Journal of Circuits, Systems, and Computers

World Scientific
www.worldscientific.com

Verification of Power-Management Specification at Early Stages of Power-Constrained Systems Design*

Dominik Macko†, Katarína Jelemenská and Pavel Čičák

Faculty of Informatics and Information Technologies,
Slovak University of Technology in Bratislava,
Ilkovičova 2, 84216 Bratislava, Slovakia
dominik.macko@stuba.sk
katarina.jelemenska@stuba.sk
pavel.cicak@stuba.sk

Received (29 September 2016)
Revised (21 December 2016)
Accepted (Day Month Year)

Nowadays, power is a dominant factor that constrains highly integrated hardware-systems designs. The implied problems of high power density, causing chip overheating, or limited power source in modern Internet-of-Things devices are most commonly dealt with the use of the dynamic power management. This method enables to use power-reduction techniques, such as clock gating, power gating, or voltage and frequency scaling. Since the adoption of power management is quite difficult in modern complex systems, there are new approaches evolving intended to simplify power-constrained systems design. We have also proposed such an approach, utilizing the system level of design abstraction and increased automation in the design process. In this paper, the proposed hybrid verification approach is described that represents an integral part of the suggested design methodology. It consists of formal and informal techniques, enabling the verification process to begin at the very early specification stage of the system development. Our approach helps a designer to create correct and consistent power-management specification and verifies whether the specified power intent is preserved after design refinement. The continuous automated verification steps can quickly find errors at early design stages and thus reduce the amount of design re-spins, which speeds-up the overall development process.

Keywords: Computer-aided design; energy efficient; hardware design; low power; power management; specification; verification.

1. Introduction

Low-power systems design is currently based on advanced power-reduction techniques that are typically applied on the functional design using a dedicated specification format, for example UPF (Unified Power Format).[1] UPF was intended for RTL (Register-Transfer Level) and lower-level modeling and enables to introduce power-management aspects to the functional design, usually modelled in HDL (Hardware Description Language). Although UPF has substantially simplified the low-power systems

* This work was presented at the 2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), April 20-22, 2016, Košice, Slovakia.
† Corresponding author.

development, it is not suitable for higher abstraction levels like ESL (Electronic System Level), where the modern design processes typically start nowadays. It can still be introduced into the design process at the RTL; however, the current complex systems are rapidly becoming manually unmanageable at this level. Therefore, new methods and methodologies have been developed to extend low-power design to the ESL, such as Refs. 2–13. Analyzing the strengths and weaknesses of the available current methods, we have proposed a novel methodology for low-power systems design that is solving some of the issues of current low-power systems design process e.g. insufficient abstraction, missing automation, or separated specifications. The methodology is based on the specification of an abstract power management at the ESL, as well as on the application of high-level synthesis to obtain the UPF-based RTL power-management model.[14–16]

In this paper, we have focused on the verification of the introduced power-management aspects at early stages of the design. The existing approaches related to this research area are analyzed in Sec. 2. In Sec. 3, the basics of the developed power management design strategy are summarized. The contributions of this paper regarding the proposed verification approach are explained in Sec. 4. The experiments, supporting the benefits of the contributions, are described in Sec. 5. Section 6 concludes the paper, summarizes the advantages of the proposed verification approach and indicates the possible future work in this research area.

2. Related Work

Currently, most of the available methods and methodologies, aimed at higher abstraction levels, target power-management abstraction and its introduction into the system-level design. Only some of them are also targeting the verification issues accompanying the power-management specification. In this section, the verification methods and approaches used in the available solutions are briefly analyzed.

The methodology proposed in Ref. 2 is aimed at the abstraction of several UPF concepts to the transaction level in order to enable power-architecture exploration at the system level. However, the power information must be manually annotated into the model. In this case, the verification of the power-management concept is based on automated generation of assume and guarantee assertions, that enable to report errors during the model simulation. Although the assertions generation process is hidden from the designer, the designer has to manually annotate the power information into the model. Another issue lies in the fact that this kind of verification does not check the power-management specification completeness, it only checks the correctness of interaction among components. In addition, the simulation-based verification is also time consuming. A very similar approach has been used in Ref. 3 targeting the clock management.

Several other methods[4–11] rely also solely on simulation-based verification. Although the offered virtual prototyping speeds-up the simulation compared to the RTL functional verification, there are other aspects (like specification completeness, its consistency with functional model *etc*.) that should also be verified; therefore, some additional verification

approaches are needed. The verification methodology proposed in Ref. 12 is also based on simulation. In this case, a semi-formal specification of use cases, power constraints and power states is used and the verification environment is then automatically generated based on this specification. Such automation speeds-up verification preparation. However, it also has the disadvantages mentioned before, namely the power-management specification completeness and its consistency with the functional model is not addressed. The authors in Ref. 13 do not use system-level power management verification. Instead, they analyze power based on the system-level simulation traces along with the functional model verification only at the RTL. In modern complex systems, the designers cannot rely solely on simulation to properly verify the design. The simulation-based verification must be combined with the formal methods to address the aspects that are not verifiable by simulation (e.g. completeness).

A more formal approach has been published in Ref. 17. The control-signals sequences represented by lower-level inter-domain assertions are automatically converted into the form usable in the abstract architectural properties of the system. The properties can then be formally proved. A similar approach is used in Ref. 18, which combines hardware and software power-management strategies. Although these approaches are definitely very helpful, they do not address the comprehensive verification of the introduced power-management aspects.

Analyzing the strengths and weaknesses of the available power-aware verification approaches, we have decided to address the verification completeness at early stages and the verification step, missing in the existing approaches, when refining the design between abstraction levels. We have suggested and integrated multiple verification steps into the proposed low-power design flow (this paper is an extended version of Ref. 19 describing more closely the proposed methods). The suggested power-aware verification methodology is based on a combination of formal and informal approaches and it relies on the following three key methods, representing our key contribution.

- *Power-management static analysis* — it is a unique method designed to verify the ESL consistency between the power-management and functional specifications. Additionally, the specification completeness is verified as well, thus leading the designer to the correct power management at the early stages of the design.
- *Power-intent equivalence checking* — this is an original method developed to verify that the power intent is preserved by the automated high-level synthesis process. The equivalence between the ESL and RTL specifications is checked formally and the process is fully automated. Thanks to that fact, this verification step is very fast.
- *Automated synthesis of power-control assertions* — inspired by the existing work[2,17], we have developed and integrated into the proposed power-management high-level synthesis a unique form of this method. The generated assertions are dedicated to functional correctness monitoring of the synthesized power-management unit. The assertions are synthesized based solely on the ESL power-management specification; therefore, no additional information need be provided by the designer (contrary to the existing methods).

3. Abstract Power Management in ESL Specification

In order to understand the proposed abstract power-management specification[14,15], three basic terms have to be clarified: power mode, power domain, and power state. ***Power state*** is an operation state of a system component that is defined by an operating frequency and a supply voltage. A collection of the system components that are always in the same power state can be grouped together into the same ***power domain***. That means, there are typically several power domains in the system and each system component belongs to exactly one power domain. Finally, the system ***power mode*** is defined by a combination of power states of the respective power domains. The system power management basic principle lies in the possibility to dynamically switch among the system power modes based on the current power requirements. The ultimate goal is to reduce energy consumption, while successfully completing the given task.

The power states usable at the system level are predefined (referred to as the abstract power states). A brief description of these states is provided in Tab. 1. The table also contains information about activation of specific power-reduction techniques in each abstract power state.

Table 1. The abstract power states overview.

Abstract power state	*Brief description*
normal	An operation state, in which no explicit power-reduction technique is activated. The components operating in this state are powered by the primary supply of the system and the operation frequency is the basic frequency of the system.
hold	This power state represents an activation of clock gating combined with operand isolation power-reduction techniques. All input signals (including the clock signal) of a component in this state are isolated, which prevents unnecessary switching activity in the component.
off, off_ret	These power states represent an application of power gating technique without and with the state retention, respectively. The power supply of a component in this state is cut off — i.e. the component is powered down (the values in memory elements of the component are saved in case of the *off_ret* state).
diff_level#	A group of abstract power states, where # represents a number to differentiate the states. Such a state enables to scale voltage and frequency levels of a component or to use multiple (fixed) voltages in the system. Each *diff_level* state represents a unique combination of voltage-frequency values.

The ESL power management is specified directly in the functional specification of the system, which simplifies the early specification (single specification language and style, in contrast to the HDL and UPF design at the RTL). The abstract power-management specification includes the following parts.

- *Component assignments* — The system components are assigned to specific power domains using their identifiers. The components always operating in the same power states are assigned to the same power domain.

- *Power domains* — An identifier for each domain is specified, along with a nonempty set of power states, in which the assigned components can operate.
- *Performance levels* — A unique voltage-frequency pair is specified for each active power state (*normal* or *diff_level*).
- *Power modes* — An identifier for each power mode is specified, along with a sequence of power states (one power state for each power domain, in the same order as they have been specified).
- *Power-management policy* — The switching among power modes is specified directly in the system function. It defines how and when the switching occurs.

To illustrate the usage of the abstract power-management specification, we specify one case of power management in a simple SoC, called *system0*. This system has four components, one instance of microprocessor *mu0* (component instance is called *CPU*) and three instances of memory *ram0*, called *M1*, *M2*, and *M3* (VHDL descriptions of these components are available in Ref. 20). These components are interconnected via the memory controller (*MCU*) that selects a memory component based on the value in the address bus. The abstract power-managed architecture of this case-study system is illustrated in Fig. 1.

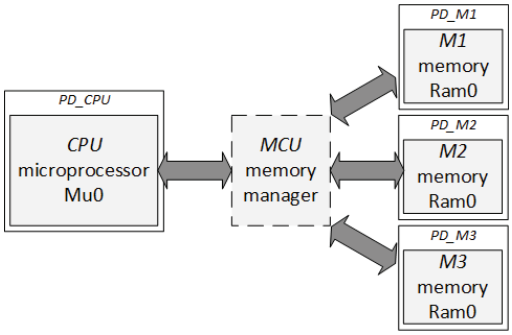

Fig. 1. The case-study system abstract power-managed architecture.

We have described this system in SystemC in order to mimic top-down design and to use the developed power-management specification. In our case, let us assume that the power consumption of these components can be managed separately (i.e. each component is assigned to the dedicated power domain: *PD_CPU*, *PD_M1*, *PD_M2*, and *PD_M3*, respectively). Let the system operate in five power modes — *PM1*, *PM2*, and *PM3* for full-speed communication between *CPU* and individual memories, *PM4* for low-power operation of *CPU* and *M1* (slower), and *PM5* for the standby mode of the whole system. It is supposed that the *MCU* behavior is specified in a SystemC process of system0, not as a separated component. Therefore, *MCU* is surrounded by a dashed line in the figure. At the system level of abstraction, the power is controlled in the functional specification, in the *MCU* process in this case. The specification code fragment in SystemC-based power management is provided below.

```
SC_MODULE(system0){
   //system components
   mu0 CPU;
   ram0 M1, M2, M3;

   //power-management declarations
   PowerDomain PD_CPU,PD_M1,PD_M2,PD_M3;
   PowerMode PM1,PM2,PM3,PM4,PM5;

   ...//part of the design omitted

 SC_CTOR(system0): CPU("CPU"), M1("M1"), M2("M2"), M3("M3")
 {
   ...//port mapping and processes omitted

   //components to power domains assignments
   PD_CPU.AddComponent("CPU");
   PD_M1.AddComponent("M1");
   PD_M2.AddComponent("M2");
   PD_M3.AddComponent("M3");

   //power-domains states specification
   PD_CPU = PD(NORMAL,DIFF_LEVEL(1),OFF);
   PD_M1 = PD(NORMAL,HOLD);
   PD_M2 = PD(NORMAL,HOLD,OFF_RET);
   PD_M3 = PD(NORMAL,HOLD,OFF_RET);

   //power-modes specification
   PM1 = PM(NORMAL,NORMAL,HOLD,HOLD);
   PM2 = PM(NORMAL,HOLD,NORMAL,HOLD);
   PM3 = PM(NORMAL,HOLD,HOLD,NORMAL);
   PM4 = PM(DIFF_LEVEL(1),NORMAL,OFF_RET,OFF_RET);
   PM5 = PM(OFF,HOLD,OFF_RET,OFF_RET);

   //initial power mode selection
   POWER_MODE = PM1;

   //performance-levels specification
   SetLevel(NORMAL,1V,50MHz);
   SetLevel(DIFF_LEVEL(1),0.9V,5MHz);
 }
};
```

In this example, *CPU* can operate in three power states (*normal*, *diff_level1* and *off*). It operates in *normal* state in the *PM1*, *PM2*, and *PM3* system modes, its voltage and frequency are scaled down in *PM4* to save power and it is completely powered-down in the standby mode (*PM5*). The *M1* memory is selected in the *PM1* and *PM4* modes, in which it operates in *normal* power state. In the standby mode, it is put into the *hold* power state to save at least dynamic power. The memories M2 and M3 are each selected in *PM2* and *PM3* respectively (the *normal* state when selected and *hold* when not), and they are powered-down in *PM4* and *PM5* with the state retention activated. We may notice that *PM1* is the default system power mode, since it is assigned to the *POWER_MODE* variable in the SystemC module constructor. The switching between power modes is then specified in the functional part of the design (i.e. some SystemC process) by assignment of other specified power mode to the *POWER_MODE* variable.

The abstract power-management specification, integrated with functional ESL specification, represents the basis of the proposed methodology for low-power systems design, which extends the traditional UPF-based design flow to the ESL. The proposed low-power design flow is illustrated in Fig. 2.

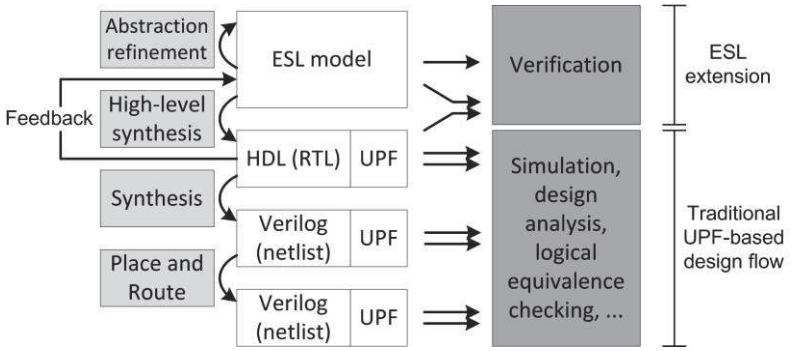

Fig. 2. The proposed low-power systems design flow.

The design starts by the specification at the system level of abstraction (ESL mode). The initial specification contains functional design as well as power-management aspects in the same specification model. The abstraction-refinement process is then used to develop the specification to the abstraction level, where the high-level synthesis tools can generate the RTL model of the specified system. The RTL model consists actually of two parts: the system functional model in an HDL and the power intent specification in UPF. The functional model is supposed to be generated by a commonly used synthesis tool, supporting the ESL modelling (e.g. Stratus[21], Catapult[22], or Synphony C Compiler[23]). The UPF specification at the RTL is generated using the proposed power-management high-level synthesis process[16] (see Fig. 3). Along with the UPF specification, the power-management unit (PMU), which will be driving control signals for power-management elements in the UPF, is automatically synthesized into the functional model in HDL. Thus, the complex RTL power management is designed much faster using the proposed method.

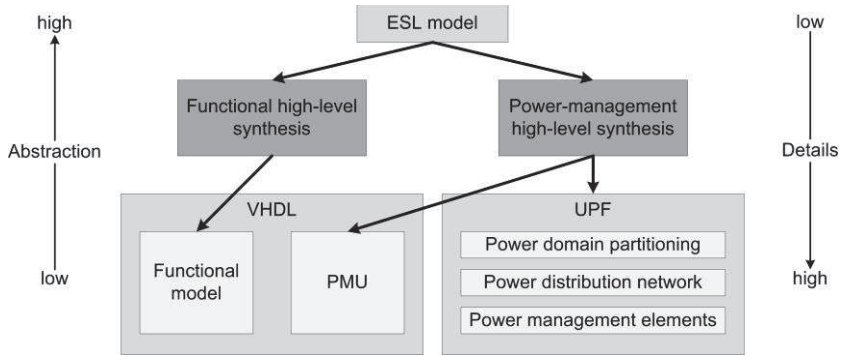

Fig. 3. The extended high-level synthesis process.

At this stage, the currently used design-automation tools can be used for design analysis and further system development as supported by the standard UPF-based low-power design flow. The verification steps in the extended part of the design flow represent its crucial part and they will be discussed in more detail in the following section.

4. The Proposed Verification Approach

The specified power management has to be verified for functional and structural correctness and completeness.[24] For a designer, it is most beneficial to accomplish this verification step as soon as possible in order to avoid design re-spins. However, the model might not be executable at the specification stage; therefore, a formal approach is suitable for such verification and validation. We combine the compilers functionality, which is able to check syntactical correctness of the specification, and the power-management static analysis detecting functional and structural inconsistencies in the abstract power-management specification at the ESL.

Afterwards, it is important to verify the correct functionality of the power-managed system, usually verified by functional simulation. In the developed low-power design flow, this verification step has to be done after the high-level synthesis, because the abstract power-management specification does not model the effects on functionality. For this purpose, an existing professional power-aware tool, such as Modelsim, can be used to simulate the functional HDL model along with the synthesized power intent in UPF. As mentioned in Ref. 25, the power-management elements (specified in UPF) are often a rich error-source, and therefore should be verified for all operating modes. Due to the high-level synthesis automation offered by the proposed methodology, we are able to avoid potential human errors in the power-management insertion at the RTL. The low-level power-management logic (such as power switches, isolation, retention, or level shifters) is completely abstracted for a designer — it is automatically (i.e. correctly) added to the specification during the power-management high-level synthesis.

However, to assure that the power intent is the same after the high-level synthesis, the equivalence between the synthesized UPF specification at the RTL and the abstract

power-management specification at the ESL should be checked (this is known as the equivalence checking verification technique). The functional description of the synthesized PMU is ideal for the assertion-based verification approach (ABV). The automatically generated assertions can verify the control-signals sequences driven by the PMU, but they can also be used for the functional-coverage measurement.

An overview of the proposed verification processes is provided in Fig. 4. We propose to use a combination of verification techniques (the dark-grey color) at various design stages, which enables to verify the low-level power-related logic based only on the ESL specification. The proposed abstraction and automation simplifies the complex verification process (especially the preparation and debugging steps), and thus can save a lot of time.

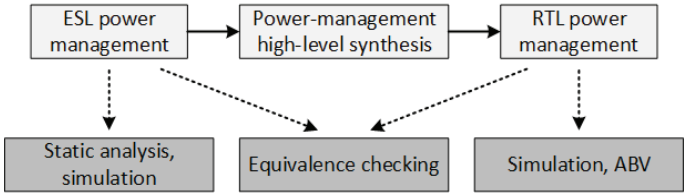

Fig. 4. The verification process overview.

4.1. *Syntax checking*

An automated verification of the power-management specification syntax is done by compilers at the compilation time (before the ESL simulation starts). The specification syntax consists of the predefined keywords, which drive a designer to specify power management correctly, using valid statements. In addition to pure syntax checking, in HSSL[14] (Hardware-Software Specification Language), the syntactical rules also assure that at least one power domain has to be specified if the power-domains section is present in the specification. In addition, each power domain is required to have at least one power state. The syntactical rules ensure that only the predefined abstract power states are valid, because they are defined as the language keywords. In addition, it is also assured that at least one power mode is specified. In all these cases, the HSSL compiler detects an error and does not compile the specification.

This kind of syntax checking is not supported in the SystemC/PMS[15] specification. The power management is specified using the macros defined in the PMS extension library. If the designer uses these macros, the compiler then ensures the correct specification. The C++ modelling, which is integrated in the PMS library, ensures that only the previously specified power mode is selected as the current power mode, and that the component instances can be assigned only to the previously specified power domains. These issues cannot be detected in HSSL syntax checking. Because the specification must be verified thoroughly, the issues that cannot be revealed in this verification step (in both the HSSL and SystemC) must be verified by another approach — during an execution time or by the static analysis.

4.2. *Run-time checking*

In SystemC/PMS specification, the run-time checks (during the ESL simulation) are used to detect some errors that could not be detected by a compiler. The run-time checking implements the conditional statements that notify the designer, if an error is detected, what part of the power management is incorrectly specified. The implemented run-time checking verifies whether the valid abstract power states are assigned to the power domains and to the power modes. As long as the predefined power-state macros are used, this condition cannot be violated. However, the designer has a choice to use or not to use the macros; therefore, it must be verified. The run-time checking also detects an error if the designer specifies some power state multiple times in some power domain. Although, these errors can be detected by the static analysis described in the next section, it requires an additional verification tool. The proposed run-time checks increase verification detection capabilities without any additional tool (it requires only a common compiler and execution of the model).

4.3. *Power-management static analysis*

The static analysis verification form is mainly focused on the power-management inconsistencies that could not be revealed in the previous verification steps. It formally analyses the specified constructs (e.g. identifiers, assignments), especially those related to the power management. The static analysis can be used from the beginning of the system specification, notifying the designer about the inconsistencies. However, it is also integrated into the high-level synthesis process, which cannot proceed unless the power-management specification is complete.

To be more specific, the proposed static analysis is able to verify formal requirements regarding the abstract power management. To provide an example, it verifies that all power modes have the correct number of power states — it has to correspond to the number of specified power domains in the system. As another example, the static analysis notifies the designer if some power domain contains no active power state or if some component instance is assigned to several power domains. It notifies the designer about the location of redundant specification constructs, such as unused power states, unused power modes, or multiple power modes with the same sequence of power states. The proposed power-management static analysis is very helpful in early versions of the specification, because it drives the designer to create a correct and complete specification, which can significantly shorten the debugging time.

The power-management specification in the case study in Sec. 3 is correct. However, to illustrate the detection capabilities of the proposed static analysis, we can modify the specification according to the code fragment below.

```
PD_CPU.AddComponent("CPU");
PD_M1.AddComponent("M1");
PD_M2.AddComponent("M2");
```

```
PD_M3.AddComponent("M3");
PD_CPU = PD(NORMAL,DIFF_LEVEL(1),OFF);
PD_M1 = PD(NORMAL,DIFF_LEVEL(1),HOLD);
PD_M2 = PD(NORMAL,HOLD,OFF_RET);
PD_M3 = PD(NORMAL,HOLD,OFF_RET);
PM1 = PM(NORMAL,NORMAL,HOLD,HOLD,NORMAL);
PM2 = PM(NORMAL,HOLD,NORMAL,HOLD);
PM3 = PM(NORMAL,HOLD,HOLD,NORMAL);
PM4 = PM(DIFF_LEVEL(1),NORMAL,OFF_RET,OFF_RET);
PM5 = PM(OFF,HOLD,OFF_RET,OFF_RET);
POWER_MODE = PM1;
SetLevel(DIFF_LEVEL(1),0.9V,5MHz);
```

The highlighted text represents the detected problems. The static analysis would detect that the *diff_level1* power state of the power domain *PD_M1* is not used in any specified power mode (the power state of *PD_M1* corresponds to the second position of states in a power mode). The designer would be notified that the performance level for the *normal* state is not assigned. An error would be reported regarding the incorrect number of power states in *PM1*. Let's say that the mode *PM4* is never assigned to the *POWER_MODE* variable in the functional part of the specification (i.e. the module constructor or some process). The static analysis would detect that this mode is redundant. Because of that, the notification would be generated that the *diff_level1* power state of *PD_CPU* is not used.

4.4. *Power-management equivalence checking*

An overview of the proposed equivalence-checking process is illustrated in Fig. 5.

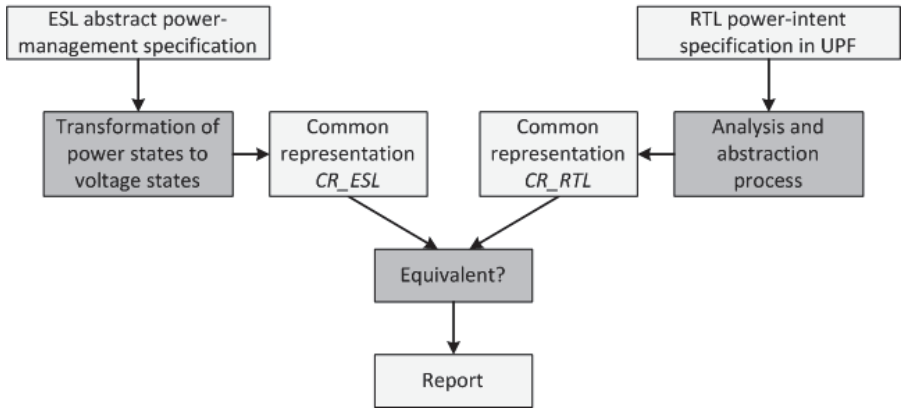

Fig. 5. The equivalence-checking process.

This verification step takes place after the RTL power management in UPF form has been synthesized using the proposed power-management high-level synthesis[16]. It

verifies the equivalency between the RTL power intent extracted from the UPF specification and the original ESL power-management specification. However, since these specifications use different level of abstraction (i.e. they contain different amount of details), a common representation has to be used for the comparison. That means, both specifications are translated into this common representation, in which the power states are represented by the corresponding voltage levels. The reason is that the UPF specification does not contain frequency aspects, because they are contained in the functional HDL model. We have formally defined the common representation (*CR*) as a tuple

$$CR = (PDI, PDS, PM), \tag{1}$$

where *PDI* is a nonempty finite set of pairs *(IDd, I)*, where *I* is a nonempty finite set of component instances assigned to the power domain represented by an identifier *IDd*; *PDS* is a nonempty finite set of pairs *(IDd, VS)*, where *VS* is a nonempty finite set of voltage states (i.e. voltages) enabled in the power domain represented by an identifier *IDd*; and *PM* is a nonempty finite set of pairs *(IDm, S)*, where *S* is a nonempty finite sequence of voltage states corresponding to the power states of power domains that represent the power mode with an identifier *IDm*.

The proposed method is checking two equivalence aspects, namely the power-management structural equivalence and the power-intent equivalence. The first one actually verifies that all power-management constructs, specified at the ESL, have been synthesized into the UPF specification. The second one verifies that the power intent is the same in both specifications. It means that the unused power-management constructs do not need to match, only the resulting impact is important.

The equivalence-checking process is starting by analysis of both specifications during which their common representations are generated. The common representation *CR_ESL* is extracted from the ESL power management quite easily, the power states are simply translated to the corresponding voltage states based on the performance-level assignments and the lists representing *VS* and *S* are filled in. For the power-intent equivalence checking, the common representation is extracted from the optimized ESL specification. It means that the redundant parts of the abstract specification are removed based on the optimization decisions of the high-level synthesis.

The *CR_ESL* extraction can be explained using the case-study ESL description in Sec. 3. The power mode *PM1* specifies that *CPU* and *M1* operate at the supply voltage of *1 V*. Also, the *M2* and *M3* components are supplied by *1 V*, because the *hold* state uses the lowest voltage ever used in the domain. Similarly, *PM4* specifies that *CPU* operate at the supply voltage of *0.9 V*, *M1* at the supply voltage of *1 V*, and *M2* and *M3* are powered-off (i.e. the voltage of *0 V*). Analogously, the voltage states can be deduced for the other specified power modes. This translation of power states in power modes to voltage states in the *CR_ESL* is illustrated in Tab. 2. The power domains with instances and voltage states are extracted into the common representation analogously.

Table 2. Power states translation to voltage states for power modes in the common representation.

Abstract power- modes specification	*Voltage states in common representation*
PM1 = PM(NORMAL,NORMAL,HOLD,HOLD);	PM1, (1.0, 1.0, 1.0, 1.0)
PM2 = PM(NORMAL,HOLD,NORMAL,HOLD);	PM2, (1.0, 1.0, 1.0, 1.0)
PM3 = PM(NORMAL,HOLD,HOLD,NORMAL);	PM3, (1.0, 1.0, 1.0, 1.0)
PM4 = PM(DIFF_LEVEL(1),NORMAL,OFF_RET,OFF_RET);	PM4, (0.9, 1.0, 0, 0)
PM5 = PM(OFF,HOLD,OFF_RET,OFF_RET);	PM5, (0, 1.0, 0, 0)

The common representation extraction from the UPF power-intent specification (referred to as *CR_RTL*) is more challenging, it uses a quasi-reverse process to the high-level synthesis. The supply voltage for the *normal* power state is deduced from the specified state of the supply port that is connected to the primary supply net of the implicitly synthesized top power domain, called *PD_top*. Other supply ports with specified states (such as *VDD_0_9_port*) imply that multiple voltages are used in the design, and thus that some *diff_level* state has been used in the abstract specification. The specified states of the power-switch output port (e.g. *PD_CPU_SW/vout*) are used to determine other possible power states of the corresponding power domain (i.e. the power-switch output port is connected to the primary net of that domain). The power modes in the UPF power-state table specify the states of the supply ports, which are used to determine the voltages for individual power domains in these power modes. An example of the extraction of a voltage state from the synthesized UPF specification, equivalent to the case study in Sec. 3, is provided in Tab. 3. The highlighted text represents the important parts of the UPF specification used for the voltage-state determination.

Table 3. A voltage state deduction from the UPF specification.

Fragments of the UPF specification	*Deduction*
set_domain_supply_net PD_CPU\ -primary_power_net VDD_PD_CPU_net ...	The *PD_CPU* power domain is powered by the power-supply net of *VDD_PD_CPU_net*.
connect_supply_net VDD_1_0_net\ -ports { VDD_1_0_port }	It is checked if the supply net is connected to a top-level supply port.
create_power_switch PD_CPU_SW\ -domain PD_CPU\ -output_supply_port\ { vout VDD_PD_CPU_net }...	Otherwise, it is checked if the supply net is connected to the output port of a power switch in that domain. It is connected to the *PD_CPU_SW/vout* port.
create_pst PST –supplies\ { VDD_1_0_port VDD_0_9_port\ VSS_0_0_port PD_CPU_SW/vout\ PD_M2_SW/vout PD_M3_SW/vout }	The *PD_CPU_SW/vout* port has the fourth position in the power-state table.
add_pst_state PM4 -pst PST –state\ { state_1_0 state_0_9 state_0_0\ state_0_9 state_0_0 state_0_0 }	For the *PM4* power mode, the port state of *state_0_9* is located in the fourth position.
add_port_state PD_CPU_SW/vout\ -state { state_0_9 0.9 }	The *state_0_9* state of the *PD_CPU_SW/vout* port corresponds to 0.9 V.

For simplicity, many commands are omitted from the UPF specification in the provided code fragments. The UPF files can be pretty large in length. The example just provides a simple insight into the complex *CR_RTL* extraction process.

After the common representations are created, the lists of power domains are compared first. *CR_ESL* has to contain each power domain in *CR_RTL* (identifiers are used for comparison), except for the top domain (implicitly added by the high-level synthesis). *CR_RTL* also has to contain each power domain in *CR_ESL*. In addition, the corresponding lists of assigned component instances and voltage states have to coincide. Next, the power modes with specific voltage-states sequences are compared analogously to the power domains. It is common that the UPF specification contains many intermediate modes, representing intermediate states added for correct switching among power states. However, these must comply with some specified power mode — there cannot be any new combination of voltage states. Therefore, the combinations of voltage states are checked and the redundant intermediate modes are not added to *CR_RTL*.

The result of this verification step is a comprehensive report, containing equivalence status along with error messages and information about the source of an error, which can speed-up the debugging process.

4.5. *Assertion-based verification*

The abstract power-management specification provides sufficient amount of details to automatically synthesize the assertions about RTL power management control-signals sequences generated by the PMU. As the first step, we synthesize the *sequence* statements that encode each power state of each power domain using the control signals for the power-management elements synthesized in UPF. The control signals for a power domain depend on the used supply voltages in the domain, the used clock frequencies, and the need for isolation and retention. An example of the synthesized *sequence* statement is provided below using the SystemVerilog Assertion language. In this example, the *off* power state is defined by the combination of control signals, in which the clock-control signal is *false*, the two power-switch control signals are *true* and *true*, and the isolation-control signal is *true*.

```
sequence PD_CPU_off;
   !DUT.PMU.PD_CPU_CLK_c1 && DUT.PMU.PD_CPU_SW_c1 &&
   DUT.PMU.PD_CPU_SW_c2 && DUT.PMU.PD_CPU_ISO_c;
endsequence
```

Secondly, we synthesize the *sequence* statements that control the correct power-state transitions. Besides the specified power states, these sequences also include the intermediate power states, through which the components must be switched to function correctly (e.g. isolation or retention states). These power-state transition sequences are determined based on the ESL power-mode specification. An example of such a sequence is provided below. When the *PD_CPU* power domain is transitioning from the *normal*

power state (1 V, 50 MHz) to the *diff_level1* power state (0.9 V, 5 MHz), it goes through an intermediate state, in which the frequency is lowered but the voltage remains the same as in normal state (1 V, 5 MHz). The operator *##1* means that in the next clock cycle the following value becomes true.

```
sequence PD_CPU_normal_diff_level1;
   PD_CPU_normal ##1 PD_CPU_inter_1_0_5_0 ##1
   PD_CPU_diff_level1;
endsequence
```

In addition, the power-state encoding sequences are used for definition of the power-mode sequences, because it is easier for a designer to monitor the system power modes. An example of such a *sequence* statement is provided below.

```
sequence mode_normal_normal_iso_iso;
   PD_CPU_normal and PD_M1_normal and PD_M2_iso and
   PD_M3_iso;
endsequence
```

The synthesized *sequence* statements (encoding as well as transitioning) are then used to measure the coverage of the generated power management during the RTL functional verification. An example of the explicit coverage directive is provided below.

```
c_PD_CPU_off: cover property (@(posedge clk)PD_CPU_off);
```

A simulator supporting assertions counts if and how many times a property, represented by a sequence, has become true. In this way, the designer can track the verification progress during the simulation or can even modify the test-bench to cover unverified power modes.

Besides the mentioned coverage assertions, we synthesize assertions monitoring an incorrect behavior of the PMU. The possible illegal sequences of control signal values are divided into seven classes — five classes reflect Ref. 17 and two more have been added (third and fourth) to cover the illegal sequences completely. Since these sequences are illegal (i.e. they should not occur), the properties representing these sequences are asserted in the opposite way. If such a property evaluates to true, it represents an erroneous behavior. These classes are briefly described below.

(1) *Illegal restoration before power-on* — the values cannot be restored before the domain is powered-on. An example of such a property is shown below. The system function *$fell()* returns true if the argument value changes from true to false, the operator |-> represents the implication, *throughout* means that the previous Boolean value needs to hold true until the following sequence becomes true, and the operator *##[1:$]* means that the following value will eventually become true.

```
property PD_M2_i_restore_on;
   @(posedge clk)
   $fell(DUT.PMU.PD_M2_RET_c) |-> !DUT.PMU.PD_M2_RET_c
   throughout DUT.PMU.PD_M2_SW_c1) ##[1:$]
   $fell(DUT.PMU.PD_M2_SW_c1);
endproperty
a_PD_M2_i_restore_on: assert property (PD_M2_i_restore_on)
   $error("Restored before powering on!"); else;
```

(2) *Illegal power-off before retention* — for the domains in which the state should be retained, the power cannot be turned off before the retention is activated.
(3) *Illegal de-isolation before power-on* — the isolation should not be disabled before the power is turned on.
(4) *Illegal power-off before isolation* — before the power is turned off, the isolation at the domain boundary should be activated.
(5) *Illegal retention before isolation* — if the retention is needed, it cannot precede the isolation procedure.
(6) *Illegal de-isolation before restoration* — the isolation should not be disabled before the state values are restored.
(7) *Illegal performance-level transition* — when a voltage-frequency pair is changed by some power-state transition, the frequency can be raised only after the voltage is raised and the voltage can be lowered only after the frequency is lowered.

In addition to the assertions checking the illegal sequences, we define another three classes of control-signals assertions. The first class monitors whether isolation is enabled during retention period, the second one controls that the isolation is enabled while the components are powered-down, and the third class verifies whether the retention is activated during the power-down period (if it should be). These are generated in the following way (the system function $rose() returns true if the argument value changes from false to true).

```
property PD_M2_iso_while_ret;
   @(posedge clk)
   $rose(DUT.PMU.PD_M2_RET_c) |-> DUT.PMU.PD_M2_ISO_c
   throughout ##[0:$] $fell(DUT.PMU.PD_M2_RET_c);
endproperty
a_PD_M2_iso_while_ret:assert property (PD_M2_iso_while_ret)
   else $error("Isolation disabled while retained!");
```

Although we have used these assertions only during simulation, based on the work of Ref. 17, such assertions can also be used in formal property verification of the PMU.

5. Experimental Evaluation

In order to validate the proposed verification approach, several experiments had to be carried out. For this purpose, a generator was developed to generate samples of ESL power-management specification. It generated samples of various complexities — the numbers of power modes, power domains, as well as power states in the power domains were scaled (from two to ten). The samples were either randomly generated, i.e. including also erroneous samples, or rules-controlled when only correct samples were generated. The experiments were divided into several stages. First, the application testing was selected to validate the verification capabilities of the implemented methods. The following artificial scenarios were adopted:

- various parts of the abstract specification were incorrectly specified (syntax errors injected) to validate the syntax checking and run-time checking detection capabilities,
- some parts of the abstract power-management specification were intentionally omitted to validate completeness-detection capabilities of the proposed static analysis method,
- inconsistent ESL power management was intentionally specified to validate the consistency-detection capabilities of the static analysis.

The application testing was carried out mostly manually covering all the possible types of faults that the designer can make in an ESL power-management specification. Then, the verification methods were applied to the random samples. In case an error was detected, the result was inspected manually to check its correctness.

In the next stage, the synthesized RTL power management was tested for correctness. These tests were focused on checking whether the proposed verification approach detected all the errors which are detectable at the RTL using the conventional methods and tools. In the evaluation, the power-aware static analysis offered by Modelsim SE 10.2c was utilized. This professional EDA (Electronic Design Automation) tool is able to detect errors in UPF specification, such as incorrectly connected power switch, missing isolation or level shifters. For this purpose, the rules-controlled generator was used to generate 15 samples of the abstract power-management specification in SystemC. The generated samples went through the respective verification steps, they were then synthesized into the UPF format using the proposed high-level synthesis process, and finally, the correctness of the RTL power-management specification was verified in Modelsim. All the synthesized samples were successfully validated in Modelsim. As a result, we can deduce that the proposed verification steps, especially the developed power-management static analysis, indeed drive a designer to the correct ESL power-management specification.

In this experiment, the developed power-management equivalence checking was also tested. First, the equivalence checking was used to compare the synthesized UPF specifications to the original ESL power management. For all the samples, the

corresponding specifications were equivalent. Then, one of the two specifications was intentionally modified (i.e. either some part of the specification was omitted or changed) and the equivalence was checked again. In this case, the equivalence checking process reported errors in all the test cases. Consequently, the results were inspected manually and they were correct in all the test cases.

In the last stage the synthesized PMUs were verified using functional verification. For this purpose, the power-aware simulation capabilities offered by Modelsim were used. The simulation was driven by the automatically generated test-benches, in which the power supplies specified in UPF were activated, the switching between various power modes was defined using a pseudorandom approach, and the synthesized assertions were used to verify the control-signals sequences generated by the PMU and to measure the functional-coverage. In this way, the communication between the PMU and the power-management elements in UPF was verified. The experiment has also proved that the synthesized assertions are helpful in RTL power management verification.

It is expected that one of the key benefits that the proposed low-power design flow will bring is the substantial reduction in specification as well as verification effort. To demonstrate the merits, the specification complexity of the generated samples (used in the previous experiment) was measured in the number of characters, as well as the number of commands. Although the second parameter is more commonly used to measure complexity of a source code, UPF is based on the TCL (Tool Command Language), which uses long inline commands with many command options. Therefore, comparison of the numbers of commands in the specifications only would not be quite adequate. However, the UPF is also significantly influenced by the synthesized identifiers impacting the number of characters; therefore, both complexity parameters were used. The experimental results are summarized in Tab. 4.

Table 4. Power-management verification evaluation samples.

#	*ESL*	*nESL*	*UPF*	*nUPF*	*PMU*	*nPM*	*Assert*	*nProp*	*nDirect*	*Coverage*
1	313	11	1680	26	3300	4	3863	9	10	100 %
2	500	16	2706	42	4452	5	4749	5	17	100 %
3	642	20	2850	47	4619	5	3194	2	15	100 %
4	643	23	6035	74	35205	46	25912	18	103	98 %
5	751	23	4658	65	8658	12	9488	14	29	100 %
6	760	24	4854	65	7017	8	10340	11	34	100 %
7	813	18	5678	74	10094	14	13433	14	41	97.5 %
8	862	24	5557	72	63304	49	17779	11	81	100 %
9	953	38	8778	99	142992	114	52330	25	175	81.1 %
10	1051	40	9399	105	131478	101	45150	28	156	96.1 %
11	1090	27	11605	124	586303	346	100721	37	421	85.5 %
12	1275	24	7443	92	199866	106	48111	21	146	89 %
13	1324	30	9039	102	107068	117	43833	25	172	91.2 %
14	1402	50	13397	147	67691	65	50911	36	126	99.2 %
15	1939	27	12232	140	214122	132	91620	25	188	82.4 %
Case Study	515	18	5612	70	35216	44	20940	23	79	100 %

The first column in the table represents the sample reference number (marked #). The *ESL* and *nESL* columns represent the number of characters and the number of commands respectively, required for the abstract ESL power-management specifications. Similarly, the *UPF* and *nUPF* columns represent the number of characters and the number of commands of the synthesized UPF specifications, respectively. The *PMU* column refers to the number of characters generated automatically to synthesize the power-management unit functional description. The difference between values in *UPF* and *ESL* columns, together with the *PMU* value, can be interpreted as the specification effort reduction. The *Assert* column denotes the number of characters required to describe the assertion statements, which can be perceived as part of the verification effort reduction. The column *nPM,* representing the number of power modes (including intermediate power modes) in the synthesized PMU, provides a perception of the respective sample complexity. The *nProp* and *nDirect* columns represent the number of properties and the number of explicit coverage directives in the generated assertions, respectively. Finally, the directive coverage, reported by Modelsim after the simulation, is provided in the *Coverage* column.

In all the cases, the coverage report is above 80 %. Although this might seem like not a very good result, the goal of this experiment was not to fully verify the PMUs, but to show that the generated assertions can be directly used for PMU verification and for coverage measurement. This has been proven. As it was mentioned earlier, the simulation was driven by the automatically generated test-benches, using a pseudorandom stimuli-generation approach, and the simulation time was rather short – 100 ms. Therefore, higher coverage can be easily reached using a different stimuli-generation approach, for example directed testing. Since the number of assertion statements can be really high (up to 37 properties and 421 coverage directives for the generated samples), their automated synthesis spares a great amount of time required otherwise for the manual verification preparation (i.e. manual creation of the assertions as well as possible debugging effort due to human errors). The developed verification processes – the power-management static analysis and the equivalence checking have taken only few seconds (up to 1.4 s for the considered samples), hence the experiments have also shown that the proposed methods are scalable. When compared to the manual effort regarding the assertion creation and possible debugging, the proposed automated method can save days, or even weeks, of development time.

To be able to compare the complexity of the tested artificial samples to our case-study system in Secs. 3 and 4, the data for the case study is provided in the last row of the table. Based on these data, one can deduce a need to also measure the specification complexity in the number of characters, because while the number of commands in UPF is only 3.89 times higher, the number of characters is 10.9 times higher. A manual description of the case-study PMU, which has taken 35216 characters, would take time and human errors could be introduced to the description during the manual process, which would prolong verification time. Therefore, the automated synthesis contributes by itself to the

verification speed-up. The assertions, used for verification of the PMU function, have taken 20940 characters, representing more space for possible human-error introduction if described manually.

6. Conclusions

This paper is dedicated to the verification approach that forms an integral part of the low-power systems design methodology published in our previous work[14–16]. The proposed design methodology is based on the abstract power-management specification and high-level synthesis methods that by themselves bring certain verification benefits. The abstraction from error-prone low-level details (e.g. power switches, isolation or retention) results in a very concise and intuitive specification, avoiding many common power-management errors. The integrated automated verification means, discussed in this paper, serve as a guide for a designer to build a correct and complete abstract specification. The verification means take several forms, namely the syntactical checks during the compilation, the run-time checks during the system-level simulation, and the proposed static analysis during the specification development and power-management synthesis. Furthermore, a novel equivalence checking approach has been proposed to verify the equivalence between the generated UPF specification and the original abstract power-management specification after the synthesis process took place. In addition to these verification steps, the controlling and coverage assertions are automatically generated during the power-management synthesis process, enabling to verify the correct operation of the power-management unit as well as to track the verification progress of power modes.

Compared to the available related work, analyzed in Sec. 2, the proposed power-management verification approach is the most robust. The means for early verification drive the designer to develop a complete and consistent system-level power-management specification and the continuous verification steps during the design flow ensure the power intent preservation throughout the whole design process. Most of the verification steps are automated; therefore, the preparation and debugging verification processes are significantly shortened.

A prototype tool has already been developed, implementing the proposed verification methods. The tool has been used in the experiments and thus tested the corresponding algorithms. These algorithms can be now used in development of standalone verification tools usable by designers. The proposed syntax checking and run-time checking is supported by any C++ compiler — i.e. they do not require any additional verification tool. When the other methods will be implemented as command line tools, these methods can be used from any commonly used development environment supporting ESL modelling and high-level synthesis, such as Stratus, Catapult, or Synphony C compiler. The proposed power-management static analysis as well as the equivalence checking could be thus used directly. Although the power-management assertions can be also generated by a standalone tool, they would require a designer to insert them into a test-

bench manually. Therefore, it would be better to integrate the power-management high-level synthesis and synthesis of the assertions into the existing high-level synthesis tools.

Further work in this research area can be oriented towards automated creation of directed power-mode switching in the synthesized test-benches. This could help to increase the coverage during the functional verification of the synthesized power management at the RTL. A possible next challenge is to deal with asynchronous systems. The power management already uses some asynchronous behavior (e.g. acknowledgment of the power domain powering up). The method of Ref. 26 could help the power management to be more adaptive to the environment, specifically to determine when the power mode can be switched. The problem is that the synthesized assertions are strongly clock-based; therefore, the functional verification of such a power management would be difficult.

Acknowledgments

This paper was created with the support of 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. This work was also partially supported by the Slovak Scientific Grant Agency (VEGA 1/0616/14 and VEGA 1/0836/16) and the Slovak Research and Development Agency (APVV-15-0789).

References

1. IEEE Standard for Design and Verification of Low Power Integrated Circuits (2013), (IEEE Std. 1801-2013).
2. O. Mbarek, A. Pegatoquet and M. Auguin, Using unified power format standard concepts for power-aware design and verification of systems-on-chip at transaction level, *IET Circ. Device. Syst.*, **6** (2012) 287–296.
3. H. Affes, M. Auguin, F. Verdier and A. Pegatoquet, A methodology for inserting clock-management strategies in transaction-level models of system-on-chips, *2015 Forum on Specification and Design Languages (FDL)* (2015), pp. 1–7.
4. J. Karmann and W. Ecker, The semantic of the power intent format UPF: Consistent power modeling from system level to implementation, *2013 23rd Int. Work. on Power and Timing Modeling, Optimization and Simulation (PATMOS 2013)* (2013), pp. 45–50.
5. F. Mischkalla and W. Mueller, Advanced SoC virtual prototyping for system-level power planning and validation, *2014 24th Int. Work. on Power and Timing Modeling, Optimization and Simulation (PATMOS 2014)* (2014), pp. 112–119.
6. Y. Xu, R. Rosales, B. Wang, M. Streubühr, R. Hasholzner, C. Haubelt and J. Teich, A very fast and quasi-accurate power-state-based system-level power modeling methodology, *ARCS'12 Proc. of the 25th Int. Conf. on Architecture of Computing Systems* (2012), pp. 37–49.
7. H. Lebreton and P. Vivet, Power modeling in SystemC at transaction level, Application to a DVFS architecture, *IEEE Computer Society Annual Symp. on VLSI* (2008), pp. 463–466.
8. T. Bouhadiba, M. Moy and F. Maraninchi, System-level modeling of energy in TLM for early validation of power and thermal management, *DATE '13 Proc. of the Conf. on Design, Automation and Test in Europe* (2013), pp. 1609–1614.

9. S. Kaiser, I. Materic and R. Saade , ESL solutions for low power design, *2010 IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)* (2010), pp. 340–343.
10. M. Streubühr, R. Rosales, R. Hasholzner, C. Haubelt and J. Teich, ESL power and performance estimation for heterogeneous MPSOCS using SystemC, *2011 Forum on Specification and Design Languages (FDL)* (2011), pp. 1–8.
11. W. T. Hsieh, J. C. Yeh, S. C. Lin, H. C. Liu and Y. S. Chen, System power analysis with DVFS on ESL virtual platform, *2011 IEEE Int. SOC Conf. (SOCC)* (2011), pp. 93–98.
12. Ch. Trummer, Ch. M. Kirchsteiger, Ch. Steger, R. Weiß, M. Pistauer and D. Dalton, Automated simulation-based verification of power requirements for Systems-on-Chips, *2010 IEEE 13th Int. Symp. on Design and Diagnostics of Electronic Circuits and Systems (DDECS)* (2010), pp. 8–11.
13. S. Ahuja, *High Level Power Estimation and Reduction Techniques for Power Aware Hardware Design*, (Faculty of the Virginia Polytechnic Institute and State University, 2010) (dissertation thesis).
14. D. Macko and K. Jelemenská, Managing digital-system power at the system level, *IEEE Africon 2013 Sustainable Engineering for a Better Future* (2013), pp. 179–183.
15. D. Macko, K. Jelemenská and P. Čičák, Power-management specification in SystemC, *2015 IEEE 18th Int. Symp. on Design and Diagnostics of Electronic Circuits and Systems* (2015), pp. 259–262.
16. D. Macko, K. Jelemenská and P. Čičák, Power-management high-level synthesis, *2015 IFIP/IEEE Int. Conf. on Very Large Scale Integration (VLSI-SoC)* (2015), pp. 63–68.
17. A. Hazra, S. Goyal, P. Dasgupta and A. Pal, Formal verification of architectural power intent, *IEEE T. VLSI Syst.*, **21** (2013) 78–91.
18. R. Mukherjee, P. Dasgupta, A. Pal and S. Mukherjee, Formal verification of hardware / software power management strategies, *2013 26th Int. Conf. on VLSI Design and 2013 12th Int. Conf. on Embedded Systems* (2013), pp. 326–331.
19. D. Macko, K. Jelemenská and P. Čičák, Early-stage verification of power-management specification in low-power systems design, *2016 IEEE 19th Int. Symp. on Design and Diagnostics of Electronic Circuits & Systems (DDECS)* (2016), pp. 259–262.
20. A. Rogers, Designing a simple system-on-a-chip in under 60 minutes with the mu0 microprocessor and Xilinx tools (2003) (tutorial), http://www.ece.uah.edu/~lacasa/tutorials/mu0/mu0tutorial.html
21. Cadence Design Systems, Stratus high-level synthesis: Industry's first high-level synthesis platform for use across your entire SoC design (2015), https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/system-design-verification/stratus-ds.pdf
22. Mentor Graphics, Catapult high-level synthesis, https://www.mentor.com/ hls-lp/catapult-high-level-synthesis/
23. Synopsys, Synphony C Compiler: High-level synthesis from C/C++ to RTL (2016), http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/SynphonyC-Compiler.aspx
24. Cadence Design Systems, A practical guide to low power design: User experience with CPF (2012).
25. N. Khan, Closed-loop verification methodology for low-power SoC design, *Special Technology Report – Low Power Design* (2008), pp. 7–8.
26. L. Nagy and V. Stopjaková, Current sensing completion detection in dual-rail asynchronous systems, *2012 IEEE 15th Int. Symp. on Design and Diagnostics of Electronic Circuits and Systems* (2012), pp. 28–41.