Progyna Khondkar

Low-Power Design and Power-Aware Verification

Springer

Low-Power Design and Power-Aware Verification

Progyna Khondkar

Low-Power Design and Power-Aware Verification

Springer

Progyna Khondkar
Design Verification Specialist
Mentor Graphics - A Siemens Business
Fremont, CA, USA

ISBN 978-3-319-66618-1 ISBN 978-3-319-66619-8 (eBook)
DOI 10.1007/978-3-319-66619-8

Library of Congress Control Number: 2017951845

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

“The existence of God is attested by everything that appeals to our imagination. And if our eye cannot reach Him it is because He has not permitted our intelligence to go so far.” Napoleon Bonaparte.

This Book is Dedicated to My and all Families of the World-

Preface

Low-power (LP) design, power-aware (PA) verification, and Unified Power Format (UPF) or IEEE-1801 power standards are no longer special features. These technologies and methodologies are now part of standard design, verification, and implementation flows (DVIFs). Almost every chip design today incorporates some kind of low-power technique through power management on chip–by dividing the design into different voltage areas and controlling the voltages, through UPF and testbench. Then, the PA dynamic and PA static verification or their combination comes into the play.

The entire LP design and PA verification process involves thousands of techniques, tools, and methodologies, employed from the register transfer level (RTL) of design abstraction down to the synthesis or place-and-route levels of physical design. These techniques, tools, and methodologies are evolving everyday through the progression of design-verification complexity and more intelligent ways of handling that complexity by engineers, researchers, and corporate engineering policy makers.

However, the industry is missing a complete knowledge base to fully comprehend LP design and PA verification techniques and methodologies. And deploy them all together in a real design verification and implementation projects. This book, "The Concepts and Fundamentals of Power Aware Verification", is the first approach to establishing a comprehensive PA knowledge base.

Writing an engineering reference book is never an easy task, especially when the perspective is wide, encompassing academic thought and the pragmatic objective of making a reference engineering handbook. The entire effort seeks to bring the complex LP design and PA verification process to a complete and one-stop platform – one where engineers will find real examples and results, corporate engineering policy makers will discover ideas for appropriate methodologies to adopt for their next design-verification project, researchers will find topics for new and innovative ideas, and EDA verification experts will be able to enhance and fine tune their tools for the best industry practices.

This book is written in a ground up manner with the objective to appeal to a wide range of readers – from beginners to experts in the low power design and power

aware verification world. However, it is highly recommended that the reader have a solid grasp of HDL (Verilog, SystemVerilog, and VHDL etc.) fundamentals before jumping into this book. This is because UPF directly inherit ports, nets, elements, instances, modules, interfaces, boundaries, hierarchical constructs, scopes, and more from HDL. And obviously, the PA verification artifacts – tools and techniques are founded on the combination of UPF and HDL constructs.

San Jose, CA, USA Progyna Khondkar

Acknowledgement

I would like to express my sincere gratitude to Ping Yeung, Jean Chapman and Jan Johnson for providing me the opportunity to write this book. I would also like to thank and acknowledge the valuable feedback provided by my colleagues, Harry Foster, Madhur Bhargava, Gabriel Chidolue, Durgesh Prasad, Vinay Singh, Pankaj Gairola, Joe Hupcey, Koster Rick, Todd Burkholder, Rebecca Granquist, Hany Amr, Mark Handover, Abhishek Ranjan, Allan Gordon, Chuck Seeley, Barry Pangrle, Tom Fitzpatrick, and many others to name. I also would like to specially thank Ferro Melissa, and Roxanne Carpenter for their sincere support. Above all, I am very grateful to my family, my wife and my son, for their encouragements and consistent support during my writings of this book.

San Jose, CA, USA Progyna Khondkar

Contents

Author Bios

Progyna Khondkar is a low power design and verification expert and senior verification engineer at Mentor Graphics in the design verification technology division (DVT). He holds two patents and has numerous publications in power aware verification. He has strong focus on electronics, computer and information science education, research and teaching experiences in top level universities in Asia. He has worked for Hardware-Software design, development, integration, test and verification in the world class ASIC & Electronic Design Automation (EDA) companies for the last 15 years. He holds a PhD in Computer Science and is a senior member of IEEE. He also serves as a member of editorial board and reviewer of Journal of INFORMATION, IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems, IEEE Transactions on Computers and Journal of VLSI Design and Verification (JVLSIDV).

Chapter 1
Introduction

This is the age of handheld devices and it is a demand of the standard design verification flow that every chip design today needs to be low-power (LP).

The question is how?

The enormous growth of battery operated handheld and IoT devices with unlimited features compel the integrated circuits and chips to operate at very low power and consume only a few 100 mW to 1000 mW of power. Even for the plugged in devices, the California Energy Commission regulated that battery-charged appliances must comply with 2013 energy efficiency standards.

With the progression of semiconductor process technologies towards 65 nm and below, the leakage power dissipation, (the alternate term "static power" is not used intentionally throughout this book, interchangeably with leakage power) is the power dissipated by a CMOS transistor when it is not actively switching. This poses another level of verification complexity along with the ever existing dynamic power.

As a precaution, design engineers started controlling voltage on the chip or partitioning the design into different voltage areas or domains to ensure minimum power dissipation or to allow the end product to operate in possible low power consumption states. The Unified Power Format (UPF, also known as IEEE-1801 Low Power Methodology Standard) is a low power specification of the real power-intent of design. The UPF makes it possible to manipulate power dissipations by controlling voltage. This voltage or power aware (PA) technique, however, directly challenges the conventional verification tools, methodologies, techniques, and flows. A reinvention in the verification tools and methodologies for low power revolutionized the entire segment of the design verification and implementation flow (DVIF) and simplified the challenges of LP designs.

The "The Concepts and Fundamentals of Power Aware Verification" provides a consolidated look at the vast PA techniques, tools, and methodologies. The concepts and fundamentals of this book are founded upon experiences from numerous successful design and verification projects, incorporating different PA techniques and perceptions gained from participation and contribution to the IEEE-1801 standardization committee.

P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8_1

The contents and contexts of this PA verification book are organized for the vast VLSI design and verification community to make it possible to simply start from scratch and develop expertise in the low power domain. The concepts and fundamentals of power aware verification conceived here with the vision to enable power aware verification at a very early design cycle in the DVIF and gradually encompass the entire flow. Hence the design and verification community can benefit from the collective experience captured in the examples and illustrations as the industry strives towards making every chip design power aware today.

Chapter 2
Background

It is now time for the chip design and verification community to take a closer look at the semiconductor physics fundamentals, since Moore's law still holds true after a half century. Gordon Moore forecast in 1965 that transistor integration in chips will double every 2 years. One of the prime reasons the law still holds true, defying chip area congestions, is because of the drastic advancement of the integrated circuits fabrication and process technology. TSMC foundry is in fabrication process of 10 and 7 nm technology from early 2017.

A higher process node is definitely attractive as more functionality integration is possible in a smaller die area at a lower cost. Typically, technology scaling contributes to reducing gate delay, which in turn further contributes to increase frequency, increase transistor density, and reduce energy per transition.

However, in reality, this comes at the cost of exponentially increasing leakage power. Because the minimum gate-to-source voltage differential that is needed in CMOS devices to create a conducting path between the source and the drain terminals, which is known as threshold voltage, has been pushed to its limit. The reason is to keep pace with smaller transistor and interconnect sizes and increasing operating frequency. Though there are promising process integration technologies that are rolling out because of advancements in silicon-on-insulator (SOI) deployment on multigates like FinFET, FlexFET, and Tri-Gate-Transistors. These provide better electrical control over the conduction channel of CMOS devices and help reducing the leakage current as well as overcoming other short-channel effects. However, there are integration challenges with multigate devices in regular semiconductor manufacturing processes, which are subject to extensive research.

The fundamental semiconductor physics for threshold voltage that causes a new phenomenal shift to higher leakage power dissipation from the predominating dynamic power issues, are mostly due to the advancement in process nodes. In other words, the leakage power is a function of the threshold voltage and at smaller device geometries its contribution to the total energy dissipation becomes significant. The total energy dissipation in a CMOS circuit can be expressed as the summation of leakage and dynamic power integration over time t as shown below.

P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8_2

$$E = \int_0^t \left(CV_{dd}^2 f_c + V_{dd} I_{leak} \right) dt.$$

Here the leakage power is directly contributed from device supply voltage and leakage current;

$$Leakage\ \ Power = V_{dd} I_{leak}.$$

While dynamic power is contributed from the switching activity of the capacitive load on supply voltage and its switching frequency;

$$Dynamic\ Power = \alpha CV_{dd}^2 f_c, \text{ where } \alpha \text{ is activity factor}.$$

It is evident from these equations that minimizing dynamic power requires minimizing switching capacitance and switching activity; whereas minimizing leakage power requires increasing threshold voltage and improving process technology.

But again it may cause higher leakage power dissipations unless new inventions, like multigate devices, are available in advanced process nodes.

However, reducing supply voltage V_{dd} provides controllability for both leakage and dynamic power dissipation. This became a major power management key in the last few years as voltage is a design parameter where the chip design and verification community has more controllability compared to the integrated circuit fabrication process technology that relies more on foundries.

Several power dissipation reduction techniques have been adopted over the past few decades for addressing both the leakage and the dynamic power, namely clock-gating, multi-threshold, body-bias, and transistor-sizing. Despite of their wide deployment for reducing dynamic and leakage power, each has its own limitations and complexity. Since power or supply voltage is the key today, more advanced power management and reduction techniques have been adopted in the last few years based on reducing or controlling the supply power. The mainstream techniques adopted today, as shown below, are mostly based on design type and complexity demands for system-on-chip (SoC), ASIC, microcontroller units (MCU), or processor core design implementations.

Example 2.1 Sample List of Mainstream Power Dissipation Reduction Techniques

- Power Gating or Power-Shut Down
- Power Gating with State/Data Retention
- Low power Standby with State/Data Retention
- Multi-Voltage Design with Performance on Demand
- Dynamic Voltage (and Frequency) Scaling
- Adaptive Voltage (and Frequency) Scaling

Although the names of these techniques suffice to understand their operations and objectives in any of the above-mentioned design implementations, however,

adoptions and verification of these techniques were impossible until the UPF or similar power formats, like CPF,[1] became available from 2007 and onward. Although CPF is not an IEEE standard itself, many of the CPF semantics were contributed to the IEEE-1801 committee to help standardize the UPF.

2.1 The Power Intent

The mainstream power management and reduction techniques listed in the previous section are solely based on direct manipulation of voltage, in terms of supply power connectivity and voltage area or power network distributions on the chip. Nevertheless, none of these were sufficient to understand and reflect the power aware verification plan or power intent to start power or voltage aware verification at the register transfer level (RTL) or even after gate-level synthesis.

Usually, RTL in Verilog, VHDL, or SystemVerilog (HDL in general), are the golden reference design, and it is totally unconventional to add supply networks and their corresponding connectivity to these references. Eventually, it is also impossible to distribute certain instances of a design hierarchy to specific voltage areas unless the design reaches the floor-planning stage of design implementation. Evidently the industry was facing the absence of a methodology that would help to define supply power connectivity, voltage areas, and power network distributions in a design, without intervening with the HDL reference design and possibly starts from RTL of the design abstraction levels. Thus adoption of any of the new or mainstream power controllable design implementation techniques through a power specification was just impossible at any design abstraction level, from RTL to place-and-route (P&R). As well, verification on power network connectivity, voltage area distribution, etc. was also impossible unless post P&R power-ground (PG) connected netlist are available at a much later part of the design implementation cycle.

2.2 The Abstraction of UPF

In early 2007, the Accellera Systems Initiative introduced the Unified Power Format (UPF), also known as UPF 1.0 that allows users to define and manage power for designs without any direct interference with the golden HDL reference design. The concept of UPF inherits, from deploying the power intent of any design directly overlying the HDL references. The overlay needs to be done without direct intervention, through a methodological approach to abstractly model the exact power supply networks, voltage, or power area distributions and corresponding power states. The methodologies are based on power specification or intent at the RTL and allow

[1] CPF: Common Power Format Specification is an initiative from Si2 (Silicon Integration Initiatives) organization.

designers to continuously refine the power network distributions of voltage areas on the power domain boundaries throughout the entire design implementation flow, specifically at the post synthesis and post P&R phases.

In general, UPF provides the notion of power management of the entire design from the design implementation and verification perspective. Hence power aware (PA) design verification and implementation automation tools are also built on UPF semantics and language references.

The early UPF 1.0 initiatives were mostly driven from the physical design perspective in terms of explicit power supply networks, supply ports, supply nets, and their power states. These physical entities are mostly absent at the RTL or higher levels of design abstraction, unless the design has already been rolled out for synthesis and P&R. This contradicts as well as delays the original power management and verification objectives that possibly require to begin at least from the RTL.

In 2009, the IEEE Standard Organization published the IEEE-1801-2009 or UPF 2.0, the first standard methodology for implementing power intent, truly for any design abstraction level. The UPF 2.0 first provides the notion of supply sets, an abstract collection of power supply nets combining power, ground, and bias functionality that represents a potential supply net connection to a corresponding portion of a design. Supply sets can be incrementally refined and extended for different functionality, depending on design abstraction level starting from RTL.

Apart from unleashing supply sets, the IEEE standard itself evolved overtime, providing additional abstraction flexibility for the other rudimentary parts of design elements, like supply power, supply networks, and supply states for design groups, models, and instances, collectively known as objects. Some of the amendments are reflected in IEEE Standard 1801-2013 or UPF 2.1 published in May 2013, while the latest update is available in IEEE Standard 1801-2015 or UPF 3.0 published in December 2015.

However, it is important to note that each amendment is not necessarily backward compatible and each updated version is usually a superset of semantic and syntactic expressions from its predecessors.

The abstraction of UPF 3.0 or UPF in general, methodically consists of power specifications that categorically itemize the design elements for a particular power block, known as a power domain (PD), along with the PD boundaries. Further, it also specifies the power supply and supply states of the power domains or supply sets, whether the supply is in On, Off, or other potential states. Depending on the power management and reduction techniques the specification adopts for the final chip, the list further extends to specify the requirements of boundary strategies for intra or inter-domain communications. These strategies may include isolations (ISO), level-shifters (LS), enable level-shifter (ELS), always-on buffers (AOB), feed through buffers or repeaters (RPT), diode clamps, retention flops (RFF), power switches (PSW), and their corresponding supply network and locations details. In short, the UPF is a precise map that models the power specification of the design and converts the same to a power aware design.

To note, the strategies in UPF actually refer to special power management multi-voltage (MV), or power aware (PA) cells, and they are usually physically inserted into the design in post-synthesis or post-layout.

Figure 2.1 shows the HDL reference based design block diagram. Among several design blocks or instances only part of the design is distributed in four different power domains just overlaid on the blocks without intervention in the instance itself.

Figure 2.2 shows a typical UPF layout for the specific design of Fig. 2.1, where the PD_top represents the default UPF top power domain which contains the cpu_top

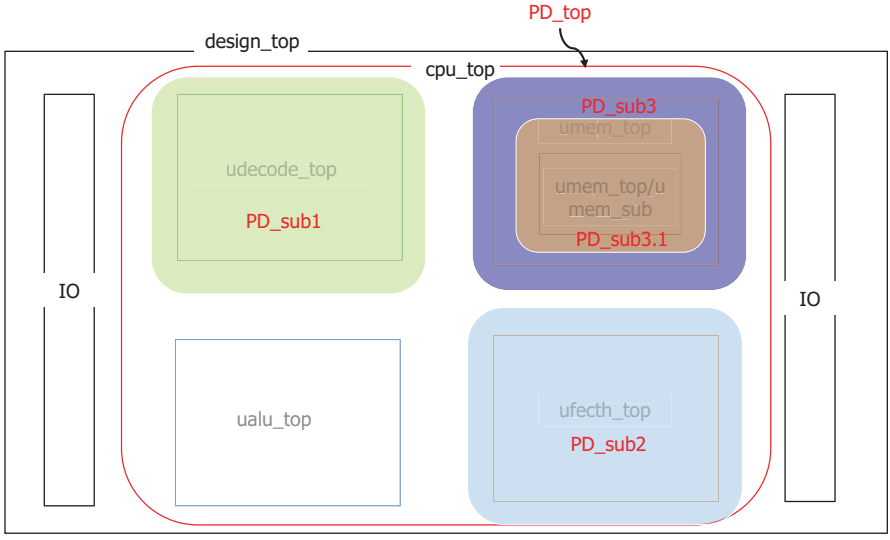

Fig. 2.1 Design block diagram with only a portion of the design overlaid by power domains with specific design instances

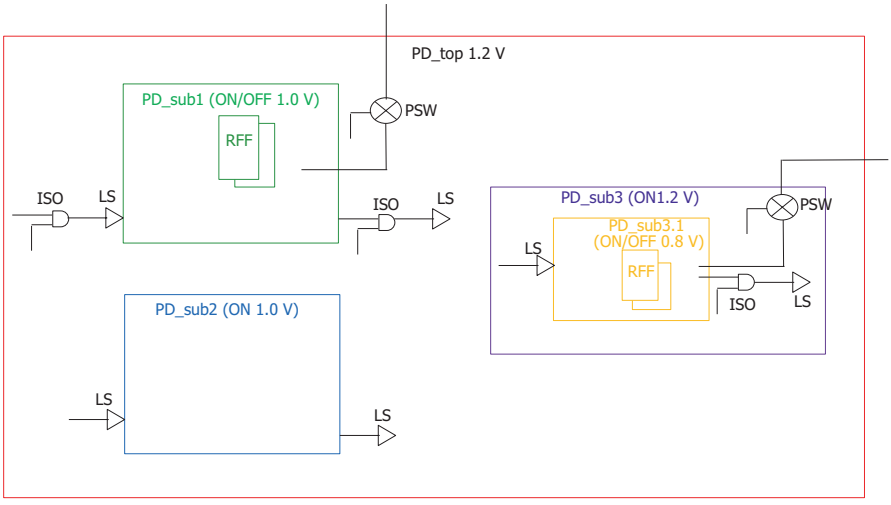

Fig. 2.2 Fundamental building block of UPF power domains, domain boundary, power network, and relevant strategies

design blocks as element. The PD_sub1, PD_sub2, and PD_sub3 power domains and PD_sub3.1 (the sub domain of PD_sub3) represent specific hierarchical design instances as their elements like udecode_top, ufetch_top, umem_top, and umem_top/umem_sub respectively. Hence the instance ualu_top naturally goes to the default PD_top power domain. The power domains also illustrate their respective ON or ON-OFF status and UPF strategies like PSW, ISO, LS, ELS (combination of ISO and LS), and RFF etc. are also shown with symbolic representation accordingly.

Epilogue: The **Chap. 1 Introduction** and **Chap. 2 Background** are the foundations of this book and are self-explanatory prologues. While introduction is purely intended to provide a brief overview of low power design and power aware verification concepts, this chapter, elaborated few fundamental aspects for power aware verification for the readers to comprehend and carry all over the book. The first crucial aspect is the list of **Mainstream Power Dissipation Reduction Techniques**. The second aspect is – power intent and how the manipulation of voltage on a design is mapped and controlled through the power intent "UPF". The third is the abstraction of UPF – very basic concepts of power domains on a design and its surroundings.

Prologue: The next chapter, **Modeling UPF** is the entrance to the power aware verification world. This chapter is dedicated to provide readers the understanding of fundamental construct of UPF. The explanation starts with the detail elaboration of power domain and domain boundary as the concepts are directly inherited from HDL (Verilog, SystemVerilog and VHDL) perspective. The fundamental constructs further explains the power supply networks, power states and power strategies in light of both the UPF Language Reference Manual (LRM) and practical usage perspective. Consequently the methodical approach for UPF construct level incremental refinements and flow level successive refinements and hierarchical UPF are also explained with realistic examples.

Chapter 3
Modeling UPF

As explained in the previous Chapter 2, the modeling of UPF comprises the precise mapping of power specifications of a design, hence the development of such a specification usually starts with the prerequisite design targets, whether it is SoC, ASIC, MCU, or processor core. It is also required to determine the best power management and reduction techniques applicable for the design. The obvious next objective is to capture every detail of the adopted techniques in parameterized attributes for both HDL references and user-defined variables in order to construct UPF files.

The details usually specify the names and number of power domains, their constituent elements in terms of HDL instances, power distribution network for the system, and corresponding power states as minimum requirements. The list further grows for different strategies depending on multiple power supply states or power domain On-Off relationship, multi-voltage, or low power standby considerations. The strategies can be extended from power switching (PSW) networks to isolation (ISO), level shifter (LS) – including enable level shifter (ELS), repeater (RPT), and retention flops (RFF) strategies.

The power specification also clarifies whether power On-Off is conducted on or off chip, requires header or footer switches, requires elements for ISO or LS with actual hierarchical paths, lists register files for RFF with possible synthesis name change conventions, and includes controls for all of these strategies. Hence, modeling of UPF is mostly governed by target design objectives and applicable power management and reduction techniques.

3.1 Fundamental Constructs of UPF

UPF is the power management methodology that facilitates the adoption of different power dissipation reduction techniques and allows to formalize the modeling and mapping of the power specification onto a design. The fundamental constituent parts for UPF constructions are broadly based on the following categories.

P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8_3

List 3.1 Fundamental Constituent Parts for UPF Constructions
- Design scopes for the UPF
- Power Domains
- Power Domain Interfaces and Power Domain Boundaries
- Power Supply and Power Supply Networks
- Primary Power and Primary Ground
- Power States and Modes of Power Operations
- Power Strategies

The rudimentary constituent parts of UPF as listed in List 3.1 are further augmented with design or HDL construct parameters. Specifically, the design scopes may include hierarchical top design module or instance name and power domain confined hierarchical instance paths to define the boundary of the domain. The power strategies – for example, isolations, level-shifters, power switches, retention flops, etc. – also refer to design or HDL instances, ports, nets for inferring or inserting corresponding cells, and connecting control signals.

For the UPF constructions, it is also required to understand that the syntax and semantics of UPF constituent parts are strictly defined in the Language Reference Manual (LRM). The syntax ensures accurate definition and semantics guides to abide with the inherent logical and lexical meaning of the defined constructs.

In addition, it is also required to comprehend that UPF is the driving force for all PA design verification and implementation automation tools. These tools interpret and analyze the UPF fundamental constructs to source and sink communication models for inter or intra domain communications, strategy association, MV or PA cell inferring or insertions, deploying corruption models, facilitate debugging, reporting results, and so on. The tool-specific UPF construct interpretation is discussed in details in Chaps. 5, 6 and 7.

The current chapter, and succeeding sections focus on the fundamental construct of UPF and its methodologies, primarily based in light of the LRM and secondarily on different design implementation choices with possible adoption of different power dissipation reduction techniques, as discussed in Chap. 2.

3.1.1 UPF Power Domain and Domain Boundary

In context to the previous Chap. 2, Sect. 2.1, as shown in Figs. 2.1 and 2.2, a power domain of a top design module named cpu_top can be defined as follows.

Example 3.1 Power Domain Definition

set_scope cpu_top

create_power_domain PD_top

Where **set_scope** specifies the current scope of the power domain in HDL hierarchical instantiation perception and **create_power_domain** defines the set of instances that are in the extent of that power domain. Even though the definition of Example 3.1 above stands for the UPF 2.0 LRM specification, however, the syntax

limits backward compatibility from UPF 2.1, where the power domain definition mandates to include *–elements* {}.

Although it is mentioned categorically in Chap. 2 that UPF is evolving with time and each release is not necessarily backward compatible. New releases are usually a superset of semantic and syntactic expressions from its predecessors. Hence power domain definition- syntax and semantics are explained below in light of UPF 2.1 and UPF 3.0.

Example 3.2 Power Domain Definition Syntax

create_power_domain domain_name
[-***atomic***]
[-***elements*** element_list]
[-***exclude_elements*** exclude_list]
[-***supply*** {supply_set_handle [supply_set_ref]}]*
[-***available_supplies*** supply_set_ref_list]
[-***define_func_type*** {supply_ function pg_type_list}]*
[-***update***]

Revisiting the power domain definition through the **create_power_domain** command, it defines a power domain and the set of instances through ***-elements*** <elements_list> options that are within the extent of the current power domain. The ***-atomic*** specifies the minimum extent of the power domain. And ***-exclude_elements*** is used to filter or exclude instances from the effective <elements_list>.

Hence UPF also imperatively defines an effective list of elements often termed as <effective_element_list> which is the result of the application of ***-elements*** and ***-exclude_elements***. However, the effective list and the term <effective_element_list> is not a UPF command or option.

The **create_power_domain** command also defines the supply sets that are used to provide power to the instances within the extent of the power domain. The ***-supply*** option defines a supply set handle for the supply set specified for a particular power domain. A domain <supply_set_handle> may be defined without an association to a <supply_set_ref>. The <supply_set_ref> may be any supply set visible in the current scope. If <supply_set_ref> is also specified, the domain <supply_set_handle> is associated with the specified <supply_set_ref>, as shown in Example 3.3 below.

Example 3.3 Supply Set Association of Power Domain

associate_supply_set supply_set_ref
-handle supply_set_handle

The ***-handle*** may also reference a power domain as follows.

-handle domain_name.supply_set_handle

More detail on supply set association with specific examples is discussed in succeeding stanzas and sections.

The ***-available_supplies*** option of **create_power_domain** provides a list of additional supply sets that are available for the domain, and the list is usually used to facilitate implementation tools to provide power connectivity to the cells inserted in this domain.

The *-define_func_type* specifies the mapping from functions of the power domain primary supply set to pg_type attribute values in the <pg_type_list>. To note, pg_type defines the ***UPF_pg_type*** or Liberty `pg_type` attribute (discussed in Chap. 4) of a supply port (e.g. primary_power, primary_ground, nwell, etc.). This mapping determines the automatic connection semantics used to connect the domain's primary supply to HDL or design instances within the extent of the domain.

The ***-update*** is relevant to incremental refinement of different parameters of a power domain, and may be used to add elements and supplies to a previously created domain, later in the design implementation phase.

It is worth noting here that the common power domain definition- syntax and semantics are also changed because the following options are deprecated.

List 3.2 Deprecated Options of create_power_domain

[-***include_scope***]
[-***scope*** instance_name]

Hence, apart from the common syntactical explanation, as illustrated above, for power domain definition through **create_power_domain**, it is also required to know the bindings of the definition that are semantically enforced. Since the listed options of List 3.2 are deprecated from UPF 2.1, the new semantics impose that the ***-elements*** option must be used at least once in the specification of a power domain, using **create_power_domain** through one of the following procedure.

List 3.3 Mandatory Usage of -elements for Power Domain Definition

– -***elements*** may be present during definition of the power domain in the first place, or
– It may be added later during the subsequent updates of the power domain using the -***update*** option.

It is also imperative that if the container of the <effective_element_list> is an empty list, a domain with the name <domain_name> will be created, but with an empty extent. Hence, a list of effective elements are required even for the default top power domain or it is recommended to define as follows.

Example 3.4 Power Domain Definition Based on UPF 2.1 and UPF 3.0

set_scope cpu_top
create_power_domain PD_top ***-elements {.}***

Where ***-elements {.}*** will include the current scope (cpu_top) and all of its descendant HDL instances in the power domain PD_top.

In contrast, power domains can also be defined as follows.

Example 3.5 Variation of Power Domain Definition

create_power_domain PD_top ***-elements*** {udecode_top ualau_top ... uN_top}

Here the extent of the PD_top will not include cpu_top in the power domain, but would only include its descendant instances, namely udecode_top, ualu_top....until the uNth_top, if or when available. Hence, when an instance is specified in the <elements_list> of the **create_power_domain** command, that instance and all its

children are added transitively to the power domain. However, there is an exception for the instance that is explicitly added to into another power domain's elements list.

In addition, it is also important to mention here that the UPF LRM specifies that **-elements** { } is implicitly followed by 'transitive TRUE' options, although '–transitive' is not an explicitly defined option in **create_power_domain** command. The transitive nature impacts the resultant design elements or the <effective_element_list> of a power domain through the ***–elements*** { } and ***-exclude_elements*** { } options as explained in Example 3.6 and Fig. 3.1 below.

Considering a design with current scope "A", contains child elements B, C, and D; the child element B further branches to E and F, C branches to G and H, and D branches to I and J elements respectively.

Example 3.6 Transitive Nature of Design Elements in Defining Power Domain

create_power_domain PD_test \
-elements {A A/C/H} \
-exclude_elements {A/C A/D}

Since –transitive TRUE is implicitly present in the Example 3.6, hence the elements processing can be represented by Fig. 3.1(b). Where the specified elements in

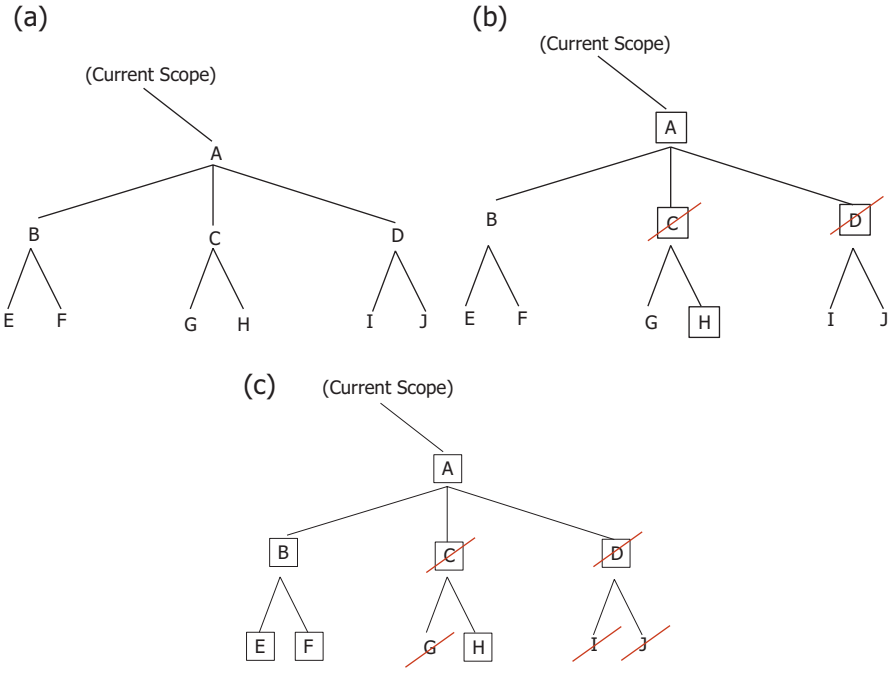

Fig. 3.1 (**a**) Example of power domain definition and elements processing (Courtesy: IEEE 1801-2015 LRM). (**b**) Example of elements processing based on the Example 3.6 (Courtesy: IEEE 1801-2015 LRM). (**c**) Resultant elements processed based on the Example 3.6 (Courtesy: IEEE 1801-2015 LRM)

create_power_domain commands that are confined in boxes and excluded are shown with strike-out lines.

Finally, the transitive nature of the UPF elements specification prevails the <effective_ element_list> as {A A/B A/B/E A/B/F A/C/H} and illustrated in Fig. 3.1(c).

Hence design elements {A/C A/C/G A/D A/D/I A/D/J} are excluded from forming the resultant power domain PD_test.

It is evident from the discussion that the fundamental concepts of power domains that specify and confine certain portions of a designs or elements, defined through UPF **create_power_domain –*elements* {}** and ***-exclude_elements* {},** play significant roles in establishing connectivity for inter-domain and intra-domain communications.

It is critical to understand that the formation of power domains inherently defines its domain boundary and domain interface through the UPF **create_power_domain** command and options. All other UPF parameters are usually relevant to power domains and are developed around the power domain boundary and domain interface. Specifically the power supply, UPF strategies, logic or supply ports and nets, corresponding connectivity, and subdomain hierarchical connections- all are established through the domain boundary and domain interface.

The formation of a power domain boundary can be further explained through exploiting the common and formal port definitions for actual signal connections in terms of hierarchical design instances.

Any port on the domain boundary possesses connectivity semantics in the higher side of design hierarchy for inter-domain communication, also known as the "HighConn" side of ports. On the other hand, the lower side of design hierarchy connectivity semantics for intra-domain communication are known as the "LowConn" side of ports.

Obviously the context of HighConn and LowConn are always dependent to the extent of the current power domain and its relation with higher or lower hierarchical power domains. Figure 3.2 shows the concepts of HighConn and LowConn side of ports for a sub hierarchical power domain PD_sub1 on the domain interface with the default top power domain PD_TOP.

Hence, it is also required to understand that the context between PD_TOP and PD_sub1, in order to realize the boundary interface of these two power domain. As shown in Fig. 3.2, PD_TOP is the parent power domain in the context of PD_sub1, hence the UPF LRM further defines two additional terminologies to establish the context of the "interface of a power domain."

When considering the HighConn side of each port of each boundary instance in the extent of PD_TOP as parent of PD_sub1, the side of the interface is known as the "lower boundary" of PD_TOP power domain.

Conversely, when considering the LowConn side of each port of each boundary in the context of PDSub1 as child of PD_TOP, the side of the interface is known as the "upper boundary" of the PD_sub1 power domain.

Obviously the union or blending of "lower" and "upper" power domains are the interface between PD_TOP and PD_sub1 power domains. It is essential to mention

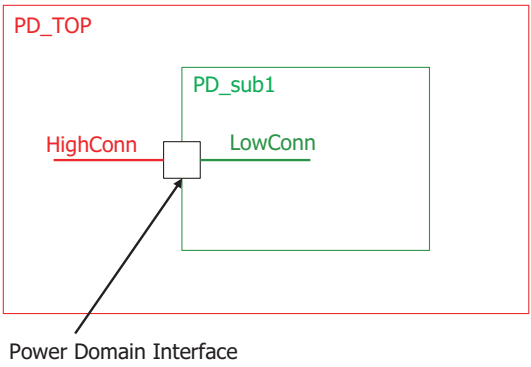

Fig. 3.2 Concepts of HighConn and LowConn side of ports for PD_sub1 on the domain boundary and domain interface with PD_TOP

here that the concepts of the HighConn and LowConn side of ports on a power domain boundary and power domain interface will govern most of the PA verification methodologies that are explained in subsequent Sections and Chapters.

It is also required to know that a logic ports can become a source, a sink, or both, based on following criterion.

List 3.4 Criterion for Logic Ports to Become Source or Sink

- The LowConn of an input or inout logic port whose HighConn is connected to an external driver is a "source".
- The HighConn of an output or inout logic port whose LowConn is connected to an internal driver is a "source".
- The LowConn of an output or inout logic port whose HighConn is connected to an external receiver is a "sink".
- The HighConn of an input or inout logic port whose LowConn is connected to an internal receiver is a "sink".
- For a logic port that is connected to a driver, the supply of the connected driver is also the driver supply of the port.

A primary input port is assumed to have an external driver and therefore is a source; such a port has a default driver supply if it does not have an explicitly defined ***UPF_driver_supply*** attribute.

An internal port that is not connected to a driver is not a source, and therefore, does not have a driver supply in the design. To model this in verification, an anonymous default driver is created for such an undriven port. This driver always drives the otherwise undriven port in a manner that results in a corrupted value on the port. Corruption detail are discussed in Chap. 5.

For a logic port that is connected to one or more receivers, the supplies of the connected receivers are all receiver supplies of the port. A primary output port is assumed to have an external receiver and therefore is a sink; such a port has a default receiver supply if it does not have an explicitly defined ***UPF_receiver_supply*** attribute.

An internal port that is not connected to a receiver is not a sink, and therefore, does not have any receiver supplies. To note, UPF attributes (***UPF_driver_supply*** or

UPF_receiver_supply) are further discussed in succeeding sections in conjunction to relevant UPF commands for UPF modeling.

Apart from the regular power domain, there is also a simple container for power domains defined through the UPF **create_composite_domain** command. The composite domains are usually composed of at least one or more domains, identified as subdomains. The syntax and semantic explanation are given below.

Example 3.7 Composite Power Domain Definition Syntax

create_composite_domain composite_domain_name
[-***subdomains*** subdomain_list]
[-***supply*** {supply_set_handle [supply_set_ref]}]
[-***update***]

Unlike a regular power domain, a composite domain has no corresponding physical region on the abstraction space or real silicon. The –***subdomains*** <subdomain_list> is the container of subdomains and -***supply*** functionality resembles exactly the same of -***supply*** for **create_power_domain**.

Though attributes like power states and the primary <supply_set_handle> can be specified on a composite domain, but these attributes have no effect on its subdomains.

However, there are specific UPF commands and strategies that can be imposed on the composite domain and will be applied to each of the subdomains in the <subdomain_list>.

The list of applicable UPF commands and strategies on composite domains are shown below. These commands will be applied to all subdomains in the <subdomain_list> only if they are appropriate.

List 3.5 Applicable UPF Commands on Composite Domains

connect_supply_net
map_power_switch
map_retention_cell
set_isolation
set_level_shifter
set_repeater
set_retention
use_interface_cell (UPF 3.0 syntax for **map_level_shifter_cell** and **map_isolation_cell**)

During the discussion on the formation of regular and composite power domains, it became evident that the following UPF attributes are established within the extent of power domains.

List 3.6 UPF Attributes Established Around Power Domains

– The power supply
– UPF strategies
– UPF logic or supply ports and nets
– Corresponding connectivity
– Subdomain hierarchical connections, etc.

Hence, in light of the Example 3.3, presumably once the power domains are defined that confines a portion of a design as elements, at first, it is required to provide a power supply to that power domain.

The UPF or IEEE-1801 LRM specifies that power domains are implicitly associated with a set of predefined primary supply sets and its handles. The supply set handles are usually the rooted name and reference to the supply set of a power domain, and also may be extended for power switching and other UPF strategies.

All elements within the extent of a power domain are implicitly connected to the primary supply set through handles of that power domain. Additional supply set handles that are local to a power domain can also be specified, depending on the power architecture and supply distribution network. The next section discusses the "supply sets", "supply set handles", and mechanism for associating the "supply set handles" with the "supply sets" as well as associating the "supply sets" with "supply nets".

3.1.2 UPF Power Supply and Supply Networks

In conjunction with the previous section, it is essential now to specify the power supply and establish the supply network on the power domains. UPF facilitates early specification of the power distribution network through power supply sets and power supply set handles without having to wait for a design to roll down to place &route to know the exact details of actual supply net connections.

A supply set can be viewed as an abstract bundle of related power supply functions where each function within the supply set is associated with a power supply net. There are six standard functions as listed in List 3.7.

List 3.7 UPF Power Supply Set Functions
- **power**
- **ground**
- **nwell**
- **pwell**
- **deepnwell** and
- **deeppwell**

However, typically, other than power and ground, which are always required, bias states represented by nwell, pwell, deepnwell, and deeppwell supply set functions are required only in special circumstances to model body biasing to reduce leakage power.

Typically, the association of supply set and primary supply set handles for an already defined power domain are accomplished through the following UPF syntax shown in Example 3.8 below.

Example 3.8 UPF Syntax for Defining Supply Sets

create_supply_set set_name

[-***function*** {func_name net_name}]*
[-***reference_gnd*** supply_net_name]
[-***update***]

Here the -***function*** option defines the function of a supply net provided for the supply set. The <net_name> is a rooted name of a supply net or supply port or a supply net handle and must be in the current scope. The <func_name> must be one of the six standard functions (e.g. power, nwell, etc.) as noted in the List 3.7 above. The –***function*** option associates the specified <func_name> (e.g. primary) of this supply set with the user specified <supply_net_name>.

It is important to note that for a particular supply set <set_name>, the same <func_name> cannot be associated with two different supply nets. The <supply_net_name> may be a reference to a supply net in the descendant hierarchy of the current scope using a supply net handle.

The -***reference_gnd*** option defines the rooted name of supply net that serves as the reference ground for the defined supply set. When this option is not specified, the voltages are evaluated without offset or scaling.

However, -***reference_gnd*** option is deprecated from UPF 3.0, because reference ground for the supply set can also be specified through the -***function*** option or dotted name, discussed in the following stanzas.

The -***update*** is used for updating an already defined supply set through the UPF **create_supply_set** command. It is also used for updating the supply set handle that was previously defined implicitly or explicitly through the **create_power_domain** UPF command. It is not permitted to reference a previously created supply set or its handle without using –***update***. Alternately, this restriction also applies to using the -***update*** options for a supply set or its handle that has not been previously defined.

The same restrictions apply for specifying a supply set handle that has not been previously defined. The following Example 3.9 shows the definition of a supply set VDD1_ss in light of the syntax and semantics shown and explained above in Example 3.8.

Example 3.9 Defining Supply Sets

create_supply_set VDD1_ss \
-***function*** {power} \
-***function*** {ground} \
-***function*** {nwell}

The above Example 3.9 defines the supply set with the required functions specification. Obviously it is required to associate them with the exact supply nets later in the DVIF, through the mandatory –***update*** option, as shown below.

Example 3.10 Association of Predefined Supply Set with Supply Nets

create_supply_set VDD1_ss –***update*** \
-***function*** {power VDD1_net} \
-***function*** {ground VSS_net} \
-***function*** {nwell VNW_net}

Hence it is evident that the supply set actually represents a potential connection to a corresponding portion of a design through supply nets. After all, supply set allows designers to define the power supply for power domains. UPF allows numerous ways to define supply set and associated handles for power domains.

It is also possible to define and associate supply set and supply set handles through UPF **create_power_domain** commands and *-**supply*** options.

Recalling the syntax and semantic of **create_power_domain** explained in Sect. 1.1, the supply set definition and supply set handle association in Examples 3.10 and 3.10 can be further made more concise and represented as follows.

Example 3.11 Defining and Association of Supply Set Through Handles for Power Domains

```
create_power_domain PD_top \
   -supply {primary VDD1_ss}
…
create_power_domain PD_sub2 \
   -supply {primary VDD2_ss}
```

The examples for creation of additional supply set handles for a power domain and their association through *–**update*** are shown below in the following two consecutive Examples 3.12 and 3.13.

Example 3.12 UPF Syntax for Defining Supply Sets

```
create_power_domain  PD_top \
-supply {iso_ss} \
-supply {ret_ss}
```

Example 3.13 UPF Syntax for Defining Supply Sets

```
create_power_domain PD_top -update \
-supply {primary VDD1_ss} \
-supply {isolation VDD2_ss} \
-supply {retention VDD2_ss}
```

The supply set handles are generally treated as a local supply set and in addition to power domain, these handles can also be defined for PSW, ISO, LS, ELS, and RFF strategies. The following examples shows the supply set handles for an ISO strategy.

Example 3.14 Supply Set Handles for ISO Strategy

```
create_power_domain PD_mem_ctrl -elements {mc0}
# Designate actual supply sets for isolation supplies of PD_mem_ctrl
associate_supply_set VDD_top_ss -handle PD_mem_ctrl.isolate_ss
```

Example 3.15 Variation of Supply Set Handles for ISO Strategy

```
create_power_domain PD_mem_ctrl -update \
   -supply { retain_ss } \
   -supply { isolate_ss }
```

```
# Designate actual supply sets for isolation supplies
set_isolation mem_ctrl_iso_1 \
   -domain PD_mem_ctrl \
   -isolation_supply_set PD_mem_ctrl.isolate_ss \
   -clamp_value 1 \
   -elements {mc0/ceb mc0/web} \
   -isolation_signal mc_iso \
   -isolation_sense high
```

However, the most obvious supply set definition and associations through their handles are done by explicitly referencing power domains and supply functions, i.e.

```
associate_supply_set VDD_top_ss -handle PD_mem_ctrl.isolate_ss
or
-isolation_supply_set PD_mem_ctrl.isolate_ss
```

These two are shown in the previous two examples, Examples 3.14 and 3.15 respectively.

The following two consecutive examples, Examples 3.16 and 3.17 show further variations of defining supply set based on the discussions and examples so far.

Example 3.16 Variation in Defining Supply Sets

```
create_supply_set VDD1_ss
create_supply_set VDD2_ss
create_supply_set VDD1_sw_ss
```

Example 3.17 Supply Set Association Through Supply Set Handles for Power Domains

```
associate_supply_set VDD1_ss -handle PD_top.primary
associate_supply_set VDD2_ss -handle PD_sub2.primary
associate_supply_set VDD1_sw_ss -handle PD_sub1.primary
```

Here the **associate_supply_set** command associates two or more supply sets. Supply set association implicitly connects corresponding functions and, as a result, makes them electrically equivalent.

The explicit *–handle* option is the rooted name of a supply set of a power domain, power switch, or strategy. Handles are names, rooted in the active scope. This allows the association of a power domain supply set with a supply set defined in a different scope by first setting the active scope to one in which both the supply set and the domain or strategy are visible.

The following lists explain a few examples of UPF LRM recommended usage of supply set handles.

Example 3.18 UPF Recommended Usage of Supply Set Handles

– The predefined supply set handle for a power domain is as follows.

– <domain_name>.*primary*

– Supply set handles for user defined supply sets of a power domain are also permitted.

– The predefined supply set handle for a power-switch switch_name is

– <switch_name>.***supply***
– To note, the UPF 3.0 LRM specifies the ***supply*** as ***switch_supply***
– The predefined supply set handles for an isolation cell strategy isolation_name of a power
– domain domain_name are
– <domain_name>.<isolation_name>.***isolation_supply_set***
– If there is only one isolation supply set, or
– <domain_name>.<isolation_name>.***isolation_supply_set***[index]
– Where index starts at 0, if there are multiple isolation supply sets.
– To note, the UPF 3.0 LRM specifies the ***isolation_supply_set*** as ***isolation_supply***.
– The predefined supply set handles for a level-shifter strategy level_shifter_name of power
– domain domain_name are
– <domain_name>.<level_shifter_name>.***input_supply_set***
– <domain_name>.<level_shifter_name>.***output_supply_set*** and
– <domain_name>.<level_shifter_name>.***internal_supply_set***
– To note, the UPF 3.0 LRM specifies these pre-defined level shifter handles as ***input_supply, output_supply,*** and ***internal_supply*** respectively.
– The predefined supply set handle for a retention strategy retention_name of a power
– domain domain_name is;
– <domain_name>.<retention_name>.***retention_supply_set***.
– To note, the UPF 3.0 LRM specifies this ***retention_supply_set*** as ***retention_supply***.

It is important to understand that the predefined supply set handles that are shown as (.) dotted references or "dotted names" for a power domain, PSW, ISO, LS, RFF, etc. are not necessarily the supply set or power supply syntax, for example:

.supply or
.switch_supply

These are not from the definition of **create_power_domain, crate_power_switch, set_isolation** etc**.** And they are known as "record field names" and are reserved in the specified context. These "records" comprise a name and a set of zero or more values. Record field names are in a local name space of the UPF object; e.g., a power domain may have strategies and supply set handles. Strategies themselves may also have supply set handles.

It is also important to understand that the dotted referencing of predefined handles are also possible through the actual power supply syntax from the definition of the commands**.** For example:

<domain_name>.<retention_name>.***retention_supply_set***

However, this is from the above list Example 3.18 and it is part of the **set_retention** command as follows.

[***-retention_supply_set*** *ret_supply_set*]

Hence it may be summarized that supply set handles are dotted names that refer to supply sets defined as part of a power domain, supply sets associated with UPF strategies, supply sets associated with a power domain, and functions of a supply set. It is visibly clear at this point that there are numerous ways to define and associate supply sets that can best be summarized as shown in the List 3.8.

List 3.8 Variations of Defining Supply Set and Association Through Supply Set Handles

- **create_supply_set with *–function***
- **create_supply_set with *–function*** and ***-update***
- **create_power_domain *–supply***
- **create_power_domain *–supply*** and ***–update***
- **create_supply_set** and **associate_supply_set** and ***–handle***
- Supply set handles with "dotted name" and "record field names".

It is evident from the discussion in this Sect. 1.2 that the defining power supply for a power domain and design elements explicitly impacts power supply network distribution in UPF strategies, like ISO, LS, etc. Obviously, UPF strategies are introduced once power domains are defined and power supplies are imposed on them. The succeeding sections will explain the UPF power strategies from the syntax, semantics, and implementation perspectives. However, it is required to define and explain the UPF power states before the power strategies are discussed.

3.1.3 UPF Power States

The third crucial and fundamental parameter for UPF modeling are the UPF strategies. However, strategies are actually based on the status of the power states of the power supplies of the power domains. Obviously, because an ISO may be required between an Off and an On power domain to isolate unknown value propagation; however, the On or Off states of the corresponding power domains are revealed through the power states of the power supplies of the domains. Hence power states are discussed in this section before going into detail on UPF power strategies in Sect. 1.4.

In fact, the UPF power state is the core of the entire UPF modelling and power aware verification. The power states are defined through UPF **add_power_state** commands that define one or more power states of an object. The object includes power domains, supply sets, composite domains, groups, models, and instances.

It is worth mentioning here that power states may also be represented through the states of a power state tables (PST), which is constructed through the combination of **add_port_state**, **add_pst_state,** and **create_pst.** However these commands are considered as legacy in UPF 2.1 LRM and onward because of their limitations to coordinate with supply sets; specifically PST states are defined based on the supply net only, and supply nets are usually available after the synthesis and post-layout

levels of design abstraction. In addition, there are no UPF methodologies for PST states to refine states through DVIF, and so on.

The following Example 3.19 shows the UPF power state syntax through the **add_power_state** command.

Example 3.19 UPF Power States Syntax

add_power_state *object_name*
[**-supply** | **-domain**]
[**-state** {*state_name*}]
[**-supply_expr** {*boolean_expression*}]
[**-logic_expr** {*boolean_expression*}]
[**-simstate** *simstate*]
[**-legal** | **-illegal**]}]*
[**-complete**]
[**-update**]

As already mentioned, the **add_power_state** defines one or more power states of an object, but each power state definition is independent of any other power state definition. Two different power states of the same object may have intersecting or overlapping ***-supply_expr*** and (or) ***-logic_expr*** expressions. Such states may have different legalities.

A power domain or a supply set may be in a state that matches more than one power state definition. Hence the object, i.e. the power domain and power states, are usually referred through the ***-domain*** and ***–supply,*** and defining any power state through <object_name> has specific rules set by the UPF LRM, depending on the type of objects used. The following list summarizes the rules at a glance.

List 3.9 UPF Power States <object_name> Depending on the Target Objects

– If ***-supply*** is specified, the <object_name> must be the name of a supply set or a supply set handle.
– If ***-domain*** is specified, the <object_name> must be the name of a power domain.
– If none of the above are specified, the type of <object_name> determines the kind of object to which the command applies.

It is also important to note that in conjunction with ***-supply*** and ***-domain***, UPF 3.0 LRM defines ***group***, ***model***, and ***instance*** as objects. Hence <object_name> for such objects also requires similar rules as those shown in List 3.9.

The UPF syntax dictates that the **add_power_state** attributes an object with a power state definition. The functionality and semantics of each option of the power state syntax shown in the Example 3.19 is explained through the examples of a complex SoC design shown in Fig. 3.3.

The SoC example is constructed with 4 CPU instances with 2 CPU cores within each CPU cluster. The default top power domain is PD_SOC. At the SoC level there are 2 clusters A and B and a "System Power Unit (SPCU)", which is a power domain PD_SPCU. Also there are interconnects and other design IPs in the design.

The "CPU Cluster A" consists of PD_CPUA0 and PD_CPUA1 power domains that are architected in such a way so that the CPU's can be implemented standalone.

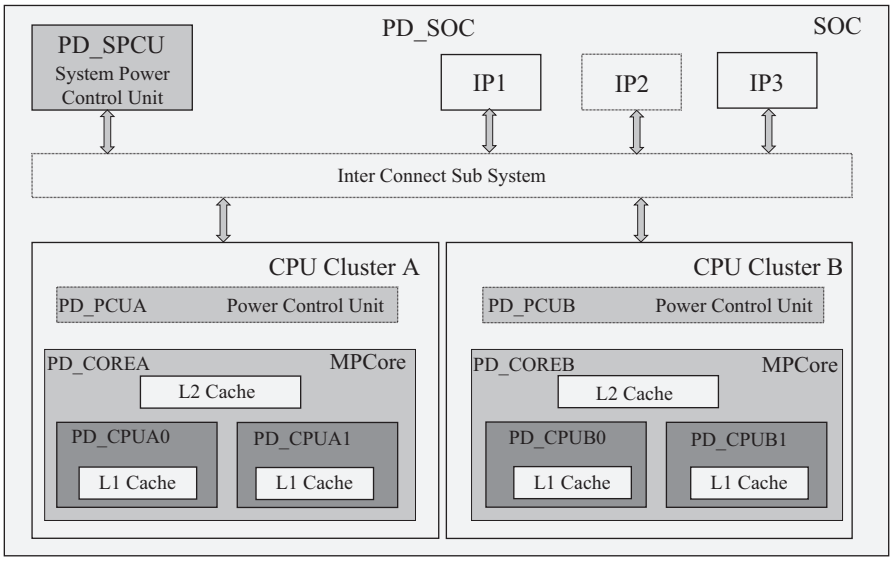

Fig. 3.3 Example of complex SoC (Courtesy: ARM SoC)

The CPUs in a cluster shares an L2 cache and each CPU has an L1 cache. These caches are considered as separate power domain PD_L2 and PD_L1.

In addition, each CPU cluster is also a power domain, e.g. PD_COREA. And each possesses a secondary "Power Control Unit" confined in PD_PCUA power domain. When the PD_COREA is turned OFF, then PD_CPUA0 and PD_CPUA1 power domains are also turned OFF. The shared L2 cache can be ON when any of the CPU is ON. The cluster level domain PD_COREA has ON, OFF, RET, MEM_RET power states and the core level CPUs power domain PD_CPUA0 has ON, OFF and RET power states.

So the hierarchical and power domain boundaries are visibly clear through the color gradation from Fig. 3.3.

As discussed so far, in order to define the power states of the SoC using **add_power_state**, starting from the cluster CPUs, the recommended approach is:

List 3.10 Recommended Approach to Define Power States for Any Design

– First create power states of the supply sets of the power domain and
– Then create the power states of the power domain in terms of its supply sets, as well state of any dependent power domains.

This hierarchical composition makes it possible for a SoC architect to relate the power states of dependent IPs (includes third party IP) to the "System Level Power States", that is further discussed in succeeding sections.

In order to define the power states of a supply set for the PD_CPU0 power domain in CPU Cluster A, based on the recommended approach of List 3.10, it is

required to define the power states through the primary of the supply set handle (PD_CPUA0A0.***primary***), as shown in Example 3.20.

Example 3.20 Power States of a Supply Set

add_power_state PD_CPUA0.***primary*** \

-***state*** {ON -***simstate*** NORMAL -***logic_expr*** {pwr_ctrl==1}
-***supply_expr*** {***power***=={FULL_ON,1.0} && ***ground***=={FULL_ON,0}} }
-***state*** {OFF -***simstate*** CORRUPT -***logic_expr*** {pwr_ctrl==0}
-***supply_expr*** {***power***==OFF && ***ground***=={FULL_ON,0}} }

Hence the **add_power_state** for the supply set can be defined through -***logic_expr***, -***supply_expr*** and -***simstate***. The logic expression is a Boolean expression defined in terms of logic ports, logic nets, interval functions, and power states of the given supply set or supply set handle (e.g. PD_CPUA0.***primary***). The supply expression is also a Boolean expression that may reference available supply nets, supply ports, and (or) functions of supply sets or supply set handles.

The –***state*** <state_name> in the syntax Example 3.19 and used in Example 3.20 is a simple name of the state that is defined and also may be refined over time throughout the DVIF.

To note, a simple name in UPF is a single identifier usually used when creating a new object in a given scope; i.e., the identifier becomes the simple name of that object.

The power states definition through the power domain for PD_CPUA0 is shown below.

Example 3.21 Power States of a Power Domain

add_power_state PD_CPUA0 \
-***state*** {ON -***logic_expr*** {PD_CPUA0.aon==ON && PD_CPUA0.***primary***==ON}}
-***state*** {RET -***logic_expr*** {PD_CPUA0.aon==ON && PD_CPUA0.***primary***==OFF}}
-***state*** {OFF -***logic_expr*** {PD_CPUA0.aon==ON && PD_CPUA0.***primary***==OFF}}

In this Example 3.21, the **add_power_state** for the power domain is defined only through -***logic_expr***, which is simply defined in terms of power states of supply sets or supply set handles, and (or) power states of lower-level power domains, logic nets etc., but without any -***supply_expr*** or -***simstate***.

From the examples, it is evident that ***logic_expr*** and ***supply_expr*** are design logic and power supply conditions, respectively, that cause the power state to become (or remain) active.

From Example 3.20, it is also distinctive that the -***simstate*** option specifies the power-related behavior of the logic expression of the power domain and implicitly connected to the supply set. Hence –***simstate*** plays a vital role in PA verification. When a simstate of a supply set is the primary power of a given power domain, it actually determines the simulation behavior of all the design elements in that domain.

The -***supply_expr*** specifies a Boolean expression defined in terms of supply ports, supply nets, and (or) supply set handle functions.

Similarly the *-logic_expr* specifies a Boolean expression defined in terms of logic nets and (or) power states of supply sets and (or) power domains.

The *-simstate* option specifies a simstate for the power states associated with a supply set. Here the ***-simstate*** <simstate> defines a simulation state early in the DVIF abstraction from RTL.

When a simstate of a supply set is the primary power of a given power domain, it actually determines the simulation behavior of all the design elements in that domain. The valid state values of simstate ranges from NORMAL to CORRUPT, with intermediate values bearing specific significance for higher sensitivity to simulation changes that could cause corruption. The corruption semantics are discussed in Chap. 5. The following list shows the valid values for simstate.

List 3.11 Valid Values for the Simstate States
– NORMAL,
– CORRUPT_ON_CHANGE,
– CORRUPT_STATE_ON_CHANGE,
– CORRUPT_STATE_ON_ACTIVITY,
– CORRUPT_ON_ACTIVITY,
– CORRUPT, and
– NOT_NORMAL.

The NOT_NORMAL simstate actually provides the flexibility to allow incremental refinement to any other simstate, other than NORMAL, through **add_power_state** ***-update*** commands and options. The detailed discussion of incrementally refinable UPF is available in next Sect.3. Regarding *-simstate* values, it is important to note that there are two additional predefined or default power state values, UNDEFINED and ERROR.

These are defined for every supply set or all UPF objects. When a supply set is defined with no *-simstate* the status is changed to UNDEFINED; however the ERROR state appears only when the simstate is CORRUPT. In fact, UNDEFINED and ERROR states are UPF 3.0 LRM semantics. The UPF 2.1 LRM does not specify UNDEFINED; instead it specifies the default DEFAULT_NORMAL, which represents a significantly different usage perspective, specifically in dynamic simulation. Although UPF 2.1, for the ERROR (or CORRUPT) states specifies a state value termed DEFAULT_CORRUPT.

Example 3.22 Definition and Usage of Simstate

```
#Without -supply or -logic_expr
add_power_state PD_COREA.primary -supply \
-state {TOP_ON -simstate NORMAL} \
-state {TOP_OFF -simstate CORRUPT}
#With -supply_expr
add_power_state PD_COREA.primary –supply -update \
-state{TOP_ON -supply_expr {power=={FULL_ON 0.81} && ground=={FULL_
ON 0.00}}} \
-state{TOP_OFF -supply_expr {power=={OFF}}}
```

#With ***-logic_expr***
add_power_state PD_COREA**.primary –supply -update** \
-state {TOP_ON **-logic_expr** {nPWRUP_CPU0==0}} \
-state {TOP_OFF **-logic_expr** {nPWRUP_CPU0==1}}
#With ***-simstate*** NORMAL, CORRUPT and with both the ***–logic_expr*** and ***–supply_expr***
add_power_state PD_COREA.***primary -supply*** \
-state {TOP_ON ***-simstate*** NORMAL ***-supply_expr*** {power == FULL_ON && ground == FULL_ON}}\

At this point, the power state is specified with a state name that consists of a logic expression, supply expression, and simstate. However, additional options of power states from the syntax Example 3.19 are discussed below.

The **-complete** option specifies that all fundamental power states to be defined for any object have been defined completely. This implies that all legal power states are defined, and any state of the object that does not match a defined state is regarded as an illegal state.

The ***-update*** indicates that this option provides additional information for a previously defined power state with the same <object_name> and that is targeted to be executed within the same scope. Hence it is possible to define a set of power states for a power domain or supply set and refine it incrementally though ***–update.***

When a power state is defined with ***-supply_expr*** or with ***-logic_expr*** and refined incrementally with another ***-supply_expr*** or ***–logic_expr*** through update, the definition becomes the conjunction of the two expressions, respectively, as shown in the following Example 3.23.

Example 3.23 Conjunction of -*supply_expr* or -*logic_expr* Through Incremental Update

supply_expr' = (previous ***-supply_expr****) && (****-update -supply_expr****)*
logic_expr' = (previous ***-logic_expr****) && (****-update -logic_expr****)*

It is also clear that the incremental refinement concept of power states (discussed in further detail in succeeding Sects. 2 and 3) allows an already defined power state to be refined through its objects if required or on demand. They derive sets of power states for different design abstraction levels starting from the RTL and applicable up to the PG-netlist through constraint, configuration, and implementation UPF (also discussed in Sects. 2 and 3).

In general, the UPF LRM specifies any power state as a "named power state" that is defined using **add_power_state** for a supply set and power domain. In addition, a named power state may also be defined for following UPF attributes.

List 3.12 Attributes for Defining Named Power State

– Composite domain,
– Group,
– Model, or Instance,
– Predefined power state with ON and OFF states for supply sets, and
– Predefined power states with UNDEFINED and ERROR states for all objects that have power states.

However, it is also important to know that named power state may also be defined using **add_port_state**, or **add_pst_state** for a supply set or power domain.

The following example shows the sample named power state defined through **add_power_state**.

Example 3.24 Sample Named Power State

```
    add_power_state PD_CPUA0.primary -supply \
   -state {UNDEFINED -logic_expr {PD_CPUA0.primary != ON && PD_CPUA0.
primary != OFF} } \
    -state {ON –simstate NORMAL \
    -supply_expr {power == FULL_ON && ground == FULL_ON} } \
    -state {OFF -simstate CORRUPT \
    -state {ERROR -simstate CORRUPT \
    -logic_expr {PD_CPUA0.primary == ON && PD_CPUA0.primary == OFF} }
```

The named power states shown in Example 3.24 is constructed from both *–logic_expr* and *–supply_expr* for power domain PD_CPUA0, and its primary supply set is denoted through the handle as PD_CPUA0.*primary*.

In addition it is worth to note that for the "state values" for the states of power supply or supply set, the UPF specifies to use one of the following.

List 3.13 State Values for the Supply Set

- OFF,
- UNDETERMINED,
- PARTIAL_ON,
- FULL_ON values.

The state value represents the ability for an object to become the root supply source to provide power. In addition to "named state", UPF LRM further defines a "fundamental power state" that is not a derivative or product of refinement of any other power state of an object.

Fundamental power states of a given object are mutually exclusive and do not refer to any other power state of the same object. The following example explains the roots of fundamental power states of an object (i.e., power domain PD_CPUA0) in accordance with UPF allowed attributes for defining named power states.

Example 3.25 Sample Fundamental Power State

```
    add_power_state PD_CPUA0 -domain \
     -state {UNDEFINED -logic_expr {PD_CPUA0 != RUN && PD_CPUA0 !=
SHD} } \
    -state {RUN –logic_expr {primary == ON} } \
    -state {SHD –logic_expr {primary == OFF} } \
    -state {ERROR -logic_expr {PD_CPUA0 == RUN && PD_CPUA0 == SHD} }
```

So it is also clear that the fundamental power states are named power states. The example explains UNDEFINED, ERROR, and two other RUN and SHD power states for the PD_CPUA0 domain that are not refined at this point. Predefined power states UNDEFINED and ERROR represent situations in which the set of power

state definitions for an object is either inconsistent or incomplete. UPF 3.0 LRM restricts the fundamental power state active status through the following guidelines.

List 3.14 UPF Guidelines for Specifying Fundamental Power States
- A fundamental power state of an object will remain active whenever any refinement of that power state is active.
- Two different refinements of the same power state must be mutually exclusive.
- The predefined ERROR power state represents the error condition in which two states that should be mutually exclusive are both active at the same time.
- The current power state of an object will be considered as an ERROR state when two different fundamental states of the same object or two different refinements of the same power state are active at the same time.
- Two fundamental power states of the same object must not remain active at the same time.
- Defining a new fundamental power state is not allowed after the power states are marked complete.

The refinement of fundamental power states through –***update*** depends on several factors, including their mutual-exclusion, overlapping or non-mutual exclusion, conjunction, and disjunction of refinement possibilities and status.

The fundamental power state refinement concept extends the UPF specification and associated PA verification boundary to early stages of RTL design and allows verification virtually at any level of design abstraction. The concept allows an already defined power state to be refined through objects on-demand and derives sets of power states for different design levels of abstraction from RTL to PG-netlist. The refinement concept is actually derived from the fundamental power states definition and their implication guidelines. The conceptual sets of refining power states are shown in List 3.15.

List 3.15 Conceptual Sets of Fundamental Power States
1. Definite Power States,
2. Deferred Power States,
3. Indefinite Power States,

These conceptual sets of the List 3.15 are defined and explained below through examples.

1. Definite Power States A definite power state is a power state whose defining expression (i.e. –***logic_expr*** {}) is made up of a single term or conjunction of terms (i.e. usage of only "&&" operator) such that each term is one of the following:

- A Boolean expression over signal in the design or
- A term of the form <object> == <state>, where <object> is the name of an object for which power states are defined and <state> is the name of a definite power state of <object>.

The following Example explains the definition.

Example 3.26 Example of Definite Power States

```
# Definite power state for power domain
add_power_state PD_COREA -domain \
-state {RUN –logic_expr {primary == ON}} \
-state {SHD –logic_expr {primary == OFF}}
# Definite Power State with Design Signals or Controls in Logic Expression
add_power_state PD_COREA \
-state ON_HIGH {-logic_expr {(PD_COREA.primary == RUN && PD_CPUA0.
primary == INT_ON && PD_L2.primary == RAM_ON && MCTL_ON ==
1'b0)}}
```

From the above Example 3.26, it is possible to conclude that the definite power state defining expression consists of a single or conjunction of terms that may be constructed with <object>==<state> or it may be a Boolean expression over signals and other power domain status in the design. It is important to note that when *-**simstate*** is not specified, the verification tool treats the expression as NORMAL. However, for UPF strategies, i.e. isolation analysis, either implementation UPF or ***–simstate*** is required, which is further discussed in succeeding sections.

2. Deferred Power States A named power state is deferred if it has no defining expression (i.e. no ***–logic_expr***{}). The "deferred" states defining expression depends on precise design implementation parameters and remains deferred until later stages of design abstraction. However, eventually, deferred states are updated later through the process termed *refinement in place* (actually through ***-update*** in **add_power_state** command) and resolved into definite states. The following Example 3.27 shows the initial state or status of deferred power states based on the definition.

Example 3.27 Definition of Deferred Power States

```
add_power_state PD_COREA.primary -supply \
-state {RUN -simstate NORMAL} \
-state {SHD -simstate CORRUPT}
```

The deferred power state is just a container of ***-state*** names and possible speculation of simulation behavior with ***–simstate*** options. However the deferred power states in Example 3.27 is more relevant for **power_expr** <power_expression>; as shown in the syntax in Example 3.19. The ***–power_expr*** specifies the power consumption of this object (PD_COREA) in this power state, or a function for computing the power consumption. UPF 3.0 LRM specifies that a power expression must be specified only for a deferred power state. The following Example 3.28 shows more relevant example of refinement of deferred power states.

Example 3.28 Definition of Deferred Power States Relevant to Refinement

```
add_power_state PD_COREA.primary \
-state ON {-logic_expr {ln3 == 1} -simstate NORMAL} \
-state SHD {-logic_expr {ln3 == 0} -simstate CORRUPT}
```

The refinement relevant deferred power state in above Example 3.28 contains the design control signal and simstate to denote more complete reference of power states for PD_COREA power domain, rather than a container of state.

In a deferred power state, for defining the Boolean expression for the ***–logic_expr*** and (or) ***-supply_expr***, the voltage values, exact names of power supply nets, any other physical or technology dependent parameters, like power switches etc. are usually become available only during implementation (synthesis or post-synthesis level) of the design. The ***–simstate*** is also equally important for the deferred power state along with HDL signal expression, specifically for UPF strategy analysis until the implementation UPF became available.

3. Indefinite Power States An Indefinite power state is a named power state that is neither a definite power state nor a deferred power state.

Example 3.29 Example of Indefinite Power States

add_power_state PD_COREA ***-domain*** \
-state {UNDEFINED ***-logic_expr*** {PD_COREA != RUN && PD_COREA != SHD} }

Obviously, as observed in the indefinite power state in Example 3.29 above, the UNDEFINED state value specifying the power state for the Boolean expression ***-logic_expr*** {PD_COREA != RUN && PD_COREA != SHD} } will never satisfy to refine. As well it is also clear that indefinite power states are usually constructed with disjunction of expressions like, <object>!=<state>.

Thus it is also clear that the definite (and indefinite as well) power states corresponds to state refinements for higher levels and deferred power states for lower level of design abstraction. This is because at a higher level, initially objects remain undefined by default unless defined with definitive power states. However once defined, it allows the definite power states to refine by derivations or be defined through a complete new states or sub state, based on its original definition (actually through updating the expression in ***–logic_expr***).

The following three Examples 3.30, 3.31, and 3.32 explains the details of *refinement by derivation* and *refinement in place* which are applicable to define and model the "definite" and "deferred" power states respectively.

Example 3.30 Refinement by Derivatives: Applicable for Definite Power States

Definite power state Refinement for power domain
add_power_state PD_COREA ***-domain –update*** \
-state NEW_RUN {***-logic_expr*** \
{(***power*** == {*FULL_ON*, 1.1}) && (***ground*** == {*FULL_ON*, 0.0})}}

From Example 3.26, for the definite power states, the previous ***-state*** {RUN ***–logic_expr*** {***primary*** == ON}} now updated with the ***–state*** NEW_RUN {(…) && (…)} as shown in the Examples 3.30 above. This approach involves defining a new power state, with a new ***–state*** name and updated ***-logic_expr***, based on the original power state.

The refinement by derivation for the other example (i.e. # Definite Power State with Design Signals or Controls in Logic Expression) given for definite power states in Examples 3.26, based on *–**logic-expr*** with a logic or control signals is also explained below.

Example 3.31 Variant of Refinement by Derivatives: Applicable for Definite Power States

Definite Power State Refinement with Design Signals or Controls in Logic Expression

add_power_state PD_COREA***.primary*** ***–supply*** ***–update*** \

-state ***ON***.ON_STATE {***-logic_expr*** {(PD_COREA***.primary*** == RUN && MCTL_ON ==1'b1)}}

So it is also clear that the refinement by derivation preserves the original power state definition and thus avoids unexpected semantic changes in other commands that refer to that original power state. This kind of refinement results in an refinement of hierarchy in which more abstract states are refined to create more specific states. Hence the refinement by derivatives reinforce the capabilities of creation of overlapping or non-mutually exclusive, new and independent power states from a more abstract state.

Example 3.32 Refinement in Place: Applicable for Deferred Power States

add_power_state PD_COREA***.primary*** ***-supply*** ***-update***\

-state {RUN ***–logic_expr*** {nPWRUP_CON==1'b0}}

In contrast, the *refinement in place* is similar to how ***-update*** works for **add_power_state** or in the case of **create_power_domain** as specified by UPF LRM. But updating the power state through ***–update*** implies that it actually modifies the original definition rather than creating a new definition or a new power state.

Although it is obvious that deferred power states eventually evolve to definite power states in the course of the design verification and implementation flow. However, the evolution of deferred to definite power states is only true in perspective of the inherent meaning of the definite power state.

The physical entity of a deferred power state even after refinement remains different, because of two distinctive reasons. The first is, deferred to definite refinement does not create any new state for supply sets or power domains or other objects, and the second is, the boundary of mutual-exclusion conditions and transition between original and refined states are indistinct.

Actually such refinements impose additional pressure on PA static verification to perform Boolean analysis of expressions. In fact, the power state refinement methodology encourages the definition of mutually exclusive power states. The deployment of static or (and) dynamic verification tools can only ensure whether refinement of definite to definite and deferred to definite power states provides expected results. And it is also distinctive that the indefinite power states do not fall in the deferred or definite categories. Therefore indefinite states are never refinable and it is usually recommended to avoid them in UPF modelling for any design.

Now the obvious question may arise, If indefinite power states are not required or recommended, then why UPF LRM defines such a state? The ultimate answer

may converge on the fact that it is defined to avoid it. Primarily, indefinite states arise from state and object inequality (!=), negation (!), exclusion (||) operator in *-logic_expr*. Secondarily, it may refer to an indefinite state of a dependent or another object. Ultimately they do not provide the option of refinement. The artifacts can be explained through the following examples; considering that PD_CPUA0 initially defines RUN as an indefinite state, as shown below.

Example 3.33 Artifacts of Indefinite Power States

```
add_power_state -domain PD_CPUA0 \
-state {RUN -logic_expr {primary != OFF }}
```

A subtle point here is that the state, RUN, may become active when primary is not an OFF state, but probably in an ON state or BIAS state, etc. But such an expression is fine if they are not intended to be refined later. Typically, indefinite states are used when composing top-level power domain power states based on lower subdomain level power states due to its convenience, as shown in Example 3.34.

Example 3.34 Artefacts of Indefinite Power States

```
# Sub domain PD_CPUA0 power state
add_power_state -domain  PD_CPUA0
  -state {RUN -logic_expr {…}  \
  -state {STBY logic_expr {…}  \
  -state {OFF logic_expr {… }
# Sub domain PD_L2 power state
add_power_state –domain PD_L2
  -state {RUN logic_expr {…}   \
  -state {RET logic_expr {… }   \
  -state {OFF logic_expr {…}
# Top domain PD_COREA power state
add_power_state -domain PD_COREA
  -state {ON -logic_expr {  { PD_L2  !=OFF } || {PD_CPUA0 == RUN }}
```

The ON state for PD_COREA is active when PD_L2 is either in the RET or RUN state or when PD_CPUA0 is in the RUN state. It is clear that there are many overlapping possible states of PD_L2 and PD_CPUA0 for the ON state of PD_COREA. So, this is an indefinite power state because of operators (!=, ||) used in the *–logic-expr*.

Despite that, it is facilitating the coordination of state for hierarchical dependency; the user needs to be aware that it implies overlapping or non-mutually exclusive states. In most cases, it is possible to rephrase the expression of ON state to be in the definite power state form for PD_COREA. Alternatively it could be intended that all indefinite states are "don't care" while definite states are state to "care". For such cases, it may also be useful to mark the indefinite power states with the UNDEFINED state value defined by UPF 3.0 LRM as shown in Example 3.24.

It is further required to explore the flexibility and controllability of power states in order to comprehend efficient UPF modeling. The flexibility and controllability of the definite and deferred power states, other than the gifted feature of *incremental refinement*, comes in a different format and flavor. For example, the hierarchical

composition of power states of power domains can be specified in terms of power states of its supply sets. It is also required to fully comprehend the power states defining a mechanism for hierarchical power domains, specifically to grasp the complex communication and dependency in defining power states for such power domains.

The following Examples 3.35 and 3.36, explains the hierarchical composition of power states. These examples are also based on the Fig. 3.3 and Examples 3.27 and 3.28 shown in the earlier part of this section.

Example 3.35 Power States for Hierarchical Domains with Domain and Supply Set Handle

```
# Sub domain PD_CPUA0 power states through <object_name> power domain
# Definite power state for power domain
add_power_state PD_CPUA0 -domain \
-state {RUN –logic_expr {primary == ON} } \
-state {SHD –logic_expr {primary == OFF} }
# Sub domain PD_CPUA0 power states through supply set handle
# Deferred power states through supply set handle
add_power_state PD_CPUA0.primary -supply \
-state {RUN -simstate NORMAL} \
-state {SHD -simstate CORRUPT}
# Optional but used to update a new state in sub domain PD_CPUA0
add_power_state PD_CPUA0 –domain –update \
  -state {RET}
```

The above Example 3.35, explains the power state defining and update mechanisms of a sub-domain PD_CPUA0, through *–domain* as <object_name>. A variation of the definition is also shown with the supply set handle PD_CPUA0*.primary* through *–supply* options.

Example 3.36 Power States for Hierarchically Dependent Power Domains

```
# Sub domain PD_CPUA0 states dependency on PD_COREA power states
add_power_state PD_COREA -domain \
-state {RUN -logic_expr {primary==ON  && PD_CPUA0==RUN}} \
-state {SHD -logic_expr {primary==OFF && PD_CPUA0==SHD}} \
-state {RET -logic_expr {primary==OFF && PD_CPUA0==RET}}
# Top domain PD_SOC states dependency on Sub Domain PD_COREA power states
add_power_state PD_MPCore -domain \
-state {RUN -logic_expr {primary==ON  && PD_L2==RUN && PD_COREA==RUN}} \
-state {DMT -logic_expr {primary==OFF && PD_L2==RUN && PD_COREA==SHD}} \
-state {SHD -logic_expr {primary==OFF && PD_L2==SHD && PD_COREA==SHD}}
```

Example 3.36 shows the hierarchically higher power domain PD_COREA power state's dependency on a lower sub-domain PD_CPUA0 state {... PD_CPUA0==RUN}

as well the top power domain PD_SOC state's dependency {… PD_COREA==RUN} on sub-domain PD_COREA power states. This is another level of flexibility.

In an IP centric design flow, **add_power_state** provides a mechanism to easily leverage the power states of IP blocks or lower level blocks when defining the states of a power domain at a higher level. The IP provider may describe a few fundamental power states for their IP; however the SoC integrator may choose to further refine some of these states to better fit in with the SoC power modes or states in the hierarchical power state definition. Such power state refinement procedures are already explained through Examples 3.30, 3.31 and 3.32. Hence IP integration and refinement of its power states is also another form of flexibility.

The UPF 3.0 LRM also introduces a grouping mechanism which allows the user to do similar compositions at a convenient design hierarchy level for very large and complex SoCs. The UPF **create_power_state_group** command defines a group name that can be used in conjunction with the **add_power_state** command. A power state group is used to collect related power states defined by **add_power_state**. The ***-group*** (<group_name>) is defined in the current scope. The power state group defines the legal combinations of power states of other objects in this scope or in the descendant subtree. That is, it represents the combinations of states of those objects that can be active at the same time during operation of the design.

Example 3.37 Grouping of Legal Power States

```
  create_power_state_group PG_SOC
  add_power_state -group PG_SOC \
 -state {RUN -logic_expr {primary==ON  && PD_L2==RUN && PD_
COREA==RUN}} \
 -state {DMT -logic_expr {primary==OFF && PD_L2==RUN && PD_
COREA==SHD}} \
  -state {SHD -logic_expr {primary==OFF && PD_L2==SHD &&
PD_COREA==SHD}}
  -state {RET -logic_expr {primary==OFF && LP_RET1N == 1'b1 PD_COREA
== OFF}} \
 -state {OFF -logic_expr {primary==OFF && LP_RET1N == 1'b0 PD_COREA
== OFF}}
  add_power_state -group PG_SOC -update \
  -state "RET.PD_CPUA0_ACT -logic_expr {PD_COREA.PD_CPUA0.ACT}
-illegal" \
  -state "OFF.PD_CPUA0_ACT -logic_expr {PD_COREA.PD_CPUA0.ACT}
-illegal"
```

The power state also provides a mechanism to control the state space of power modes of design operation through explicitly or implicitly marking state legality. The legality of power states can be specified through ***–illegal*** options during ***–update*** or through ***–complete***. The ***-complete*** signifies that all undefined power states beyond are illegal. This is explained in the following examples.

Example 3.38 Legality of Power States

```
#Explicitly specifying illegal power states through -illegal
```

```
add_power_state PD_COREA-update \
-state {PD_CPUA0_RET_ONLY -illegal \
-logic_expr {primary == ON && PD_L2 == RUN && PD_CPUA0 == RET}}
# Implicitly specifying of illegal power states through -complete
add_power_state PD_COREA –update –complete
```

The inherent advantages of different fundamental power states may be best summarized as follows.

List 3.16 Advantages of Power State from Its Controllability and Flexibilities

- Allows modeling UPF; i.e., the power management architecture from a very early stage of design.
- Allows integrating design IP any time in the power management architecture.
- Allows analyzing and validating UPF strategy requirements.
- Allows computing accurate state transition coverage information through interdependent states.
- Prevents intermediate state transitions of definite and deferred power states during refinement.

UPF LRM also provides strict guidelines for defining power states which is not only for modelling UPF for a design but also for achieving efficient and expected results from verification tools.

List 3.17 Power State Definition Guidelines

1. Power states defined through *–supply_expr* for supply set or supply set handles:
   - Must refer to only supply ports, nets, and functions of the given supply set or supply set handle,
   - Must at least refer to one of the power or ground functions of the supply set,
   - Is not allowed to refer functions of another supply set or supply set handle.
2. Power states defined through *–logic_expr* for supply set or supply set handle:
   - Must refer to only logic ports, logic nets, interval functions,
   - Must be allowed to refer to power states of the given supply set or supply set handle,
   - Is not allowed to refer functions of another supply set or supply set handle,
   - Is not allowed to refer power states of another supply set or supply set handle,
   - Is not allowed to refer power states of a domain.
3. Power states defined through *–logic_expr* for a given power domain:
   - Must refer to only logic ports, logic nets, interval functions, power states of supply sets or supply set handles that are available in the domain, and power states of power domains.
   - Must refer to the power states of all supply sets of the domain that have more than one legal power state,
   - Is not allowed to refer supply ports, supply nets, or functions of a supply set or supply set handle.

4. The *-simstate* must be associated with a power state of a power domain or power states of other objects like composite domains, groups, modules, and instances.
5. When ***-simstate*** is specified as NOT_NORMAL it is the same as CORRUPT but in subsequent refinement it is allowed to refine to other state values except NORMAL.
6. It is not allowed to change the state values of *–simstate* during updating through the **add_power_state *–update*** command if the state values are previously specified from one of the lists of the state values shown in List 3.11.
7. The *-simstate* for a predefined power state **ON** is **NORMAL**.
8. The *-simstate* for predefined power states **OFF** and **ERROR** is **CORRUPT**.

The guidelines for defining power states are comprehensive for both UPF modelling and design verification automation methodologies. Although the above lists are curtailed for efficient UPF modeling purposes, however it is still extensive to accommodate with all detail examples and explanation provided in this section. Since the ultimate objective of power states are to denote states for power domains and supply sets, hence the obvious objectives and recommendations in defining power states can be best summarize as follows.

List 3.18 Recommendation for Defining Power State in UPF Modelling

– It is recommended to define "power state" of primary supply set in terms of logic expression of the design (HDL) objects,
– It is best to define power domain "power states" in terms of the states of its primary supply set handle and, if applicable, the states of any dependent power domains,
– It is also recommended to define power states such that they are definite in order to minimize unintended state overlap.

The power state semantics explained throughout this Sect. 1.3 with different fundamental, refinement, mutually exclusive, and non-mutually exclusive concepts and practices, including power states and dependency in hierarchical power domains, are demonstrated with real examples. The guidelines and recommendations further augment these examples to indicate numerous ways to model UPF efficiently.

It is evident from the discussion that the composition of power states for power domains and power supply sets typically constructed with followings:

List 3.19 Composition of Power States for Power Domain and Power Supply Set

– Supply set handles through *-supply*,
– Power domain with *-domain*,
– Boolean expression through *–logic_expr*, *–supply_expr,* or
– Their combination and design control signals, predefined values for simulation states through *–simstate* etc.

These constituent parts of power states actually govern the status of the power domains and (or) relevant power supplies, as well as signify whether the communicating source-sink power domain requires ISO, LS, ELS, RFF, RPT, as well as PSW etc. Specifically the power states indicate whether different power management MV

cells are required in the design. This is actually processed with the definition or specification of strategies and this was one of the reasons to discuss UPF power states before discussing UPF strategies in the next section.

3.1.4 UPF Power Strategies

Apart from the power domain, supply set, and power states that are regarded as elementary constituent parts for constructing and modelling complete UPF, there is one more significant part; the UPF strategies that include ISO, LS, ELS, RFF, PSW, and RPT etc. These are also required to complete the UPF construction even at the higher level of design abstractions at RTL. Although for pure RTL, these strategies and their controls are either place-holders (specifically for PSW) or virtual inferences by different dynamic and static verification tools (discussed in Chaps. 5, 6 and 7) (specifically for ISO, LS, ELS, and RFF etc.).

Isolation (ISO) strategies are fundamental and common for any low power or power aware design. The syntax of ISO strategy definition in UPF is shown in the following Example 3.39.

Example 3.39 Isolation Strategy Definition Syntax

set_isolation strategy_name -***domain*** domain_name
[-***elements*** element_list]
[-***exclude_elements*** exclude_list]
[-***source*** <source_domain_name | source_supply_ref >]
[-***sink*** <sink_domain_name | sink_supply_ref >]
[-***diff_supply_only*** [<TRUE | FALSE>]]
[-***use_equivalence*** [<TRUE | FALSE>]]
[-***applies_to*** <inputs | outputs | both>]
[-***applies_to_boundary*** <lower | upper | both>]
[-***applies_to_clamp*** <0 | 1 | any | Z | latch | value>]
[-***applies_to_sink_off_clamp*** <0 | 1 | any | Z | latch | value>]
[-***applies_to_source_off_clamp*** <0 | 1 | any | Z | latch | value>]
[-***no_isolation***]
[-***force_isolation***]
[-***location*** <***self*** | ***other*** | ***parent*** | ***fanou***t>]
[-***clamp_value*** <0 | 1 | Z | latch | value | {<0 | 1 | Z | latch | value>*}>]
[-***isolation_signal*** signal_list [-isolation_sense <high | low | {<high | low>*}>]]
[-***isolation_supply*** supply_set_list]
[-***name_prefix*** pattern]
[-***name_suffix*** pattern]
[-***instance*** {{instance_name port_name}*}]
[-***update***]

As can be observed, isolation has abundant options in the syntax; hence the most obvious and crucial items are explained that are most frequently used for efficient UPF modeling. The –***domain*** name specifies the power domain for which the ISO strategy is defined.

The ***-exclude_elements*** <exclude_list> for ISO resemble the definition of the ***-exclude_elements*** command and the resultant <effective_element_list> for the power domain explained in Sect. 1.

The ***-exclude_elements*** explicitly identify a set of rooted names of instances or ports of the domain to which this strategy does not apply. The <effective_element_list> will exclude all instances or ports either specified in base commands or through ***–update***. Although the ISO strategy also has a similar ***-no_isolation*** option that functions exactly the same as the ***–exclude_elements***, but with the added benefit of not specifying a list of instances or ports in customizable way. This filter is rather useful when the entire list of elements needs to exclude from imposing isolation strategies.

In addition, there are several other filters to customize or restrict the elements or set of ports from applying isolation strategies in **set_isolation** commands. These are listed below.

List 3.20 Additional Filtering Options for ISO Strategies

-source,
-sink,
-diff_supply_only,
-applies_to,
-applies_to_clamp,
-applies_to_sink_off_clamp, and
-applies_to_source_off_clamp

The ***–source*** or ***–sink*** specifies the name of a supply set or power domain. When a domain name is specified, it implicitly represents the primary supply of that domain. However, these filters requires to have a driver and a receiver for ports respectively to satisfy the filtering criterion. The following examples explain the ***–source*** and ***–sink*** usage in the isolation strategy definition.

The following Example 3.40 shows a complete but typical ISO strategy. ISOs are usually special types of AND gates, OR gates, or Latches with control signals, sitting between source and sink power domains to clamp the output signals to a predefined value destined to the sink. This is done in order to avoid 1`bx or unknown value propagation when the source power domain is in an OFF state.

Example 3.40 Typical Isolation Strategy

```
set_isolation iso_1 \
    -domain PD_sub3 \
    -isolation_supply_set PD_sub3.isolate_ss \
    -clamp_value 1 \
    -elements {mc0/ceb mc0/web} \
    -isolation_signal mc_iso \
    -isolation_sense high
```

In the above ISO definition the ***–clamp_value*** 1 signifies an OR type of ISO that clamps all signals to destination sink power domain to 1'b1. The other option ***-isolation_sense*** *high* signifies the controllability of an ISO enable signal to assert ISO when it's high or 1`b1.

Also to note that although the semantic is the same, but the ISO supply set to provide powers to ISO cells are defined in UPF 2.1 LRM as ***-isolation_supply_set*** <supply_set_list>, whereas it is ***-isolation_supply*** <supply_set_list> in UPF 3.0.

The ***-location*** <***self*** | ***other*** | ***parent*** | ***fanout***> options in Example 3.39 determines the location domain in the logic hierarchy where the isolation cell will be inserted or inferred. Here ***self*** indicates the isolation cell will be placed inside the self-domain, i.e., the domain whose port is being isolated. This is the default option. The ***parent*** indicates the isolation cell will be placed in the parent domain of the port being isolated. UPF LRM restricts that the isolated port in this case cannot be a port of a design top module, or port of a lower boundary. When the ***other*** option is used, the isolation cell will be placed in the parent domain for an upper boundary port, and in the child domain for a lower boundary port. Similarly the UPF LRM restricts that this isolated port cannot be an upper boundary port of a design top module, or a lower boundary port of a leaf cell.

The ***fanout*** option is used when it is required to place the isolation cell in each fanout domain boundary that is closest to the receiving logic. However, if the receiving logic is in a macro cell instance, the isolation cell will be inferred at the input port of that macro cell instance, on the lower boundary of the location domain; otherwise the isolation cell will be inserted at the location domain port that is driven by the port to which the strategy applies. It is also required to know when ***-location fanout*** is not specified, but ***-sink*** <domain_name> is specified, then the sink domain determines whether the isolation cell is inserted at an input port or an output port of the location domain. However, if neither ***-location fanout*** nor ***-sink*** <domain_name> are specified, then the isolation cell is inserted at the port of the location domain that is the port to which the strategy applies.

For simplification, the ISO strategy is explained in conjunction with several power domains with single and heterogeneous fanout connections, as shown in the Fig. 3.4 below.

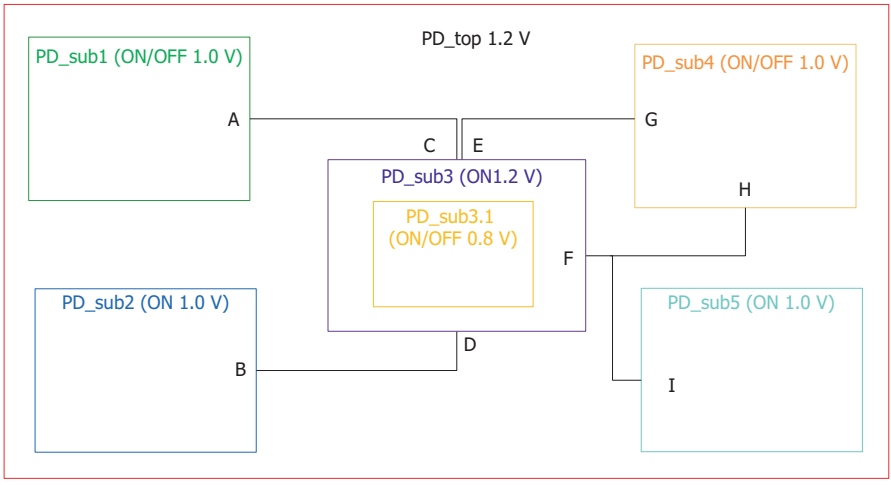

Fig. 3.4 Explaining different ISO strategy and filtering mechanism

Example 3.41 Isolation Strategy with -*source* and –*sink Filtering Options*

```
# UPF Code snippet to define power domains of Fig. 3.4 and relevant Supplies
create_supply_set PD_sub1_SS ...
    create_power_domain PD_sub1 ...
    associate_supply_set PD_sub1_SS -handle PD_sub1.primary
    create_supply_set PD_sub2_SS ...
    create_power_domain PD_sub2 ...
    associate_supply_set PD_sub2_SS -handle PD2.primary
    create_power_domain PD_sub3 ...
    # - source Examples for isolation cell at Port C and isolate Path A-C
    set_isolation iso1 -domain PD_sub3 -source PD_sub1_SS -location parent
...
    # -source Example for isolation cell at Port B and isolate Path B-D.
    set_isolation iso2 -domain PD_sub3 -source PD_sub2_SS -location self ...
    ## -sink Examples for isolation cells at Port G and Port H and isolate Path E-G
and F-H.
    set_isolation iso1 -domain PD_sub3 -sink PD_sub4_SS -location fanout ...
```

The *–diff_supply_only* is another influential and convenient filtering option for an isolation strategy applicable to heterogeneous fan-out ports between the source and the sink power domains. This option indicates to apply an isolation strategy with different power supply between the driving logic and receiving logic ports or for which either the driving or receiving, or both, supply sets cannot be determined. The following example explains the usage of this option in conjunction to *–source* and *–sink* options.

Example 3.42 Isolation Strategy with –*source, –sink* and *–diff_supply_only* Filtering *Options*

```
# UPF Code snippet to define power domains of Fig. 3.4 and relevant Supplies
create_supply_set PD_sub2_SS ...
    create_power_domain PD_sub2 ...
    associate_supply_set PD_sub2_SS -handle PD_sub2.primary
    associate_supply_set PD_sub2_SS -handle PD_sub3.primary
    associate_supply_set PD_sub2_SS -handle PD_sub4.primary
    ## -diff_supply_only Example for ISO at Ports C and F and isolate Paths A-C
and F-I.
    ## But do not isolate path F-H.
    set_isolation iso1 -domain PD_sub3 -applies_to both \
        -diff_supply_only TRUE -location parent ...
    ## -diff_supply_only Example for ISO at Ports A and I and isolate Paths A-C
and F-I.
    set_isolation iso2 -domain PD_sub3 -applies_to both \
        -diff_supply_only TRUE -location self ...
    ## -diff_supply_only Example for ISO at Port A and isolate Path A-C.
    set_isolation iso3 -domain PD_sub3 -applies_to both -source PD_sub1_SS
    -diff_supply_only TRUE -location other ...
```

The *-diff_supply_only* TRUE filter will be satisfied only if the driver supply and receiver supply of the port are neither identical nor equivalent. Otherwise, the *-source* and *-sink* filters will match the named supply set or any supply set that is equivalent to the named supply set. To note, here the equivalency is determined through the *-use_equivalence* option as listed in Example 3.39. This specifies whether supply set equivalence is to be considered in determining when two supply sets match. If *-use_equivalence* is specified with the False, the *-source* and *-sink* filters usually match only the named supply set.

The LS or ELS are required between two communicating power domain when both the source and sink are in an On state but have different supply voltages. LS or ELS ensures accurate resolutions of signal values propagating between a source and sink by maintaining either a high-to-low or low-to-high voltage conversion mechanism, as required.

Example 3.43 Level-Shifter Strategy Definition Syntax

set_level_shifter strategy_name
-domain domain_name
[*-elements* element_list]
[*-exclude_elements* exclude_list]
[*-source* <source_domain_name | source_supply_ref>]
[*-sink* <sink_domain_name | sink_supply_ref>]
[*-use_equivalence* [<TRUE | FALSE>]]
[*-applies_to* <inputs | outputs | both>]
[*-applies_to_boundary* <lower | upper | both>]
[*-rule* <low_to_high | high_to_low | both>]
[*-threshold* <value>]
[*-no_shift*] [-force_shift]
[*-location* <self | other | parent | fanout>]
[*-input_supply* *supply_set_ref*]
[*-output_supply* *supply_set_ref*]
[*-internal_supply* *supply_set_ref*]
[*-name_prefix* pattern] [*-name_suffix* pattern]
[*-instance* {{instance_name port_name}*}]
[*-update*]

Although the ISO and LS or ELS strategies are completely different in their functionalities, many syntactical options like *–elements*, *-exclude_elements*, *-source*, *-sink* etc. are used for the same purpose in both strategies. Hence instead of discussing common options, the explanation of LS semantics will focus on LS specific options. However, most of them, like *-applies_to_boundary*, *-rule*, and *–threshold*, serve as filters that restrict the set of ports to which a given **set_level_shifter** command applies.

For example, *-rule* restricts the strategy to apply only to ports that require a given level-shifting direction, with the default being both. The *-threshold* <value> is satisfied by any port for which the magnitude of the difference between the driver and

receiver supply voltages can exceed a specified threshold value. The nominal power and ground of the port's driver supply are compared with the nominal power and ground of the port's receiver supply to determine if level-shifting is required.

The variation ranges of respective power and ground supplies are also considered. This option requires tools to use information defined in power states of the supplies involved in a given interconnection between objects with different supplies. If ***-threshold*** is not specified, it defaults to 0, which means that a level-shifter will be inserted for a given port if there is any voltage difference.

The ***-input_supply, -output_supply, and -internal_supply*** options in defining LS strategies are also predefined names in the LS strategy scope. These supply sets are used to power the input portion, output portion, and internal circuitry of the LS. This can be best pictured through the following Fig. 3.5.

Here VDDI and VDDO correspond to input and output supplies; whereas for pin A, the related power and ground VDDI and VSS are the internal supply. The ***-internal_supply*** specifies the supply set that will be used to provide power to supply ports that are not related to the inputs or outputs of the level-shifter. However, more detail on LS is discussed in Chap. 4.

The default for ***-input_supply*** is the supply of the logic driving the level-shifter input. The default is used if and only if that supply set is available in the domain in which the level-shifter will be located. The default for ***-output_supply*** is the supply of the logic receiving the level-shifter output. The default is used if and only if that supply set is available in the domain in which the level-shifter will be located.

However, there is no default supply set defined for ***-internal_supply***. Default input and output supply set definitions apply only if exactly one level-shifter strategy applies to a given port, all drivers of that port have equivalent supplies, and all receivers of that port have equivalent supplies. For more complex cases, the required supply sets should be explicitly specified. If the level-shifter strategy is mapped to a library cell that requires only a single supply, then explicit specification of an input supply set is not required. As well any explicit input supply set specification is ignored, and the default input supply set does not apply; only the output supply set is used.

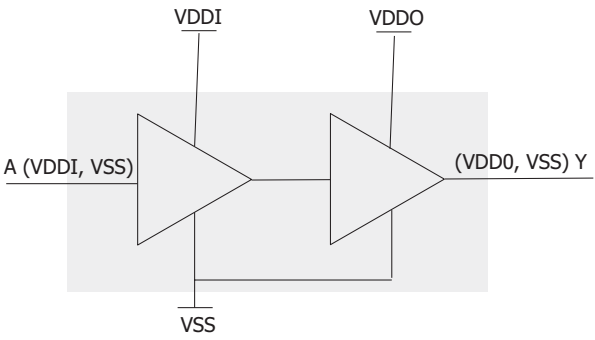

Fig. 3.5 Physical construction of level-shifters

Example 3.44 Typical Level-Shifter Strategy

```
# UPF Code snippet to update relevant Supplies from Example 3.41   create_supply_set PDV_sub1.primary -update \
            -function { power V_sub1 } \
            -function { nwell V_sub1N }
    …….
    create_supply_set PD_top.primary -update \
        -function {power V_top } \
        -function { nwell V_topN}
    ……..
    # LS Example for paths from PD_top (ON 1.2V) to PD_sub1 (ON/OFF 1.0V)
    set_level_shifter ls_1 \
        -domain PD_top \
        -source PD_top \
        -sink PD_sub1 \
        -location self \
        -input_supply PD_top.primary \
        -output_supply PD_sub1.primary
```

The LS definition as it appears is very straightforward. Here ***–source***, ***–sink,*** and ***–location*** identify the inferred location at the power domain boundary on an HDL hierarchical port or path instances where the LS will be actually inserted in the GL-netlist or later in the DVIF. It is important to note that there are no dedicated UPF commands and options to denote ELS, it comprises combining one ISO and one LS command, where the control of ISO becomes the enable signal of ELS. It is also required to note that although the semantics are same but the syntax for input or output or internal supply is different in UPF LRM 2.1 and contains "***_set***" as follows; ***-input_supply_set, -output_supply_set, -internal_supply_set***.

In UPF, RFF strategies are defined as a set of sequential elements that need to retain the state or data. It also identifies the save and restore control and condition behavior for the retention registers. When the RFF definition specifies instance or process, all registers within are inferred to retention registers. Also when the reg, signal, and variables infers sequential elements, they are also subject to attain the UPF retention strategies; i.e. they are inferred to retention registers.

The following example shows the retention syntax.

Example 3.45 Retention Strategy Definition Syntax

```
set_retention retention_name
-domain domain_name
[-elements element_list]
[-exclude_elements exclude_list]
[-retention_supply ret_supply_set]
[-no_retention]
[-save_signal {logic_net <high | low | posedge | negedge>}]
[-restore_signal {logic_net <high | low | posedge | negedge>}]
[-save_condition {boolean_expression}]
[-restore_condition {boolean_expression}]
```

[-***retention_condition*** {boolean_expression}]
[-***use_retention_as_primary***]
[-***parameters*** {<RET_SUP_COR | NO_RET_SUP_COR |SAV_RES_COR | NO_SAV_RES_COR> *}]
[-***instance*** {{instance_name [signal_name]}*}]
[-***update***]

Similar to other strategies, -***domain***, -***elements***, and -***exclude_elements*** etc. options possess similar functionalities, hence they are not discussed here. Instead, the discussion is focused on a retention specific semantic explanation. For example, when –***elements*** is used with –***update***, it imperatively indicates additive design elements to the existing set of HDL elements or signals.

The -***save_signal*** and -***restore_signal*** options specify a rooted name of a logic net or port and its active level or edge. The -***save_condition*** and -***restore_condition*** specifies a Boolean expression. When –***save_signal*** and –***restore_signal*** are specified, the -***save_condition*** or (and) -***restore_condition*** becomes "don't care."

The -***retention_condition*** defines the retention behavior of the retention element. If the -***retention_condition*** is specified, it must evaluate to TRUE for the value of the state element to be retained. If the retention condition evaluates to FALSE and the primary supply is not NORMAL, the retained value of the state element is corrupted. The receiving supply of any pin listed in the -***retention_condition*** will be at least "On" as the retention supply of the retention strategy.

The -***retention_supply*** (or -***retention_supply_set*** in UPF LRM 2.1) specifies the supply set that will be used to power the state element holding the retained value. It also holds the values of any available control logic that evaluates the -***save_condition, -restore_condition***, and -***retention_condition***. The supply set specified by -***retention_supply*** is implicitly connected to the retention logic inferred by **set_retention** command.

The following example provides a case of a typical RFF strategy.

Example 3.46 Typical RFF Strategy

```
set_retention  ret_1 \
   -domain PD_sub1 \
   -retention_supply PD_sub1.retain_ss \
   -retention_condition {(mc0/clk == 1'b0)} \
   -elements {mem_ctrl_ret_list} \
   -save_signal {mc_save high} \
   -restore_signal {mc_restore low}
```

The **set_retention** UPF command can be extended and combined with another UPF command, **set_retention_elements,** which creates a named list of elements whose collective states are maintained physically from the list of elements used in **set_retention** or **map_retention_cell** commands, if retention is applied to any of the elements in the list.

Hence this command actually captures the <elements_list> of –***elements*** { mem_ctrl_ret_list } as shown in Example 3.46 and can be specified in the **set_retention_elements** as shown below.

Example 3.47 Sample of RFF Collective Elements

set_retention_elements mem_ctrl_ret_list \
-elements { mc0/present_state mc0/addr}

This actually creates a named list of elements whose collective states are maintained coherently when it is used with a **set_retention** command and ***–elements*** {elements_list} as shown in the above examples.

Although the syntax of **set_retention_elements** have additional options as shown below, but their significance and functionalities resembles with the options of other strategies and are already explained in different strategies above.

Example 3.48 Syntax for RFF Collective Elements

set_retention_elements retention_list_name
-elements element_list
[***-applies_to*** <required | not_optional | not_required | optional>]
[***-exclude_elements*** exclude_list]
[***-retention_purpose*** <required | optional>]
[***-transitive*** [<TRUE | FALSE>]]

At this point it is required to explain that, depending on how the retained value is stored and retrieved, there are at least two different flavors of retention registers or latches or flip-flop, as follows.

List 3.21 Different Types of Retention Flops (RFF)

– Balloon-style retention
– Master or slave-alive retention

In a "balloon" type retention register, the retained value is held in a shadow or additional latch, often called the balloon latch and usually powered by the ***–retention_supply***. In this case, the balloon element is not in the functional data-path of the register. On the other hand, for a master or slave-alive retention register, the retained value is held in the master or slave latch and also powered by the ***–retention_supply***. In this case, the retention element is in the functional data-path of the register.

A "balloon" type retention register typically has additional controls to transfer data from a storage element to the balloon latch, also called the save stage, and transfer data from the balloon latch to the storage element, also called the restore stage. The ports to control the save and restore pins of the balloon-style retention register need to be available in the design to describe and implement this type of retention registers.

A "master or slave-alive" type retention register typically does not have additional save or restore controls as the storage element is the same as the retention element. Additional control(s) on the register may park the register into a quiescent state and protect some of the internal circuitry during power-down state, and thus the retention state is maintained. The restore in such registers typically happens upon power-up, again owing to the storage element being the same as the retention element. Thus, this style of registers may not specify save and (or) restore signals, but may specify a retention condition that could take the register in and out of retention state.

Following are few examples of these two different type of RFF strategies.

Example 3.49 Separate Save and Restore Pin Type for Balloon RFF

```
set_retention my_ret \
  -save_signal {save high} \
  -restore_signal {restore low} \
...
```

Here the RFF strategy specifies explicitly save and restore pin, which performs the save and (or) the restore functionalities.

Example 3.50 Single Retention Control (Save and Restore) Pin Type for Balloon RFF

```
set_retention my_ret \
  -save_signal {ret posedge} \
  -restore_signal {ret negedge} \
  -retention_condition {ret} \
...
```

For this example, the RFF strategy specifies a single pin that performs both the save and the restore functions. However, the retention pin is required to keep a certain value (0 or 1) to retain and remain in a retention state.

Example 3.51 Single Retention Control Pin Type for Slave-Alive RFF

```
set_retention my_ret \
  -retention_condition {ret} \
...
```

In this case as well, RFF specifies only a single retention control pin, but there is no explicit save and (or) restore is involved. Hence the retention condition has to be true during retention mode. It is also mandatory in this case that the slave latch (or storage element or output) is powered by the *–retention_supply*. In addition, master or slave-alive RFF requires the clocks and asynchronous resets to be related to the retention condition and retain a certain value (0 or 1 for example) during retention mode. Further, it also requires the *-use_retention_as_primary* to be specified, as the output is expected to be powered by the retention supply. To note, the *-use_retention_as_primary* option specifies that the storage element and its output are powered by the retention supply.

These extended features are shown in Example 3.52 below.

Example 3.52 Single Retention Control Pin Type for Slave-Alive RFF with Clock and Reset

```
set_retention my_ret \
  -retention_condition {!clock && nreset} \
  -retention_supply PD_sub2.primary \
  -use_retention_as_primary \
  ...
```

The power switch (PSW) strategy defines an abstract model of a single power switch which may involve additional detail and multiple, distributed power switch chains during implementation. PSW is usually considered as a place holder during

RTL and as well during GL-netlist design verification procedures. Although physical PSW are usually available after P&R, however, they play significant roles in verification from a very early stage of design abstraction. Hence the PSW strategy is also very crucial for UPF modeling. UPF PSW syntax is shown in Example 3.53.

Example 3.53 Syntax of PSW

create_power_switch switch_name
[-***switch_type*** <fine_grain | coarse_grain | both>]
[-***output_supply_port*** {port_name [supply_net_name]}]
{-***input_supply_port*** {port_name [supply_net_name]}}*
{-***control_port*** {port_name [net_name]}}*
{-***on_state*** {state_name input_supply_ port {boolean_expression}}}*
[-***off_state*** {state_name {boolean_expression}}]*
[-***supply_set*** supply_set_ref]
[-***on_partial_state*** {state_name input_supply_ port {boolean_expression}}]*
[-***ack_port*** {port_name net_name [boolean_expression]}]*
[-***ack_delay*** {port_name delay}]*
[-***error_state*** {state_name {boolean_expression}}]*
[-***domain*** domain_name]
[-***instances*** instance_list]
[-***update***]

The –***switch_type,*** which consist of ***coarse_grain*** or ***fine_grain*** or both, is actually determined by the Liberty or UPF attributes denoted by "`switch_cell_type`" and "**UPF_switch_cell_type**" respectively. The ***coarse_grain*** is default.

The ***-intput_supply_port***, ***-output_supply_port***, ***control_port*** may also be used with a placeholder port and (or) net name. However PSW port names and port state names are defined in the scope of the switch instance and, therefore, they can be referenced with a hierarchical name in the same way that any other instance port can be referenced. The feature is explained in conjunction with the following Example 3.54.

Example 3.54 PSW Ports and Port States Name Referencing

create_power_switch PS1
-output_supply_port {outp}
-input_supply_port {inp}
...

The PSW Example 3.54 creates an instance PS1 in the current scope and creates supply ports "outp" and "inp" within the PS1 instance. The switch supply ports can then be referred as "PS1/inp" and "PS1/outp". The abstract PSW model allows to denote one or multiple input supply ports but only one output supply port.

Output supply port is specified only when the switch type is ***coarse_grain*** or ***both***. Each input supply port is effectively gated by one or more control expression defined by ***-on_state*** or ***-on_ partial_state*** expressions.

The ***-on_state*** expression specifies when a given input supply contributes to the output without limiting the supply of (electric) current. However, the ***-on_partial_state***

expression specifies when a given input supply contributes to the output in a limited (electric) current manner. Each input supply may have multiple ***-on_state*** and (or) multiple ***-on_partial_state*** expressions.

The PSW model may also have one or more ***-error_state*** expression. Any ***-error_state*** expressions defined for a given power switch represent the controls of the input conditions that are illegal for that switch. The PSW may also have a single ***-off_state*** expression. The ***off_state*** expression represents the condition under which no ***-on_state*** or ***-on_partial_state*** expression is TRUE. If it is not specified explicitly, the ***-off_state*** expression defaults to the complement of the disjunction of all the ***-on_state***, ***-on_partial_state***, and ***-error_state*** expressions defined for the power switch.

At any given time, the ***–on_state*** or the ***–on_ partial_state*** contributes a value to the output port of the PSW. The UNDETERMINED status comes into existence only when the ON or partial ON state Boolean expression for a given input supply port, which is not in OFF state, refers to an object with an unknown (X or Z) value. Then the contributed value at the output of PSW remains {UNDETERMINED, unspecified}. However, in the power aware simulators or PA-SIM, the UNDETERMINED states are interpreted as ERROR states. Hence the PSW itself also has the state values, as shown List 3.22.

The following list shows the state values of PSW and its all controls and Ack ports.

List 3.22 State Values of Power Switch, Control and Acknowledge Ports

– FULL ON (or ON) state,
– OFF state,
– Partial ON state and
– UNDETERMINED state.

In addition, if an ***on_partial_state*** Boolean expression for a given input supply port evaluates to True, then the contributed value is the degraded value of that input supply port.

The ***–ack_port*** specifies acknowledgment of a PSW state transition. When an argument is specified, an acknowledged value is driven onto the specified <port_name delay> time units after the switch output transitions to an FULL_ON state. The inverse acknowledges value is driven onto the specified <port_name delay> time units after the switch output transitions to an OFF state.

If the supply set of the PSW is in NORMAL simstate, then the acknowledge value is a logic 0 or logic 1. If a <logic_value> is specified for ***-ack_port***, that logic value will be used as the acknowledge value for a transition to FULL_ON, and its negation is used as the acknowledge value for a transition to OFF; otherwise the acknowledge value defaults to logic 1 for a transition to FULL_ON and logic 0 for a transition to OFF.

If ***-ack_delay*** is specified, the delay may be specified as a time unit, or it may be specified as a natural integer, in which case the time unit shall be the same as the simulation precision; otherwise, the delay defaults to 0.

The typical PSW example is shown below in Example 3.54.

Example 3.55 Typical PSW Strategy

create_power_switch top_sw \
  -***domain*** PD_top \
  -***output_supply_port*** {vout_p 0d81_sw_ss.***power***} \
  -***input_supply_port*** {vin_p 0d81_ss.***power***} \
  -***control_port*** {ctrl_p mc_pwr} \
  -***on_state*** {normal_working vin_p {ctrl_p}} \
  -***off_state*** {off_state {!ctrl_p}}

This is a header switch as it is constructed on the power supply side of supply set 0d81_sw_ss. In contrast, the ground side of a supply set PSW are known as a footer. Nevertheless, this definition of PSW is an abstract placeholder that usually will be physically implemented at or after P&R in the DVIF. As well it is mentioned already that the actual implementation of PSW varies greatly depending on switching granularity requirements at the final design circuitry, and often switches are cascaded in a daisy chain to avoid spurious current flow causing from On and Off states of power supply.

The UPF repeaters (RPT) define a strategy for inserting repeaters or feedthrough buffers for ports on the interface of a power domain. The RPT are placed within the domain, driven by input ports of the domain, and drive output ports of the domain. Usually feedthrough buffers are required when a power domain sits between a source and sink power domain and it is required to feedthrough some signal through this intermediate power domain from source to sink. The following example shows the syntax of RPT.

Example 3.56 Syntax of RPT

set_repeater strategy_name
-**domain** domain_name
[-***elements*** element_list]
[-***exclude_elements*** exclude_list]
[-***source*** <source_domain_name | source_supply_ref >]
[-***sink*** <sink_domain_name | sink_supply_ref >]
[-***use_equivalence*** [<TRUE | FALSE>]]
[-***applies_to*** <inputs | outputs | both>]
[-***applies_to_boundary*** <lower | upper | both>]
[-***repeater_supply*** supply_set_ref]
[-***name_prefix*** string]
[-***name_suffix*** string]
[-***instance*** {{instance_name port_name}*}]
[-***update***]

Again there are obviously common options for RPTs as well, which resembles to the options of other strategies at least in terms of functionalities already discussed so far. Hence this discussion focuses on RPT specific options, for example ***–repeater_supply*** (or ***repreater_supply_set*** for UPF LRM 2.1).

The ***-repeater_supply*** is implicitly connected to the primary or backup supply ports of the buffer cell. If this supply is not specified and the power domain contains

the driver of the RPT cell, then the primary supply set of the domain is used as the default supply. Hence it is mandatory to specify the ***-repeater_supply*** when the default supply is not available in the domain.

The following example shows a typical RPT strategy.

Example 3.57 Typical RPT Strategy

set_repeater feedthrough_buffer1
-domain PD_sub3.1 ***-applies_to*** outputs

Here Example 3.57 defines a RPT cell to infer at the output of the domain boundary of PD_sub3.1. RPT strategies are comparatively new hence must be defined with caution.

UPF power strategies are essential parts of UPF modeling. There may be a situation where multiple strategies are required to apply to the same ports or HDL instances; hence it is important to understand the order of precedence in which these strategies are actually applied. For example, if multiple **set_isolation**, **set_level_shifter,** or **set_repeater** commands are potentially applied to the same port, the following criteria determine the relative precedence of the commands, and only the command(s) with the highest precedence will actually apply.

List 3.23 Criteria to Determine the Relative Precedence from Highest to Lowest

– Command that applies to part of a multi-bit port specified explicitly by name
– Command that applies to a whole port specified explicitly by name
– Command that applies to all ports of an instance specified explicitly by name
– Command that applies to a port of a specified power domain with a given sink and source
– Command that applies to a port of a specified power domain with a given sink or source
– Command that applies to all ports of a specified power domain with a given direction
– Command that applies to all ports of a specified power domain

If multiple strategies of the same type have the same highest precedence, then all of those commands actually apply to the port or part thereof, to the extent allowed by the strategy.

Similarly the ***–name_prefix*** or ***–name_suffix*** options for ISO, LS, and RPT used to create names for inserted isolation, level-shifter, and repeater cells, respectively, also have order of precedence over another UPF command **name_format**.

The **name_format** is used to define the format for constructing names for implicitly created objects like isolation, level-shifter, and their supply net or ports prefix or suffix etc. This is further discussed in Chap. 5.

During the course of discussion throughout the previous sections of this Chapter, it is visible that the UPF modeling is mainly based on power domain, power domain boundary, power supply, power supply network, power states, fundamental power states, and UPF power strategies. And these are the as rudimentary parts for UPF constructions. It is prevailed from these discussions that there are several other sub-

tle concepts inherent in UPF semantics that also drives the construction and modeling of UPF. For example, refinement of power domain elements, refinement of UPF strategies, refinement of power states through derivatives or through in place, power state overlap, inter-domain dependency etc. These features mainly govern how UPF evolves through the DVIF at different design abstraction levels through refinement and contributes in PA design integrations, implementations, and verifications perspective. The succeeding sections will discuss these features and flows in detail.

3.2 Successively Refinable UPF

The successive refinement of UPF corresponds power management and reduction throughout the DVIF where UPF development starts from early RTL and gradually refined down the implementation flow at GL-netlist and PG-netlist, augmented with appropriate detail for each design-implementation phase.

As mentioned earlier in introductory sections that IEEE 1801 standardization provides the flexibility to develop UPF from RTL with the perspective to adopt appropriate power reduction techniques at very early stage of design abstraction. This enable to comprehend the primary UPF constraints including rudimentary parts, like power domain with minimum extent of design instances as elements (also known as atomic power domain), abstract supply sets, and possible power states. In addition it also comprehend to virtually infer boundary strategies, like ISO, LS, ELS, RFF, PSW, RPT, etc. that in turn allows to simulate or verify power management and reduction techniques and pre-assess many post synthesis or post P&R power implementation artifacts. However logical ports and specific control information are not in the scope of constraint UPF.

In addition, many internally developed RTL design blocks and externally procured IPs are reused over few generation of a design, hence it is also important to enumerate any power management information of reusable IP at RTL. One of the major concern here that these IPs accompany technology or implementation independent UPF. Hence successive refinement of UPF allows IP provider to deliver UPF constraints of IP component along with the RTL functional specification that ensures the component will interact correctly in any given application with the existing power management and reduction techniques for the entire system. Basically the contents of UPF constraints for an IP are exactly same as the regular portion of the design as explained above.

The successive refinement of UPF complete flow is summarized in the following Fig. 3.6 represented from IEEE 1801-2015 Language Reference Manual (LRM).

Recalling the Fig. 3.3 Example of Complex SoC, the following Example 3.58 shows the sample of constraint UPF.

Example 3.58 Sample Constraint UPF

create_power_domain PD_SOC –***elements*** {.} -***atomic***
create_power_domain PD_COREA –***elements*** {u_ca_cpu0} -***atomic***

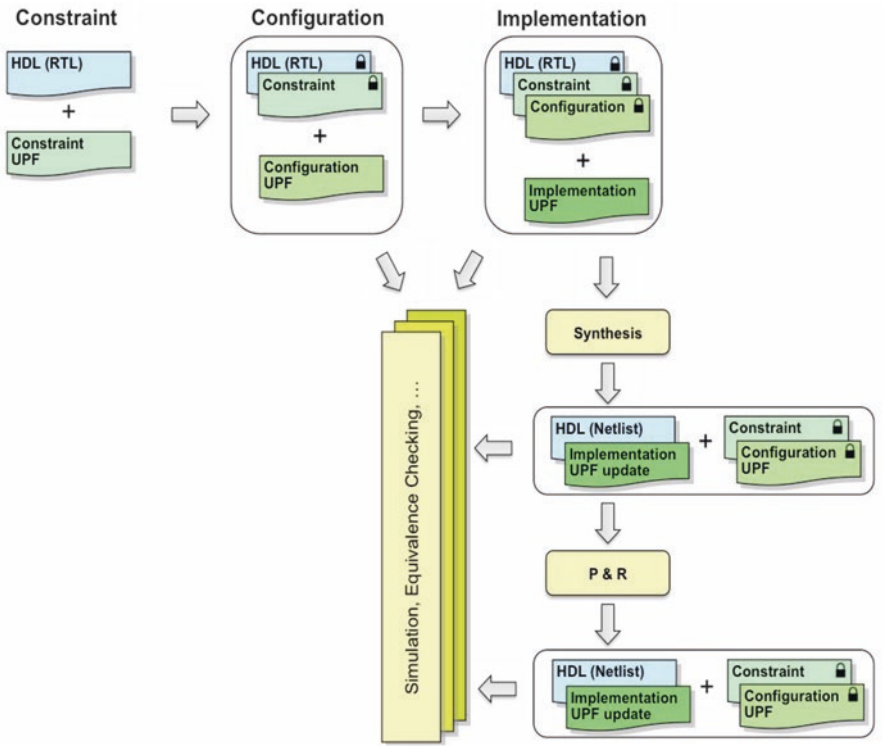

Fig. 3.6 Successive refinement of power management in the DVIF with successively adding constraint, configuration and implementation UPF (Courtesy: IEEE 1801-2015 LRM)

```
create_power_domain PD_CPU0 –elements {u_ca_advcpu0} –atomic
# Create the L2 Cache domain
 create_power_domain PD_L2 –elements {u_ca_l2/u_ca_l2_datarams u_ca_
l2/u_ca_l2_tagrams u_ca_l2/u_cascu_l1d_tagrams} –atomic
# RFF
set_retention_elements PD_COREA_RETN –elements “u_ca_cpu0”
# ISO
set_port_attributes –elements {u_ca_hierarchy}
  -applies_to outputs
  -clamp_value 0
set_port_attributes -ports u_ca_hierarchy/output_port_a
-clamp_value 1
# Power States
add_power_state PD_CPU0 –domain \
-state {RUN -logic_expr {primary == ON}} \
-state {SHD -logic_expr {primary == OFF}}
add_power_state PD_CPU0.primary –supply \
```

*-**state*** {ON -*simstate NORMAL*} \
*-**state*** {OFF -*simstate CORRUPT*}

When the design development further progresses in the flow and the design building blocks or the reusable IPs are in the integration process with the entire design, it is required to specify the configuration of the design and UPF as well. The configuration UPF which is an extension of existing constraint UPF, basically works in combination and adds the missing portion of constraint UPF. Which are logic and power management control ports and detail of strategies like ISO on crossing different power domain boundaries, RFF for the entire design or only a portion of the regular flops etc.

Another important aspect of configuration is the power states are updated with logic expression to accommodate the control inputs. However since the technology specific implementation detail is still not available at system integration of DVIF phase; hence configuration UPF is limited to only the attributes mentioned so far and typically do not consider voltage values, logical reference of PSW or any specific cell information.

The following example denotes the configuration UPF for the SoC design shown in Fig. 3.3 and works in combination with the constraint UPF is shown in Example 3.58.

Example 3.59 Sample Configuration UPF

```
# Logic and power management control ports
create_logic_port -direction in nRETNCPU0
create_logic_net nRETNCPU0
connect_logic_net nRETNCPU0 -ports nRETNCPU0
# RFF
set_retention ret_cpu0 -domain PD_COREA \
  -retention_supply_set PD_COREA.default_retention \
  -save_signal {nRETNCPU0 negedge}
  -restore_signal {nRETNCPU0 posedge}
# ISO
set_isolation iso_cpu_0 -domain PD_COREA \
  -isolation_supply_set PD_SOC.primary \
  -clamp_value 0 \
  -applies_to outputs \
  -isolation_signal nISOLATECPU0 \
  -isolation_sense low
# Power states updates for PD
add_power_state PD_CPU0 –domain -update \
 -state {RUN -logic_expr {nPWRUP_CPU0 == 0 && nPWRUPRetn_CPU0 ==
0}} \
  -state {RET -logic_expr {nPWRUP_CPU0 == 1 && nPWRUPRetn_CPU0 ==
0 && nRETN_CPU0 == 0 && nISOLATE_CPU0== 0}} \
  -state {SHD -logic_expr {nPWRUP_CPU0 == 1 && nPWRUPRetn_CPU0 ==
1}}
# Power state update for supply set
```

add_power_state PD_CPU0.***primary*** –***supply*** -***update*** \
-***state*** {ON -***supply_expr*** {*power* == FULL_ON && ground == FULL_ON} \
-***logic_expr*** {nPWRUP_CPU0 == 0}} \
-***state*** {OFF -supply_expr {power == OFF || ground == OFF} \
-***logic_expr*** {nPWRUP_CPU0 == 1}}

Finally, when the design is ready for implementation, it is the time to furnish UPF with technology specific physical entities like supply net and supply port for specific supply set, ISO or RFF cell, and their mapping information, logical and physical control and supply connectivity of PSW. As well further updates on power states with supply expression and exact supply voltage information. However, this final implementation UPF is concatenated on previous constraint and configuration UPF, and their combination completes the UPF modeling for the entire design.

Conventionally the RTL along with the constraint and configuration UPF are referred as the golden source and remain fixed throughout the entire DVIF. Once the configuration specified by the golden source are validated, it is then combined with implementation UPF to complete synthesis, post synthesis and post layout implementation and verification processes. The following example denotes the implementation UPF for the SoC design shown in Fig. 3.1 and works in combination with the constraint and configuration UPF.

Example 3.60 Sample Implementation UPF

```
# Supply Port & Net
create_supply_port VDD
create_supply_net VDD -domain PD_SOC
create_supply_net VDDSOC -domain PD_SOC
# Physical PSW connectivity
create_power_switch ps_SOC_primary –domain PD_SOC \
  -input_supply_port { VDD VDD } \
  -output_supply_port { VDDSOC VDDSOC } \
  -control_port { nPWRUPSOC nPWRUPSOC } \
  -on_state { on_state VDD {!nPWRUPSOC} } \
  -off_state { off_state {nPWRUPSOC} }
# Physical Supply net connectivity for Supply Set
create_supply_set PD_COREA.primary -update \
  -function {power VDDCPU0} -function {ground VSS}
create_supply_set PD_COREA.default_retention -update \
  -function {power VDDRCPU0} -function {ground VSS}
# Update of Power state with supply voltage
add_power_state PD_COREA.primary –supply -update \
 -state {ON -supply_expr {power == {FULL_ON 0.81} && ground == {FULL_
ON 0.00}}} \
  -state {OFF -supply_expr {power == OFF|| ground == OFF}}
add_power_state PD_COREA.default_retention –supply -update \
 -state {ON -supply_expr {power == {FULL_ON 0.81} && ground == {FULL_
ON 0.00}}} \
-state {OFF -supply_expr {power == OFF|| ground == OFF}}
```

The following List 3.24 summarizes the required features for modeling constraint, configuration and implementation UPF.

List 3.24 Required Feature Summery of Constraint, Configuration and Implementation UPF

Constraint UPF

- Atomic power domains
- Clamp value requirements
- Retention requirements
- Fundamental power states
- Legal, illegal states, transitions

Configuration UPF

- Actual power domains
- Additional domain supplies
- Additional power states
- Isolation and Retention strategies
- Control signals for power management

Implementation UPF

- Voltage updates for power states
- Level Shifter strategies
- Mapping to Library power management cells
- Location updates for Isolation
- Supply ports, nets, switches, and sets
- Port states or Power state tables

It is evident that the successive refinement of UPF addresses the needs of various design or IP resources within their own scope in the DVIF. For example, the IP developers who create reusable IP with independent power manageable constraints, the system integrator who configure IP blocks and integrate them together with power management logic, the chip implementers who map the logical design onto a technology specific physical implementations.

One of the major challenges of successive refinement of UPF is to identify power management and reduction constraints that will remain unchanged for a particular design life span. Hence this UPF modeling style sometimes tends to be more biased towards the verification objectives, specifically power intent verification and simulation targets at constraints and configuration UPF level, while structural, architectural and functional verification and simulation at implementation levels. Nevertheless the UPF abstraction choice is originated from verification objectives more than just modelling power management and reduction techniques from a designer perspective. Although the incrementally refinable UPF discussed in next Sect. 3, provides greater flexibility for designers to define UPF objects once and refine indefinitely when require.

3.3 Incrementally Refinable UPF

Incrementally refinable UPF is primarily referred to the refinement process by derivatives and refinement in place of UPF fundamental power states. These methodological approaches are deeply discussed in Sect. 1.3, with relevant Examples 3.30, 3.31, 3.30 etc. for definite and deferred power states.

The refinement by derivation creates a set of mutually-exclusive sub states or a completely new state based on the original power state. This is actually done through updating the logic expression (***–logic_expr***). The refinement in place actually modifies the original power states instead of creating a new power state. This is actually done through the UPF ***–update*** options of the power state in **add_power_state** command.

Both the refinement procedures are conducted in an incremental manner. Because, as the design rolls from RTL to GL-netlist or PG-netlist, more precise design information and implementation specific parameters became available incrementally. Thus the definite power states create more specific power states from initial abstract states and deferred power state becomes definite with updating implementation specific information like voltage values, power switch types etc.

There are several other UPF commands and some of their arguments supports incremental refinement and termed as *refinable commands* and *refining arguments* respectively.

A refinable command may be invoked multiple times on the same object and each invocation may add additional arguments to those specified in previous invocations. Actually the arguments of a refinable command that may be added after the first invocation are called refining arguments. Such arguments added in a consecutive manner on each iteration of incremental refinement and may have additional information about that argument added after the first invocation of the command.

Hence secondarily, apart from UPF power states, which is the core of UPF modeling and PA verification, the incrementally refinable UPF correspond to other UPF commands as well. For example power domain, power supply set, and different UPF strategies are also subject to refinement in place through ***–update*** options.

So apart from power states, there are multitude of such refinable commands and refining arguments, it is more appropriate to understand the incrementally refinable UPF for the fundamental constituent parts like power domains, power supply set and UPF strategies and then extend the contexts for modeling UPF for the entire design. In the following subsections, some of the refinable commands and refining arguments semantics for the fundamental constituent parts are discussed in detail.

Refinement of Power Domains

The UPF **create_power_domain** command defines power domains also have ***–update*** option and may be used in much similar style of power state to incrementally refine some of the options and possibly their arguments like ***–elements***, ***-subdomains***, ***-supply*** etc. of an already defined power domain. The following example explains the power domain refinement through ***–update*** on ***–supply*** arguments at

configuration level which was previously defined without –***supply*** information at constraint level.

Example 3.61 Refinement of Power Domain Through –*supply* **Arguments**

```
# Snippet of CONSTRAINT UPF
set_design_top rtl_top
# Create power domain for memory controller
create_power_domain PD_mem_ctrl -elements {mc0}
# Snippet of CONFIGURATION UPF
# Update PD_mem_ctrl to include the isolation and retention supply sets
create_power_domain PD_mem_ctrl -update \
   -supply { retain_ss } \
   -supply { isolate_ss }
```

The incrementally updated ISO and RFF supplies in Example 3.61 will enable to define corresponding supply set at later phase of DVIF after they are associated with the corresponding power domains as follows.

Example 3.62 Application of Power Domain Refinement at a Later Stage in DVIF

```
# Snippet of CONSTRAINT UPF
# add supply_sets still no voltage information needed
create_supply_set 0d81_ss
    # designate actual supply_sets for isolation and retention supples for PD_mem_ctrl
associate_supply_set 0d81_ss -handle PD_mem_ctrl.isolate_ss
associate_supply_set 0d81_ss -handle PD_mem_ctrl.retain_ss
# Snippet of CONFIGURATION UPF
# Setup retention strategy for memory controller domain
set_retention mem_ctrl_ret \
   -domain PD_mem_ctrl \
   -retention_supply_set PD_mem_ctrl.retain_ss \
   -retention_condition {(mc0/clk == 1'b0)} \
   -elements {mem_ctrl_ret_list} \
   -save_signal {mc_save high} \
   -restore_signal {mc_restore low}
```

Refinement of Supply Set

The supply set refining arguments are –***update*** and –***function*** and obviously works in much same way as incremental refinement of power domain works. The UPF **create_supply_set** command that defines a new supply set requires to update in implementation UPF with physical power and ground entities as follows in Example 3.63. However, it is important to understand that these supply set were just abstractly defined in constraint level so far and associated to different power domains as primary without specific power and ground (PG) port details. It is also required to know that the PG detail is mandatory to route supply power and ground to every design elements or cell-specifically in P&R level.

Example 3.63 Refinement of Power Supply Set through *–function* Arguments

```
# Snippet of CONSTRAINT UPF
# add supply_sets still no voltage information needed
create_supply_set 0d99_ss
create_supply_set 0d81_ss
create_supply_set 0d81_sw_ss
# Designate primary supplies for all domains
associate_supply_set 0d99_ss -handle PD_top.primary
associate_supply_set 0d99_ss -handle PD_sram.primary
associate_supply_set 0d81_ss -handle PD_intl.primary
associate_supply_set 0d81_sw_ss -handle PD_mem_ctrl.primary
# Snippet of IMPLEMENTATION UPF
create_supply_set 0d99_ss -function {power VDD_0d99 } -function {ground VSS} -update
create_supply_set 0d81_ss -function {power VDD_0d81 } -function {ground VSS} -update
create_supply_set 0d81_sw_ss -function {power VDD_0d81_SW } -function {ground VSS} -update
```

In the implementation UPF snippet above, the power and ground functions are actually physical ports and nets and must be defined and connected as shown below before they are updated.

Example 3.64 Application of Power Supply Set Refinement at a Later Stage in DVIF

```
# Snippet of IMPLEMENTATION UPF
# Create top level power domain supply ports
create_supply_port VDD_0d99 -domain PD_top
create_supply_port VDD_0d81 -domain PD_top
create_supply_port VSS      -domain PD_top
# Create supply nets
create_supply_net VDD_0d99 -domain PD_top
create_supply_net VDD_0d81 -domain PD_top
create_supply_net VSS -domain PD_top
create_supply_net VDD_0d81_sw -domain PD_mem_ctrl
# Connect top level power domain supply ports to supply nets
connect_supply_net VDD_0d99 -ports VDD_0d99
connect_supply_net VDD_0d81 -ports VDD_0d81
connect_supply_net VSS      -ports VSS
```

Refinement of UPF Power Strategies

UPF strategies among ISO, LS, ELS, RFF, PSW and RPT etc. also have refinable commands and handful of refining options and arguments. However, all the strategies inherit incremental refinement through *–update*. Though for ISO, LS or ELS and RPT the *–update* command allows refinement through supplementary information on a previously defined command for the same strategy and same power domain, and also requires that the strategy execution occurs in the same scope.

While for PSW and RFF refinement allows the addition of instances and retention strategies.

The following Example 3.65 shows the refinement of ISO as a representative of all other UPF strategies.

Example 3.65 Refinement of ISO Through *–update*

```
# Snippet of CONFIGURATION UPF
# Setup ISO strategy for memory controller domain; No location information yet
set_isolation mem_ctrl_iso_1 \
   -domain PD_mem_ctrl \
   -isolation_supply_set PD_mem_ctrl.isolate_ss \
   -clamp_value 1 \
   -elements {mc0/ceb mc0/web} \
   -isolation_signal mc_iso \
   -isolation_sense high
# Update PD_mem_ctrl isolation strategies with -location info
set_isolation mem_ctrl_iso_1 -update \
  -domain PD_mem_ctrl \
  -location parent
```

Hence it must be distinctive at this point that the fundamental difference of *incremental refinement* and *successive refinement* terminology as consciously used in UPF has significant role on evolving UPF over a particular DVIF life cycle. Where in generic form, the first, *incremental refinement* corresponds to the pertinent refinement of UPF semantic components and the later *successive refinement* corresponds to utilize the refined components to model the complete UPF for the entire design to be consumed in the verification and implementation flow.

More specifically *incremental refinement* denotes how and *successive refinement* denotes when the refinement actually occurs. However, both occurs within the same bounded space with different perspectives.

3.4 Hierarchical UPF

Hierarchical UPF has a complete different focus in modeling UPF from that of successively or incrementally refinable UPF, though these both are tightly integrated while developing hierarchical UPF. Here the effort is to develop UPF at design block level within its own HDL reference instance scope and gradually or in parallel build up UPF for the entire design in bottom-up manner by calling or loading them from the extent to their immediate or prospective hierarchical top of HDL references.

However, this UPF development strictly relies on *successively refinable* flow and *incrementally refinable* methodologies. But in broader aspect, hierarchical UPF is a flow in PA design verification and implementation process.

Primarily hierarchical UPF objective is to simplify block level design verification, facilitates maintenance of block level design independently, and as well implementation of such block level design through the synthesis process. Because block level PA

verification and synthesis with UPF is always simpler than perusing the same for the entire SoC. The following example shows the constructions and constituent parts of hierarchical UPF based on the design example shown in Sect. 2.2, Figs. 2.1 and 2.2.

Example 3.66 Sample Hierarchical UPF from Verification and Implementation Perspective

set_design_top cpu_top
Load Successively Refinable UPF
load_upf cpu_constraints.upf
load_upf cpu_configuration.upf
load_upf cpu_implementation.upf
Fundamental Constituent Part of cpu_top UPF
Creating PD, supply set, power states etc.
create_power_domain PD_top
……..
load hierarchical UPF
load_upf PD_sub3.upf ***-scope*** umem_top
load_upf PD_sub2.upf ***-scope*** udecode_top
New scope for mem sub domain
set_scope cpu_top/umem_top
Further load of sub block Successively Refinable UPF
load_upf mem_constraints.upf
load_upf mem_configuration.upf
Fundamental Constituent Part of mem UPF
Creating PD, supply set, power states etc.
create_power_domain PD_sub3 ***–elements*** {.}
……..
load hierarchical UPF for mem sub domain
load_upf sub3.1_PD.upf ***-scope*** cpu_top/umem_top/umem_sub

As shown in the example above, the hierarchy navigation in UPF occurs as the commands are executed in the context of a scope within the logic hierarchy. In the latter part of the above example, the **set_scope** UPF command is used to navigate within the hierarchy and to set the current scope to cpu_top/umem_top, within which commands are executed. The current scope is represented by a relative path-name from the design top instance.

The **set_scope** command changes the current scope locally within the subtree depending on the current design top instance or module. Since the design top instance is typically an instance of the design top module, they both have the same hierarchical substructure; therefore, **set_scope** can be written relative to the module, but still work correctly when applied to an instance. The **set_scope** command is only allowed to change scope within this subtree. It cannot change the scope above the design top instance or to a scope that is below in the leaf-level instance.

Hence based on the scope level, it is required to understand the extent of design top instance designated as top power domain.

The design top instance and design top module are typically paired in hieratical UPF which is cpu_top shown in the first part in the above Example 3.66. The top

instance is represented by a hierarchical name relative to the root scope. It is consistent with SystemVerilog $root, the root of the logic hierarchy is the scope in UPF where the top modules are implicitly instantiated. Other locations within the logic hierarchy are referred as the design top instance, which has a corresponding design top module, and the current scope.

There is an UPF command **set_design_top** that actually associates the design top module to each instance in a larger hierarchical UPF flow. Generally, in such a flow, it is required to manually set the design top instance and design top module. When a sub domain UPF is called or load from its relative top power domain through **load_upf** command, the design top instance are implicitly changed to the current scope of the invoked sub domain with the **-scope** argument as shown in the Example 3.66 for PD_sub3.upf and PD_sub2.upf, where the scopes are umem_top & udecode_top respectively.

However, after the subdomain ***set_scope*** cpu_top/umem_top is defined, the scope changes as current to this instances locally and includes all descendent subtree within the same scope. Hence sub3.1_PD.upf may be allowed to invoke under the current scope with ***load_upf –scope*** cpu_top/umem_top/umem_sub command.

Thus block level UPF can be developed independently and augment to the corresponding design top any time in a bottom-up way. Hence hierarchical UPF as a flow ease the modeling, maintenance and verification of UPF for a large design and must accommodate the recommended successive refinable flow, formats and incrementally refinable semantics every when they are required.

However, there are other prospect of modeling hierarchical UPF. Specifically integration of Macro cell (soft or hard Macro both) in hierarchical order. Hence this is the secondary artifact of hierarchical UPF that also facilitates Macro integration in conjunction with simplifying block level design verification, maintenance, and as well synthesis.

The Macro integration is done through another special UPF **set_port_attribute** commands.

Example 3.67 Syntax UPF Command set_port_attribute to Supplement Hierarchical Flow

set_port_attributes
[-***model*** name]
[-***elements*** element_list]
[-***exclude_elements*** element_exclude_list]
[-***ports*** port_list]
[-***exclude_ports*** port_exclude_list]
[-***applies_to*** <inputs | outputs | inouts | {<inputs | outputs | inouts >*}>]
[-***attribute*** {name value}]*
[-***clamp_value*** <0 | 1 | any | Z | latch | value>]
[-***sink_off_clamp*** <0 | 1 | any | Z | latch | value>]
[-***source_off_clamp*** <0 | 1 | any | Z | latch | value>]
[-***driver_supply*** supply_set_ref]
[-***receiver_supply*** supply_set_ref]
[-***literal_supply*** supply_set_ref]
[-***pg_type*** pg_type_value]

[-***related_power_port*** supply_ port_name]
[-***related_ground_port*** supply_ port_name]
[-***related_bias_ports*** supply_ port_name_list]
[-***feedthrough***]
[-***unconnected***]
[-***is_analog***]
[-***is_isolated***]

The **set_port_attributes** command is for specifying information associated with ports of models or instances. Model ports are referenced by –***model*** and instance ports are referenced by using either ***-elements*** or ***-ports*** (and obviously without –***model***) option(s). If -***model*** is specified and the model name is (.) (a dot), the command applies to the model corresponding to the current scope.

Despite other options, the ***-driver_supply*** and ***-receiver_supply*** corresponds to the UPF attributes ***UPF_driver_supply*** and ***UPF_receiver_supply*** respectively as noted for logic port source and sink driving mechanism in Sect. 1.

These options or attributes can be used to specify the driver supply of a Macro cell output port or the receiver supply of a Macro cell input port. They can also be used to specify the assumed driver supply of external logic driving a primary input or to specify the assumed receiver supply of external logic receiving a primary output; specifically when the Macro is implemented separately from the context in which it will be instantiated. These attributes are ignored if applied to a port that is not on a Macro boundary.

Recalling the Fig. 3.3 of example of a complex SoC, and considering the CPU Cluster A, where any soft Macro is required to integrate on the PD_CPU0 as reference power domain, the redrawn figure is shown in Fig. 3.7.

The relevant **set_port_attribute** commands for the Macro with corresponding driver and receiver supply is shown in Example 3.68.

Example 3.68 Snippet of UPF for Macro Integration at PD_CPU0 Hieratical Level

```
# UPF in Context to PD_CPU
set_port_attributes –ports inP -driver_supply PD_CPU0.primary
set_port_attributes –ports outP -receiver_supply PD_CPU0.primary
```

Here the external Macro which is represented by the symbolic logic clouds in Fig. 3.7 are defined in terms of the interface or boundary context of the driver and receiver supply for inP or outP ports of PD_CPU0 power domain. Although the supplies are modelled using available supplies or handles as (PD_CPU0.***primary***) of PD_CPU0.

However when considering the Macro integration from a higher level of hierarchical perspective, for example, CPU Cluster A, which is actually the entire SoC or PD_SOC, as redrawn in Fig. 3.8 from Fig. 3.3.

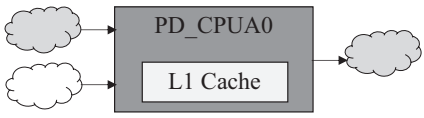

Fig. 3.7 Soft macro integration perspective for PD_CPU0 power domain

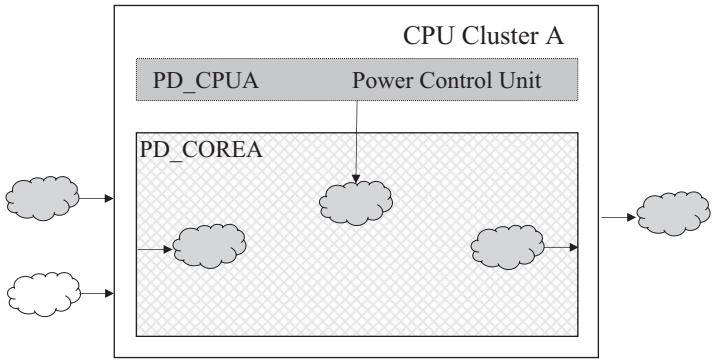

Fig. 3.8 Soft macro integration perspective for the CPU cluster A (actually PD_SOC)

Example 3.69 Snippet of UPF for Macro Integration at PD_SOC Hieratical Level

```
# UPF in Context to PD_CPU
set_port_attributes –ports inP -receiver_supply PD_CPU0.primary
set_port_attributes –ports outP -driver_supply PD_CPU0.primary
# UPF in Context to CPU Cluster A or PD_SOC
set_port_attributes –ports top/inP -driver_supply PD_SOC.primary
set_port_attributes –ports top/outP -receiver_supply PD_SOC.primary
```

It is required to notice here that the boundary interface contexts of Macro for PD_CPU0 is changed while considering a higher level of hierarchical context from PS_SOC. That is the driver and receiver supply for inP and outP ports are reversed in Example 3.69 compared to Example 3.68.

Hence it is comprehensive that hierarchical UPF simplifies the block level design verification, block level synthesis of the design with block level and its independent UPF, and as well facilitates Macro integration at any hierarchical contexts.

In this concluding part of this chapter it is distinctly visible that UPF modeling is primarily governed by PA design verification and secondarily to accommodate the verification in different level of design implementation in the DVIF efficiently.

Epilogue: The **Chapter 3 – Modeling UPF** is an enormous chapter that covers the fundamentals and advanced topics for efficient and effective UPF modeling. It provided the key to the readers to explore all possible artifacts of low power design and power aware verification for succeeding chapters of this book. This chapter started with explaining the fundamental constituent parts of general UPF constructs and explains power domains through **create_power_domain** – ***elements*** {.} UPF constructs. The HighConn, LowConn, lower or upper domain boundary, domain interface, ports etc. are explained through exploiting the common and formal port definitions for actual signal connections in terms of hierarchical design instances.

When a portion of design element is confined within a domain, the next obvious aspects is to provide the power supply. The power supply abstractions are explained through the **create_supply_set** ***–function*** or (and) ***–update*** constructs in conjunctions with **associate_supply_set** ***–handle*** and **create_power_domain** ***–supply*** and ***–update*** constructs. Next the **add_power_state** construct for power state – the pivotal factor and the central force of UPF modeling and power aware verification is explained through complex SoC design example in comprehensive manner. While mentioning the specific reason of conventional PST (**add_port_state**, **add_pst_state** and **create_pst**) becoming obsolete; the superiority of **add_power_state** covers detail through refinement by derivatives for definite power state through "*new state*", and refinement in place for deferred power state through "***-update***".

The guidelines, recommendations and compositions summery of power states listed in Lists 3.17, 3.18, and 3.19 will remain as point of reference while writing UPF or conducting PA verification. The power strategies are also constituent part of UPF and ISO, LS, PSW, RFF, RPT are explained in this chapter with syntax, semantics and practical usage perspective. Finally the UPF modeling flow and constructional refinement features are explained through successive and incremental refinements aspects. In addition, benefit and usage of hierarchical construction of UPF are also covered at the end of the chapter.

Prologue: The next chapter, **Power Aware Standardization of Library** will discuss and explain potential attributes of standard Liberty libraries for special cells from UPF power strategies and Macros. This chapter also explains the constructions, usage and requirements of other non-standard simulation model libraries in PA verification.

Chapter 4
Power Aware Standardization of Library

Multivoltage (MV) based power aware (PA) design verification and implementation methodologies require special power management attributes in libraries for standard, MV and Macro cells for two distinctive reasons. The first aspect is to provide power and ground (also bias) supply or PG-pin information, which is mandatory for PA verification. The second reason is to provide a distinctive attribute between a special power management MV cell and a regular standard cell. The special MV cells include isolation (ISO), level-shifters (LS), enable level-shifter (ELS), always-on buffers (AOB), feed through buffers or repeaters (RPT), diode clamps, retention flops (RFF), power switches (PSW), multi- and single- rail Macros. This Chapter describes the standard requirements of Power Aware or PG-pin libraries and provides modeling examples of the MV cells from UPF based PA verification perspective.

Libraries play a crucial role in the entire design verification and implementation flow (DVIF). Specifically for PA design verification and implementation, special design attributes are mandatory in an industry standard library format, known as the Liberty library description syntax. The Liberty syntax is usually available with (.lib) file extensions. In PA verification, special Liberty cell-level and pin-level attributes are required to characterize standard, MV and Macro cells and identify their corresponding supply or power-ground (PG) pin connectivity.

Generic and specific cell Liberty syntax file (.lib) examples are shown below for LS, as a representative MV cell. The specific cell example will be used to provide a simplified explanation of UPF-based PA verification in the succeeding sections.

Example 4.1 LS Cell Generic Liberty Syntax

```
cell(level_shifter) {
is_level_shifter : true ;
level_shifter_type : HL | LH | HL_LH ;
input_voltage_range ("float, float");
output_voltage_range ("float, float");
...
```


P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8_4

```
pg_pin(pg_pin_name_P) {
pg_type : primary_power;
std_cell_main_rail : true;
...
}
pg_pin(pg_pin_name_G) {
pg_type : primary_ground;
...
}
pin (data) {
direction : input;
input_signal_level : "voltage_rail_name";
input_voltage_range ("float , float");
related_power/ground_pin : pg_pin_name_P/pg_pin_name_G ;
related_bias_pin : "bias_pin_P bias_pin_G";
level_shifter_data_pin : true ;
...
}/* End pin group */
pin (enable) {
direction : input;
input_voltage_range ("float , float");
level_shifter_enable_pin : true ;
...
}/* End pin group */
pin (output) {
direction : output;
output_voltage_range ("float , float");
power_down_function : (!pg_pin_name_P + pg_pin_name_G);
...
}/* End pin group */
...
}/* End Cell group */
```

Example 4.2 LS Cell Specific Liberty Sample

```
cell(A2LVLUO) {
is_level_shifter : true ;
level_shifter_type : HL_LH ;
input_voltage_range(0.8, 1.2);
output_voltage_range(0.8, 1.2);
....
pg_pin(VNW) { pg_type : nwell;
pg_pin(VPW) { pg_type : pwell;
pg_pin(VDDO){ pg_type : primary_power ;
```

```
pg_pin(VSS) { pg_type : primary_ground ;
pg_pin(VDD) { pg_type : primary_power ;
std_cell_main_rail : true ;
....
pin(A) {
related_power/ground_pin : VDD/VSS ;
related_bias_pin : "VNW VPW";
level_shifter_data_pin : true ;
....
pin(EN) {
related_power/ground_pin : VDDO/VSS ;
related_bias_pin : "VPW";
level_shifter_enable_pin : true ;
....
pin(Y) {
related_power/ground_pin : VDDO/VSS ;
related_bias_pin : "VPW";
power_down_function : "!VDDO+(!VDD&EN)+VSS+VPW+!BIASNW";
```

4.1 Liberty Power Management Attributes

The generic and specific examples given in previous section prevails several attributes, for instance, the followings are known as special cell-level attributes that categorize this particular cell as an LS.

Example 4.3 Special Cell-Level Attributes for LS

- `is_level_shifter:true,`
- `level_shifter_type:HL_LH,`
- `input_voltage_range` and
- `output_voltage_range`

Hence when these attributes are missing, the LS will be treated as just a regular standard cell. All the remaining attributes in these examples as listed below in Example 4.4 are termed as pin-level attributes.

Example 4.4 Special Pin-Level Attributes for LS

```
pg_pin,
pg_type,
related_power,
related_ground,
bias_pin,
std_cell_main_rail, and
power_down_function'
```

It is worth mentioning that some of these Liberty attributes are also implied in UPF as predefined attribute names. UPF supports the specification of attributes of objects in a design. Hence UPF allows such attributes to be used with HDL specifications in design code or with Liberty attribute specifications in a Liberty library. Table 4.1 shows a few of the Liberty attributes that are relevant to the UPF predefined attribute names.

To note, there are other such UPF attributes implied with the Liberty syntax that is not relevant to the context discussed here.

Continuing with the LS examples, the 'pg_pin' and 'pg_type' attribute together facilitate specification of power, ground, and bias pin connectivity of the cell. These in general will correspond to the primary power (VDD, VDDO), ground (VSS), and bias (VNW, VPW) supply of the power domain where this cell actually belongs, as specified in UPF.

The 'input_voltage_range' and 'output_voltage_range' are the voltage ranges (from 0.8 to 1.2 volt) for all the input or output pins of the cell under all possible operating conditions. The 'related_power/ground_pin' and 'related_bias_pin' provide the related power, ground, and bias supply connectivity information for each input or output logical port or pin of the cell.

Related supply is augmented with 'pg_pin' attributes that indicate the functionality of the supply, whether it is primary power or primary ground. For single rail cells, when there is only a single set of power and ground supplies, all the inputs or outputs have only one set of related supplies. However for multi-rail cells, especially the MV and Macro cells, like the LS in this example (which is a MV cell), usually possesses different related supplies for the input and output.

Figure 4.1 shows the diagram of LS cell explained above with related PG-pin information for all input and output pin. Here the related supplies for input pin (A) is (VDD/VSS) and for output pin (Y) is (VDDO/VSS).

The 'std_cell_main_rail' attribute defines the primary power pin (VDD) that will be considered as the main rail. This is a power supply connectivity parameter that is required when the cell is placed and routed at the post P&R level. The 'power_down_function' expression identifies the Boolean condition that consists of the combination of the different statuses of the power, ground, and bias pins, which signifies when the output pin of the cell will be turned off by these power,

Table 4.1 Liberty and relevant UPF attributes

Liberty attribute	UPF predefined attribute name
pg_type	UPF_pg_type
related_power_pin	UPF_related_power_port
related_ground_pin	UPF_related_ground_port
related_bias_pins	UPF_related_bias_ports
is_hard_macro	UPF_is_hard_macro

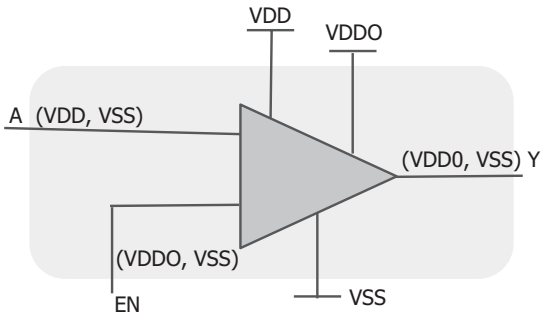

Fig. 4.1 Level-shifter with related PG-pin information

ground, and bias pins. The '`power_down_function`' is defined in the library exclusively for verification purposes only. More precisely, the function is entirely for UPF-based dynamic simulation, where it facilitates the verification tool to comprehend the corruption behavior for the cell when the power domain (where the cell resides) is powered-Off or switched to a shutoff condition.

The corruption semantics are imposed by the PA simulator imply on internal sequential elements, input and output signals, ports, and pins of the cell to change their regular known values to unknown values, in the event of power down or inadequate power supply to drive those logics correctly. So it actually defines a rule set that decides, how logic elements essentially behave in response to reduction and disconnection of power. Corruption typically refers to the situation where the value of logic elements or signals becomes unpredictable. Hence, corrupted logic is usually assigned to 1'bx or 1'b0 or hi'z depending on logic types or user preferences. The detail on corruption semantics is further discussed in Chap. 5.

The detail on power down functions are further discussed in succeeding sections for other library formats. Liberty libraries are mandatory for post-synthesis, gate-level PA static verification and in certain circumstances they are also required for PA dynamic simulation. From the examples and discussion above, it is evident that both the cell-level and pin-level attributes are mandatory for any MV or Macro cell Liberty syntax. However, only pin-level attributes are applicable for standard cells used in PA verification, since no special attribute is required to distinguish a standard cell. UPF 2.1 and 3.0 LRM mention supporting the Liberty 2009.06 release syntax; however the latest release by the open Liberty organization is 2015.12.

4.2 Power Aware Verification Model Libraries

Apart from the Liberty standard library syntax, there are other formats of non-standard behavioral model libraries, mostly modeled in HDL with (.v or .vhd) file extensions. The (.v or .vhd) models are specifically required for dynamic simulation-based verification and obviously available in two different formats, PA and non-PA simulation (.v or .vhd) model libraries. Hence these libraries may or may not include the supply power and ground pin information. Standard and MV cell models are usually written

as Verilog HDL modules and use constructs such as Verilog built-in primitives or user-defined primitives (UDPs) to express the relatively simple behavior of a standard cell. They may also be written as VHDL design entities (entity and architecture pairs) using VITAL package, which provides Verilog-like primitive modeling capabilities.

Similarly, hard Macro simulation model libraries can be written in either language, using more complex behavioral constructs such as Verilog initial blocks and always blocks, or VHDL processes and concurrent statements. However, PA-Simulation Model and Non-PA-Simulation Model libraries are not standardized like the Liberty standards. The following are the examples of LS cell behavioral simulation libraries, modeled in Verilog HDL.

Example 4.5 LS Cell Behavioral Non-PA-Simulation Model Library

```
module A2LVLUO (Y, A, EN);
output Y;
input A, EN;
and I0 (Y, A, EN);
endmodule
```

Example 4.6 LS Cell Behavioral PA-Simulation Model Library

```
module A2LVLUO (Y, A, EN, VDD, VDDO, VSS, VNW, VPW);
output Y;
input A, EN;
inout VDDO, VDD, VSS, VNW, VPW;
and I0 (out_temp, A, EN);
assign Y = ((VDDO === 1'b1) && (BIASNW === 1'b1) && (VPW ===
1'b0) && (VSS === 1'b0) && (!EN|VDD === 1'b1))? out_temp : 1'bx;
endmodul0065
```

It is evident from these examples that Non-PA-Simulation Model libraries are just functional behavioral models without any power, ground, and bias, or PG-pin information of a cell. Whereas PA-Simulation Model libraries provide more accurate power information with detail and exact matching PG-pin with its counterpart of standard Liberty (.lib) library. The PA-Simulation Model library also contains the power-down function (conditional assignment statement in the Example 4.6) similar to its (.lib) file.

As the naming of simulation model libraries suggests, their construct actually governs the fundamental aspects of PA simulation. Hence it's worth noting that a UPF-based PA verification environment may have all of these three different types of libraries; namely PA-Simulation Model, Non-PA-Simulation Model, and Liberty for any particular cell on different purposes, times, and phases of the design verification and implementation flow. However it completely depends on the simulation tool and UPF methodologies how these libraries are consumed, individually or in combination, how they are interpreted, and how they are processed to produce targeted PA verification results.

The next sections highlight the use-models of the previously defined Non-PA and PA simulation libraries (Examples 4.5 and 4.6); specifically how these libraries are combined with UPF and their corresponding counter part of the standard Liberty (.lib) library to furnish PG-pin information to the verification tool and accomplish PG-pin connectivity, and power-down corruption activity for accurate PA simulation based verifications.

4.2.1 Non-PA Simulation Model Library

As mentioned earlier, the Non-PA-Simulation Model libraries are just behavioral functional models without any representation of supply ports or affiliated power states, in contrast to the power down functionalities available in Liberty libraries or PA-Simulation Models. The verification tool, specifically a simulator, identifies a simulation model as non-PA only when the PG-pin declarations are not available within the model. The simulator usually resolves PG-pin connectivity and power-related simulation corruption semantics in either of the following ways.

First the tool searches for the corresponding Liberty (.lib) file for the cell and connects all the PG-pins from the Liberty library to the corresponding power domain where the cell actually resides, specified in the UPF. Since the corresponding Liberty library of the cell is available, the output corruption of the cell is also performed based on the power-down functionality from the Liberty library. The second procedure emerges only when the Liberty library for the cell is not available. Hence the tool implicitly connects the cell to the primary supply of the power domain where it resides. The power-related corruption for this case, applies to the output of the cell, based on the status of the power states of the primary supply of the power domain, usually expressed as – ***simstate*** and shown below.

Example 4.7 Supply State Statues Through -*simstate*

add_power_state PD_sub1.***primary*** ***-state*** INT_ON { ***-supply_expr*** { (***power*** == *FULL_ON*) && (***ground*** == *FULL_ON*) } ***-simstate*** NORMAL}
add_power_state PD_sub1.***primary*** ***-state*** INT_OFF { ***-supply_expr*** { (***power*** == *FULL_OFF*) && (***ground*** == *FULL_OFF*) } ***-simstate*** CORRUPT}

Here, it is assumed that the Non-PA-Simulation Model is instantiated with the hierarchical elements that belong to the PD_sub1 power domain. There are two power states INT_ON with ***-simstate*** NORMAL when the output of the cell will remain unaffected; however, during INT_OFF with ***-simstate*** CORRUPT, the output will be corrupted.

The Non-PA-Simulation Models are well suited for modeling standard cells with a single rail and are traditionally used for post-synthesis, gate-level functional verification or logic simulation in a purely non-PA verification environment. However, they can be straightaway accommodated to the PA simulation verification environment at the post-synthesis gate-level by combining with UPF and the corresponding Liberty standard library.

4.2.2 PA-Simulation Model Library

On the contrary, a PA-Simulation Model library completely represents all the power, ground, bias, and related supply ports or PG-pins of a cell. As well it defines the power down functionalities. The PG-pins in PA-Simulation Models are primarily defined as input and output ports; however they may also be defined as internal registers, wires, or as supply_net_type, supply0 and supply1 type etc. Although the internal types of PG-pins are more common in an extended PA-Simulation Model library explained in the next section.

The PA-Simulation Model also includes the behavioral code that monitors the supply ports and appropriately corrupts its internal states and outputs, in response to events or values on the power supply and logic ports. However, the explicit connection of an external testbench and UPF power supply to these PA model's supply ports is mandatory through UPF **connect_supply_net** or **connect_supply_set** explicit commands. The explicit UPF connections disable the simstate-based corruption semantics, unlike non-PA models explained in the previous section. Hence Questa® power aware simulator or PA-SIM allows the PA-Simulation Model library (.v) to take precedence and apply corruption semantics by itself. The simulator drives only the appropriate supply values to the PG-pin of the cell — when VDD is turned off (Example 4.6), the output Y will become 1'bx.

Obviously a corresponding Liberty (.lib) library is unnecessary for PA verification with PA-Simulation Model library. The PA-Simulation Models are more suitable for modelling multi-rail macros and specifically they are created for PA simulation-based verification at the post place-and-route PG-netlist level, since PG-netlists contain PG-pin connectivity as well as the logical functionality of a cell.

Questa® PA-SIM also supports UPF predefined attribute-based automatic connections, if a supply port has the **UPF_pg_type** attribute associated with it — either by an HDL attribute specification or a UPF **set_port_attributes** command, based upon its pg_type. In this case, the appropriate value conversion table (VCT) will also be inserted based on the pg_type of the port.

Example 4.8 UPF Predefined Attributes Usage in HDL

For VHDL: attribute UPF_pg_type of vdd_backup: signal is "backup_power"
For System Verilog: (* UPF_pg_type = "backup_power" *) input vdd_backup;

4.2.3 Extended-PA-Simulation Model Library

In addition to Non-PA-Simulation Model and PA-Simulation Model libraries, there is a combined form of functional and power aware simulation model libraries, often termed as Integrated or Extended-PA-Simulation Model library. The requirements of such extended libraries come from the distinctive verification artifacts that makes them usable for both the non-PA regular functional (logic) verification and PA verification

environment, while keeping both the functional and power features active internally in both environments. This is desirable because it is sometimes convenient to have simulation models specifically for hard Macros that can be used in both of these verification environments, without adding extra levels of hierarchy and without leaving power ports unconnected. Only extended power aware simulation models make this possible.

The fundamental power-related construction of Extended-PA Models differs from PA-Simulation Models in power port declaration semantics. While PA-Simulation Models declare ports on the interface of the model as input or output types, Extended-PA supply ports are defined as internal wires or registers as well as supply_net_type, supply0, and supply1 types within the HDL simulation models. The following example explains the construct of Extended-PA-Simulation Model.

Example 4.9 A Macro Cell Behavioral Integrated or Extended-PA-Simulation Model Library

```
module PVSENSE (RTO, SNS, RETON);
output RTO,SNS;
input RETON;
supply1 DVDD, VDD;
supply0 DVSS, VSS;
reg RTO_reg, SNS_reg;
assign RTO = DVDD? RTO_reg:1'bx;
assign SNS = DVDD? SNS_reg:1'bx;
always @(VDD)
if (!VDD) RTO_reg = !RETON; SNS_reg = RETON;
else RTO_reg = 1'b1; SNS_reg = 1'b1;
endmodule
```

The integrated model defines only the power supplies (DVDD, VDD, DVSS, and VSS) despite of other PG-pins, including bias and related power and ground pins usually available in its counterpart in the Liberty (.lib) format. However, the supplies are defined as internal objects representing assigned default constant values that enable normal operational mode for non-PA simulation. During PA simulation with Questa® PA-SIM, when UPF supply nets are connected to those internal objects through explicit connections via **connect_supply_net** or **connect_supply_set** commands or through automatic connections based on the **UPF_pg_type** attribute (similar to PA-Simulation Models), the UPF supply net overrides the default constant values of the model, and the model then behaves as a PA-Simulation Model.

More specifically from Example 4.9, the power (VDD) and ground (VSS) are defined as supply1 and supply0, which complies with the UPF LRM specification for designating power as 1'b1 and ground as 1'b0 logic resolution, when both are in the On state. During PA simulation, Questa® PA-SIM provides the connectivity to the VDD and VSS of the model with the corresponding power domain primary and ground specified in the UPF. Hence such models are readily usable in RTL (considering power as 1'b1 and ground as 1'b0 as constant values) as well as in post-synthesis

gate-level PA-Simulation with UPF, where the actual physical Macro cells are already inserted (considering VDD and VSS are connecting through the UPF supply net and then they can be driven from testbench).

Even the Extended-PA-Simulation Models are also usable for post-layout PG-netlist level PA simulation with UPF. This is because the physical connections of power and ground of the Macro cell are already available in the netlist, and UPF provides the hooks to the internally defined power (VDD) and ground (VSS) through supply net connections similar to post-synthesis gate-level PA-Simulation. It is worth mentioning here that the PG-netlist based PA dynamic simulation verification does not require UPF, when a regular PA-Simulation Model is available. Table 4.2 summarizes the library requirements for Questa® PA-SIM.

It is distinctive at this point that Extended-PA-Simulation Models have greater adaptability in different simulation environments and different levels of design abstraction — from RTL to PG-netlist. However, depending on the verification targets and objectives, particularly for PA verification, all four types of libraries may become relevant and useful. Although the requirements of these libraries differ, depending on design abstraction levels, both the Liberty (.lib) and simulation-models (.v) may not be required simultaneously for a design in a PA simulation environment. Hence it is also important to know how Questa® PA-SIM processes libraries and specifies the order of precedence when considering only a particular type or when

Table 4.2 Library requirements for PA verification

Design flow	Design abstraction flow	Library for PA-static checks	Library for PA-dynamic checks & simulation
Design Entry from Integration/ Algorithm/Spec/IP	Pure RTL	(.lib) – Good to have for MV, Macro (.v) Model – No	(.lib) – No (.v) Model – Yes
	MV/Macro Instantiated RTL	(.lib) – Yes for MV, Macro (.v) Model – Yes for MV, Macro	(.lib) – No (.v) Model – Yes for MV, Macro
Synthesis	Gate-Level (GL)	(.lib) – Yes (.v) Model – Yes	(.lib) – No (.v) Model – Yes
P&R/Layout	PG-Netlist	(.lib) – Yes (.v) Model – Yes	(.lib) – No (.v) Model – Yes
	Non PG-netlist	(.lib) – Yes (.v) Model – Yes	(.lib) – No (.v) Model – Yes

(.lib) – PG-pin Liberty library
(.v) – PA-Simulation Model Library (fully matching PG-pin with its counterpart of Liberty library)

multiple types of libraries are available in a simulation environment. The succeeding Chap. 5 explains the detail on tool specific library and attributes order of precedence processing.

Epilogue: The Chap. 4 – **Power Aware Standardization of Library** vividly explained the essential Liberty (or .lib) attributes of MV cells and Macros, crucial and mandatory for PA static and PA dynamic verification. These attributes are `pg_pin, pg_type, related_power, related_ground, bias_pin, std_cell_main_rail,` and `power_down_function.` The `power_down_function` is specially required only for PA dynamic simulation to manipulate the corruption behavior for the cell, when the power domain (where the cell resides) is powered-Off.

Apart from the Liberty standard library, there are three types of non-standard behavioral model (or .v or .vhd) libraries, namely non-PA, PA and extended-PA-Simulation Model libraries. The PG or power ground pin or supply port information are not available in non-PA while in PA and Extended-PA-Simulation Model libraries, PG pin are explicitly declared as in, out, inout or supply_net_type, supply0, and supply1 type. As well it is mandatory that the PG-pin and logic-pins or ports of these PA-Simulation Model libraries exactly matches the corresponding Liberty libraries. The non-PA model libraries are mainly used for dynamic simulator compilation purpose and hooked with the primary supply of the power domain where the cell actually exists.

The UPF ***–simstate*** of the power state remain responsible for corruption of a non-PA model. The PA and Extended-PA simulation model libraries remain responsible to apply corruption semantics by itself. Because these PA models are mandatorily connected to the UPF power supply and supply net of external testbench through UPF **connect_supply_net** or **connect_supply_set** explicit commands. The explicit UPF connections disable the **–simstate** based corruption semantics, unlike non-PA models explained in Sect. 2. Hence dynamic simulator e.g. Questa® PA-SIM allows the PA-Simulation Model library (.v) to take precedence and apply corruption semantics by itself. One important artifact about the Extended-PA-Simulation Model library is that it is usable for both the non-PA regular functional (logic) verification and PA verification environment, while keeping both the functional and power features active internally in both environments. In addition the Table 4.2 **Library Requirements for PA Verification** will remain a source of reference for the readers to understanding and actually running static and dynamic verification.

Prologue: The next chapter, **UPF Based Power Aware Dynamic Simulation** provides a generic foundation of PA dynamic simulation based verification. However, as a supplement, the chapter also provides a practical approach through explaining the verification practices of Questa® PA-SIM. The beginning of the chapter explains the fundamental topics of PA simulation and distinction between the non-PA simulation environments. Gradually the readers will find the advanced topics of PA dynamic verification features, library and testbench requirements, custom PA dynamic checkers, gate-level PA simulation and overall PA simulation debugging techniques.

Chapter 5
UPF Based Power Aware Dynamic Simulation

UPF based power aware (PA) verification adopts several power dissipation reduction techniques based on the target design implementation and UPF power specification or intent, as discussed in Chap. 2. These techniques introduce numerous and complex verification issues and challenges in the functional and structural aspects of the design. Such artifacts are completely nonexistent in a non-PA verification environment. For example, the power aware requirements may affect the design functionality in terms of power On-Off sequences, different modes of power operation, state or data preservation operations, data propagation, logic resolution, power state transitions, power state transition coverage, power state cross coverage, and more.

These structural issues affect a design's architectural and microarchitectural aspects physically, which reveals the requirements of physical insertions of PSW, ISO, LS, ELS, RPT, and RFF cells. These power management and multivoltage (MV) cells are required for power shutdown for several reasons. They prevent inaccurate data propagation between Off and On power domains and ensure accurate logic resolution between high-to-low or low-to-high voltage power domains. They also ensure feed-through buffers are inserted properly on control, clock, and reset signals that are passing through off power domains. As well data or state retention, primary power, ground, bias, related and backup power connectivity, etc. Obviously, such functional issues and physical accumulation of special cells or structural changes can be properly addressed by deploying PA-SIM and PA-Static verification techniques with suitable methodologies.

The succeeding sections discuss these verification techniques and methodologies for debugging and resolving all the power related issues for any possible power dissipation reduction techniques on any design, from SoC to processor core.

P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8_5

5.1 PA Dynamic Verification Techniques

PA dynamic verification is primarily based on the techniques of simulating a design dynamically for power related functionalities that adopt certain power dissipation reduction techniques through power intent or UPF. Evidently such verification requires a PA enabled functional testbench or stimulus to ascertain that the design will function exactly as intended in the real silicon implementation on-chip, as altered structurally and as defined in the power specification in UPF.

In addition to the PA enabled testbench, UPF, and design under PA verification, the PA dynamic simulation (PA-SIM) environment requires the PG-pin enabled standard Liberty libraries and non-standard simulation model libraries as discussed in Chap. 4. Although the requirements of these libraries differ, depending on design abstraction levels from RTL to PG-netlist, both of the Liberty (.lib) and simulation-models (.v) may not be required simultaneously for a design to be verified in a PA-SIM environment. The Questa® PA Simulation (PA-SIM) tool library requirements are summarized in Table 4.2 in Chap. 4.

The succeeding sections in this Chapter explain the techniques and requirements of PA dynamic simulation. Eventually the tool specific commands, options, verification environment setup, input requirements, output results, debugging methodologies, automated verification protocols and special features like power related state transition, transition coverage etc. are discussed in possible sequential steps in the current and subsequent Chapters. This will simplify to fully comprehend the PA dynamic simulation technology and adopt it proficiently in every chip design today.

5.2 PA Dynamic Simulation: Fundamentals

PA simulation has several distinctive features that differ from the regular functional simulation environments. One distinction is corruption semantics that allows the internal sequential elements and output signals to assign an unknown value during power down. Consequently, PA-SIM also incorporates power On-Off situations and sequences on various portions of the design based on the UPF specification but processed within the tool.

The corruption semantics which are actually initiated through the UPF –***simstate*** (discussed in Chap. 3), Liberty libraries, and PA-simulation model libraries (discussed in Chap. 4) are just the catalyst of corruption. The actual corruption depends on HDL semantics and PA-SIM internal instrumentation capabilities. It is worth noting that corruption semantics significantly impact overall PA simulation behavior, specifically simulation results and debugging techniques.

In a design under PA simulation, the corruption semantics are implied at the RTL through driver-based corruption using the UPF specification. And at the GL-netlist through Liberty based corruption. The tool usually corrupts the design's internal sequential elements and internal and (or) output signals.

This aims to imitate real world phenomena by changing the regular known values within the logics to unknown (1'bx or high impedance `hiz or 1'b0), in the event of power down or inadequate power supply to drive those logics correctly. So it actually defines the rules that decide how a logic element essentially behaves in response to reduction and disconnection of power. Hence corruption typically refers to the situation where the value of logic elements or signals becomes unpredictable and hence it's usually assigned to 1'bx (for 4-state logic) or 1'b0 (for 2-state logic) or `hiz, based on user preferences.

In PA-SIM environments, the corruption is typically applied to the drivers of a signal and will propagate to all sink fanouts of that signal, unless it is isolated from the source. When a power domain or, in general, a design instance is turned off, every sequential element within and every signal driven out from this particular power down instance will be corrupted.

So in general, as long as the power remains off, no additional activity takes place within the power down instance. In practice, it is the responsibility of the simulator to identify a driver from a design and apply corruptions accordingly. However the concept of a driver varies depending on the level of design abstraction. For instance, at RTL, GL-netlist, and PG-netlist, by default Questa® PA-SIM finds a driver and applies corruption based on the following rules.

Example 5.1 Sample Rules for Finding Drivers at Different Design Abstraction
At RTL,

- Any statements involving arithmetic or logical operations or conditional execution are interpreted as representing drivers, and Questa® PA-SIM applies corruption when powered down.
- Unconditional assignments and Verilog 'buf' primitives do not represent drivers and therefore do not cause corruption when powered down, but they do not isolate and may propagate corrupted signals from upstream drivers.

AT GL-netlist and PG-netlist,

- All cell instances are interpreted as containing drivers. As a result, buffer cell instances in a gate level netlist will cause corruption when powered down.

When the power of the design instance is turned on, PA-SIM automatically withdraws the corruption activities from all the sequential elements and signals within the power down instance. All conditional continuous assignment statement lists once again become sensitive to changes to their right-hand side expressions.

Similarly, any other combinational processes, such as an **`always_comb`** blocks, resume their normal sensitivity list operations. And it's essential to note that internal sequential elements are re-evaluated on the next clock cycle after power up. All continuous assignments and other combinational processes are evaluated at power up to ensure that constant values and current input values are properly propagated.

Traditionally the simulation tools were built with the assumption that the design captured in different HDL reference languages – like Verilog, SystemVerilog,

VHDL, etc. – will be powered on at the initiation of simulation cycles and remain powered on until the termination of all simulation events. However, this supposition does not comply with the current trends of design power requirements. Because of numerous features and functionalities that are integrated on a single chip today.

One of the several power dissipation reduction techniques, as discussed in Chap. 1, are usually adopted for these designs depending on their complexities and implementation objectives. The verification tools, including the dynamic simulator, need instrumentation to extend their capabilities in order to comply with such multitude of power design functionalities and one or more variant of power dissipation reduction techniques incorporated in such designs.

The traditional power On assumption of HDL during the simulation cycle and PA-SIM corruption semantics together introduce a special impact on the behavioral portion of code in the design under verification. Specifically the "initial begin ~ end" block in the behavioral portion is executed only once by the non-PA as well as the PA simulator and preserves the evaluated state or values. Hence the behavioral code in a design during PA-SIM requires special care because of corruption semantics.

The behavior code must be identified beforehand and needs to put in the always-on power domain. The reason is that often such code consists of initial blocks and, during a power down session, the variable within these initial blocks is corrupted and even after power On or resume these corruption semantics are conserved. Hence it is obligatory for the PA-SIM tool to devise a mechanism to identify behavioral code in order to isolate them in always-on domains or a device retriggering mechanism for the initial block to reinitialize after the power restore session.

The Questa® PA-SIM dynamic simulation verification environment supports all variants and any combinations of the power reduction techniques that can be adopted in any design today. This ranges from SoC, ASIC, MCU to the processor core and in any level of design abstraction, starting from RTL, to post-synthesis GL-netlist as well as the post-layout non-PG or PG-netlist.

The PA instrumentation of Questa® PA-SIM makes it possible to simulate all power On-Off situations or sequences and apply corruption semantics accordingly through inferring virtual power domains, their associated supply architectures, power states, state transitions, and different power strategies (ISO, ELS, RFF, PSW etc.) from the UPF specification, even when they are not physically inserted in the design. These provide the flexibility to abstract the complete power management infrastructure within the simulator, based on the power-specification, and allows evaluation of how the design will behave exactly on silicon, from a very early stage of design-implementation cycles.

Questa® PA-SIM fundamentally provides the virtual inferring of UPF objects, parameters, and strategies at the RTL; however such capabilities are also extended for mixed-RTL when only a few of the MV or Macro cells are instantiated. It is further extended in the GL-netlist, when PSW or RPT are not implemented yet. However appropriate power annotated testbenches are required on top of the regular functional verification testbenches, in order to simulate the complete power management architecture, either virtually inferred or physically present in the design. The details of a PA testbench are discussed in succeeding sections.

5.3 PA Dynamic Simulation: Verification Features

As discussed on several occasions in previous Chapters, PA dynamic (as well PA-Static) verification is required in every design phase in DVIF as shown in Fig. 5.1 below. Questa® PA-SIM provides dynamic simulation capabilities in all design abstraction levels – from pre and post-synthesis up to the post P&R netlist.

The input requirements, in general, for PA-SIM at the RTL are the constraint and configuration UPF, PA augmented testbench, and design under verification in any format of HDL. Once the PA dynamic simulation passes, the design usually goes through the synthesis process and the GL-netlist is generated with actual standard, MV and Macro cells inserted in the netlist. The UPF is also extended with implementation related information before and after the synthesis process.

The PA-SIM input requirements at the post-synthesis level are the GL-netlist, extended implementation UPF, and Liberty standard or PA-Simulation Model libraries. Only the testbench can be reused as is from the RTL PA-SIM environment. Hence fundamental PA-SIM verification objectives at the GL-netlist level is the same as at the RTL. However, the physical presence of MV and Macro cells initiates auxiliary verification based on the control, enable, clock, and reset, as well as register related synthesized information in some cases.

Further, after P&R, the PG-netlist is generated with the detailed power switch network, always on connectivity, and power, ground, and bias connectivity for all

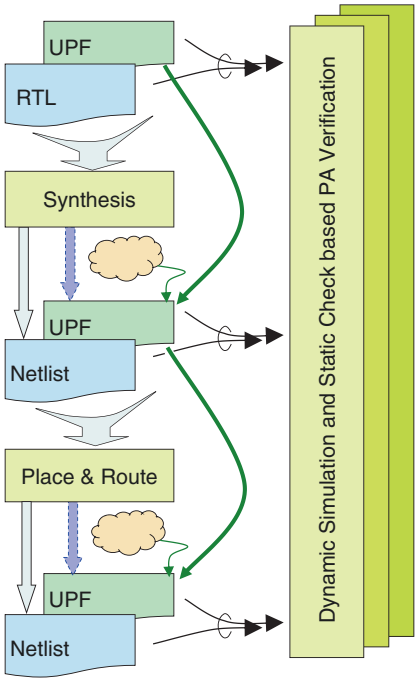

Fig. 5.1 PA dynamic simulation and PA-static verification requirements for different levels of the Design Verification And Implementation Flow (DVIF)

the cells. Obviously, UPF at P&R is supplemented with detailed implementation technology dependent parameters, like PSW and PG connectivity for all MV and Macro cells. But for the PG-netlist, a PA-SIM with UPF is optional, because all the PG connectivity of all the MV and Macro cell information is already available in the PG-netlist. Hence the PG-netlist, testbench, and PA-Simulation Models that include functionality for the cells are sufficient to perform PA dynamic simulation even and ideally, with any non-PA regular simulator.

However, it is often required to cross compare the dynamic simulation results at post-synthesis with P&R results, specifically when PSW, feed-through buffers, and always-on cells are not placed yet; hence, optionally, non-PG-netlists are generated at the P&R level. The non PG-netlist and GL-netlist simulation through PA-SIM must be conducted with exactly same inputs, like same Liberty, same PA-Simulation Model libraries etc.

So for the PA design verification criterion, input requirements as well as output results from the tool widely varies depending on the design abstraction level. Even a new degree of verification variations are added on top, based on the adoption of power dissipation reduction techniques discussed in Chap. 2. Nevertheless, the PA-SIM verification criterion are immensely extensive, complex, and often have design implementation specific aspects to consider. But still as a general rule of thumb, it is often required to verify the following PA features in PA-SIM to ascertain whether-

List 5.1 Generic Requirements for PA Dynamic Verification

- The UPF strategies (ISO, LS, ELS, RFF, PSW, RPT etc.) are working properly,
- The power intent specified in UPF are correct and matching with the strategies implemented in the design,
- The power On-Off sequences are in the expected order and are enabling or disabling the required UPF strategies,
- The control signals of UPF strategies like ISO and ELS enable, RFF save and restore, PSW control and acknowledgement signals are in proper order to the power On-Off sequences,
- The state and data retention register values are retaining accurate state and data,
- The entire design is reinitiated (initial blocks) properly on the power resume from power down state,
- The behavioral portion of code in the design is separated in the always-on domain,
- The registers and sequential elements are reset on power resume,
- The power state transitions are in proper sequence,
- The state transitions are fully under coverage.

Questa® PA-SIM provides dynamic verification for all the features mentioned above. More specifically, with a PA annotated and properly designed testbench, PA-SIM will assert automated power aware assertions, also known as dynamic sequence checkers, through a built-in automated messaging system. These sequence

assertions are spontaneous and usually fire only in the event of occurrence of a power related anomalies in simulation. For instance, during simulating the save and restore sequences, as well as verifying accurate data restoration from retention registers after power resumes, the Questa® PA-SIM will fire assertions for the following categories along with the PA annotated simulation results that are possible to dump through HDL system tasks in wave-shapes.

List 5.2 Sample of Power Sequence Assertions

- **Power Off Assertions** – During the power down of the power domain, if retention conditions (save) are not asserted, the tool generates assertion for "Retention Condition not asserted".
- **Power On Assertions** – During the power On or resume of the power domain, if retention conditions (restore) are not asserted, the tool generates an assertion for "Retention Condition not asserted", as well as "power On and retention condition are not in proper sequence".
- The above two assertions work for both the Master-Slave and Balloon-Latch configuration of retention registers equally, discussed in this chapter.
- **Clock or Latch Enable Assertions** – During power down, power On, or resume for the Master-Slave type RFF configuration, this assertion flags when the retention condition (save or restore) is not at the required level (1'b1 or 1'b0) to trigger retention functionalities.
- However, for the Balloon-Latch configuration, it is often required to set clock or latch enable to a certain specific value (1'b1 or 1'b0) during save or restore occurrences, in order to avoid potential race conditions within the retention register stored data.
- **Clock Toggle Assertions** – During the event of a power down, it is required that the clock also stops toggling. Although such assertions are universally true for any power down situation regardless of retention strategies, however clock toggling may corrupt internally retained data, hence this assertion is targeted to points out if the clock toggles during power down and retention.

These are just a few among several automated assertions available for retention simulation alone, including retention condition off at save (enabling retention) or at restore (disabling retention), retention condition toggling, and so on.

Obviously, similar abundant resources of automated assertions are required in PA-SIM environments for most of the UPF strategies, including ISO, PSW, ELS, and LS as well as for their control, enable, and acknowledgement signals. Since the PA-SIM built-in automated assertions internally coordinate with the testbench, the resultant messages always accompany the progression of exact simulation time stamps. Hence they always remain as a popular engineering choice for debugging along with the PA annotated dumped wave-shapes. The main purpose of these automated assertions are primarily to uncover various power sequencing anomalies; hence they are also considered to be distinctive and obligatory features of PA-SIM, in contrast to a regular functional (logic) and non-PA dynamic verification simulator.

5.4 PA Dynamic Simulation: Verification Practices

The foundation of PA verification methodologies, power reduction techniques, library requirements, special tool features, and tool instrumenting artifacts have already been established in previous Chapters and Sections. This section elaborates on the Questa® PA-SIM verification environment set up through the commands, options, and techniques required to accomplish a successful PA dynamic simulation of a design. The fundamental flow of PA-SIM consists of design compile, analyze, build, and execute (or run) cycle, similar to any regular or Questa®-SIM for non-PA functional verification.

It is already evident that PA dynamic simulation, or verification in general, imposes several additional layers of tool complexity and requirements. Specifically, any PA simulator requires to analyze the UPF and overlay all of the required power instrumentation on the design. This includes inferring power domain hierarchies with certain design instances as well as inferring different UPF strategies (like ISO, LS, ELS, PSW, RFF, etc.), power supply, and power network architecture as discussed in Sect. 1. With the progression of design implementation, the simulator also needs to coordinate with libraries and power-ground (PG) pin connectivity information.

Fortunately Questa® PA-SIM simplifies all PA related difficult obstacles through instrumenting PA features under the hood and incorporating the same simple three-steps (compile, optimize, and simulate) flow, used in non-PA Questa®-SIM environment. Hence it is only required to substitute PA related special commands and options in an already running non-PA functional verification environment. The PA related special commands and options are based on several aspects, and a few are listed below.

List 5.3 Generic Criterion for Using PA-SIM Commands and Options
- Verification objectives and extents,
- Input requirements for the tool,
- Contents and extents of output results,
- Debugging capabilities inclusion and so on.

There are different variations of simulation flows available in regular Questa®-SIM; however Questa® PA-SIM supports the “Standard Simulation Flow” for PA dynamic verification.

The standard simulation is a three-step flow and is also standard for PA-SIM. In general, PA-SIM requires a complete HDL representation of the design in Verilog, SystemVerilog, or VHDL, or any combination of these languages. And it may be synthesizable RTL code, behavioral RTL code, a gate-level netlist, or any combination of these forms. The first step is to compile the design through `vlog` for Verilog and SystemVerilog and `vcom` for VHDL commands, depending on the design available in different HDL formats.

The next is optimizing the compiled design through the `vopt` command. It is worth mentioning here that `vopt`, essentially meant for performing design optimi-

zations to maximize simulator performance at the cost of limiting the visibility of certain design objects that may not be essential for debugging during a particular pass. However, it sometimes requires making tradeoffs between debug-ability and run time performance, depending on verification circumstances and adopted verification methodologies, for example overnight regression runs.

Unlike the non-PA Questa® SIM environment, `vopt` for PA-SIM processes the UPF power intent specifications and Liberty library and accepts all other power related verification commands and options as arguments.

The final step is executing the simulation through the `vsim` command, which primarily applies PA semantics to the design and runs the PA simulation on the optimized design. It also allows certain power related commands and options as arguments. The typical Questa® PA-SIM commands and options format for a Standard PA Simulation flow is shown below.

Example 5.2 Typical Command Format for Standard PA Simulation Flow

Compile: `vlog -work` work `-f` design_rtl.v
Optimize: `vopt -work` work \
`-pa_upf` test.upf \
`-pa_top` "top/dut" \
`-o Opt_design` \
<others PA commands>
Simulate: `vsim Opt_design` \
`-pa_lib` work \
<others PA sim control command>

In the standard simulation three-step flow, the `-o` arguments at `vopt` actually generate an optimized design `<Opt_design>`. In the next step at `vsim`, the optimized design `<Opt_design>` is actually invoked to perform simulation on that `<Opt_design>`.

Apart from the fundamental PA-SIM execution phases and procedures discussed above, in practice PA-SIM requires special procedures to coordinate with several other PA aspects; including corruption, behavioral code identification, re-initialization of certain design elements, power relevant simulation events, reset and register initialization after power resume, and so on. One of these special procedures is discussed below.

Re-initialization PA-SIM provides a mechanism that allows the re-evaluation of a Verilog HDL initial block, through the UPF augmentation on **set_design_attributes** commands, as follows.

Example 5.3 Example for Initial Block Re-evaluation

set_design_attributes *-attribute* qpa_replay_init **TRUE**
-elements {top/bot1/sram}

To note, the **set_design_attributes** *-attribute* qpa_replay_init **TRUE** with either ***–elements*** { } or ***–models*** options will serve the same purpose. Here in Example 5.3 the command takes arguments to specify the module or instance in ***–elements*** **{}** whose initial blocks are re-triggered at power up. However, any initial block con-

taining forever, fork join, break continue, wait, disable, and rand sequence statements will not be re-evaluated.

The following UPF commands with the special "***UPF_dont_touch***" attribute also allow excluding any specified module, architecture, entity instance, and signals from power aware behavior from re-triggering on power resume or On. This PA behavior exclusion disables power aware instrumentation for corruption, retention, isolation, level_shifter, or buffers on the specified item.

Example 5.4 UPF Command for Customizing Design for PA Attributes
set_design_attributes and
set_port_attributes

Recalling the syntax of these UPF commands as explained in Example 3.67 in Chap. 3, the following example represents disabling of a design module from initial block re-evaluation PA semantics.

Example 5.5 Sample for Excluding a Design Module from Regular PA Semantics
set_design_attributes *-attribute UPF_dont_touch* **TRUE** *-models* alu_*

As mentioned earlier the library plays a vital role in PA-SIM. The succeeding section will discuss PA-SIM library processing fundamentals in combination with the explanation in Chap. 4.

5.5 PA Dynamic Simulation: Library Processing

Chapter 4 explained the fundamental construction, functionalities, and attributes of libraries from a PA verification perspective in a generic way. However, in the case of a real design project, PG-pin Liberty and PA-Simulation Model libraries may not be readily available from the beginning. As well, construction and attributes of non-standard simulation model libraries may varies widely. Even at times a standard Liberty library may lack all the required attributes for PA verification. Hence it often depends on the verification tool to process the missing part of libraries and generate ad hoc functionalities within the tool to complete the verification.

On the other hand, when multiple libraries are present in a verification environment, the simulator decides the order of precedence for processing corruptions. But the corruption processing criterion used by a simulator for Macro cells and all other cell libraries widely varies because of the nature of their construction and the visibility of logic and functionalities within those cells.

Although Table 4.2 in Chap. 4 summarizes the library requirements for Questa® PA-SIM for different levels of DVIF, this Section further elaborates on library processing in PA-SIM verification tools and environments. For PA verification, all four library types – PG-pin Liberty (.lib), non-PA, PA, and Extended-PA-Simulation

Model (.v) – libraries may become relevant and useful. Although the requirements of these libraries differ, depending on design abstraction levels from RTL to PG-netlist, as already mentioned, both the Liberty (.lib) and simulation-models (.v) may not be required simultaneously for a design in a PA simulation environment. Hence it is also important to know how Questa® PA-SIM processes libraries and specifies the order of precedence when considering only a particular type or when multiple types of libraries are available in a simulation environment.

When both PA-SIM Model lib (.v) and Liberty (.lib) are available, the Questa® PA simulator allows the PA-SIM Model (.v) to take precedence and corrupt internals or outputs of the cell based on its own power down function (detailed in Sect. 4.2). But when only a Non-PA-SIM Model (.v) is available, PA-SIM initiates driver or UPF ***-simstate*** based corruption.

On the other hand, when only (.lib) is available and gets the highest precedence, Questa® PA-SIM deploys a new set of analytical approaches for the (.lib) to proceed with corruption. At first PA-SIM searches for cell-level attributes to identify if the cell is a Macro through the `is_macro_cell:true` attribute. If this attribute is absent or different, then obviously the cell is not a Macro, hence Questa® proceeds with driver based corruption.

But when the cell is a Macro, then Questa® corrupts the input ports with either of the following, in the listed order of precedence, depending on their availability: (1a). `related_power/ground_pin : VDD/VSS;`

(1b). `related_bias_pin : "VNW VPW";`

For the output ports of the same Macro cell, Questa® looks for the following attributes also in the listed order of precedence and dependent on their availability:(2a). `power_down_function : "!VDDO+(!VDD&EN)+VSS+VPW+!VNW";`

But if the power_down_function is absent, the tool searches for the following attributes again in the order they are listed: (2b). `related_power/ground_pin : VDD/VSS ;`

(2c). `related_bias_pin : "VNW VPW";`

And the tool internally generates the new `power_down_function` from (2a) and (2b) with the following assumptions:(~related_power_pin+related_ground_pin+~related_bias_pin+ related_bias_pin)

Even Questa® PA-SIM extends Macro library processing flexibility further, in case the related or bias PG-pin (2b) and (2c) attributes are missing. Hence Questa® will proceed with corruption with the bias pins only from their corresponding `pg_pin` and `pg_type` attributes, as shown: (3a). `pg_pin(VNW) pg_type : nwell;`

(3b). `pg_pin(VPW) pg_type : pwell;`

Essentially, Questa® PA-SIM addresses numerous, possible complex combinations of Liberty, non-PA, PA, and Extended-PA simulation model libraries, with extended flexibility to afford accurate PA dynamic simulation even when the Liberty syntax and attributes are inadequate.

5.6 PA Dynamic Simulation: Testbench Requirements

It is evident from the PA dynamic simulation and verification features, functionalities, practices, and PA-SIM input requirements discussed in previous sections that a special UPF or PA annotated testbench is required on top of a regular functional testbench. The PA annotation on the regular testbench is done in order to manipulate the power supply networks that are inferred and overlaid on the design by the simulation tool through the UPF analyze process explained in Sect. 3.

The top level UPF supply ports and supply nets provide hooks for the design, libraries, and annotated testbenches through the UPF **`connect_supply_net`** and **`connect_supply_set`** commands. Thus the complete power network connectivity in a PA-SIM environment are established. The top level UPF supply ports and supply nets are collectively known as supply pads or supply pins (e.g. `VDD, VSS` etc.). The IEEE 1801 standard recommends that supply pads may be referenced in the testbenches and extended to manipulate power network connectivity in a PA-SIM.

Hence it becomes possible to control power On and Off for any power domain in the design through the supply pad referenced in the testbench. The HDL testbench annotations are done through importing UPF packages available under the PA simulator distribution environment. The IEEE 1801 LRM provides standard UPF packages for Verilog, SystemVerilog, and VHDL testbenches to import the appropriate UPF packages to manipulate the supply pads of the design under verification in PA-SIM. The following are syntax examples for UPF packages to be imported or used in different HDL variants.

Example 5.6 UPF Package for Verilog and SystemVerilog Testbench

```
import UPF::*;
module testbench;
...
endmodule
```

Example 5.7 UPF Package for VHDL Testbench

```
library ieee;
use ieee.UPF.all;

entity dut is
...
end entity;
architecture arch of dut is
begin
...
end arch;
```

For Verilog or SystemVerilog testbenches, UPF packages can be imported within or outside of the `module-endmodule` declaration. The "`import UPF::*`" package and "`use ieee.UPF.all;`" library actually embeds the functions that are used

to utilize and drive the design supply pads directly from the testbench. Once these packages are referenced in the testbench, the simulator automatically searches for them from the simulator installation locations and makes the built-in functions of these packages available to utilize in the PA-SIM environment. The following examples explain these functions, namely `supply_on` and `supply_off` with their detail arguments.

Example 5.8 Functions for Verilog and SystemVerilog Testbench to Drive Supply Pad

```
supply_on( string pad_name, real value = 1.0, string file_info = "");
supply_off( string pad_name, string file_info = "" );
```

It is important to note that the user do not requires to consider about the third argument; `string file_info = "",` will be automatically processed by the PA-SIM verification tool.

Example 5.9 Functions for VHDL Testbench to Drive Supply Pad

```
supply_on ( pad_name : IN string ; value : IN real ) return
boolean;
supply_off ( pad_name : IN string ) return boolean;
```

The `pad_name` must be a string constant, and a valid top level UPF supply port must be passed to this argument along with a "non-zero" real value to denote power On, or "empty" to denote power Off. PA-SIM obtains the top module design name from the UPF **set_scope** commands.

The succeeding sections explain the mechanism for controlling the design supply pad through a testbench combined with the **connect_supply_net** or **connect_supply_set** specification in UPF. Recalling the implementation snippet UPF example from Chap. 3, its **connect_supply_net** for supply port and supply nets are reused here.

Example 5.10 UPF with connect_supply_net for Utilizing Supply Pad from Testbench

set_scope cpu_top
create_power_domain PD_top
……
IMPLEMENTATION UPF Snippet
Create top level power domain supply ports
create_supply_port VDD_0d99 *-**domain*** PD_top
create_supply_port VDD_0d81 *-**domain*** PD_top
create_supply_port VSS *-**domain*** PD_top
Create supply nets
create_supply_net VDD_0d99 *-**domain*** PD_top
create_supply_net VDD_0d81 *-**domain*** PD_top
create_supply_net VSS *-**domain*** PD_top
create_supply_net VDD_0d81_sw *-**domain*** PD_mem_ctrl
Connect top level power domain supply ports to supply nets
connect_supply_net VDD_0d99 *-**ports*** VDD_0d99

connect_supply_net VDD_0d81 ***-ports*** VDD_0d81
connect_supply_net VSS ***-ports*** VSS

The UPF **connect_supply_net** specified supply ports for example, VDD_0d99, VDD_0d81, VSS etc. can be directly driven from the testbench shown in the following example.

Example 5.11 Testbench for Driving Supply Pad

```
import UPF::*;
module testbench;
...
reg VDD_0d99, VDD_0d81, VSS;
reg ISO_ctrl;
...
initial begin
#100
ISO_ctrl = 1'b1;
supply_on (VDD_0d99, 1.10); // Values represent voltage & non zero
value
                            // (1.10) signifies Power On
supply_on (VSS, 0.0);     // UPF LRM Specifies Ground VSS On at 0.0
...
#200
supply_on (VDD_0d81, 1.10);
...
#400
supply_off  (VDD_0d99);  //  Here  empty  real  value  argument
indicates
                           // Power Off
...
end
endmodule
```

Hence it is possible to design a voltage regulator (VR) or a power management unit (PMU) on the testbench through the functions `supply_on` and `supply_off` to mimic a real chip power operation. There are more functions available under these UPF packages, and their use models are discussed in succeeding sections.

5.7 PA Dynamic Simulation: Custom PA Checkers and Monitors

Although Questa® PA-SIM provides a wide range of automated assertions in form of dynamic sequence checkers that cover every possible PA dynamic verification scenarios. However, design specific PA verification complexities may arise from

adoption of one or multiple power dissipation reduction techniques, from multitude of design features, and from target implementation objectives. Hence apart from tool automated checks and PA annotated testbenches, additional and customized PA assertions, checkers, and their monitors are sometimes required to be incorporated in the design.

But a design may already contain plentiful assertions from functional verification parts, often written in SystemVerilog (SVA) and bind with the language bind construct. SystemVerilog provides a powerful bind construct that is used to specify one or more instantiations of a module, interface, program, or checker without modifying the code of the target. So, for example, instrumentation code or assertions that are encapsulated in a module, interface, program, or checker can be instantiated in a target module or a module instance in a non-intrusive manner. Still, customized PA checks, assertions, and monitors are often anticipated to keep separate, not only from the design code but also from functional SVA.

UPF provides a mechanism to separate the binding of such customized PA assertions from both functional SVA and design. The UPF **bind_checker** command and its affiliated options allows users to insert checker modules into a design without modifying and interfering with the original design code or introducing functional changes. However, UPF inherits the mechanism for binding the checkers to design instances from the SystemVerilog bind directive. Hence similar to SVA, the UPF **bind_checker** directive causes one module to be instantiated within another without having to explicitly alter the code of either. This facilitates the complete separation between the design implementation and any associated verification code.

Signals in the target instance are bound by position to inputs in the bind checker module through the port list, exactly the same as in the case for SVA bindings. Thus, the bind module has access to any and all signals in the scope of the target instance by simply adding them to the port list, which facilitates sampling of arbitrary design signals.

The UPF **bind_checker** syntax and "use models" to create custom PA assertions for a design and bind the checker through UPF **bind_checker** are shown in detail in the following four successive examples.

Example 5.12 UPF bind_checker Syntax

bind_checker < instance_name> \
    -module <checker_name> \
    -elements <element_list> \
-bind_to module [**-arch** name]
-ports {{port_name net_name}*}

In the syntax, the < instance_name> is the "instance" name (e.g. iso_supply_chk) of the checker module <checker_name> (e.g. `ISO_SUPPLY_CHECKER`). The **–elements** <elements_list> is the list of design elements where the checker "instance" will be inserted. The **–module** <checker_name> is the name of a SystemVerilog module for which the verification code is targeted. The verification modules are generally coded in SystemVerilog, but bind to either a SystemVerilog or VHDL instance through **-bind_to module** [-arch name].

Also to note that, **-ports**{} are the association of design signals to the checker ports. The <net_name> argument accepts the symbolic references for signals, power supply ports, supply nets, and supply sets defined in UPF for various UPF strategies. For example, among others, just as representative, **isolation_signal** or **retention_power_net** can be referenced as follows.

Example 5.13 <net_name> Symbolic Referencing for Various UPF Strategies

`<design_scope_name>.<powerdomain_name>.<iso_stratgy_name>`**.isolation_signal**

`<design_scope_name>.< powerdomain_name>`**.retention_power_net**

Example 5.14 A Customize Checker Sample for ISO Control Related Assertion

```
module ISO_SUPPLY_CHECKER(ISO_CTRL,ISO_PWR,ISO_GND);
import UPF::*;
input ISO_CTRL;
input supply_net_type ISO_PWR;
input supply_net_type ISO_GND;
reg ISO_pg_sig;
assign ISO_pg_sig = get_supply_on_state(ISO_PWR) && \
get_supply_on_state(ISO_GND);
always @(negedge ISO_pg_sig)
assert(!(ISO_CTRL)) else \
$display("\n At time %0d isolation supply is switched OFF \
during isolation period, ISO_CTRL=%b", $time, ISO_CTRL);
endmodule
```

Example 5.15 Snippet of UPF Code for Binding the ISO_SUPPLY_CHECKER Assertion

set_scope /tb/TOP
create_supply_net ISO_PWR
create_supply_net ISO_GND
create_supply_port ISO_PWR_PORT
create_supply_port ISO_GND_PORT
connect_supply_net ISO_PWR ***-port*** ISO_PWR_PORT
connect_supply_net ISO_GND ***-port*** ISO_GND_PORT
create_supply_set ISO_SS ***-function*** {***power*** ISO_PWR} \
    -function {***ground*** ISO_GND}
create_power_domain PD_mid1 ***-supply*** {***primary*** ISO_SS}
set_isolation iso_PD_mid1 \
    -domain PD_mid1\
    -applies_to outputs\
    -isolation_supply_set ISO_SS\
    -location self\
    -isolation_signal ctrl

```
## The ISO_SUPPLY_CHECKER Checker binding in UPF
bind_checker iso_supply_chk \
        -module ISO_SUPPLY_CHECKER \
        -bind_to mid_vl \
        -ports {\
        {ISO_CTRL PD_mid1.iso_PD_mid1.isolation_signal} \
        {ISO_PWR ./ISO_SS.power} \
        {ISO_GND ./ISO_SS.ground}
}
```

Example 5.16 Design Completely Separate from Checker and Binding

```
module tb();
...
top top(...);
...
endmodule

module top(...);
mid_vl test1_vl(...);
mid_vl test2_vl(...);
mid_vl test3_vl(...);
endmodule;

module mid_vl(...);
...
endmodule
```

The combined Examples 5.14, 5.15, and 5.16 shown above respectively explains how to design a PA custom assertion, how to bind such assertions in UPF, and how the target design totally separates the assertions and its bindings. It is worth noting that the assertions sample in Example 5.14 imports IEEE standards package `import UPF::*;` similar to a PA annotated testbench (discussed in Sect. 6), in order to utilize a different types of function shown below.

Example 5.17 UPF Import Package Function for Custom Checker

```
get_supply_on_state( supply_net_type arg );
```

This function actually is used for driving and providing the connectivity for `supply_net_type ISO_PWR;` and `ISO_GND;` from the checker module, for the design instance `mid_vl` which is under the **`-bind_to`** command. The PA-SIM access `mid_vl` module with the `(ISO_SUPPLY_CHECKER)` checker available under the `\tb\top` hierarchical paths of the design and defined through the **set_scope** commands in UPF is shown in Example 5.15. Thus it is distinctive that UPF provides a powerful mechanism to define a custom PA assertion and provide a layer to completely separate it from design codes, by embedding the binding within the UPF file.

5.8 PA Dynamic Simulation: Post-Synthesis Gate-Level Simulation

The post-synthesis gate-level netlist (GL-netlist) based PA simulation input requirements are mostly the same as RTL simulation. However, the design under verification here is the GL-netlist from synthesis, so logic gates from standard, MV and Macro cell Liberty libraries are already inserted or instantiated in the design. Hence PA-SIM at post-synthesis also requires Liberty libraries as input in order to accumulate different cell-level attributes and power down functions. The tool utilizes these attributes and functionalities of Liberty for the following.

List 5.4 Different Liberty Attributes Required at GL-Netlist PA-SIM
- Identifying a cell,
- Conduct power aware simulation based on the cell identification along with UPF specification, and
- Apply appropriate corruption semantics accordingly.

One significant aspect of PA-SIM at the GL-netlist is that all cell instances are interpreted as containing drivers because these cells are usually leaf level cells or they are an instance that has no descendants. As a result, buffer cell instances in a GL-netlist will cause corruption when powered down. However recall that, as discussed in Sect. 2, this is in contradiction to RTL Verilog 'buf' primitives that do not represent drivers and therefore out of scope from corruption when powered down (at RTL).

During GL-netlist power aware simulation, corruption will occur on the output ports and sequential logic of any detected gate-level cells. In addition, power aware simulation automatically treats a module as a gate-level cell if the module contains the `celldefine attribute or the `specify blocks in HDL code. Even these cells are not defined in the Liberty syntax; the standard processing of driver-based corruption is still applied to these cells, similar to that for RTL cell designs.

UPF 1801-2013 or UPF 2.1 LRM provides a dominant mechanism to devise a driver-based corruption on any HDL cell, even when there is no `celldefine or `specify block, through the **set_design_attribute** [-attribute {name value}]* command. PA-SIM treats all module, entity, or design elements as a gate-level or leaf cell when the following syntax is applied on them.

Example 5.18 Leaf-level or Gate-level Cell Treatment of Design for Driver-Based Corruption

Define through UPF File:
set_design_attributes -models FIFO **-attribute** {**UPF_is_leaf_cell** TRUE}
Define through HDL Annotation:
SystemVerilog or Verilog Attribute Specification:
(* **UPF_is_leaf_cell=**"**TRUE**" *) module FIFO (<port list>);
VHDL Attribute Specification:

attribute **UPF_is_leaf_cell** : STD.Standard.String;
attribute **UPF_is_leaf_cell** of FIFO : entity is "TRUE";

Though the latest UPF 1801-2015 or UPF 3.0 LRM revised the syntax for leaf-level or gate-level cell definition to enable driver-based corruption through **UPF_is_hard_macro** instead of **UPF_is_leaf_cell** attributes, the semantics and use model remain identical.

During GL-netlist power aware simulation, apart from detecting the standard and Macro cells and applying corruption accordingly, the simulator also required to automatically identify special power management or MV cells, like ISO, LS, RFF, etc. from the design. The detection of MV cells is primarily done through the cell level attributes available in corresponding Liberty libraries and are usually cross-compared with the corresponding definition of strategies in UPF. Recalling the syntax and example of ISO, LS, and RFF from Chap. 3, and from the fact that a GL-netlist contains at least ISO, LS, and RFF through synthesis, hence most of these cells are already specified either through following UPF commands and options or through tool auto detection processes.

Example 5.19 ISO, LS, RFF Automatic Detection in GL-netlist Simulation through UPF Command

ISO cells:set_isolation strategy_name [***-instance*** {{instance_name port_name}*}]
Where the <`instance_name`> is a technology leaf-cell instance and the <`port_name`> of the logic port that it isolates.

LS cells:set_level_shifter strategy_name ***-instance*** {{instance_name port_name}*}
Similarly here the <`instance_name`> is a technology library leaf-cell instance and the <`port_name`> of the logic port that it level-shifts.

RFF cells:set_retention retention_name ***-instance*** {{instance_name[signal_name]}*}

Here in this case as well the <`instance_name`> is a technology library leaf-cell instance and the optional <`signal_name`> is the HDL signal that controls retention. If this instance has any unconnected supply ports or save and restore control ports, then these ports need to have identifying attributes in the cell model, and the ports shall be connected in accordance with this **`set_retention`** command.

In the GL-netlist PA simulation, the tool's auto detection process of MV cells actually refers to the cells that are not specified through the ***-instance*** but through Liberty or other attributes. Hence for the rest of the cells that are not specified in the UPF file, PA-SIM during GL-netlist simulation automatically detects the right UPF strategy to which they belong and treats them in a similar way to cells of that strategy specified with an ***-instance*** argument. Questa® PA-SIM detects power management cells based on one of the following information.

List 5.5 Liberty Cell-Level Attributes

- `is_isolation_cell`
- `is_level_shifter`
- `retention_cell`

List 5.6 Library Cell Name from UPF Commands

- **map_isolation_cell** isolation_name [***-lib_cells*** lib_cells_list]
- **map_level_shifter_cell** level_shifter_strategy [***-lib_cells*** list]
- **map_retention_cell** retention_name_list [***-lib_cells*** lib_cell_list]
- **use_interface_cell** ***-strategy*** list_of_isolation_level_shifter_strategies [***-lib_cells*** lib_cell_list]

To note that **map_isolation_cell** and **map_level_shifter_cell** are deprecated from UPF LRM 3.0 with **use_interface_cell** command. Unlike **map_isolation_cell** and **map_level_shifter_cell**, the **use_interface_cell** can be used to manually map any isolation (ISO), level-shifter (LS), or combined isolation level-shifter (ELS) cells.

List 5.7 UPF name_format Command for Defining Names of Implicit Objects

- [-isolation_prefix string]
- [-isolation_suffix string]
- [-level_shift_prefix string]
- [-level_shift_suffix string]

List 5.8 Synthesis Pragmas

- isolation_upf
- retention_upf

Although the PA GL-netlist simulation does not have any exceptions from the fundamental concept of PA-SIM at the RTL. However, the tool procedure requires additional commands to process information discussed and listed above.

Tool Procedures for Liberty Processing at GL-netlist:

Compile: No Change

Optimize: vopt- requires to include either “vopt-pa_libertyfiles” or “vopt -pa_dumplibertydb”

Simulate: No Change

The following list explains the Liberty library referencing methods for GL-netlist PA-SIM.

List 5.9 Liberty Referencing in PA-SIM at GL-netlist

- pa_libertyfiles-

  Specifies the Liberty files to read. It is also possible to specify multiple files by separating file names with a comma.
  e.g. vopt –pa_libertyfiles=a.lib,b.lib

- pa_dumplibertydb-

  Specifies the name of the Liberty attribute library database for future reference.
  e.g. vopt -pa_dumplibertydb=lib_datafile

Apart from detecting standard, Macro and MV cells, the PA-SIM also is required to virtually infer missing MV cells in the design. In general the virtual inferring process is limited to RTL where physical MV cells are not instantiated yet. Inferring may also be required at Mixed-RTL, where some of the MV cells are still missing. Hence during GL-netlist, such virtual inferring is redundant. However, PA-SIM provides user controllability through tool procedures where it is possible to control the inference.

Tool Procedures for Controlling Virtual Inferring of MV Cells:
Compile: No change
Optimize: -vopt requires to add one of the following to disable auto inference.

"vopt -pa_disable=insertiso"- Disable ISO cell insertion
"vopt -pa_disable=insertls" - Disable LS cell insertion
"vopt-pa_disable=insertret" - Disable RFF cell insertion

Simulate: No Change

Using one of the above, based on requirements, will allow the tool not to infer the appropriate cells virtually at any design abstraction level. But since physical MV cells are already inserted in post-synthesis GL-netlist designs, hence using a tool procedure during optimization as follows will instruct the tool to disable all the three virtual insertions for ISO, LS, and RFF at once.

Required Tool Procedure for GL-netlist PA-SIM
Optimize: vopt- requires to include "vopt -pa_gls"

Hence the PA-SIM with GL-netlist as well as Mixed RTL mechanisms can be summarized as follows:

List 5.10 Summarization of PA-SIM Mechanism for GL-netlist and Mixed-RTL Design

- Detect standard and Macro cells in the netlist and perform corruption based on the driver, UPF strategies, or power down functionalities from Liberty.
- Detect MV cells in the netlist and match them with corresponding UPF strategies at that point in the design.
- Virtually infer MV cells if missing in the netlist based on UPF strategies
- Conduct automated power aware sequence checks and testbench based simulation similar to RTL PA-SIM

So once the cell detection or inferring process is completed as discussed above, the tool conducts power aware simulation on the GL-netlist similar to the RTL design. Although the Liberty file is required as additional input in the GL-netlist, however it is recommended to use the same testbench from RTL stage to conform verification consistency.

This chapter is dedicated to focus on UPF-based PA dynamic simulation from fundamentals to advanced topics, like verification practices, automated sequence checkers, library and testbench requirements and their processing techniques, as well as PA simulation requirements for GL-netlists. The next section will further emphasize the representation techniques of PA dynamic simulation results for efficient

debugging, based on all the verification artifacts discussed so far in this Chapter. It is obvious that the efficient debugging process leverages a bug free design in terms of power specification, power intent, power architecture, and power sequences. As mentioned, the primary objectives of the results are debugging and confirming that the verification is completed through incorporating PA annotated testbenches, tool automated sequential checkers, custom checkers, and so on.

5.9 PA Dynamic Simulation: Simulation Results and Debugging Techniques

The PA-SIM reporting and messaging formats as well as simulation wave-shape dumps through the system tasks differ widely from the regular non-PA-SIM verification environment. This is because the verification objectives are different, as well as how the tool processes the UPF, instruments the design with power architectures, corrupts the required design elements on power events, generates automated assertions on power sequence anomalies, and such other diversified factors.

The tool generated verification messages and results, like dumped wave-shapes, incorporate all the above mentioned factors to denote the design under verification is functionally and structurally correct from a power aware perspective. Though structural verifications are discussed in Chap. 7, the accumulation of different UPF strategies in design, either by synthesis or by tool inferring, causes structural design changes and impacts dynamic simulation as well. Hence it is required to generate PA-SIM oriented results at least containing and conveying the following information.

List 5.11 Minimum Set of Information to Represent in PA-SIM Results
- PA design from UPF mapping,
- PA logic instrumentation and corruption,
- UPF objects,
- Virtual inferring of UPF strategies in RTL,
- Physical representation of MV cells in GL-netlist

Though these information are PA based and not exhaustive, however these are added on top of regular dynamic simulation results. Evidently PA-SIM results reporting for efficient debugging is challenging. Specifically, incorporating the above aspects in a simulated wave-shape and correlating them with other forms of visualization systems (like schematic viewer, source code viewer, hierarchical viewer, state-transition viewer, coverage metric viewer) which are mandatory for debugging verification results efficiently.

Apart from PA relevant information in simulation results, it is well known that any regular verification environment consumes more than 75% efforts in debugging for any chip design projects. Hence, for efficient debugging it is crucial to represent maximum information consuming minimum user effort. Obviously, the visual representation of results in a PA simulation environment are considered to be the best for debugging with maximum information and minimum user effort.

Often it is also required to correlate the visual representation of results with the text based results appearing from various built-in sequence checkers, tool generated messages, and testbenches discussed in previous sections. The following example shows the snippet of text message from PA-SIM auto asserted for ISO control anomalies.

Example 5.20 Snippet of Text Reporting Results from Automated Sequence Checker

```
# ** Error: (vsim-8918) QPA_ISO_EN_PSO: Time: 167715000 ps,
Isolation control (0) is not enabled when power is switched OFF for
the following:
# Port: /interleaver_tester/dut/mc0/ceb,
# Port: /interleaver_tester/dut/mc0/web.
# File: rtl_top.upf, Line:          90, Power Domain:PD_mem_ctrl
```

Hence the tool generated text messages from automated sequence checkers are intuitive and facilitate finding issues with simulation time stamps and UPF strategy name (QPA_ISO_EN_PSO), pinpointing the reference design (port name) and UPF files (with line number). In order to correlate the auto fired assertion message (e.g. Time: 167715000 ps etc.) similar to Example 5.20, as well all other PA specific semantics and features within the PA-SIM visualization system, it is required to comprehend at least the following aspects within the PA-SIM tool environment.

List 5.12 Additional Aspects for PA-SIM Results Representation

- Color tagging of corruption and PA instrumentation semantics
- Highlighting PA features to distinguish from generic features
- Interpreting UPF objects to physical entity
- Interpreting UPF strategies in inferred (or inserted) cells
- Finding PA Drivers from power source and sink communication models
- Formulating schematic symbols from inferred or synthesized cells
- Establishing physically visual power supply network on HDL design
- Identifying UPF power domain hierarchy
- Identifying HDL correspondence of power domain hierarchies
- Instrumenting and facilitating TB based debugging (Top-Down)
- Instrumenting and facilitating for design based debugging (Bottom-up)
- Interpreting PA state machine
- Interpreting coverage results in targets and goals for each PA design elements
- Device mechanism to visualize the effect of power states and transitions.

The following figure shows a sample that correlates the automated assertion message in Example 5.20 with the waveform shown in the exact 167715000 ps time stamp.

The above verification aspects and visual systems are very primitive and obvious for any PA simulation environment. However, additional visual information like power domain hierarchy, power domain sources and sink pairs, corruptions, and UPF strategy based status on ports, nets, wires and sequential, PA annotated design, UPF and testbench source view, ISO, LS, PSW and relevant power domain schematic, etc. facilitates verification and debugging in practical, efficient ways.

Following are examples of essential features on specific visualization windows.

Example 5.21 Visualization Items on Various Windows:

(1) **PA Information in Hierarchy Window:**

- This window displays all the instances in the hierarchy window with the corresponding power domain name. PA annotation may further reveal corresponding state information for the power domains from any designs scope.

(2) **PA Annotated Source Code Browsing Window:**

- This window displays power aware annotation, instrumentation, and corruption semantics of design source code, often in different color tags and balloon help format. The annotations are usually visible on the followings HDL parameters:

- Input
- Output
- Wires
- Registers

(3) **Power Domains Window:**

- Used to visualize all the objects defined in the UPF file and correlated with design instances, UPF, and design sources from windows stated in (1) and (2) above. The UPF objects that are usually displayed are listed as follows:

- Power Domains
- Supply Nets
- Logic Nets
- Logics Ports
- UPF Strategies (ISO, LS, ELS, RFF, PSW etc.)

(4) **Power Domain Crossing Window:**

- This window usually displays all the source and sink pairs of power domains in terms of communication crossings, based on the UPF specification. Hence this power domain crossing window is equally essential in both dynamic and static verification.

(5) **PA Automated Checks Windows:**

- This window primary lists all the automated sequence checkers that are asserted during the simulation. The list contains specific information on sequence checks, whether it is conducted for ISO, LS, ELS, RFF, PSW etc., corresponding violations, and correlation with simulated wave shapes.

(6) **PA Schematic Window:**

- The schematic window is crucial to display PA relevant only logic based information. The following are more often available on schematic viewer.

- PA annotated and PA interface logics
- Power supply network connectivity
- Connectivity as well the symbolic representation of inferred and synthesized power management MV cells

(7) **PA Wave-shape Window:**

- The wave shape window is one of the crucial visual systems for any dynamic simulation. Hence to make it more efficient for PA simulation, the power aware wave shapes usually contain special color tagging schemes to differentiate the PA status of signals from relevant regular non-PA simulation semantics. The following are more obvious color tagging features often available in the PA annotated wave-shape window.

- Corruption semantics
- Isolation status
- Retention data in registers
- Retention save and restore status
- Power on and off status
- Power on, off, and UPF strategies based sequence anomalies
- Reset, registers initialization after power resume status

(8) **PG-pin Library Information Window:**

- The PG-pin information from Liberty and UPF for Macro models are often required for accurate PA simulation based corruption. The library window provides visual information on Macro cells with their relative supply information for input and output logic pins, power down function, and UPF supplies connected to them.

(9) **PA Coverage Information Window:**

- Although the PA coverage is discussed in Chap. 6, however the coverage collected from PA dynamic verification display window based relevant information is discussed here. The coverage information is often displayed in the following format in order to provide options for further increasing the coverage metric in successive simulation executions.

- Coverage inclusive and exclusive instances in hierarchical order
- Coverage reports in different file formats (text, HTML, xml) with goals and achievements
- Power state machine diagram view explaining PA states and transitions based on UPF `add_power_state`, `add_port_state`, and `add_pst_state` semantics.

In addition, the tool requires additional commands to represent results for efficient debugging.

Tool Procedure for Efficient PA Debug
Compile: No Change
Optimize: vopt- requires to add "vopt – debug, pa_coverage=checks, pa_enable=highlight", etc.
Simulate: vsim- requires to add "vsim –pa –assertcover, etc.

Even though, from the above discussion, it is clear that the dynamic simulation results are obvious resources for debugging as well as first-rate resources for finding

design, power-specification, and verification methodology relevant bugs and anomalies. However, there are threshold limits in exercising every possible corner of any design with any arbitrary size of verification suits. This is very generic for any non-PA or PA-SIM environment.

Dynamic simulation results are inconclusive in nature and often requires to be quantified with definitive metrics that possibly denote a verification coverage closure in percentages with appropriate design parameters. Power aware coverage data ensures that the regression suites are adequately testing power aware elements of the design, intent, and even the test suite itself. The succeeding sections of Chap. 6, discuss such coherent verification closure issues from both generic and tool specific solution perspectives.

Epilogue: Chapter 5- **UPF Based Power Aware Dynamic Simulation** offered readers the concepts and practice of PA dynamic simulation from the ground up. The special features of PA-SIM like corruption semantics are implied at the RTL through driver-based corruption using the UPF specification (UPF `-simstate`) and at the GL-netlist through Liberty (`power_down_function`) based corruption. Consecutively the chapter also explains the initialization requirements of initial blocks in the design at power resume through **`set_design_attributes -attribute`** *`qpa_replay_init`* **`TRUE`** UPF construct. However, the UPF **`set_design_attributes -attribute UPF_dont_touch TRUE -models`** <> may be used to exclude any specified module, architecture, entity instance, and signals from power aware behavior from re-triggering on power resume or On. This PA behavior exclusion disables power aware instrumentation for corruption, retention, isolation, level_shifter, or buffers on the specified item.

This chapter showed that the PA dynamic (as well PA-Static) verification is required in every design phase in DVIF. The input requirements, in general, for PA-SIM at the RTL are the constraint and configuration UPF, PA augmented testbench, and design under verification in any format of HDL. The testbench augmentation is done through the **`supply_on`** `( string pad_name, real value = 1.0, string file_info = "");` and **`supply_off`** `(string pad_name, string file_info = "" );` functions and importing the "`import UPF::*`" package and "use ieee. UPF.all;" library in the testbench. The generic requirements to verify in a PA dynamic simulation are clearly listed in List 5.1. The readers can take this list along with List 5.2 **Sample of Power Sequence Assertions** to peruse dynamic verification based on 4 **PA Dynamic Simulation: Verification Practices**. Readers may also find the procedure of developing custom PA assertions through UPF **`bind_checker`** syntax, which is very easy to develop and maintain as it remain totally isolated from the main design.

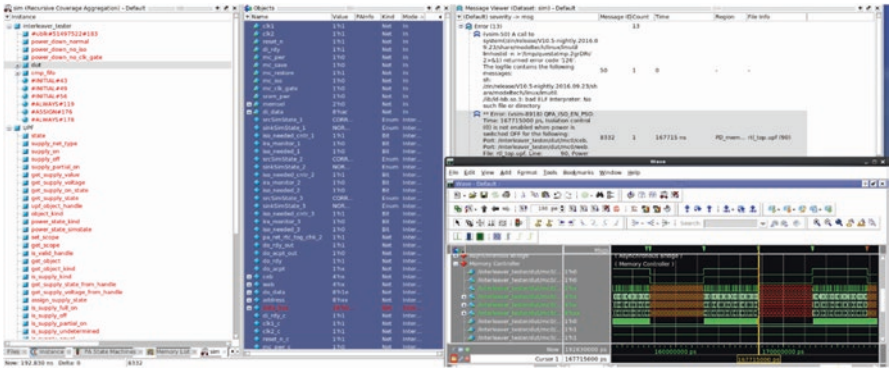

Fig. 5.2 Correlation of results on visual platform

This Chap. 5 also explained the requirements and order of execution precedence of various libraries (e.g. Liberty and PA-Simulation Model libraries) in PA-SIM in conjunction with Chap. 4. It is important that readers grasp the concept of precedence when both PA-SIM Model lib (.v) and Liberty (.lib) are available. Finally for efficient PA dynamic verification and debugging the List 5.11 **Minimum Set of Information to Represent in PA-SIM Results**, List 5.12 **Additional Aspects for PA-SIM Results Representation**, and Example 5.21 **Visualization Items on Various Windows** along with example Fig. 5.2 **Correlation of Results on Visual Platform** will be reference for readers to resolve real-life PA-SIM issues.

Prologue: The next Chap. 6, **Power Aware Dynamic Simulation Coverage** is equally important similar to previous Chap. 5 because dynamic simulation results are inconclusive by nature and often requires to be quantified with definitive metrics that possibly denote verification coverage closure through numeric values (in percentages) in conjunction with appropriate design parameters. However, unlike non-PA-SIM, the PA-SIM coverage metric are difficult to measure because characteristics of power states machines are different. The Unified Coverage Interoperability (UCIS) API as well UPF does not provide complete format or standard semantics for power aware coverage computation models. This chapter offered readers a systematic approach to find all possible source of power states and transitions to explain and develop a coherent PA coverage computation model with practical examples.

Chapter 6
Power Aware Dynamic Simulation Coverage

Coverage provides meaningful insight into design verification completeness. The coverage metric in dynamic simulation is a system or standard of measurement used to describe the degree to which the design is exercised with certain design parameters for a particular test-suite or test-plan execution. Even the test-plan is subject to measurement using a weighted metric and recapitulated to contribute to the total resultant coverage metric. The design and verification parameters that contribute to the combined coverage metric in non-PA simulation is summarized as follows:

List 6.1 Design and Verification Parameters Contributing to Non-PA Coverage Metrics

- Code coverage (this includes branch, condition, expression, statement, and toggle coverage information)
- Finite State Machine (FSM) coverage
- SystemVerilog covergroup coverage
- SystemVerilog and PSL assertion coverage
- Assertion data (including immediate and concurrent assertions, pass, non-vacuous pass, fail, attempt, and other counts)
- Formal analysis results based on different properties
- Verification Plan data
- Links between the Verification Plan and coverage data
- User-defined data
- Test data

Hence, it is comprehensive in the sense that the coverage is a combination of collection, analysis, and reporting (in general, a coverage computation model) of diversified design and verification parameters. Often the resultant metrics from such diversified parameters are stored in a common and unified coverage database (UCDB) for analytical and result representation purposes. UCDB provides accessibility to further enhance coverage metrics with new coverage results from different new sources through coverage merging, as well as a mechanism to analyze and generate the coverage reports through an API. Unfortunately Unified Coverage Interoperability

P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8_6

(UCIS) Standard Version 1.0 does not provide any format or extensions for power aware coverage computation models. Apparently the standard is not enhanced to include the covergroup modeling of PA metrics (discussed in Sect. 3.1). However, UCDB provides the mechanism to merge PA and non-PA coverage metrics to contribute towards 100 percent design verification closure.

6.1 PA Dynamic Simulation: Coverage Fundamentals

Now obviously PA-SIM verification environment coverage metrics are power-related and recapitulate on the non-PA coverage results in the UCDB, as discussed above. The power-related coverage metric originates broadly from the following PA verification perspective.

List 6.2 Primary Source of PA Coverage Metrics

1. Coverage information from PA Dynamic Checks based on:
   - PA testbench (Code Coverage),
   - Automated PA Sequence Checkers,
   - Custom PA Checkers.
2. Coverage information from power-states and power-state-transitions
   - States and their transitions for power domains, supply sets, ports, power state tables, ISO, RFF, PSW Control and Acknowledgement signals.

However similar to UCIS, UPF also does not provide any semantics or standard guidelines for PA coverage computation models. Moreover, in the power aware dynamic simulation state space, the UPF power states have the following unique but very contradictory features with respect to traditional coverage computation models; for example, the finite state machine state transition mechanism.

List 6.3 Contradiction of PA vs. Non PA State Transition Mechanism

- Power state and transitions are asynchronous in nature,
- More than one power state can be true at a time,
- A state may be marked as illegal any time by user,
- Power states of any UPF object may refer to the power states of other UPF objects.

While the coverage collection, analysis, and reporting for PA dynamic checks originating from a PA testbench and different dynamic assertions are very straight forward; however, due to the above listed contradictions, the coverage information extraction from power-states and their transitions requires an exhaustive process. Because the characteristics of a power-state can be best summarized in the following categories.

List 6.4 Characteristics of Power States

- Power states are abstract at higher-levels and physical (supply port and nets) at lower-levels of design abstraction,

- They are applicable for different UPF objects that include rudimentary parts of supply networks and design elements; e.g. power supplies, power domains, design groups, design models, and design instances,
- Power states may reference descendant power-domains or power supply states from the scope of top domains,
- Power states denote different operation modes based on different combinations of power-domains and their power-supplies,
- They are subject to interdependency between different UPF objects; e.g. power domain and supply set
- It is possible that power states may be exposed to simultaneous state transitions between interdependent objects.

Hence, from the above lists at first it is required to apprehend all possible sources of state and transitions for PA coverage metric modeling. Regardless of UPF versions and releases, it is distinctive that the power states and their transition state machines in a PA-SIM environment can originate from one or a combination of the following UPF constructs and UPF commands, their relevant options and objects.

List 6.5 The Source of Power States and their Transitions from UPF Constructs and Objects

- Supply Port States from **add_port_state,**
- Supply Net States from Power state table (PST),
- PST States from **add_pst_state**,
- Power Domain States from **add_power_state**,
- Supply Port, Supply Net, and Supply Set Functions States from **add_supply_state,**
- Power States of the Power Supply Sets from **add_power_state,** etc.

It is also evident that the power states originating from these different types of UPF constructs and objects directly affect the requirements and placement of special power management MV cells, such as ISO, ELS, PSW, and RFF in the design. Hence it is also required to collect coverage information from these MV cells. However, for dynamic simulation the coverage information for MV cells, apart from PSW, may only be collected from the different states and transitions of their controls and acknowledgement signals. Specifically the signals are:

List 6.6 The Source of Power States and Their Transitions from UPF Strategies

- Isolation “Enable” signal,
- Retention “Save and Restore” signals,
- Power Switch States and Transitions,
- Power Switch “Control Port”,
- Power Switch “Ack Port”

Recalling the syntax and example of ISO, RFF, and PSW in Chap. 3, it is evident that all of the above mentioned control and acknowledgement signals have the following transitions:

List 6.7 Transitions of Control Signals for UPF Strategies
- High-to-Low and
- Low-to-High transitions.

As well, during simulation, the status of these signals may remain in one of the following states:

List 6.8 States of Control Signals for UPF Strategies
- Active through presenting a value (level sensitive) or transition (edge sensitive),
- Inactive (opposite to the active)
- Active x (driving unknown)
- Active z (remain floating or un-driven)

In addition, apart from the control and acknowledgement signals, the functionality of the switch itself allows following state values and their possible combination of transitions as defined in the IEEE 1801 specification:

List 6.9 State Values of Power Switch, Control, and Acknowledge Ports
- ON state,
- OFF state,
- Partial ON state and
- UNDETERMINED (ERROR) state.

At any given time, the ON or the partial ON state contributes a value to the output port of the PSW. The UNDETERMINED status comes into existence only when the ON or partial ON state Boolean expression for a given input supply port, which is not in OFF state, refers to an object with an unknown (X or Z) value, then the contributed value at the output of PSW remains {UNDETERMINED, unspecified}. However, in PA-SIM the UNDETERMINED states are interpreted as ERROR states. Hence the PSW itself also has the states as shown in List 6.9. The coverage metric is required to cover all possible transitions between these states (Fig. 6.1).

Hence coverage information modeling from power-states and their transitions based on fundamental aspects of different UPF constructs from **add_port_state, add_pst_state**, and **add_power_state** are also straightforward. Even for the PSW as well as controls and acknowledgment signals of different MV cells, states-transition coverage can be collected through properly exercising the appropriate properties of the design functionalities and elements.

However, all the transitions from the above discussions are not naturally spontaneous in PA-SIM and require user intervention to specify that they are required to be included in the coverage information database. The UPF semantics in 1801–2013 LRM or UPF 2.1 defines the **describe_state_transition** command for monitoring the legality of power state transitions from one named power state to another. As well, 1801–2015 LRM or UPF 3.0 provide similar semantics through the **add_state_transition** command to define named state transitions between power states of an object. These commands provide the methodologies for PA simulation tools to accommodate customized coverage from any state or transitions in PA-SIM. For simplicity, the

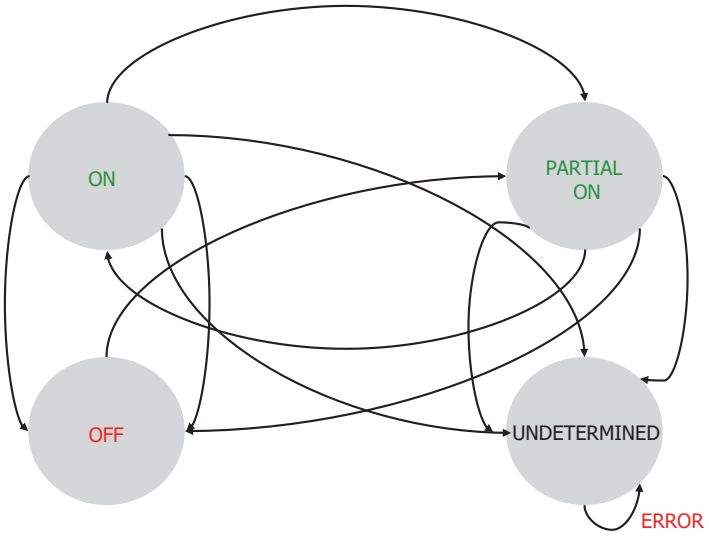

Fig. 6.1 State-transition diagram of PSW

construct detail of **describe_state_transition** is discussed below as representative of a custom state or transition coverage inclusion mechanism.

Example 6.1 Syntax for Monitoring the Legality of Power State Transition

describe_state_transition *transition_name*
-object *object_name*
[**-from** *from_list* **-to** *to_list*]
[**-paired {{***from_state to_state***}*}**]
[**-legal** | **-illegal**]

Here <transition_name> is just a simple name provided by the user. The ***-object*** <object_name> is also a simple name of a power domain or supply set. Often PA simulation tools augment acceptable arguments in ***-objects*** names to possibly extend the scope of PA coverage. The ***-from*** argument in the <from_list> accepts unordered lists of power state names active before a state transition, and <to_list> accepts the unordered list of power state names active after a state transition. The ***-paired*** {{from_state to_state}*} actually accepts the list of transition state pairs name for the <from-state> name and <to-state> name. Finally, the ***-legal*** and ***-illegal*** pair is used to tag a state transition whether it is legal or illegal. By default all state transitions are legal.

Likewise, in a complex hierarchical UPF flow, the power state of the top power domain may remain dependent to the power states of subdomains. Further, power states also specify states for the associated supply sets of these interdependent power domains. Such power state dependency impacts coverage collection and are not as straight forward as the simple state transition coverage discussed so far. A power state dependency based special coverage mechanism is explained in separate Sects. 2 and 3 respectively.

6.2 PA Dynamic Simulation: Coverage Features

The exhaustive lists of coverage extraction procedures from power states and their transitions shown in the beginning part of the previous Sect. 1 actually forecasts the complexity of the overall power-state based coverage computation process. Specifically the abstract nature of power-states, their interdependency, and simultaneous transition occurrence of these interdependent power states requires special attention for coverage collection, analysis, and results.

In order to fully comprehend and confront such coverage complexities, it is required to assess the power state constructs from the component level. Recalling the **add_power_state** examples in Chap. 3, fundamentally, UPF defines a power-state for a power domain and its associated supply networks. The definition further allows referencing the port state of any supply port or supply net in the descendant subtree from the scope of the top power domain. The UPF **add_power_state** syntax is shown below:

Example 6.2 UPF Syntax for add_power_state

add_power_state
[*-supply* | *-domain* | *-group* | *-model* | *-instance*] object_name
[*-update*]
[*-state* {state_name
[*-logic_expr* {boolean_expression}]**
[*-supply_expr* {boolean_expression}]**
[*-power_expr* {power_expression}]**
[*-simstate* simstate]
[*-legal* | *-illegal*]}]*
[*-complete*]

As already discussed, the **add_power_state** command provides the capability to augment power states to different UPF objects through the <object_name>. The <state_name> represents the user specified name for a power state that is just defined or updated for the <object_name>. The ***–logic_expr*** is used to define the power states for either supply sets or power domains. The expressions are constructed by referencing control conditions, clock frequencies, and power states of the domain supply sets. On the other hand ***-supply_expr*** is used only for power states of supply sets; it refers to the functions of supply states of the supply set. Each of these expressions can be further defined as either legal or illegal.

Hence it is distinctive that the logic and supply expressions in the **add_power_state** definition are based on various conditions through Boolean expressions. These Boolean expressions may contain control conditions, design parameters as well as power state information from different power domains including hierarchically lower level domains. Though such subtree power state referencing from top, allows symmetrical hierarchical representations of power domains, supply network, and their corresponding power-states, but they may also impose inter-state dependency that is often difficult to track. The following example shows the power states of PD_top domain in terms of logic expression.

Example 6.3 UPF add_power_state Sample Example with -*logic_expr*

```
add_power_state PD_top -state SYS_ON { -logic_expr \
        {PD_sub1 != SUBSYS1_OFF  && PD_sub2 == SUBSYS2_ON }}
add_power_state PD_top -state SYS_OFF {-logic_expr \
        {PD_sub1 == SUBSYS1_OFF  && PD_sub2 == SUBSYS2_OFF }}
```

These examples are proof of dependency of power states among power domains from PD_top to subtree domains, PD_sub1 and PD_sub2 (reference to Fig. 2.1, Chap. 2). As discussed on several previous occasions, the power states actually remain at the core and drive the entire UPF based PA verification. Hence it is required to devise a mechanism that provides a comprehensive and coherent coverage computation model for the following:

List 6.10 Additional Sources of Power States and Transitions

1. All possible combinations of interdependent power states,
2. As well as their possible simultaneous transitions.

Since these power states and transitions are highly interdependent, the coverage metrics generally aim to find their possible cross combination for transition occurrence and hence the coverage for such models are termed, *cross-coverage*. Although it is already mentioned in the previous section that the UPF semantics in 1801–2013 LRM or UPF 2.1 define **describe_state_transition** for monitoring the legality of power state transitions from one named power state to another, but their semantics do not comply with the context and requirements of cross-coverage computation. (To note, the background detail for the coverage computation model based on the **describe_state_transition** is already discussed in the previous Sect. 1. The command syntax and PA-SIM specific implementation to compute coverage on these semantics are discussed in the succeeding sections).

Hence the PA-SIM verification environment requires to develop an internal mechanism to establish the *cross-coverage computation model*. For simplicity, the cross-coverage computing flow can be best represented by a simple dependency graph. The nodes of the graph will be based on power state logics and supply expressions, where the nodes will represent power states and their edges will represent transitions between the nodes. A path between the nodes will denote a sequence of nodes and edges, connecting a node with its descendant and (or) dependent.

The PA-SIM verification environment required to monitor and capture the transitions from the edges of the graph. The path will provide the depth for a group of inter-dependent nodes and helps to untie the dependency among these nodes. Since there is no UPF semantic, hence tool process, the above coverage metric is based on the following extended methodology.

Example 6.4 Methodology Extension for Cross-Coverage

```
describe_state_cross_coverage
  [-domains domains_list]
  [-depth cross_coverage_depth]
```

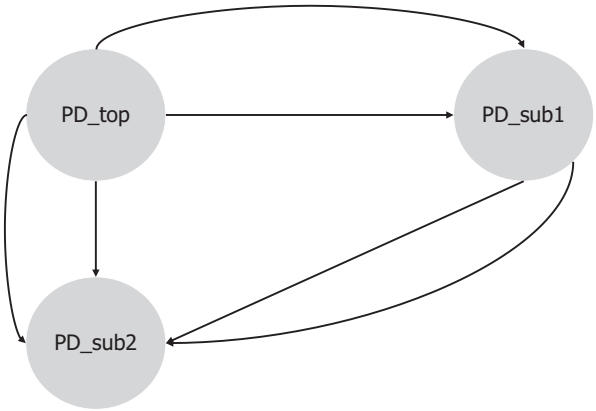

Fig. 6.2 Dependency graph for cross-coverage computation model for power domains group PD_top->PD_sub1->PD_sub2

Table 6.1 Dependent states for power domains

Power domains	PD_top	PD_sub1	PD_sub2
Power-states	SYS_ON	SUBSYS1_ON	SUBSYS2_ON
Power-states	SYS_ON	SUBSYS1_RET	SUBSYS2_ON
Power-states	SYS_OFF	SUBSYS1_OFF	SUBSYS2_OFF

Table 6.2 Cross-coverage data for *-depth*=1 (default) for PD_top->PD_sub1->PD_sub2

SYS_ON -> SUBSYS1_ON -> SUBSYS2_ON
SYS_ON -> SUBSYS1_RET -> SUBSYS2_ON
SYS_OFF -> SUBSYS1_OFF -> SUBSYS2_OFF

The **describe_state_cross_coverage** command is an augmentation of current UPF intent to supplement the missing *cross-coverage computation semantic model*. The ***–domain*** defines the list of power domains for which cross-coverage needs to be computed. The ***-depth*** is the number of power-domains that are involved in the cross-coverage computation and are interdependent. By default, the computation starts with depth one; however the tool will figure out the list of dependent power domains of a particular top or adjacent domains. These dependent power-domains together with the specified top domain will form a domain group. Then the tool computes the cross-coverage results for this particular group of power domains. Hence for the ***–logic_expr*** example of **add_power_state**, shown above and from the Fig. 6.2, PA-SIM will develop the cross-coverage computing model for PD_top->PD_sub1->PD_sub2 as follows (Tables 6.1 and 6.2).

The PA-SIM tool specific details on coverage collection, analysis, and obtaining the results from PA dynamic checks based on the PA testbench, automated PA sequence checkers, custom PA checkers, and power state transitions, as well as coverage from the cross-coverage models are discussed in Sect. 3.

6.3 PA Dynamic Simulation: Coverage Practices

From the discussion in the previous two Sects. 1 and 2, it is apparent that designing a complete PA coverage computation model is complicated and has to rely intensely on various nonstandard and nontraditional aspects of coverage measurements. However, through a comprehensive and coherent analytical model, tool, and methodology augmentation, it is possible to completely enumerate the coverage collection, analysis, and results representation through PA-SIM. The entire PA coverage metric computation model, which was listed in List 6.2, can be updated to a new list based on the analytical discussions in the previous two sections.

List 6.11 Updated List for the Complete Sources of PA Coverage Computation Model

1. Coverage information from **PA Dynamic Checks** based on;
   - PA testbench (Code Coverage),
   - Automated PA Sequence Checkers and
   - Custom PA Checkers,
2. Coverage information from **Power states** and **power state transitions** based on;
   - Design controls,
   - Supply ports and nets created in the UPF and design,
   - Power domains and their power states,
   - Supply sets and their states,
   - Power Switch States and their Transitions,
   - State transitions for ISO, RFF, PSW Control and Acknowledgement signals,
3. Coverage Information from **Cross-Coverage** based on;
   - All possible combinations of interdependent power states,
   - As well as their possible simultaneous transitions.

On the basis of the above categorization, the dynamic simulation tool processes the coverage collection, analysis, and representation of the results from power aware verifications. The succeeding stanzas will cover the tool specific approach for all the coverage information listed in the above three categories. The tool computes the coverage in the following phases:

List 6.12 PA Coverage Computation Phases through Dynamic Simulation

Step 1: Collecting PA coverage
Step 2: Analyze the coverage through generated UCDB for merging
Step 3: Reporting the results
Step 4: Analyze post-simulation coverage data from UCDB
Step 5: (Optional) Coverage Exclusion

6.3.1 Coverage Computation Model: For PA Dynamic Checks

The five steps for the coverage computation model from PA dynamic checks are discussed in detail below.

Step 1: Collecting PA Coverage Information from PA Dynamic Checks
Recalling the standard three-step PA simulation flow discussed in Sect. 5.3, the coverage collections processes are exercised in the following way:

Standard Coverage Processing:

(i) Compile: vlog- No special commands or options required (Not applicable onward)
(ii) Optimize: vopt- requires to add "vopt -pa-coverage=checks"
(iii) Simulate: vsim- No special commands or options required

Advanced Coverage Processing:

This process actually marks a power aware dynamic check as "covered" when the check has never failed and has passed at least once (fail count = 0 and pass count > 0). This result can be obtained from the following tool configurations.

(i) Compile: vlog – Not applicable
(ii) Optimize: vopt- requires to add "vopt -pa_coverage=checks"
(iii) Simulate: vsim- requires to add "vsim –assertcover"

Identifying Unattempt Checks:

In addition, PA-SIM also allows distinguishing any unattempt power aware check assertions. Specifically, the coverage data includes information about any unattempt power aware checks.

(i) Compile: vlog –Not applicable
(ii) Optimize: vopt- requires to add "vopt -pa_coverage=checks"
(iii) Simulate: vsim- requires to add "vsim – unattemptedimmed"

Step 2: Coverage Analysis and Generate UCDB for PA Dynamic Checks:
The next step after collection is coverage analysis. However this is a process to integrate and accumulate coverage data to the UCDB with .ucdb file extension as follows for the tool to analyze or facilitate the user to merge with any or all other coverage information collected.

CLI> coverage save <name>.ucdb
or
CLI> coverage save –pa <name>.ucdb

In the later step, the UCDB accumulated data is used for analyzing and reporting purpose.

Step 3: Reporting Results for PA Dynamic Checks:
PA coverage reports are generated from the data stored in the UCDB. Even though the results can be represented in numerous ways similar to non-PA coverage reports including HTML, XML or plain text, inclusion of verbosity often provides additional guide-points to further improve the achieved metric.

GUI> coverage report -assert –pa (optionally: verbose or summery)
or
CLI> vcover report pa.ucdb -assert –pa

Step 4: Analyze Post-Simulation Coverage Data from UCDB for PA Dynamic Checks:
UCDB provides comprehensive resources for coverage data analysis and generating numerous forms of reports even after completion of the simulation process. The complete coverage closure metric from non-PA and PA simulation environments can be accommodated through loading the database in a simulation window and generate the reports through the following process. Load UCDB in GUI:

GUI> vsim -viewcov <name>.ucdb
Generate reports:
GUI> coverage report -pa [-details]

Step 5: (Optional) Coverage Exclusion from PA Dynamic Checks:
There are no specific fine-grain controls and options to exclude specific coverage features from PA dynamic checks. Alternatively, not using the procedure in the tool Optimization phase explained above will automatically exclude coverage information from PA dynamic checks.

6.3.2 Coverage Computation Model: For Power States and Power State Transitions

Similarly the five steps for a coverage computation model from Power State Transitions in PA dynamic simulation are discussed in detail below.

Step 1: Collecting Power Aware Coverage Information on Power States and Power State Transitions
The basic power state and transitions coverage are primarily collected from the UPF commands explained in Sect. 1 that can originate from one or a combination of **add_port_state**, **add_pst_state**, **add_power_state** and their relevant options and objects.

Standard Power State Transition Coverage Processing:

(i) Compile: vlog – Not applicable
(ii) Optimize: vopt- requires to add "vopt -pa_coverage"
(iii) Simulate: vsim- requires to add "vsim –coverage"

Controlling State Transition Coverage Collection:

Fine-grain user controllability is also available for state-transition coverage collection from ISO, RFF, PSW Control, and Acknowledgement signals through the following tool procedures.

(Option at Optimization: vopt – may requires to add one or all of the followings)

- For Power Switches (PSW states, control, and ack port) "vopt–pa_coverage=switch"
- For Isolation Enable "vopt –pa_coverage=iso"
- For Retention Save and Restore "vopt –pa_coverage=ret"

Controlling Transition Coverage Collection:

The UPF **describe_state_transition** command and its augmentation also provide fine-grain controllability for collecting *power-state-transition-only* coverage.

Recalling the semantics explained in Sect. 1, Example 6.1, the PA-SIM provides the mechanism to include the following coverage information through the **–object <**object_name> options in **describe_state_transition** command.

List 6.13 Sources for PA Transitions for PA Coverage Computation Model

1. Transition Coverage:
   From Power States for
   - Power Domain,
   - Supply Sets,
   - PST states

However, any state marked as illegal will not to be covered. As well, it is also required to generate reports in a separate coverage section, to be more specific to transitions defined and marked legal from **describe_state_transition** command.

The other transition coverage that are also possible to collect from the augmentation of the **describe_state_transition** semantics are from the following categories.

List 6.14 Additional Sources for PA Transitions for PA Coverage Computation Model

1. Transition Coverage:
   From Power states for
   - PST,
   - Supply Ports,
   - Power Switches

In this case as well, when transitions are marked illegal they will not be reported in the transition coverage reports. For this category, it is also required to add the following procedure in the Optimization phase for tool processing along with all other relevant power aware dynamic simulation commands.

Compile: No change
Optimize: vopt- requires to add "vopt -pa_enable=statetransition" along with "-pa_coverage"
Simulate: vsim- requires to add "vsim –coverage"
UCDB Generation and Coverage Reporting: No change

Hence for the controlling transition coverage collection, the PA-SIM compilation and coverage reporting do not require any special procedures.

It is also required to understand that this UPF **describe_state_transition** command is exclusive for describing a state transition legality. Hence it is exploitable for coverage collection controllability. So, when this command is not used, all transitions are legal and reported in coverage results.

Alternately, when control is defined in the UPF intent file through the **describe_state_transition** command, the tool will generate a coverage report only for the specified transition in **–object <**object_name> as shown in following Example 6.5. In order to realize the complete transition coverage depiction, a corresponding PSW example is also shown adjacent to the controlling of the transition coverage commands.

Example 6.5 Controlling Transition Coverage by UPF describe_state_transition Command

```
# PSW example for Collecting State Transition Coverage
create_power_switch IN_sw \
    -domain PD_SUBSYS2 \
    -output_supply_port {vout_p VDD_IN_net} \
    -input_supply_port {vin_p MAIN_PWR_moderate} \
    -control_port {ctrl_p IN_PWR} \
    -on_state {normal_working vin_p {ctrl_p}} \
    -off_state {off_state {!ctrl_p}}

# Controlling State Transition Coverage by UPF for PSW (IN_sw) shown above
describe_state_transition ts -object IN_sw -from {ON} -to {}
describe_state_transition ts1 -object IN_sw -from {ON} -to {OFF}
describe_state_transition ts2 -object IN_sw -from {ON} -to {}
describe_state_transition ts3 -object IN_sw -from {ON} -to {ERROR} -illegal
```

A sample report for Example 6.5 is shown below.

Example 6.6 Sample State Transition Coverage Reports from describe_state_transition for PSW

```
 UPF OBJECT                                  Metric  Goal   Status
-------------------------------------------------------------------

PSW State Coverage Sample Report:
TYPE : Power Switch /alu_tester/dut/IN_sw  50.0%   100    Uncovered
Power Switch coverage instance \/alu_tester/dut/pa_coverageinfo/
IN_sw/IN_sw_PS/PS_IN_sw
                                           50.0%    100   Uncovered
     Power State ON                       100.0%   100   Covered
         bin ACTIVE                          3       1   Covered
     Power State OFF                      100.0%   100   Covered
         bin ACTIVE                          2       1   Covered
     Power State PARTIAL_ON                 0.0%   100   ZERO
         bin ACTIVE                          0       1   ZERO
     Power State ERROR                      0.0%   100   ZERO
         bin ACTIVE                          0       1   ZERO

PSW Transition Coverage Sample Report:
TYPE : Power Switch /alu_tester/dut/IN_sw   33.3%   100   Uncovered
Power Switch coverage instance \/alu_tester/dut/pa_coverageinfo/
IN_sw/IN_sw_PS/PS_TRANS_IN_sw
                                           33.3%    100   Uncovered

Power State Transitions                    33.3%    100   Uncovered
         illegal_bin ON -> ERROR              0           ZERO
         bin ON -> PARTIAL_ON                 0       1   ZERO
```

```
        bin ON -> ON                          0     1   ZERO
        bin ON -> OFF                         2     1   Covered

PSW Control Port State Coverage Sample Report:
TYPE : Power Switch Control Port /alu_tester/dut/IN_sw/ctrl_p
                                          50.0%   100   Uncovered
Power Switch Control Port coverage instance \/alu_tester/dut/pa_
coverageinfo/IN_sw/ctrl_p/PS_ctrl_p
                                         50.0%   100    Uncovered
     Power State ACTIVE_LEVEL           100.0%   100    Covered
         bin ACTIVE                        3      1    Covered
     Power State INACTIVE               100.0%   100    Covered
         bin ACTIVE                         2     1    Covered
     Power State ACTIVE_Z                0.0%    100    ZERO
         bin ACTIVE                        0      1    ZERO
     Power State ACTIVE_X                0.0%    100    ZERO
         bin ACTIVE                        0      1    ZERO
```

Now, the "**Step2:** through **Step 4:**"for the coverage information processing on "**States and Transitions:**" are exactly the same as for "**PA Dynamic Checks:"** shown above. Hence they are not duplicated here once again, except the following **step 5**.

Step 5: (Optional) Coverage Exclusion from Power States and Power State Transitions

The PA-SIM provides fine granularity for controlling specific power states to be excluded from coverage reporting. The "coverage exclude" procedure syntax is as follows:

CLI> coverage exclude –scope <scope_path>
–pstype {portstate | pststate | powerstate}
–pstate <state_obj> <state_list>
–ptrans <state_obj> <transition_list>

Where,

portstate: state on a port and net added via **add_power_state** or **add_port_state**

pststate: state on a pst added via **add_pst_state**

powerstate: state on a supply set and power domain added via **add_power_state**

The -pstate <state_obj> <state_list>: Exclude only listed states from state_list of object state_obj

The -ptrans <state_obj> <transition_list>: Exclude only transitions mentioned in transition_list of object state_obj.

More detailed explanation of "coverage exclude" usage related to **pstate** and **ptrans**:

State exclusion (-**pstate**):CLI> coverage exclude –scope <scope_path> -**pstate** <state_object> <state list>

Where,<scope_path> is the scope of state object listed in List 6.15.

List 6.15 The List of <state_object>
- Supply port
- PST
- Supply Set
- Power Domain
- Power Switch
- Power Switch Control & Ack
- Isolation Signal (ISO_SIG)
- Retention Save (SAVE_SIG), Retention Restore Signal (RESTORE_SIG)

All transitions to and from this state will also be excluded.

Transition exclusion (-**ptrans**):CLI> coverage exclude –scope <scope_path> -**ptrans** <state_object> <transition list>

Where,<transition> is actually transition between state1 -> state2

In addition, tool also provide support for s1->* and *->S1 (where * denotes a simplified wildcard) type of transition coverage exclusion.

It is also worth noting that, during PA-SIM, all power states originating from **add_port_state**, **add_pst_state**, **add_power_state** may not remain active all the time. In general these commands specify predefined and named power states explicitly for UPF objects; such as power domain, supply set, supply port, and power state tables. The named state and their transitions are automatically covered by the tool. However, in a particular verification environment, the UPF intent file may not specify all possible power states for all of the above objects. Hence the tool specifies the missing power states as "undefined" power states and assigns these states as active when the named states are inactive. Hence the coverage data may also include "undefined" power states and transition information in the coverage data.

6.3.3 Autotestplan Generation: From PA Dynamic Checks and Power State Transitions

The coverage information created from List 6.11, the (1) PA Dynamic Checks and (2) Power States and Power State Transitions, as discussed in previous sections, provides an extra level of PA verification confidence. However, apart from providing the coverage closure metric, their coverage information also facilitates verification onward by auto generating a PA testplan as a constructive derivative in the process. The automated testplan or "Autotestplan" is generated on the basis of coverage information from the following categories of coverage collection procedure.

List 6.16 PA Checks Contributing Autotestplan Generation
- Dynamic PA checks,
- Power states and transitions (mainly from power domains and supply sets),
- Port states and transitions,
- Power State Table states and transitions

Autotestplan Generation Process:

(i) Compile: vlog – Not applicable
(ii) Optimize: vopt- requires to add "vopt -pa_enable=autotestplan"
(iii) Simulate: vsim- requires to add "vsim –coverage –assertcover"
(Optionally -unattemptedimmed)

Coverage Analysis and Generate UCDB for Autotestplan:
coverage save -pa pa.ucdb
(Optionally) Generate XML format for the Autotestplan:
pa autotestplan -format=xml -filename=foo.xml
Merge the coverage UCDB with Autotestplan UCDB:
vcover merge merged.ucdb pa.ucdb QuestaPowerAwareTestplan.ucdb

To note, QuestaPowerAwareTestplan.ucdb will be generated during the simulation process when the tool procedures are used as noted in "**Autotestplan Generation Process**".

Visualize the merged UCDB to analyze the testplan and results on verification planner window:vsim -viewcov merged.ucdb

The PA testplan may be used on verification onward for correlating the achieved coverage results and improving overall test suite specification and plan further. To assist the process, as noted before, the testplan also can be generated in various formats, including spreadsheets. The PA-SIM visualization system enables providing the PA coverage scenarios at a glance based on the automated testplan. Figures 6.3 and 6.4 below show an example of an XML format of a testplan and its visual contents, respectively. The hierarchical construct of the "testplan section", "coverage status", and the "% of goal" achieved features are productive resources to apprehend the verification status.

	A	B	C	D	E	F	G	H
1		Testplan For Power Aware Coverage						
2								
3		#	Section	Description	Link	Type	Weight	Goal
4		1	alu_tester	Design hierarchical scope			1	100
5		1.1	dut	Design hierarchical scope			1	100
6		1.1.1	OUT_RET_sw	Power Switch			1	100
7		1.1.1.1	Power State Coverage	To cover Power States	OUT_RET_sw_STATE_COVERAGE.\alu_teste	COVERGROUP	1	100
8		1.1.1.2	Power State Transition	To cover Power State Transitions	OUT_RET_sw_TRANSITION_COVERAGE.\al	COVERGROUP	1	100
9		1.1.1.3	ctrl_p	Control Port			1	100
10		1.1.1.3.1	Power State Coverage	To cover Power States	ctrl_p_STATE_COVERAGE.\alu_tester/dut/pa	COVERGROUP	1	100
11		1.1.1.3.2	Power State Transition	To cover Power State Transitions	ctrl_p_TRANSITION_COVERAGE.\alu_tester/d	COVERGROUP	1	100
12		1.1.2	OUT_sw	Power Switch			1	100
13		1.1.2.1	Power State Coverage	To cover Power States	OUT_sw_STATE_COVERAGE.\alu_tester/dut/	COVERGROUP	1	100
14		1.1.2.2	Power State Transition	To cover Power State Transitions	OUT_sw_TRANSITION_COVERAGE.\alu_test	COVERGROUP	1	100
15		1.1.2.3	ctrl_p	Control Port			1	100
16		1.1.2.3.1	Power State Coverage	To cover Power States	ctrl_p_STATE_COVERAGE.\alu_tester/dut/pa	COVERGROUP	1	100
17		1.1.2.3.2	Power State Transition	To cover Power State Transitions	ctrl_p_TRANSITION_COVERAGE.\alu_tester/d	COVERGROUP	1	100
18		1.1.3	ALU_sw	Power Switch			1	100
19		1.1.3.1	Power State Coverage	To cover Power States	ALU_sw_STATE_COVERAGE.\alu_tester/dut/	COVERGROUP	1	100
20		1.1.3.2	Power State Transition	To cover Power State Transitions	ALU_sw_TRANSITION_COVERAGE.\alu_test	COVERGROUP	1	100
21		1.1.3.3	ctrl_high	Control Port			1	100
22		1.1.3.3.1	Power State Coverage	To cover Power States	ctrl_high_STATE_COVERAGE.\alu_tester/dut/	COVERGROUP	1	100
23		1.1.3.3.2	Power State Transition	To cover Power State Transitions	ctrl_high_TRANSITION_COVERAGE.\alu_test	COVERGROUP	1	100
24		1.1.3.4	ctrl_moderate	Control Port			1	100
25		1.1.3.4.1	Power State Coverage	To cover Power States	ctrl_moderate_STATE_COVERAGE.\alu_tester	COVERGROUP	1	100
26		1.1.3.4.2	Power State Transition	To cover Power State Transitions	ctrl_moderate_TRANSITION_COVERAGE.\alu	COVERGROUP	1	100
27		1.1.3.5	ctrl_low	Control Port			1	100
28		1.1.3.5.1	Power State Coverage	To cover Power States	ctrl_low_STATE_COVERAGE.\alu_tester/dut/p	COVERGROUP	1	100
29		1.1.3.5.2	Power State Transition	To cover Power State Transitions	ctrl_low_TRANSITION_COVERAGE.\alu_teste	COVERGROUP	1	100
30		1.1.4	IN_ISO_sw	Power Switch			1	100
31		1.1.4.1	Power State Coverage	To cover Power States	IN_ISO_sw_STATE_COVERAGE.\alu_tester/d	COVERGROUP	1	100
32		1.1.4.2	Power State Transition	To cover Power State Transitions	IN_ISO_sw_TRANSITION_COVERAGE.\alu_t	COVERGROUP	1	100
33		1.1.4.3	ctrl_p	Control Port			1	100

Fig. 6.3 Spreadsheet format of automated testplan

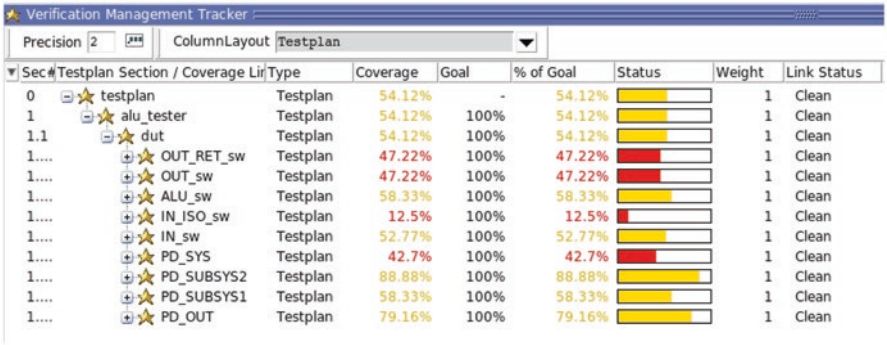

Fig. 6.4 Visualization of automated testplan features

6.3.4 Coverage Computation Model: For Cross-Coverage

The cross-coverage computation model is fundamentally based on the power state's Boolean expressions defined either in terms of logic nets, power states of supply sets, power domains or their combination, that evaluates to be true when the object is in the state being defined. The ***-logic_expr*** in **add_power_state** semantic specifies such Boolean expressions and are the keys to the cross-coverage extraction process.

Recalling the Example 6.4 in Sect. 2, the extended methodology for collecting cross-coverage through the **describe_state_cross_coverage** semantic is explained with a specific example of logic expression in **add_power_state** command below.

Step 1: Collecting Coverage Information from Cross-Coverage
Example 6.6 Sample example for cross-coverage collection from **add_power_state**

```
add_power_state PD_OUT -state PD_OUT_on {-logic_expr {PD_OUT.primary
== PD_OUT_primary_on}}
add_power_state PD_OUT -state PD_OUT_off {-logic_expr {PD_OUT.primary
== PD_OUT_primary_off}}
add_power_state PD_OUT -state PD_OUT_ret \
        {-logic_expr {PD_OUT.primary == PD_OUT_primary_off && PD_
OUT.default_retention == PD_OUT_ret_on}}
add_power_state PD_OUT2 -state PD_OUT_on {-logic_expr {PD_OUT ==
PD_OUT_on}}
add_power_state PD_SUBSYS2 -state PD_SUBSYS2_on \
        {-logic_expr {PD_SUBSYS2.primary == PD_SUBSYS2_primary_on}}
add_power_state PD_SUBSYS2 -state PD_SUBSYS2_off \
        {-logic_expr {PD_SUBSYS2.primary == PD_SUBSYS2_primary_off}}
add_power_state PD_SUBSYS1 -state PD_SUBSYS1_on \
        {-logic_expr {PD_SUBSYS1.primary == PD_SUBSYS1_high_volt &&
PD_OUT2 == PD_OUT_on}}
```

add_power_state PD_SYS ***-state*** RUN \
{***-logic_expr*** {PD_SUBSYS1 == PD_SUBSYS1_on && PD_SUBSYS2 == PD_SUBSYS2_on}}
add_power_state PD_SYS ***-state*** SLEEP \
{***-logic_expr*** {PD_SUBSYS1 != PD_SUBSYS1_on && PD_SUBSYS2 != PD_SUBSYS2_on}}
configure cross coverage
describe_state_cross_coverage ***-domains*** {PD_SYS} ***-depth*** 3
describe_state_cross_coverage ***-domains*** {PD_SUBSYS1} ***-depth*** 2
describe_state_cross_coverage ***-domains*** {PD_OUT2}

In the above examples, the **describe_state_cross_coverage** is given for three different power domains to find the cross-coverage data for different depths ranging from none, which is default and depth 1 for PD_OUT2, depth 2 for PD_SUBSYS1, and depth 3 for PD_SYS. The tool will process the dependency graph for each category based on the depth information and extract the state transition from the **add_power_state** logic expression components like "{PD_SUBSYS1 != PD_SUBSYS1_on && PD_SUBSYS2 != PD_SUBSYS2_on}".

Though the testbench is the actual trigger for a power state to make a transition to another state, the cross-coverage collection procedure depends more on unrolling the dependency of these transitions on power states from hierarchical paths. Following are the cross-coverage collection procedures for PA-SIM.

Compile: No change.
Optimize: vopt – requires to add "vopt –pa_coverage"
Simulate: vsim – requires to add "vsim -coverage –assertcover"

Step 2: Coverage Analysis and Generate UCDB for Cross-Coverage
The cross-coverage database generation from the collected coverage in step 1 is similar to any other UCDB generation process.

CLI> coverage save pacross.ucdb

Step 3: Reporting Results for Cross-Coverage
As well, cross-coverage result analysis and representation from UCDB are also similar to any other coverage reporting process.

CLI> vcover report -detail -pa pacross.ucdb

The following Example 6.7 shows a sample reports for cross-coverage.

Example 6.7 Sample Report for Cross-Coverage Based on Example 6.6

```
POWER STATE COVERAGE:
---------------------------------------------------------------------
UPF OBJECT                                  Metric     Goal   Status
---------------------------------------------------------------------
TYPE : POWER STATE CROSS
/alu_tester/dut/PD_SYS(ID:PD1),
/alu_tester/dut/PD_SUBSYS2(ID:PD2),
/alu_tester/dut/PD_SUBSYS1(ID:PD3),
```

```
/alu_tester/dut/PD_OUT2(ID:PD4),
/alu_tester/dut/PD_OUT(ID:PD5)              100.0%   100   Covered
POWER STATE CROSS coverage instance \/alu_tester/dut/pa_coverage-
info/PD_SYS/PD_SYS_PS_CROSS/PS_CROSS_PD_SYS
                                          100.0%   100   Covered
Power State Cross                         100.0%   100   Covered
bin \PD1:SLEEP-PD2:PD_SUBSYS2_off            2       1   Covered
bin \PD1:RUN-PD2:PD_SUBSYS2_on-PD3:PD_SUBSYS1_on-PD4:PD_OUT_on-PD5:
PD_OUT_on

                                             2       1   Covered
TYPE: POWER STATE CROSS
/alu_tester/dut/PD_SUBSYS1(ID:PD1),
/alu_tester/dut/PD_OUT2(ID:PD2),
/alu_tester/dut/PD_OUT(ID:PD3)              100.0%   100   Covered
POWER STATE CROSS coverage instance \/alu_tester/dut/pa_coverage-
info/PD_SUBSYS1/PD_SUBSYS1_PS_CROSS/PS_CROSS_PD_SUBSYS1
                                          100.0%   100   Covered

    Power State Cross                     100.0%   100   Covered
        bin \PD1:PD_SUBSYS1_on-PD2:PD_OUT_on-PD3:PD_OUT_on
                                             2       1   Covered
```

Step 4: Analyze Post-Simulation Coverage Data from UCDB
This is also exactly the same as in the case for PA Dynamic Checks explained in Sect. 3.1. However for simplicity the commands are reproduced here as well. Load UCDB in GUI:

GUI> vsim -viewcov <name>.ucdb
Generate reports:
GUI> coverage report -pa [-details]

Step 5: (Optional) Coverage Exclusion from Cross-Coverage
Since cross-coverage is more oriented toward capturing the coverage of inter-dependent power states, apparently based on the logic expression components of **add_power_state**, the possible combination of power states may explode and may diverge from the actual point of interest. Hence, the tool provides abundant fine grain controllability to customize report generation through exclusion. Following are the tool procedures for cross-coverage exclusion along with simple examples.
CLI> coverage exclude –scope <scope_path>
–**pstate** <state_obj> <state_list>
–**ptrans** <state_obj> <transition_list>
–**pcross**

Where,

The -pstate <state_obj> <state_list>: Exclude only listed states from state_list of object state_obj

The -ptrans <state_obj> <transition_list>: Exclude only transitions mentioned in transition_list of object state_obj.

The -pcross: Exclude cross coverage of the listed –pstate or –ptrans.

Cross Coverage exclusion (-pcross):

CLI> coverage exclude -pstype powerstate -pstate PD_SYS

This will exclude the cross coverage bins for PD_SYS like PD_SYS_PD_SUBSYS1_PD_SUBSYS2

CLI> coverage exclude -pstype powerstate -pstate PD_SYS RUN

However this will be as follows:

Exclude - {RUN PD_SUBSYS1_on PD_SUBSYS2_on}

Include - {SLEEP PD_SUBSYS1_on PD_SUBSYS2_on}

CLI> coverage exclude -pstype powerstate -pstate PD_SYS SLEEP –pcross

Exclude - all the cross coverage bins of SLEEP (PD_SYS)

(Include - this will not exclude the state and transition coverage).

It is worth mentioning here that the cross-coverage theory explained in Sect. 2 is implemented based on SystemVerilog covergroup semantics. The "cross-coverage bin" is the extractions of coverage point transitions (edge of the graph). More precisely, a covergroup can contain one or more coverage points. A coverage point can be an integral variable or an integral expression. Each coverage point includes a set of bins associated with its sampled values or its value transitions. The bins, as seen in the above examples, can be explicitly defined by the user or automatically created by the tool exploiting SystemVerilog semantics.

Regarding coverage for Mixed-RTL and the GL-netlist, it is also required to know that during PA-SIM, the power aware automated checks are not conducted on the gate level cell instance scopes as discussed in Chap. 5. Hence they are excluded from any coverage calculation and the tool generates the list of all the excluded assertion information in the report file named "pachecks.excluded.txt".

Coverage is an enormous topic and hence explained in a separate Chap. 6, in conjunction with the PA dynamic simulation, since coverage computation models are possible to develop only during the simulation process. Unlike the non-PA, the PA coverage computation models are exceptional, complicated, and robust in nature as explained and observed in this **Chapter**.

Hence the PA dynamic verification through simulation is also a robust, critical, yet mandatory verification process in every phase of DVIF that combines diversified power aware features, functionalities, computation models, techniques, and trends for the tool, design, and methodologies. PA dynamic simulation is the testimony of the functional correctness of the design under verification from the power specification and power intent perspectives. However, such power specifications also alter the design structurally from the power architecture and microarchitectural points of observation. Often verifying power-related physical and structural changes in a design through simulation is expensive in terms of verification time, resources, and efficiency.

The next Chap. 7 will discuss a different perspective of PA verification, which is PA-Static. The PA-Static verification tool with built-in PA architectural and micro-architectural verification rules is the best suited solution for verifying power intent and design from physical and structural points of view. The next Chap. 7 will discuss the PA-Static verification techniques. Although chronologically the PA-SIM verification discussion appears before the PA-Static, there is no specific precedence to deploy such tools in the DVIF in the same order. The order of PA tool deployment for PA verification, broadly, depends on the design abstraction levels. Correlating the results from both the verification environment are instinctive.

Epilogue: The Chap. 6 provided an exhaustive but coherent solution to device and measure PA coverage metric. The readers will find the exhaustive List 6.4 for understanding the PA state machine mechanism and Lists 6.5, 6.6, 6.7, 6.8, 6.13, and 6.14 as the complete sources of power states and their transitions to compute complete PA coverage. Specifically the List 6.11 **Updated List for the Complete Sources of PA Coverage Computation Model** will remain as the primary source for all type of PA coverage computations. Although **add_power_state** remains as standard source but this chapter also explains the missing part for interstate overlap or cross-coverage computation through extended UPF command **describe_state_cross_coverage**. In addition, Autotestplan generation, merging of PA coverage metric in UCDB, and fine-grain controlling of exclusion of coverage data are explained with practical examples.

Prologue: The Chap. 7 – **UPF Based Power Aware Static Verification**, just like Chap. 5, provides a generic foundation for PA-Static verification. The chapter explains the advanced techniques for PA-Static verification with detail example of built-in rules of the static checker. In addition, the chapter also includes an interactive PA-Static examples that explains the fundamental aspects of how to debug and resolve real power aware static issues in a design.

Chapter 7
UPF Based Power Aware Static Verification

PA-Static verification, more popularly known as PA-Static checks, are performed on designs that adopt certain power dissipation reduction techniques through the power intent or UPF. The term *static* originates from verification tools and methodologies that applies a set of pre-defined power aware (PA) or multi-voltage (MV) rules based on the power requirements, statically on the structure of the design. More precisely, the rule sets are applied on the physical structure, architecture, and micro-architecture of the design, in conjunction with the UPF specification but without the requirements of any external stimulus or testbenches.

However, in addition to the UPF and design under PA verification, PG-pin enable standard Liberty libraries are required. Optionally, simulation model libraries are required too, specifically when cells are instantiated at the RTL, synthesized, or placed & routed in the design. Table 4.2 in Chap. 4 summarizes the library requirements for PA-Static verification for different levels of the design verification and implementation flow (DVIF). This Chapter will further elaborate upon the exact library requirements and processing mechanisms for PA-Static verification tools and techniques, along with all other static verification features.

7.1 PA Static Checks: Fundamental Techniques

As mentioned in Chap. 5, PA-Static verification is primarily targeted to uncover the power aware structural issues that affects design physically in architectural and microarchitectural aspects. The structural changes that occurs in a PA design are mostly due to physical insertions of special power management and MV cells; such as PSW, ISO, LS, ELS, RPT, and RFF.

P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8_7

These power management MV cells are essential for power-shutdown. The generic functionalities of these cells may be best summarized as follows.

List 7.1 Generic Functionalities of Power Management and MV Cells
- Prevent inaccurate data propagation between Off and On power domains
- Provides accurate logic resolution between high-to-low or low-to-high voltage power domains
- Allows the control, clock, and reset signal to feed through Off-power domains
- Allows data and state retention during power Off or power reductions
- Provide primary power, ground, bias, related, and backup power connectivity

However, these features and functionalities mediated through MV cells are obtained at different levels of design abstraction. Further, these cells are defined through UPF strategies and Liberty libraries.

The fundamental technique that a PA-Static checking tool enforces to verify a design statically involves ascertaining the compliance of the MV or PA rules with the power intent or UPF specifications and Liberty libraries. Eventually, the tool performs all other possible syntax, semantic, and structural checks. Obviously all are based on internally integrated or pre-designed PA rules.

There are also provisions to add custom rules based on demand, through an external interface of the tool through Tcl procedures. The PA rules are essential to verify or validate a design from RTL to PG-netlist, in conjunction with UPF and Liberty libraries, in the PA verification and implementation environment.

For different levels of design abstraction, the essential PA-Static checks can be best summarized by the following categories:

List 7.2 Essential PA-Static Checks at Different Design Abstraction Level
At RTL:

1. Power intent syntax and UPF consistency checks – for design elements, data, and control signals or ports against the design and UPF.
2. Power Architectural checks – for ISO, LS, ELS, RFF strategies against the power states or power state table definitions in UPF.

   At GL-netlist:
3. Microarchitectural checks – for relative power on or always-on ordering of power domains for control signals, clocks, resets, etc., ensuring those are originated from relatively On- or always-on power domains and RPTs or feed-through buffers are present when passing through Off-power domains. These checks are performed against the design and UPF.

   At both PG-netlist and GL-netlist:
4. Physical structural checks – for implemented PSW, ISO, ELS, LS, RPT, RFF against the UPF, specifications, Liberty libraries and MV or Macro cell inserted or instantiated design.

   At PG-netlist:

5. PG-pin connectivity checks – for power-ground, bias, and backup power pins as well as identifying open supply nets and pins against the Liberty libraries, design, and UPF.

It is evident from the above categorization that PA-Static checks can start as early as the RTL, for consistency and architectural checks based on UPF specifications, and extend to the GL-netlist, for microarchitectural and physical structural checks.

The PG-pin connectivity checks can be performed only at the PG-netlist level. Although there are certain physical structural checks that are possible to accomplish only at the PG-netlist level, specifically PSW, RPT, etc. are usually implemented during the Place & Route (P&R) process, and the strategies physically become available for static checks only at the PG-netlist that is extracted from P&R.

7.2 PA Static Checks: Verification Features

Recalling the PA dynamic simulation and PA-Static verification requirements for different levels of the DVIF as shown in Fig. 5.1 in Chap. 5; it is already known that PA-Static verification is mandatory for all stages of DVIF, along with PA-SIM. However, PA-Static verification provides more significant insight into the design at the GL-netlist and PG-netlist levels. This is because special power management MV and Macro cells are physically available in the design only at these netlist levels and provide detail of their PG connectivity.

As briefly mentioned in Sect. 1, for PA-Static verification the input requirements for the tool are as follows.

List 7.3 Input Requirements for PA-Static Verification

- The PA design under verification
- The UPF with proper definition of UPF strategies, power states, or a power state table
- PG-pin Liberty library for MV, Macro, and all other cells (specifically at or after the GL-netlist),

It is also important to mention here that the PA-Static checker optionally requires the PA Sim-Model library for MV and Macro cells for compiling purposes only, when these cells are instantiated in the design. The requirements of such model libraries are explained through the code snippet in the following Example 7.1.

Example 7.1 PA-Static Tool Requirement of PA Sim-Model Library for Compilation Purpose

// The RTL design contains LS cell instantiated as follows:

module memory (input mem_shift, output mem_state);

…..

Line 68 *LS_HL mem_ls_lh3 (.I(mem_ shift), .Z(mem_state));*
endmodule
// During compilation it is required to include the LS_LH module defined in (.v) file,
`celldefine
module LS_HL (input I, output Z);
buf (Z, I);
specify
(I => Z) = (0, 0);
endspecify
endmodule
`endcelldefine
Otherwise, the simulation analysis process will generate the following Error:

Example 7.2 PA-SIM Analysis Error during Compilation of the Design

```
** Error: memory.v(68): Module 'LS_HL' is not defined.
```

Here an LS LH_HL is instantiated in a design. The corresponding snippet of a model library and LS cell Liberty library is also presented. Although the tool procedures are discussed in the succeeding Sect. 4 **PA Static Checks: Verification Practices,** however, for simplicity the design, UPF, Liberty and libraries compiling mechanism by PA-Static tool is explained here with other relevant tool procedures.

The PA-Static checker tool actually analyzes the input information before conducting verification statically. The MV or PA rules applied to the design under verification are based on the following information, extracted and analyzed within the tool from the UPF, Liberty libraries, and the design itself, specifically when the design is at the GL-netlist after synthesis.

The PA-Static tool extracted and internally analyzed information can be summarized as follows:

List 7.4 Summary of Information Analyzed by PA-Static Tool
- Power domains,
- Power domain boundary,
- Power domain crossings,
- Power states,
- ISO, LS, ELS, RFF, PSW, RTP, etc. UPF strategies,
- Cell-level attributes,
- Pin-level attributes (PG-pin only)

Recalling the **`create_power_domain -elements {}`** syntax and semantics explained in Chap. 3, the tool processes and creates the power domains based on the UPF definition and HDL hierarchical instances from the design. The fundamental concepts of power domains that specify and confine certain portions of a design or elements play a significant role in establishing connectivity for inter-domain and intra-domain communications.

As also explained in Chap. 3 and in Fig. 3.1, the formation of power domains inherently defines its domain boundary and domain interface through the **create_power_domain** UPF command and option combinations. Specifically, power supply, power strategies, logic port and net connectivity, and subdomain hierarchical connections are established through the domain boundary and domain interface. Hence, the power domain boundaries are the foundation of UPF methodologies, based on which all UPF strategies and source-sink communication models are established.

The power domain crossings, which are more PA or MV terminology and relevant to the PA-Static tool, actually identify two or more relevant power domains that are communicating through HDL signal wires, nets, and ports. The power domain boundaries and their crossings actually formulate the source-sink communication model within the tool, not just considering HDL connectivity and hierarchical connectivity (HighConn and LowConn), but also coordinating other substantial factors defined within the UPF, design, and Liberty. These factors are the states of supply set or supply net, the states of the power domains, corresponding supply port and net names, as well as the combination of supply sets or supply nets for power domains that form different operating modes for the source-sink communication models or for the entire design.

The states of supply set, supply net, power domains, and their combinations that form the operating modes are composed from the **add_power_state** and PSTs that are usually constructed with the **add_power_state, add_port_state,** and **add_pst_state** semantics in UPF. The following examples show the pertinent components that facilitate forming the operating modes through power states as well as the power domain crossings that finally reinforce the source-sink communication models.

Example 7.3 UPF Power States from PST for the Power Domain

```
set_scope cpu_top
create_power_domain PD_top
create_power_domain PD_sub1 –elements {/udecode_topp}
….
set_domain_supply_net PD_ top \
    -primary_power_net VDD \
    -primary_ground_net VSS
….
set_domain_supply_net PD_ sub1 \
    -primary_power_net VDD1 \
    -primary_ground_net VSS
create_pst soc_pt          -supplies        { VDD VSS VDD1}
add_pst_state ON  –pst soc_pt –state    { on  on  on}
add_pst_state OFF –pst soc_pt –state    { on  on   off}
```

Example 7.4 UPF Power States from add_power_states for the Power Domain

```
add_power_state PD_top.primary
    -state {TOP_ON –logic_expr {pwr_ctrl ==1}
    {-supply_expr { ( power ==  FULL_ON, 1.0 ) && ( ground == FULL_ON )
}
```

```
    -simstate NORMAL}
  add_power_state PD_sub1.primary
    -state {SUB1_ON –logic_expr {pwr_ctrl ==1}
    { -supply_expr { ( power == FULL_ON, 1.0 ) && ( ground == FULL_ON, 0 ) }
    –simstate NORMAL}
  add_power_state PD_sub1.primary
    -state {SUB1_OFF –logic_expr {pwr_ctrl ==0}
    { -supply_expr { ( power == FULL_ON, 0 ) && ( ground == FULL_ON, 0 ) }
    -simstate CORRUPT}
```

Examples 7.3 and 7.4 are based on the UPF 1.0 and UPF 2.1 LRM specifications respectively, and are alternates of each other, representing the same information in different releases or versions of the UPF. These examples explain that `PD_sub1` and `PD_top` contain instances that are parent and child in the hierarchical tree; hence there is HDL hierarchical connectivity. As well, the power states and operating modes reveal that PD_sub1 has both the ON and OFF modes, whereas `PD_top` only has the ON mode. Hence cross power domain information is generated between `PD_sub1` and `PD_top` within the tool. Also, recall that the UPF strategies from Chap. 3 (like ISO, LS, RFF, PSW etc.) are explicitly defined in UPF with relevance to particular source-sink communication models as shown in Example 7.5.

Example 7.5 UPF Snippet for ISO Strategy and Corresponding Power Domain

```
set_isolation Sub1_iso -domain PD_sub1 \
    -isolation_power_net VDD1 \
    -isolation_ground_net VSS \
    -elements {mid_1/mt_1/camera_instance}
    -clamp_value 0 \
    -applies_to outputs
set_isolation_control Sub1_iso -domain PD_sub1 \
     -isolation_signal {/tb/is_camera_sleep_or_off_tb} \
    -isolation_sense high \
    -location parent
```

In this example, the ISO strategy is applied at the boundary of the `PD_sub1` power domain that implies a source. Obviously all signals from the domain boundary that propagate to any destination are implicitly becoming sinks; however, the source-sink model formation also coordinates with the power states as defined in Examples 7.3 and 7.4.

The last two of the tool analyzed and extracted information in the List 7.4, the cell-level and the pin-level attributes, are very crucial information in PA-Static verification. Because, as mentioned in Chap. 4, the cell-level attributes actually categorize a cell whether it is ISO or LS or RFF etc. Hence it stands as a differentiator between a special power management MV cell and any regular standard cell.

The pin-level attributes, unlike the PA-SIM, PA-Static require only the PG-pin information, which are listed below.

List 7.5 PG Pin-Level Attributes of Liberty Libraries

```
pg_pin,
pg_type,
related_power,
related_ground,
bias_pin,
related_bias,
std_cell_main_rail.
```

This is already discussed in Chap. 4. Only the '`power_down_function`' is exclusive for PA-SIM, and it is not required for PA-Static verification.

So once all these different categories of information are extracted and analyzed within the tool, the PA-Static verification or the checks can be started. As shown in Sect. 1, the static verification or checking criterion are different for different design abstraction levels; hence, the tool may conduct verification as early as from the RTL with only the first five of the listed seven analyzed information in the List 7.4, (i.e., power domains, power domain boundary, power domain crossings, and power states). The last two, the cell and pin-level attributes, are mandatory for the GL-netlist and PG-netlist levels of the design.

Nevertheless, the static checks that are conducted at a higher level of design abstraction, such as the RTL, must be repeated exactly at lower levels; at the GL-netlist and PG-netlist levels on top of their own dedicated checks; just to ensure and achieve consistent PA-Static verification results throughout the entire verification process. However, conducting checks appropriate for GL or PG-netlist at the RTL, will definitely not provide target results. This is because PA-Static checks at the RTL are limited to the Power Intent Syntax and UPF Consistency Checks and Power Architectural checks, for ISO, LS, ELS, RFF strategy definitions only, as listed in Sect. 1, List 7.2.

In general, the PA-Static tool performs verification on the collected, extracted, and analyzed information along with the built-in MV or PA rules. The methodology that the tool imposes for matching the built-in MV or PA rules with the UPF specifications, physical design, library attributes, and analyzed information are based on the following aspects.

List 7.6 PA-Static Rule Versus UPF Specification Analytical Approaches

- **UPF Strategy**: PA rules imply to check if UPF strategy definitions are correct, incorrect, missing, or redundant.
- **Power Management Special MV Cell**: PA rules imply to check if special MV cells are correct, incorrect, missing, or redundant.

Alternatively, the checks are performed for cross comparing the above two in the following manner as well.

– **UPF Strategies and MV Cell Cross Comparison**: Whether the UPF strategies are present but cells are missing, or vice versa.

Here, the correct, incorrect, missing, and redundant cells for any of the above **UPF Strategy** or **MV Cell** aspects are not only for the syntax and semantics definition of MV cells and MV cell type checks, but are also responsible for checking the locations – including the domain boundary interface, ports, nets, and hierarchical instance path – where the strategies are applied or cells are actually inserted.

However, the tool sometimes may not perform checks on certain cases, specifically where the power states or PST states between any source-sink power domain communication models are missing. Hence, such situation are usually referred to as **Not-Analyzed**.

In addition, the PA-Static checker also requires conducting checks on the control and acknowledgement signals of these UPF strategies, like ISO, ELS, RFF, and PSW etc. Specifically to ensure that control signals are not originated from relatively OFF power domains instead of the locations where the strategy is applied or cells actually reside. Besides, it is also required to confirm following aspects for the control signals.

List 7.7 PA-Static Check Requirements for the UPF Strategy's Control Signals

– Must not cross any relatively OFF power domains,
– Must not originate or driven from any relatively OFF power domain,
– Must not propagate unknown values,
– Possess the correct polarity, and
– Reachable.

As well, it is also required to perform checks on the strategies to ensure that:

– Strategies are not applied and MV cells are not physically inserted on the control signal path of another MV cell, or
– On any control signal paths of the design, like the scan control.

Further, it need to ensure that the MV cells are not inserted:

– On or before any combinational logics on a source-sink communication path, design clock, design reset, pull-up and pull-down nets, as well as on nets or ports with constant values specifically at the RTL, which may become pull-up or pull-down logic later in synthesis.

It is also worth mentioning here that the PA-Static checks at the RTL for physical presence of MV cells, their types (AND, OR, NOR, Latches), or their locations will generate false erroneous results.

This is because such cells are available only after synthesis or only if they are manually instantiated as Mixed-RTL. Although a PA-Static checker provides options for virtually inferring cells at the RTL as appropriate, based on UPF definitions and tool internal analytical capabilities, it may be worth either ignore or switch off such design checks or inferring options and focus on categorized checks for RTL as listed in Sect. 1, List 7.2.

7.3 PA Static Checks: Library Processing

As mentioned in Sects. 1 and 2, cell-level and pin-level attributes from Liberty are mandatorily required for accurate PA-Static verification at the GL-netlist (post-synthesis) and PG-netlist (post P&R) levels of the design. Recalling the generic and specific Examples 4.1 and 4.2 of Liberty LS cell in Chap. 4, the following are known as special cell-level attributes that categorize this particular cell as LS.

Example 7.6 Liberty Cell-Level Attributes for LS

```
is_level_shifter: true,
level_shifter_type: HL_LH,
input_voltage_range,
output_voltage_range.
```

The PA-Static checker searches for these attributes to identify the cell as LS as well as the operating voltage ranges of the LS. From Chap. 4, Example 4, the voltage ranges are from `1.2 to 0.8` volt for the input of LS which is HL (High-to-Low) and from `0.8 to 1.2` volt for output which is LH (Low-to-How). However, this could be the opposite for different LS configurations, and hence the general operating voltage ranges from `0.8 to 1.2` volt.

All other remaining attributes are termed as pin-level attributes as shown in List 7.5. The PA-Static checker tool collects the primary power and ground (as well bias) pin or port information from `pg_pin` and `pg_type` attributes together.

The '`related_power/ground_pin`' or '`related_bias_pin`' provides the related power, ground, or bias supply connectivity information for each input or output logical port or pin of the cell. Related supply is augmented with `pg_pin` and `pg_type` attributes that indicate the functionality of the supply whether it is primary power, primary ground, or an N or P-WELL bias pin, as shown below.

Example 7.7 Related Power, Ground or Bias Pin of LS

```
pg_pin(VNW) {pg_type : nwell;
pg_pin(VPW) {pg_type : pwell;
pg_pin(VDDO){pg_type : primary_power ;
pg_pin(VSS) {pg_type : primary_ground ;
pg_pin(VDD) {pg_type : primary_power ;
std_cell_main_rail   : true ;
....
pin(A) {
related_power/ground_pin : VDD/VSS ;
related_bias_pin : "VNW VPW";
level_shifter_data_pin : true ;
....
```

Hence, for multi-rail cells, specifically the MV and Macro cells – like the LS in this Examples 4.1 and 4.2 (which is a MV cell) – usually possesses different related

supplies for the input, which is pin (A) with related supplies (VDD/VSS), and the output, which is pin (Y) with related supplies (VDDO/VSS).

The 'std_cell_main_rail' attribute defines the primary power pin (VDD) that will be considered as the main rail, a power supply connectivity parameter which is required when the cell is placed and routed at the post P&R level. However, at the GL-netlist, the PA-Static checker utilizes this information for analyzing the main or primary power of the MV or Macro cells.

The std_cell_main_rail checks are done based on the following Liberty attributes.

Example 7.8 Liberty Syntax for std_cell_main_rail

```
pg_pin(VDD) {
voltage_name : VDD;
pg_type : primary_power;
std_cell_main_rail : true;
}
pg_pin(VDDO) {voltage_name : VDDO;
pg_type : primary_power;
}
```

The std_cell_main_rail attribute is defined in a primary_power power pin. When the attribute is set to True, the pg_pin is used to determine which power pin is the main rail in the cell. This is VDD in the above Example 7.3.5. Actually the implementation (synthesis) tool looks at the std_cell_main_rail (not the voltage_name) specifically for LS (and Macro), and connects or inserts LS accordingly in the design.

The Example 7.9 shows the snippet of the results for std_cell_main_rail analysis from PA-Static verification.

Example 7.9 Snippet of PA-Static Verification Result for std_cell_main_rail

```
VDD
std_cell_main_rail: true,      File: ls.lib (15)
pg_type: primary_power,     File: ls.lib (13)
VDDL
pg_type: primary_power,      File: ls.lib (18)
VSS
pg_type: primary_ground,     File: ls.lib (22)
```

Although, it is categorically mentioned, in Sect. 2, that the PA Sim-model library requirements are optional for a PA-Static tool for conducting verification and are used only for compilation purposes. However, the PA-Static tool conducts consistency checks between the PA Sim-model libraries and its corresponding counterpart

in the Liberty library to ascertain whether the Sim-model library is power aware. The consistency checks compare the power supply port and the net or pin names for all the power, ground, related, and bias pins.

The PA-Static checker further reveals the logic pin equivalency between these two libraries since related power and ground information are relevant to the logic ports. If the supply and logic ports or pins for both the libraries are matching, the simulation model library is regarded as power aware (or PA Sim-model library). However, the `power_down_function` are not compared between Sim-model and Liberty libraries, as the corruption semantics through the model or Liberty `power_down_function` is exclusively driven by PA-SIM and such mechanism is explained in Sect. 5.5, Chap. 5.

7.4 PA Static Checks: Verification Practices

The PA verification fundamentals are already discussed and a persuasive foundation of PA-Sim and PA-Static verification platforms are established through Chaps. 5 and 6, and previous sections of this Chapter. It is revealing that PA methodologies and techniques impose enormous challenges in functional and structural paradigms of the design under verification.

However, it is observed that a clear perception of the design and power specification, power intent, adopted verification techniques, inherent tool features, and subtle methodologies qualifies for the successful accomplishment of power aware verification. Even though the structural issues affect a design physically in architectural and microarchitectural aspects, however the following standpoints simplifies the PA-Static verification procedures in such perspective.

List 7.8 Simplified PA-Static Verification Standpoints

- Identifying the exact verification criterion for every design abstraction level,
- Understanding the input requirements of the tools,
- Grasping a clear conception of tool internal analytical approaches,
- Realizing the mechanism of static deployment of internally built-in MV rules on the design

Evidently these aspects will ensure the achievement of clean PA design in terms of architectural and microarchitectural perspective.

Unlike the Questa® PA-SIM which requires three-step flow for dynamic verification (discussed in Chap. 5), the PA-Static verification tool procedure is based on only two-steps: compile and optimize.

Since the compilation criterion of a design under verification is exactly the same for both PA-SIM and PA-Static, as well the optimization is the phase for both PA verification tools, where the power aware objects (e.g. Power Domain, Power Supply, Power Strategies etc.) are articulated throughout the compiled design from the UPF.

Obviously, testbenches are unnecessary during compilation of design targeted for PA-Static verification. Also it is important to note that here are certain PA-Static specific tool procedures available at the optimization phase. These procedures ensure appropriate extraction and accumulation of power information from the UPF, Liberty, and design to conduct internal analysis and impose built-in or user-specified external MV or PA rules on the design accordingly. The PA-Static related special commands and options are based on several aspects as follows:

List 7.9 Fundamental Aspects to Drive PA-Static Verification

– Verification objectives and extents,
– Input requirements for the tool,
– Contents and extents of output results

These are almost similar to PA-SIM (as shown in Sect. 5.5, Chap. 5), only the inclusion of "Debugging capabilities" are redundant for PA-Static, since the results of static verification are available at the optimization phase. However, there are different result reporting verbosity levels for PA-Static that are generated in the optimization phase and are discussed in succeeding sections.

The design entry requirements for PA-Static are also exactly the same as for the PA-SIM, as explained in Chap. 5. PA-Static also requires a complete HDL representation of the design in Verilog, SystemVerilog, VHDL, or any mix combination of these languages. Though it is highly recommended that the HDL design entries are in synthesizable RTL, Gate-Level netlist, PG-netlist, or any combination of these forms. The first step is to compile the design through `vlog` or `vcom` commands for Verilog & SystemVerilog and for VHDL respectively.

The next phase, optimizing the compiled design through the `vopt` command is the most crucial part for static verification. Similar to PA-SIM, `vopt` for PA-Static processes the UPF power intent specifications, Liberty libraries and accepts all other power-related verification commands and options as an arguments. The Questa® PA-Static typical commands and options format is shown below.

Example 7.10 Typical Command Format for Standard PA-Static Flow

```
Compile: vlog -work work -f design_rtl.v
Optimize: vopt -work work \
     -pa_upf test.upf \
     -pa_top "top/dut" \
     -o Opt_design  \
     -pa_checks=s \
     <Other PA commands>
```

To note, for the vopt procedure, `-pa_checks=s`, here "`s`" stands for all the possible and available static checks within the PA-Static tool. However, fine-grain control options allows conducting or disabling any particular checks, like conducting only ISO relevant checks, are possible through the following tool procedures at vopt.

Example 7.11 Controlling and Conducting Specific PA-Static Checks for ISO Only
Compile: No Change

Optimization: `vopt-` requires to add "`vopt -pa_checks=smi, sri, sii, svi, sni, sdi, si`", along with all other commands and options as required.

Similarly for conducting PA-Static verification on a GL-netlist and PG-netlist, the following tool procedures are required.

Example 7.12 Conducting PA-Static Checks on GL-netlist
Compile: No Change

Optimization: `vopt-` requires to add "`vopt -pa_checks=s+gls_checks`".

Example 7.13 Conducting PA-Static Checks on PG-netlist
Compile: No Change

Optimization: `vopt-` requires to add "`vopt -pa_enable=pgconn`", along with "`vopt -pa_checks=s+gls_checks`" and all other commands and options as required.

As mentioned earlier in Sect. 2, the Liberty or (.lib) file is mandatory for PA-Static checks, specifically from the GL-netlist up to the PG-netlist, or even for Mixed-RTL. Fortunately, the tool procedures for Liberty processing during PA-Sim, as discussed in Sect. 5.8, Chap. 5, are exactly the same for PA-Static. The detail for tool procedures for Liberty processing at GL-netlist are explained in Sect. 5.8.

7.5 PA Static Checks: Static Checker Results and Debugging Techniques

PA-Static verification reporting of results are very straightforward and widely differ from PA-SIM in terms of format, contents, and representation. PA-Static reporting is mostly text based; however, visual representations of PA relevant logics and relevant UPF objects on schematic viewers often adds advantages for pin-pointing bugs and fixing them quickly.

PA-Static verification are mainly UPF strategy oriented, like PSW, ISO, ELS, LS, RFF, and RTP etc. These are defined within the UPF explicitly through different UPF strategy commands, like **`set_isolation, set_level_shifter,`** etc. and their association with different power domains are validated through their definition and **`add_power_state`** or PST states, as discussed in Sect. 2.

As the PA-Static tool conducts internal analysis on these UPF definitions and objects and formalizes the source-sink communication models, it is required to represent this information in an organized and simple searchable format.

While reviewing the PA-Static reports and results to pursue structural anomalies efficiently, it is required to fully comprehend the contents and context of these reports, the source of the contents, organization format, special terminologies, relevancy, and correlation between the represented information, and so on. Based on the discussion in earlier sections of this Chapter, it is clear that PA-Static checks are originated and relevant to the following different resources.

List 7.10 UPF and Design Objects Resources Relevant to PA-Static Checks
- Power Domains, Domain Boundary, Domain Interface, HighConn, and LowConn side of ports, Domain Crossings, Source-Sink Communication Models,
- The relevant Power States for Supply Sets, Supply Ports, and Supply Nets,
- Power States from **`add_power_state`** or PSTs,
- Design abstraction levels (RTL, GL-netlist, PG-netlist etc.),
- Tool built-in MV and PA rules,
- Cell and Pin-level Attributes from Liberty.

These resources are for presenting quality PA-Static results, and they are very generic as well as comprehensive for any PA-Static verification environment. The organization chart of reports and results further depends on the analytical approaches and processing capabilities of the tool which can be best summarized as follows.

List 7.11 UPF and Design Objects Resources Relevant to PA-Static Result Representation
- **Syntax and Semantics of UPF definition** – Revealing syntactic and semantic correctness of power intent in UPF files through referencing appropriate revision and release of UPF LRM,
- **UPF Strategy vs. MV Cell** – Revealing correct, incorrect, missing , redundant, and not analyzed cells or strategies,
- **UPF Strategy and MV Cell Mix** – Revealing location, type, control signal polarity, library consistency, and PG-pin connectivity of the MV cells,
- **Power Supplies** – Revealing MV cell relevant supply set, supply port, supply net,
- **Design Attributes** – Revealing design control signals (like scan control), clock, reset, combinational logic path, etc. are safe for deployment of UPF strategy or MV Cells

Considering the exhaustive lists of resources and reporting organization approaches mentioned above, it is required to represent PA-Static results according to design abstraction levels and on the basis of UPF, Liberty, and design analysis. It is also evident that static verification reporting is overly mediated through UPF strategies and corresponding MV cells. Hence, PA-Static verification tools are often considered as ISO, ELS, LS, PSW, RFF, and RPT etc. checkers. Though the consideration is not wrong, the scope of the PA-Static verification are generally extended beyond checking only UPF strategies. The checker also validates UPF semantics and Liberty library consistency.

Thus reflecting all the facts and features about PA-Static verification discussed so far, it is evident that a standard PA-Static tool prevails at the minimum following set of reports for efficient debugging to fully leverage structurally clean design.

List 7.12 Minimum Requirements of PA-Static Reports for Efficient Debugging

Power Intent and UPF Consistency Reports:

- Power Domains (PDs)
- Hierarchical path of the created scope
- Primary power ground of PDs
- Supply set handles associated with PDs
- Reference ground of supply set

UPF Strategy (considering ISO as an example) Reports:

- ISO strategy name and Relevant UPF file name,
- ISO supplies,
- ISO control, sense, clamp value,
- Isolated signals,

PA Design Elements Reports:

PA Static Reports: (Contents may vary depending on design at RTL, GL-netlist, or PG-netlist)

MV and Macro Cell Reports:

MV and Macro Cell Connection Reports:

Power States and PST Analysis Reports:

The following examples shows the practical UPF ISO strategy and sample results or reports for the ISO strategy (CAM_iso) checks.

Example 7.14 Sample of PA-Static Reporting for ISO Checks

```
---- Snippet of UPF file for ISO Strategies ----
set_isolation CAM_iso -domain PD_camera \
   -isolation_power_net VDD_cm \
   -isolation_ground_net VGD \
   -elements {mid_1/mt_1/camera_instance} \
   -clamp_value 0 -applies_to outputs
set_isolation_control CAM_iso -domain PD_camera \
   -isolation_signal /tb/is_camera_sleep_or_off_tb \
   -isolation_sense high \
   -location parent
map_isolation_cell CAM_iso -domain PD_camera -lib_cells {ISO_LO}
```

Explanation of UPF Strategies The snippet UPF code contains the definition of the ISO strategy name `CAM_iso`. The ISO cell is and AND type (clamp_value 0) and is located on the parent domain of `PD_camera`.

Example 7.15 Snippet of PA Architecture Report File for CAM_iso Checks

```
     Power Domain: PD_camera, File: ./mobile_top.upf(7).
             Creation Scope: /tb
             Extents:
                  1. Instance : ~/camera_instance
                Isolation Strategy: CAM_iso, File: ./mobile_top.
upf(64).
               Isolation Supplies:
                            power  : /tb/VDD_cm
                            ground : /tb/VGD
              Isolation Control (/tb/is_camera_sleep_or_off_tb),
Isolation Sense (HIGH), Clamp Value (0), Location (parent)
             Signals with default isolation cells:
            1. Signal : ~/camera_instance/out_data, isolation cell
: ~/out_data_UPF_ISO
             Signals with -instance isolation cells:
            1. Signal : ~/camera_instance/out_data, isolation cell
: ~/iso_cm_o
             2. Signal : ~/camera_instance/camera_state[0], isola-
tion cell : ~/iso_cm_s0
             3. Signal : ~/camera_instance/camera_state[1], isola-
tion cell : ~/iso_cm_s1
```

Explanation of PA Architecture Report The snippet shows all PA architecture information, like PD, relevant UPF file, hierarchical instance scope of PDs, ISO supply, ISO control, type, strategy deployed locations, and instance lists of already inserted ISO cells that may be essential to debug ISO issues later reported in a static report file.

Example 7.16 Snippet of PA Design Element Report File

```
     PD_camera.CAM_iso-instance: {Path7} = scope ~/iso_cm_s1
     PD_camera.CAM_iso-instance: {Path8} = scope ~/iso_cm_s0
     PD_camera.CAM_iso-instance: {Path9} = scope ~/iso_cm_o
     PD_camera.CAM_iso-instance: {Path7}/Z
     PD_camera.CAM_iso-instance: {Path8}/Z
     PD_camera.CAM_iso-instance: {Path9}/Z
```

Explanation of PA Design Element Report The design elements paths are listed in this reports which are relevant to the scope of instantiated ISO cells.

Example 7.17 Snippet of MV and Macro Cell Connectivity Report File

```
     Verification Model File : isolation.v(4)
        Liberty Model File : isolation.lib(7)
        is_isolation_cell : true, File : isolation.lib(7)
        Ports :
          ISO_EN, File : isolation.v(4)
               direction : input
               related_power_pin : VDD, File : isolation.lib(27)
               related_ground_pin : VSS, File : isolation.lib(26)
              isolation_cell_enable_pin : true, File : isolation.
lib(25)
          I, File : isolation.v(4)
               direction : input
               related_power_pin : VDD, File : isolation.lib(21)
               related_ground_pin : VSS, File : isolation.lib(20)
                isolation_cell_data_pin : true, File : isolation.
lib(19)
          Z, File : isolation.v(4)
               direction : output
               functionality : ((~ISO_EN) & I), File : isolation.
lib(32)
              power_down_function : ((~VDD) | VSS), File : isola-
tion.lib(31)
               related_power_pin : VDD, File : isolation.lib(34)
               related_ground_pin : VSS, File : isolation.lib(33)
          VDD, File : isolation.lib(9)
               direction : input, File : isolation.lib(9)
               pg_type : primary_power, File : isolation.lib(10)
          VSS, File : isolation.lib(13)
               direction : input, File : isolation.lib(13)
               pg_type : primary_ground, File : isolation.lib(14)
```

Explanation of MV and Macro Cell Connectivity Report The reports in this file are extracted from the Liberty of corresponding MV (ISO in this example) and Macro cells.

Example 7.18 Snippet of PA Static Report File

```
+=================================================================
==============================+
||                          Domain Crossing Checks Summary report
+=================================================================
==============================+
 Check-ID                                Count            Severity
Description
-----------------------------------------------------------------------------
 ISO_VALID                        7          Note            Valid
isolation cells
-----------------------------------------------------------------------------
+=================================================================
==============================+
||                                     GLS Checks Summary report
+=================================================================
==============================+
 Check-ID                 Count     Severity          Description
-----------------------------------------------------------------------------
ISO_VALID_CELL_WITH_STRATEGY    7           Note           Valid
Isolation cell and strategy
-----------------------------------------------------------------------------
ISO_VALID_STRATEGY_NO_CELL     1          Note         Isolation
strategy with NO cell
-----------------------------------------------------------------------------
ISO_CTRL_REACH_DATA              3            Warning   Isolation
cell with control reaching data
-----------------------------------------------------------------------------
+=================================================================
==============================+
||                            Domain Crossing wise static Checks
Detailed report                                        +===========
=================================================================
==================+
-----------------------------------------------------------------------
2                                                                    .
S           o           u           r           c           e
power domain: PD_camera to Sink power domain: PD_mobile_top.
    Total 4(1*) Valid isolation cells
    2.1. Source port: ~/camera_instance/camera_state[0] [LowConn]
to Sink port: ~/iso_cm_s0/Z [LowConn], width:1
    Total 1 Valid isolation cells
    2.1.1. Inferred type: ISO_VALID, count: 1
     Candidate Port: ~/camera_instance/camera_state[0], Isolation
Cell: ~/iso_cm_s0(ISO_LO), Strategy: CAM_iso,  Domain: PD_camera
```

```
     Possible reason: 'Isolation is required and is present from
(PD_camera) => (PD_mobile_top)'
    Analysis link: [PD1_to_PD2]

+================================================================
================================+
||                                      GLS Checks Detailed report
================================================================
================================+
+................................................................
.............+
||                                            ----  ISOLATION  CELLS
----                           ||
+................................................................
.............+
+--------------------------------------------------+
|| DESIGN CELL : ~/iso_mem_s1(ISO_LO) ||
+--------------------------------------------------+
    CELL TYPE     : Isolation Cell
    LIBERTY INFO  : Model : ISO_LO,
     Verification Model File: ~/libraries/isolation/isolation.v(4),
        Liberty Model File     : ~/libraries/isolation/isolation.
lib(7)

    CHECK TYPE: ISO_CTRL_REACH_DATA
        STRATEGY        : MEM_iso, File: ./mobile_top.upf(68)
       POWER DOMAIN    : /tb/PD_memory, File: ./mobile_top.upf(8)
       SOURCE PORT     : ~/internal_memory/mem_ls_lh3/I (PD: /tb/
PD_memory)
        SINK PORT       : ~/iso_mem_s1/Z (PD: /tb/PD_mobile_top)
       ANALYSIS REASON : Isolation is required and is present from
(PD_memory) => (PD_mobile_top). However ISO enable signal for UPF
strategy reaches the data pin of the cell

+----------------------+
|| MISSING DESIGN CELL ||
+----------------------+

    CHECK TYPE : ISO_VALID_STRATEGY_NO_CELL
        STRATEGY        : CAM_iso, File: ./mobile_top.upf(64)
       POWER DOMAIN    : /tb/PD_camera, File: ./mobile_top.upf(7)
         SOURCE PORT      : ~/camera_instance/out_data (PD: /tb/
PD_camera)
         SINK PORT        : ~/camera_instance/out_data (PD: /tb/
PD_mobile_top)
       ANALYSIS REASON : Isolation is required and is present from
```

`(PD_camera) => (PD_mobile_top). However corresponding cell is not found, hence inferred`

Explanation of PA Static Report This is the key report that summarizes the results of static checks specifically on MV cells. However, when there are issues and anomalies, the corresponding issues are clearly explained in detail in several subsections.

For example, the results report `ISO_CTRL_REACH_DATA` with severity Warning, which requires the reports to be addressed and issues possibly corrected. The ISOLATION Cells subsection provides every possible detail of information relevant to the issues for the cell with the following sub header;

`CHECK TYPE: ISO_CTRL_REACH_DATA.`

It is clear that the issue occurred for a cell, which is Isolation cell, Library cell name is `ISO_LO`, strategy name `CAM_iso`, relevant power domain is `PD_camera`. It also provides the source and sink port names, which are `~/camera_instance/out_data and ~/iso_cm_o/Z` respectively.

As well it identifies the source-sink communication models, which are `(PD_camera) => (PD_mobile_top)`. Hence in order to resolve the issue, it is required to understand the inherent meaning of the "Mnemonic Identifier" `ISO_CTRL_REACH_DATA`. Literally and technically this mnemonic refers that "a valid isolation strategy and cell are present from `(SRC PD) => (SINK PD)`. However, ISO enable signal for UPF strategy reaches the data pin of the cell.

The succeeding report files actually facilitate pursuing the static issues and are equally important in the debugging phases.

Example 7.19 Snippet of UPF, HDL and Liberty Connectivity Report File

```
     -- This is the report of connections between UPF, HDL and
     -- liberty files.
     -- Format :
     -- Hierarchical path of net/port [UNIQUE ID] <pg_type> <LHS
VCT> <=/<=> <RHS VCT> Hierarchical path of net/port [UNIQUE ID]
<pg_type>
     -- where :
     -- UNIQUE ID : Unique id for UPF/HDL/Liberty nets/ports
     --               UPFSN# ==> UPF supply net
     --               UPFSP# ==> UPF supply port
     --               UPFCP# ==> UPF control port
     --               UPFAP# ==> UPF acknowledge port
     --               UPFLP# ==> UPF logic port
     --               UPFLN# ==> UPF logic net
     --               HDL# ==> HDL port/net
     --               LIB# ==> LIBERTY port/net
     -- <= : LHS net/port is driven by RHS net/port
     -- <=> : Inout connection
     -- LHS VCT : VCT used for LHS side of connection
     -- RHS VCT : VCT used for RHS side of connection
     ------------------------------------------------------------
    /tb/PD_camera.CAM_iso.PWR [UPFSP2] <= /tb/VDD_cm [UPFSN2]. ./
mobile_top.upf(64)
     /tb/PD_camera.CAM_iso.GND [UPFSP5] <= /tb/VGD [UPFSN3]. ./
mobile_top.upf(64)
     /tb/PD_camera.CAM_iso.isolation_signal [UPFCP1] <= /tb/is_
camera_sleep_or_off_tb [HDL1]. ./mobile_top.upf(65)
```

Explanation of UPF, HDL and Liberty Connectivity Report File This file facilitates identifying that the `isolation_signal` which is a control port [UPFCP1] defined in UPF file name "`mobile_top.upf`" line number 65, is connected to HDL [HDL1] port "`is_camera_sleep_or_off_tb`" of design instance "`/tb`", specifically (`/tb/ is_camera_sleep_or_off_tb`).

Checking the "mobile_top.upf" line 65 reveals following information:*Line 65 of mobile_top.upf: set_isolation_control CAM_iso -domain PD_camera -isolation_signal* ***/tb/is_camera_sleep_or_off_tb*** *-isolation_sense high -location parent*

However, checking the actual HDL design, where the ISO cell "ISO_LO" is instantiated with instance name "iso_cm_o", is as follows: *ISO_LO iso_cm_o (.I(camera_out_data),*

.ISO_EN (is_camera_sleep_or_off), .Z(camera_out_data_isolated));

As well this is also prominent from the information provided on the PA Static Report File, where the sink port is as follows:*SINK PORT: ~/iso_cm_o/Z (PD: /tb/PD_mobile_top)*

Since the ISO strategy is applied at the output of PD_camera (~/camera_instance), and cell location is specified to be parents from the UPF file, as well as "ISO_EN" is the enable port for the ISO cell as noted in Example 7.17 **Snippet of MV and Macro Cell Connectivity Report File**, hence it is required to modify the HDL instance. *ISO_EN (is_camera_sleep_or_off)* to correct the `ISO_CTRL_REACH_DATA` Warning.

Once corrected, it is required to rerun the design and confirm that the ISO_EN or isolation control signal is connected to the proper HDL port or net.

Example 7.20 Snippet of PST Analysis Report File

```
3. PD_camera -> PD_mobile_top [PD1_to_PD2]:
    Isolation: Required
    Level shifter: Not Required
      Maximum voltage difference: 0.00 V
    Details:
      Analysis between supplies:
       Source Supply: VDD_cm, drives PD_camera.primary.power
       Source Supply: VSS, drives PD_camera.primary.ground
       Sink   Supply: VDD, drives PD_mobile_top.primary.power
       Sink   Supply: VSS, drives PD_mobile_top.primary.ground
          Reason: Same source and sink GROUND supplies.
       PST used for analysis: /tb/top_pst
      +-------------+------------------+
      | State(s)    | VDD_cm   | VDD   |
      |             | {Source} | {Sink} |
      +-------------+------------------+
      | {ON}        | ON_1_0   | ON_1_0 |
      | {SLEEP,OFF} | OFF      | ON_1_0 |Iso
      +-------------+------------------+
```

Explanation of PST Analysis Report File PST analysis reports either from PST states or from **`add_power_states`** provide the logical proposition of UPF strategy requirements as shown above. Here `VDD_cm` and `VDD` are power supplies for source and sink respectively, and there is a state where `VDD_cm` requires to go OFF while VDD is still ON. Hence ISO requirements are marked adjacent to the tabular format. Though this may appear insignificant for a small number of power domains and combinations of their states, but for large number, this analysis report appears to be an accommodating resource.

In addition, although the verification provides different severity levels, like Error, Warning, Info or Note, there are different schemes to handle the severity levels. Primarily Error and Warnings are subject to be properly addressed and fixed, while

there are certain classes of Warnings which may be waived, suppressed, or downgraded to Info or Notes depending on the sources of the issues. As well they are usually resolved on a case by case basis. These classes of Warnings may broadly originate from DFT, Scan insertion, ECO etc. since they are often performed after synthesis or after major MV cell physical insertion processes.

The primary objective of any verification environment is to achieve a bug-free design for implementation. Similarly, PA-Static verification requires to ensure that the design is physically correct in terms of power architecture and microarchitecture. It is evident that efficient debugging and fixing of issues, bugs, and anomalies are dependent on clear perceptions of the design specification, power specification, tool techniques, and methodologies, as well as comprehensive understanding of the reporting mechanism. Although UPF imposes an extra layer of physical complexity on PA design but static verification does not require a testbench, stimulus as well input requirements are simple and straightforward. The PA-Static verification is mandatory along with PA-SIM throughout the entire DVIF.

Epilogue: This chapter offered readers a consolidated approach to understand verification tools and methodologies that applies a set of pre-defined power aware (PA) or multi-voltage (MV) rules based on the power requirements, statically on the structures of the design. More precisely, the rule sets are applied on the physical structure, architecture, and microarchitecture of the design, in conjunction with the UPF specification but without the requirements of any external stimulus or testbenches. This actually makes the PA-Static tool simpler and popular choice to verify various features of UPF strategies (e.g. UPF strategy definitions are correct, incorrect, missing, or redundant, MV cells are correct, incorrect, missing, or redundant, UPF strategies are present but cells are missing, or vice versa).

The chapter also explained that PA-Static checks works beyond UPF strategies. The List 7.2 **Essential PA-Static Checks at Different Design Abstraction Level**, is the best reference for complete list of PA-Static checks. In addition, List 7.4 **Summary of Information Analyzed by PA-Static Tool**, listed that the tool may be deployed to conduct verification as early as from the RTL with the first five of the listed information (i.e., power domains, power domain boundary, power domain crossings, and power states). The last two, the cell and pin-level attributes, are mandatory information for the GL-netlist and PG-netlist levels of the design. The Sect. 3 revisits the library requirements and processing techniques for PA-Static tools. This chapters finally closes with a real example to analyze PA-Static results and reporting, as well efficient debugging PA-Static anomalies.

Bibliography

1. Khondkar, P., Yeung, P., et al.: Free Yourself from the Tyranny of Power State Tables with Incrementally Refinable UPF. DVCon' (February–March 2017)
2. Design Automation Committee of the IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group: IEEE Standard for Design and Verification of Low-Power Integrated Circuits. Revision of IEEE Std 1801–2009, 6 March 2013
3. Design Automation Standards Committee of the IEEE Computer Society: IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems. IEEE Std 1801–2015, 5 December 2015
4. Marschner, E., Biggs, J.: Unleashing the Full Power of UPF Power States. DVCon' (2015)
5. Prasad, D., Bhargava, M., Bansal, J., Seeley, C.: Debug Challenges in Low-Power Design and Verification. DVCon' (2015)
6. Bhargava, M., Gairola, P.: Power State to PST Conversion: Simplifying Static Analysis and Debugging of Power Aware Designs. DVCon' (2016)
7. Wei Tu, S., Lin, T., Feng, A., Ping, C.Y.: UPF Code Coverage and Corresponding Power Domain Hierarchical Tree for Debugging. DVCon' (2015)
8. Vikram, V., Awashesh K.S.: Cross Coverage of Power States. DVCon' (2016)
9. Desinghu, P.S., Khan, A., Marschner, E., Chidolue, G.: Refining Successive Refinement: Improving a Methodology for Incremental Specification of Power Intent. DVCon' Europe (2015)
10. Bhargava, M., Gairola, P.: Power State to PST Conversion: Simplifying Static Analysis and Debugging of Power Aware Designs. DVCon (2015)
11. Dwivedi, P.K., Srivastava, A., Vikram S.V.: Lets disCOVER Power States. DVCon' (2015)
12. Khondkar, P.: Power aware libraries: standardization and requirements for Questa® Power Aware. Verific. Horiz. J. **12**(3), 28–34 (2016)
13. Khondkar, P., Yeung, P., Prasad, D., Chidolue, G., Bhargava, M.: Crafting power aware coverage: verification closure with UPF IEEE 1801. J. VLSI Des. Verific. **1**, 6–17 (2017)
14. Khondkar, P., Bhargava, M.: The Fundamental Power States: The Core of UPF Modeling and Power Aware Verification. Whitepaper at mentor.com. (December' 2016)

P. Khondkar, *Low-Power Design and Power-Aware Verification*,
DOI 10.1007/978-3-319-66619-8