Simplifying Low-Power SoC Top-Down Design Using the System-Level Abstraction and the Increased Automation

Dominik Macko*, Katarína Jelemenská, Pavel Čičák

Slovak University of Technology in Bratislava, Ilkovičova 2, 84216 Bratislava, Slovakia

Abstract

Since power is the key aspect in modern systems on chips, many power-reduction techniques are adopted in the design process, mostly applied through power management. Its standard specification lacks the abstraction required by complex designs and therefore becomes difficult and error-prone. In this work, higher abstraction is introduced into the power-management specification and it is integrated with the functional model of the system. It simplifies the specification approximately 16.8 times and enables the automatic generation and verification of the equivalent standard specification. The error-prone nature of the power-management specification is thus alleviated and the difficult verification process is relieved.

Keywords: high-level synthesis, low power design, power management, system-level specification, verification

1. Introduction

The power consumption is a great concern for hardware systems developers mainly due to increasing power density in systems on chips (SoC). Therefore, power must be dealt-with during the whole design process, usually by utilizing a so-called power management. However, it complicates the already too complex design process even more, and therefore abstraction and automation must be used to cope with the complexity (also to increase the productivity). The electronic system level (ESL) abstraction slowly becomes the design starting point in the industry and several methods have been developed (e.g. [1, 2]) to also raise the abstraction level for adoption of power management into the design. However, they are either too dependant on design reuse, use too much lower level details for specification, or do not provide sufficient automation.

*Corresponding author
Email address: dominik.macko@stuba.sk (Dominik Macko)

This paper presents a new low power systems design methodology, which eliminates weaknesses and builds up on the strengths of the existing methods. It integrates the power-management specification directly into the functional model of the system at the ESL (in SystemC) and uses high-level synthesis to automatically obtain the standard power-managed register-transfer level (RTL) model of the same system. The simplified abstract specification makes the designer's input more efficient (approximately 16.8 times) and the automatically synthesized RTL model enables more accurate design analysis. This methodology is enhanced by automated verification processes, which drive a designer to the correct and complete specification.

This paper is organized as follows. The next section (Section 2) provides a deeper background and motivation for our research. In Section 3, the state-of-the-art in the area of ESL-based power management is described. Section 4 provides an overview of the proposed low-power SoC design methodology along with a description of the utilized new methods of abstract power-management specification and power-management high-level synthesis. Before the conclusion, the experimental results (Section 5), illustrating the usefulness of the methodology, and comparison to related works (Section 6) are presented.

2. Background

In order to reduce power consumption, one must understand what factors influence the power. The total power in CMOS (Complementary Metal-Oxide Semiconductor) technology is a function of switching activity, capacitance, voltage, and transistor physical properties [3]. Formally, it is expressed by

$$P = P_{SW} + P_{SC} + P_L \tag{1}$$

where P is the total power, P_{SW} is the switching power, P_{SC} is the short-circuit power, and P_L is the leakage power. The leakage power is also called the static power. The switching power together with the short-circuit power are referred to as the dynamic power (P_D). Components of the dynamic power are defined in the following equations.

$$P_D = P_{SW} + P_{SC} \tag{2}$$

$$P_{SW} = a.f.C_{eff}.V_{dd}^2 \tag{3}$$

$$P_{SC} = I_{SC}.V_{dd} \tag{4}$$

In these equations, a represents the switching activity, f is the switching frequency, C_{eff} is the effective capacitance, V_{dd} is the supply voltage, and I_{SC} is the short-circuit current. Therefore, the dynamic power can be lowered by reducing switching activity and clock frequency (affecting performance), or by reducing capacitance and supply voltage. Leakage power is a function of the

supply voltage, the switching threshold voltage, and the transistor size. It is dissipated continuously, because of the leakage current, and thus design techniques (such as enabling of powering-down the circuit when not used) must be used to reduce it [3]. All of these factors are substantially used to reduce power in modern SoCs. Based on which power-affecting factor is targeted, various power-reduction techniques have been developed. The following subsection summarizes the most popular techniques and their standard application in the design.

2.1. Power-Reduction Techniques Application

The existing power-reduction techniques can be divided into three categories:

- *Circuit-optimization techniques* – This category contains techniques that change physical parameters of the designed circuit (e.g. structure and size), such as logic restructuring, transistor resizing, pin swapping, or multiple supply voltages.

- *Power-management techniques* – These techniques utilize a dynamic power management. It enables the device to temporarily switch the operating mode in order to save the energy. Some portions of the device can stop its operation, isolate the spreading of the signals, adjust the voltage or frequency, or even can be powered down. These techniques include clock gating, operand isolation, substrate biasing, voltage and frequency scaling, and power gating.

- *Architectural techniques* – This category is a hybrid one containing rather the architectural choices enabling other power-reduction techniques to be applied. These techniques include, for example, memory or bus segmentation and hardware acceleration.

Since the system power highly depends on the used implementation technology, it is coupled with the physical level of the design. However, hardware designs are too complex at such a low level, and therefore the adoption of advanced power-reduction techniques (working with multiple voltages) would be very difficult and error-prone. It would require full-chip functional verification, which would be unbearable (in terms of time). In order to deal with design complexity and to reduce the number of design re-spins the IEEE standard for design and verification of low-power integrated circuits [4] has been developed (commonly known as UPF – Unified Power Format). There is another widely used standard, known as CPF (Common Power Format) [5], which has similar capabilities to the UPF. However, since these standards are unifying in new versions of UPF, we focus only on this one. The UPF has offered a clear design flow utilizing the power-management techniques and enabled RTL (Register-Transfer Level) power-aware verification. It contains the constructs for specification of low-level power-related aspects during the design stage, when mostly the HDL (Hardware Description Language) modelling is used. In such a way, the whole design (the functional HDL model along with the UPF power management) can be verified.

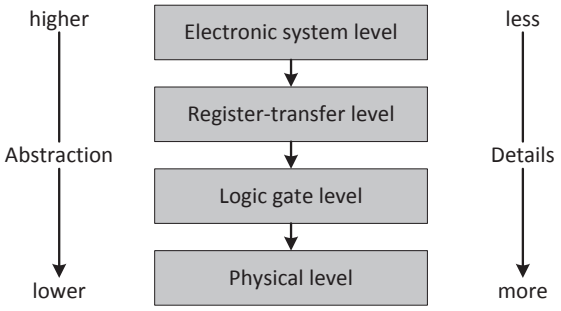

Figure 1: Design process abstraction levels.

The key UPF concept is to provide the means for dividing the system into power domains. The power domain is a collection of design blocks that always operate at the same supply voltage level. UPF enables the designer to specify which blocks are grouped into the power domain, what voltage levels the power domain can operate at, what the power-down condition for each power domain is, where the isolation cells and level shifters should be used, and so on.

However, the use of this standard is still rather complicated (error-prone and time-consuming) especially in modern SoC designs that are becoming more and more complex. To reflect the current trend of adopting more-abstract level above the RTL [6], the so-called electronic system level (ESL), the UPF has been updated to the version 3.0 [7]. It enables to specify power intent and verify it in context of IP blocks (Intellectual Properties) in the ESL model. It standardizes the power model, in which the designer can specify power consumption of an IP in various power states. The IP power characterization way is, however, outside the scope of this standard. For accurate power analysis, these data can be obtained from the previous implementation of the IP block, and therefore such a method is highly dependent on the design-reuse concept. Although an algorithmic model can be synthesized into IP block based model (with known power characterization) and UPF 3.0 then used for power management, it complicates the top-down design process and limits flexibility to only pre-designed IP blocks. Moreover, the specification of power management (e.g. power-supply networks, power-management elements, or power switching) does not correspond to this abstraction level (i.e. it uses too low-level details). The next subsection introduces the ESL-based design and problems of integrating UPF in such a design process.

2.2. System-Level Design

In the top-down design process utilizing the ESL, the order of abstraction levels is illustrated in Figure 1. At the highest abstraction level, the most widely used modelling approaches are based on C or C++ languages for system description (SystemC [8] is the most popular along with its TLM extension – Transaction Level Modelling), because they can be used for both algorithmic and architectural modelling. The top-down transitions between the abstraction

Table 1: Overview of the Selected System-Level EDA Tools

Tool	Vendor	Language	Power management	High-level synthesis
Intel Docea Power Simulator [9]	Intel Corporation (DOCEA Power)	TLM	Yes	No
Stratus [10]	Cadence Design Systems	C/C++/SystemC	Clock gating	Yes
Catapult [11]	Mentor Graphics (Calypto)	C/C++/SystemC	Clock gating	Yes
Platform Architect [12]	Synopsys	SystemC/TLM	No	No
Synphony C Compiler [13]	Synopsys	C/C++	Clock gating	Yes
Vista [14]	Mentor Graphics	SystemC/TLM	No	No
Vivado Design Suite [15]	Xilinx	C/C++/SystemC	Clock gating	Yes

levels are achieved through the synthesis processes – high-level synthesis, logic synthesis, and place and route processes. The development process is nowadays usually accelerated by the use of various EDA (Electronic Design Automation) tools for synthesis or verification. The RTL and lower levels, along with the transitions between these levels, are well-supported by such tools in all aspects. Regarding the ESL, several EDA tools enable the system architecture definition in the SystemC/TLM form, or enable capturing the functionality in the C/C++/SystemC algorithmic manner. Some of them support even the high-level synthesis. The best-known of these tools are summarized in Table 1. These tools promote the easier use of the ESL in the design process. At first, the ESL was adopted in the industry mainly for verification purpose (e.g. virtual prototyping). Higher abstraction enables faster verification of the functionality. The verified ESL model further serves as a golden model for equivalence checking with the more complex RTL implementation model. However, the advances in high-level synthesis in the modern EDA tools enable the adoption of ESL as an implementation starting point. It means that the RTL model is automatically generated from the ESL model, according to some predefined constraints. It reduces the number of human errors in the design and shortens the verification time. Moreover, the automated high-level synthesis enables to get the results of more accurate design analysis at the RTL sooner, and thus it enables to find the right trade-off among multiple parameters (power, performance, area) much faster.

As shown in Table 1, SystemC has basically become a standard for ESL design. The analysed high-level synthesis tools do not offer much power-management support (besides simple clock gating or memory architecture selection). On the other hand, the power-management capabilities of Intel Docea Power Simulator are highly dependent on design reuse; therefore, it is not suitable for the top-down design process. The abstraction of the UPF power-intent specification is not sufficient for the ESL (i.e. it includes low-level details, such as power-supply networks or power-management elements) and its ESL-based power modelling is too much dependent on design reuse (to obtain power values

for accurate power-analysis results). However, if the power-management specification is omitted at the ESL, its introduction in the automatically synthesized RTL model represents an intrusion that would require enormous manual effort, and thus getting the power architecture right would take too much time. Moreover, a multi-parameter trade-off achieved by a high-level synthesis EDA tool would be disrupted. Therefore, the power management should be adopted from the beginning of the design process and thus an extension of the standard low-power design flow is needed in order to utilize the advantages of the ESL (concise and less error-prone specification, faster verification). An integration of the new design-automation techniques into such a flow should result in faster low-power SoC design process with fewer errors. Some ESL-based power-management methods have already been developed, as described in the following section. However, they lack the required abstraction and automation. Therefore, we propose a new low-power SoC design methodology, extending the standard low-power design flow to the system level, combining strengths and eliminating weaknesses of the current methods.

3. Related Works

The ESL power management research area has gained attention in the past few years. Several published research approaches are focused on the extension of the power-management specification up to the system level.

Mbarek et al. [16] have proposed a framework enabling the exploration of different power architectures at the transaction level (a part of the system level). It augments an existing SystemC/TLM model with abstract UPF concepts, which are used for ESL simulation. The augmented ESL model then serves as a reference model for the RTL implementation, which is manual and thus error prone. It has been further extended by Affes et al. [1] to incorporate clock management. The proposed framework is mainly targeted towards TLM power estimation. The power consumption of the system components has to be already known and its introduction into the design represents an additional manual effort. Other disadvantages of this approach include the fact that the architecture exploration does not take into account other important parameters, such as area or performance. Also, the power-management is specified separately and using the language and style different from the functional specification. At the system level, it would be more appropriate for designers to use the same language for all aspects of the system specification.

Mischkalla and Mueller [2] have proposed a SystemC-based virtual prototyping approach capturing power intent at the ESL. The used specification method, extending UPF concepts to the ESL, includes abstract specification of voltage relationships, operating conditions, TLM power states, and so on. The approach is primarily focused on modelling power intent in temporal decoupled simulation. The authors extended power-domain UPF concept in such a way that they mapped each power domain to one clock domain. Thus, the components in a power domain must always have the same supply voltage state as well as the same clock frequency. The COMPLEX framework, proposed by Grüttner

et al. [17], represents yet another virtual prototyping approach. It enables to evaluate different power strategies at the system level. Such virtual prototyping approaches are suitable for early power-architecture exploration in top-down design flows. However, the analysed approaches have similar drawbacks as [1] in terms of missing automation towards lower abstraction levels, separated specifications for functionality and power intent, and omission of the other important design parameters besides power.

Similarly to [16], Karmann and Ecker [18] have also augmented the ESL functional model with the power intent model and power data model for power estimation. The power intent model, specifying the power management, is based on the UPF standard concepts. The power data model is created based on the design reuse. It utilizes the data from lower level power estimations and technology libraries. Therefore, it is not suitable for top-down design process. Similarly, Gagarski et al. [19] have proposed SCPower extension of SystemC, which integrates power intent specification into SystemC model description and enables to generate UPF automatically. The disadvantage of these approaches is that the proposed power intent models contain the information approximately at the same level of abstraction as the UPF standard, and thus, it is just rewritten to the standard form. It is not convenient to specify low-level details at the ESL specification stage, such as supply ports, supply nets, isolation cells, or power switches. The ESL methodology should abstract from as many details as possible and introduce them implicitly during the synthesis of RTL model.

A different approach has been proposed by Xu et al. [20]. It is based on annotation of components power requirements in various power states, and thus it makes ESL power estimation more accurate. The performance data are analysed during simulation, and the whole system power consumption is calculated. Although the proposed approach enables to take into account dynamic power management while estimating power, it cannot model UPF-based power-management concepts. The automation in the transition to the RTL is missing and the equivalence between ESL and RTL power management is difficult to verify. This approach is convenient for SoC architecture exploration and for determination of the proper power-management strategy. However, power data have to be known beforehand; therefore, it is not suitable for top-down design process. Also, the proposed power annotation uses XML-based information, and thus the language and style is not consistent with the functional design, captured in SystemC.

Similar approach is used by Lebreton and Vivet [21] and Bouhadiba et al. [22]. The approaches also utilize the ESL power modelling, enabling faster simulation and different power-architectures exploration. In these methods, the components power information has to also be manually annotated in the ESL model (either data-sheets or RTL estimations are used). The proposed power-management modelling approaches are based on power-state models; otherwise, they are not based on the standard UPF concepts. It complicates the verification of the equivalence between ESL and RTL power management. In addition to the power-state models, Bouhadiba et al. [22] have proposed the use of traffic models enabling to compute a more-accurate power profile of the components.

It is combined with the temperature-aware simulation, and thus it enables the exploration of a power-management policy effect on chip temperature and functionality. However, the traffic models provide additional overhead (manual annotation) and the resulted accuracy is questionable compared to the RTL design analysis (including the temperature). There are other ESL approaches, which can be used to explore power architectures but are not based on the standard UPF concepts, such as those proposed by Keiser et al. [23], Streubühr et al. [24], and Hsieh et al. [25]. The missing automated transition and complicated verification between ESL models and lower-level models are their greatest drawbacks.

Ahuja et al. [26] have presented an approach, which focuses on high-level power estimation based on statistical regression power models. At lower levels, the proposed method uses ESL simulation traces to estimate power. The power reduction at the system level is focused on the clock-gating technique. However, the use of clock gating is manually annotated in a form of macro, directly in the C source code. Based on the C-based specification, the high-level synthesis tool implements the clock gating cells in the generated RTL model. The drawback of this approach is that a designer has to manually specify the precise location of clock gating, which complicates the system-level specification. Moreover, the specification method does not include other usable and effective power-reduction techniques, such as power gating or voltage and frequency scaling. A similar approach utilizing the high-level synthesis is proposed by Qamar et al. [27]. The proposed LP-HLS methodology enables to automatically generate CPF commands, required for power gating (referred to as power shut-off), based on annotations in the SystemC model. The power is analysed in the synthesized model at the RTL. This is a promising approach; however, the used ESL specification method uses rather RTL perspective of power management. For example, a designer has to specify shutoff and isolation conditions. Such information is then just rewritten in a CPF style. It would be better to use the system-level perspective of what needs to be done (e.g. power-down the block), not how (e.g. isolate the block and then activate the power switch).

Besides the ESL-based power management, there is a significant focus on system-level power estimation in the research community, for example, Kornaros [28], Giammarini et al. [29], Greaves and Yasin [30], or Rigo et al. [31]. The developed power-estimation methods enable the architecture exploration, which helps to select the most power-efficient system architecture. The energy consumption of the individual monitored elements also needs to be manually specified. If this information is missing, the predefined default values can be used in some cases. Since these methods do not contain the constructs for power management, the ESL power-management specification is not supported, i.e. different power-management policy cannot be explored. The result is that if the power management is introduced into the design at the lower levels, the obtained ESL power estimation does not correspond to the actual power consumption of the power-managed SoC.

Based on the analysis of the existing methods and approaches, we have identified their advantages and drawbacks. According to our knowledge, there

has not yet been published any approach targeting a combination of existing methods to eliminate their weaknesses (e.g. insufficient abstraction, missing automation, insufficient verification). Therefore, we have decided to propose such a solution. A new ESL-based design methodology combining the strengths of the existing approaches could make the low-power SoC design process more efficient. The standard-based abstract system-level power management can be inspired by [7, 16, 2]. If the concepts of power management are the same at both abstraction levels (ESL and RTL), the equivalence is verified much easier. The unification of power and clock management, as proposed in [1, 2], would simplify the specification. Generation of a fully standardized (UPF) power management at the RTL, as used in [18, 19, 9] (improved by a higher abstraction and a complete automation), would ensure the compatibility with the existing EDA tools. The use of high-level synthesis in the power-architecture analysis process (similarly to [26, 27]) should provide better trade-off (performance, power, and area) and more accurate analysis. Such an approach, exploiting more automation, will also reduce the possibility of human errors. The RTL implementation stage will be achieved more quickly, and therefore a more mature RTL verification process could start earlier. The abstract power management should be easier to understand; thus, even the designers not familiar with the details of power-management techniques should be able to design for low power. We have developed such a methodology, which is described in the remaining part of this paper.

4. A New Methodology for Low-Power SoC Design

As a result of the analysis, we have specified the following requirements for the new design methodology.

1. The design process should start at the system level of abstraction, in order to deal with the complexity of modern designs.
2. The specification at the system level should be as simple as possible, in order to avoid specification errors, since there is no reference model against which it could be verified.
3. The application of various power-reduction techniques into the design should be unified, in order to simplify their adoption.
4. The errors should be revealed as soon as possible, in order to avoid difficult, time-consuming and costly debugging process.
5. The design analysis should be accurate, in order to be able to make the right trade-off among multiple parameters.
6. The design and verification processes should be automated as much as possible, in order to shorten the development time.
7. The manual intervention in the design at later design stages should be minimized as much as possible, in order to avoid human-errors introduction.

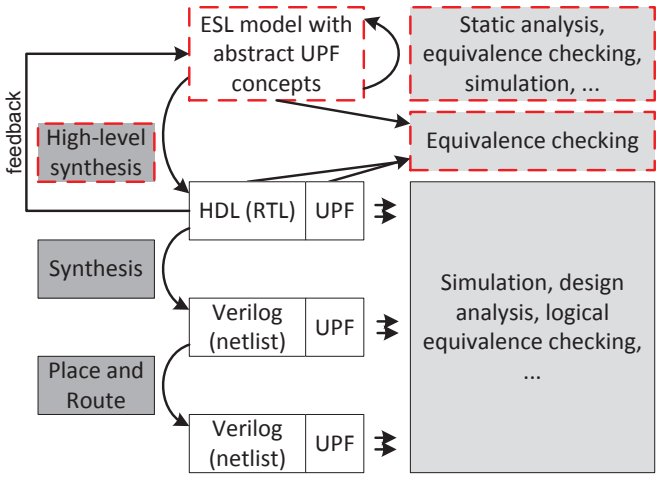

Figure 2: The proposed low-power systems design flow.

To fulfil these requirements, the proposed low-power systems design methodology uses a novel design flow (Figure 2), extending the standard UPF-based flow to the system level of abstraction. The steps of the UPF-based design flow [4] remains intact; thus, the existing methods and tools can be used at the RTL and lower levels.

The proposed methodology starts at the ESL, where the standard specification model is extended by abstract power-management specification. The ESL model traditionally goes through the abstraction refinement process, which clarifies all the functional details. During this process, the equivalence checking (formal or simulation-based) is usually used to verify that the key functionality is not changed at respective refinement stages. Power-management specification is also participating in the refinement process, during which a designer refines the ESL power management until it contains all the necessary details. In this process, the proposed static analysis helps the designer to specify correct and complete power management. After the ESL model is sufficiently refined, it is translated to the RTL representation using the high-level synthesis process. The high-level synthesis generates the functional model in VHDL or Verilog (depends on the used EDA tool) and the power-intent specification in UPF. The proposed equivalence checking is used after the high-level synthesis to verify that the power intent was not changed during this process. At the RTL, the existing EDA tools are used to functionally verify the RTL model and to analyze the design in terms of power. Based on the design analysis results, the ESL power-management specification can be changed and the high-level synthesis can be repeated. This enables to explore various power architectures of the system and find the suitable one.

This methodology is based on the new power-intent specification method [32], unifying the functional and power-management design styles. The proposed

abstract power management incorporates multiple carefully selected power-reduction techniques, suitable for the system level of abstraction. The cornerstone of the methodology is the novel power-management high-level synthesis process [33], enabling the design automation. The methodology incorporates multiple new verification methods [34], ensuring the correct power-management specification. These methods have been further refined regarding the issues rising from their integration into a single methodology. These aspects are closely described in the following subsections.

4.1. Power-Reduction Techniques Selection

The considered techniques are well-known, usually implemented in the current low-power designs. Their description can be found in many sources, such as [3, 35, 36, 37, 38]. This subsection focuses on reasoning behind the selection of those techniques that are incorporated into the proposed methodology. The selection criteria are mostly based on suitability for the system level of abstraction and impact on eventual power consumption.

Clock gating has a high impact on dynamic power consumption. At the system level of abstraction, clock signal can be represented by a clock synchronization variable. Functionality of this technique can be seen as enabling/disabling an operation of the whole synchronous block (IP core) of the system. *Operand isolation* prevents the switching of inactive datapath elements. From the system level perspective, it can be also seen as enabling/disabling an operation of the system block, but it focuses on combinational parts not currently needed. These two techniques can be modelled at the system level by means of a specific power state, represented by the state variable value. When the block is in this state it stops its operation.

Logic restructuring, *pin swapping*, and *transistor resizing* are techniques tightly coupled with lower abstraction levels (logic or physical level). Since at the specification stage the designer does not know how the system will be implemented, these techniques cannot be used so early in the design process. Moreover, these techniques are mostly automated in the modern synthesis tools. The *substrate-biasing* technique depends on the individual transistor type and its threshold voltage. This technique can be used at the system level by means of another specific power state – while the block is in this state, the bias is activated, the threshold voltage is raised, and thus the leakage power is reduced. However, this technique loses its effectiveness in smaller technologies (below 65 nm) because of its area requirements, and thus it is omitted in our methodology.

The *voltage-scaling* technique switches among multiple supply-voltage levels. At the system level, it is not suitable to specify such low-level details as the voltage-switch elements. However, the state variable of the individual block can reflect also the supply-voltage level of that block. When the state variable changes its value, the block adjusts the supply-voltage level. Similarly, the *multiple supply voltages* technique can be used. The difference is that the state variable reflecting the supply-voltage level of the block is not changing in time. This means that the techniques working with the supply voltages can be used at the ESL, but in a highly abstract form. The *frequency-scaling* technique is

usually coupled with the voltage scaling. Similarly as the previous techniques, when the block state variable, allocated to this technique, changes its value, the operation speed of the block is adjusted. *Power gating* with or without the state retention is a powerful leakage-power reduction technique that powers-down the currently unused blocks. Since this technique also basically works with supply voltage, this concept can be modelled by means of the block state variable mentioned earlier. When the block state has to be retained, different value of the block state variable has to be used.

As mentioned in Section 2, memory or bus segmentation and hardware acceleration techniques are rather micro-architectural choices for individual IP cores enabling other power-reduction techniques to be used inside these blocks. They imply the power management to have hierarchical nature, which enables the local power management to be used inside the system blocks.

The proposed power-intent specification is based on the power management, which is abstracted from the unnecessary details. It is augmented by the techniques that were not originally developed as power-management techniques (clock gating, operand isolation, multiple supply voltages) but can be modelled by the power-management constructs. Thus, the proposed system-level power-management specification supports the clock and power gating, voltage and frequency scaling, operand isolation, and multiple supply voltages techniques in a highly abstracted form.

4.2. A new Power-Intent Specification Method

For the specification purpose, the special states (power states) are assigned to the system blocks and the switching among these states is enabled. In order the power management to be efficient, the concept of power domains is supported, based on the UPF standard. It enables to group together the system blocks that are always in the same power state. The required power states that are allowed at the system level are provided in Table 2.

At the system level (specification stage), the designer has to split the system into power domains. Therefore, each system block (component) has to be assigned to a power domain. If the block is not explicitly assigned to any power domain, it belongs to the so-called top power domain, i.e. it always operates in the *normal* power state. The power domain defines a set of power states, which the domain (and all the internal blocks) can reach. The power mode of the system is then represented by a combination of power states of the individual power domains. Since it is unlikely for the system to use all possible combinations, a common way is to specify the allowed combinations and thus to reduce the number of system power modes.

For the integration of the proposed power-management specification, we have chosen the widespread SystemC modelling. In order not to disrupt the support in the existing EDA tools, we have decided to implement the SystemC extension in a form of C++ library for the power-management constructs, which has been named PMS (Power-Management Specification). It is intended to be used alongside the SystemC library. The use of the library follows the rules of the C-

Table 2: Possible ESL Power States

Abstract power states	Power-state description
normal	The block (or more blocks in a single power domain) operates at the main supply-voltage level and at the basic operation speed (frequency). No explicit power-reduction technique is activated in this state.
diff_level#	The block operates at the voltage level different from normal and/or at the adjusted speed. # represents the ordinal number enabling the specification of several different levels. These states enable voltage and/or frequency scaling and usage of multiple fixed supply voltages in the design.
hold	The block stops its operation but remains powered. This state implies activation of clock gating and operand isolation power-reduction techniques.
off	The whole block is powered off. The power gating without state retention is activated in this state.
off_ret	While the block is powered off, the state is retained (state elements remain powered). This state represents activation of power gating with state retention.

based libraries – the header file has to be imported by the preprocessor command include. A portion of this library is illustrated in Algorithm 1.

Firstly, the allowed power states have to be specified. In order to check the syntax of power states at the compilation time, these are specified in a form of predefined macros. The number identifying the actual *diff_level* power state is passed to the macro as an argument, because it cannot be predefined in a static way. Since the SystemC is based on C++ language, we can use the class definition for modelling the power modes and power domains. Both of these classes use vector of string literals to keep the states specified to be part of this power mode or power domain. The power mode contains the power states for the individual power domains. The number of states in the power mode has to be always the same as the number of the power domains in the system. The order of the specified states is significant, i.e. the first state represents the state of the first specified power domain in the system, and so on. The power domain class contains the additional vector, called components, to keep the identifiers of components assigned to that power domain object. The constructors of these classes have "variadic" nature (i.e. they can take variable number of string arguments, at least one) in order the specification to be scalable. The problem is that the number of arguments is unknown, and thus this solution can easily cause memory violation. Therefore, we check for the last argument to be NULL. This is the constraint the designer has to keep in mind. To alleviate this constraint, we created additional macros that serve as constructors for these two kinds of objects (*PM* and *PD* macros). These macros take the arguments and call the actual constructors with additional NULL argument at the end of the argument

Algorithm 1: A Part of Power-Management Specification Extension Library

```
#define NORMAL "normal"
#define DIFF_LEVEL(i) "diff_level"#i
#define HOLD "hold"
#define OFF "off"
#define OFF_RET "off_ret"

#define PM(...) PowerMode(__VA_ARGS__, NULL)
#define PD(...) PowerDomain(__VA_ARGS__,NULL)

#define SetLevel(state, voltage, frequency)

static PowerMode POWER_MODE(NULL);

class PowerMode
{
    std::vector¡std::string¿ states;
    public:
    PowerMode(const char* state, ...);
};

PowerMode::PowerMode(const char* state, ...)
{
    va_list args;
    va_start(args, state);
    for (va_start(args, state); state != NULL; state = va_arg(args, const
     char*))
        this-¿states.push_back(state);
    va_end(args);
}

class PowerDomain
{
    std::vector¡std::string¿ states;
    std::vector¡std::string¿ components;
    public:
    PowerDomain(const char* state, ...);
    void AddComponent(std::string component);
};
```

list. By utilizing these macros, the designer does not have to explicitly provide the NULL argument. Thus, the designer has two options of object instantiation (shown below). *PowerDomain* objects are instantiated in analogous way.

$$PowerMode\ power\_mode1(OFF, NORMAL, OFF, NULL);$$
$$PowerMode\ power\_mode2 = PM(OFF, NORMAL, OFF);$$

Note, that by using the macro, adding NULL argument is considered a valid specification (makes no difference). This is a recommended instantiation style, because the designer cannot cause the memory violation (with or without the NULL argument). Although the power modes can be specified, we need to keep the current power mode of the system. For this purpose, the global state variable is created with a static name *POWER_MODE* (in order the high-level synthesis tool to recognize it). The object in this static variable is also an instance of the class *PowerMode*. Since there are originally no explicit power domains present, this instance has no states assigned. It is intended to be used in the functional model to specify changes in the power mode of the system (POWER_MODE = power_mode1). The method *AddComponent* of the *PowerDomain* class takes string identifier of some component and adds it to the power domain list. This way, the components to power domains assignment is specified and kept. The *SetLevel* macro is used for specification of a performance level (i.e. the actual voltage and frequency values) for the *normal* and *diff_level* states. The performance levels specification is required for detection whether two power domains operate at the same voltage or frequency levels. It is important for the high-level synthesis process, and therefore it has to be verified at the ESL. The actual voltages are also used for power estimation at the RTL. The ESL specification prevents the need to modify the synthesized UPF specification. This macro has three arguments. The first represents the power state – its name, the second represents the voltage value, and the last is the frequency value. The units can be also specified but are not required – defaults are V and MHz.

For illustration of the PMS library usage, we provide a simple case study, in which a microprocessor (*CPU*) is communicating with two memories (*RAM1*, *RAM2*) via a memory-management unit (*MMU*). The SystemC functional description of the microprocessor and memory modules has been based on the *mu0* and *ram0* VHDL entities available in [39]. The functionality is for such a showcase not important; therefore, it is omitted from the example. Only the power-management aspects are provided in the SystemC/PMS specification fragment illustrated in Algorithm 2.

Such a specification divides the *SoC* system into three power domains, namely *PD_CPU*, *PD_RAM1* and *PD_RAM2*. Each domain contains one component, corresponding to its identifier (see Figure 3). *MMU* is not modelled as a separate module, but as a SystemC process of the *SoC* module (dashed line in the figure). It is not assigned to any explicit power domain; therefore, it belongs to the implicit top power domain (i.e. its power cannot be managed).

According to the specification, the *CPU* component, as a part of the *PD_CPU* power domain, can operate in the *normal* and *off* power states. *RAM1* can operate in the *normal* state and its operation can be stopped in the *hold* state. *RAM2* can also operate in the *normal* state, but it can be also powered-down with its state retained (*off_ret*). There are four power modes specified: *PM1* for normal operation of each component (i.e. no power-reduction technique is activated), *PM2* for operation of *CPU* and *RAM1* (*RAM2* is powered down – i.e. power gating with state retention is activated for this component), *PM3* for operation of *CPU* and *RAM2* (*RAM1* is in *hold* in this mode – i.e. the clock

Algorithm 2: A Case-Study System Specification Example Using the SystemC/PMS Library

```
#include "systemc.h"
#include "pms.h"
SC_MODULE(SoC){
    //power-management declarations
    PowerDomain PD_CPU,PD_RAM1,PD_RAM2;
    PowerMode PM1,PM2,PM3,PM4;
    ...//part of the system omitted
    SC_CTOR(SoC){
        //components to power domains assignments
        PD_CPU.AddComponent("CPU");
        PD_RAM1.AddComponent("RAM1");
        PD_RAM2.AddComponent("RAM2");
        //power-domains states specifications
        PD_CPU = PD(NORMAL,OFF);
        PD_RAM1 = PD(NORMAL,HOLD);
        PD_RAM2 = PD(NORMAL,OFF_RET);
        //power-modes specification
        PM1 = PM(NORMAL,NORMAL,NORMAL);
        PM2 = PM(NORMAL,NORMAL,OFF_RET);
        PM3 = PM(NORMAL,HOLD,NORMAL);
        PM4 = PM(OFF,HOLD,OFF_RET);
        //initial power mode specification
        POWER_MODE = PM1;
        //performance-levels specification
        SetLevel(NORMAL,0.8,10);
        ...//port mapping and processes omitted
    }
};
```

gating and operand isolation are activated for this component), and *PM4* for a stand-by operation (*CPU* is powered down, *RAM1* is stopped, and *RAM2* is also powered down but the state is saved – i.e. power gating without state retention is activated for *CPU*, clock gating and operand isolation are activated for *RAM1*, and power gating with state retention is activated for *RAM2*). The initial power mode of the system is set to be *PM1*. And finally, the *normal* power state is defined to be powered by 0.8 V with the operation frequency of 10 MHz. The actual switching among power modes is modelled in the *MMU* process (not shown in Algorithm 2). When *CPU* is communicating with *RAM1*, the *PM2* power mode is assigned to the *POWER_MODE* variable. Analogously, *PM3* is activated when *RAM2* is used. And when the processing is finished, the system is switched into *PM4*.

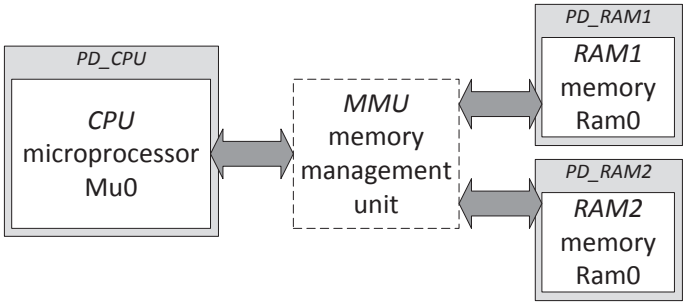

Figure 3: The abstract architecture of the case-study system.

4.3. Validation of the Specified Power Management

When the power intent is specified, it has to be verified for functional and structural correctness. However, the proposed specification method does not model the effect of power management on the system functionality. Therefore, functional correctness has to be verified at the RTL using the commonly used verification tools. At the early specification stage, there are several automatic verification steps to validate the structural correctness and completeness of the intended power management. The most crucial problem is to check consistency between various power-management constructs and consistency of power intent with the functional specification. The most basic verification steps comprise the syntactic and run-time checks. The syntactic checks are achieved using a C++ compiler, which reveals any syntactic error in the specification. During this step, the macro-based specification shows its value, because only the predefined abstract power states are valid. The run-time checks utilize the common programming verification approach, using conditional statements in the source code to detect erroneous operation. For example, this step can reveal a redundant assignment of the same power state to some power domain. However, the run-time checks can be used only during execution time; thus, the ESL model has to be executable (not always possible at early specification stage). The most robust verification step is the proposed power-management static analysis. It statically analysis the specified power intent and checks whether all the required data are present. It analyses the relations between power domains and power modes to detect any inconsistency. For example, it detects if some component instance is assigned to multiple power domains, if a power state used in some power mode is not specified for the corresponding power domain, or if some power domain does not contain any active power state.

For illustration of the capabilities of the proposed static analysis, the case-study system from the previous section is used. Since the provided specification is correct, *PM4* is modified according to the code below (the boldfaced text represents the change).

$$PM4 = PM(\mathbf{DIFF\_LEVEL(1)}, HOLD, OFF\_RET);$$

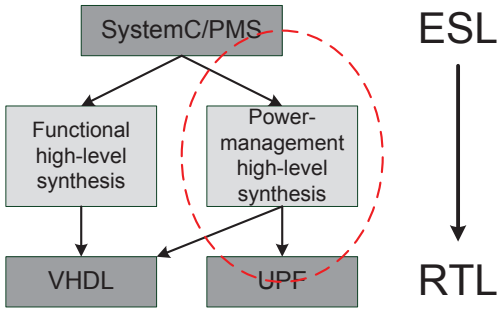

Figure 4: High-level synthesis process.

In this case, the static analysis would reveal that the *diff_level1* power state does not have a performance level assigned. It would also detect that *PD_CPU* is not allowed to operate in *diff_level1* according to the power-domain specification. Moreover, it would detect that the *off* state of *PD_CPU* is not used in any power mode, and therefore it would notify a designer that it is redundant.

Inconsistency errors can cause that the high-level synthesis cannot proceed; therefore, the static analysis is essential to be run prior to the synthesis. However, it can also be used as a separate step at early specification stages to drive a designer to the correct power-management specification. It must be noted that the proposed power-management static analysis is not supported by the existing C++ compilers (in contrast to the syntactic and run-time checks), it requires an additional verification tool implementing the proposed method. We have used our experimental tool called PMHLS, which implements all the proposed methods described in this paper.

4.4. Power-Management High-Level Synthesis

The specification described in the previous section serves as a starting point for the high-level synthesis process (manual of automated). It can be divided into two parts: the commonly used functional high-level synthesis (there exist many tools for automation of this process – summarized in Table 1) and the high-level synthesis for power management. It is clearly illustrated in Figure 4. In the figure, SystemC/PMS represents a SystemC model with the power management specified using the proposed PMS library. VHDL represents a functional model at the RTL level described in VHDL language. It can be described in Verilog as well – most EDA tools support both of them. UPF represents the synthesized power intent in the standard-based format.

During this process, the power-management related information is extracted from the functional model and augmented with the power-management elements that are implicitly required for a correct UPF specification at the RTL. The proposed power-management high-level synthesis also automatically generates the power-management unit, which is integrated into functional model at the RTL. A new entity is created that is required to be manually instantiated in the top-level entity of the synthesized system functional model.

Algorithm 3: Synthesis of Power Intent in UPF

Input: Extracted power-management related information from ESL specification.

Output: UPF specification.

(1) **Power domains** creation based on ESL specification.
(2) **Supply ports** creation based on performance-level assignments.
(3) Creation and connection of **supply nets** based on abstract power states.
(4) Creation of **power switches** based on states influencing voltage level.
(5) **Isolation setting** based on power states and power-domains relations.
(6) **Level-shifters setting** based on voltage states and domains relations.
(7) Application of **state retention** based on off_ret state.
(8) Creation of **ports states** and a **power-state table (PST)** based on power modes.

Thus, the power-management high-level synthesis can itself be divided into two processes: the UPF synthesis and the PMU synthesis. The UPF contains the operating part of power management, i.e. the power-management executive logic. The PMU represents the control part of power management, driving control signals for the power-management elements. The UPF synthesis steps are illustrated in Algorithm 3. A close explanation of these steps is provided in [33].

However, not all of the ESL power-management information is extracted from the functional model. The switching between power modes has been directly integrated into functional model in SystemC; therefore, it is just modified in such a way to be recognized by a functional high-level synthesis tool. This modification represents the change of the *POWER_MODE* variable type to enumerated, with the power modes identifiers posing as its enumerators. This ensures the support by the high-level synthesis tool (the enumerated type is supported in SystemC as well as in VHDL and Verilog). The switching between power modes at the RTL then actually represents the switching between unsigned integer values of the corresponding enumerators. The *POWER_MODE* value is an input of the PMU, which determines the power mode based on its binary representation. For clarification, an overview of the synthesized power management is provided in Figure 5. In VHDL, the original functional model contains power-management policy algorithm (*PMPA*) – i.e. decisions for switching between the power modes based on some system conditions. The *PMU*, also described in VHDL, contains the power-mode determination part (*PMD*) and the power-state machine (*PSM*). The *PMD* determines the control signals based encoding for any input power mode binary representation (a *POWER_MODE* value). The *PSM* is then handling the actual transition to

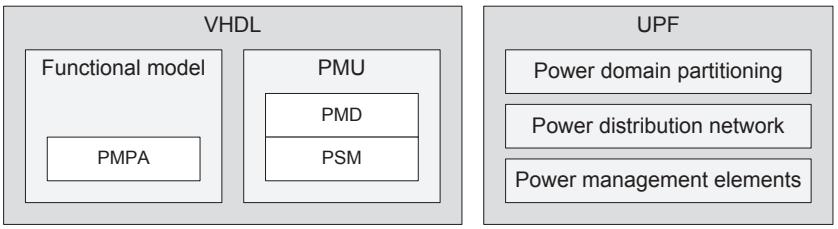

Figure 5: The synthesized RTL power management.

Algorithm 4: A Part of the Synthesized Power Intent in UPF

```
#power switch for PD_RAM2 domain
create_power_switch PD_RAM2_SW -domain PD_RAM2
 -input_supply_port { vin_0_8 VDD_0_8_net } -output_supply_port
 { vout VDD_PD_RAM2_net } -control_port { ctrl_sig1
 PMU/PD_RAM2_SW_ctrl1 } -on_state { state_0_8 vin_0_8 { !
 ctrl_sig1 } } -off_state { state_0_0 { ctrl_sig1 } } -ack_port { ap
 PMU/PD_RAM2_SW_ack { ! ( ctrl_sig1 ) } }  -supply_set
 PD_top.primary

#port voltage states
add_port_state PD_RAM2_SW/vout -state { state_0_8 0.8 }
add_port_state PD_RAM2_SW/vout -state { state_0_0 off }

#power-state table
create_pst PST -supplies { VDD_0_8_port VSS_0_0_port
 PD_CPU_SW/vout PD_RAM2_SW/vout }
add_pst_state internal1 -pst PST -state { state_0_8 state_0_0
 state_0_0 state_0_8 }
```

such a target power mode by generating correct control-signals sequences for the power-management elements specified in *UPF*. The *UPF* also contains the power-domains specification and the power-supply network.

There was a quite large UPF file synthesized for our simple case-study system from the previous sections. However, to illustrate its complexity, we at least provide a part of the generated power intent (Algorithm 4). It represents only the specification of a power switch enabling to power-down the *RAM2* memory, the specification of voltage states of the power-switch output port, and the specification of the power-state table with one implicitly specified (transient) power mode of the system.

A portion of the corresponding PMU design, synthesized in VHDL, is shown in the code fragment in Algorithm 5. It represents a single *when* statement of the PMU transition logic (the trickiest part of the PMU) specifying allowed transitions from the current power mode to a target mode via a next mode of the system. These modes are represented by vectors consisting of control-signals values for the power-management elements in UPF. The comments in the code

Algorithm 5: A Portion of the Synthesized Power-Management Unit in VHDL

```
case CurrentMode is
    when "000000" =>                    --(normal,normal,normal)
        if (TargetMode="000000") then
            NextMode <= "000000";       --(normal,normal,normal)
        elsif (TargetMode="010011") then
            NextMode <= "000010";       --(normal,normal,iso)
        elsif (TargetMode="000100") then
            NextMode <= "000100";       --(normal,iso,normal)
        elsif (TargetMode="111111") then
            NextMode <= "001110";       --(iso,iso,iso)
        end if;
    ...--other when statements omitted
end case;
```

provide the abstract meaning of these power modes. *iso* represents a state in which the isolation is activated in the domain (in case of the *hold* abstract power state or a transient state). Notice that the next mode is either directly the target mode, or an internal transient mode, generated for the power management to operate correctly. This is the last case, in which the target mode is *PM4* and the next mode is an internal mode activating isolation in all the domains. The synthesized PMU contains 15 such *when* statements and a single wrong value in some control signal can damage the system. Therefore, automation of this synthesis process is really helpful.

4.5. Post-Synthesis Power-Management Verification

The post-synthesis verification is oriented towards two aspects: satisfaction of the specification and functional verification. The first aspect is verified by checking whether the synthesized power management corresponds to the specification, i.e. whether the designer's power intent is preserved. The functional verification checks whether the system with the integrated power management functions as specified. It checks whether the power-management elements and the switching between power states or modes does not cause the system not to function properly. Since the functional aspects of power management have not been verified at the ESL, they have to be verified after the high-level synthesis. The advantage is that commonly used methods and EDA tools can be used at the RTL. Also, the effect of power management on the system power consumption is estimated with higher accuracy.

For the verification of the power-intent preservation after the synthesis, we have proposed a unique equivalence-checking method. It transforms both, the ESL power-management specification and the UPF power-intent specification into a common representation and checks the equivalence. The common representation is necessary because the UPF-based power states do not reflect the

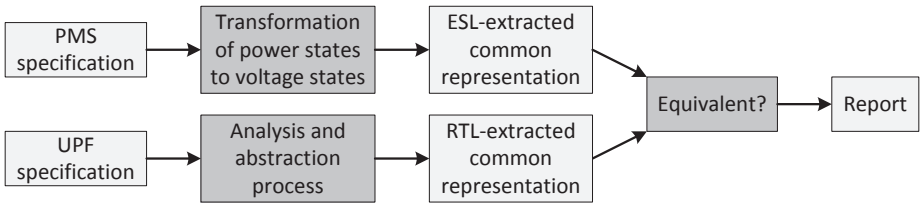

Figure 6: Illustration of the proposed equivalence-checking process.

Algorithm 6: An Example of the Generated Assertion in SVA

```
property PD_CPU_iso_while_off;
    @(posedge clk)
    $rose(DUT.PMU.PD_CPU_SW_ctrl1)
     |->DUT.PMU.PD_CPU_ISO_ctrl
    throughout ##[0:$] $fell(DUT.PMU.PD_CPU_SW_ctrl1);
endproperty
a_PD_CPU_iso_while_off: assert property (PD_CPU_iso_while_off)
 else
    $error("%t: Isolation disabled while powered-off!", $realtime);
```

operating frequency. Therefore, the ESL power states are transformed to the corresponding voltage states (analogously the power modes), what creates the ESL-extracted common representation. A quasi-reverse process to the high-level synthesis is used on the UPF specification, what creates the RTL-extracted common representation. These are then compared to each other to check the equivalence. An overview of the equivalence-checking process is illustrated in Figure 6.

The power-aware functional verification utilizes existing EDA tools, which statically check completeness and correctness of the generated UPF specification as well as the functionality through simulation. To ensure the synthesized PMU correctly generates the control-signals sequences, the PMU assertions are also automatically synthesized during the power-management high-level synthesis. These assertions enable to check a violation of correct sequences as well as the functional coverage during simulation. The reported error messages drive a designer towards the source of an error. Moreover, the provided coverage measurement notifies the designer what power modes have been verified and which of them yet needs to be exercised. An example of the generated assertion is shown in Algorithm 6. During the simulation, this assertion checks whether the isolation is enabled while the power domain is powered down. The code is provided in the SystemVerilog Assertion (SVA) language.

4.6. Methodology Adoption

For designers that are already using the functional high-level synthesis, the methodology adoption is really straightforward. The designer should empirically

divide the system architecture into power domains at the functional specification stage. Then, the power states have to be assigned to the domains and the basic power modes should be specified. Since the abstract power-management specification does not impact the ESL model function, the designer does not need to worry for the functionality to be disrupted. The proposed power-management static analysis helps to create a structurally correct and consistent specification. A suitable way to refine the preliminary ESL power-management specification is to simulate the design. The simulation reveals any gap in the specified power-mode switching, since the designer can observe the components state. For example, when the designer notices that the state of some power domain is *normal* but the internal components are inactive, a power-saving state (e.g. *hold* or *off_ret*) should be used for that power domain during such a period. During the high-level synthesis, the power-management static analysis and equivalence checking inform the designer if something goes wrong.

After the synthesis, the synthesized *PowerManagementUnit* entity is required to be instantiated inside the top-level functional design entity and the *POWER_MODE* and *clock* signals have to be mapped to the PMU. During functional verification at the RTL, the power analysis takes place and the power-management effect on the system functionality is verified. This step is a part of the current UPF design flow, i.e. it utilizes the existing methods and tools. After the power analysis, the designer can modify the ESL specification and repeat the high-level synthesis. The power-management high-level synthesis is much faster than the functional synthesis; therefore, it is a good practice to explore various power architectures upon the synthesized system architecture. However, if the power-management strategy algorithm or the power modes are modified, the corresponding functional components have to be resynthesized. A trade-off between power, performance, and area should be used to select the suitable system architecture with the respective power management.

5. Experimental Results

We provide the experimental results divided into three groups. The first group is focused on usefulness determination of the proposed abstract power-management specification method, specifically to determine simplification of the specification compared to the standardized UPF form. The second group of experiments is focused on verification of the proposed power-management high-level synthesis process. It is achieved by verifying the correctness of the synthesized RTL power management using the professional verification tool. The last experiment is focused on the case-study system used in the previous sections to illustrate the proposed methods. In this experiment, we show that the proposed methodology is usable and that the proposed methods can be used for low-power design.

5.1. Simplification of Power-Management Specification

For the evaluation of the proposed methodology, we have generated over ten thousand ESL power-management specification samples in SystemC and synthe-

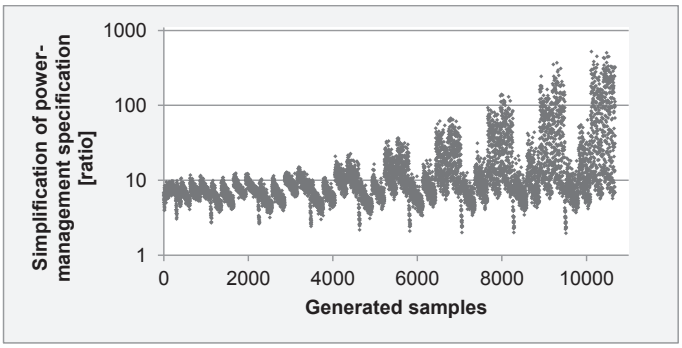

Figure 7: Comparison of ESL and RTL power-management specification complexity.

sized the standard UPF specifications using the proposed power-management high-level synthesis. The goal was to compare the UPF and SystemC power-intent specification complexity in terms of the number of characters required for the specification. It roughly represents the time that the designer needs to manually create the specification. Thus, such a comparison provides an illustration of the simplification, achieved using the proposed power-management specification method. To properly determine the simplification, we have scaled the number of power modes, the number of power domains, the average number of power states per power domain, and the average number of instances per power domain in the generated samples. The average values in the third and fourth parameters were used, because it is unusual to have the same number of power states or instances in power domains.

The results of the comparison of SystemC and UPF power-management specifications are illustrated in Figure 7. The ratio (the vertical axis) is provided in a logarithmic scale. It represents how many times fewer characters are required for the ESL power-management specification when compared to the RTL one. Thus, it actually represents the specification simplification (complexity reduction). For the generated samples, there was the simplification ratio achieved up to 521.8 times. In average, the ESL power management is approximately 16.8 times less complex than the equivalent specification in the UPF form. The minimum of achieved simplification was approximately 2 times. The experiment has proven that the use of the proposed ESL power-management specification is beneficial, because it is less complex for a designer to describe. Because of its more-concise form, it represents fewer opportunities to introduce a human error into the specification, and thus potentially shortens the time-consuming debugging process. Because of the enhanced abstraction used in the ESL power-management specification, it is also less complex to understand and actually use.

To determine which of the generated samples produced the highest specification simplification, we divided the samples into five groups, each with the same amount of the samples. The samples were sorted according the report-

Table 3: The Groups of the Generated Samples Based on the Simplification Ratio

Group	Power modes	Power domains	Power states	Instances
1	7.49	8.14	4.08	2.75
2	4.75	6.30	3.71	2.27
3	4.37	5.22	3.36	2.86
4	4.49	4.95	2.95	3.45
5	4.29	5.59	2.02	4.02

ed simplification ratio, and thus the first group contains the samples with the highest ratio and the fifth group with the lowest one. Table 3 reports average values for the monitored parameters of the samples in the individual groups. The results show that the highest simplification is achieved when specifying many power modes and domains, with a high amount of power states. The main reason is that the power states, which are specified using only the power-states macros in SystemC, produce the power-management elements in UPF (e.g. power switches, isolation, or level shifters). The high dependency on power modes is observed because the ESL specification contains only the explicit power modes, whereas the RTL specification also contains the intermediate power modes, synthesized to correctly switch between the explicit power modes. The conjunction with a high number of power domains produces even higher dependency, due to additional power states of additional supply ports in the UPF specification. The inconclusiveness resulted from the number of intermediate modes in UPF. With the lowering number, the simplification ratio is dependent on the specification of the explicit power modes only. Because of the abstraction from power-management elements at the ESL, the explicit modes specification represents a significant portion of the SystemC specification, but only a small portion of the UPF specification. The same is true regarding the power-domains specification. The average number of instances in power domains has negative impact on the simplification. The main reason is that the method *AddComponent* assigns an individual instance to a power domain in SystemC, whereas the list of instances is assigned to a power domain in a single statement in UPF. Thus, it takes a higher number of characters to assign an instance to a power domain in SystemC than in UPF. However, this parameter loses the simplification impact compared to the others (the instance to domain assignment is not as much significant), when a higher number of power modes and domains are specified.

5.2. Verification of the Synthesized Power Management

For verification of the synthesized power management, we selected fifteen power-management specification samples with various complexities. These samples were synthesized into the RTL form and verified in the professional verification tool Modelsim SE 10.2c. Specifically, we have used its UPF static analysis capabilities and its power-aware simulation option. For the simulation purpose, we have created the test-benches for the samples and pseudo-randomly switched

Table 4: The Power-Management Verification on Selected Samples

#	PM	PD	PS/PD	I/PD	ESL	UPF	PMU	Assertions	Coverage[%]
1	2	1	2	3	313	1680	3300	3863	100
2	3	2	2	2.5	500	2706	4452	4749	100
3	3	3	1.67	3	642	2850	4619	3194	100
4	5	3	4	2	643	6035	35205	25912	98
5	3	4	1.75	3	751	4658	8658	9488	100
6	3	4	2.25	3	760	4854	7017	10340	100
7	3	4	2.25	3	813	5678	10094	13433	97.5
8	10	3	3	2	862	5557	63304	17779	100
9	7	4	3.5	2	953	8778	142992	52330	81.1
10	7	5	3	2	1051	9399	131478	45150	96.1
11	10	4	4	2	1090	11605	586303	100721	85.5
12	10	5	2.4	2	1275	7443	199866	48111	89
13	5	5	3	5	1324	9039	107068	43833	91.2
14	3	10	2.2	2	1402	13397	67691	50911	99.2
15	5	10	2.1	3	1939	12232	214122	91620	82.4

between the power modes during the 10 ms runtime. The experiment data are provided in Table 4. In the left part of the table, which contains the parameters of the samples, *PM* denotes the number of power modes, *PD* is the number of power domains, *PS/PD* represents the average number of power states per power domain, and *I/PD* is the average number of instances per power domain. The right part of the table contains the number of characters for ESL power-management specification (*ESL*), UPF specification (*UPF*), power-management unit description (*PMU*), and synthesized assertions (*Assertions*). The last column (*Coverage*) reports the measured directive coverage, achieved using the generated coverage assertions for power modes, power states, and transitions.

The ESL power-management specification is scaling from 313 to 1939 characters, the UPF from 1680 to 13397 characters, and the PMU description from 3300 to 586303 characters. Thus, the samples provided sufficient specification-complexity variation. The samples were successfully synthesized and verified using the described approach with no error reported. It has proven the syntactical correctness and completeness of the UPF specifications, as well as the syntactical correctness of the synthesized PMUs and their readiness for functional verification. To achieve 100 % coverage, we would have to use another stimuli-generation approach than pseudorandom generation, such as directed tests. During the simulation, the generated assertions were also checking the sequences of control signals for power-management elements generated by the PMUs. It provides a higher assurance of the PMUs correctness. The summarized number of characters for *UPF* and *PMU*, along with *Assertions*, is much higher than the number of characters required for power-management specification at the ESL (up to approximately 641 times). Therefore, the proposed methodology is even more beneficial than reported in the first experiment (comparing only *UPF*, without *PMU* and *Assertions*). The potential errors, introduced during the manual description of the PMU and the assertions preparation, are avoided;

Table 5: Parameters of the Case-Study System

PM	PD	PS/PD	I/PD	ESL	UPF	PMU	Assertions	Coverage[%]
4	3	2	1	402	3291	9800	8217	100

Table 6: Power and Area Estimation of the Case-Study System

	Total Cell Area	Total Power [μW]
without power management	2107.933	99.2
with power management	2324.103	81
Difference	+10,26 %	-18,35 %

therefore, even much verification time is saved.

5.3. Case-Study System Evaluation

The usefulness of the methodology is shown using the experimental case-study system, mentioned in Section 4. For comparison of its complexity to the complexity of samples in Table 4, we summarize the same parameters of the case-study system in Table 5. Based on these data, the power-management specification simplification is quite obvious. Only 402 characters have been required for the ESL power-management specification. On the other hand, the RTL power management has taken 13091 characters ($UPF+PMU$). Moreover, the automatically generated assertions, used in verification, have taken another 8217 characters. Therefore, even for such a simple case study, the power-management specification was simplified 32 times (53 times when including assertions) by using the proposed methods.

To show that the proposed methodology can be indeed used for low-power design, we provide the power-estimation data for the case-study system, generated by the Power Compiler [40] (version K-2015.06-SP4) and the technology library NanGate_15nm_OCL [41]. The results in Table 6 show that the usage of power management resulted in increased area of the system by 10 %; however, the power is reduced by 18 %.

Since the modification of the ESL power management is really simple and straightforward, a designer can easily generate another design alternative. Because of the offered automation, the designer can explore various power architectures of the system in a short time, and thus select the most suitable design alternative (based on a tradeoff between power, performance, and area).

6. Comparison to Related Works and Discussion

The purpose of the proposed methodology and also of the related works is not to reduce more power than is possible in commonly used UPF/CPF at lower abstraction levels, but to deal with the complexity of systems and to increase productivity. We think that all of the proposed methodologies have presented in experiments that are usable to reduce power (as we have in the previous

section). It is difficult to objectively compare multiple methodologies to each other, since they are mostly evaluated based on some supporting implemented libraries or tools, which are not publicly available. However, based on the analysis, we can compare various aspects (concerning productivity increase) of the related methods and methodologies (including the proposed one) to illustrate the strengths and weaknesses of the proposed methodology. Such a comparison is summarized in Table 7.

In the table, the *Methodology* column represents a work similar to the proposed methodology. The next column (*Specification abstraction*) represents the abstraction level used for power-management specification. There are three values used: high – the specification does not contain low-level detail; medium – some aspects are specified with enough abstraction, but there are still some low-level details present; low – most of the specification contains low-level details, which are commonly used at RTL (in UPF/CPF). The *Single style* column illustrate whether the power-management aspects are specified directly in the system functional specification using the same specification style, or there is another specification file, language, and style used. *Supported techniques* refers to the power-reduction techniques that are supported by the methodology. We have used these acronyms for the techniques: CG – clock gating, OI – operand isolation, VS – voltage scaling, MV – multiple supply voltages, FS – frequency scaling, and PG – power gating. The *High-level synthesis* column refers whether the methodology supports automated transformation of the specified power intent into the standard form at lower abstraction levels. The next column (*Power analysis*) illustrates whether the power analysis is done at the system level (faster but lower accuracy) or at lower levels (slower but higher accuracy). The last column (*Verification*) refers whether the methodology offers some means to verify the specified power management.

The comparison in the table shows that there are several methodologies offering sufficiently high abstraction of the power-management specification. However, it comes at a price of no systematic connection to lower abstraction levels. The abstract specification is not based on standard concepts used at lower levels and thus it is very difficult or even impossible to derive equivalent power intent at the RTL. We have overcome this limitation by proposing a highly abstract specification that is based on UPF concepts, and thus an RTL power intent can be automatically synthesized. Most of the specification methods are based on separation of concerns principle. Therefore, the power intent is specified (or modelled) in a side-file using the specification style and language different from the functional model. It is important for design reuse and for making next generations of some system or its part at the RTL modelling stage (as a way to deal with design complexity to increase productivity). However, it is not suitable for system-level abstraction, where the abstract model should be simple enough. The common specification style makes easier to capture both functional and power intent. Therefore, we have chosen the single specification language and style for the abstract specification. The proposed methodology supports the highest amount of power-reduction techniques. The other methodologies target only some subset of the techniques. On the other hand, we have

Table 7: Comparison of Related Methods and Methodologies

Methodology	Specification abstraction	Single style	Supported techniques	High-level synthesis	Power analysis	Verification
The proposed methodology	High	Yes	CG, OI, VS, MV, FS, PG	Yes	RTL	Yes
PwClkARCH [1]	Medium	No	CG, VS, FS, PG	No	ESL	Yes
Ref. [2]	High	No	VS, FS, PG	No	ESL	Yes
COMPLEX [17]	High	No	CG, MV, PG	No	ESL	Yes
Ref. [18]	Low	No	OI, MV, PG	Yes	ESL	No
SCPower [19]	Low	Yes	OI, MV, PG	Yes	ESL	Yes
Ref. [26]	Medium	Yes	CG	Yes	RTL	No
LP-HLS [27]	Medium	No	CG, PG	Yes	RTL	No
Others	High	No	VS, FS, MV, PG	No	ESL	No

targeted all the techniques that we have evaluated as suitable for system abstraction level (see Section 4). Some of the analysed methodologies support the generation of standard power-intent specification at RTL, what speeds-up the development process. However, most of them are just rewriting the specification from some form to another because of the same amount of specified details. Others are rather limited regarding the supported power-reduction techniques. Only the proposed methodology supports a true high-level synthesis of power-management specification, which takes-in a highly abstract specification and automatically deduce the necessary lower-level details required for a UPF specification at the RTL. Moreover, the proposed method also automatically synthesizes a functional model (in VHDL) of the corresponding application-specific power-management unit, which drives the control signals in UPF. As already pointed out in Section 3, all of the analysed approaches that enable ESL power analysis require some sort of power annotation to the system model. It usually comes from previous implementations of the components (using the design-reuse concept), which is unsuitable for the top-down design approach targeted in our work. If the power values are not available, they use rough estimations, what makes the power analysis highly inaccurate. Therefore, we find the used high-level synthesis process with subsequent RTL power estimation as a more appropriate approach. Regarding the power-management verification capabilities, most of the methodologies support at least some basic checks of power-management aspects required for ESL simulation. However, in comparison to the analysed existing approaches, we have proposed the most robust power-management verification approach. Syntactic and run-time checks, along with the early static analysis, drive a designer to develop a complete and consistent power-management specification at the system level. Static analysis

during the power-management high-level synthesis ensures that all the required information is included and the equivalence checking ensures that the synthesis process preserves the power intent. Moreover, during the power-aware functional verification at the RTL (inevitable for power-architecture exploration), the assertion-based verification is used to verify the correctness of control sequences, generated by the synthesized power-management unit. It also enables to measure assertion-based coverage, ensuring that all the power modes have been exercised during the simulation. Most of these verification steps are automated; thus, the preparation and debugging verification processes are shortened.

7. Conclusions

This paper presents the novel methodology for low-power systems design, utilizing the system abstraction level and the high-level synthesis for design automation. The goal of this work was to eliminate the identified weaknesses of the similar methodologies (discussed in the previous section). The proposed methodology is directly based on the standard UPF design flow; thus, the existing methods and design-automation tools for verification, analysis, or power estimation can be used at lower levels. The key benefit of the proposed methodology is that the power-management specification is simplified, what directly corresponds to the trend of dealing with the complexity of modern designs through abstraction. The experiments have shown that the power-management specification using the proposed method at the system level is approximately 16.8 times less complex (in terms of a number of characters required for the specification) in average, compared to the UPF specification with the same power intent. Considering the automated synthesis of the application-specific power-management unit, which is not present in the system-level specification, even more complexity is reduced. The enhanced automation reduces the possibility of introducing human errors to the design, what reduces the verification burden, specifically the time-consuming debugging process. Although there is a design stage added into the UPF-based design flow, the overall development time is reduced by days or even weeks of manual work. Automated verification steps during the initial power-management specification help to create a structurally correct and complete specification. In the experimental results, we have shown that the proposed methodology is usable and useful in low-power systems design.

The work presented in this paper has opened the door for further enhancements of power management efficient adoption, which were not easily accomplished in other existing methodologies. The future work will be oriented towards a complete abstraction from power management during the system specification and its implicit automated introduction into the design. It will involve the automated partitioning of the system into power domains, automated assignment of power states to the domains, automated synchronization of clock-domain boundaries, and automated determination of suitable power mode according to the current requirements. It will simplify the specification even

more, what will help to design the power-efficient systems even by designers not familiar with the power-reduction techniques.

Acknowledgements

This work was supported by the Ministry of Education, Science, Research and Sport of the Slovak Republic within the Research and Development Operational Programme for the project "University Science Park of STU Bratislava", ITMS 26240220084, co-funded by the European Regional Development Fund. The work was also supported by the Slovak Scientific Grant Agency (VEGA 1/0836/16), the Slovak Research and Development Agency (APVV-15-0789), the Slovak Cultural and Educational Grant Agency (KEGA 011STU-4/2017).

References

[1] H. Affes, A. B. Ameur, M. Auguin, F. Verdier, C. Barnes, An ESL framework for low power architecture design space exploration, in: 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), IEEE, 2016, pp. 227–228. doi:10.1109/ASAP.2016.7760801.

[2] F. Mischkalla, W. Mueller, Advanced SoC virtual prototyping for system-level power planning and validation, in: 2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), IEEE, 2014, pp. 112–119. doi:10.1109/PATMOS.2014.6951882.

[3] A Practical Guide to Low Power Design: User Experience with CPF, Cadence Design Systems, 2012.
URL http://www.si2.org/?page=1061

[4] IEEE Standard for Design and Verification of Low Power Integrated Circuits, IEEE, 2013, IEEE Std 1801-2013.

[5] S. Carver, A. Mathur, L. Sharma, P. Subbarao, S. Urish, Q. Wang, Low-power design using the Si2 common power format, IEEE Design & Test of Computers 29 (2) (2012) 62–70. doi:10.1109/MDT.2012.2183574.

[6] International Technology Roadmap for Semiconductors: Design, ITRS, 2011.

[7] IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems, IEEE, 2015, IEEE Std 1801-2015.

[8] IEEE Standard for Standard SystemC Language Reference Manual, IEEE, 2012, IEEE Std 1666-2011.

[9] Intel Docea Power Simulator: A power and thermal virtual prototyping solution (2017).
URL http://www.intel.com/content/www/us/en/system-modeling-and-simulation/docea/power-simulator.html

[10] Stratus high-level synthesis: Industry's first high-level synthesis platform for use across your entire SoC design (2015).
URL https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/digital-design-signoff/stratus-ds.pdf

[11] Catapult high-level synthesis (2017).
URL https://www.mentor.com/hls-lp/catapult-high-level-synthesis/

[12] Platform Architect MCO: SoC architecture analysis and optimization for performance and power (2017).
URL https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html

[13] Synphony C Compiler: High-level synthesis from C/C++ to RTL (2017).
URL http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/SynphonyC-Compiler.aspx

[14] Vista Architect: System level design solution for performance and power (2009).
URL https://www.mentor.com/esl/vista/upload/vista-007d1f5d-41a7-41cd-97f4-06bb92d1a2eb

[15] Vivado design suite HLx editions - Accelerating high level design (2017).
URL https://www.xilinx.com/products/design-tools/vivado.html

[16] O. Mbarek, A. Pegatoquet, M. Auguin, Using Unified Power Format standard concepts for power-aware design and verification of systems-on-chip at transaction level, IET Circuits, Devices & Systems 6 (5) (2012) 287–296. doi:10.1049/iet-cds.2011.0352.

[17] K. Grüttner, P. A. Hartmann, K. Hylla, S. Rosinger, W. Nebel, F. Herrera, E. Villar, C. Brandolese, W. Fornaciari, G. Palermo, et al., The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration, Microprocessors and Microsystems 37 (8) (2013) 966–980. doi:10.1016/j.micpro.2013.09.001.

[18] J. Karmann, W. Ecker, The semantic of the power intent format UPF: Consistent power modeling from system level to implementation, in: 2013 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), IEEE, 2013, pp. 45–50. doi:10.1109/PATMOS.2013.6662154.

[19] K. Gagarski, M. Petrov, M. Moiseev, I. Klotchkov, Power specification, simulation and verification of SystemC designs, in: 2016 IEEE East-West

Design & Test Symposium (EWDTS), IEEE, 2016, pp. 1–4. doi:10.1109/EWDTS.2016.7807731.

[20] Y. Xu, R. Rosales, B. Wang, M. Streubűhr, R. Hasholzner, C. Haubelt, J. Teich, A very fast and quasi-accurate power-state-based system-level power modeling methodology, in: ARCS'12 Proceedings of the 25th International Conference on Architecture of Computing Systems, Springer-Verlag, Berlin Heidelberg, 2012, pp. 37–49. doi:10.1007/978-3-642-28293-5_4.

[21] H. Lebreton, P. Vivet, Power modeling in SystemC at transaction level, Application to a DVFS architecture, in: IEEE Computer Society Annual Symposium on VLSI, IEEE, 2008, pp. 463–466. doi:10.1109/ISVLSI.2008.71.

[22] T. Bouhadiba, M. Moy, F. Maraninchi, System-level modeling of energy in TLM for early validation of power and thermal management, in: DATE '13 Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, San Jose, CA, 2013, pp. 1609–1614. doi:10.7873/DATE.2013.327.

[23] S. Kaiser, I. Materic, R. Saade, ESL solutions for low power design, in: Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on, IEEE, 2010, pp. 340–343. doi:10.1109/ICCAD.2010.5653615.

[24] M. Streubühr, R. Rosales, R. Hasholzner, C. Haubelt, J. Teich, ESL power and performance estimation for heterogeneous MPSOCS using SystemC, in: Specification and Design Languages (FDL), 2011 Forum on, IEEE, 2011, pp. 1–8.

[25] W.-T. Hsieh, J.-C. Yeh, S.-C. Lin, H.-C. Liu, Y.-S. Chen, System power analysis with DVFS on ESL virtual platform, in: SoC Conference (SOCC), 2011 IEEE International, IEEE, 2011, pp. 93–98. doi:10.1109/SOCC.2011.6085102.

[26] S. Ahuja, A. Lakshminarayana, S. K. Shukla, Low Power Design with High-Level Power Estimation and Power-Aware Synthesis, Springer-Verlag, New York, NY, 2012. doi:10.1007/978-1-4614-0872-7.

[27] A. Qamar, F. B. Muslim, J. Iqbal, L. Lavagno, LP-HLS: Automatic power-intent generation for high-level synthesis based hardware implementation flow, Microprocessors and Microsystems 50 (2017) 26–38. doi:10.1016/j.micpro.2017.02.002.

[28] G. Kornaros (Ed.), Power Optimization in Multi-Core System-on-Chip. Multi-Core Embedded Systems, CRC Press, 2010. doi:10.1201/9781439811627-c3.

[29] M. Giammarini, M. Conti, S. Orcioni, System-level energy estimation with Powersim, in: 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS), IEEE, 2011, pp. 723–726. doi:10.1109/ICECS.2011.6122376.

[30] D. Greaves, M. Yasin, TLM POWER3: Power estimation methodology for SystemC TLM 2.0, in: J. Haase (Ed.), Models, Methods, and Tools for Complex Chip Design: Selected Contributions from FDL 2012, Vol. 265 of Lecture Notes in Electrical Engineering, Springer International Publishing, 2014, pp. 53–68. doi:10.1007/978-3-319-01418-0_4.

[31] S. Rigo, R. Azevedo, L. Santos (Eds.), Electronic System Level Design: An Open-Source Approach, Springer Netherlands, 2011. doi:10.1007/978-1-4020-9940-3_8.

[32] D. Macko, K. Jelemenská, P. Čičák, Power-management specification in SystemC, in: Proceedings of the 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, IEEE, 2015, pp. 259–262. doi:10.1109/DDECS.2015.16.

[33] D. Macko, K. Jelemenská, P. Čičák, Power-management high-level synthesis, in: The 23rd IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), IEEE, 2015, pp. 63–68. doi:10.1109/VLSI-SoC.2015.7314393.

[34] D. Macko, K. Jelemenská, P. Čičák, Verification of power-management specification at early stages of power-constrained systems design, Journal of Circuits, Systems and Computers 26 (08) (2017) 1740002. doi:10.1142/S0218126617400023.

[35] C.-M. Kyung, S. Yoo (Eds.), Energy-Aware System Design: Algorithms and Architectures, Springer Netherlands, 2011. doi:10.1007/978-94-007-1679-7.

[36] P. R. Panda, B. V. N. Silpa, A. Shrivastava, K. Gummidipudi, Power-Efficient System Design, Springer US, 2010. doi:10.1007/978-1-4419-6388-8.

[37] V. Venkatachalam, M. Franz, Power reduction techniques for microprocessor systems, ACM Computing Surveys 37 (3) (2005) 195–237. doi:10.1145/1108956.1108957.

[38] L. Benini, G. D. Micheli, Dynamic Power Management: Design Techniques and CAD Tools, Kluwer Academic Publishers, Norwell, MA, 1998. doi:10.1007/978-1-4615-5455-4.

[39] A. Rogers, Designing a simple system-on-a-chip in under 60 minutes with the mu0 microprocessor and Xilinx tools (2003).
URL http://www.ece.uah.edu/~lacasa/tutorials/mu0/mu0tutorial.html

[40] Power Compiler: Power optimization in Design Compiler (2017).
URL https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/power-compiler.html

[41] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, J. Michelsen, Open cell library in 15nm FreePDK technology, in: Proceedings of the 2015 Symposium on International Symposium on Physical Design (ISPD '15), ACM, New York, NY, USA, 2015, pp. 171–178. doi:10.1145/2717764.2717783.