Formal Verification of Architectural Power Intent

Aritra Hazra, *Student Member, IEEE*, Sahil Goyal, Pallab Dasgupta, *Senior Member, IEEE*, and Ajit Pal, *Senior Member, IEEE*

***Abstract*—This paper presents a verification framework that attempts to bridge the disconnect between high-level properties capturing the architectural power management strategy and the implementation of the power management control logic using low-level per-domain control signals. The novelty of the proposed framework is in demonstrating that the architectural power intent properties developed using high-level artifacts can be automatically translated into properties over low-level control sequences gleaned from UPF specifications of power domains, and that the resulting properties can be used to formally verify the global on-chip power management logic. The proposed translation uses a considerable amount of domain knowledge and is also not purely syntactic, because it requires formal extraction of timing information for the low-level control sequences. We present a tool, called `POWER-TRUCTOR` which enables the proposed framework, and several test cases of significant complexity to demonstrate the feasibility of the proposed framework.**

***Index Terms*—Assertion, formal verification, low-power verification, power intent verification.**

I. Introduction

In order to meet the stringent power budgets of low-power digital integrated circuit designs, several power management techniques have evolved and have been used in industrial practice [6], [9]. This includes techniques for managing dynamic power such as clock gating, voltage and frequency scaling, and managing leakage power such as power gating and adaptive body-biasing [18], [26]. In a complex architecture, a combination of these strategies may be used for better power management.

In most complex integrated low-power circuits, the power management strategy is laid out at the micro-architectural level [4], [18]. Several tools have been developed for early estimation of power performance, exploring alternative strategies and converging on the most efficient one. These include tools like Turandot/MET [30], PowerTimer [8], Spyglass-Power [28], etc. Power performance analysis typically leads to the development of a *global* power management strategy, which demarcates the boundaries of various architectural power domains and specifies the properties relating these power domains at a high-level of abstraction. The architectural power management strategy may entail simple properties (such as domain-A and domain-B will not be active simultaneously) or complicated properties, such as ones which specify start-up sequences between the power domains in a complex system-on-chip (SoC).

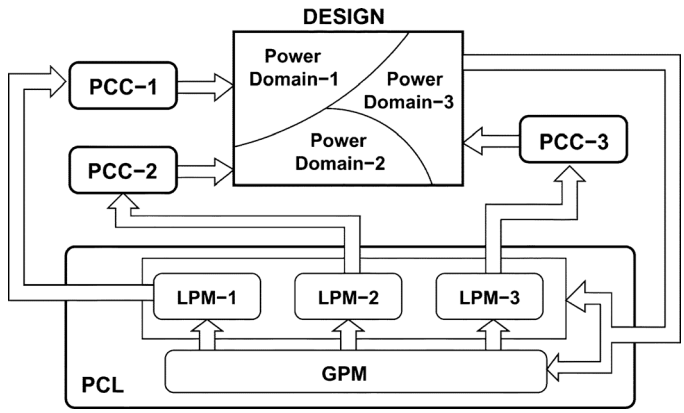

Fig. 1. Power-managed design components.

Though global power management strategies are designed up front, they can be implemented only much later in the design flow. This is because, power intent specification is not supported in high-level design languages such as Verilog [15] or SystemVerilog [29], which are used to enter the design implementation at behavioral or register-transfer level (RTL).

Typically, from a power management perspective, a low-power digital integrated circuit can be viewed as consisting of the following three components, as shown in Fig. 1.

1) The digital logic of the circuit, partitioned into a set of power domains. We shall call this the *design*.
2) The power control interface, consisting of isolation cells, voltage regulators, frequency converters, level shifters, switches and retention cells. We call these the *power control circuitry* (PCC).
3) The power control logic, which drives (digital) control inputs to the PCC to effect changes in the power states of a power managed domain. We shall call this the *power control logic* (PCL). For large circuits, the PCL consists of two entities, namely:
   a) A set of per-domain *local power managers* (LPM). Each LPM is responsible for issuing valid control sequences for enabling transitions between the power states of that power domain.
   b) The *global power manager* (GPM) which orchestrates the LPMs to implement the global power management strategy.

The above architectural view from a power management perspective has become quite accepted among architects of digital integrated circuits [16], [20].

Manuscript received March 28, 2011; revised August 19, 2011 and October 24, 2011; accepted December 07, 2011. Date of publication January 17, 2012; date of current version December 19, 2012. This work was supported by Synopsys Inc. under Synopsys-IITKGP CAD Laboratory Projects. The work of A. Hazra was supported by Microsoft Corporation and Microsoft Research India under the Microsoft Research India Ph.D. Fellowship Award.

A. Hazra, P. Dasgupta, and A. Pal are with the Indian Institute of Technology, Kharagpur, India, West 721302 Bengal, India (e-mail: aritrah@cse.iitkgp.ernet.in; pallab@cse.iitkgp.ernet.in; apal@cse.iitkgp.ernet.in).

S. Goyal is with Barclays Capital, SG—018983 Singapore (e-mail: sahil.kgp@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2011.2180548

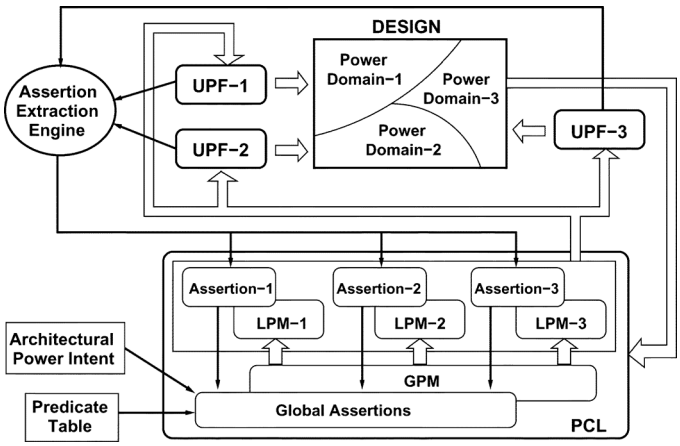

Fig. 2. Schematic flow for architectural power intent property generation using UPF-extracted assertions.

Typically, both the design and the PCL can be expressed in digital logic and can be coded using hardware description languages, such as Verilog or VHDL. Since the power lines, voltage regulators or level-shifter circuits cannot be described in RTL, it is not possible to directly express the PCC in RTL. Therefore in the traditional design flow, the PCL can be integrated with the design very late, that is, after the design has been synthesized and the PCC has been added into the netlist. Verifying the PCL becomes challenging due to the cost of simulation at this level of abstraction.

The focus of this paper is on verifying the PCL at a higher level of abstraction, namely at the register transfer level. The golden reference for this verification should be the architectural properties capturing the global power management strategy. However, architects are able to specify the power management strategy through high-level artifacts, which include names associated with architectural power domains and names associated with the power states of a domain [1]. On the other hand, the control signals driven by the PCL to the PCC are low-level signals controlling specific actions like isolation and retention of individual power domains. Our main contribution in this paper is to bridge the existing disconnection between the architectural level artifacts and the low-level control signals over which the PCL is defined, thereby paving the way for high-level validation of the PCL using formal verification.

The novelty of the proposed framework is in automating the translation of assertions developed using architectural power management artifacts into assertions defined over the signals of the PCL, which are then formally verified over the PCL. This translation takes advantage of the emerging industrial adoption of languages such as unified power format (UPF) [31] and common power format (CPF) [25] for describing the PCC at a higher level of abstraction [12]. The components of the proposed verification tool flow are as follows.

1) We present a novel extension of UPF, which may be used to declare the architectural power management artifacts, including the names of the architectural power domains and the names and nature of the power states for each power domain.
2) We present a novel framework for developing inter-domain power management properties using high-level power state predicates and power state transition predicates which are automatically constructed from the declaration of architectural power domains.
3) We present a novel methodology for automatically extracting timed control sequences corresponding to these predicates from per-domain UPF specifications. The LPM logic for the domain is formally analyzed to extract the accurate timing for these control sequences. This step of extracting accurate timing information has important ramifications towards the scalability of our approach, as described later.
4) We present a novel translation procedure for rewriting the inter-domain architectural power management properties using the per-domain control sequences extracted from UPF.
5) We present a tool flow integrating the above (as shown in Fig. 2), in which the formal assertions generated by our approach are verified formally using standard off-the-shelf industrial strength model checking tools.

The notion of formally verifying the architectural power intent was introduced in an earlier paper [13]. This paper presents the complete framework for this verification task and experimental results on more advanced test cases. In this paper, we introduce a key step into our approach, that is, extracting accurate real-time bounds on the per-domain sequence expressions, before they are used in the architectural power assertions. We present results to demonstrate the degradation in the performance of formal verification task when this step is absent, and present an approach for extracting the per-domain time bounds. This paper also extends our earlier work by including artifacts of more advanced power management strategies such as dynamic voltage and frequency scaling (DVFS) and adaptive body biasing.

Verification of low-power designs [17] involves verification of the design in multiple power states, and verifying that only the intended transitions and sequences of transitions have occurred. The proposed approach not only enables formal analysis of reachable power states and transitions, but also bridges the disconnection between high-level architectural power intent properties and low-level per-domain sequencing properties extracted from UPF, thereby paving the way for formal verification of the PCL.

Existing techniques for power-aware simulation [11] and static analysis related to state retention bugs [5], [7], [10], [24] are typically employed low down in the design flow. Recent attempts towards tackling verification problems for power management include several state-of-the-art procedures for assertion extraction from UPF [19], [27]. These methods do not address the problem of verifying architectural power intent assertions and therefore are not suitable for verifying the global power management strategy. Our method is possibly the first attempt to solve the problem of verifying the global power management strategy in a formal verification setting.

II. Overview and Tool Flow

Fig. 3 illustrates the main steps in the proposed approach for formal verification of the PCL. This flow is enabled by our tool,

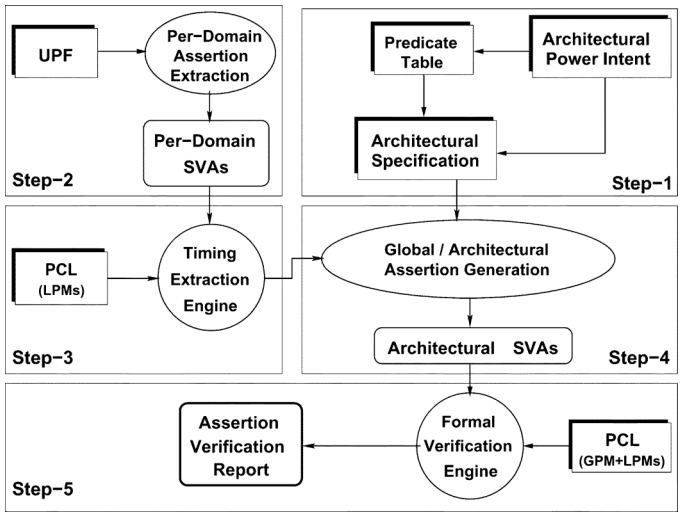

Fig. 3. POWER-TRUCTOR tool-flow for assertion synthesis and formal assertion verification.

called POWER-TRUCTOR. Each of these steps are based on several novel contributions outlined in the following sections. In this section, we present a summary of these steps, and then elaborate them through the subsequent sections.

The tool-flow for POWER-TRUCTOR (see Fig. 3) has the following five major steps.

Step 1) *Formalizing the Architectural Power Intent*. For capturing the architectural power management strategy, we need a set of high-level artifacts, such as architectural power domains and power states. We extend the syntax of UPF to enable the declaration of the names of the power domains and the power states in each domain. Given this declaration, our tool automatically generates a set of predicates for power states and transitions, which serves as the propositions using which the formal properties capturing the architectural power intent can be developed. Section III elaborates this step.

Step 2) *Control Sequence Extraction from UPF*. This step generates the low-level control sequences for enabling local power state transitions in individual power domains. The tool accepts the UPF specification of the individual power domain and uses a considerable amount of built-in domain knowledge to automatically extract these sequences in SystemVerilog Assertions (SVA) [29]. This step is carried out for each power domain individually. Section IV elaborates this step.

Step 3) *Timing Extraction for Sequences*. This step uses formal methods for analyzing the accurate time boundaries for the control sequences extracted in Step 2). The control sequences are automatically refined with this analysis. Section V elaborates this step. This step has a significant impact on the performance of Step 5), which is supported through experimental results presented in Section VII.

Step 4) *Global Assertion Generation*. The assertions developed in Step 1) use high-level predicates. These predicates must be translated into low-level control sequences so that the resulting assertions can be formally verified on the PCL. Section VI elaborates this step.

Step 5) *Formal Verification*. In this step, standard industrial strength model checking tools, like *Magellan* [21], is used to formally verify the PCL with the assertions generated in Step-4. Experimental results on several test cases are presented in Section VII.

At the end, POWER-TRUCTOR invokes Magellan to produce an assertion verification report of the generated assertions, where the success and failure of the synthesized assertions are presented. It may be noted that POWER-TRUCTOR uses Magellan only at the back end, and can use any other model checking tool as well.

Steps 1)–4) enable Step 5), that is, our objective of formally verifying the PCL with respect to the architectural power management strategy. These steps are necessary for the following reason. Model checking tools take two inputs—a formal property and a finite state machine (formally called a Kripke structure). The formal property must be defined over the labels of the Kripke structure, that is, the atomic propositions used in the property are functions of the state variables. In our problem, this is not the case. The global power management properties are defined by chip architects using architectural artifacts, that is, the names of the power domains and the names of the power modes. On the other hand, the Kripke structure derived from the RTL description of the PCL does not have these names as labels. Instead, the states of the PCL are labeled by low-level control signals for controlling local power control circuitry, such as power gating, isolation and retention. Therefore, the labels of the Kripke structure states are different from the propositions used in the formal properties capturing the power management strategy, and there is no Boolean map between them. We cannot use the model checker directly for this reason.

We must, therefore, rewrite the formal properties so that they are defined over the labels of the Kripke structure, while capturing the same design intent. This task of rewriting is non-trivial because each high-level transition between power states involves a non-trivial control sequence at the low-level spanning many cycles. Steps 1)–4) automate this translation.

III. Formalizing the Architectural Power Intent

We demonstrate the idea of specifying the architectural power intent through an example. Let the overall power architecture of a design (single-core *eLeon*3) consist of *seven* power domains, namely, a *primary integer unit*, a *secondary integer unit*, a *multiplication unit*, a *division unit*, a *memory controller unit*, a *cache unit*, and a *storage elements unit*. Fig. 4 demonstrates different power domains (illustrated using different colors/shades) of the *eLeon*3 design.

The architect decides the following power modes for these power domains.

1) The primary integer unit (PIU) acts in three power modes, namely, *ACTIVE*, *IDLE*, and *OFF*.
2) The secondary integer unit (SIU) also acts in three power modes, namely, *ACTIVE*, *IDLE*, and *OFF*.
3) The multiplication unit (MULT) operates in two power modes, namely, *ON* and *OFF*.

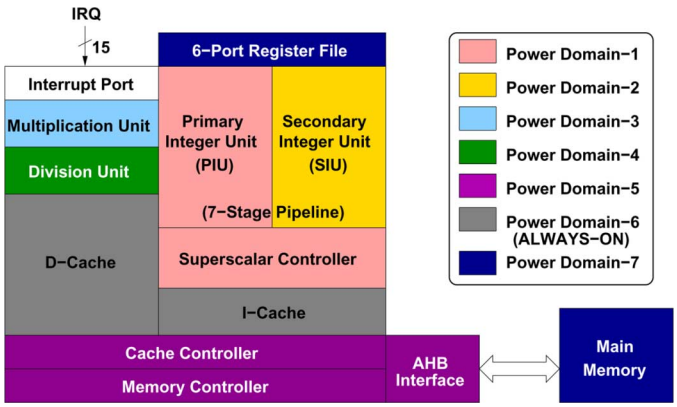

Fig. 4. Different power domains of *eLeon*3 processor.

4) The division unit (DIV) also operates in two power modes, namely, *ON* and *OFF*.
5) The memory controller unit (MEM_CTLR) acts in two power modes, namely, *ON* and *OFF*.
6) The cache unit (CACHE) operates only in two power modes, that is, *FULL_ON* and *PARTIAL_ON*.
7) The storage elements unit (STORAGE_ELM) acts in two power modes, namely, *ON* and *OFF*.

Among all the power modes of these power domains, the *ACTIVE* power modes of both PIU and SIU power domains are characterized using higher voltage and frequency pairs compared to *IDLE* power modes of PIU and SIU, which are characterized using relatively lower voltage and frequency pairs. In our platform, the architect uses two constructs to specify this architecture, namely:

- *create_power_domains*: specifies power-domain names;
- *create_power_states*: specifies different power states for a particular domain and mentions their types (on/off). The power states are characterized using the following three flags:
  1) *-voltage* flag: specifies the voltage-level of a power state;
  2) *-frequency* flag: specifies the frequency-level of a power state;
  3) *-bias* flag: specifies the bias voltage-level of a power state.

For example, the architectural power intent specification of *eLeon*3 is as follows:

```
begin_power_architecture(eLeon3)
   create_power_domains {PIU SIU MULT DIV
           MEM_CTLR CACHE STORAGE_ELM}
   create_power_states -domain PIU
      -on_state {ACTIVE -voltage Vh_piu
              -frequency Fh_piu}
      -on_state {IDLE -voltage Vl_piu
              -frequency Fl_piu}
      -off_state {OFF}
   create_power_states -domain SIU
      -on_state {ACTIVE -voltage Vh_siu
              -frequency Fh_siu}
      -on_state {IDLE -voltage Vl_siu
              -frequency Fl_siu}
      -off_state {OFF}
   create_power_states -domain MULT
      -on_state {ON -voltage V_mult}
      -off_state {OFF}
   create_power_states -domain DIV
      -on_state {ON -voltage V_div}
      -off_state {OFF}
   create_power_states -domain MEM_CTLR
      -on_state {ON -voltage V_mem_ctlr}
      -off_state {OFF}
   create_power_states -domain CACHE
      -on_state {FULL_ON -voltage Vh_cache}
      -on_state {PARTIAL_ON -voltage Vl_cache}
   create_power_states -domain STORAGE_ELM
      -on_state {ON -voltage V_storage_elm}
      -off_state {OFF}
end_power_architecture
```

The above example does not incorporate body-bias voltage for any domain of *eLeon*3. We can, though, easily extend the architectural power intent description to include the same using *-bias* flag. The specification of this flag is syntactically similar as for the other *-voltage* or *-frequency* flags.

Implicit in this specification are the potential transient states between an *on_state* and the *off_state* of a power domain, namely, as follows.

- *Isolation-Enabled*. The domain has been isolated by setting isolation values at its interface.
- *Isolation-Disabled*. The isolation at the interface of the power domain has been removed.
- *Retention-Completed*. The state of the domain has been saved in the retention registers.
- *Retention-Restored*. The state of the power domain has been restored from the retention registers.

Apart from these transient states, there can be other intermediate states between two *on_states*, which are characterized using different voltage-frequency pairs. This is because, the transition from a high voltage-frequency state to a low voltage-frequency state will take place through a high voltage and low frequency state because the frequency has to be lowered strictly before the voltage is lowered. Likewise, the transition from a low voltage-frequency state to a high voltage-frequency state will pass through a high voltage and low frequency state. A low voltage-high frequency state will typically be rendered unreachable by the strategy.

It may be noted that the transient or intermediate states are potentially possible, though a specific implementation may not have one or more of these states. Figs. 5 and 6 present the *envisaged* power state machine for the LPM of the PIU-domain and MULT-domain in *eLeon*3, respectively.

A. Generation of Predicate Table

Our platform creates a table of predicates from the power architecture specification. These predicates are of the forms:

```
⟨power-domain ⟩ (state) OR ⟨power-domain⟩ (state, state)
```

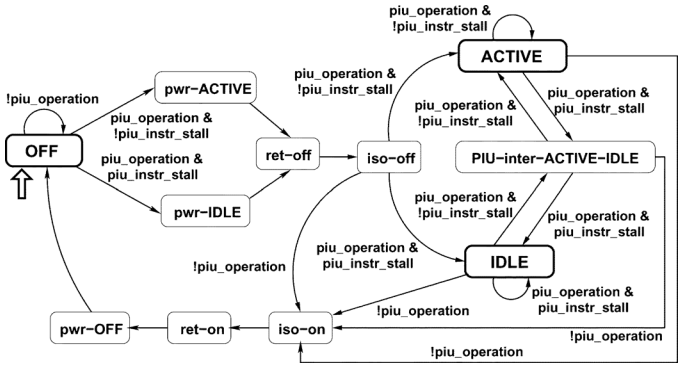

Fig. 5. Power state machine for LPM of PIU-domain in *eLeon*3 design.

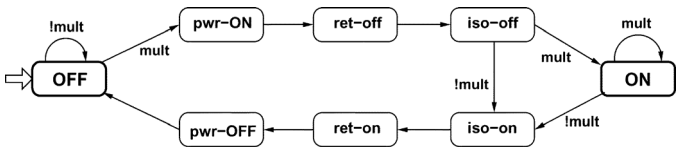

Fig. 6. Power state machine for LPM of MULT-domain in *eLeon*3 design.

where *power-domain* is the name of the power domain and *state* represents a specific power-state or transient-state of that domain. For example, for the PIU domain in our example, we have two on-states (*ACTIVE* and *IDLE*) and therefore, two potential power-up sequences, namely, PIU(OFF, ACTIVE) and PIU(OFF, IDLE); and two potential power-down sequences, namely, PIU(ACTIVE, OFF) and PIU(IDLE, OFF). These predicates become true when the corresponding sequence takes place. Additionally, we have the following predicates for capturing the transient states of the PIU domain.

- *PIU(ACTIVE), PIU(IDLE), PIU(OFF)*. These predicates become true when the power domain, PIU, reaches the *ACTIVE*, *IDLE* and *OFF* states, respectively.
- *PIU(pwr-ACTIVE), PIU(pwr-IDLE), PIU(pwr-OFF)*. These predicates become true when the PCC has applied the appropriate voltage on the supply rails. In transitions from the off-state to an on-state, this is followed by restoring the retention values and disabling the isolation.
- *PIU(iso-on), PIU(iso-off)*. PIU(iso-on) becomes true when the power domain, PIU, reaches the state where isolation has been enabled during power-down. PIU(iso-off) becomes true when the domain reaches the state where isolation has been disabled during power-up.
- *PIU(ret-on), PIU(ret-off)*. These are similarly defined for retention completion (during power-down) and retention restoration (during power-up) events, respectively.

Apart from the predicates for these power up/down sequences, power states and transient states, we also have some predicates capturing the intermediate state sequences. For example, for the PIU domain in our example, we have two on-states (*ACTIVE* and *IDLE*) and therefore, one *intermediate state*, namely, *PIU-inter-ACTIVE-IDLE* (shown in Fig. 5) and two potential predicates for the intermediate state transition sequences, namely, *PIU-inter(ACTIVE, IDLE)* and *PIU-inter(IDLE, ACTIVE)* (as indicated in Table I). The intermediate state, namely `PIU-inter-ACTIVE-IDLE` for the PIU-domain of *eLeon*3), may be reached during the transition between two on-states, which are characterized by different voltage-frequency levels.

The complete set of predicates generated by our tool for our example is shown in Table I. The predicates are based on the envisaged power state machines of the LPMs. When the UPF of these domains become available, SVA sequences corresponding to each predicate are generated automatically. At this stage, it may be discovered from the UPF specification that some transient state is actually not reachable. For example, the MEM_CTLR unit of *eLeon*3 did not have a retention mechanism, which was discovered from the UPF specification. Therefore the predicates *MEM_CTLR(ret-on)* and *MEM_CTLR(ret-off)* were removed from Table I after processing the UPF.

Similarly, domains controlled only by multiple voltages (having no frequency control) will have no intermediate states. In Table I the power domains MULT, DIV, MEM_CTLR, STORAGE_ELM, and CACHE belong to this category.

B. Expressing Architectural Power Intent

The architectural power intent expresses the global power management strategy between the power domains. For example, in the *eLeon*3 design, some of the power intent properties are as follows.

1) The multiplication unit (MULT-domain) and the division unit (DIV-domain) are never ON together.
2) The integer units (PIU and SIU-domains) cannot be ON together with the multiplication unit (MULT-domain).
3) The integer units (PIU and SIU-domains) cannot be ON together with the division unit (DIV-domain).
4) Only after putting the memory controller unit (MEM_CTLR-domain) fully powered-up, the storage-elements (STORAGE_ELM-domain) are powered-up.
5) Only after the storage-elements (STORAGE_ELM-domain) finish their restoration, the power of the integer units (either PIU and/or SIU) can be given.

The predicate table generated by our tool can be used to define such inter-domain architectural power intent properties. The language used for this specification is similar to SystemVerilog assertions (SVA) [29]. For example, the above mentioned properties can be expressed using the generated predicates as shown in the following:

1) `not(MULT(ON) and DIV(ON));`
2) `not(PIU(ACTIVE) and SIU(ACTIVE) and MULT(ON))`
   `and not(PIU(ACTIVE) and SIU(IDLE) and MULT(ON))`
   `and not(PIU(IDLE) and SIU(ACTIVE) and MULT(ON))`
   `and not(PIU(IDLE) and SIU(IDLE) and MULT(ON));`
3) `not(PIU(ACTIVE) and SIU(ACTIVE) and DIV(ON))`
   `and not(PIU(ACTIVE) and SIU(IDLE) and DIV(ON))`
   `and not(PIU(IDLE) and SIU(ACTIVE) and DIV(ON))`
   `and not(PIU(IDLE) and SIU(IDLE) and DIV(ON));`
4) `MEM_CTLR(OFF, ON)` |−> ##[1:$]
   `STORAGE_ELM(OFF, ON);`
5) `STORAGE_ELM(ret-off)` |−> ##[1:$]
   `(PIU(pwr-ACTIVE) or PIU(pwr-IDLE)) or`
   `(SIU(pwr-ACTIVE) or SIU(pwr-IDLE));`

By using these high-level predicates, we mask the low-level control sequences that enable individual domains to power up or power down. In the next section, we will show how low-level control sequences corresponding to these predicates can be extracted automatically from UPF specifications of the per-domain PCC.

TABLE I
PREDICATE TABLE FOR THE *eLEON*3 DESIGN

Power Domain Names	Predicate Types	Generated Predicates		
		Power Up/Down Sequences	Power States (Multi V-F)	Intermediate State Sequences
PIU	Power Up	PIU(OFF,ACTIVE) PIU(OFF,IDLE)	PIU(ACTIVE) PIU(IDLE)	PIU-inter(ACTIVE,IDLE) PIU-inter(IDLE,ACTIVE)
	Power Down	PIU(ACTIVE,OFF) PIU(IDLE,OFF)	PIU(OFF)	
SIU	Power Up	SIU(OFF,ACTIVE) SIU(OFF,IDLE)	SIU(ACTIVE) SIU(IDLE)	SIU-inter(ACTIVE,IDLE) SIU-inter(IDLE,ACTIVE)
	Power Down	SIU(ACTIVE,OFF) SIU(IDLE,OFF)	SIU(OFF)	
CACHE	Always Power-On	CACHE(PARTIAL_ON,FULL_ON) CACHE(FULL_ON,PARTIAL_ON)	CACHE(PARTIAL_ON) CACHE(FULL_ON)	N/A

Power Domain Names	Predicate Types	Generated Predicates	
		Power Up/Down Sequences	Power States (Multi V-F)
MULT	Power Up	MULT(OFF,ON)	MULT(ON)
	Power Down	MULT(ON,OFF)	MULT(OFF)
DIV	Power Up	DIV(OFF,ON)	DIV(ON)
	Power Down	DIV(ON,OFF)	DIV(OFF)
MEM_CTLR	Power Up	MEM_CTLR(OFF,ON)	MEM_CTLR(ON)
	Power Down	MEM_CTLR(ON,OFF)	MEM_CTLR(OFF)
STORAGE_ELM	Power Up	STORAGE_ELM(OFF,ON)	STORAGE_ELM(ON)
	Power Down	STORAGE_ELM(ON,OFF)	STORAGE_ELM(OFF)

Power Domain Names	Predicate Types	Generated Predicates for Transient States		
		Power-Gated States	Isolation States	Retention States
PIU	Power Up	PIU(pwr-ACTIVE) PIU(pwr-IDLE)	PIU(iso-off)	PIU(ret-off)
	Power Down	PIU(pwr-OFF)	PIU(iso-on)	PIU(ret-on)
SIU	Power Up	SIU(pwr-ACTIVE) SIU(pwr-IDLE)	SIU(iso-off)	SIU(ret-off)
	Power Down	SIU(pwr-OFF)	SIU(iso-on)	SIU(ret-on)
CACHE	Power Up	CACHE(pwr-PARTIALON) CACHE(pwr-FULLON)	N/A	N/A
MULT	Power Up	MULT(pwr-ON)	MULT(iso-off)	MULT(ret-off)
	Power Down	MULT(pwr-OFF)	MULT(iso-on)	MULT(ret-on)
DIV	Power Up	DIV(pwr-ON)	DIV(iso-off)	DIV(ret-off)
	Power Down	DIV(pwr-OFF)	DIV(iso-on)	DIV(ret-on)
MEM_CTLR	Power-Up	MEM_CTLR(pwr-ON)	MEM_CTLR(iso-off)	N/A
	Power-Down	MEM_CTLR(pwr-OFF)	MEM_CTLR(iso-on)	
STORAGE_ELM	Power-Up	STORAGE_ELM(pwr-ON)	STORAGE_ELM(iso-off)	STORAGE_ELM(ret-off)
	Power-Down	STORAGE_ELM(pwr-OFF)	STORAGE_ELM(iso-on)	STORAGE_ELM(ret-on)

An important point to note here is that the architectural power intent properties are not clocked. This is because, the level of abstraction at which these are defined may not have definitions of clocks and timing. On the other hand, these properties can easily express the sequence in which events related to power state transitions should take place. While mapping these properties in terms of the low-level control sequences, we must introduce timing using some strategy as will be explained later.

IV. CONTROL SEQUENCE EXTRACTION FROM UPF

Fig. 7 shows the supply distribution network of the *eLeon*3 design. The Appendix demonstrates the UPF specification for this design. In this section, we demonstrate the automatic extraction of control sequences for the predicates of the previous section (see Table I) from such UPF specifications.

Our tool parses the UPF specification and automatically generates SVA sequences corresponding to the predicates. There are the following several utilities of such UPF-extracted sequences.

1) Since the predicates correspond to different power on/off, transient and intermediate states of the LPM state machine, hence, monitoring the coverage for these sequences ensures the state and transition sequence coverage of LPM.

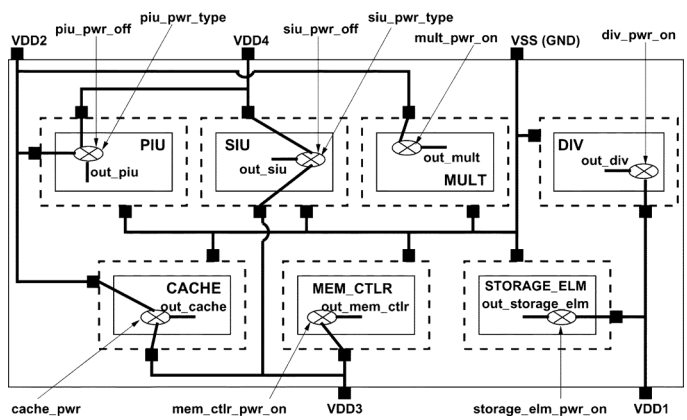

Fig. 7. Power domain and supply distribution network of *eLeon*3 design.

2) These extracted SVA sequences help in constructing global/architectural power intent properties, whose coverage can, then, be monitored either formally or during simulation to indicate the functional coverage of PCL.

In the following sub-sections, we shall discuss the extraction of different types of SVA sequences over the UPF-example provided for *eLeon*3 design (in Appendix).

A. Sequences for Power On/Off State Predicates

From the UPF specification of *eLeon*3 (given in the Appendix), we can extract two sequences for power-on states and one sequence for power-off state for the power domain, PIU. Both the two on-states of PIU are characterized using unique voltage-frequency combinations. We find that the *create_power_switch* statement for this domain has two control signals to control the voltage-rails, namely, *piu_pwr_off* and *piu_pwr_type*. Moreover, the *add_power_state* statement for this domain has one control signal to control the frequency-values, namely, *piu_high_freq*. These signals are driven by the LPM of the domain and the combination of these signals select the power on/off state of the domain.

Further, the isolation/retention signals can be easily identified from the UPF as follows. The isolation signal is *piu_iso* and to enable isolation during shut-down phase, we shall raise it to *high* value (gleaned from the *set_isolation_control* statement). For retention, the saving is done (during the shut-down phase) at the *high*-value of the *piu_ret* signal and the restoration is performed (during the power-up phase) at the *low*-value of the *piu_ret* signal.

The sequences extracted by our tool corresponding to the predicates for the power on/off states, namely, PIU(ACTIVE), PIU(IDLE), and PIU(OFF) are as follows:

```
sequence PIU_ACTIVE;
   (!piu_pwr_off && piu_pwr_type && piu_high_freq)
        && $fell(piu_ret) && $fell(piu_iso);
endsequence
sequence PIU_IDLE;
   (!piu_pwr_off && !piu_pwr_type && !piu_high_freq)
        && $fell(piu_ret) && $fell(piu_iso);
endsequence
sequence PIU_OFF;
   $rose(piu_iso) && $rose(piu_ret) && (piu_pwr_off);
endsequence
```

B. Sequences for Transient State Predicates

Basically, the sequences extracted for transient state predicates are the sub-part of the power state sequences derived in the previous section. We give one such sequence for each of the *power- gated*, *isolation* and *retention* transient states, namely, PIU(pwr-IDLE), PIU(iso-on) and PIU(ret-off) respectively, in the following.

```
sequence PIU_pwr-IDLE;
   (!piu_pwr_off && !piu_pwr_type && !piu_high_freq);
endsequence
sequence PIU_iso-on;
   $rose(piu_iso);
endsequence
sequence PIU_ret-off;
   PIU_ret-off_ACTIVE or PIU_ret-off_IDLE;
endsequence
sequence PIU_ret-off_ACTIVE;
   ((!piu_pwr_off && piu_pwr_type && piu_high_freq)
                  ##[1:$] $fell(piu_ret));
endsequence
sequence PIU_ret-off_IDLE;
   ((!piu_pwr_off && !piu_pwr_type && !piu_high_freq)
                  ##[1:$] $fell(piu_ret));
endsequence
```

The transient states corresponding to the predicates, PIU(iso-off) and PIU(ret-off), can be reached via two paths—either through *pwr-ACTIVE* state or through *pwr-IDLE* states. Hence, to model both of them, the sequence `PIU_ret-off` has an *or* construct (similar will be the case for `PIU_iso-off` sequence). The sequences for rest of the transient state predicates can be derived similarly.

C. Sequences for Intermediate State Predicates

Intermediate states are reached during the transition between two on-states, that are characterized using two different voltage-frequency combinations. Our tool generates sequences for such intermediate state transitions.

For example, in case of PIU domain in *eLeon*3, two legal intermediate state sequences can be extracted, since we can visit the intermediate state when moving from *ACTIVE* state to *IDLE* state and also from *IDLE* state to *ACTIVE* state. These sequences correspond to the predicates PIU-inter(ACTIVE, IDLE) and PIU-inter(IDLE, ACTIVE), respectively, and are given as follows:

```
sequence PIU-inter_ACTIVE_IDLE;
  (!piu_pwr_off && piu_pwr_type && piu_high_freq) ##[1:$]
  (!piu_pwr_off && piu_pwr_type && !piu_high_freq) ##[1:$]
      (!piu_pwr_off && !piu_pwr_type && !piu_high_freq);
endsequence
sequence PIU-inter_IDLE_ACTIVE;
  (!piu_pwr_off && !piu_pwr_type && !piu_high_freq) ##[1:$]
  (!piu_pwr_off && piu_pwr_type && !piu_high_freq) ##[1:$]
      (!piu_pwr_off && piu_pwr_type && piu_high_freq);
endsequence
```

If there are n on-states for a power domain, then there can be $O(n(n-1))$ transitions between them (considering both directions). Each transition may have a transient state. For example a transition from a state, $\langle V_1, F_1 \rangle$, to a state, $\langle V_2, F_2 \rangle$, (where V and F represent voltage and frequency) will typically go through a transient state $\langle V_2, F_1 \rangle$, if$V_2 > V_1$ and $F_2 > F_1$, because supply voltage must be raised strictly before raising the frequency. Typically, all on-states do not have transitions between them, and hence, lesser number of such intermediate states and sequences are required.

D. Sequences for Power Up/Down Predicates

The sequence expressions for power up/down predicates can be derived with the help of the sequences for power on/off states. Since we have two on-states *ACTIVE* and *IDLE* for PIU-domain of *eLeon*3, there can be two power-down sequences, namely, PIU(ACTIVE, OFF) and PIU(IDLE, OFF), and two power-up sequences, namely, PIU(OFF, ACTIVE) and PIU(OFF, IDLE), given as follows:

```
sequence PIU_ACTIVE_OFF;
   PIU_ACTIVE ##[1:$] $rose(piu_iso) ##[1:$]
          $rose(piu_ret) ##[1:$] (piu_pwr_off);
```

```
endsequence
sequence PIU_IDLE_OFF;
   PIU_IDLE ##[1:$] $rose(piu_iso) ##[1:$]
            $rose(piu_ret) ##[1:$] (piu_pwr_off);
endsequence
sequence PIU_OFF_ACTIVE;
   PIU_OFF ##[1:$] (!piu_pwr_off && piu_pwr_type
   && piu_high_freq) ##[1:$] $fell(piu_ret)
                     ##[1:$] $fell(piu_iso);
endsequence
sequence PIU_OFF_IDLE;
   PIU_OFF ##[1:$] (!piu_pwr_off && !piu_pwr_type
   && !piu_high_freq) ##[1:$] $fell(piu_ret)
                     ##[1:$] $fell(piu_iso);
endsequence
```

Here, PIU_ACTIVE, PIU_IDLE, and PIU_OFF are the power on/off sequences produced in Section IV-A.

E. Illegal Sequences

To identify the misbehavior in ascertaining proper control sequence outputs by the PCL, our tool also generates the SystemVerilog Assertions (SVA) for such illegal behaviors. Figs. 8 and 9 illustrate two examples of such illegal scenarios for power-down and power-up sequence, respectively.

This set of assertions are not visible in the predicate table from architectural power intent of a circuit, but are useful to locate the bug and improve the coverage of PCL. We classify these properties into the following five categories.

1) *Illegal Restoration before Power-on*. During power-up of a domain, the restoration should never take place before power-on. For the PIU-domain, the properties corresponding to such behavior are:

```
property PIU_restoreBeforePoweron_ACTIVE_illegal;
   @(posedge CLOCK)
      $fell(piu_ret) |-> ##[1:$]
      (!piu_pwr_off && piu_pwr_type && piu_high_freq);
endproperty
property PIU_restoreBeforePoweron_IDLE_illegal;
   @(posedge CLOCK)
      $fell(piu ret) |-> ##[1:$]
      (!piu_pwr_off && !piu_pwr_type && !piu_high_freq);
endproperty
```

2) *Illegal Power-off before Retention*. During shut-down of a domain, the power-off should never happen before retention. For the PIU-domain, the property corresponding to such behavior is:

```
property PIU_poweroffBeforeRetention_illegal;
      @(posedge CLOCK)
            (piu_pwr_off) |-> ##[1:$] $rose(piu_ret);
endproperty
```

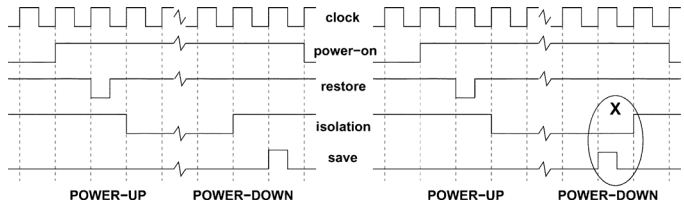

Fig. 8. Legal and illegal power-down (illegal retention before isolation).

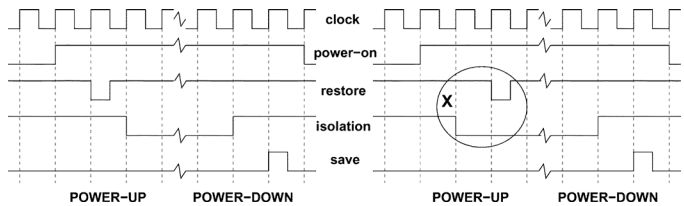

Fig. 9. Legal and illegal power-up (illegal de-isolation before restoration).

3) *Illegal Retention before Isolation*. During shut-down of a domain, the retention should never take place before isolation. For the PIU-domain, the property corresponding to such behavior is:

```
property PIU_retentionBeforeIsolation_illegal;
      @(posedge CLOCK)
            $rose(piu_ret) |-> ##[1:$] $rose(piu_iso);
endproperty
```

Fig. 8 illustrates such an illegal scenario, where retention is performed before isolation during power-down.

4) *Illegal De-isolation before Restoration*. During power-up of a domain, the isolation should never be disabled before restoration takes place. For the PIU-domain, the property corresponding to such behavior is:

```
property PIU_deisolationBeforeRestoration_illegal;
      @(posedge CLOCK)
            $fell(piu_iso) |-> ##[1:$] $fell(piu_ret);
endproperty
```

Fig. 9 illustrates such an illegal scenario, where isolation is disabled before restoration during power-up.

5) *Illegal Intermediate State Transitions*. During the transition from one on-state to another on-state, the voltage-frequency combination also changes. A particular control sequence rule is followed during such transition as given below:
   a) When transition is taking place from a high voltage-high frequency power state to a low voltage-low frequency power state, then the act of lowering frequency should precede the act of lowering voltage.
   b) When transition is taking place from a low voltage-low frequency power state to a high voltage-high frequency power state, then the act of raising voltage should precede the act of raising frequency.

   Any control sequence that violates the above rules is an illegal control sequence. For example, to check whether a given *eLeon*3 implementation violates any of the above rules, we can check for the occurrence of the following sequences.

```
sequence PIU-inter_ACTIVE_IDLE_illegal;
```

```
(!piu_pwr_off && piu_pwr_type && piu_high_freq) ##[1:$]
(!piu_pwr_off && !piu_pwr_type && piu_high_freq) ##[1:$]
(!piu_pwr_off && !piu_pwr_type && !piu_high_freq);
endsequence
sequence PIU-inter_IDLE_ACTIVE_illegal;
(!piu_pwr_off && !piu_pwr_type && !piu_high_freq) ##[1:$]
(!piu_pwr_off && !piu_pwr_type && piu_high_freq) ##[1:$]
(!piu_pwr_off && piu_pwr_type && piu_high_freq);
endsequence
```

Overall, the sequences produced in Section IV are automatically substituted in place of the predicates in the architectural power intent properties (as demonstrated in the next section). This bridges the gap between high-level global architectural power intent specification and low-level per-domain UPF specifications without additional user intervention.

V. Timing Extraction for Sequences

The properties extracted from the UPF express sequencing constraints between the output signals of the LPM. It is important to note that though the sequence in which the LPM must assert its outputs is standard in low-power design practices, the timing underlying the sequences can vary from one domain to another depending on several other power management factors.

So far in our presentation, we have allowed unrestricted stuttering between events in a sequence. For example, consider the following assertion:

```
property power-up;
      @(posedge CLOCK)
            $rose(pwr-on) |-> ##[1:$] $fell(restore)
                  ##[1:$] $fell(isolation);
endproperty
```

In this assertion, there is no bound on the time between the occurrences of the events, such as `$rose(pwr-on)`, `$fell(restore)` and `$fell(isolation)`. This may lead to bugs being missed in the verification process, as demonstrated using Fig. 10. In this figure, two power up sequences are shown. The first sequence is erroneous since the `restore` signal does not fall after the rise of the `pwr-on` signal. However, this bug is masked by the second power up sequence because the unrestricted stuttering in the assertions allows the `$fell(restore)` event of the second power up sequence to account for both `$rose(pwr-on)` events.

Timing information of such nature is not provided in the UPF specification of a power domain. However, providing safe upper bounds on the power up/power down timings is quite feasible in practice, because the number of cycles spent in these sequences are significantly smaller than the number of cycles spent between successive transitions among power states [2], [3], [14]. Therefore it is not hard for the designer to find, for each interval of the form [1:$] in a sequence, a suitable value of k such that the interval [1:$] can safely be replaced with the interval $[1:k]$. Typically a designer will choose a pessimistic value of k so as to be on the safe side.

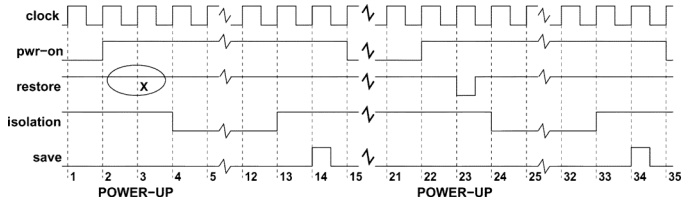

Fig. 10. Example of power up sequence error (unidentified by assertions) due to inaccurate timing information.

The performance of formal property verification is highly sensitive to the sequential depth of the property. The architectural power intent properties typically contain two or more UPF extracted sequences—the sequential depth of these sequences directly contribute to the sequential depth of the architectural property. We present experimental results to demonstrate that the quality of the bounds on the time intervals in individual sequences significantly influences the performance of formal verification on the GPM.

We then present an approach which improves the user defined bounds by automatically extracting the worst case times between successive events by formally analyzing the LPM of individual domains. Though this step comes with additional computational cost, we show that the sequential depth of the architectural assertions can be reduced considerably, leading to overall performance improvement.

We now discuss the procedure for computing worst case time intervals for the UPF-extracted per-domain sequences. Let us consider the following assertion in which we are interested in determining a tight bound on the parameter, $\mathcal{T}$.

```
property IU_powerup__idle_restore;
      @(posedge CLOCK)
            ($fell(iu_pwr_off) && !iu_pwr_type &&
            !iu_high_freq) |-> ##[1:T] $fell(iu_ret);
endproperty
```

Suppose, we are given a user-defined (pessimistic) bound, say B, for $\mathcal{T}$. Using a binary search technique between 1 and B, we can successively improve the bound B while ensuring at the end of each iteration that the LPM satisfies the property under the new bound B.

The experimental results (refer to Table IV in Section VII) demonstrate the benefits of extracting exact time delay (upper-bound) information from LPMs. In these results, binary search was used to extract the minimum admissible bounds. In practice, a user may choose to refine the bounds only to the extent that makes the architectural verification feasible.

For example, the extracted timing information for *eLeon*3 design reveals the fact that during the power-down phase for *MULT*-domain, from isolation state the LPM asserts signals for saving data within two cycles `(##[1:2])`, but from save state it takes a maximum of three cycles `(##[1:3])` to provide signals for powering the domain off. On the other hand, during the power-on phase of *MULT*-domain, the LPM takes only one cycle `(##[1:1])` before moving to restoration step; and from the restoration step it takes a maximum of three cycles `(##[1:3])` before it moves to the de-isolation step. In the next section (Section VI), we shall discuss how these sequences with accurate time boundaries can be used to generate the global assertions.

VI. Global Assertion Generation

Let us revisit the following power intent properties of *eLeon*3 design, expressing the requirements as mentioned in Section III-B. To build these global assertions for *eLeon*3, the required UPF-extracted sequences after applying accurate timing extraction procedure are given as follows:

```
sequence MULT_ON;
     mult_pwr_on && $fell(mult_ret) && $fell(mult_iso);
endsequence
sequence DIV_ON;
     div_pwr_on && $fell(div_ret) && $fell(div_iso);
endsequence
sequence PIU_ACTIVE;
     (!piu_pwr_off && piu_pwr_type && piu_high_freq)
          && $fell(piu_ret) && $fell(piu_iso);
endsequence
sequence PIU_IDLE;
     (!piu_pwr_off && !piu_pwr_type && !piu_high_freq)
          && $fell(piu_ret) && $fell(piu_iso);
endsequence
sequence SIU_ACTIVE;
     (!siu_pwr_off && siu_pwr_type && siu_high_freq)
          && $fell(siu_ret) && $fell(siu_iso);
endsequence
sequence SIU_IDLE;
     (!siu_pwr_off && !siu_pwr_type && !siu_high_freq)
          && $fell(siu_ret) && $fell(siu_iso);
endsequence
sequence MEM_CTLR_OFF_ON;
     (!mem_ctlr_pwr_on && mem_ctlr_iso) ##1
          mem_ctlr_pwr_on ##[1:2] !mem_ctlr_iso;
endsequence
sequence STORAGE_ELM_OFF_ON;
     (!storage_elm_pwr_on && storage_elm_iso &&
     storage_elm_ret) ##1 storage_elm_pwr_on
     ##[1:2] !storage_elm_ret ##[1:3] !storage_elm_iso;
endsequence
sequence STORAGE_ELM_ret-off;
     storage_elm_pwr_on ##[1:2] !storage_elm_ret;
endsequence
sequence PIU_pwr-ACTIVE;
     (!piu_pwr_off && piu_pwr_type && piu_high_freq);
endsequence
sequence PIU_pwr-IDLE;
     (!piu_pwr_off && !piu_pwr_type && !piu_high_freq);
endsequence
sequence SIU_pwr-ACTIVE;
     (!siu_pwr_off && siu_pwr_type && siu_high_freq);
endsequence
sequence SIU_pwr-IDLE;
     (!siu_pwr_off && !siu_pwr_type && !siu_high_freq);
endsequence
```

Using the SVA sequences extracted from UPF specifications, we transform the five architectural power intent properties (as given in Section III-B) of our example, *eLeon*3, into the following SVA properties.

1) For the property, *the multiplication unit (MULT-domain) and the division unit (DIV-domain) are never ON together*, the corresponding SVA is:

```
property arch_prop1;
     @(posedge CLOCK)
          not (MULT_ON and DIV_ON);
endproperty
```

2) For the property, *the integer units (PIU and SIU-domains) can not be ON together with the multiplication unit (MULT-domain)*, the corresponding SVA is:
3) For the property, *the integer units (PIU and SIU-domains) can not be ON together with the division unit (DIV-domain)*, the corresponding SVA is:

```
property arch_prop3;
     @(posedge CLOCK)
     not(PIU_ACTIVE and SIU_ACTIVE and DIV_ON)
     and not(PIU_ACTIVE and SIU_IDLE and DIV_ON)
     and not(PIU_IDLE and SIU_ACTIVE and DIV_ON)
     not(PIU_IDLE and SIU_IDLE and DIV_ON);
endproperty
```

4) For the property, *only after putting the memory controller unit (MEM_CTLR-domain) fully powered-up, the storage-elements (STORAGE_ELM-domain) are powered-up*, the corresponding SVA is:

```
property arch_prop4;
     @(posedge CLOCK)
          MEM_CTLR_OFF_ON |-> 
                ##[1:$] STORAGE_ELM_OFF_ON;
endproperty
```

5) For the property, *only after the storage-elements (STORAGE_ELM-domain) finish their restoration, the power of the integer units (either PIU and/or SIU) can be given*, the corresponding SVA is:

```
property arch_prop5;
     @(posedge CLOCK)
          STORAGE_ELM_ret-off |-> ##[1:$]
               (PIU_pwr-ACTIVE or PIU_pwr-IDLE)
               or (SIU_pwr-ACTIVE or SIU_pwr-IDLE);
endproperty
```

Here, the CLOCK-variable used to model the above properties is the system clock of the PCL. It is assumed that the name of the PCL-clock is known (or given) before we generate these global properties.

The set of generated assertions are, then, verified formally using an industrial strength formal verification tool, Magellan [21]. Verification of UPF-extracted per-domain assertions ensure the correctness of the LPMs, whereas formal verification

TABLE II
DESIGN STATISTICS FOR POWER MANAGEMENT UNITS (PCL)

Name of Design	# I/Ps	# O/Ps	# Seq. Elmts.	# Comb. Elmts.	# Nets	# Pwr. Doms.
PowerTrans	9	11	39	200	234	3
PowerCounter	8	20	47	272	332	4
Promethues	2	6	7	131	140	1
eLeon3-SingleCore	11	22	70	619	700	7
eLeon3-DualCore	20	30	94	847	960	7
eLeon3-QuadCore	32	42	147	1892	2025	15

TABLE III
UPF-SYNTHESIZED ASSERTIONS AND ARCHITECTURAL PROPERTIES

Name of Design	Number of Assertions					
	Per-Domain					Arch
	Power on/off	Transient	Intermediate	Power Up/Dn	Illegal	
PowerTrans	8	14	0	8	10	7
PowerCounter	8	24	0	8	16	8
Promethues	2	6	0	2	4	1
eLeon3-SingleCore	16	36	4	16	28	14
eLeon3-DualCore	16	36	4	16	28	22
eLeon3-QuadCore	34	62	8	28	50	40

of the global assertions ensures the correctness of the architectural power management strategy implemented by the PCL. In the next section, we provide the experimental results obtained by such formal verification procedure.

VII. EXPERIMENTAL RESULTS

In this section, we present experimental results on six case studies, namely *PowerTrans*, *PowerCounter*, *Promethues* and three low-power *Enhanced Leon3* benchmarks [22], [23], namely, *eLeon3-SingleCore*, *eLeon3-DualCore*, and *eLeon3-QuadCore*.

Table II shows the statistics *for only the PCLs of these designs*. It may be noted that the entire integrated circuit is much larger and not shown here, since we are concerned with the size of the PCLs only. The number of inputs and outputs of the PCL are shown in Columns 2 and 3, respectively. Columns 4–7 show the number of sequential elements, combinational elements, nets, and power domains, respectively, in the PCLs. In our experiments, the PCLs are given in Verilog HDL [15].

Table III shows the number of per-domain assertions extracted in Step 2) of our approach, and also the number of architectural assertions developed in Step 1). Columns 2–5 present the number of per-domain sequences extracted for four categories of predicates, namely, power on/off state predicates, transient state predicates, intermediate state transition predicates, and power up/down sequence predicates. Column 6 presents the number of sequences capturing illegal transitions. The runtime for extracting the per-domain assertions is negligible and hence not reported here. Column 7 shows the number of properties developed (manually) from the architectural power intent using the predicates generated in Step 1).

Table IV demonstrates the impact of Step 3) on the performance of Step-5. All our experiments were performed on a 2.8 GHz. Intel XEON processor with 4 GB RAM. Column 2 presents the number of architectural (global) properties used for the experiment. Column 3 denotes the time to extract the accurate upper bound using Step 3). Column 4 reports the formal verification time (building + running) of the architectural properties with the extracted bound. Column 5 is the sum

TABLE IV
FORMAL VERIFICATION WITH ACCURATE TIMING INFORMATION

Name of Design	No. of Arch. Prop.	Total Time (in sec.) Using Extracted Bound			Verification Time (in sec.) Using Pre-defined Bound		
		Extr.	Verif.	Total	#[1:4]	#[1:5]	#[1:6]
Power-Trans	7	68	95	163	228	524	1743
Power-Counter	8	94	133	227	321	735	2428
eLeon3-SingleCore	14	102	472	574	968	2631	Mem. O/F
eLeon3-DualCore	22	114	561	675	1041	2839	Mem. O/F
eLeon3-QuadCore	40	317	1044	1361	3602	Mem. O/F	Mem. O/F

TABLE V
FORMAL VERIFICATION RESULTS ON MAGELLAN

Name of Design	Number of Assertions		Formal Verification Time	
	Per-Domain	Arch	Build-Time	Run-Time
PowerTrans	40	7	172 sec.	216 sec.
PowerCounter	56	8	214 sec.	285 sec.
Promethues	14	1	27 sec.	33 sec.
eLeon3-SingleCore	100	14	673 sec.	891 sec.
eLeon3-DualCore	100	22	726 sec.	978 sec.
eLeon3-QuadCore	192	40	1656 sec.	2219 sec.

of Columns 3 and 4, and it represents the total time required for formal verification using our approach (including the timing for real-time boundary extraction). Columns 6–8 reports the run-times of formal verification without using the timing extraction step (that is, Step 3))—in these cases we used pre-defined (pessimistic) guesses on the time bounds, such as `#[1:4]` for Column 6, `#[1:5]` for Column 7, and `#[1:6]` for Column 8. Comparing these columns with Column 5 demonstrates the relevance of Step 3 in our approach. In all these experiments, the extracted time-bounds were within the range of `#[1:3]`. In spite of the fact that the guesses were only marginally higher, we had a significant impact on performance of formal verification. This is because, the inaccuracies in the sequential depth of individual control sequences add up to slightly larger sequential depths for the global assertions, resulting in exposing the known sensitivity of formal verification with sequential depth. For the larger *eLeon3* PCLs, formal verification failed to scale (abbreviated as *"Mem. O/F"* in Table IV) in the absence of Step 3).

Table V presents the formal verification results (obtained in Step 5)) including their building and running times using Magellan [21] as the formal verification tool. It shows that with accurate extracted time bounds for per-domain sequences, the industrial formal verification tools are capable of formally verifying the global power management logic. Columns 2 and 3 show the total number of extracted per-domain assertions by our tool and the number of generated global assertions, respectively. The time taken (in seconds) for building and running the design are presented in Columns 4 and 5, respectively. This makes PCLs very good candidates for formal verification as demonstrated by the run-times in Table V.

It is anticipated that future SOCs with more than 100 cores may have many more power domains and power states, but we believe that PCLs will still remain within feasible limits for verifying by our approach.

VIII. Conclusion

The verification community has been working on automatic extraction of per-domain assertions from UPF specifications in recent times. We show that automatically extracted sequences from per-domain UPF specifications can be used to automatically translate inter-domain global power intent properties into SystemVerilog assertions in terms of low-level per-domain control signals—thereby bridging the disconnect between architectural power intent properties which are expressed in terms of the major power states of multiple power domains and the low-level control sequences which model transitions between local power states. We further extended our approach to extract accurate real-time bounds on the per-domain sequences before they can be used within the architectural assertions. We integrated the whole methodology into our tool (`POWER-TRUCTOR`) and produced relevant results over some industrial test-cases. Our methodology helps the verification engineer not only to find deep low-power functional bugs in the design but also to shorten the time to find bugs, thus increasing verification productivity. Using high-level models to express properties at both the per-domain and the inter-domain level, the user can automate the assertion generation capabilities which could now be proved formally.

Appendix
UPF Specification for Enhanced Leon3 (*eLeon*3)

A UPF specification for a design consists of the definitions of the power domains, the supply networks and logic specification for isolation, retention and level shifting strategies. We describe the UPF specification for the PIU-domain of *eLeon*3. The UPF for the SIU, MULT, DIV, MEM_CTLR, CACHE, and STORAGE_ELM domains are defined similarly.

Creating Power Domains

A *power domain* is a collection of design elements that share a primary supply set. Fig. 7 shows the *seven* power domains in *eLeon*3, namely, *PIU*, *SIU*, *MULT*, *DIV*, *MEM_CTLR*, *CACHE*, and *STORAGE_ELM*. We declare these power domains in UPF 2.0 as:

```
create_power_domain TOP -include_scope -scope /top_eleon3
create_power_domain PIU -include_scope
   -elements { /top_eleon3/primary_IU
        /top_eleon3/superscalar_controller }
create_power_domain SIU -include_scope
   -elements { /top_eleon3/secondary_IU }
create_power_domain MULT -include_scope
   -elements { /top_eleon3/multiplication_unit }
create_power_domain DIV -include_scope
   -elements { /top_eleon3/division_unit }
create_power_domain MEM_CTLR -include_scope
   -elements { /topeleon3/cache_controller
        /top_eleon3/memory_controller
        /top_eleon3/memory_ahb_interface }
create_power_domain CACHE -include_scope
   -elements { /top_eleon3/iCache /top_eleon3/dCache }
create_power_domain STORAGE_ELM -include_scope
   -elements { /top_eleon3/memory
        /top_eleon3/register_filebank }
```

Creating Supply Network

The UPF supply network creation commands designate the power supply network that connects power supplies to the design elements in a design. The supply network is a set of supply nets, supply ports, switches, and potentially, voltage regulators and generators. We express the power supply network (nets, ports and power switches) for the domain, PIU, of *eLeon*3 in UPF 2.0 as follows:

```
## Supply Network and Supply Port Creation ##
create_supply_port VDD_high_port -direction in
create_supply_port VDD_low_port -direction in
create_supply_port VSS_gnd_port -direction in
create_supply_net VDD_high -domain PIU
create_supply_net VDD_low -domain PIU
create_supply_net VSS_gnd -domain PIU
create_supply_net VDD_piu -domain PIU
connect_supply_net VDD_high -ports { VDD_high_port }
connect_supply_net VDD_low -ports { VDD_low_port }
connect_supply_net VSS_gnd -ports { VSS_gnd_port }
set_domain_supply_net PIU
   -primary_power_net VDD_piu
   -primary_ground_net VSS_gnd
## Power Switch Creation ##
create_power_switch SW_piu -domain PIU
   -output_supply_port { out_piu VDD_piu }
   -input_supply_port { inhigh_piu VDD_high }
   -input_supply_port { inlow_piu VDD_low }
   -control_port {piu_cp1 /pmu/pmu_IUprimary/piu_pwr_off}
   -control_port {piu_cp2 /pmu/pmu_IUprimary/piu_pwr_type}
   -on_state {ACTIVE_piu inhigh_piu {!piu_cp1 && piu_cp2}}
   -on_partial_state { IDLE_piu inlow_piu
                        {!piu_cp1 && !piu_cp2} }
   -off_state { OFF_piu {piu_cp1} }
## Supply Port State (Voltage Values) Assignment ##
add_port_state VDD_high_port
   -state { VDD_high_on 1.0 }
   -state { VDD_high_off off }
add_port_state VDD_low_port
   -state { VDD_low_on 0.6 }
   -state { VDD_low_off off }
add_port_state VSS_gnd_port
   -state { VSS_gnd_on 0.0 }
   -state {VSS_gnd_off off}
add_port_state SW_piu/out_piu
   -state { out_piu_active 1.0 }
   -state { out_piu_idle 0.6 }
   -state { out_piu_off off }
```

Creating Multiple Voltage-Frequency Combinations

UPF can specify the power states for every power domain, through which multiple voltage and frequency attributes of a power domain can be suitably expressed. We express the power states for the domain, PIU, of *eLeon*3 in UPF 2.0 as follows:

```
add_power_state -domain PIU
   -state { ACTIVE_piu_MODE
         -logic_eq { SW_piu && piu_high_freq &&
               interval(clk posedge negedge)>=100ns }
         -supply_eq { {power == `{FULL_ON, 1.0}} &&
                     {ground == `{FULL_ON, 0}} }
         -simstate NORMAL }
         -state { IDLE_piu_MODE
         -logic_eq { SW_piu && !piu_high_freq &&
               interval(clk posedge negedge)>=200ns }
               -supply_eq { {power == `{FULL_ON, 0.6}}
                && {ground == `{FULL_ON, 0}} }
               -simstate NORMAL }
   -state { OFF_piu_MODE
         -logic_eq { !SW_piu }
         -supply_eq { {power == `{OFF}} }
         -simstate CORRUPT }
   -legal
```

Extending Logic Specification

UPF defines extensions of the logic design with power-specific capabilities and constraints without modifying the original logic specification. Usually, *Isolation*, *Retention*, and *Level-Shifter* are the basic power specifications which will result in extended logic circuitry, while UPF gets synthesized into gate-level power circuitry or PCC. The following extends the UPF specification of *eLeon*3 by introducing isolation and retention strategies for PIU power-domain:

```
## Isolation Definition ##
set_isolation ISO_piu -domain PIU
       -isolation_power_net VDD_high
       -isolation_ground_net VSS_gnd
       -clamp_value { 0 }
       -applies_to outputs
set_isolation_control ISO_piu -domain PIU
       -isolation_signal /pmu/pmu_IUprimary/piu_iso
       -isolation_sense high
       -location self
## Retention Definition ##
set_retention RET_piu -domain PIU
       -retention_power_net VDD_high
       -retention_ground_net VSS_gnd
       -elements { ASR20 IRQ }
set_retention_control RET_piu -domain PIU
      -save_signal { /pmu/pmu_IUprimary/piu_ret high }
      -restore_signal { /pmu/pmu_IUprimary/piu_ret low }
```

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments and constructive suggestions which have enriched this paper significantly and the editor for handling this paper.

REFERENCES

[1] ACPI, "Advanced Configuration and Power Interface," [Online]. Available: www.acpi.info
[2] K. Agarwal, K. Nowka, H. Deogun, and D. Sylvester, "Power gating with multiple sleep modes," in *Proc. 7th Int. Symp. Quality Electron. Design (ISQED)*, 2006, pp. 633–637.
[3] N. Agarwal and N. J. Dimopoulos, "Automated power gating of registers using CoDeL and FSM branch prediction," in *Proc. 7th Int. Conf. Embed. Comput. Syst.: Arch., Model., Simulation (SAMOS)*, 2007, pp. 294–303.
[4] S. Bailey, G. Chidolue, and A. Crone, "Low power design and verification techniques," Mentor Graphics, White Paper, 2007. [Online]. Available: http://low-powerdesign.com/Low_Power_WP_9-13-07.pdf
[5] S. Bailey, A. Srivastava, M. Gorrie, and R. Mukherjee, "To retain or not to retain: How do I verify the state elements of my low power design?," in *Proc. DVCon*, 2008, pp. 11–17.
[6] A. Bellaouar and M. I. Elmasry, *Low-Power Digital VLSI Design*. Norwell, MA: Kluwer, 1995.
[7] F. Bembaron, S. Kakkar, R. Mukherjee, and A. Srivastava, "Low power verification methodology using UPF," in *Proc. DVCon*, 2009, pp. 228–233.
[8] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM J. R&D*, vol. 47, no. 5/6, pp. 653–670, 2003.
[9] A. R. Chandrakasan and R. W. Brodersen, *Low-Power Digital CMOS Design*. Norwell, MA: Kluwer, 1995.
[10] G. Chidolue and B. Ramanandin, "Upping verification productivity of low power designs," in *Proc. DVCon*, 2008, pp. 3–10.
[11] A. Crone and G. Chidolue, "Functional verification of low power designs at RTL," in *Workshop for Power Tim. Model., Opt. Simulation (PATMOS), LNCS-4644*, 2007, pp. 288–299.
[12] D. Flynn, "Design for power gating—and what UPF can, and cannot, do for you!," presented at the SNUG, San Jose, CA, 2009.
[13] A. Hazra, S. Mitra, P. Dasgupta, A. Pal, D. Bagchi, and K. Guha, "Leveraging UPF-extracted assertions for modeling and verification of architectural power intent," in *Proc. 47th Design Autom. Conf. (DAC)*, 2010, pp. 773–776.
[14] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2004, pp. 32–37.
[15] *IEEE VHDL Standard*, IEEE 1364-2005 Standard Verilog Hardware Description Language, 2006 [Online]. Available: http://ieeexplore.ieee.org/xpls/abs all.jsp?arnumber=1620780
[16] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarch. (MICRO)*, 2006, pp. 347–358.
[17] S. Jadcherla, J. Bergeron, Y. Inoue, and D. Flynn, "Verification methodology manual for low power (VMM-LP)," 2009.
[18] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual (LPMM)—For System-on-Chip Design*, 2nd ed. New York: Springer, 2008.
[19] N. Khan and W. Winkeler, "Power assertions and coverage for improving quality of low power verification and closure of power intent," in *Proc. DVCon*, 2008, pp. 53–58.
[20] A. Lungu, P. Bose, D. J. Sorin, S. German, and G. Janssen, "Multicore power management: Ensuring robustness via early-stage formal verification," in *Proc. 7th IEEE/ACM Int. Conf. Formal Methods Models for Co-Design (MEMOCODE)*, 2009, pp. 78–87.
[21] Magellan, "An Industrial Formal Verification Tool From Synopsys," [Online]. Available: www.synopsys.com/tools/verification/functionalverification/pages/magellan.aspx
[22] K. Marcinek, A. W. Luczyk, and W. A. Pleskacz, "Enhanced LEON3 core for superscalar processing," in *Proc. 12th Int. Symp. Design Diagnostics Electron. Circuits Syst. (DDECS)*, 2009, pp. 238–241.

[23] K. Marcinek, A. W. Luczyk, and W. A. Pleskacz, "Enhanced LEON3 low power IP core for DSM technologies," in *Proc. 16th Int. Conf. Mixed Design Integr. Circuits Syst. (MIXDES)*, 2009, pp. 262–265.
[24] R. Mukherjee, A. Srivastava, and S. Bailey, "Static and formal verification of power aware designs at the RTL using UPF," in *Proc. DVCon*, 2008, pp. 42–47.
[25] Power Forward, "A Practical Guide for Low Power Design—An Experience With CPF," 2008. [Online]. Available: http://www.powerforward.org/
[26] K. Roy and S. C. Prasad, *Low-Power CMOS VLSI Circuit Design*. Singapore: Wiley-Interscience, 2000.
[27] K. Snyder, C. Deaton, and D. Smith, "Formal Verification Checks IC Power Reduction Features," Sep. 3, 2008. [Online]. Available: http://www.scdsource.com/article.php?id=309
[28] Spyglass-Power, "Early Power Estimation Tool From Atrenta," [Online]. Available: http://www.atrenta.com/solutions/spyglass-family/spyglass power.htm
[29] SystemVerilog LRM, "SystemVerilog LRM 3.1a by Accellera," 2004. [Online]. Available: http://www.systemverilog.org
[30] Turandot, "A Power-Performance Analysis Simulator From IBM," [Online]. Available: http://www.research.ibm.com/MET/Toolset/Turandot/turandot.html
[31] *UPF-2.0. Unified Power Format 2.0 Standard [Draft Version]—IEEE Draft Standard for Design and Verification of Low Power Integrated Circuits*, IEEE P1801/D18, Oct. 23, 2008.

Aritra Hazra (S'08) received the B.E. degree from the Department of Computer Science and Engineering, Jadavpur University, Kolkata, India, in 2006 and the M.S. degree from the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India, in 2010, where he is currently pursuing the Ph.D. degree from the Department of Computer Science and Engineering.

His research interests include verification of VLSI designs, power intent verification, and functional reliability analysis. He has published several research papers in various international conferences.

Mr. Hazra was a recipient of a Best Student Paper in VLSI Design Conference (2010). He has also been awarded with the Microsoft Research (India) Fellowship in 2011.

Sahil Goyal received the B.Tech. degree (with honors) in computer science and engineering from Indian Institute of Technology, Kharagpur, India, in 2010.

He is an Analyst with Global Technology, Barclays Capital, Singapore. His research interests include formal verification, data mining, and artificial intelligence. Previously he has worked with Minekey, India.

Pallab Dasgupta (SM'99) received the B.Tech., M.Tech., and Ph.D. degrees in computer science from Indian Institute of Technology, Kharagpur (IIT Kharagpur), India.

He is currently a Professor with the Department of Computer Science and Engineering, IIT Kharagpur. His research interests include Formal verification, artificial intelligence, and VLSI. He has over 100 research papers and two books in these areas. He currently leads the Formal Verification Group, Department of Computer Science and Engineering, IIT Kharagpur (www.facweb.iitkgp.ernet.in/~pallab/forverif.html). He is also the Co-Director of Synopsys CAD Lab, IIT Kharagpur.

Ajit Pal (SM'92) received the M.Tech. and Ph.D. degrees from the Institute of Radio Physics and Electronics, Calcutta University, West Bengal, India, in 1971 and 1976, respectively.

He is presently a Professor with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India. Before joining IIT Kharagpur in 1982, he was with the Indian Statistical Institute, Calcutta, ITI, Naini, and DLRL, Hyderabad, in various capacities. His research interests include real time systems, CAD for VLSI, and computer networks. He has over 90 publications in reputed journals and conference proceedings and a book entitled *Microprocessors: Principles and Applications* (TMH, 1990).

Prof. Pal is the Fellow of the IETE, India.