version in pseudocode of the algorithm with comments (for the extraction of PM and CS) is presented in Algorithm 1.

A perhaps more intuitive way to understand how this is performed benefits from presenting the activity profile in a different, alternative form.

Consider the way the activity profile for the Pipe example is depicted in Fig. 5.4

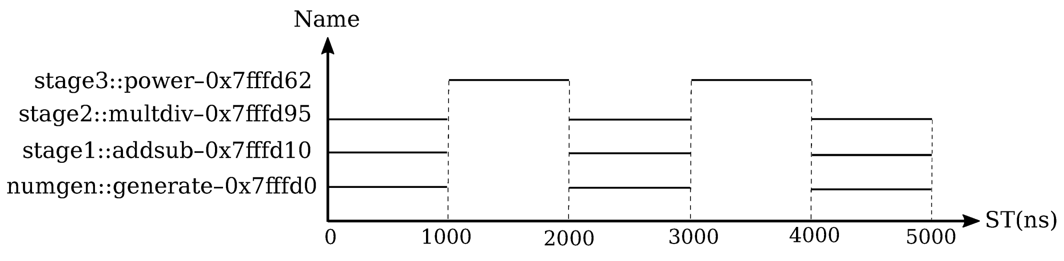


Fig. 5.4: Alternative Form of the Activity Profile of the Pipe Example

In Fig. 5.4, the X axis show the execution time (simulation time stamps) and the Y axis shows the the name active module (including function's name and its module's root and instance number), respectively. This activity profile should be parsed in the

Algorithm 1 Algorithm for PM and CS Extraction

```
1: procedure PM EXTRACTION
2:     ▷ l(F)=List of functions
3:     ▷ l(P)=List of Timestamp Patterns
4:     ▷ l(UP-PM)=List of Unique Patterns-PM
5:     for all $T$ do:                                        ▷ For all timestamps
6:         $l(P)_T \leftarrow l(F) == 1$    ▷ Create a pattern for each timestamp out of the active functions
7:     end for
8:     if $(P)_T$ is unique then
9:         $l(UP-PM) \leftarrow l(P)_T$    ▷ Only add unique timestamp patterns to the list of Unique Patterns-PM
10:    end if
11:    $PM \leftarrow int(l(UP-PM))$                          ▷ Take the integer count for the list
12: end procedure
13: procedure CS EXTRACTION
14:    ▷ l(A)=List of active timestamps
15:    ▷ l(F)=List of functions
16:    ▷ l(UP-CS)=List of Unique Patterns-CS
17:    for all $F$ do:                                        ▷ For all functions
18:        $l(A)_f \leftarrow l(F)$==1 Create a list of patterns for which each function is active
19:    end for
20:    if $l(A)_f$ is unique then
21:        $l(UP-CS) \leftarrow l(A)_f$    ▷ Only add unique patterns to the list of Unique Patterns-CS
22:    end if
23:    $CS \leftarrow int(l(UP-CS))$                          ▷ Take the integer count for the list
24: end procedure
```

following manner for all module instances. "The power function of module stage3 with instance number 0x7fffd62 is only activated at the period of [1000, 2000] and [3000, 4000] during the execution".

The information conveyed by the reading of the alternative form on the activity profile leads to a series of rules of thumb for a graphical approximation to the setting of Power Domains, Control Signals and Power Modes. In essence, these rules can be summarized in reverse fashion as:

- The PM can be obtained through identifying the unique time based patterns in the activity profile.
- The CS can be obtained through identifying the number of coinciding. modules per identified PM.
- The PD can be obtained by assuming at least one PD per CS.

Looking back at Fig. 5.4:

For the PM: Modules *stage1*, *stage2* and *numgen* share the same active/inactive periods. leading to them being in PM1, while *Stage3* has different activity periods, starting from simulation time 1000, leading to a second PM: PM2.

For the CS: *stage2*, *stage1* and *numgen* are active for PM1, but not for PM2. Consequently, the three modules can be controlled by a single CS: CS1. *Stage 3* is active for PM1 and inactive for PM2, leading to a different CS: CS 2.

For the PD: CS 1 and CS2 can each control a Power Domain. Hence, two PD (PD1 and PD2) are required for the PMS.

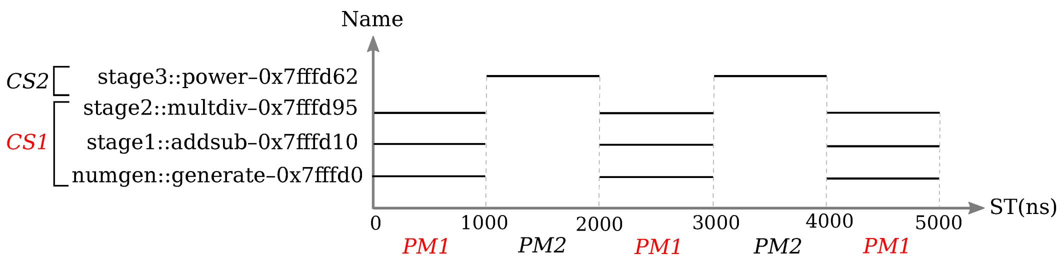


Fig. 5.5: Control Signals and Power Modes for the Pipe Example

A graphical way to see how this is depicted in Fig. 5.5.

PM1 is reached when CS1 is on and CS2 is off. This is the case from timestamp 0 to 1000 and at intervals of 1000 units of time thereafter. PM2 (when CS is off and CS2 is on) happens from timestamp 2000 to 3000 and at intervals of 1000 units of time thereafter, the PM is again.

It is important to noticed that the *sc_main* function is not considered a true design module, as it is a SystemC construct that will not be synthesized,

The above graphically based explanation of the parsing of a SystemC VP for power related information to set PD, CS and PM is obviously not scalable and presupposes that an activity profile can be derived from the VP. How this is done is not the focus of this chapter, but it is nonetheless explained in short form.

The method used to obtain the activity profile consists of using a DU tool. This type of tool parses SystemC code seeking to understand not only its structure but also the behavior coded in it. While there are various approaching to SystemC parsing, a very useful one is to use debug symbols as the building stone to analyze the run-time behavior of the VP

Let the DU tool have two input blocks: the (SystemC VP) and its (Debug Symbols). GDB is a tool that allows access to debug symbols. Hence, besides (GDB) the DU tool requires a control file (GDB Script) to produce two outputs (Run-time Log, Activity Profile). Following the lead of the DU approach from Goli et al.[80], GDB is set to only retrieve and log the functions' activity of all modules' instances. The debug symbols offer access to static information, which includes the modules' name and their corresponding member functions and methods. This structural information is used by the DU tool, through the GDB script to to trace its functions' activities at run-time.

The tracing mechanism is based on the setting of breakpoints at the beginning of each module's function in the GDB script file. Once a function of a module is fired the breakpoint stops the execution and the state of execution (information about the parent module of the faction and its instance) is logged in the Run- time Log file.

Resuming execution and as the next breakpoints are hit, the Run-time Log file is generated, and the VP is translated into a structured model of the VP, with the activity profile being the way the activities of the modules' functions are presented with regards to the simulation time. This can be seen in succinct form in Fig. 5.6.

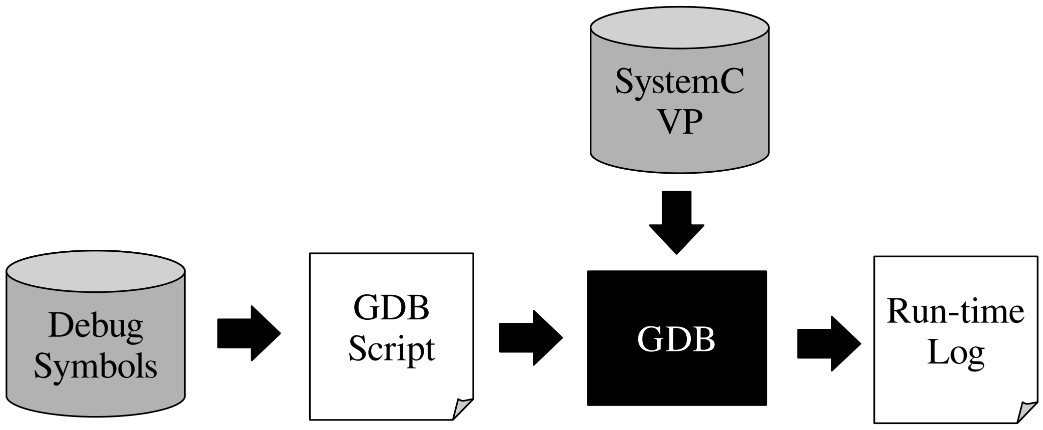


Fig. 5.6: The Design Understanding Approach

DU tools (such as the one highlighted here) used in the context of DSE are represented in several relevant works which are mentioned in the following section.

5.2 Relevant Work

From the beginning of the 21st century, there have been constant advances in ESL power modeling and estimation to provide designers with ways to evaluate alternative VP, such as the work of Mbarek et al. [81] [82]. Other works following the idea to introduce power-awareness at the ESL have lead to power estimation tools such as Powersim [72] [83].

Furthermore, for the verification side of the Power Management Strategy, Hazra et al. have contributed extensively [84]. Similar acknowledgments go towards the work of Affes et al., which has concentrated on a systematic approach to very high level SoC behavior for the modeling of an appropriate *Power Management Strategy* (PMS) [85] [86].

It can be stated that for the setting of the parameters of a *Power Management Unit* (PMU) (PD, CS and PM), the main relevant works can be sorted into two categories: High Level Synthesis (HLS) and Design Space Exploration (DSE) oriented approaches.

HLS approaches are based on the premise of being able to offer the designer a way to synthesize (physically implement) the PMS from the ESL [87, 88, 89] . Usually this is enabled by the extension of the SystemC constructs to incorporate power concepts described in *Unified Power Format* (UPF) descriptions. DU is not the goal of these approaches, which are traditionally dependent on manual and large programming efforts.

DSE approaches are characterized by allowing for the evaluation of different versions of VP and other abstract designs (even UML based designs) to be able to generate PMUs. The focus lies on the synthesizable nature of the PMU and how it can be optimized, once the modeling has been done manually, like is the case of the above cited works by Affes et al..

Other relevant works more related to using DU tools and approaches using the concept of PD, CS and PM are those of: Wang et al. [37], who evaluate alternative PD partitioning schemes for PMU designs that were obtained through an Evolutionary Algorithm being applied to the analysis of a SoC; and Macko [90], which introduced automated analysis of the activity profile of a VP. These two works are, unfortunately, resting on non automated and intrusive ways of dealing with the system-level specification design themselves so as to provide a way to properly manage the PD, CS and PM values.

The response framework proposed in this work pursues similar ideas to those behind the work of Wang et al. and Macko, following the approach of extracting information and analyzing the activity profile of a VP in order to set the PD, CS and PM. Unlike those two works, however, *the research contribution presented by the response framework focuses on the algorithm for the system-level automated non-intrusive extraction of PM and CS (and PD) from the activity profile, which leads to understanding the Power Management Strategy.*

5.3 Response Framework

One need that has been permeating the ESL world has been that of supporting highly abstract timing information. That is, having execution time from the running of a VP be independent of the physical timing of the implementation of the VP in hardware. In order to allow this, *Transaction Level Modeling* (TLM) comes to the forefront. TLM allows timing in VP to be abstract, which favors rapid prototyping, but significantly hampers the possibility of linking an activity profile to actual expected timing behavior in a hardware implementation.

The response framework addresses the difficulty mentioned above by supporting TLM constructs. Additionally, it allows for a basic generated PMU with the PD, PM and CS that can be set via the analysis of the VP, although this basic PMU is not the main goal of the response framework. A diagram of the framework can be seen in Fig. 5.7

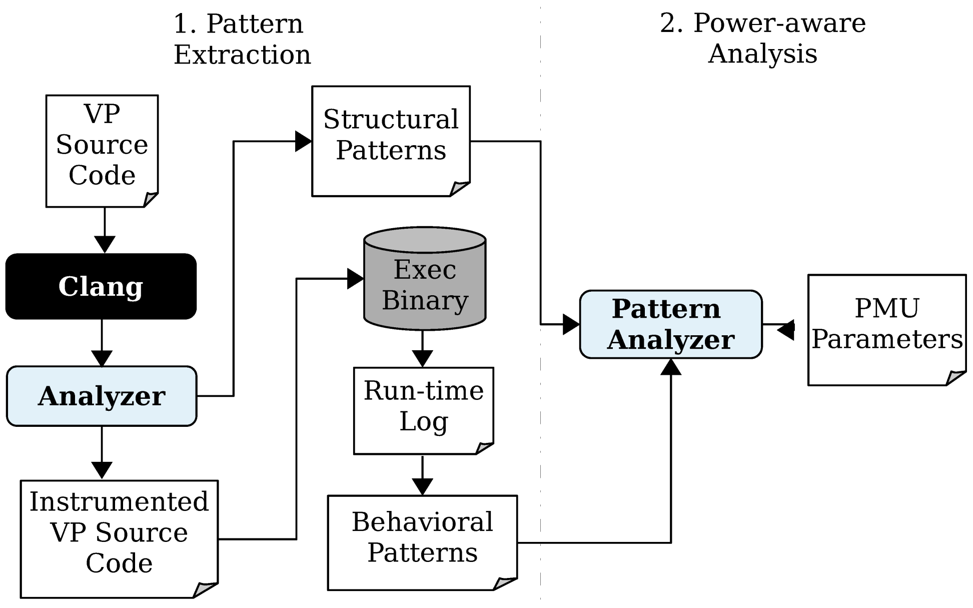


Fig. 5.7: Overview of the Response Framework

5.3.1 The First Phase: Pattern Extraction

The first phase of the response framework is to access the VP's structural information (including the name of TLM modules, their sockets, signals and member functions) and also the behavior at run-time (dynamic information) which is defined in terms of transactions for TLM based designs. The response framework uses a Clang [91]

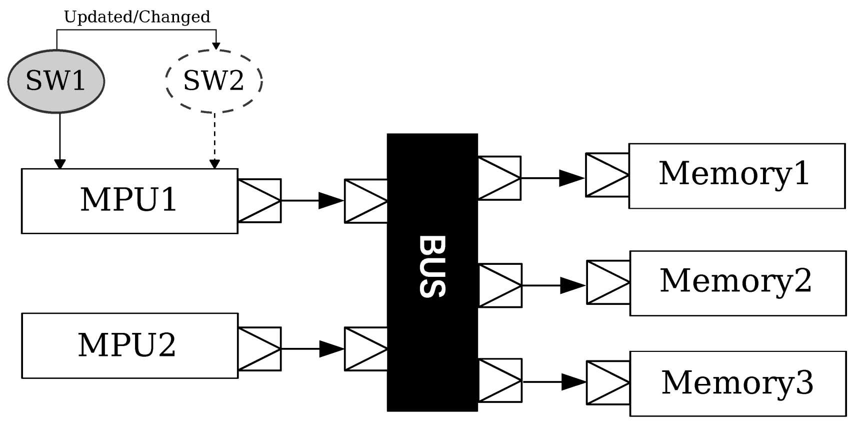


Fig. 5.8: The Architecture of the Motivating Example

compiler, following the lead of the work by Goli et al.[92], but can also use the GDB based approach by the same authors [80] with the restructuring of its code. The structural information accessed is extracted as the corresponding node in the *Abstract Syntax Tree* (AST) of the VP which is generated by Clang from its source code and constitute the *Structural Patterns*.

To access the run-time information, the response framework utilizes an instrumented version of the VP source code which generated by the Analyzer module. The instrumentation of the VP's source code consists of the insertion of retrieving statements in the original VP source code. The retrieving statements include the instructions that properly trace the transactions. In order to properly trace transactions, the reference address, the value of the transaction's attributes and related timing annotation are required. The retrieving statements are inserted in two locations within the instrumented source code: the line of code where the transaction is defined (e.g., as a function arguments), and the function call (e.g., transport interface) where the transaction object is used as an input argument.

By the execution of the instrumented source code via the executable binary of the VP, the behavior of said VP is stored in the Run-time Log file. Once this is done, further analysis is performed on the Run-time Log file to derive the activity profile, which constitute the *Behavioral Patterns* of the VP.

So as to make this phase more understandable to the reader, let there be a motivating example, the case of the *LT_BUS* VP in Fig. 5.8 implemented using SystemC TLM-2.0 (inspired by [93]). The VP consists of two microprocessor units (*MPU1* and *MPU2*), an interconnect *LT_BUS* and three memory modules (*Memory1* to *Memory3*). The *MPU1* and *MPU2* modules serve as initiator, the interconnect module *BUS* works as a router while *Memory1* to *Memory3* are the target modules. Access to *Memory1*

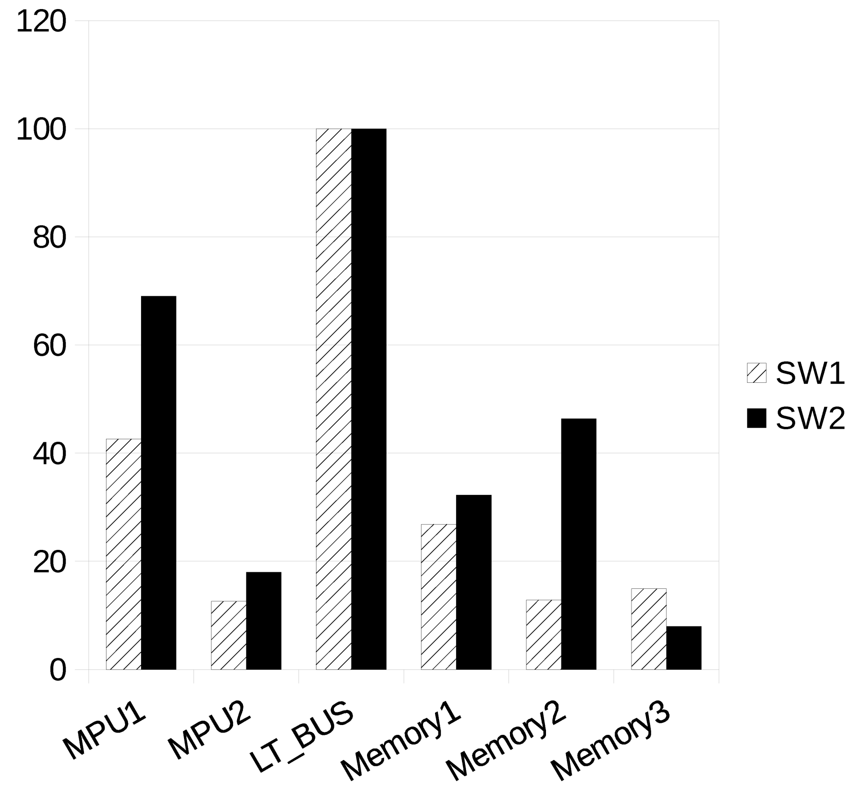


Fig. 5.9: Activity Percentage of the LT_BUS VP modules for Different Workloads

and *Memory3* is allowed for *MPU1* which runs a software named *SW1* that, in turn, generates a set of transactions. *MPU2*, unlike *MPU1* module accesses only *Memory2* (the software run by this MPU does not matter for the comprehension of the use case and is therefore disregarded). The above case has an architecture that can benefit from a power-aware analysis leading to a Power Management Unit (PMU) to manage its power consumption. This PMU needs to take into account the activity patterns of the VP for its initiators, interconnect and target modules. The gray bar (*SW1*) in Fig. 5.9 shows the activity percentage of the VP modules when the *SW1* is running on the *MPU1*.

As a VP, it is expected that the prototype will suffer changes throughout the design phase to accommodate refinements and even bigger transformations of the architecture. However, the most frequent change experienced by a VP is the change in the software that it runs. Therefore, a typical scenario at the ESL is that the piece of software that runs on some modules of the VP is modified, which constitutes a *behavioral change*. When there is a transformation (refinement) of the architecture there is a *structural change*.

Structural changes are much less frequent than their behavioral counterparts, since they are not only costly in terms of design time, but also fundamentally alter the architecture of a VP. That is why, for most intents and purposes, once an architecture has been selected for the VP a design decision, this architecture is very likely to remain constant.

Behavioral changes are the type of changes that routinely emanate from the software than can run on a given module of the VP (such as a processor). This type of changes are expected, because software updates need to be accommodated even up to the point of reaching the synthesizing stages. Consequently, it becomes important to address these changes or updates during the design process or even after the synthesize process, from the perspective of the ESL VP prototype.

As an example of changing/modifying the software part, consider the scenario that the *SW1* running on the *MPU1* is modified to *SW2* to increase the overall performance of the VP. While *SW1* and *SW2* may not be substantially different, any software change affects the behavior of the *MPU1*. This change becomes evident as the number of generated transactions, time of activation and access to the memories are not the same for *SW1* and *SW2*. It is because of the alterations made to the behavioral patterns of the VP that there exists a noticeable impact on the most appropriate Power Management Strategy (PMS) to be implemented by the PMU of the VP.

The changes differentiating *SW1* and *SW2* can be seen in summarized form in Fig. 5.9. In it, the black bar (*SW2*) in Fig. 5.9 shows the activity percentage of the VP modules based on the new behavioral pattern of the VP, while the other bar shows the equivalent activity percentage for *SW1* the running software of the *MPU1* module. Since the bar pattern is different, the parameters related to the realization of the PMS differ from *SW1* to *SW2* and this reality means that there will be a need for an update on them.

Let the *MPU1* module be considered as the object under analysis. As a TLM based module from the VP, there is a need to trace all transactions related to it, as well as other functions (e.g.*thread_process*) in which a transaction object is referenced. As previously stated, this is done through extracting the VP's AST using the *Analyzer* module of Fig. 5.7. Essentially, the introduction of a *retrieving* statement into the source code is required. So as to show how this in a clearer manner, please look at Fig. 5.10.

In line 5 of Fig. 5.10, the transaction object *trans* is used as a function argument of the *b_transport* interface. To trace the transaction and consequently the activity of *MPU1*, the *retrieving* statement *Fout*, which is present in line 6, is automatically generated and inserted after the function call in the instrumented source code.

The main purpose of the instrumented source code is to populate the *Run-time Log* via the *Executable Binary* in order to provide the information required for the extraction of the *Behavioral Patterns*. It is those behaviors that condense most of the changes or updates to the patterns related to the PMS. In Fig. 5.11, it is possible to see some of the transactions that characterize the Behavioral Patterns as sequences (SQ).

From inspecting SQ1 from Fig. 5.11 it is possible to say that module *MPU1* does have a sequence that writes data in module *Memory1* through the *LT_BUS* module. The sequence shows the delay and simulation time for each phase of transaction that underlines the sequence, as well as the complete the transaction which is *20 ns*. For a graphical depiction of the same information which can aid in further understanding, please look at Fig. 5.12.

The way in which both the *Behavioral Patterns* (from the Instrumented Source Code) and the *Structural Patterns* (from the AST) are analyzed in the context of a *Power-aware Analysis* constitutes the main part of the second phase of the framework, which is explained in the next subsection.

5.3.2 The Second Phase: Power-aware Analysis

The second phase of the response frameworks uses a *Pattern Analyzer* based on the algorithms described in the first section of this chapter to yield a set of PMU parameters (PM and CS). This phase basically implements the same type of algorithms described in Algorithm 1, only adapted to trace the transactions that have timing information that can be used instead of the timestamps of the original algorithm. In this phase, both *Behavioral Patterns* and Structural Patterns are the information

```
1  struct MPU1: sc_module {
2   tlm_utils::simple_target_socket<MPU1,32> socket;
3   void thread_process(){
4   ...
5   socket->b_transport(*trans, delay);
6   Fout<<‘‘ MPU1:thread_process:ID = ’’<<trans<<‘‘ Cmd = "<< trans->
       get_command ()<<‘‘ address = ’’ << hex << trans->get_address
       () <<‘‘ at time ’’ << sc_time_stamp()<<‘‘ delay = ’’ << delay
        << endl;
7   ...}
```

Fig. 5.10: Part of the Instrumented Source Code of the VP

```
        SQ1: ([MPU1, thread_process, 20ns, 0x6631b0],[0x006, W, 5ns])
        SQ2: ([LT_BUS, b_transport, 25ns, 0x6631b0],[0x003, W, 5ns])
        SQ3: ([Memory1, b_transport, 30ns, 0x6631b0],[0x003, W, 5ns])
        SQ4: ([LT_BUS, b_transport, 35ns, 0x6631b0],[0x003, W, 5ns])
        SQ5: ([MPU1, thread_process, 40ns, 0x6631b0],[0x006, W, 5ns])
```

Fig. 5.11: A Part of the Run-time Log of the VP

sources from which the Power Domain partitioning scheme and setting of *Control Signals* (CS) and *Power Modes* (PM) will be derived.

The structural patterns are analyzed to understand the main functional blocks of the VP, thereby revealing its structure, via properties of the described modules of the VP, such as their name, instance, number of sockets and class (initiator, target, interconnect). The importance of the aforementioned properties is related to the fact that a functional block representing a Power Domain, requires at least **1 CS** to control the internal power states, which means a CS is to be inserted in each Power Domain. Assuming this sensible rule, *the number of CS is to be considered equal to the number of Power Domains*.

The Behavioral patterns are analyzed to gather the activity of each module of the VP. The most important factor is the activity of each module at each time unit (a period between two simulation time steps). The activity, which is revealed by the sequences in the Run-time Log previously shown, lead the way to extracting the different Power Modes of the VP, as they are characterized by the internal power states (ON/OFF states, for simplicity) of each module at a given time.

It must be remembered that the Power-aware Analysis phase of the response framework has to adapt to the changing nature of the VP. As has been pointed out before, the changing nature of the VP is mainly seen through updates (changes) in the software than runs in some of the modules (the processing units), but also in whatever architectural change that the designers may wish to entertain as the VP is refined. It is then a requirement for this second phase to allow for the parameters that define the PMS to be consequently alterable as per the changes in the VP. In essence, the

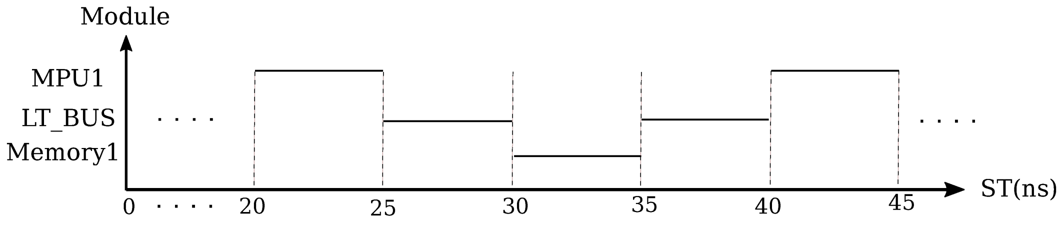


Fig. 5.12: A Part of the Activity Pattern of the LT_BUS VP Modules

PMU that the framework can output needs to have a degree of programmability, not unlike that of a regular module of the VP.

As a result of the demand for the generated PMU (stemming mainly from forced changes in the PM and CS), the analysis in these phase needs to consider the Power Management Strategy to be "run" in an internal read-only memory. The rationale is as follows: the PMU must be able to read from the memory all the setting of PM and CS that can be derived from the changes within the VP (for simplicity, these changes will be considered only as software changes for the Main Processing Unit). That is to say, the Power-aware Analysis phase has to provide the PMU with information encoded a read-only memory for each new iteration of the changing software run in the modules.

The information within the read only memory is meant to identify the modules for whom the PMU has to issue an activation/deactivation CS and the time unit for that CS (the duration of the ON/OFF state of the issued activation/deactivation). The read only memory is then the receptacle of the PMS and its encoded properties are dependent on the number of time-units (extracted by analyzing the Behavioral Pattern) and the number of bits to code the sequences defining the Behavioral Patterns in each time unit (a construct called "pattern data"), among other values. The pattern data construct consists of two main parts which are 1) the amount of bits that will encode the maximum difference between two time-units and, 2) the amount of bits required to code each module of the design with an unique ID. By storing the pattern data and the Control Signals to drive the modules in the read-only memory at the beginning of each iteration in the design of the VP, the PMS can be updated according to the changes reflected in every iteration.

While the generation of a basic PMU (a System/TLM module to implement as an added module of the VP) is not the main goal of this second phase, it is nonetheless revealing to explain how it can come into existence. By following the structure of the read only memory for the PMS, with the "pattern data" as what is encoded and stored for the implementation of the PMS, it is possible to generate a basic PMU, which will integrate 3 key elements:

- Output signals (defined by the number of modules in the VP, to act as Control Signals).
- A internal process (defining the behavior of the PMU itself).
- A set of functions (to decode the pattern data stored in memory so as to implement the PMS).

```
 1  #define patternDataSize 6
 2  #define moduleNum 6
 3  #define memSize 1024
 4  /*...*/
 5  struct PMU: sc_module{
 6   sc_out<bool> CS[moduleNum];
 7   sc_bv<patternDataSize> memory[memSize];
 8
 9   void process ();
10   sc_bv<moduleNum> decode_CS (sc_bv<patternDataSize>);
11   int decode_time (sc_bv<patternDataSize>);
12   /*...*/
13   SC_CTOR(PMU){
14    SC_THREAD (process);}
15  };
16  /*...*/
17  void PMU::process () {
18   int memAddress = 0; //starting point
19   int delay; //last 10 bits
20   sc_bv<moduleNum> CStemp;
21   /*...*/
22   while(1) {
23    if (memAddress < memSize){
24     delay = decode_time (memory[memAddress]);
25     CStemp = decode_CS (memory[memAddress]);
26     for (int i = 0; i < moduleNum; ++i)
27     CS[i].write(CStemp[i]);
28     wait(delay);
29     memAddress++;}
30    /*...*/
31   }
```

Fig. 5.13: Part of a Basic PMU for the LT_BUS VP

For a clearer representation of what the 3 elements look like within a PMU, please look at Fig. 5.13.

Fig. 5.13 represents part of a generated PMU for the *LT_BUS* VP that has been the running use case. From a rapid inspection of the code shown in Fig. 5.13, it is possible to identify that the PMU is built as a module with output signals, with a thread called process that explains its behavior and instructions that correspond to the functions that decode the pattern data.

As an explanation of the basic PMU, it can be said, that lines 1 to 3, state the parameters *patternDataSize*, *moduleNum* and *memSize*, which depict the size of each memory line that keeps a pattern data, the number of modules in the VP and the size of the memory that includes all pattern data, respectively.

The rest of the code within the definition of the PMU module refers to information provided by the analysis of both the Structural and the Behavioral Patterns. From the latter, it is possible to obtain the maximum difference between two time-units, a value which is 5 ns (which can also be noticed by analyzing Fig. 5.12). This maximum difference leads to three bits being defined to properly encode the difference between two time units. Moreover, three extra bits are used to code the modules of the VP (generating a unique ID for each module), which is a value that is obtained from the analysis of the Structural Pattern. As a consequence, the parameter *patternDataSize* in the code is set to six.

Going through the *process* thread (lines 17 to 29), it is possible to observe how it defines the way the PMU is to read each line of memory, decoding the pattern data to obtain: the duration of modules' activation (represented by the value of *delay* (line 24)) and the modules that must be activated and deactivated in the current time unit (represented by *CStemp* (line 25)). Once the decoding of the pattern data is finished, the values stored in *CStemp* are assigned to the corresponding output signals to and these are transmitted to the appropriate other modules of the VP, which receive them as Control Signals. These output signals are meant to lock the values of *CStemp* until the next address of memory (the next line) is read for the same process to be continued until the end of the execution (which in this case means the simulation of the VP).

5.3.3 Limitations of the Response Framework

The description of the phases of the response framework has included caveats. The caveats are mainly related to specific presuppositions and simplifications that do not invalidate the efficacy of the framework, but put into perspective that the framework should operate under a set of conditions. The limitations pointed to by the caveats represent improvement areas and can be grouped into 3 categories of assumptions:

1. Analytical assumptions: these are presuppositions that are related to simplifications. For instance, the number of PD is considered equal to the number of CS in an effort to curb the need to develop a specific algorithm for the extraction of PD. The supposition that a CS only enables two Power States (ON/OFF) is also an example of this type of presuppositions. The aforementioned are logically constructed simplifications that are meant to reduce the complexity of the analysis.

2. Input assumptions: these are limitations related to presuppositions regarding the VP being analyzed so as to make sure the framework can properly operate.

For instance, the VP is expected to be written in a way that implements proper timing constructs (either Cycle-Accurate or Approximate or Loosely Timed-in TLM) because these ensure that the activity profile can be retrieved and then analyzed. Non standard or untimed constructs are fortunately atypical, so this kind of assumption is not as relevant as a limitation.

3. Technical limitation assumptions: these are limitations related to certain constraints posed by the tools used in the framework. For instance, in the running case described in this chapter, there exists a constraint resting on the assumption that the softwares (*SW1* and *SW2*) run on the processing unit are statically defined. Statically defined here means that these softwares cannot themselves contain variable dynamic input, which means they have to be self contained pieces of code capable of being run without any further intervention. This limitation is caused by the way the activity profile is constructed, which rests on the ability to trace the transactions through the instrumented code by considering them immutable in number. This limitation could be thought as troublesome for the universal appeal of the response framework, but it is important to remember that ASIC designs tend to have extremely limited variable dynamic input use cases. Since the main reason for the design of ASIC is to conduct properly defined and fixed (static) computing or communication tasks, dynamic inputs are rare.

The limitations of the response framework can be grouped under the categories described above, with the categories representing potential areas of improvement. These limitations signal that the response framework needs certain conditions in order to remain efficacious, with said conditions not preventing the framework from serving its intended purpose.

5.4 Concluding Remarks

Specifications in a system-level technical language such as SystemC can be parsed and analyzed in order not only to conduct power-aware Design Understanding, but also to unveil the power architecture of the design and thus extract the possible Power Modes and Control Signals that, together with the Power Domain partitioning scheme, lead to an appropriate Power Management Strategy. The main goal of the parsing and analysis of natural language specifications is to be able to serve as the conduit for proper Design Space Exploration (DSE).

Based on a Design Understanding tool, a response framework has been devised to not only perform power-aware Design Understanding in an automated non intrusive manner, but also to be able to conduct power-aware analysis in a way that adapts to changes in the design under such analysis. The response framework can analyzed pure SystemC designs as well as Systemc/TLM designs (whose variable nature as Virtual Prototypes requires adaptability). The response framework is shown to consist of two phases: one initial phase that obtains the Structural and Behavioral Patterns of a design, through a modified Design Understanding tool, and a second phase which analyzes the patterns with a specialized algorithm so as to be able to extract the Power Modes, Controls Signals and Power Domains to condense into a Power Management Strategy as integral part of the DSE process.

6 Application of the Response Framework

6.1 ASIC Designs from Technical Language

When the designers of an ASIC think of a specification that they can rapidly use to build a working prototype, they think of one in a technical language. This is typically the case when the requirements for the ASIC have been made clear and there is no need to consult a technical standard or any other natural language specification. Typically, this scenario is seen throughout the industry when a certain ASIC needs to be developed and there are previous prototypes available or when the design of the ASIC lends itself to easy and rapid prototyping.

As an example of the above, consider a Hamming encoder. Once again, an encoder is a typical element of a design susceptible to be considered ASIC material. Hamming encoders need to comply with the encoding algorithm they use to provide valid output. Said output is a new way of presenting the input according to the encoding rules. This "simplicity" leads to designers rapidly prototyping an encoder, producing or obtaining a *Virtual Prototype* (VP) in SystemC or in a similar language.

The fact that at the *Electronic System Level* (ESL), the VP can be easily grouped and simulated with other components in a *System on Chip* (SoC) makes an Encoder VP an interesting use case for the response framework. Alternative VP for the same encoder can be compared against each other by using the response framework in a *Design Space Exploration* (DSE) fashion, eventually yielding the parameters for the respective *Power Management Unit* (PMU) for those alternative Encoder VP. In this use case (and for similar designs and circuits), the Design Understanding (DU) aspect of the response framework is especially useful since it supports system-level power-aware DSE. This same aspect can be applied to other similar designs acting as benchmarks for the response framework.

As was shown in the previous chapter, the response framework yields a set of *Power Domains* (PD), *Control Signals* (CS) and *Power Modes* (PM), which is a fine way to assist designers in the power-aware design of competing designs, showing which alternative VP may present the most complex *Power Management Strategy* (PMS) leading to an equivalently complex PMU. A complex PMU could be the one with a highest area overhead from the required extra logic, but one the one that reduces the overall power consumption the most. A complex PMU could also be complex to implement, because it requires a more detailed and more robust module to be devised and more complex functions related to management of the PM and CS. Decisions as to which VP is "better" based on their respective appropriate PMS and PMU is made possible thanks to the both phases of the response framework.

To show how the response framework is capable of handling more complex VP with more components, it becomes important to test it in other situations besides the comparison of Encoders. This can be done through the use of the response framework in VP implemented following *Transaction Level Modeling* constructs. The timing information in said type of VP make it harder for power-aware frameworks to adequately set the PD, CS and PM. The proposed response framework does, however, extracts the PD, CS and PM, that is to say, it provides an appropriate PMS.

A set of benchmarks consisting of varied designs (some with TLM constructs) constitute a validating mechanism for the efficacy and reliability of the response framework. Since supporting TLM constructs and the changing nature of VP is an important feature of the response framework, it is reasonable to test how the response framework fares in the context of a far more complex and realistic use case whose design is larger (yet similar) to that taken as running case in Subsection 5.3.2.

For the evaluation of the adaptability of the response framework as advertised in Subsection 5.3.2, the response framework is presented with a LEON3 processor-based VP called SoCRocket [94]. This VP in SoC is the only freely available VP with support for power modeling and estimation and is chosen because those features show the effectiveness of the PMS that the response framework can obtain from different workload scenarios run within the LEON3 processor. The size and complexity of the VP are a test to the potential scalability of the framework, as well as to its adaptability to different workload scenarios.

6.2 The Hamming Encoders and Other Benchmarks

Hamming encoders are well known because of the simplicity of their design. They have been prototype in a variety of technical languages at various abstraction levels: from UML dialects [95] to Verilog at the RTL [96]. Not surprisingly, prototypes have been made at the ESL in SystemC as well.

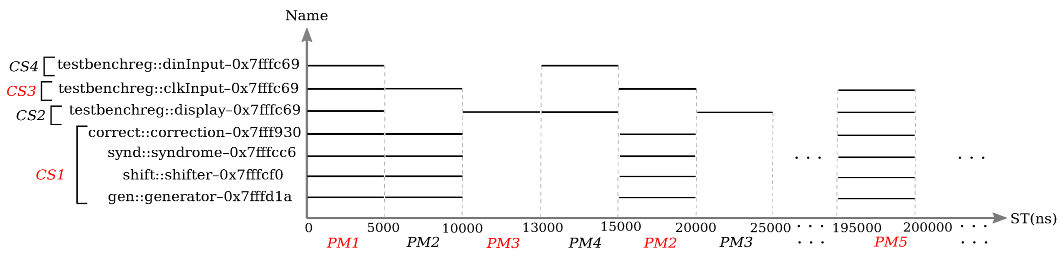


Fig. 6.1: Part of the Activity Profile for Hamming-seq

Consider two alternative Hamming Encoder prototypes (specified in SystemC): one of the encoders [97] is a VP that does the encoding process fully sequentially and thus receives the label *Hamming-seq*, the other is a VP [98] that does the encoding process in a combinational way which is not fully sequential, earning the label *Hamming-comb*.

Both Hamming-seq and Hamming-comb VP provide designers with encoding/decoding for the (15,4) Hamming code algorithm so it is fair to compare them on the basis of which appears to be more power-aware, according to the setting of PD, CS and PM. For this use case, understanding the architecture of the design is the most relevant task of the response framework.

By running the response framework to compare the two alternative VP, it is possible to compare graphically by using a simplified activity profile (as per the previous chapter), in which CS and PM can be easily noticed. In the activity profile, the Structural and Behavioral Patterns become the salient elements from which both VP can be analyzed. The Power-aware Analysis phase, done by the Pattern Analyzer can be rather straightforward, since the architecture of the VP under analysis is bound to be similar.

In order to unveil the intrinsic power architecture of the alternative VP it is reasonable to assume certain simplifications expressed in Subsection 5.3.3. This means, for instance that for each CS there is one PD, effectively equating the number of PD to that of CS, under the simplification that a PD always requires a CS.

Fig. 6.1, represents part of the activity profile for *Hamming-seq*. It shows that this VP requires **4 CS** and **5 PM** for its 5 SystemC modules (*testbenchreg*, *correct*, *synd*, *shift* and *gen*)

Fig. 6.2, represents part of the activity profile for *Hamming-comb*. It shows that *Hamming-comb* requires **6 CS** and **5 PM** for its 6 SystemC modules (*tbreg*, *HamReg*, *correct*, *Corrector*, *HCGen* and *SynGen*).

Type of Encoder	Number of CS	Number of PM	Number of PD	Number of Modules
Hamming-seq	4	5	4	5
Hamming-comb	6	5	6	6

Tab. 6.1: Power Parameters for the Alternative Hamming Encoders

By inspecting the figures, it can be noticed that the CS (with the functions within the modules under their control) are represented are in the Y axis, while on the Y axis are the timestamps as well as the PM that is active at that particular time frame. A summary of the information above in table form can be seen in Table 6.1.

A difference of 2 CS (**6** for *Hamming-comb* as opposed to the **4** for *Hamming-seq*) may not seem transcendent, but it is important to remember that an extra Control Signal means extra control logic and, according to the simplifications made in this particular situation, an extra PD. Based on what was uncovered, *Hamming-seq* is likely to be more power-aware and lead to a less complex and less taxing PMU than the alternative *Hamming-comb*.

Comparison of the nature of the one just made are not meant to be definitive. They serve as ways to save designers' time and effort by being automated through the response framework. The associated costs of choosing an inconvenient architecture (for example here, the Hamming-comb one) require early decision making that is enabled by the response framework.

While a Hamming Encoder is a design that is squarely within the ASIC realm, the response framework can tackle the challenge of extracting (or setting) PD, CS and PM for other VP of different characteristics, such as size and coding style.

For a condensed view of the benchmarks consider Table 6.2:

The TLM designs *Example-5*, *Example-6*, *AT-example* and *Locking-two* come from Doulos [93]. The *VGA Controller* example is provided by the S2C Benchmark suite [99]. The *FiR Filter*, *Packet Switch,RISC CPU* and *Simple Bus* designs are available through Accellera [44].

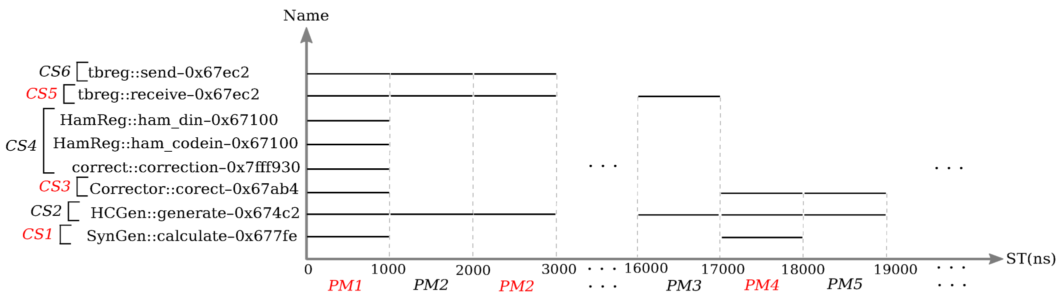


Fig. 6.2: Part of the Activity Profile for Hamming-comb

Type	VP Name	LoC	#CS	#PM
SystemC	FIR Filter	834	3	4
	VGA Controller	856	4	3
	Packet Switch	1020	10	74
	RISC CPU	1960	12	18
	Simple Bus	2100	9	8
TLM-2.0	Example-5	650	21	14
	Example-6	713	36	34
	AT-example	2942	41	29
	Locking-two	3831	42	35

Tab. 6.2: The Response Framework for other Benchmark VP

As can be gleaned from Table 6.2, the SystemC designs tend to be small (under 2500 Lines of Code.-LoC). However, it is important to notice that even in an apparently simple VP, such as a *VGA controller* 3 PM and 3 CS are still required. Typical small CPUs for ancillary use, represented in this table by *RISC CPU* can require many more PM (18) as well as more CS (7). The increase in the size of the VP is responsible for the increase in the number of CS and PM (and by extension, PD), but this is not a linear ratio.

For TLM designs, the number of LoC of the most complex examples is above 2000, but the number of CS and PM is much greater than for the SystemC designs. This highlights how sensitive the algorithms of the response framework are towards the way in which abstract timing plays a role in the setting of CS and PM. The more abstract the timing information, the greater the number of CS and PM based on the fact that many transactions expressed in TLM constructs require at least a Control Signal to ascertain the delivery of the message, whereas the multiple independent global states the design is able to reach requires more Power Modes.

As a conclusion it can be stated that the response framework is capable of handling many typical circuits that are candidate for ASIC design. It allows for early stage power-ware comparison between alternative VP for a some function and successfully works with designs of different domains, from encoder, to graphic controllers, from ancillary CPU to TLM designs of medium complexity.

6.3 A Larger TLM VP Use Case

Since it is important that the response frameworks adequately deals with designs whose Behavioral Patterns are in flux (that is to say, they can change at the system-level) during early stage design, a SoC type of VP that consists of multiple SystemC/TLM modules and the ability to run different workload scenarios is a very natural fit as a test case. The choice for this more complex VP is *SocRocket*[94].

SocRocket contains more than 50,000 lines of code in SystemC/TLM, representing modules working together in *master* or *slave* mode. As natural for most VP of this nature, *SocRocket* is bus-centric, meaning the modules are connected to each other through a bus, which in this case is a *AMBA-2.0 AHB* (Advanced High-performance Bus) bus. The main processing unit of the VP is the LEON3 processor (*leon3_0*), which directly connected with the bus as a AHBMaster device.

Other modules contained within the VP are:

- A memory controller (*mctrl*) connected as AHBSlave to the bus, serving the memories.
- A ROM module (*rom*), which remains almost unused.
- An SDRAM module (*sdram*), which represents one heavily used memory.
- A bus control module (*ahbctrl*).
- Two initiator modules working as TLM initators in master mode (*ahbin1* and *ahbin2*).
- Two memories working as TLM targets in slave mode (*ahbmem1* and *ahbmem2*).

The *LEON3 processor* in this case will run **6** different softwares which represent **4** different workload scenarios. The rationale behind analyzing these differing workload scenarios (stemming from different softwares) is that the power parameters and characteristics, as well as corner cases for each of the workload scenarios vary significantly. It is important to show how adaptable the response framework can be to the variable workload scenarios.

The 4 *workload scenarios* can be summarized and defined as follows:

- **S1:** Light-weight access to one memory and intensive access to other.
- **S2:** Intensive access to all memories.
- **S3:** Light-weight workload on processor (i.e. LEON3 processor).
- **S4:** Intensive workload on processor (i.e. LEON3 processor).

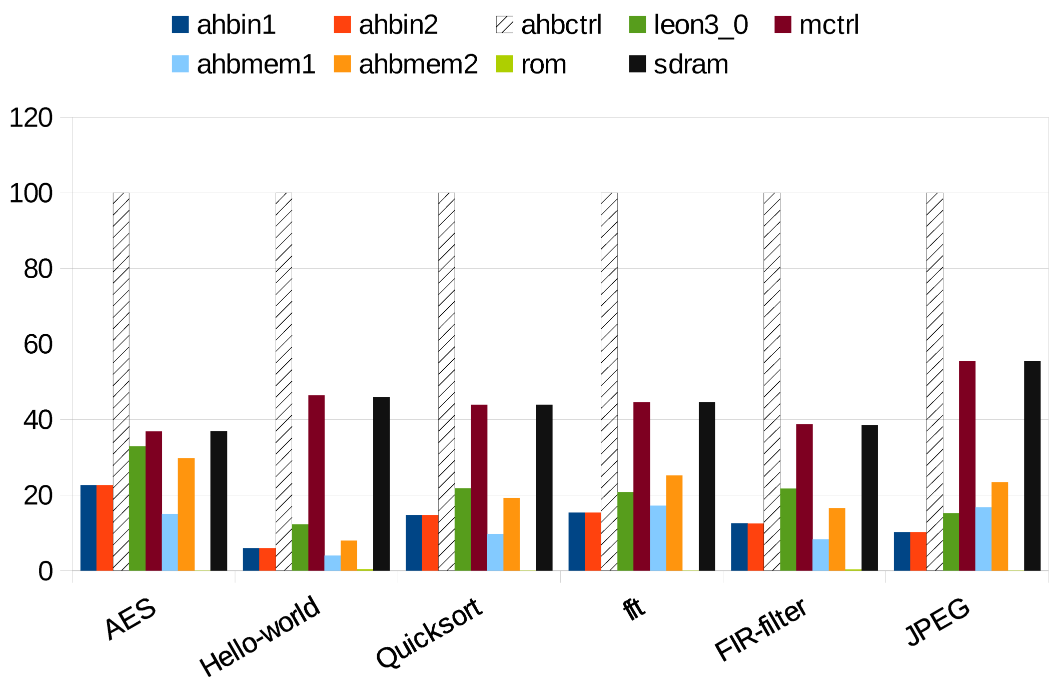


Fig. 6.3: Activity Percentage of the VP modules for the Different Softwares

As *ahbin1* and *ahbin2* are initiator modules, they are used to generate two different sets of transactions for scenarios S1 and S2. Notwithstanding the aformentioned, all four scenarios (S1, S2, S3 and S4) can be represented by the transactions coming from the 6 different softwares (implemented in C within the LEON3 processor):

- AES
- Hello-world
- Quicksort
- fft
- FIR-filter
- JPEG

Scenarios based on changing software are interesting to analyze, since they show how the software makes use of the different modules of the VP. This usage values (the activity percentage, ranging from no use (0) to continuous use (100)) reveal information about the Behavioral Patterns of the VP. This information can be seen in graphical form in Fig. 6.3.

Module	Activity percentage
ahbin1	10.1775684001
ahbin2	10.1739445552
ahbctrl	100.0
leon3_0	15.189345896
mctrl	55.4683819533
ahbmem1	6.73491574561
ahbmem2	13.3973545932
rom	0.0525457510419
sdram	55.4393911941

Tab. 6.3: Activity percentage of the modules for the JPEG software

Module	Activity percentage
ahbin1	22.6298063676
ahbin2	22.6311602178
ahbctrl	100.0
leon3_0	12.8614979318
mctrl	36.858175424
ahbmem1	14.9553866517
ahbmem2	29.742019849
rom	0.00115475464251
sdram	36.8978352731

Tab. 6.4: Activity Percentage of the Modules for the AES Software

So as to better understand the varying impact of the software representing the different scenarios, let us see the activity profiles for the *JPEG* software representing scenarios S2 and S3 in Table 6.3, and for the *AES* software, representing scenarios S2 and S4 in Table 6.4.

From the activity profiles it can be read that the activity percentages for *ahbin1* and *ahbin2* are more than doubled for the *AES* software when compared to the *JPEG* software. A very similar thing happens for *ahbmem1* (6.73 vs 14.95) and *ahbmem2* (13.39 vs 29.74). These ratios can be explained by the fact that the softwares belong to different scenarios. The ratios are relevant not only in that way, but also in the fact that they significantly impact the power consumption of the VP.

The way the impact of the different activity percentages for the different modules is felt according to the scenarios, as related to the power parameters (PM, CS and PD) of the PMU. However, on account of the complexity of the design, the impact can be summarized by the effect of the PMU built upon the power parameters. As the PMU is added to the VP, it will reduce the power consumption of the VP and this serves as validation of the response framework by showcasing how effective the PMS implemented by the PMU can be for a large use case.

The validating and summarizing results of the response framework applied in the context of the SocRocket based VP are listed in Table 6.5.

Scenario	Software	#Lines of Code	#Transactions	Power consumption (uW) without PMU	with PMU	Difference
S1,S3	Hello-world	20	5,143	1026942	659337	-35.79%
S1,S4	FIR-filter	123	7,422	1280771	492147	-61.57%
S2,S3	JPEG	943	43,063	1359092	913069	-31.82%
S2,S4	fft	652	124,701	1866998	1011110	-45.84%
S2,S4	Quicksort (QS)	43	133,310	1824925	965922	-47.07%
S2,S4	AES	490	2,090,740	20822862	7518165	-63.89%

Tab. 6.5: PMU Validation for the Different Scenarios for the SocRocket VP

Table 6.5 is easy to read. Columns *Software* and *Lines of Code* show the name the software on the LEON3 processor and its length in term of lines of code, respectively. The column *#Transaction* states how many transactions are generated by each software. Let it be remembered that the transactions are fundamental in the first phase of the response framework, as the core of the Behavioral Patterns.

Column *Power Consumption* offers the estimated power consumption values of the SocRocket VP for each software. Subcolumn *without PMU* shows the VP power consumption when no PMU is integrated with the VP (e.g. the VP works in full power mode), while subcolumn *with PMU* presents the VP power consumption when a PMU is added to the VP. As a way to show how *without PMU* and with PMU differ, Column *Different* shows the percentage of power reduction per each software. Given the fact that the PMU is an added module that consumes its own power, this latter value is already included in the subcolumn *with-PMU*. Note that this PMU self power consumption is variable with the software, but does not exceed 5% of the total (value that corresponds to the *AES* software, which means that a generated PMU presents very low power consumption overhead.

The estimated power consumption values are based on SoCRocket internal power modeling parameters for the modules of the VP. For the PMU power consumption estimation, these values are obtained by modeling a PMU as a *ahbin* type initiator TLM module from which the Control Signals are sent. This specific module for the PMU includes the internal read-only memory and the functions that implement the PMS.

Within the SocRocket SoC [94], a power modeling report function is available. This feature estimates the power consumed by the modules within the SocRocket platform, based on a set of back annotations from a 90 nm technology node. The power consumption is estimated as the sum of the static and dynamic power consumption, by assimilating physical measurements (such as voltage, current and capacitance that come from the back annotations from the technology nodes) to elements of the TLM methodology. Every TLM module (regardless if an initiator, interconnect or target), has a value for the static and dynamic power, which is estimated following the next set of equations:

$$p_{static} = p_{staticnorm} * (T_{end} - T_{start}) \tag{6.1}$$

$$p_{dynamic} = \frac{(e_{read} * n_{reads}) + (e_{writes} * n_{writes})}{T_{end} - T_{start}} \tag{6.2}$$

The static power consumption, which originates from the amount of gates of the modules fed by a voltage source at any given time, as well as the dynamic power consumption (originating from the way the modules switch their own power states), are both targets for power reduction. The estimation (calculation) of both static and dynamic power consumption does rest on the elements of the equations, which are explained below:

- $p_{staticnorm}$ is a fixed value for normalized static power consumption only affected by the size of the technology node.
- T_{end} and T_{start} are extracted from simulation log files as metrics for the activity period of a given module.
- e_{read} , e_{writes} , n_{reads} and n_{writes} are the fixed value for normalized energy consumption per read and write instruction and the number of read and write operations respectively. Each of these is also only affected by the size of the technology node.

For the power estimation values of Table 6.5 the normalized values for *AHBIN*, *AHBCTRL* and *AHBMEM* (as models for initiator, interconnector and target modules, respectively) are taken into consideration. While this is a simplification that may impact the precision of the numbers, it is a reasonable assumption. Arguably the most powerful technique to curb both static and dynamic power consumption is *Power Gating* (powering off unused parts of the design), which is considered as the technique of use for the generated PMU whose effect is stated in Table 6.5.

From inspecting the power consumption values with and without PMU as stated in Table 6.5, it is possible to conclude that the the percentage difference ranges from approximately **33%** to **64%**, which highlights the applicability of the PMU (implementing the PMS) that the response framework is able to offer.

6.4 Concluding Remarks

The response framework for technical language specifications is useful for Design Understanding at the system level, as well as for power-aware analysis leading to a generated PMU for a variety of designs (in pure SystemC or in SystemC/TLM, from processors to controllers). It has been shown to be a tool to rapidly compare alternative architectures (combinational and sequential) for a Hamming Encoder (a widely used ASIC worthy design) on the basis of their possible Power Modes, Control Signals and Power Domains. The unveiling of the power parameters from the design, as per the non intrusive and automated nature of response framework, showcases its usefulness at early design stages.

It is also clear from a complex and large TLM based VP (the SocRocket use case) that the response framework is able to deal with complex designs, yielding a generated PMU that does make a substantial difference in a VP's power consumption. The generated PMU achieves results of up to 64% power reduction for the use case while showing how effective it is in dealing with changing Behavioral Patterns, stemming from changes in the software run on the VP. The adaptability of the response framework is therefore ascertained by this effectiveness.

7 Conclusion

7.1 Summary

The research presented in this document constitutes both: a summary of the knowledge acquired and produced about specification analysis for power-aware design concept in ASICs as well as the presentation of contributions enabling said type of specification analysis to be of practical use. After an introduction to the research field, the reader was presented with the concept of power-aware design, its system-level implications and challenges, as well as relevant work on the topic that underlines the need for specification analysis at the system-level in order to address said implications and challenges. This document addresses the aforementioned aspects by presenting 2 frameworks, which constitute the 2 parts of the thesis: Part I (Chapter 3 and Chapter 4) dealing with specifications written in a typical natural language (ie. English), and Part II (Chapter 5 and Chapter 6), dealing with specifications written in a typical technical language (SystemC/TLM). The contributions of the thesis are centered around the frameworks for specification analysis presented in Chapter 3 and 5, which have been validated by their applications in the use cases presented in Chapter 4 and 6, respectively.

Chapter 2 introduced the reader to the key elements of power-aware design, as well as to what the core implications and challenges are. Chapter 2, in particular, presented the far reaching implications of power-aware design within the ASIC design processes. These implications impact not only how power and energy are managed in and by the digital circuit, but also the circuit's architecture. The centrality of the implications was highlighted by the presentation of the challenges associated with them. Whether via the expounding of techniques such as grouping (for functional blocks) or via the introduction of metrics that need to be properly managed (area overhead), the reader was acquainted with the relevant responses to the implications and their associated challenges. Chapter 2 thus introduced the setting of the *Power Domains* (PD), *Power Modes*(PM) and *Control Signals* (CS) that the research work tackled via specification analysis. The responses were presented in the context of the Power Management Unit (PMU), the archetypal component that implements the design's Power Management Strategy (PMS), responsible for making the ASIC power-aware. The need to analyze available system-level ASIC specifications for power-aware design was revealed to be crucial.

Chapter 3 presented the reader with a short introduction to natural language specification analysis as a field of research, focusing on the salient challenges posed by the ambiguous nature of natural language. Related works and the state of the art centered around Information Extraction (IE) and semantic analysis were thereby discussed, with the proposed framework being described in detail. Via the use of a semantically oriented analytical scheme, the framework was shown to be empowered to analyze relevant sentences extracted from the natural language specifications. This analysis, coupled with the use of rules in an Expert System led to appropriate responses to the core implications and challenges presented in Chapter 2. The framework is centered around the contribution of rules for the Expert System to output an appropriate PMS.

Chapter 4 showcased the effectiveness of the framework from Chapter 3 as a means to respond to the challenges of setting the PD, CS and PM (that is to say, the PMS). The use cases are representative of components likely to be developed under the ASIC paradigm (an LZW encoder, as well as an HDMI Port Processor). The validation of the framework was shown as the setting of PD, CS and PM are found to be in line with existing validated solutions.

In **Chapter 5**, the same structure of Chapter 3 was followed. Thus, the reader was introduced to the field of technical language specification analysis. Very closely linked to the field of *Design Understanding* (DU), the main challenges in this type of specification analysis relates to the need to perform analysis of *Virtual Prototypes* (VP), as VP are the traditional system-level technical language based specifications in need of analysis for power-aware design. The proposed framework (specifically its algorithms) was discussed therein. The algorithms (essentially, rules) constitute the core contribution of the framework, enabling a PMS to be extracted from the structural and behavioral information which are, in turn, derived from a technical language specification.

In a similar fashion to Chapter 4, **Chapter 6** offered the reader the results obtained by the use of the proposed framework introduced in the preceding chapter. In this chapter, the framework was shown to be applied to alternative encoders (Hamming based), as well as to a set of classical benchmark designs and to a Virtual Prototype (VP) of a System on Chip (SoC) running different softwares in its processing unit. The results obtained by the use of the framework were validated by their usability in the context of a zero-knowledge power-aware exploration of the VP.

7.2 Future Work

While the proposed frameworks for both natural language and technical language specifications have been proven to be effective, effectiveness is just one factor in any long term evaluation of success. As a proof of concept, the frameworks show what is possible through specification analysis for a power-aware design of ASICs. These frameworks are hopefully a stepping stone in further research.

As ASICs continue increasing their presence in the world, mainly through the expected manyfold growth of the *Internet of Things* (IoT), the challenges and issues that relate to power-aware design will only appear more urgent. And as natural language and technical language specifications are likely to continue to be the first design documents in a top-down down approach to digital circuit design, frameworks to process these documents will need to progress further.

The state of the art for natural language specification analysis for ASIC design is still it its infancy, with *Expert Systems* (ES) frameworks being in desperate need of groundbreaking efforts. Industrial grade applications of any kind of framework for natural language specification analysis are very slowly becoming more common in select areas of *Electronic Design Automation* (EDA), such as system level verification. However, power-aware design and security aware design are areas yet to see a similar upsurge in usage.

The state of the art for technical language specification analysis is in a more vibrant state compared to its natural language counterpart. However, frameworks and research lines in this field still lack a robust approach to the implications of power management, with verification tasks being almost their exclusive focus. Frameworks to perform DU are slowly gaining traction as the need for holistic *Design Space Exploration* (DSE) grows unstoppable.

Power-aware design of ASICs at the system level is to be developed to allow designers to make decisions about the best architecture to manage the IC's power implications. As any decision making process, power-aware design at the system level needs to conducted via an holistic understanding that mimics the way designers put their domain expertise to use. So as to make the holistic understanding more effective and reliable, the following are just possible lines of work that can be explored:

- **Knowledge-based Ontology for power-aware design concepts at system level.** This line of work would potentially allow for a systematization of the already available knowledge in the area. Research on ontologies is likely to attract computer linguists and other specialists to the area of specification analysis, in which they are sorely needed.

- **Improved decision support based rules for ES and DSE tools.** In order to better approximate the type of information extraction and processing performed by human designers, the rules for ES and DU (as well as for any other decision support metric) have to be enhanced. For the enhancement to take place, better and deeper encoding of said rules has to be developed. This line of work is ripe for knowledge engineers to start contributing extensively.

- **Power-aware technical specification and standards.** Technical standards (or other similar specifications) for the ICs in the IoT or even for existing complex system where ASICs are deployed are bound to require writing that aids in power-aware design decisions. Standard committees and other such authors may be prodded to start including chapters devoted to power in specifications, thus leading this line of work to get the influx and contributions of technical writers and of technical developers alike.

Ultimately, any future line of work should aim to help ease the transition of proof of concept works and frameworks into the industrial world. While the transition itself may be filled with obstacles, the history of the EDA community is equally filled with successful examples overcoming those obstacles. As it is common for many human endeavors, only time will tell how the transition takes places and how and when the ASIC design community starts to reap the benefits.

Bibliography

[1] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013 (cit. on p. 5).

[2] Jack S Kilby. “Origins of The Integrated Circuit”. In: *International Symposium on Silicon Materials Science and Technology*. 1998, pp. 342–349 (cit. on p. 5).

[3] David C Brock and Gordon E Moore. *Understanding Moore’s law: four decades of innovation*. Chemical Heritage Foundation, 2006 (cit. on p. 5).

[4] Carver Mead and Lynn Conway. *Introduction to VLSI systems*. Addison-Wesley Reading, MA, 1980 (cit. on p. 6).

[5] Don MacMillen, Raul Camposano, D Hill, and Thomas W Williams. “An industrial view of electronic design automation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19.3 (2000), pp. 1428–1448 (cit. on p. 6).

[6] IS Dhingra. “Formal validation of an integrated circuit design style”. In: *VLSI Specification, Verification and Synthesis*. Springer, 1988, pp. 293–321 (cit. on p. 6).

[7] Louis Scheffer, Luciano Lavagno, and Grant Martin. *EDA for IC system design, verification, and testing*. CRC press, 2018 (cit. on p. 6).

[8] Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994 (cit. on p. 6).

[9] Allen Dewey and Anthony Gadient. “VHDL motivation”. In: *IEEE Design and Test of Computers* 3.2 (1986), pp. 12–16 (cit. on p. 6).

[10] Lionel Bening and Harry Foster. “Optimizing multiple EDA tools within the ASIC design flow”. In: *IEEE Design and Test of Computers* 18.4 (2001), pp. 46–55 (cit. on p. 7).

[11] Kazutoshi Wakabayashi and Takumi Okamoto. “C-based SoC design flow and EDA tools: An ASIC and system vendor perspective”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19.12 (2000), pp. 1507–1522 (cit. on p. 7).

[12] Michael John Sebastian Smith. *Application-specific integrated circuits*. Addison-Wesley, 1997 (cit. on p. 7).

[13] Jörg Henkel. “Closing the SoC design gap”. In: *Computer* 36.9 (2003), pp. 119–121 (cit. on p. 7).

[14] *IEEE Standard for Standard SystemC Language Reference Manual, Std 1666-2011 (Revision of IEEE Std 1666-2005)*. 2012 (cit. on p. 7).

[15]Lukai Cai and Daniel Gajski. “Transaction level modeling: an overview”. In: *International Conference on Hardware Software Codesign and Systems Synthesis*. IEEE. 2003, pp. 19–24 (cit. on p. 8).

[16]Vaibbhav Taraate. “ASIC Design and SOC Prototyping”. In: *Logic Synthesis and SOC Prototyping*. Springer, 2020, pp. 15–28 (cit. on p. 8).

[17]David E Ott. “IoT Security through the Lens of Energy Efficiency: Energy as a First-Order Security Consideration”. In: *Cybersecurity Symposium (CYBERSEC)*. IEEE. 2016, pp. 20–25 (cit. on p. 8).

[18]David Chinnery and Kurt Keutzer. *Closing the power gap between ASIC and custom: tools and techniques for low power design*. Springer Science and Business Media, 2008 (cit. on p. 8).

[19]Rakesh Chadha and Jayaram Bhasker. *An ASIC low power primer: analysis, techniques and specification*. Springer Science and Business Media, 2012 (cit. on p. 8).

[20]Mathias Soeken and Rolf Drechsler. *Formal specification level*. Springer, 2016 (cit. on p. 8).

[21]Hamilton Carter. “Low-Power Roundtable: Verification and Power vs Performance”. In: *Embeddded Intel Solutions, Spring 2014*. Intel (cit. on p. 12).

[22]Massoud Pedram and Jan M Rabaey. *Power aware design methodologies*. Springer Science and Business Media, 2002 (cit. on p. 13).

[23]Osman S Unsal and Israel Koren. “System-level power-aware design techniques in real-time systems”. In: *Proceedings of the IEEE* 91.7 (2003), pp. 1055–1069 (cit. on p. 14).

[24]Jan M Rabaey and Massoud Pedram. *Low power design methodologies*. Springer Science and Business Media, 2012 (cit. on p. 14).

[25]Zhiru Zhang, Deming Chen, Steve Dai, and Keith Campbell. “High-level synthesis for low-power design”. In: *IPSJ Transactions on System LSI Design Methodology* 8 (2015), pp. 12–25 (cit. on p. 16).

[26]Nazish Aslam, Tughrul Arslan, and Ahmet Erdogan. “Algorithmic level design space exploration tool for creation of highly optimized synthesizable circuits”. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2007, pp. II–81 (cit. on p. 15).

[27]Giuseppe Ascia, Vincenzo Catania, Alessandro G Di Nuovo, Maurizio Palesi, and Davide Patti. “Efficient design space exploration for application specific systems-on-a-chip”. In: *Journal of Systems Architecture* 53.10 (2007), pp. 733–750 (cit. on p. 15).

[28]R Shathanaa and N Ramasubramanian. “Design Space Exploration for Architectural Synthesis—A Survey”. In: *Recent Findings in Intelligent Computing Techniques*. Springer, 2018, pp. 519–527 (cit. on p. 15).

[29]Deming Chen, Jason Cong, Yiping Fan, and Zhiru Zhang. “High-level power estimation and low-power design space exploration for FPGAs”. In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2007, pp. 529–534 (cit. on p. 15).

[30] Goran Panić. “A methodology for designing low power sensor node hardware systems”. PhD thesis. Brandenburgische Technische Universität Cottbus-Senftenberg, 2015 (cit. on p. 17).

[31] Dominik Macko. “System-level Power-managament Specification”. In: *Počítačové architekturyc & diagnostika (PAD)* (2013), p. 87 (cit. on p. 18).

[32] David Pursley and Tung-Hua Yeh. “High-level low-power system design optimization”. In: *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE. 2017, pp. 1–4 (cit. on p. 18).

[33] Abdellatif Bellaouar and Mohamed Elmasry. *Low-power digital VLSI design: circuits and systems*. Springer Science and Business Media, 2012 (cit. on p. 18).

[34] Robert A Walker and Donald E Thomas. “A model of design representation and synthesis”. In: *Design Automation Conference (DAC)*. IEEE. 1985, pp. 453–459 (cit. on p. 22).

[35] Matt Hogan. *SoC Reliability Verification Doesn’t Just Happen, You Know?* `http://semimd.com/favre/2014/07/16/soc-reliability-verification-doesn%E2%80%99t-just-happen-you-know/`. Mentor Graphics (cit. on p. 26).

[36] Abhinav Agarwal et al. “Leveraging rule-based designs for automatic power domain partitioning”. In: *International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2013, pp. 326–333 (cit. on p. 25).

[37] Bo Wang et al. “Exploration of Power Domain Partitioning for Application-Specific SoCs in System-Level Design”. In: *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. VDE. 2016 (cit. on pp. 25, 83).

[38] Dominik Macko. “Rapid Estimation of Power-Management Unit Overhead from System-Level Specification”. In: *Euromicro Conference on Digital System Design (DSD)*. IEEE. 2017, pp. 9–13 (cit. on p. 27).

[39] Rich Collins. *Addressing IoT Connectivity Challenges with a Low-power IoT Communications Solution*. `https://www.synopsys.com/designware-ip/technical-bulletin/iot-connectivity-low-power-solution.html`. Synopsys (cit. on pp. 27, 28).

[40] Yves Vanderperren and Wim Dehaene. “A model-driven development process for low power soc using uml”. In: *UML for SOC design*. Springer, 2005, pp. 223–252 (cit. on p. 31).

[41] Claire Cardie. “Empirical methods in information extraction”. In: *AI magazine* 18.4 (1997), p. 65 (cit. on p. 32).

[42] Zhanjun Li and Karthik Ramani. “Ontology-based design information extraction and retrieval”. In: *AI EDAM: Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* 21.2 (2007), pp. 137–154 (cit. on p. 32).

[43] Zeeshan Ahmed. “Domain specific information extraction for semantic annotation”. PhD thesis. Univerzita Karlova, Matematicko-fyzikální fakulta, 2010 (cit. on p. 32).

[44] Accellera Systems Initiative. *OSCI SystemC 2.3*. `http://www.accellera.org/` (cit. on pp. 33, 77, 98).

[45] Christopher B Harris and Ian G Harris. “Generating formal hardware verification properties from Natural Language documentation”. In: *International Conference on Semantic Computing (ICSC)*. 2015, pp. 49–56 (cit. on p. 37).

[46] Bhanu Singh, Arunprasath Shankar, Yuriy Shiyanovskii, et al. “Knowledge-guided methodology for specification analysis”. In: *International Conference on Tools with Artificial Intelligence (ICTAI)*. 2013, pp. 749–754 (cit. on p. 37).

[47] Arunprasath Shankar, Bhanu Singh, Francis Wolff, and Christos Papachristou. “Ontology-guided conceptual analysis of design specifications”. In: *Design Automation Conference (DAC)*. IEEE. 2014, pp. 1–6 (cit. on p. 37).

[48] Bhanu Singh, Arunprasath Shankar, Francis Wolff, and Christos Papachristou. “An Expert System Based Tool for Pre-design Chip Power Estimation”. In: *DVCon*. 2014 (cit. on p. 37).

[49] Andreas W Liehr, Klaus J Buchenrieder, and Ulrich Nageldinger. “Visual feedback for design-space exploration with UML MARTE”. In: *International Conference on Innovations in Information Technology*. IEEE. 2008, pp. 44–48 (cit. on p. 38).

[50] Robin L Steele. “An expert system application in semicustom VLSI design”. In: *Design Automation Conference (DAC)*. IEEE. 1987, pp. 679–688 (cit. on p. 38).

[51] Rick L Spickelmier and A Richard Newton. “Critic: A knowledge-based program for critiquing circuit designs”. In: *International Conference on Computer Design: VLSI*. IEEE. 1988, pp. 324–327 (cit. on p. 38).

[52] Carol E Brown. “Expert systems in public accounting: Current practice and future directions”. In: *Expert Systems with Applications* 3.1 (1991), pp. 3–18 (cit. on p. 38).

[53] PM Macey, JSJ Li, and RPK Ford. “Expert system for the detection of apnoea”. In: *Engineering Applications of Artificial Intelligence* 11.3 (1998), pp. 425–438 (cit. on p. 38).

[54] Christine Chan and Patrick Lau. “An expert system architectural framework for engineering selection”. In: *Engineering Applications of Artificial Intelligence* 10.4 (1997), pp. 357–367 (cit. on p. 38).

[55] PA Subrahmanyam. “Synapse: an expert system for VLSI design”. In: *Computer* 19.7 (1986), pp. 78–89 (cit. on p. 39).

[56] Jung-Gen Wu, Yu Hen Hu, WP-C Ho, and David YY Yun. “A model-based expert system for digital system design”. In: *IEEE Design and Test of Computers* 7.6 (1990), pp. 24–40 (cit. on p. 39).

[57] Pat Pannuto, Yoonmyung Lee, Ye-Sheng Kuo, et al. “MBus: A System Integration Bus for the Modular Micro-Scale Computing Class”. In: *IEEE Micro: Special Issue on Top Picks from Computer Architecture Conferences*. Micro Top Picks (2016) (cit. on p. 41).

[58] Pat Pannuto, Yoonmyung Lee, Ye-Sheng Kuo, et al. *MBus Specification, Revision 1.0-alpha*. 2015 (cit. on p. 41).

[59] Pat Pannuto, Yoonmyung Lee, Ye-Sheng Kuo, et al. *MBus M3 Implementation Specification, Revision 0.3*. 2014 (cit. on p. 41).

[60] Vladimir Zdraveski and Dimitar Trajanov. “VHDL IP cores ontology”. In: *Conference for Informatics and Information Technology*. 2013, pp. 240–243 (cit. on p. 43).

[61] Fateh Boutekkouk. “Towards an ontology-driven intellectual properties reuse for systems on chip design”. In: *International Conference on Systems, Control, Signal Processing and Informatics*. 2013, pp. 394–399 (cit. on p. 43).

[62] Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, et al. “Universal dependencies v1: A multilingual treebank collection”. In: *International Conference on Language Resources and Evaluation (LREC)*. 2016, pp. 1659–1666 (cit. on p. 43).

[63] Timothy Osborne and Kim Gerdes. “The status of function words in dependency grammar: A critique of Universal Dependencies (UD)”. In: *Glossa: a journal of general linguistics* 4.1 (2019) (cit. on p. 43).

[64] Aaron Steven White, Drew Reisinger, Keisuke Sakaguchi, et al. “Universal decompositional semantics on universal dependencies”. In: *Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 1713–1723 (cit. on pp. 43, 44).

[65] Sheng Zhang, Rachel Rudinger, and Benjamin Van Durme. “An evaluation of predpatt and open ie via stage 1 semantic role labeling”. In: *IWCS International Conference on Computational Semantics—Short papers*. 2017 (cit. on p. 44).

[66] Michael J Pazzani and Clifford A Brunk. “Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning”. In: *Knowledge acquisition* 3.2 (1991), pp. 157–173 (cit. on p. 46).

[67] R. Martinez Perez. *Experta, Expert Systems for Python*. `https://github.com/nilp0inter/experta` (cit. on p. 50).

[68] Ming-Bo Lin. “A parallel VLSI architecture for the LZW data compression algorithm”. In: *International Symposium on VLSI Technology, Systems, and Applications*. IEEE. 1997, pp. 98–99 (cit. on p. 58).

[69] Wei Cui. “New LZW data compression algorithm and its FPGA implementation”. In: *Picture Coding Symposium (PCS)*. 2007 (cit. on p. 58).

[70] Malek Safieh and Jürgen Freudenberger. “Efficient VLSI architecture for the parallel dictionary LZW data compression algorithm”. In: *IET Circuits, Devices and Systems* 13.5 (2019), pp. 576–583 (cit. on p. 58).

[71] Orahyn Ltd. *An implementation of LZW encoder in SystemC and verified in FPGA*. `https://github.com/arshadri/lzw_systemc/` (cit. on p. 58).

[72] Giovanni B Vece and Massimo Conti. “Power estimation in embedded systems within a SystemC-based design context: the PKtool environment”. In: *Workshop on Intelligent solutions in Embedded Systems*. IEEE. 2009, pp. 179–184 (cit. on pp. 64, 82).

[73] LLC HDMI Licensing. *High-Definition Multimedia Interface Specification 1.3a*. 2006 (cit. on p. 66).

[74] Silicon Image. *Data Sheet SiI9187B HDMI Port Processor*. `https://datasheet.lcsc.com/szlcsc/1903061001_Lattice-SII9187BCNU_C369566.pdf` (cit. on pp. 66, 67, 72, 73).

[75] Tiantian Xie and Bogdan Wilamowski. “Recent advances in power aware design”. In: *Annual Conference on IEEE Industrial Electronics Society (IECON)*. IEEE. 2011, pp. 4632–4635 (cit. on p. 76).

[76] Frank Schirrmeister. *Design for low-power at the electronic system level*. 2009 (cit. on p. 76).

[77] B. Bailey. *Power limits of EDA*. `http://semiengineering.com/power-limits-of-eda` (cit. on p. 77).

[78] Yasaman Samei. “Automated Power-Aware System-Level Design with the MAVO Framework”. PhD thesis. University of California, Irvine, 2014 (cit. on p. 77).

[79] Ameni Ben Mrad, Michel Auguin, François Verdier, and Amal Ben Ameur. “A framework for system level low power design space exploration”. In: *International Conference on Electronics, Circuits and Systems (ICECS)*. 2017, pp. 437–441 (cit. on p. 77).

[80] Mehran Goli, Jannis Stoppe, and Rolf Drechsler. “AIBA: An Automated Intra-cycle Behavioral Analysis for SystemC-based design exploration”. In: *International Conference on Computer Design (ICCD)*. IEEE. 2016, pp. 360–363 (cit. on pp. 81, 85).

[81] Ons Mbarek, Amani Khecharem, Alain Pegatoquet, and Michel Auguin. “Using model driven engineering to reliably accelerate early low power intent exploration for a system-on-chip design”. In: *Annual ACM Symposium on Applied Computing*. ACM. 2012, pp. 1580–1587 (cit. on p. 82).

[82] Ons Mbarek. “An electronic system level modeling approach for the design and verification of low-power systems-on chip”. PhD thesis. Université Nice Sophia Antipolis, 2013 (cit. on p. 82).

[83] Simone Orcioni, Marco Giammarini, Cristiano Scavongelli, Giovanni B Vece, and Massimo Conti. “Energy estimation in SystemC with Powersim”. In: *Integration, the VLSI Journal* 55 (2016), pp. 118–128 (cit. on p. 82).

[84] Aritra Hazra, Rajdeep Mukherjee, Pallab Dasgupta, et al. “POWER-TRUCTOR: An integrated tool flow for formal verification and coverage of architectural power intent”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.11 (2013), pp. 1801–1813 (cit. on p. 82).

[85] Hend Affes and Michel Auguin. “SOC Power Management Strategy Based on Global Hardware Functional State Analysis”. In: *Euromicro Conference on Digital System Design (DSD)*. 2015, pp. 614–620 (cit. on p. 83).

[86] Hend Affes, Amal Ben Ameur, Michel Auguin, François Verdier, and Calypso Barnes. “An ESL framework for low power architecture design space exploration”. In: *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2016, pp. 227–228 (cit. on p. 83).

[87] Katarina Jelemenska and Dominik Macko. “Adopting High-level Synthesis Approach to Accelerate Power Management Design”. In: *International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)*. 2018, pp. 124–130 (cit. on p. 83).

[88] Michal Škuta and Dominik Macko. “Automated Integration of Dynamic Power Management into FPGA-Based Design”. In: *International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE. 2019, pp. 1–4 (cit. on p. 83).

[89] V. Herdt, H. M. Le, D. Große, and R. Drechsler. “Towards early validation of firmware-based power management using virtual prototypes: A constrained random approach”. In: *Forum on Specification and Design Languages (FDL)*. IEEE. 2017, pp. 1–8 (cit. on p. 83).

[90] Dominik Macko. “Contribution to Automated Generating of System Power-Management Specification”. In: *International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE. 2018, pp. 27–32 (cit. on p. 83).

[91] The Clang Team. *Clang: a C language family frontend for LLVM*. `https://clang.llvm.org/` (cit. on p. 84).

[92] Mehran Goli, Muhammad Hassan, Daniel Große, and Rolf Drechsler. “Automated Analysis of Virtual Prototypes at Electronic System Level”. In: *Great Lakes Symposium on VLSI (GLSVLSI)*. 2019, pp. 307–310 (cit. on p. 85).

[93] J. Aynsley. *TLM-2.0 Base Protocol Checker*. `https://www.doulos.com/knowhow/systemc/tlm2` (cit. on pp. 85, 98).

[94] T. Schuster, R. Meyer, and R. et al. Buchty. “SoCRocket-A virtual platform for the European Space Agency’s SoC development”. In: *International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. 2014, pp. 1–7 (cit. on pp. 96, 100, 103).

[95] Dominique Cansell, Stefan Hallerstede, and Ian Oliver. “UML-B specification and hardware implementation of a hamming coder/decoder”. In: *UML-B Specification for Proven Embedded Systems Design*. Springer, 2004, pp. 261–277 (cit. on p. 96).

[96] Varun Jindal. “Design of Hamming code using Verilog HDL”. In: *Electronics For You* (February 2006) (cit. on p. 96).

[97] *HammingCodeForFPGA*. `https://github.com/ChDuhil/HammingCodeForFPGA` (cit. on p. 97).

[98] *Hamming-Code-SystemC*. `https://github.com/SamTheDev/Hamming-Code-SystemC` (cit. on p. 97).

[99] Benjamin Carrion Schafer and Anushree Mahapatra. “S2CBench: Synthesizable SystemC Benchmark Suite for High-Level”. In: *IEEE Embedded Systems Letters* 6.3 (2014), pp. 53–56 (cit. on p. 98).