Efficient Power Intent Validation Using Loosely-Timed Simulation Models

Fabian Mischkalla, Wolfgang Mueller  
University of Paderborn, C-LAB  
Fuerstenallee 11, D-33102 Paderborn, Germany

Abstract—Faced with increasing demands on energy efficiency, current electronic systems operate according to complex power management schemes including more and more fine-grained voltage frequency scaling and power shutdown scenarios. Consequently, validation of the power design intent should begin as early as possible at electronic system-level (ESL) together with first executable system specifications for integrity tests. However, today's system-level design methodologies usually focus on the abstraction of digital logic and time, so that typical low-power aspects cannot be considered so far.

In this paper, we present a high-level modeling approach on top of the SystemC/TLM standard to simulate power distribution and voltage based implications in a "loosely-timed" functional execution context. The approach reuses legacy TLM models and prevents the need for detailed lock-step process synchronization in contrast to existing methods. A case study derived from an open source low-power design demonstrates the efficiency of our approach in terms of simulation performance and testability.

Keywords—SystemC/TLM, Loosely-Timed, IEEE 1801-2013 UPP, Power-Aware Simulation, Validation.

I. INTRODUCTION

Nowadays system-wide low-power techniques impact almost every electronic system design [1]. Furthermore, there is a trend towards the integration of more and more power management (PM) functionality into software (SW) layers working in close interaction with the hardware (HW). Due to this increasing complexity, applying traditional power-aware verification at late implementation stages is no longer sufficient. Rather early available electronic system-level (ESL) models should be used for validating low-power design decisions under realistic workload scenarios.

In fact, several use cases have been emerged related to power-aware modeling at system-level. As illustrated in Figure 1, they can be classified in terms of design scope and abstraction levels. The key driver for early chip/system planning activities is power analysis for power budgeting and coarse optimization. Later on at architectural level, power intent specification and exploration become more important. Finally, a correct implementation and synthesis of power control structures is the focus in traditional register-transfer-level (RTL) design flows.

In the last years, there has been a lot of research work [2, 3, 4] in the context of early power analysis. Consequently, several EDA vendors provide proprietary simulation driven environments [5, 6] for system-level power and energy estimation. Based on virtual prototypes (VP), most of these frameworks are suitable for full chip observation. For this, VPs simulate with limited time information and minor relationship to real implementation. They are developed using languages such as C/C++/SystemC and interact through standardized transaction-level modeling (TLM) interfaces. Power consumption is calculated by accumulating power values related to simulation events like transactions or operational states. However, most of these approaches are not intended to cover functional variations caused by low-power design intent. Furthermore, the individual components still behave in an always-on fashion. Thus, taking different supply levels or corrupted regions into account is not possible.

In traditional SoC design flows power models as specified by [7, 8] describe detailed power architectures on top of registers and wires as well as cycle-accurate semantic. For this, RTL behavioral models of power-aware cells and control elements are mandatory to verify correct working control sequences and advanced low-power design techniques. Given these capabilities, power-aware RTL simulators are highly effective, but may be too slow to cover the growing power state space of complex systems.

Today only very few approaches exist to instrument ESL models for early power intent simulation and validation. In [9, 10], for instance, the authors describe voltage and power sensitivity still based on detailed supply network topology. Thus, both modeling approaches are bottom-up oriented and assume that implementation and layout information are available at the time of modeling. Additionally, they do not explicitly cope with "loosely-timed" coding styles that often occur in early system specifications.

In contrast to latter approaches, we present a top-down abstraction method to capture power intent information free of any implementation details. For this, we developed i) a reference
architecture representing PM concepts at an early application-driven design stage, ii) a non-invasive instrumentation method working with the blocking TLM communication protocol, and iii) a dynamic and accurate power state synchronization mechanism. For architectural PM modeling, we extended well-known power models from IEEE 1801, also known as Unified Power Format (UPF) [7], with adequate properties and semantic for a “loosely-timed” functional simulation context. Additionally, we developed a model library that provides automatic validation support in terms of voltage-aware functional simulation and built-in assertions. Power estimation although conceivable as described by [10] was not in the focus of this work.

The remainder of this paper is organized as follows. The next section introduces basic power intent abstraction principles and outlines applicable validation scenarios. Section III explains modeling, implementation and simulation details, and in IV we describe our evaluation results. Finally, before concluding the paper, we compare our approach against related work in section V.

II. AN ESL LOW-POWER DESIGN METHODOLOGY

First low-power design decisions are often made by system engineers. Thus, recent modeling standards [7, 8] specify power intent initially in an implementation independent manner. On the other hand, they define only RTL semantic for power control and supply elements and voltage-aware simulation. However, an ESL abstraction and proper validation methodology can be derived as follow.

A. Low-Power Abstraction

From a system point of view low-power designs share common characteristics in order to realize PM techniques as shown in Figure 2. Working on system segments, appropriate architectures are divided into several regions. This is done by three types of domains, i.e., voltage, power, and clock domains, which are abstract concepts to group functional elements sharing common properties like frequency or voltage levels. At the topmost level there are voltage domains indicating different voltages as well as clusters of power domains. Each power domain then can be separately switched off or model virtual power supply sources for other domains. Finally, inner clock domains again enclose functional entities driven by the same clock signal.

Power domains have additional properties that must be described by explicit power intent rules. Such rules define implicit gate-level behavior providing several power management mechanisms. This includes level-shifters for signals between different voltage levels, as well as state retention and isolation strategies. Retention allows sequential data storage in case of power shutdowns, whereas isolation logic avoids floating values across power domain boundaries.

Since power architectures are static in nature, power supply may vary over time. There exist, for instance, dynamic voltage changes or several gating scenarios. Thus, at runtime functional elements may be in one of many possible power states also denoted as the underlying power management scheme (Figure 3). For early analysis these schemes are usually established in form power state tables. For this, system engineers rely on global power modes and transitions between these modes. Transitions wait for trigger events and usually take time. A system power mode is defined as a configuration of power domain states. A state of power domain again combines so-called nominal operating conditions (OC) and clock domain information. Additionally, an OC contains a discrete voltage level and a simstate determining finally the functional behavior which must be mimicked for power-aware simulation.

B. Power Intent Validation

Besides commonly requested power estimation, it is also important to ensure consistency of the underlying power intent specifications. Therefore, early power intent validation must focus on both a valid power architecture definition and the correctness of the dynamic power supply scheme. Additionally, an ESL approach should expose system-level PM bugs for reducing the state space and verification efforts in later design stages. Therefore, two questions are considered:

1) Does the power intent specification meet the requirements?

2) Does system behavior continues to operate properly?

Consequently, an ESL validation methodology needs to address validation use cases that can already be captured at higher abstraction levels. For this, we analyzed common RTL low-power design verification scenarios explained in [11] and derived adequate rules and guidelines for ESL. In addition, we developed appropriate assertion strategies for a loosely-timed TLM context. Thereby, the main focus was on a systematic integration without legacy code modifications and additional synchronization requirements. An overview showing the derived ESL subset is illustrated in table I. We defined five global
classes covering typical low-power design errors related to the before mentioned questions. For each class we provide several assertions divided into assert and cover directives that are explained later in section IV.

Finally, power-aware simulation as described in the following section together with these assertions enables automatic validation of power management specifications.

### III. Power-Aware TLM Simulation

Typical power-aware (PA) simulation is characterized by the following steps:

- identification of functional design elements,
- automatic instrumentation with power intent models, and
- dynamic adjustment to reflect power supply effects in the original simulation context.

For this, we developed a proof of concept framework mainly based on a C++ class library that augments SystemC/TLM loosely-timed models with power-aware behavior and systematic assertions. For the first step, transaction monitors, as proposed by [12], are automatically instantiated and configured at compile time. Afterwards instrumentation is done during SystemC’s elaboration phase by parsing Tcl based power intent specification commands. Finally, runtime observation provokes voltage-aware behavior at important power-aware timing points (PATP).

One advantage of TLM is the ability to speed-up simulation performance of standard system designs. For this, it abstracts communication protocols between SystemC entities and defines two coding styles: an approximately-timed (AT) and a loosely-timed (LT) coding style. TLM-AT is based on explicit process synchronization before global data accesses. TLM-LT allows active processes to run ahead of simulation time, also known as temporal decoupled execution semantic. Each process keeps track of its local time offset and synchronizes only if a predefined global time quantum is passed. TLM-AT is often applied for lower-level architectural exploration and concrete performance analysis, whereas early functional or application-driven architectural models, e.g., virtual platforms, are usually implemented at TLM-LT. However, even if TLM-LT enables to boost performance, functional accuracy cannot be assured for causal data dependencies during a quantum period. Therefore, it is often left to the system engineer to define an optimal global quantum, which is quite complicated and can vary from time to time.

For accurate power-aware simulation, transactions must be permanently aware of initiator and target power states [13]. If, for instance, a functional component reads from an entity that is actually switched off, it will get an incorrect reply instead of the correct power down response. Obviously, this is not acceptable in terms of functional simulation and validation. On the other side, processes that synchronize each time before sending messages will reduce the performance down to TLM-AT which is also not aimed. Therefore, an efficient power-aware simulation framework synchronizes only if necessary, i.e., only if PATPs are reached.

#### A. Implementation

An overview of the main classes for power intent abstraction as described in section II is given in table II. All components inherit from SystemC base classes (sc_module, sc_object) and rely almost on standard power modeling concepts specified in IEEE 1801-2013, but are constrained or extended for TLM. Internal properties are defined as attributes and relationships are modeled as associations meaning references to other instances. In particular, the focus was only on UPF’s conceptual and simulation layer, ignoring additional implementation details completely. Besides a description of power domains, power management specific rules, as well as different kinds of power states, we also provide adequate corruption semantic for TLM primitives. The goal was a close relationship to UPF conventions to maintain further application of the results as golden reference in later implementation and synthesis flows.

In general, low-power specifications are defined besides the functional system specification. For this, a power_model forms a unique container to cover all other objects. A power_model can be either at the top-level scope or be assigned to multiple functional components having the same power intent. Another central library element is the power_domain class abstracting a set of different physical supply functions. Moreover, a power_domain can be mapped to one or more TLM interfaces sharing the same abstract supply characteristics. Finally, some power_domains also define reference supplies for others or model abstract voltage regulator characteristics in terms of transition slope. The class clock_domain has actually no direct UPF equivalent, but add functional clock tree information in terms of active/non-active states and frequencies into the power model. In contrast, the UPF standard provides here only lower-level clock cycles references.

Voltage-aware semantic is provided by four runtime models for light-weight TLM communication data modification. First, we implemented an “output only” corruption model that modifies transaction data according to the state of the corresponding power_domain. At this moment, the model mimics the UPF standard normal and corruption semantic. However, since TLM payload values may have no undetermined ’X’ equivalent as required by UPF, the model provides abstract alternatives. By default, it changes the transaction’s metadata in terms of command and response attributes to TLM_IGNORE_COMMAND and TLM_GENERIC_ERROR_RESPONSE, respectively. Additionally, the model is able to adopt strategies specified on each power_domain in terms of all_high, all_low, or random payload data. The isolation and level shifting models correspond to UPF. Isolation functionality clamps transaction data to a predefined value and level-shifters act as simple time consuming buffers.

<table>
<thead>
<tr>
<th>Class</th>
<th>Desc.</th>
<th>Scope</th>
<th>Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Missing/redundant power rules</td>
<td>2</td>
<td>cover</td>
</tr>
<tr>
<td>2</td>
<td>Control sequence errors</td>
<td>2</td>
<td>assert/cover</td>
</tr>
<tr>
<td>3</td>
<td>Mode/state coverage</td>
<td>1</td>
<td>cover</td>
</tr>
<tr>
<td>4</td>
<td>Power domain ordering</td>
<td>1</td>
<td>assert</td>
</tr>
<tr>
<td>5</td>
<td>Behavioral maintenance</td>
<td>2</td>
<td>assert</td>
</tr>
</tbody>
</table>
Thus, even if there is no real level-shifting behavior needed, an appropriate modeling allows verifying missing and/or incorrect specifications. As TLM works only on IO memory and model internals are not fully register accurate, the retention model is yet restricted to entire TLM components instead of partial inner memory assignments. For this, it provides assertion directives for correct definition and sufficient supply. In addition, the current retention model enables a correct reset observation after power up situations.

Dynamic power intent modeling is based on hierarchical states at several levels. First, there are states for supply ports and/or nets which define supply voltage levels. In our framework, this is abstracted by a system-level operating_condition class avoiding detailed supply network models. However, this class provides also information in terms of primary voltage levels and simstates. On the next level, there are states for supply sets that group several nets together. For this, we developed a power state class that maps a power_domain and operational_condition together. Finally, there are system states specifying illegal or valid power domain configuration at any given point in time. This is modeled via power modes class. Additionally, a transition class indicates valid transitions between power modes.

### B. Instrumentation and Observation

The Instrumentation of TLM designs is completely done during SystemC’s elaboration phase. After a preprocessor-macro supported insertion of transaction monitors, the complete power model is generated on top of SystemC’s module hierarchy. For smart integration into existing low power design flows, the framework implements a Tool Command Language (Tcl) interpreter for IEEE 1801-2013 standard conform commands.

More details on this command subset can be found in the appendix. Using this interpreter, the framework reads user-defined power intent constraints and creates the complete power architecture by instantiating components from table II. Additionally, it provides a power management service API for handling and querying power model objects as well as changing comfortable between different power modes during simulation.

A small example is illustrated in Figure 4. The functional TLM design is composed of an initiator, a target, and the incorporated transaction handler. Such a handler executes as a TLM interconnect that forwards transactions from initiator to target, but also notifies registered power model objects. The sockets are counterparts to the sockets of connected TLM components which can automatically be determined at compile time. For simplification, the small example is arranged in one power domain, thus only two corruption models, one for forward and one for backward path, has been generated and bound to transaction delegates. However, since TLM components might be in different power domains, up to four different corruption models are conceivable, two for each socket representing the sender and receiver supply in each case. In this example, there is also only one power mode specified which defines the power domain pd1 in an OFF power state.

In contrast to UPF’s corruption semantic, TLM models and their inner functionality cannot be easily deactivated in case of power-down occurrence. An abstract paused process that still has to drive other powered-on TLM interfaces would lead to incorrect behavior simulation. Instead, as already mentioned, our framework provides light-weight TLM data modifications as outlined in Figure 6. In particular, we avoid deep payload copies since this would significantly decrease simulation performance. For instance, if power mode m1 is requested, the state of power domain pd1 changes immediately to OFF. Thus, ongoing

### Table II: Basic power intent modeling classes

<table>
<thead>
<tr>
<th>Class (base)</th>
<th>Attribute</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>power_model (sc_module)</td>
<td>-</td>
<td>domains, clks, conditions, modes, transitions</td>
</tr>
<tr>
<td>power_domain (sc_object)</td>
<td>slope</td>
<td>elements, ports, base, states</td>
</tr>
<tr>
<td>clock_domain (sc_object)</td>
<td>freq, state</td>
<td>domain</td>
</tr>
<tr>
<td>corruption (sc_object)</td>
<td>explicit</td>
<td>domain</td>
</tr>
<tr>
<td>isolation (sc_object)</td>
<td>-</td>
<td>domain, supply, clamp</td>
</tr>
<tr>
<td>retention (sc_object)</td>
<td>-</td>
<td>domain, supply</td>
</tr>
<tr>
<td>level-shifting (sc_object)</td>
<td>-</td>
<td>from, to, rule, supplies</td>
</tr>
<tr>
<td>operating_condition (sc_object)</td>
<td>voltage, simstate</td>
<td>-</td>
</tr>
<tr>
<td>power_state (sc_object)</td>
<td>-</td>
<td>domain, op, clks</td>
</tr>
<tr>
<td>transition (sc_object)</td>
<td>-</td>
<td>from/to, clk</td>
</tr>
<tr>
<td>power_mode (sc_object)</td>
<td>illegal</td>
<td>states, trans</td>
</tr>
</tbody>
</table>

Figure 4: Non-intrusive TLM monitoring

Figure 6: Power-aware TLM datapath modifications
transactions will be corrupted in terms of their response status. Moreover, all further transactions will be sent as messages to be ignored. Even though this models no real event deactivation as proposed by IEEE 1801, the modifications are sufficient for proper power-aware validation.

C. Adaptive Power-Aware Synchronization

Power-aware simulation expects that TLM transactions are aware of ongoing power domain states for each participating communication entity. In other words, operational conditions must be accurately modeled. This implies, without explicit knowledge of state transition events, that all SystemC processes have to synchronize with the global time before and after a transaction call. As a result simulation performance would reduce to TLM-AT. However, for early PM exploration, TLM-AT performance might be not available or even too slow, so that TLM-LT abstraction is often a desirable compromise. For this, a simulation layer must rely on synchronization requirements that sufficiently cover potential power states changes of power domains, also called power-aware timing points (PATP). Assuming a power domain model as shown in Figure 5, PATPs can be automatically derived.

For a loosely-timed execution semantic, the TLM standard provides a quantum-keeper class by default. It synchronizes only if accumulated time offsets exceeds a local process quantum which again is derived from a firm global quantum period. However, based on a static global quantum, the quantum-keeper class is not applicable for ever-changing data dependent synchronization needs. Therefore, we developed a mechanism that overlays existing quantum-keepers in TLM designs and automatically adapts quantum periods with regards to PATPs. Consequently, TLM components become untimed and synchronization tasks are forced to power models. Subsequently, a next synchronization point can be calculated by means of future PATPs. As shown in Algorithm 1 this is either based on the end time of an ongoing transition $T$ or on the earliest possible switching time of current power state $S$. Surely, in both cases the user-defined global quantum is still considered as upper bound.

To illustrate this, Figure 5 outlines an exemplary TLM communication sequence of three subsequent interface message calls (IMC). The time progress is represented without time units. The sequence starts at time 100 in power domain state S1. The local quantum is 60 due to the lower bound of transition $T_{1-2}$. Accordingly, the first IMC succeeds without additional synchronization. It takes only two checkable checks one on each path for synchronization needs. During the next IMC, simulation reaches a PATP and requires synchronization. The current time increases to 200 and state transition event $e_1$ has been noticed at 180. Consequently, the new local quantum is 40. Afterwards, local time is reset to zero and simulation proceeds. The last sequence synchronizes again, so meanwhile current state is already S3. Thus, the next PATP is 300, as earliest possible state change could happen 10 units later. Finally, as algorithm computation time and additional checks are faster than TLM-AT, overall simulation performance is higher.

While this approach keeps important power state information, it is certainly not appropriate for more fine-grained signal-based PM analysis. In fact, the underlying abstraction level permits still functional inaccuracy at timing points either not covered by PATPs or the global quantum period. However, the underlying power models are not restricted to any abstraction level and can be also applied down to cycle-accurate TLM models. Nevertheless, this causes extra amount of synchronization overhead that might not be required in early design phases.

IV. Experimental Results

The following evaluation relies on an open source low power design from [14]. Originally developed at RTL, the implementation has been translated almost one-to-one to SystemC/TLM. Indeed, contradictory to the original intent of TLM abstraction, this permits to compare our methodology with existing power-aware simulation environments. For this, we applied Mentor Graphics’ QuestaSim together with its UPF support as reference simulator. With main focus on low-power concepts, the overall system functionality is kept very basic. Nevertheless, as the functional entities produce frequent interaction, it is most suitable for an evaluation on top of TLM.

A. Application Scenario

The system is composed of five successive components and produces a binary-coded output via sequence detection on a linear feedback shift register. Using the TLM library from OSCI, each component has been equipped with in- and outgoing blocking interfaces. Furthermore, we wrapped the behavior of each component into a SystemC thread. Each process is also coded in a loosely-timed (LT) as well as a cycle-accurate (CA) variant. In both cases computation, communication, and
synchronization is hooked on clock cycle periods as outlined in Listing 1 and Listing 2.

Additional low-power intent is specified via standard UPF commands with minor adaptations (see the Appendix) for ESL. It is based on six power domains, an abstract supply distribution description, and additional power management mechanisms like isolation, retention, and level-shifting. In particular, there is one wake-up power domain and five switchable subdomains, one for each component. One subdomain is either in a "on" or "off" state. The remaining four subdomains, driven by the same clock signal, are either simultaneously "on-high" or "on-low". Additionally, each can be also independently "off". Due to this, a total number of 62 valid power modes and appropriate transitions have been defined.

```c
while(1)
{
    // functional code
    ...
    offset += period;
    out->b_transport(txn,
        offset);
    ...
    if(offset >local_quantum)
    {
        wait(offset);
        offset = SC_ZERO_TIME;
    }
}
```

### Listing 1: TLM-LT coding style

```c
while(1)
{
    // functional code
    ...
    wait(period);
    out->b_transport(txn,
        SC_ZERO_TIME);
    ...
}
```

### Listing 2: TLM-CA coding style

**B. Performance Analysis**

For a synthetic test scenario, we implemented a stimulus pattern that addresses the complete range of power state configurations and tests some functional requirements in each step. Afterwards, we compare our framework with QuestaSim’s power-aware functional simulator. For this, in a first experiment we run pure functional as well as power-aware simulations on both RTL and TLM-CA models. Figure 7 (a) shows the results in terms of CPU time normalized to the functional RTL simulation. Since the TLM-CA coding style has no real abstraction, this variant is even a little bit slower than RTL. However, considering only the relative simulation overhead induced by additional power intent it is close to RTL. In general, we see that a cycle-accurate simulation extended with our power models is about two times slower than the original design. Nevertheless, as this factor complies with the overhead of a state-of-the-art RTL simulator it should be acceptable.

In a second experiment, we evaluated performance impacts on more abstract TLM scenarios, i.e., using the loosely-timed coding style. Therefore, we apply the TLM-LT implementation with a global quantum value able to simulate the sequence detection rate. Power-aware simulation was then performed by different synchronization constraints in terms of power state transition delays. In particular, we simulated with the following synchronization schemes: (1) on each communication message start and end point, (2) in between each global quantum period and (3) randomly, but not in every quantum period. In 1), indicating worst-case scenario, the simulation overhead will not exceed that of power-aware TLM-CA models. However, if power intent specifications become more realistic, the overhead reduces significantly.

**C. Validation Survey**

Traditional power-aware verification is based on property checking using executable RTL models. For this, QuestaSim provides built-in rule checkers that cover typical use cases to assist low-power design verification as described in [15]. Usually, these checkers are RTL assertions automatically inserted into the model. For supporting evidence, we implemented appropriate rules into our proof-of-concept framework. Thereby, only use cases verifiable at power-aware timing points can be considered and are implemented by C++ assertions. The checks performed by our framework are shown in table III. Hereby, we differentiate ESL and implementation dependent (RTL) checks as latter require information either not modeled in early functional system specifications or not accessible from interfaces of components. Furthermore, we name the global validation class to which the checker belongs. Consequently, using our approach a system engineer is able to cover up to 67% of the overall amount and even 90% of all system-level checks supported by QuestaSim.

**V. RELATED WORK**

The term low power design as initially understood by the electronic industry is mainly related to power intent implementation in traditional hardware synthesis flows. Hence, PM logic and different power supplies permitting advanced power reduction techniques are usually included at RTL [16].
or even later in physical models. For this, several EDA vendors offer tools and model libraries which automatically instrument legacy RTL models for power-aware simulation. The tools insert implicit supply networks as well as executable models mimicking dedicated power cell behavior on RTL primitives like registers, latches, and wires. Since 2006, there are also design standardization efforts [7, 8] mainly based on former proprietary formats. Hereby, power intent is specified by Tool Command Language (Tcl) extensions in side-files applicable throughout the entire RTL-based synthesis process. Recently, there are also alignment activities [17] as both standards have similar concepts. However, as these concepts are outlined for RTL-to-GDSII design flows, the kind of power intent abstraction is not well-suited for early system-level design stages. In particular, there is neither a mapping of power state simulation semantic nor a system-level interpretation of pin/register related rules for retention, isolation, level shifting.

In case of typical verification scenarios, as described in [11, 18, 19], issues like corruption or power state switching sequences must be investigated. For this, verification engineers rely on RTL property checking techniques. Appropriate rules are either implemented as simulator built-in checks or user-definable by assertion languages like SystemVerilog or the Property Specification Language (PSL) [20]. Besides simulation approaches, there are also semi-formal methods [21], but still require RTL logic as input.

Simulation and validation of power intent at system-level using higher abstraction models is certainly a very novel area and there exist only very few solutions. The first SystemC based solutions continue to be limited to RTL conform modeling constructs. In [22] for instance they explicitly adapt the inner SystemC kernel for power aware simulation on primitive hardware data types and clocked processes. In [23], they generate pin-accurate PM structures according to deprecated UPF syntax and execute them in conjunction with cycle-accurate SystemC models. A first adequate abstraction approach for TLM has been proposed in [13]. However, while the author provides only general remarks, [24, 25] propose applicable solutions for TLM coding styles. [24] describes power aware virtual prototyping by means of supply-, reset- and clock tree modeling, and [25] proposes cycle-approximate UPF power models. Nevertheless, as both methods still require implementation details they are targeting more bottom-up oriented platform-based design flows. For instance, both model the supply net topology (ports, nets, switches) instead of abstract representations starting at system level. Moreover, component interfaces connected to multiple inner power domains are not addressed. Finally, none of these abstract methods consider dynamic power state synchronization. Thus, accurate voltage awareness is either lock-step dependent or not obtainable.

### VI. Conclusion

In this paper, we presented a novel approach for power intent abstraction in early available "loosely-timed" functional system specifications. This closes the gap in existing system-level design methodologies to model and reflect low-power intent design aspects under realistic workload scenarios. For power-aware modeling, we adopted the newest IEEE 1801 low-power design and verification standard with rules and semantic for SystemC/TLM. The implemented framework imposes then automatically non-intrusive runtime power models into legacy TLM designs and overlays the default functionality with voltage-aware behavior. Additionally, all runtime modifications are efficiently performed only at power-aware timing points derived from user-definable power intent constraints. The experimental results have shown that the performance shrinkage caused by the instrumentation itself complies with the overhead in proprietary power-aware RTL simulators. Thus, the instrumented models achieve execution times that are at most 2 times slower than that of the original models. Moreover the approach supports power state accuracy in a "loosely-timed" simulation context, but consumes less execution time than in alternative lock-step simulation. Finally, testability has been approved by means of a coverage analysis on common low-power design errors. Thereby, up to 67% of built-in checks provided by a state-of-the-art RTL-based reference simulator could be incorporated.

Since the proposed validation solution is mainly simulation-based, we see promising enhancements by including static power intent checks at compile time. For this, we are currently investigating whether our power intent abstraction models satisfy appropriate demands. For this, potential application scenarios might be power domain dependency scans, power mode completeness and integrity proofs, as well as missing or redundant rules for domain crossings. These techniques along with power-aware simulation capabilities would form then a complete ESL front-end complementing existing low-power design processes.

### Appendix

Listing 39 shows an excerpt of the top-level power intent specification used during evaluation. The Tcl commands mostly comply with IEEE 1801-2013 with minor adaptation denoted by emphasized text. In summary we define power domains, nominal operating conditions, and power management strategies without details involving supply network and implementation knowledge originating from later design stages.

### Table III: Overview of built-in power intent validation checks

<table>
<thead>
<tr>
<th>Check</th>
<th>Layer</th>
<th>Support</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level Shifter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing/Incorrect</td>
<td>ESL</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>Isolation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing</td>
<td>ESL</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>Power On</td>
<td>ESL</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Enable</td>
<td>ESL</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Disable</td>
<td>ESL</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Race/Functionality</td>
<td>ESL/RTL</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Clamp Value/Toggle</td>
<td>RTL</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Retention</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Off</td>
<td>ESL</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Power On</td>
<td>ESL</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Level/Toggle</td>
<td>RTL</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standby Corruption</td>
<td>ESL</td>
<td>Yes</td>
<td>5</td>
</tr>
<tr>
<td>Primary Supply</td>
<td>ESL</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>Domain Status</td>
<td>ESL</td>
<td>Yes</td>
<td>3,4</td>
</tr>
<tr>
<td>Glitch Detection</td>
<td>RTL</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Register Reset</td>
<td>RTL</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Acknowledgments

This work was partially funded by the German Ministry of Education and Research (BMBF) through the project ARAMIS (01IS11035). We greatly appreciate the cooperation with the project partners.

References


