Published in IET Circuits, Devices & Systems
Received on 30th November 2011
Revised on 7th June 2012
doi: 10.1049/iet-cds.2011.0352

Special Issue on Thermal, Power and Timing Modeling, Design and Simulation

ISSN 1751-858X

Using unified power format standard concepts for power-aware design and verification of systems-on-chip at transaction level

O. Mbarek   A. Pegatoquet   M. Auguin
LEAT, University of Nice-Sophia Antipolis-CNRS, 250-Rue Albert Einstein, Bâtiment-4, Valbonne 06560, France
E-mail: mbarek@unice.fr

Abstract: Building efficient and correct system power-management strategies relies on efficient power architecture decision making as well as respecting structural dependencies induced by such architecture. Transaction level modelling allows a rapid exploration, verification and evaluation of alternative power-management architectures and strategies. This study introduces an efficient methodology for making system power decisions at transaction level (TL) by adding and verifying power intent and management capabilities into TL models. A generic framework that abstracts relevant concepts of the IEEE 1801 unified power format standard and implements assertion-based contracts is used throughout the methodology. A TL-model example is considered to validate the methodology.

1 Introduction

As systems-on-chip (SoCs) grow in embedded functionalities and complexity, the importance of power management increases as well. Approaches such as power gating and adaptive voltage scaling are widely used to reduce power in SoCs. The most basic form of these approaches is to partition the chip logic into multiple voltage regions or power domains, each with its own power supply and power control unit. This leads to introduction of additional power elements mediating interfaces between power domains [1]. These elements contribute in defining the per-domain power states. So, the different operating power modes of a SoC can be seen as the different combinations of these power domains' states. They are controlled by a power-management block to implement an appropriate power-management strategy. A good strategy would be to efficiently enable functional resources based on the required application load (e.g. reading an email on a smartphone or capturing pictures). In such a case, each operating power mode corresponds to a different system use case. This requires a good understanding of both hardware and software parts of the final system, as well as their interaction with power management. Indeed, placement and behaviour of each element in power architecture can create dependencies between power domains which constraints the legal operating power modes. For example, a wrong placement of power elements or selection of irrelevant behaviour for them can alter the intended system functionality. Therefore verification becomes compulsory and constitutes a complex task. The recent unified power format (UPF) standardisation [2] enables defining and verifying the power intent that represents the power-management architecture (set of power domains, power switches supply networks etc.) and strategy (legal system power modes and power transitions) specifications, throughout a register-transfer level (RTL) to graphic data system II (GDSII) design flow [3]. However, earlier power intent is added to the system, higher the power reduction and easier the verification would be. Transaction-level (TL) models [4] provide faster simulation times than RTL ones and are mainly dedicated to verify the whole system including the embedded software. Hence, specifying power intent and the corresponding power-management block with the above-mentioned capabilities at TL is a promising solution to validate a power-management structure early and to handle complex verification issues. Besides, this allows a rapid exploration of different power design alternatives and early decision making of the most energy-efficient one before starting design at RTL. Unfortunately, taking advantage of UPF standard capabilities at TL is not possible since there is still no power-aware TL-simulator understanding power intent and constraints as captured by UPF.

In this paper, we propose a 'PwARCH' generic framework that abstracts relevant power concepts specified by the IEEE 1801 (UPF) standard. By following a four-stage methodology, this framework allows adding high-level power architecture to a TL-model and building a power-management strategy upon it. A verification process has been also built in order to check different classes of contracts that express properties between power and functional architecture and are implemented using assume and guarantee assertion types. Moreover, our methodology enables exploring different power design alternatives and choosing the most energy-efficient one.

The paper is organised as follows. Section 2 gives an overview of related works and highlights our contributions. Section 3 describes the 'PwARCH' framework. In Section 4, our methodology flow is explained. The case study in Section 5 applies our proposal to a TL-model example. Finally, Section 6 summarises our paper.

2 Related work

Many *ad hoc* approaches have addressed power modelling and estimation at different transaction levels. Since there is no standard way to create TL-power models for a system or intellectual property (IP) cores, each of these approaches support different criteria to explore power profiles in a transaction-level modelling (TLM) context. Most of the proposed approaches focus on the instrumentation of existing TL simulation platforms to apply such profiles. Usually, power profiles are fed with power consumption metrics coming typically from IP datasheets or low-level simulations. Instrumentation-based solutions mainly range from transaction-based power modelling solutions to component-centric ones. The authors in [5] propose a method to build transaction-based models that resume all types and granularities of transfers between the different blocks of an existing TL-platform as well as the different relationships among them. However, Lee *et al.* [6] and Ben *et al.* [7] focus on power modelling of each hardware component in a TL-platform. These two solutions are not generic enough since they suppose that a cycle-accurate simulation platform already exists, which is not always the case. Similarly, the authors in [8] adopt a state-based power profiling technique applied to each component in a TL-platform. Nevertheless, this method assumes that dynamic power management) and dynamic voltage and frequency scaling power architectures of each considered IP core are available. Contrary to this work, we consider in this paper a strong relationship between specific low-power architecture, a system power-management strategy controlling it and the resulting power savings. Our approach proposes a way to investigate this relationship starting from TL-models. Compared with the previously mentioned works, it is also an instrumentation-based approach. In particular, it is generic enough to be applied to any TL-model. Furthermore, our power models are neither component-based ones, nor transaction-based. Instead, we use state-based models relying on power-domain reasoning. Such reasoning has been involved throughout the proposed methodology by abstracting relevant UPF concepts to adapt them to a TL modelling usage. Some works [9, 10] have considered IEEE 1801 (UPF) specifications and simulation semantics to achieve simulation-based or formal power-aware verification. However, as far as we know, none of the state-of-the art works have used UPF semantics at TL. The key difference with the above-mentioned related works is that we not only perform early TL power estimation, but also deal with power-aware design, management and verification issues at this modelling level.

3 Overview of the PwARCH framework

Our main objective is to augment an existing SystemC/TLM model with power intent and control capabilities including power-aware verification. The 'PwARCH' framework has been developed as a static software library to help achieve that goal in an instrumentation-based manner. Fig. 1 depicts its generic set of C++ classes where each group serves a specific purpose. The main features of the framework are explained in the following sections.

3.1 Abstracting UPF concepts

3.1.1 UPF standard concepts: The UPF standard [2] offers semantics for implementation and verification of low-power design intent. It describes a hardware description language (HDL) functionality subset of a power distribution and the behaviour of its power elements in a side file. This file serves the entire RTL to GDSII design flow and can be modified throughout this flow in an incremental process

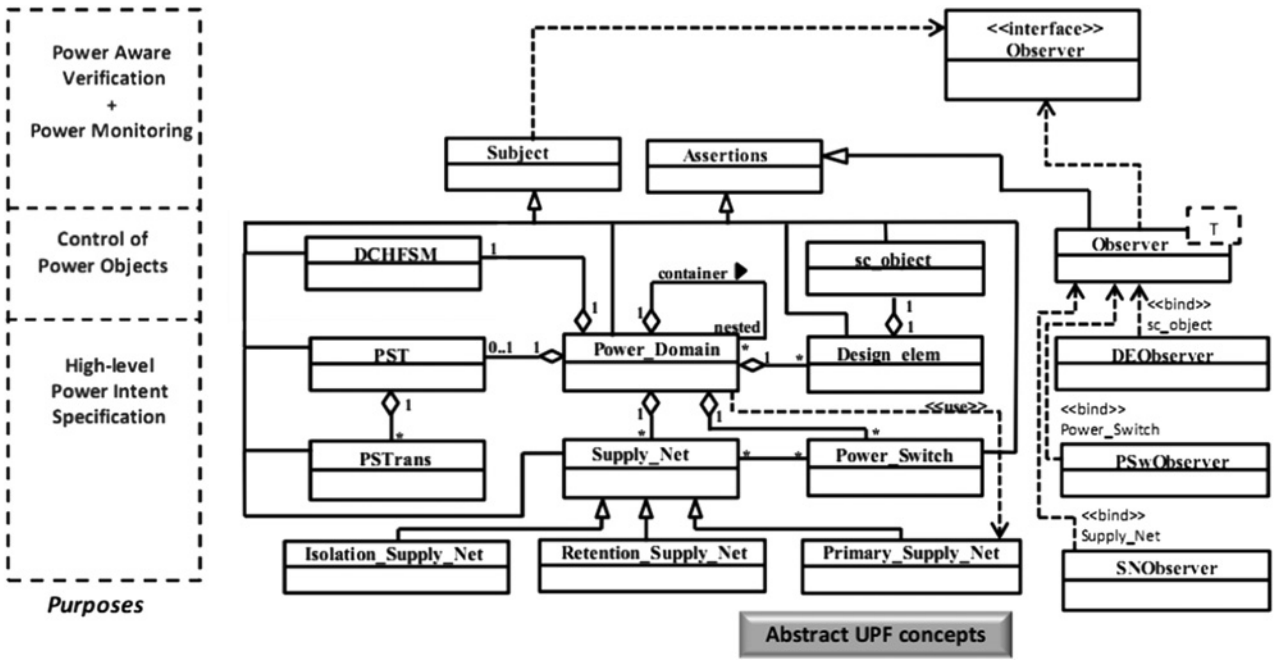


Fig. 1 *PwARCH framework structure: UML class diagram*

(Fig. 2). Fig. 3 gives an example [11] of the main power elements used by UPF semantics to specify the power design intent explained as follows: the power-domain concept in the UPF standard is a centric concept and is defined as the group of elements from the logic hierarchy that share the same primary supply nets (SNs) [2]. In other words, each power domain (e.g. CRC_GEN power domain in Fig. 3*a*) is supplied by a power net (e.g. VDD_HIGH in Fig. 3*a*) and overlays a functional block (e.g. the Checker block in Fig. 3*a*). As a consequence, each power domain can be controlled individually. Power switches are in charge of shutting down or powering up the power domains depending on the value changes of their control signals (e.g. crc_sd signal). Retention registers (RRs) are used to save the internal state of crucial modules when they are switched off. Level shifters are used for communication between domains with different supply voltages. Isolation elements are used to avoid undefined signal values at the output of a power-gated domain. Among the main concepts of UPF, we find the power state table (PST) which defines the system power-management strategy. Fig. 3*b* depicts an example of a PST for the power distribution architecture of Fig. 3*a* as it can be specified using the UPF standard. Columns of a PST represent local states of power domains in terms of their power SN states. However, lines of a PST represent the different system power modes. Each line corresponds to one legal combination of specific power-domain states. In general, a system power mode (line of a PST) refers to a set of functional tasks matching a specific system use scenario. For instance, the RX_ON power mode in Fig. 3*b* corresponds to the receiving with disabled cyclic redundancy check (CRC) checking scenario.

As can be seen in Fig. 3*a*, a power controller must be defined as an HDL functional block that uses the PST in order to control states of all these power elements through specific control signals (e.g. crc_sd control signal in Fig. 3*a* is used to control the state of the CRC_GEN domain's power switch).

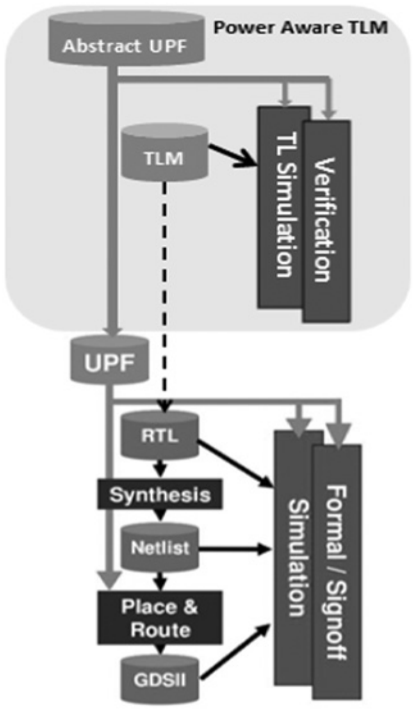
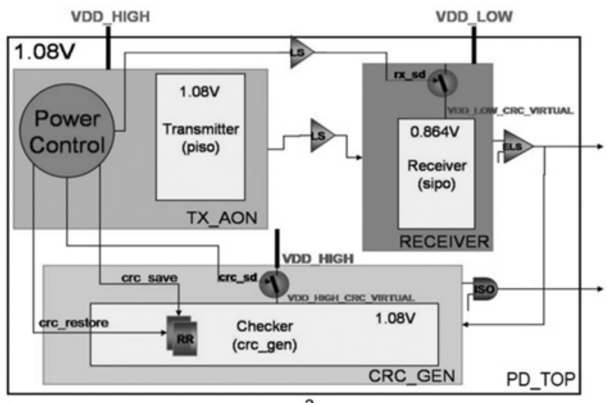


Fig. 2 *Starting UPF flow from TL*



a

	VDD_HIGH	VDD_LOW	VDD_HIGH_ CRC_VIRTUAL	VDD_LOW_ CRC_VIRTUAL
PRE_BOOT	High_Voltage	Low_Voltage	CRC_OFF	RX_OFF
CRC_ON	High_Voltage	Low_Voltage	High_Voltage	RX_OFF
RX_ON	High_Voltage	Low_Voltage	CRC_OFF	Low_Voltage
ALL_ON	High_Voltage	Low_Voltage	High_Voltage	Low_Voltage

b

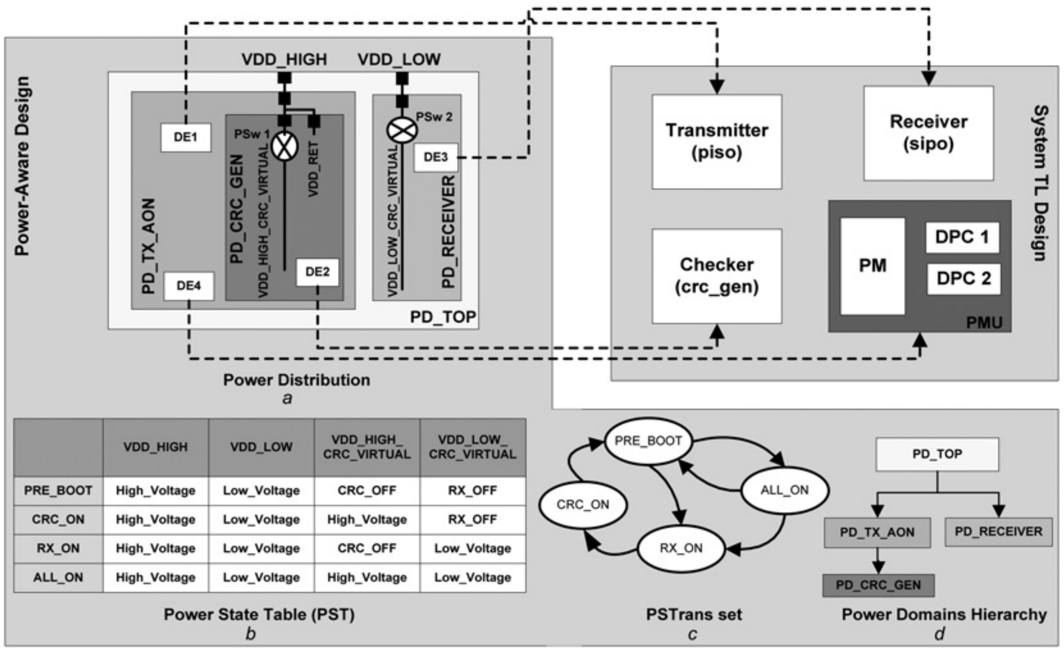
Fig. 3 *Example of UPF concepts*

a UPF-based power distribution elements
b PST for power control

3.1.2 Using UPF concepts at TL of modelling: As far as we know, TL-simulators that capture UPF semantics are not available yet. Moreover, we can observe that only a subset of power components from UPF semantics can be used to express TL power intent, as their behaviour is compatible with the TLM system view. Despite this compatibility, some attributes and simulation behaviour of UPF components considered at TL must even be abstracted. For instance, power switch controls have to be included transparently in a TLM simulation. The 'PwARCH' framework helps to build abstracted UPF specifications according to power gating and multi-voltage power requirements. By starting power design intent at TL, we aim at generating a UPF constraints file describing the most energy-efficient system power architecture, and being the golden low-power reference to the RTL design team (Fig. 2). As shown in Fig. 1 (abstract UPF concepts part), composition hierarchy between abstract UPF components in 'PwARCH' is the same as in UPF-defined semantics. For instance, SNs are instantiated in the context of an existing power domain. Unlike UPF, we added other power components [such as design element (DE) objects] and composition constraints (e.g. PST objects are only instantiated in the context of a composite power domain). This facilitates managing hierarchical structure and control and mapping the functional design to the power-aware one.

Fig. 4 represents the power-aware design corresponding to the UPF-based one in Fig. 3 which is built using the abstract UPF concepts in a PwARCH library. Fig. 4 also shows how such a constructed power-aware design is mapped to the functional TL design. In the following, basic features of each power component that are captured in our framework are highlighted and exemplified using Fig. 4.

Each functional SystemC module is attached to a DE object. For instance, the design element DE3 in the power-aware design part in Fig. 4 points to the 'Receiver' functional block in the system TL design part (dashed lines in Fig. 4 represent C + + pointers to objects). As can be



	VDD_HIGH	VDD_LOW	VDD_HIGH_CRC_VIRTUAL	VDD_LOW_CRC_VIRTUAL
PRE_BOOT	High_Voltage	Low_Voltage	CRC_OFF	RX_OFF
CRC_ON	High_Voltage	Low_Voltage	High_Voltage	RX_OFF
RX_ON	High_Voltage	Low_Voltage	CRC_OFF	Low_Voltage
ALL_ON	High_Voltage	Low_Voltage	High_Voltage	Low_Voltage

Fig. 4 *Relationship between a power-aware design built using PwARCH and the system TL design*

a Power distribution
b PST
c PSTrans set
d Power domains hierarchy

seen in Fig. 4, the DE concept of PwARCH is used to bridge a power-aware design to a target system TL design.

The key element in our approach is a power-domain (PD) object that consists in a collection of DE objects that share the same primary SNs, hence can be controlled individually. Indeed, we preserved the same definition of a PD as in the UPF standard specification. A hierarchical instantiation of PD objects is allowed in PwARCH. This facilitates the control of their states and their attached power components. For that, we have defined a PD of type ‘container’ as a power domain that includes at least another PD instantiated in its context that we called a ‘nested’ PD. In particular, a nested PD can also be a ‘container PD’ for other power domain. Fig. 4*d* presents the PDs hierarchy structure of the power distribution in Fig. 4*a*. For instance, the PD_TX_AON power domain is here a ‘container PD’ with regard to the PD_CRC_GEN power domain and ‘a nested PD’ with regard to the PD_TOP power domain. To apply the power gating technique, abstract power switch (PSw) objects must be added at the boundary of the switched PDs. Each one is characterised by at least one input SN and only one output SN. Controlling the state of a PSw is equivalent to changing the state of its output SN Supply net objects can be either of type ‘primary’ (hence of type power or ground net), or ‘retention’ or ‘isolation’. In Fig. 4*a*, VDD_HIGH is a ‘primary’ SN for the PD_TX_AON power domain, whereas VDD_RET is a ‘retention’ SN for the PD_CRC_GEN power domain. Since it supplies retention registers of a PD so that their states remain internally saved during a power off period, a retention SN must be an ‘always on’ power supply and must never be switched off. Each SN is characterised by a (state,voltage) pair. In particular, the output SN of a PSw is of type ‘switched’ which is in addition characterised by a (state,input net) pair used to control the states of PSw objects. A ‘Switched’ SN must be specified as the ‘primary’ power net of a power-gated PD. For instance, the output SN of the power switch PSw1 in Fig. 4*a*, named VDD_HIGH_CRC_VIRTUAL, is the primary SN of the PD_CRC_GEN power domain. Given the definition of a PD adopted in PWARCH and mentioned earlier, putting the ‘Transmitter’ block (DE1) and the ‘Checker’ block (DE2) in two separate PDs (respectively in PD_TX_AON and PD_CRC_GEN) is explained by the fact that these blocks are supplied by distinct primary power nets (respectively) VDD_HIGH and VDD_HIGH_CRC_VIRTUAL primary SNs). It is also worth mentioning that depending only on its attached primary power net, a PD can be ‘power-gated’ if its power net is of type switched (e.g. PD_CRC_GEN and PD_RECEIVER in Fig. 4*a* are power-gated domains). In this case, it can be entirely powered-down. Moreover, a PD can be of type ‘voltage-scaled’ if its attached primary power net has more than one state. Otherwise, a PD can be ‘non-scaled’ if it has a unique primary power net with a single state and not of a switched type (e.g. PD_TX_AON power domain in Fig. 4*a* is a ‘non-scaled PD’ having VDD_HIGH as a primary power net).

Concerning the PST concept, one of the fundamental concepts of UPF, we have preserved in PwARCH its semantics given by the UPF specification. In particular, a PST object instantiated from PwARCH represents a two-dimensional static table that captures the global power states of a PD of type ‘container’. For example, although the PST in Fig. 4*b* has been kept the same as that of Fig. 3*b*, it has been attached to the PD_TOP power domain when specified using PwARCH since it resumes the power-management strategy of the whole system. In the rest of this paper, we denote each column of the PST by local power state (LPS) and each line by global power state (GPS) .

According to a specified PST, legal transitions between its GPSs must be specified using PSTrans objects. Fig. 4*c* illustrates an example of legal transitions specified according to the PST of Fig. 4*b*. For instance, note that transitioning from the global power state CRC_ON to

RX_ON is specified as unauthorised. A PST object and its related PSTrans set are required by a power-management unit (PMU) functional block in order to dynamically put the system in the appropriate system power mode, hence accordingly adjust the power distribution state. The features and structure of this functional block added to the system TL design (Fig. 4) are explained later.

3.2 Building power-aware verification

Our verification solution is not only similar to UPF that mainly focuses on verifying functional failure and wrong placement of power components but also emphasises verification of different component interactions resulting from our methodology. Depending on the types of communicating components, possible errors have been classified into well-defined categories of contracts as explained later. According to our approach, two kinds of interactions between two components can be distinguished. A component may simply require using another component to perform a specific functionality. It can also modify some of the other component's characteristics as a part of its functionality. To be carried out in a safe and correct way, these kinds of interactions must be characterised by a set of assume/guarantee properties [12] that form a component contract. Several supports for the use of contracts have been proposed whether as development methodologies or even as programming languages. In many of them [12–14], contracts are part of the program, and the compiler generates defensive code that may raise exceptions at run-time when a contract is violated. As we target a simulation-based verification, contracts in our work consist in executable specifications that are monitored at run-time. In 'PwARCH', assume and guarantee properties are seen as two types of assertions that form a contract and raise an exception or report an error when a property is violated during simulation. A generic class 'assertions' has been defined (Fig. 1) with corresponding assume and guarantee methods. A Satisfy method can be called as well by assume or guarantee methods to verify that a specific condition is satisfied.

To add contracts inside a class, the latter must inherit from the assertions class in the 'PwARCH' library. Note that in Fig. 1, the assertions class is inherited by all classes of the 'PwARCH' library which are involved in the power architecture specification or its control. Furthermore, the assume and guarantee assertions of a class can be switched on or off. Hence, power-aware verification will not be processed unless it has been explicitly enabled by the user before starting a simulation. These added assertions, neither affect the functional behaviour of the system, nor cause side effects on individual objects. Nevertheless, defensive programming implies writing additional code used in particular to verify the arguments of the assume, guarantee and satisfy methods. Such additional codes have been implemented in the 'PwARCH' library and hence, are hidden to the user in order to facilitate and speed up the verification process.

3.3 Power analysis and estimation

A power monitor is attached to each PD object and it automatically updates its power values using its power structure and its LPS. Each power monitor is triggered when its PD receives a power event. When such an event occurs, it provokes a change in at least a non-switched SN state or a PSw state. To capture such events, an observer such as an SNObserver object is attached to each non-switched SN and a PSwObserver object is attached to each PSw (Fig. 1). Owing to the hierarchical PDs construction, a PD state change during the simulation will automatically and recursively update the power values of PDs in higher levels of hierarchy.

4 Power domain-based methodology

We propose a power-domain-based methodology to add power-management capabilities to the different functional parts of a SoC described in SystemC/TLM. Fig. 5 illustrates the overall flow of the proposed methodology. It is mainly composed of three stages processed sequentially as indicated in Fig. 5. A fourth stage is dedicated for verification and occupies an orthogonal position regarding the previous stages. The methodology is iterative allowing the designer to explore different power design alternatives and retain the most energy-efficient one. Note also that the different stages are complementary and dependent on the'PwARCH' library. In the following, the main purposes of each stage are explained.

4.1 Power intent specification stage

At this stage, the designer configures the system power intent by instantiating adequate objects from 'PwARCH'. In the first phase, the transaction traces resulting from a TL functional simulation of the embedded software inform about possible correlations between hardware (HW) blocks in order to identify power reduction opportunities (e.g. gathering for instance the strongly correlated blocks in the same PD). Then, a power design is specified by instantiating from PwARCH the required PDs, supply network distribution and observers. This power design is mapped to the existing HW architecture by attaching DE objects to functional

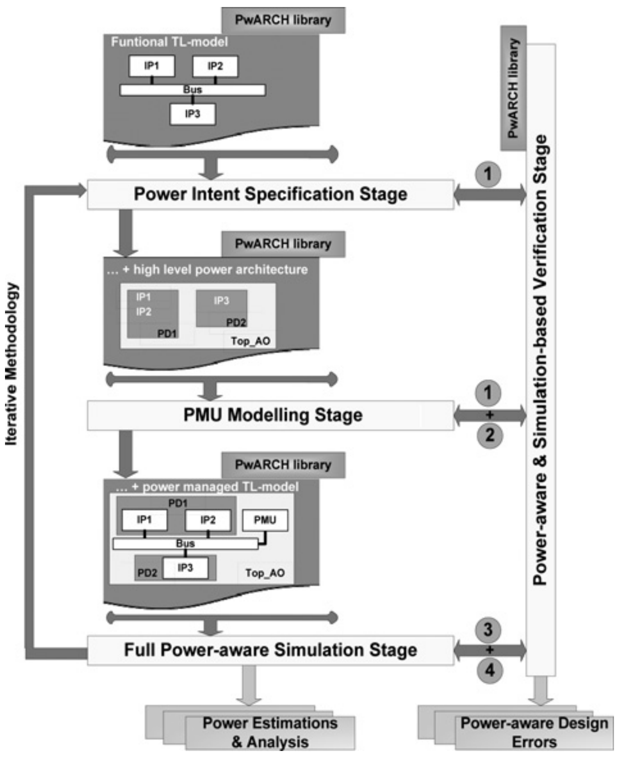

PwARCH library
Funtional TL-model
IP1
IP2
Bus
IP3
Power Intent Specification Stage
PwARCH library
... + high level power architecture
IP1
IP2
PD1
IP3
PD2
Top_AO
PMU Modelling Stage
PwARCH library
... + power managed TL-model
PD1
IP1
IP2
PMU
Bus
PD2
IP3
Top_AO
Full Power-aware Simulation Stage
Power Estimations & Analysis
Power-aware Design Errors
Iterative Methodology
PwARCH library
Power-aware & Simulation-based Verification Stage
1
1 + 2
3 + 4

Fig. 5 *TL power-aware methodology flow*

modules. Technology data and available low-level power values are attributed to DE objects and used to update power consumption values. In the second phase, according to the PDs boundaries and transaction traces, a system power-management strategy is established by specifying a PST and its PSTrans set. Here, each possible combination of PDs states is bound to an appropriate part of the software (SW) flow, with respect to inter-power objects dependencies.

4.2 PMU modelling stage

To control local and global power states, a TL PMU is modelled as an additional SystemC module communicating with other modules of the platform through bus transactions. Since it uses a PST, a PMU activity only makes an impact on the GPS of the container PD of this PST. For each GPS in a PST will correspond to a transaction in the functional SystemC code that must be preceded by a power control transaction (PwCTr) one. A PwCTr writes to a PMU control register requesting this PMU to set the container PD in a specific GPS. The DE issuing such a transaction must be blocked as long as the PMU performs the required transitions. We consider that each DE holds a particular event and remains waiting for it whenever it issues a PwCTr. When the PMU finishes its activity, it notifies this event so that the suspended design element resumes its activity. Our PMU generic model is mainly composed of a power manager (PM) and a set of domain power controllers (DPCs). Each DPC module changes the LPS of a power-gated domain between sleep and wake-up states. The PM module only changes the LPS of the non-switched PDs and requests adequate DPCs to change their domains' states. For instance, Fig. 4 depicts a PMU block composed of a DPC1 to control the state of the PD_CRC_GEN power-gated domain, a DPC2 to control that of the PD_RECEIVER power-gated domain and a PM that coordinates the activity of DPC1 and DPC2 according to the requested power mode.

4.3 Full power-aware simulation stage

At this stage, we simulate the resulting TL power-managed behaviour. It is processed in parallel with the verification one. During simulation, functional coherence between the augmented TL-model and the power design needs to be verified. The system power-aware behaviour is proved coherent if no verification properties are violated during simulation. Furthermore, PSwObservers and SNObservers instantiated at the power intent specification stage will handle the power and energy values update at run-time and generate log files at the end of the simulation. When plotted, these files' values help in analysing and comparing different power-management solutions, as well as selecting the most energy-efficient power design.

4.4 Power-aware verification stage

Power-management features added throughout our methodology flow may generate different types of errors. In order to ensure that each stage has been correctly performed, a contract-based dynamic verification process is added. As shown in Fig. 5, this verification stage is processed orthogonally to the previous ones. The objective is to identify bugs related to added power-management features. The functional behaviour of the TL-platform is supposed to be correct before applying it to our methodology.

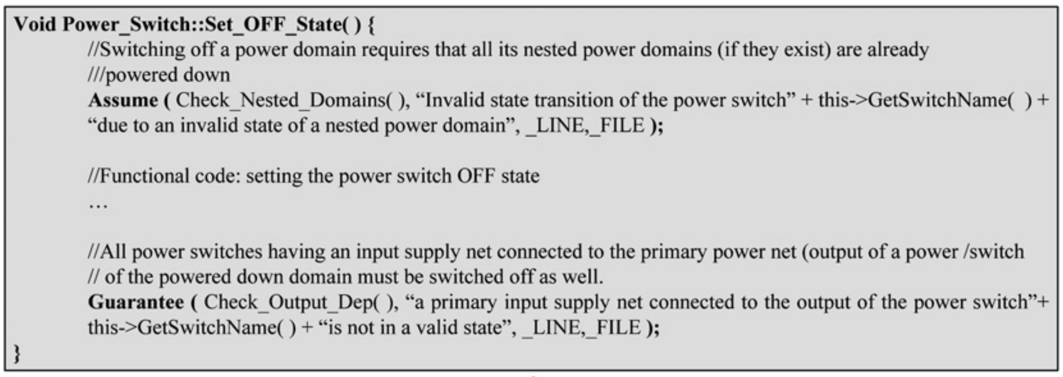
Three types of components are handled throughout the methodology flow:

- *Power components*: represent power objects from the abstract UPF concepts part of 'PwARCH' used to specify the power intent of a TL-design.
- *Functional components*: represent IPs of the considered TL-model.
- *Mixed components*: represent PMU modules and their sub-modules (i.e. DPC and PM modules). They are responsible to set power states of functional components according to a power-management architecture.

Depending on the required interaction of components at each stage of the methodology, contracts have been classified into four different classes. Each class gathers all possible assume (precondition), guarantee (postcondition) and satisfy (invariant) properties between two specific types of components. Properties belonging to each class of contracts ensure that each component uses the other components safely and correctly during simulation.

The different classes of contracts are detailed in the following sections:

4.4.1 Contracts of class 1: This class of contracts specifies the interactions between the power components of a design. All properties related to this class are inserted into the appropriate power components' code. Actually, they are already fully implemented in 'PwARCH'. Their objective is to verify the correctness of a power architecture structure including the hierarchy and composition relationships between its power elements. A typical error is to forget to attach at least one design element to a PD. In that case, the PD is not necessary. Furthermore, each system power mode specification (a line of a PST) must respect structural dependencies between power-domain partitions. This kind of error can be detected when simulating the system after the power intent specification stage. In addition, this class of contracts is used to ensure that the power domain ordering rules are not violated during simulation. These rules define the order that must be respected to turn some PDs on or off. They are imposed by a specific hierarchical composition of PDs and a particular placement of power switches. For instance, let us consider the following example: given a container power domain PD_0 and a PD_{01} power domain that is nested in PD_0. As a consequence, PD_0 and PD_{01} can be individually switched using, respectively, their power switches PSw_0 and PSw_{01}. Obviously, by considering the hierarchical relationship between PD_0 and PD_{01}, the output SN of PSw_0 represents an input SN for PSw_{01}. Therefore PD_{01} must be already switched off before turning off the PD_0 power domain. This property can be considered as an assume part of a contract. It must also be checked that all PDs which are powered by the PSw_0 output SN (whether nested in PD_0 or not) are powered down once PD_0 is switched off. This property represents the guarantee part of the same contract. Note that such a contract specifies the behaviour of the power switches according to the SNs. The properties used to check this kind of specification are added to the power switch class of 'PwARCH' as shown in Fig. 6*a*. The errors related to PD ordering rules are examples of violated contracts of class 1 which can only be checked after the PMU modelling stage. That is, because the power-

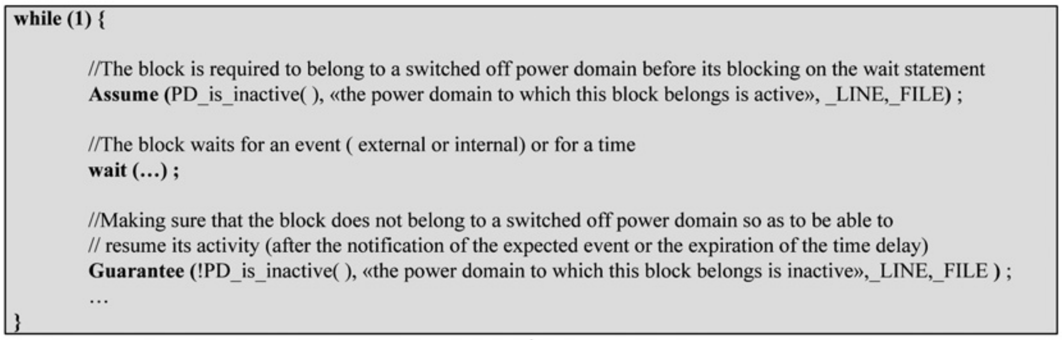


```
Void Power_Switch::Set_OFF_State( ) {
        //Switching off a power domain requires that all its nested power domains (if they exist) are already
        ///powered down
        Assume ( Check_Nested_Domains( ), "Invalid state transition of the power switch" + this->GetSwitchName( ) +
        "due to an invalid state of a nested power domain", _LINE,_FILE );

        //Functional code: setting the power switch OFF state
        …

        //All power switches having an input supply net connected to the primary power net (output of a power /switch
        // of the powered down domain must be switched off as well.
        Guarantee ( Check_Output_Dep( ), "a primary input supply net connected to the output of the power switch"+
        this->GetSwitchName( ) + "is not in a valid state", _LINE,_FILE );
}
```

a



```
while (1) {

        //The block is required to belong to a switched off power domain before its blocking on the wait statement
        Assume (PD_is_inactive( ), «the power domain to which this block belongs is active», _LINE,_FILE) ;

        //The block waits for an event ( external or internal) or for a time
        wait (…) ;

        //Making sure that the block does not belong to a switched off power domain so as to be able to
        // resume its activity (after the notification of the expected event or the expiration of the time delay)
        Guarantee (!PD_is_inactive( ), «the power domain to which this block belongs is inactive»,_LINE,_FILE ) ;
        …
}
```

b

Fig. 6 *Examples of contract-checking instrumentation*

a Example of a class 1 contract inserted in the 'Power_Switch' class of 'PwARCH' library
b Example of a class 3 contract inserted in a hardware block module

management behaviour is only defined at this stage through the addition of power control transactions and the integration of the PMU in the platform.

4.4.2 Contracts of class 2: Class 2 contracts target the specification of interactions between 'mixed components', that is, between the PM and DPC components inside each PMU. Furthermore, it concerns the specification of interactions between 'mixed components' and 'power components'. All contracts of class 2 have to be manually added inside a PMU source code. This is the reason why they can only be verified after the PMU modelling stage. These contracts aim at checking the correct functionality of PMU modules as well as their integration in TL-models. For instance, a power state transition can be required during simulation whereas it is missing in the graph of power transitions (set of PSTrans). Such a miss can be corrected in two different ways: either, a new transition (PSTrans) is specified, or the PMU performs legal intermediate transitions until reaching the required system power mode.

Another example of class 2 contracts consists in checking that each DPC correctly performs the wake-up or sleep transition sequences. During such transitions, it must be verified that the states of specific power objects (power switches and retention SNs) in a switched PD have been changed in a specific order by the corresponding DPC.

The specifications of interactions between a PM and DPCs belong to contracts of class 2 as well. A DPC that has switched off a power domain whereas the PM has requested to power it on represents a serious error. This kind of issue is caused by an erroneous functionality of the PM or the DPC generic module. Furthermore, the PMU functionality must identify and respect specific PDs dependencies. For instance, let us consider again the example of PD_0 and PD_{01} mentioned earlier (in class 1 contracts section). In this case, the PM is not allowed to request a PD_0 switch-off as long as the transition of PD_{01} to OFF is not over. More generally, simultaneous transition requests (to DPCs) to switch-off or on a power domain can be error-prone. These contracts of class 2 are used to specify an order of transitions between specific states of PDs.

4.4.3 Contracts of class 3: Contracts of class 3 specify relations between 'functional' and 'power components'. They are checked at the final stage (i.e. when simulating the power-managed behaviour of a TL-model).

On the one side, a functional hardware block can perform different activities. Each of these activities can be launched further to specific settings of internal registers of this block, the exchanged transactions at its interface, or its internally triggered events. To be performed, an activity can require specific power properties to be satisfied by the block. Among such properties, we can mention a specific state of the block's power domain or a specific value of an internal block's register.

For instance, when a functional block receives or transmits a transaction, its power domain must not be switched off. Otherwise, an error must be detected reporting a wrong PST line specification. The mechanism used to check these properties is based on the observers (DEObserver class from 'PwARCH') attached to each DE object. These observers will indicate an error whenever the power architecture properties of the block do not match with the power requirements of the executed activity.

On the other side, a block becomes most of the time functionally idle when a wait statement in its source code is reached. In this case, the power requirements just before and after wait statements may be different. For instance,

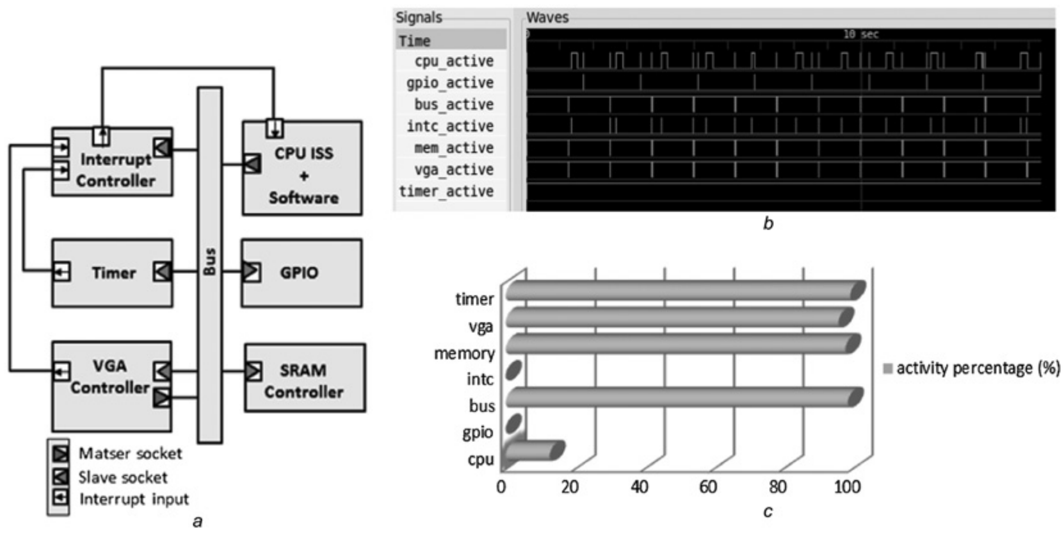


Fig. 7 *Case-study: architecture and transaction flow analysis*

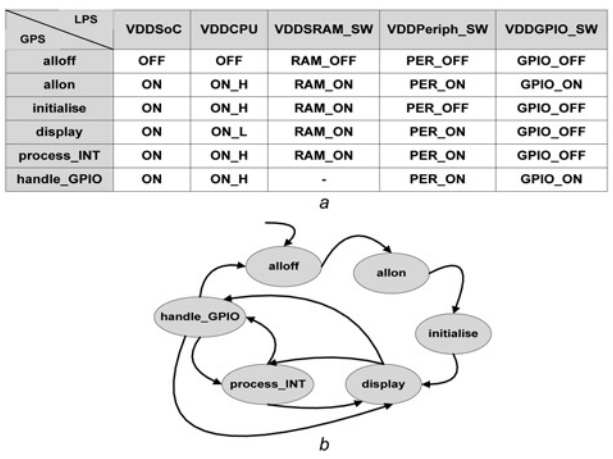
a Case study platform
b Activity waveforms of hardware components
c Activity percentage per component

when a wait on an event statement is reached in a block, the power domain of this block can be downright powered down. However, it must be verified that this power domain has been already woken up just before the expected event is triggered and before the block resumes its normal activity. Fig. 6*b* illustrates how these properties can be specified by instrumenting the blocks' codes with class 3 contracts. We note that the source code of the whole TL-model is supposed to be accessible in this work and can be instrumented. Hence, adding this kind of class 3 contracts is done in general by surrounding the SystemC wait statement with assume and guarantee properties.

4.4.4 Contracts of class 4:

Contracts of class 4 specify relations between 'functional' and 'mixed components' at the final sequential stage (Fig. 5). Verification focuses here on the compatibility between the PMU functionality and the activity of hardware modules extended with power control transactions. Indeed, the PMU activity must not alter the hardware modules activity during simulation. For example, to set up a system power mode, a PMU performs specific power domain state transitions as specified in the corresponding PST line. However, performing a power domain state transition requires that all the hardware modules of that PD are functionally idle (i.e. waiting for an event, time duration or a signal) during this transition. This contract represents an invariant property, that is, checked before and after a power-domain transition is performed by a PMU sub-component. Similarly, when an activity is detected in a hardware block, it must be verified that the power domain of this block is not undergoing a power state transition. A violation of this contract proves a wrong synchronisation between this hardware block and the PMU.

Ideally, contracts of each class should be checked after a specific sequential stage. This facilitates identifying the sources of errors. However, our verification process is flexible since each class of contracts can also be verified after a following stage. For instance, if the contracts of class 2 have not been verified after the PMU modelling stage, they can be checked during the full power-aware simulation stage.



GPS \ LPS	VDDSoC	VDDCPU	VDDSRAM_SW	VDDPeriph_SW	VDDGPIO_SW
alloff	OFF	OFF	RAM_OFF	PER_OFF	GPIO_OFF
allon	ON	ON_H	RAM_ON	PER_ON	GPIO_ON
initialise	ON	ON_H	RAM_ON	PER_OFF	GPIO_OFF
display	ON	ON_L	RAM_ON	PER_ON	GPIO_OFF
process_INT	ON	ON_H	RAM_ON	PER_ON	GPIO_OFF
handle_GPIO	ON	ON_H	-	PER_ON	GPIO_ON

Fig. 8 *Application of the power intent specification stage*

a PST
b Set of PSTrans

5 Application to a case study

To demonstrate our methodology, we consider an existing approximately timed [4] TL-platform (Fig. 7*a*) with no power-management features. The embedded application implements Conway's game of life. First, a software flow analysis is performed in order to determine possible system use cases. This task was automated by attaching observers on input and output ports of each component. By detecting these ports state changes, these observers trace the activity of the corresponding component during simulation. As a result, the waveform shown in Fig. 7*b* was obtained and statistics about the total percentage activity of each component was reported as depicted in Fig. 7*c*. Contrary to the VGA, memory and bus components which were active most of the simulation time, the CPU component was functionally idle for successive time durations.

A viable power architecture solution must hence allow energy savings of the CPU during its periods of idleness. This is achieved by powering the CPU down (by placing a

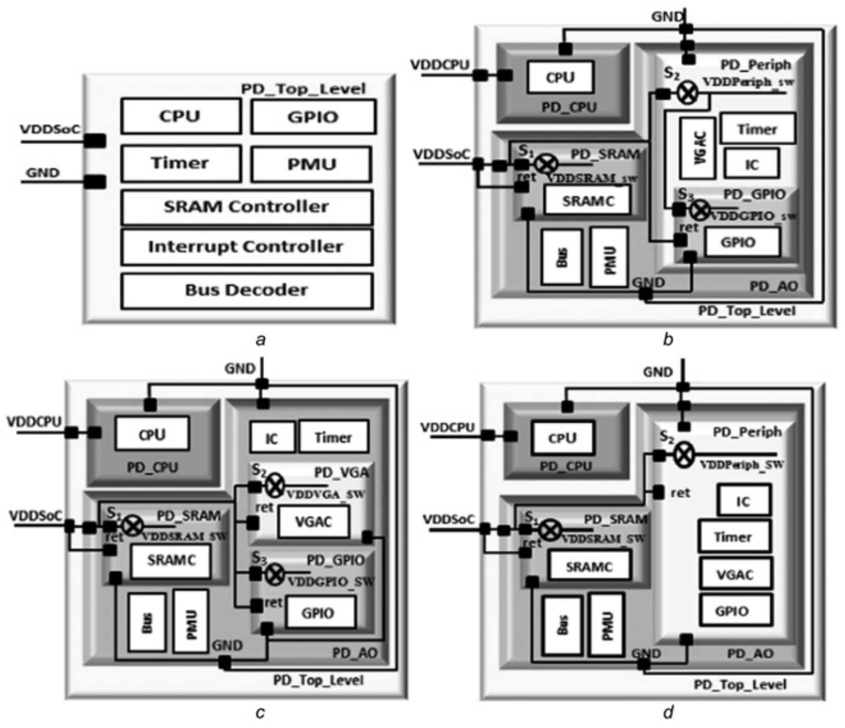
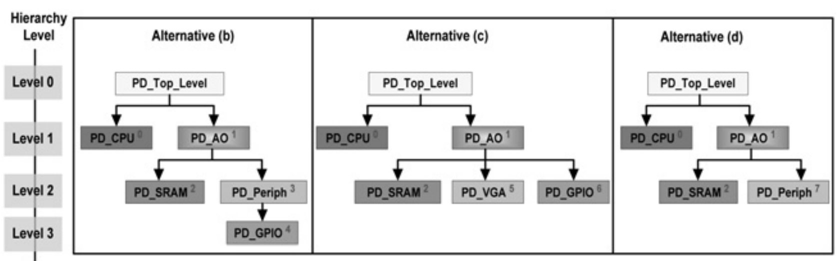


Fig. 9 *Power-aware architecture alternatives*

power switch in its power domain) or by supplying it with a lower primary supply voltage (as considered in our power intent solutions). Furthermore, note that such activity traces facilitate defining a PST according to a power architecture specification. For instance, the VGA and the memory activities are strongly correlated when displaying an image (Fig. 7*b*). Therefore in a 'display' system power mode of a PST, the power domains of these components must be both powered-up. In the following, we give scenarios that match with the GPSs of the PST shown in Fig. 8*a* according to the power design of Fig. 9*b*. The CPU computes a first image by reading and writing from/to the SRAM (initialise scenario). Then, the peripherals are initialised (allon scenario). The VGA controller uses a double buffer to avoid visual glitches when the image changes. First, it reads the image from the memory (first buffer) and displays it (display scenario). Games of life iterations are cadenced by the timer. Hence, an interrupt that is raised by the timer, is driven to the interrupt controller which drives it to the CPU. Then, the CPU handles this interrupt by computing a new image in a second buffer while communicating again with the SRAM (process_INT scenario). Henceforth, the VGA controller is informed by the CPU about the new image address and will display this new image after the display reaches the end of the screen (display scenario again). A button mapped as a general purpose input/output (GPIO) is checked periodically (handle_GPIO scenario). This SW flow is then periodically repeated. As shown in Fig. 9, different power architecture alternatives have been elaborated and evaluated while taking into account this SW flow. Fig. 10 depicts as well the hierarchy and characteristics of the different power domains according to alternatives (b)–(d) of Fig. 9. Note that the power domains partitioning and hierarchy, as well as the membership of hardware blocks (design elements) per power domain are different in each of these alternatives. In particular, alternative (a) corresponds to a unique and always-on (i.e. never switched off) power domain that groups all the platform HW blocks (Fig. 9*a*). Owing to lack of space, only



Exponent	Power Domain Characteristics *
0	voltage-scaled (VDDCPU=1.2 V/0.5 V)
1	non-scaled (always-on) & VDDSoC= 0.8 V & container
2	power-gated with retention
3	power-gated without retention & container

Exponent	Power Domain Characteristics *
4	power-gated with retention & state dependency on the state of the PD with index 2
5	Power-gated with retention
6	Power-gated with retention
7	Power-gated with retention

Except the PD_Top_Level, each power domain is nested in at least one other power domain (see the power domains hierarchy relationship above). In case of a power domain of type container, we particularly mention this in this list.

Fig. 10 *Power domains hierarchy and characteristics in each power domain partitioning alternative*

Table 1 Excerpts of power-aware verification results

Inserted fault	Total errors number	Violated assertions	The component throwing the assertion error	Contract class
the transition from display to handle_GPIO PST power modes is not authorised	155	the transition from display to handle_GPIO PST power modes is not authorised	PMU	2
		An activity is noticed in the GPIO etc block whereas PD_GPIO is inactive	GPIO hardware block	3
in all_on PST power mode, PD_Periph is powered down whereas the PD_GPIO is powered on	6	Invalid state transition of the power switch S_3 owing to an invalid state of its input supply net	power switch S_3	1
		An activity is noticed in design element VGA whereas PD_Periph is inactive	VGA hardware block	3
		An activity is noticed in design element timer whereas PD_Periph is inactive	timer hardware block	3
only the VGA unit activity is blocked just after sending a power control transaction to the PMU in order to set the display power mode	44	The PM cannot perform a power state transition of the PD_CPU because CPU design element is still functionally active	PM	4
		To display an image, the system power mode does not match the display power mode configuration as specified in the PST	PM	3

the PST and legal power state transitions (PSTrans) corresponding to alternative (b) are shown in Fig. 8.

HW components of this TL-model have been implemented on a Virtex-4 FPGA device. The Xilinx Power Estimator tool has been then used to get technology-dependent power characteristics (such as leakage current and load capacitance) which are used to feed power models of each DE. The results show that (b)–(d) alternatives in Fig. 9 provide at least 90% of the energy savings compared with a unique power-domain design [(a) alternative]. The (b) alternative represents the most energy-efficient power domains partitioning since about 58% of energy savings is observed compared with (d) alternative and 7.3% compared with (c). Furthermore, the obtained power-aware simulation speed remains similar compared with the non-instrumented version. For instance, simulation time for alternative (b) is only 0.03% slower than alternative (a).

Table 1 shows a set of violated contracts further to errors made when elaborating the (b) alternative (Fig. 9*b*) using our methodology. Here, simulation is only run after the implementation of all stages. Violated assume and guarantee properties were reported during the simulation period (16 s) in a log file. Note that because of a single inserted fault multiple violated contracts of different classes were detected. This demonstrates the strong complementarity and coherence between all classes of contracts implemented by our methodology.

6 Conclusion and future work

We have presented a novel, efficient and generic methodology to augment a TL-model with power including verification capabilities. This methodology mainly aims at an early decision making of the most energy-efficient and correct power-management design alternative. It also allows mapping TL hardware architecture to a power one using UPF-based concepts in a generic 'PwRACH' framework. Future works will focus on the power intent construction and exploration of power design alternatives as well as on automating the insertion of the remaining types of contracts into the TL simulation model. We are also investigating the formal validation of our methodology to enable a maximum low-power property-coverage testing.

7 Acknowledgment

This work was supported by the French National Research Agency (ANR) project HELP bearing reference ANR-09-SEGI-006.

8 References

1 Keating, M., Flynn, D., Aitken, R., Gibbons, A., Shi, K.: 'Low power methodology manual: for system-on-chip design', in 'integrated circuits and systems' (Springer, 2007)

2 Unified Power Format (UPF 2.0) Standard: 'IEEE standard for design and verification of low power integrated circuits'. IEEE 1801^{TM}, March, 2009

3 Bembaron, F., Kakkar, S., Mukherjee, R., Srivastava, A.: 'Low power verification methodology using UPF'. Proc. Design and Verification Conf. and Exhibition (DVCon), San Jose, CA, 2009, pp. 228–233

4 Open SystemC initiative: 'SystemC transaction level modeling library 2.1.0', 2009, www.systemc.org

5 Dhanwada, N., Lin, I.-C., Narayanan, V.: 'A power estimation methodology for systemC transaction level models'. Proc. Third IEEE/ACM/IFIP Conf. on Hardaware/Software Codesign and System Synthesis (CODES + ISSS), Jersey City, NJ, September 2005, pp. 142–147

6 Lee, I., Kim, H., Yang, P., *et al.*: 'PowerViP: Soc power estimation framework at transaction level'. Proc. 11th Asia and South Pacific Design Automation Conf. (ASP-DAC), Japan, 2006, pp. 551–558

7 Ben Atitallah, R., Niar, S., Dekeyser, J.L.: 'MPSOC power estimation framework at transaction level modeling'. Proc. 19th Int. Conf. on Microelectronics (ICM), Egypt, 2007, pp. 245–248

8 Lebreton, H., Vivet, P.: 'Power modeling in systemC at transaction level, application to a DVFS architecture'. Proc. IEEE Computer Society Annual Symp. on VLSI, France, 2008, pp. 463–466

9 Hazra, A.S., Mitra, A., Dasgupta, P., Pal, A., Bagchi, D., Guha, K.: 'Leveraging UPF-extracted assertions for modeling and formal verification of architectural power intent'. Proc. 47th Design Automation Conf. (DAC), Anaheim, CA, 2010, pp. 773–776

10 Trummer, C., Kirchsteiger, C.M., Weiss, R., Dalton, D., Pistaur, M.: 'Simulation-based verification of power aware system-on-chip designs using UPF IEEE 1801'. Proc. 27th NORCHIP Conf., Trondheim, Norway, 2009, pp. 1–4

11 Magic Blue Smoke blog: http://synopsysoc.org/magicbluesmoke/2008/05

12 Meyer, B.: 'Applying design by contract', *IEEE Comput.*, 1992, **25**, pp. 40–51

13 Kramer, R.: 'iContract-the java design by contrect tool'. Proc. TOOLS26, Los Alamitos, CA, pp. 295–307

14 Rational Software Corp. and al.: 'Object constraint language specification, version 1.1', 1997, www.rational.com/uml/resources/documentation/formats.jtmpl