

ABOUT YOUR PERSONALIZED PDF EDITION OF THE VMM-LP

This copy of the electronic (.pdf) edition of the *Verification Methodology Manual for Low Power* (VMM-LP) version 090302 is provided to you for your use only and is electronically marked with your identification. You may not distribute this copy to others; instead, please refer them to download their own copy at: www.synopsys.com/vmmlp or www.arm.com/vmmlp.

Unauthorized reproduction or distribution is strictly prohibited. You are authorized to reproduce sections up to 10 pages for your company's use; such limited reproductions are considered reasonable use. Other reproductions may not be shared or transmitted in any form or by any means, electronic or mechanical, without the express written consent of ARM, Renesas Technology Corp., or Synopsys.

You may purchase the print edition of the VMM-LP through Amazon.com.

For more information about the VMM-LP as well as any addenda or errata published subsequent to this edition, please refer to the VMM-LP website: vmmcentral.org/vmmlp.

Verification Methodology Manual for Low Power

Srikanth Jadcherla
Janick Bergeron
Yoshio Inoue
David Flynn

ARM RENESAS SYNOPSYS

VERIFICATION METHODOLOGY MANUAL FOR LOW POWER

Srikanth Jadcherla, Janick Bergeron, Yoshio Inoue and David Flynn

HISILICON

"Low power verification is the key challenge in low power design. The VMM-LP helps create a reusable verification environment for low power that can leverage best practices from industry experts. It helps find low power bugs and finds them early in the design cycle rather than waiting for silicon – savings in terms of mask costs and engineering debug time can be huge."

Hisilicon K3 LP Group

SAMSUNG

"The task of verifying low power designs presents a significant challenge for today's verification engineers, as most are not yet well-trained on low power concepts. The Verification Methodology Manual for Low Power is a timely and valuable resource that addresses all aspects of low power verification, providing detailed rules and guidelines."

Jianfeng Liu, Senior Low Power Verification Methodology Engineer, Samsung Electronics

STARC

"We see a prevalence of low power designs in Japan and a strong need for a comprehensive verification methodology to tape out such designs with confidence. VMM-LP is the answer to this market need and completely and elegantly addresses all aspects of low power verification. The book covers what is needed to verify low power designs and get it right – the first time around."

Nobuyuki Nishiguchi, Vice President and General Manager, Development Department 1, STARC (Semiconductor Technology Academic Research Center)

SUNPLUS

"Because power consumption is one of the most critical factors of today's SoCs for mobile applications, the ability to accurately verify low power functionality is essential to achieving first-pass silicon success. The Verification Methodology Manual for Low Power is a comprehensive collection of necessary and reliable techniques that should help simplify and accelerate the complex task of verifying power-managed designs."

Ying-Chih Yang, Technical Director of Home Entertainment Products, Sunplus Technology

Verification Methodology Manual for Low Power

Verification Methodology Manual for Low Power

Srikanth Jadcherla
Synopsys, Inc.

Janick Bergeron
Synopsys, Inc.

Yoshio Inoue
Renesas Technology Corp

David Flynn
ARM Limited

Srikanth Jadcherla
Synopsys, Inc.
Mountain View, CA
USA

Janick Bergeron
Synopsys, Inc.
Nepean, Ontario
Canada

Yoshio Inoue
Renesas Technology Corp.
Tokyo
Japan

David Flynn
ARM Limited
Cambridge
United Kingdom

Library of Congress Control Number: 2009922213

ISBN 978-1-60743-413-9

Published by Synopsys, Inc., Mountain View, CA, USA

vmmcentral.org/vmmlp

Printed in the United States of America
February 2009

TRADEMARKS

Synopsys is a registered trademark of Synopsys, Inc.

ARM and AMBA are registered trademards of ARM Limited. “ARM” is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Ltd.; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Belgium N.V.; AXYS Design Automation Inc.; AXYS Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, AS; and ARM Sweden AB.

All other brands or product names are the property of their respective holders.

DISCLAIMER

All content included in this Verification Methodology Manual for Low Power is the result of the combined efforts of ARM Limited, Renesas Technology Corp., and Synopsys, Inc. Because of the possibility of human or mechanical error, neither the authors, ARM Limited, Renesas Technology Corp., Synopsys, Inc., nor any of their affiliates guarantees the accuracy, adequacy or completeness of any information contained herein and are not responsible for any errors or omissions, or for the results obtained from the use of such information. THERE ARE NO EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE relating to the Verification Methodology Manual for Low Power. In no event shall the authors, ARM Limited, Renesas Technology Corp., Synopsys, Inc., or their affiliates be liable for any indirect, special or consequential damages in connection with the information provided herein.

TABLE OF CONTENTS

FOREWORD

Some years ago, while working at a different company, I led the verification effort for a very complex system-on-chip (SoC). The schedule for this project was tight, and as we were to target a very specific market segment where power wasn't an issue, we were told, “Don't worry about power, just get the chip out the door!” After many months of effort, the chip taped-out nearly on time, hitting all of its original design requirements. Of course by that time the product landscape had shifted, and rather than receiving our expected praise, all we heard from our marketing team was, “If only the power numbers were better, we could target so many more applications!” After several re-spins of the chip to achieve better power numbers, the project leader later summed up in a post-mortem review meeting what he felt was the main lesson learned during the course of the project — “Power always matters.”

That statement is even truer today than it was back then. Battery life is a key product differentiator in the consumer world, and the continual push towards smaller and leakier processes makes this an even more daunting task. Even systems running on wall power are not exempt from power issues as ever increasing clock frequencies carry with them enormous issues with cooling, coupled with growing public awareness to be more eco-friendly. No matter what product you might find yourself designing today, power always matters.

In the past, many verification engineers tended to tune out whenever they heard the word “power.” Isn't that a circuit issue after all? Since you now hold in your hands a book which has both the word 'power' and 'verification' in its title, I will assume that you already understand that power is very much a verification issue, or at least you are wondering what all the fuss about low power is. While it's true that many of the low power design techniques require some fairly clever circuit design, when these

designs fail, they exhibit failure in very functional (or non-functional) ways. Clock gating, power gating, isolation cells, and state retention are features that can not simply be inspected for correctness in a design review, they must be thoroughly planned for, tested and simulated. I think many engineers would be astonished at the complexity of verifying the power management scheme of a modern SoC.

That's where this book can help. This is not a dry, academic treatise on the theory of low power verification, but rather a “roll up your sleeves and get your hands dirty” book, written by verification engineers, for verification engineers. Included are sections which discuss verification techniques using real world examples based on the authors' own experiences, along with detailed discussion about the types of bugs you're likely to uncover. More than simply discussing the issues of low power verification, this book outlines a comprehensive methodology.

VMM first burst upon the scene after publication of the “Verification Methodology Manual for SystemVerilog” in 2005[1]. Today VMM is the most widely accepted SystemVerilog class library in use. It makes sense, then, that VMM has now been extended to address the needs of the verification engineer to deal with verification issues for low power. This is the companion volume to that earlier work, and while some of the methodologies described in this book will feel like natural extensions to existing methodologies to the experienced verification engineer, others will require more careful study.

Kelly D. Larson

Mediatek Wireless, Inc.

CHAPTER 1 INTRODUCTION

ABSTRACT

Low Power Design is driven by market pressures, regulatory aspects and technology migration to 65nm and lower. Multi-Voltage design is hence essential. Multi-Voltage design brings many new challenges to design as well as verification.

1.1 INTRODUCTION

Almost every engineer who designs and verifies integrated circuits (ICs) today is under tremendous pressure to reduce power. In a rather unprecedented way, we are faced with market pressures, regulatory pressures and process technology factors to reduce power, all at once. This multi-pronged pressure on power consumption has led many designs to adopt aggressive low power design techniques that involve the control of voltage. This book looks at the emerging voltage controlled techniques used today and how this new generation of power managed designs is verified. As is often the case with most design processes, verification is often a larger task than the design effort itself. The problem with power managed ICs is worse because this is not an area where traditional verification techniques can be easily applied. As we establish later in the chapter, an altogether new view of looking at boolean logic and the verification process is needed. With this book, we endeavor to develop a re-usable, rigorous and comprehensive methodology to verify power managed designs.

For over 25 years, the Electronic Design Application (EDA) industry did not have to deal with the various complexities of voltage controlled power managed designs. This was because most IC power management was handled at the system level with little

impact to the specification and verification process and hence out of the scope of EDA tools used in the IC design process. Chip level aspects were restricted to clock gating, which caused a significant change to synthesis, placement and routing portions of the design flow almost a decade ago. These however, did not deal (and did not need to deal) with the demands of voltage based control of ICs or IC components. Both automated analysis and implementation capabilities were based on hardware description languages (HDLs) that excluded a notion of voltage connections to the underlying logic. The exclusion of voltage from HDLs and the current adoption of voltage based control of ICs are at odds with each other, causing enormous disruption in the world of IC design and verification.

Many collective years of intellectual property (IP), existing code bases, EDA tools and flows have been built on the existing paradigm. Voltage control of CMOS devices now needs to be accounted for in these databases and processes: it is the desired output of a control system involving hardware, software, digital circuits and analog blocks. The mastery of this complexity is now the job of front end RTL design and verification engineers as well as back end implementation and signoff engineers.

In the following sections we look at what is driving power management first. Then we proceed to look at how designs are responding by adopting voltage control techniques. That leads us to the main problem—verification of such design techniques. The chapter concludes with a section on the structure of the book and methodology adoption.

1.2 Driving Factors for Power Management

Semiconductors are increasingly being used in mobile/consumer devices. Mobile devices are driven by battery life and form factor primarily, though performance is equally important, especially in the consumer market and for mobile applications involving multi-media. Even in the wired power segment, there is enormous pressure to reduce form factor and deliver better performance, as the availability of energy supply and runtime electricity costs gains prominence. There is also increased awareness that the installed base of semiconductors contributes more to global warming than the entire airline industry. Hence, many governments have sought to regulate the electronics industry. Lastly, technology migration which typically helps achieve lower power consumption has complicated matters. This section looks at these three aspects—market, regulations and technology and their impact. However, before we dive into that, we need to take a deeper look at the word “power” itself.

1.2.1 A Deeper Look at The Impact of Power

Perhaps the biggest factor in power management is that the main problem is not often "power"; it is a combination of Density, Delivery, Leakage, and Lifetime. This four-pronged problem can be described as follows:

1.2.1.1 Density

Density refers to the amount of power consumed within an area, hence the heat dissipated in an area. Consider a 1cm x 1cm packaged IC dissipating an average of 1W. This alone amounts to a power density of 10GW per square kilometer, the power of many nuclear reactors combined! Needless to say, the heating produced by such a density is enormous: skin temperatures on the package often reach 100 degrees Celsius, with die/junction temperatures reaching 125 degrees. Heat dissipation leads to cost in terms of both components required to dissipate it such as a better package, heat sink, fan etc. and a run time cost in terms of operating the fan, cooling system etc. Often an overheated device collection such as a server farm incurs an equal or greater run time cost than the initial cost itself. In some extreme cases, the excess temperature being in proximity to a battery has caused fires and explosions in cell phones and laptops.

Often, people confuse density to be *the* power problem. While heat and associated costs are prominent in the bill of materials and system design, this is not the sole concern.

1.2.1.2 Delivery

Delivery is the problem relating to the amount of current that has to be delivered at progressively lower supply voltages and the ability to withstand fluctuations in current. Delivery is the most misunderstood of power requirements. However, it constitutes one of the most important aspects of power management. As process technology shrinks in its feature length, the current supplied is actually increasing; hence the fluctuations, especially the rapid ones in current requirements are especially troubling to design. Delivery is commonly classified as IR drop and di/dt problems. One of the nastiest aspects of power management is that density and leakage mitigation techniques often cause problems in delivery of current. Chapter 3 covers one such bug ("Power Gating Collapse" on page 66).

Often systems come with specifications on maximum current draw from subsystems. For instance, a USB port in a mobile computer often imposes this restriction. This

amounts to a design constraint on current delivery to any devices connected to that port.

1.2.1.3 LEAKAGE

Leakage is the current consumed by the chip even when there is no activity. In prior generations of process technology, such idle current was a negligible entity. In deep submicron designs, this is no longer the case: causing immense problems for mobile devices, where it has a severe adverse effect on battery life. Leakage is not merely a battery powered device problem. With the advent of “Green Design,” there is an immense impact of leakage on all electronic systems. This is covered under the regulatory pressures section. Note that transistors have leakage current even when there is activity.

1.2.1.4 LIFETIME

Lifetime refers to the decreasing reliability of chips caused due to higher current densities. The cross section of wires in the ICs of today is quite narrow compared to that of yesteryears. Coupled with an increase in absolute current itself, the implication is that the material in the wires is likely to degrade much faster. A reduction in current, therefore benefits lifetime concerns tremendously. Electronics sold into various markets have requirements on lifetime, often accompanied by manufacturer warranties. Hence, ensuring reliable operation for that period is a key design constraint.

Given these four ways in which the impact of power consumptions is felt, we can now look at how markets, technology migration and regulatory aspects are reacting to these issues.

1.2.2 MARKET PRESSURE TO REDUCE POWER

Semiconductor products are increasingly designed into Consumer Electronic content: mobile or otherwise. The consumer market is driven by rich multimedia experience, slim form factors and of course, long battery life in mobile devices. This market is also extremely cost sensitive—an extremely low bill of materials is desired.

In the mobile segment, density and leakage are important design considerations. Handheld devices cannot overheat (or it will lead to product failure), but cannot accommodate heatsinks and fans either. Both add to the cost of the end device and form factor. The weight of the battery is an issue as well; the lower the better. Hence, their components need to be extremely efficient in their use of energy. Further, they

may not have any wasted energy in the form of leakage. Hence, mobile devices try extremely hard to minimize both active power as well as leakage.

In the wired power segment, density is often the primary constraint. The cost of the package, heatsink and fan add significant cost to the bill of materials. Further, some multimedia devices such as TVs do not have the option of using a fan, which might disrupt the audio experience. Form factor is also an extremely important sale factor in this segment. Hence, there is a major incentive to reduce power.

In markets such as servers, enterprise networking etc., power is *the* limiting factor to increase performance. Thus lowering power actually helps boost the product into higher performance or higher port density. Further, these devices are measured by the amount of electricity consumed not just for operation, but cooling the ambience as well. Further, special wiring for power delivery is required in this segment at additional cost. Costs reduce as power consumption decreases, hence power is a critical vendor selection criteria in this market.

Market forces from the energy sector are shaping up as well. The rapid rise in energy demand across the world, especially in households has put a strain on generation and distribution in many countries, both in developed nations in Europe as well as emerging economies. This in turn has caused a rise in solar energy when feasible. One of the most important aspects of solar power is that it is locally generated in the form of direct current, not alternating current. It is often inverted into AC and supplied to the household. Ironically, the AC supply is converted back to DC within each electronics appliance that we use.

It may be surprising to many, but about 30% of the energy in such an appliance can be consumed for AC-DC conversion. Combined with the DC-AC inversion, the efficiency is further reduced: needless cost is added to the overall system. This has lead to the birth of Direct DC appliances. In fact, most of computing, communication, entertainment, networking and even lighting, heating and ventilation (with fans) can be operated from DC sources. The rise of Direct DC appliances has the advantage of eliminating cost in the overall system as well as the fact that each appliance now consumes much less power, thereby reducing the need for installed capacity as well. This cost reduction is quite significant and can reduce the cost of the solar system by almost 50% in most cases.

It is with the advent of such a technology that idle energy (mostly leakage) is simply *unaffordable*. Idle energy amounts to draining the battery of the solar system and/or preventing its charging. The amount of idle energy consumed by the system is directly related to the cost of the solar panel system and the battery capacity, a fact

that will put stringent market pressures on semiconductors to “idle efficiently,” consuming almost zero power in idle mode.

1.2.3 Technology Migration and Power

Until 90nm technology became available, the cure for power ailments was simple: migrate the chip to a lower geometry, benefiting from the lower capacitance and the lower supply voltage. The onset of leakage has spoiled that easy route, and problems in the basic scaling for delivery and lifetime have started creeping up. For a quantitative illustration, consider the integration of IC generations in Figure 1-1. Two chips A and B at 0.18um are integrated into a single chip AB at 0.13um. This reduces power overall, but notice that the current is now higher (at 0.45A), supplied at a lower voltage. When the next generation of integration with a similar Chip C, providing additional functionality, is done at 90nm, chip ABC consumes about 0.7W, as opposed to say, an expected 0.4-0.5W. This is problematic because scaling down process generations is broken at 90nm and beyond. s

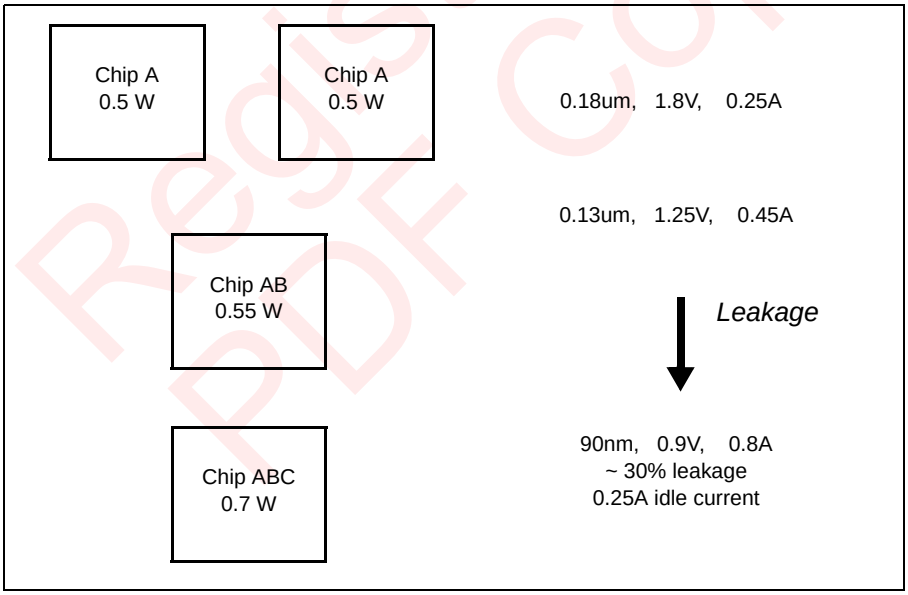


Figure 1-1. Process Migration

The current requirements at 90nm are quite high and the cost of such a current delivery mechanism is higher compared to previous generations, since the voltage is

also quite low. Further, the fact that almost 30% of the current is not from any activity is troublesome: a “tax” is paid in leakage current for each transistor that is integrated onto the die. In fact, current levels in deep submicron designs even when idle, are sufficient to heat the die significantly. Note that traditional low power techniques like clock gating do not help; they do not have any impact on leakage (other than arresting dynamic activity which may lower temperature eventually, which leads to lower leakage current).

Leakage has led to an interesting corollary in the regulation of electronic devices, as we will see in the next section.

1.2.4 Regulatory Aspects

It is not just the problem of the individual device leaking that is an issue. All over the world, in both developed and developing economies, there is a massive proliferation of consumer electronic devices, electronic control/interface functions in household appliances and automotive components. The proliferation of electronics in the household is compounded by the fact that they are increasingly affordable. Millions of households have now acquired electronic devices and therein lies a major problem: the problem of supplying enough energy to support this growth. Regulators all over the world have taken a hard look at the energy wasted by idle electronics and then it has further led to regulations on even active power. In the U.S., Energy Star [32] regulations imposed around the time of the first edition of this book require less than 1W standby power for most consumer devices. Further there are regulations on active power based on the class and size of device as well. Europe, Japan, China and India are likely to follow suit.

1.2.4.1 Idling Efficiently

There is some amount of idle power consumed by household devices such as TVs, DVD players, set top boxes, gaming devices etc., even when the device is not used.Most devices have a clock and a remote controlled function, that is running and waiting for user input even when the device is functionally “off.” For example, consider a plasma TV. While the screen may have been powered off, there are sufficient electronics in the TV that remember channel history, screen settings etc., and these are waiting for the next command input from the remote control. It may not sound like much to do, but this may take as much as 5–8W on some models!

Another example is that of a home network router. Network traffic is by nature, very bursty. This is coupled with long periods of idleness. Most models of routers consume

around 10-50mA of current, i.e., about 1–6W of power (at 110V nominal), just being idle.

In both the above examples, the fact that leads to wasted power is that even when idle, we require a certain “state of readiness” from the device and this state is not efficiently implemented in most of our current devices.

Let us assume that cumulatively, all the devices in a household consume about 10W in idle state. (Reality is estimated closer to 60W in some studies). If that is scaled across just 25% of the 70 Million households in the United States, that is a whopping 175 Mega Watts, leaking all the time. Just to contrast, for scale, power plants range from 50-1000MW in installed capacity: the idle power is equivalent to a medium sized power plant running all the time. In a single day, we waste about 4.2GWhr energy units, just in the U.S. and just on idle devices. In fact, U.S. Department of Energy estimates indicate that 40% of a household’s electronics’ energy consumption is from idle power. It gets even scarier when we compute this usage across the world.

For instance, a typical plasma TV consumes about 166W in active mode, but only 8W in idle mode. Even though the idle power in the example is only 5% of the total, in the span of a day, the energy consumed (and hence billed electricity) while idling is roughly 55% of the active usage energy of the device and about 35% of the total as shown in Table 1-1 below. In some models, the idle power is as high as 30W, causing an even greater waste of energy.

This trend has caused many governmental organizations across the world to impose stringent regulations on device power consumption, especially for idle mode. This is bound to have a significant effect on design styles and lead to new architectures and micro-architectures altogether.

Table 1-1. Active and Idle Power in a Plasma TV

Device	Plasma TV
Active	166W
Average usage	2 hours
Active energy usage/day	0.33kWh
Idle power	8W
Idle energy/day	0.18Wh

1.2.4.2 Active Power Regulations

As we mentioned before, governments around the world are faced with the problem of supplying energy to their citizens. New power plants take enormous effort and time to bring up; additionally there are environmental concerns that point to an emphasis on energy efficiency as opposed to a mere increase in generation capacity.

Active power regulations have hence kicked in even in the consumer market. To a certain extent, these are not new. Enterprise, industrial and commercial customers have been subject to active power regulations in local markets for many years now. In some cases, this has been applied in the form of increased tariffs, required wiring class specifications, additional safety measures etc. Hence, end customers apply pressure in these markets on their electronics' vendors.

At the time of this book's first edition, the new regulatory aspect however, is a broad attempt by various regulatory bodies to restrict the active power of consumer electronics devices. In some cases this extends all the way from small appliances like cordless phones to large television sets. Combined with the fact that the consumer market is the dominant driver of the electronics devices these days, this is bound to cause major shifts in how most of the semiconductors in electronic systems are designed.

1.3 Emergence of Voltage Control

In previous generations of ICs, clock gating was used to primarily control the amount of capacitance switching (toggling), hence reducing the dynamic power. Non-critical paths were implemented with higher threshold voltages, thereby reducing leakage. The power savings offered by these techniques are no longer considered competitive in the market or sufficient to deal with the complexities of current process generations. Hence, a shift towards the more aggressive design styles of voltage based control is seen across a wide swath of designs and applications.

1.3.1 CMOS and Voltage

Why is voltage control of CMOS gaining prominence? The answer lies in device physics. CMOS is a voltage controlled current source technology. Current determines the performance delivered as well as power consumed by a device. Both dynamic and leakage power of CMOS ICs are dominantly dependent on voltage.

As shown in Figure 1-2, dynamic power in CMOS is directly proportional to the square of the Voltage applied. A mere 10% drop in operating voltage can result in

20% lower dynamic power. Further, leakage power is also dominated by voltage in that leakage current Ileak is exponentially proportional to the applied supply voltage and inversely related to the threshold voltage of the transistor (equations not shown here). Hence even a 10% drop in voltage yields substantial reduction in leakage current.

$$E = \int_0^t (V_{DD} I_{leak} + C V_{DD}^2 f_c) dt$$

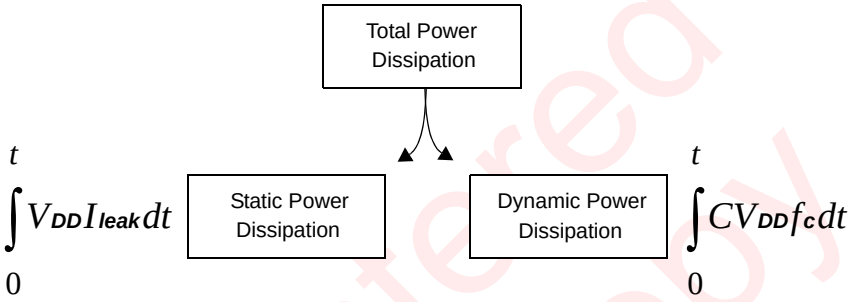


Figure 1-2. CMOS Dynamic/Leakage Power Equations.

Voltage control, however, is easier said than done. In real life, architectures that control supply voltage are quite complex. A MOS transistor is a 4 terminal device; hence there are multiple ways to control voltage. In the next subsection, we look at the basics of how voltage control is done, though this covered more extensively in Chapter 2.

1.3.2 Multi-Voltage Design in Practice

First, one must recognize the fact that once we resort to voltage control techniques, voltage is no longer a constant in space or time. The voltage applied to different transistors may be different even on the same die. The voltage applied to the same circuit may vary in time. Both of these effects can be termed as *Multi-Voltage* design.

Let us begin by looking at idle power. Fortunately, idle power is systemically easy to exploit for reduction. We never use all the systems or all the functions of a system all the time. The opportunity to reduce wasted energy is tremendous. Hence, we *must* design electronics to “idle efficiently.” *They must not consume energy when they are idling.*

Idle mode efficiency can be envisaged at the design level to be an “Off by Design” architecture. Such a design must ideally satisfy two fundamental properties:

- Any part of the system must be powered up only when needed.
- Any part of the system that is not in use must be turned off

Compliance with these two rules in the architecture implies that there is a “power manager” built into every IC and system that we build. Such a power manager would have to be both hardware and software savvy, as most modern systems and ICs are very heavily coupled with software for their active/idle profile.

The reader may point out a subtle corollary to the “Off by Design” concept. A system does not need to perform at the same level all the time. For example, one may buy a High Definition television, but may be watching a low resolution program or photo images on screen. Or a smartphone may be used as just a phone most of the time, a phone, as opposed to its use as a camera or a video player. So, while there may be some components that don't have enough opportunities to be turned off, there may be opportunities to run them at significantly lower power. A technique like this can be used to reduce the active power consumed by the device. In these lower performance modes, there is an opportunity to lower the supply voltage and hence the active power is reduced. Hence, a good corollary to “Off by Design” is as follows:

- A circuit must always be operated at the lowest possible voltage at which its performance goals are met.

In common practice, some of the techniques used are as follows, as illustrated in Figure 1-3: These are also described in detail in Chapter 2 from the ground up.

- Multi-Vdd: partitioning the chip into multiple supply areas and connecting each to its own power supply. Thus non-critical areas are operated at lower supply voltages, saving power.
- Power Gating: using on-chip power switches to locally “cut off” power supply to functional blocks and hence reducing leakage/idle current.
- Back Bias: applying reverse bias to the bulk connection to increase threshold voltage and hence reduce leakage.
- Low Vdd Standby: reducing voltage to a low level when idle, but retaining state. This is a leakage control technique.
- Multi-rail retention: using a backup power supply to hold state in low leakage latches, while main power supply is turned off.
- State Retention with Power Gating: retaining power only to certain state elements while powering down the rest of the logic.

- DVFS: dynamically varying the voltage/frequency to achieve optimal power/performance. Primary benefit is for dynamic power, while leakage is also reduced.

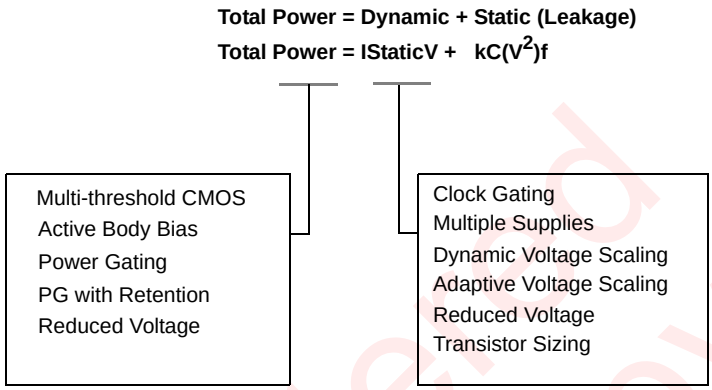


Figure 1-3. CMOS Power Mitigation

Each of the above techniques has its own application space and utility. Each also comes with its set of challenges for architecture selection, design implementation and verification. In general, what is observed in common practice today for architecture selection is as follows:

1.3.2.1 Dynamic Power Reduction

Dynamic Power reduction aims to control one of the following: a reduction in the amount of capacitance toggling, a reduction in the frequency of operation and/or a reduction in the supply voltage applied.

As such many designs see minimization of area (viz. capacitance) along high activity paths as well as reducing the overall capacitance to a minimum. One such effective method practiced widely is clock gating, which effectively reduces the capacitance of high activity networks. Commercial tools and solutions have delivered this feature in an automated way all the way from early synthesis to post layout optimization. Such designs do not suffer from the immense increase in verification as seen by voltage controlled designs.

Frequency reduction is architecturally driven and is controlled through hardware and software. Frequently, this is accompanied by dynamic voltage reduction as well, since we need only a lower supply voltage to meet the timing requirements of a slower

cycle time. In this book, we focus more on this latter method, since it represents the emerging challenge to traditional design and verification.

There are other methods to reduce voltage and frequency. If all parts of a chip do not need to be at high frequency, we can use a lower supply voltage for that part of the chip. Such a partition is typically called a multi-VDD architecture and is used in lieu of dynamic voltage scaling.

1.3.2.2 LEAKAGE POWER REDUCTION

Leakage current relies mostly on control of threshold voltage and/or supply voltage (since we really cannot control temperature variations). In the simplest of techniques, a static assignment of transistors (or library cells) to either a high Vt (slow speed-low leakage) version, or a low Vt (high speed-high leakage version). Typically, only about 5-15% of the timing paths in a large chip are critical. Hence, this technique produces quite a bit of leakage power savings. Commercial tools are also quite adept at automatic optimization of this technique, provided the libraries are available.

In more aggressive techniques, system or block level idleness can be exploited to further dramatically reduce leakage power and hence idle energy. One such technique is reverse-bias. As we describe further in Chapter 2, this is a technique where the Vt of transistors is increased by application of a voltage. This reduces leakage, although no operations can be performed (or performed only at a much reduced speed).

Another way to exploit system idleness is to reduce the supply voltage to a point where the state is not corrupted. This technique known as low Vdd Standby, is quite effective at reducing leakage: just as with reverse bias, at speed operation is not possible.

In both reverse bias and low Vdd Standby, there is no loss of state, unless something goes wrong. They are therefore quite useful for blocks with large amounts of memory. However, for many logic blocks, there is no need to hold state or perhaps there is a only combinatorial logic internally. This provides the opportunity to cut power off altogether to this block, a technique known as power gating which is again covered in further chapters.

Selection of the optimal architecture is alas, not as simple as this in real life. Every chip faces multiple constraints at the system level in terms of its average draw, battery life, bill of materials, latency and throughput in different modes. The optimization of such architecture is well beyond the scope of this book. Often, design teams do not spend time optimizing the architecture. It is hard enough to make a given architecture

work! This is an area where verification helps and we confine ourselves to this problem.

1.3.3 Control System View of Multi-Voltage

Irrespective of the power reduction target and mechanism, one of the most significant endeavors of this book is to encourage thinking that a power management scheme is a control system: a system that attempts to regulate the voltage of devices with power and energy as the end functions, and activity, thermal conditions and resource availability as inputs. It is also a control system that has both hardware and software inputs and outputs. The control system view of the power management scheme greatly helps to understand the loop between voltage regulators (voltage source), power management unit (PMU), and functional blocks (controllees) as shown below in Figure 1-4.

Perhaps the most significant aspect of the information being exchanged in this control system is that these are not always digital signals generated by a hardware finite state machine. Some of these are asynchronously generated analog/mixed signal entities and some of them are digital signals controlled by State Machines or software. Especially so, voltage is not a “logic signal”: it does not have the same synchronous behavior as a normal logic signal associated with CMOS finite state machines. *In fact, voltage cannot be clocked.* This is a tricky aspect that has made verification very difficult. In subsequent chapters of this book, we look in further detail at this control loop and the challenges of making such a system work correctly and efficiently.

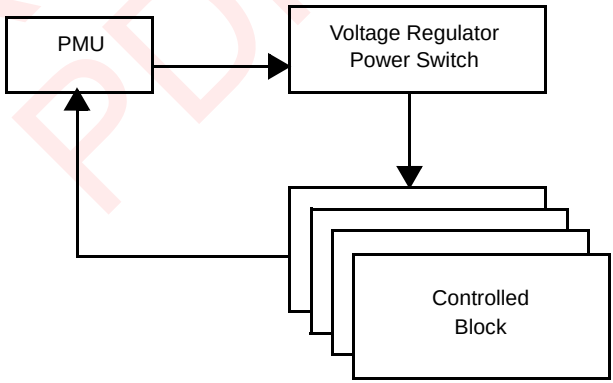


Figure 1-4. Control Loop Abstraction

In the context of a modern SoC, the picture of a power managed anatomy looks more like the picture in Figure 1-5, which implements the control loop: a power management unit (PMU) that controls/observes various functional blocks and issues out control signals to the various units, including the voltage regulators, forms the hardware component. A CPU or micro-controller that runs the software often forms an important part of both the control and observation of power management. Lastly, the mixed signal components such as voltage regulators, power switches etc., whether on die or off-chip are the third dimension to this anatomy. Chapter 5 discusses this structure and components in greater detail. Real life systems get very complex and often have hierarchical sub systems: a power managed system being a component of a larger power management system as a client. For example: an MP3 player that is connected to a USB port of a laptop.

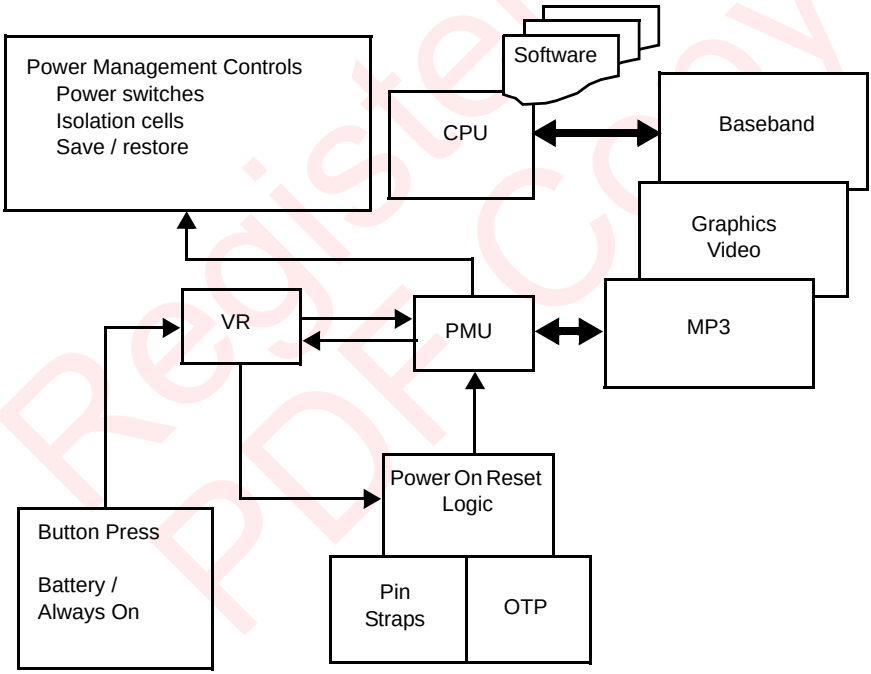


Figure 1-5. Control Loop Detail

1.4 VERIFICATION COMPONENTS

The preceding discussion brings us to the topic of the current book, *Verification of the Voltage Controlled Design*. Many verification engineers are astounded by the complexity of verifying power management schemes. The size of the power management unit may be quite small, but it does get under the skin of almost everything the chip does. Hence, power management schemes need to be integrated into essential functional verification. At the least, the device must be proven to work in all of its modes and deliver the power savings desires. Further, it must be verified that the device can transition from one mode of operation to another. That brings up the notion of coverage to mind with regards to the modes. Similarly assertions and other testbench infrastructure needs to be revised for the effects of power management.

1.4.1 HISTORICAL PERSPECTIVE

Traditionally, multi-voltage design, as we will refer to any design invoking voltage control as its power management technique, has been accomplished rather painfully due to the following factors.

- There is no unified mechanism to express architecture, especially the various constraints that lead to the choice of an architecture and the implications thereon.
- Hardware description languages are painfully ignorant of voltage, which is a rather fundamental entity to this process.

Unlike an optimization based on user driven constraints in the implementation process, verification needs to account for all the system level conditions and environments that the chip could be subject to. Power management schemes, while software intensive, cannot be debugged on emulators and FPGA schemes, severely handicapping the process. Further, logic simulators have been severely handicapped by the assumption that all parts of the chip are always on. For example, the model for most tool flows has been that the voltage is consistently applied across the chip at all times. There was no notion of blocks operating at different voltages or going off. In fact, logic simulation itself begins in the “all-on” state, assuming a constant, unspecified voltage. As designers have tried to overcome this limitation in logic simulators, there have been four generations of power management verification solutions:

1. Gate level models: these models used the notion of Vdd nets as 1/0 levels to indicate on/off and force cell outputs to “X” in shutdown. They were limited by the fact that voltages don't turn on/off abruptly and many events such as power on

reset are based on actual voltage levels. These were also limited by their inability to handle large designs and long vectors. Given their inability to factor in voltages as real, continuous entities, they did not work for common design styles such as low Vdd standby.

2. RTL level force: these models forced “X” at the boundary of power domains in shutdown. These overcame the capacity limitations of gate level models, but came with severe restrictions on partition structures, the inability to model voltage transitions, certain design styles etc. Some of the restrictions came from the languages such as Verilog/VHDL themselves and were hard to overcome.
3. Intrinsic simulator corruption: this is a modification of (2) in which the simulator supports a common power format to enforce shutdown. This suffers from the same drawbacks as (2) and is limited in its ability to support many design styles in use today.
4. Voltage aware simulators: this is a technique used to truly model the behavior of the circuit, considering the dynamic value of the voltage applied/generated at any point in time. These simulators are quite capable of handling almost any power management effect at RTL/netlist level.

1.4.2 Voltage Aware Boolean Analysis

As designers debugged with simulators of types 1–3 above, many bugs that are control oriented in nature, but electrically manifested in the device were missed, only to be found on silicon. The assumption (or model) that circuits instantaneously shutdown and wakeup is a big pitfall of non-voltage-aware simulation. The other limitation is that these models work only for shutdown related effects. Examples of such missed bugs are as follows:

- Incorrect application of voltages/unplanned voltage states
- Voltage scheduling errors
- Power on reset sequences
- Voltage monitoring and handshake logic errors
- Logic conversion errors
- Memory corruption/voltage race errors

In Chapter 3, “Power Management Bugs”, we cover examples of such bugs in detail and how they are found/debugged. An important point to digest will be that power management bugs are extremely hard to debug. They may not even manifest as a “1/0” logic error (waveform visible, to put it another way); all one might notice is excess current consumption in a certain mode or less than desired battery life. A

traditional tester is not of much direct use because of the dependence on system specifics such as pin straps and voltage regulators and also due to the dependence on being able to execute software sequences as if “in-system.” For example, it is quite difficult to reproduce an overheat situation in a laboratory or tester environment. Innovative, realistic system level debug techniques need to be used to find these problems, often running software.

Adding to this complexity is the extent of integration itself. About 10 years ago, power management bugs could be debugged by probing board level signals between various ICs. In the SoC era, entire systems and subsystems are integrated into nanometer geometries, often involving multiple layers of metal connectivity, all in one package. This aspect has made debug tremendously more complex when chips fail or a system level failure is seen. This implies delays as well as cost to the design process, resulting in a schedule push out. Hence, the arrival of voltage aware simulation is timely: Complex electrical bugs can be found at the RTL level itself, resulting in successful silicon and speedy firmware debug.

An example of Voltage Aware Simulation can be illustrated as follows:

```
// original expression
A = B && C;
// reality as below
If (Vdd of A is Not ON) A = X; else
if (Vdd of A is ON AND Vdd of B/C are sufficient) A = B && C;
else A = X;  ....
// Is this sufficient??
// what defines sufficient??
```

In reality, logic values are *dynamically* dependent on the exact relationships between the voltages influencing the drivers of signals A, B and C. This goes well beyond just the Vdd connections or on/off behavior. To re-emphasize, all of historic logic simulation makes the assumption that all the logic levels are the same, i.e., they are driven by the same voltage all the time and there is no turning blocks on or off. This simply does not work for power managed designs. We need logic simulation to reflect the real behavior of silicon in order to be able to verify and debug designs.

Obviously, changing the underlying notion of boolean analysis is bound to have a huge impact on the rest of simulation and verification. Standards in this space as to simulation semantics are only emerging and still need work to be comprehensive enough to accommodate all design situations. *Power Management Verification User Guide* [3] contains a more detailed discussion on how voltage aware booleans can be used in a real life design cycle.

Adding to this quandary is the fact that the problem space is extremely complicated as well: new coverage metrics, testbench building techniques, assertion categories, etc., need to be added to the verification process, as well as an effective debug mechanism. For example, consider the design and the accompanying power management state diagram in Figure 1-6 below.

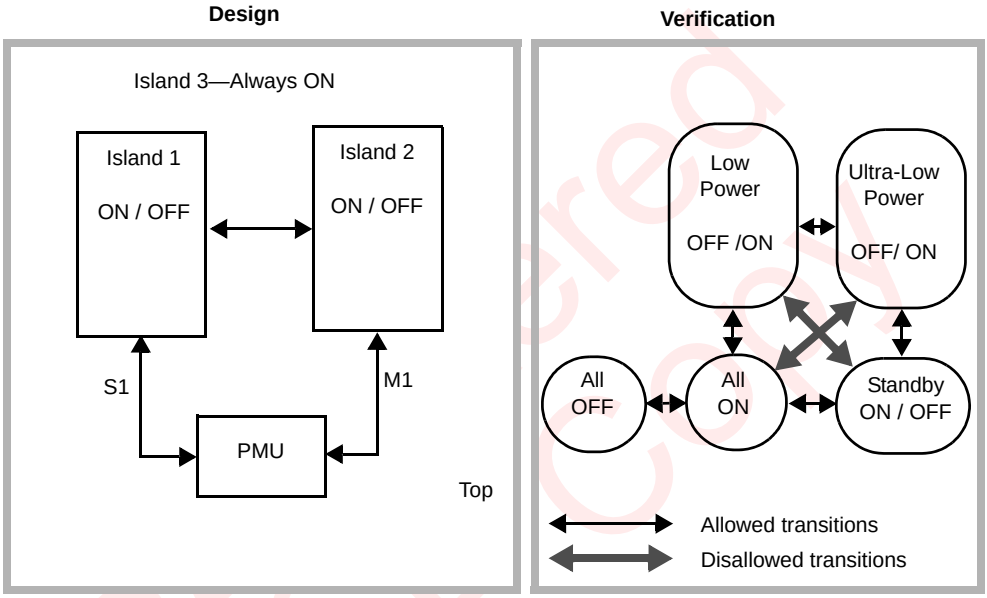


Figure 1-6. Low Power Design Verification

In the traditional world of verification, all effort was focused on the “all-on” state. In the power managed design, the following must be verified:

- The design is functional in each power state/mode
- In each state, the design can execute the transitions required
- No illegal transitions or states are executed
- The design responds to stimuli appropriately to go through the appropriate state/ transition sequences.

This is quite complex a problem; apart from voltage aware simulation, we need a smart formal/static technique to analyze power management schemes. Unfortunately, many methodologies in the industry have defined power management verification as

a set of mere structural rules. This is severely limiting and error prone. It is true that structural errors, such as missing/incorrect circuit elements and power connections, can be found statically and are important bugs. However, Chapter 3 discusses in detail quite a few bugs which cannot merely be detected by static analysis. In general, the effective verification of power management requires a smart hybrid approach: vectorless analysis combined with electrically accurate logic simulation to find bugs and attain first pass success on silicon. In Chapter 5 – Chapter 7, we cover static verification as well as dynamic verification.

1.5 Methodology Adoption and Implementation

1.5.1 Methodology Differences

Voltage aware analysis is essential for realistic simulation, but the task of verification goes well beyond that. Probably the foremost of problems in automated design flows for voltage controlled power management schemes is in the specification itself. As we see later in the book, power management schemes are elaborate state machines that have strict requirements on sequencing, asynchronous handshakes and on the temporal flow of control. The problem of missing voltage in HDL specifications of the design is compounded by the fact that there is no protocol specification mechanism in our current languages.

It is also our experience that most engineers are not cognizant of what could be wrong, because they are not being trained in the process and methodology of specification and verification of voltage controlled designs. Power management schemes are subject to many new types and profiles of design bugs that often do not seem to be functional errors at all. Rather many bugs cause electrical failures such as overheating or excess power consumption: a situation that is traditionally out of the scope of a “logic verification” process. However, a closer look always yields the fact that an error in control has happened. The architecture and microarchitecture often do not take the various phenomena of power management into account. Quite frequently, an error in the exercise of control results in problems that are electrical or power oriented in nature.

For example, consider a bug in an SoC that inadvertently fails to shut off a large block when not in use, when that is indeed a requirement to attain reduced power. A (positive test) functional pattern would not be able to identify such an error. On the other hand, the device may display an excess of current consumption, manifested as peak (current draw, electrical integrity based failure), excess heat or deteriorated

battery life. These are errors against the system level functionality: no “1/0” error in the response of the device may be observed. A problem such as this cannot be specified in traditional methodology. It is unverifiable, except by conscious application of ad hoc methodology.

Another area where best practices are just evolving industry wide is the area of low power testbench creation. Most verification engineers adapt their logic simulation harness to accommodate the power management scheme. This largely works but for a few key factors, as described below. This topic is further discussed in Chapter 5.

- Key system level components, such as voltage regulators, etc., need to be modeled for their response. This is an important part of the power management control system.
- Power management was not quite amenable to random testing methods or even constrained random methods. This is because we did not have formal constraint mechanisms on power management state representations as of yet. Also, the mix of both hardware/software into power management makes this more difficult.
- Note that in a multi-voltage simulation, the testbench is not “watching” for power management failure conditions. apart from the simulation, we need smart “watchers” to not only help monitor error conditions, but to provide appropriate messaging for effective debug.
- Any assertions, monitors, checksums, etc., now need to account for blocks going to power down. This often involves considerable effort. In addition, *new* assertions need to be written for power management, involving voltage rails, power states, sequences etc. This activity is going to be a major migration effort in the near future for designs adopting power management. Quantitatively, even a small So of about 2M gates could potentially need about 100,000–200,000 checks in place for various conditions
- The mind-set for debug needs to be different. A register being corrupted to logic value X may be due to a power supply error or retention scheme error. There are new failure mechanisms that debug tools now need to account for.

In this book, therefore, we seek to achieve the dual objectives of education as well as execution in the area of verifying power managed designs. Given the mind-set of traditional design and verification processes, it is quite important to bring forth a well documented body of knowledge from our collective experience. Beyond the educational aspect, various automation capabilities are required for verification productivity and comprehensiveness, which we will discuss as well.

One of the biggest endeavors of this book is to advocate a SystemVerilog based strategy for power management verification—a task that has not been attempted before. This has especially been hard in the absence of a specification language such

as SystemVerilog for either the design or the verification environment. Constrained random testing has been extremely successful in achieving almost 100% of the required stimulus patterns in the verification of non-power managed ICs. Coverage metrics have accompanied this approach to quantify progress. Given the asynchronous and often conflicting nature of how power management events occur in real life, random approaches should be quite successful in bringing out the hard to test corner cases.

The methodology presented in this book is quite extensive. It contains several different—but interrelated—facets and elements. The increase in productivity that can be obtained by this methodology comes from its breath and depth. Perhaps more fundamental a task is for the verification engineer to be aware of the failure mechanisms in power management and accordingly architect a test plan and a testing strategy. We look at many real-life designs and system examples to illustrate this point, and draw from these specifics a generic methodology that is scalable and reusable.

Here, an aspect of the original VMM, code re-use becomes quite relevant. As we extend VMM for Low Power and Power Management, the objective is that we are intercepting a nascent market with the right methodology to create reusable testbenches and verification utilities. Reusing code avoids having to duplicate its functionality. Reuse is not limited to re-using code across projects. First-order re-use occurs when the same verification environment is reused across multiple test cases on the same project. By re-using code as much as possible, a feature can be verified using just a few lines of additional code. Ultimately, test cases should become simple reconfigurations of highly reusable verification components forming a design-specific verification environment or platform.

1.5.2 Methodology Adoption

The purpose of this book is not to extol the virtues of SystemVerilog or any particular commercial solution and the verification methodology it can support. Rather, like its predecessor the *Verification Methodology Manual* (VMM) [1], this book is focused on providing clear guidelines to help the reader make the most effective use of SystemVerilog and implement a productive verification methodology. The book does not claim that its methodology is the only way to use SystemVerilog for power management verification. It presents what the authors believe to be the best practices in any context.

It is not necessary to adopt all the elements of the methodology presented in the following chapters. However, it is recommended strongly that the first step to

adoption be an education/information approach followed by the formulation of a verification strategy across the project(s) and team(s). Obviously, maximum productivity is achieved when all of the synergies between the elements of the methodology are realized. But real projects, with real people and schedules, may not be able to afford the ramp-up time necessary for a wholesale adoption. Individual elements of the methodology can still be adopted and provide incremental benefits to a project.

Some of the major elements in the methodology that are required by almost every design are as follows:

- The use of voltage aware booleans for logic behavior and voltage aware models for mixed signal components
- The vigorous testing of reset and retention, especially power on reset conditions
- The use of power aware assertions and power management coverage to drive the verification process
- The use of random stimulus to exercise the asynchronous and concurrent nature of power management events
- The adoption of a firmware validation approach to verification to exercise the true in-system functionality of the device

1.5.2.1 Differences with VMM

There have been both extensions as well as additions to VMM so that we can verify low power designs. These can be found in Chapter 7, Appendix A and Appendix B. The approach is incremental to the original Class Library specification. However, the user must be aware that the underlying boolean analysis has changed and that some of the changes presented are to account for these.

Early in the process of drafting the book, the authors felt that the book should move away from containing a lot of source code examples inline. Rather, we have chosen to make these available for download. This makes their reuse directly possible.

Unlike the *Verification Methodology Manual* [1], we have actually put in quite a few design rules and recommendations as well and compliance to them must be deemed the task of verification. The reason, partly, is that many new design concepts are being presented to the reader in the area of design.

1.5.3 RULES AND GUIDELINES

Not all guidelines are created equal, and the guidelines in this book are classified according to their importance. More important, Rules should be adopted first, then eventually supported by a greater set of less important recommendations and suggestions. However, it is important to recognize the synergies that exist among the guidelines presented in this book. Even if they are of lesser importance, adopting more of the guidelines will generally result in greater overall efficiency in the verification process.

As we mentioned earlier we have actually put in quite a few design rules and recommendations as well and compliance to them must be deemed the task of verification.

Rules — A rule is a guideline that must be followed to implement the methodology. Not following a rule will jeopardize the productivity gains that are offered by other aspects of the methodology. *SystemVerilog Verification Methodology Manual*-compatibility (VMM-compatibility) requires adherence to all rules. VMM-compliance requires that all rules be followed. Further, with respect to power management, some of these rules have been formulated to be *preventive* in nature with respect to either the occurrence of bugs or the prospect of their escaping the verification process. Hence a violation of some of the rules prescribed may in some cases be a recipe for functional failure.

Recommendations — A recommendation is a guideline that should be followed. In many cases, the detail of the guideline is not important, such as a naming convention, and can be customized. Adherence to all recommendations within a verification team or business unit is strongly recommended to ensure a consistent and portable implementation of the methodology.

Suggestions — Suggestions are recommendations that will make the life of a verification team easier. Like recommendations, the detailed implementation of a suggestion may not be important and may be customizable.

The guidelines in this book focus on the methodology, not the tools that support SystemVerilog or other aspects of this methodology. Additional guidelines may be required to optimize the methodology with a particular toolset.

1.6 Structure of The Book

Chapter 1 through Chapter 4 constitute an in-depth introduction to power management, focusing on design aspects and more importantly, what the likely bugs are. Chapter 2 is an in-depth look into the basic building blocks of multi-voltage. Chapter 3 follows up with a look at typical bug profiles and ventures just a bit into how they can be detected. Chapter 4 is an in depth look at Retention, which presents a new logic design and verification challenge in itself.

Chapter 5 through Chapter 7 focus on verification. Chapter 5 deals with the creation of a testbench or migrating one from a non-low power version. Chapter 6 and Chapter 7 focus on static and dynamic verification, assertions, coverage etc.

The Appendices contain the Base Class specification as well as a summary of the rules and guidelines sprinkled through out the book. They also contain a list of static checks as a reference.

The book is intended to be read in serial order of chapters, though readers very familiar with low power may skip ahead to Chapter 5. If retention is not adopted on the design being verified, Chapter 4 may be skipped.

CHAPTER 2 MULTI-VOLTAGE BASICS

ABSTRACT

Many styles of voltage control for CMOS are possible. In this chapter, we look into the basics of this voltage control. The details will be in terms of the building blocks (design elements) and the design styles that are commonly used. Wherever possible, we adhere to the terminology adopted by the emerging IEEE (P)1801 standard.

2.1 DESIGN ELEMENTS

2.1.1 RAIL/POWER NET

A virtual or physical network that is connected to the output of a voltage source, possibly controlled through some switching or value setting mechanism. Typically, this network does not have a logic value associated with it. However, it is common to find Logic 1 and Logic 0 assigned to their representations in HDLs, especially for “wire” constructs in Verilog HDL. A rail or a power net is a controlling entity for the transistors, but is often a controlled entity of the power management scheme.

2.1.2 VOLTAGE REGULATOR

This is a handily available component in the industry, both as a discrete component or a mixed-signal intellectual property block. However, its foray into the digital world is

new facet brought about by power management. A voltage regulator usually has a source power supply (connected to AC mains, battery, or another voltage regulator), an output “regulated voltage,” digital control bits to set the voltage or turn the regulator on/off, and some status/handshake signals to indicate stability of output. Obviously, many more sophisticated implementations are possible.

From the power management point of view, the voltage regulator squarely injects itself into the midst of the control scheme. The result is that a change in power management state must interact with the voltage regulation scheme in some fashion and then wait for a response.

2.1.3 Primary or Driving Rails

Primary rails actively drive logic values on signals and are responsible for supplying or charging/discharging current at the cell’s output. In current CMOS standard cell based implementations, the Vdd/Vss connections of a CMOS cell are often the functional rails. As illustrated in Figure 2-1, the Vdd rail supplies the current to charge the output node to Logic 1 when needed and the drive to maintain that level. Similarly, the Vss rail of Figure 2-1 discharges the output node as needed and maintains a Logic 0.

2.1.4 Secondary Rails

Secondary rails control the behavior of the cell, but do not actively supply current for charge/discharge of the cell’s output. In Figure 2-1, the output of the charge pump, Vslp is connected to the gate of a transistor labeled Footer. Variations of Vslp influence the ability to functional rails to charge/discharge, but Vslp itself is not part of the charge/discharge path. Note that the Gate connection (node G in Figure 2-1) of the CMOS cell is also a secondary rail in that it influences the cell behavior but does not connect with the charge/discharge path. A secondary rail at one cell can be primary to another and vice versa. The interaction of the rail with CMOS logic elements determines its nature within the context of the logic element.

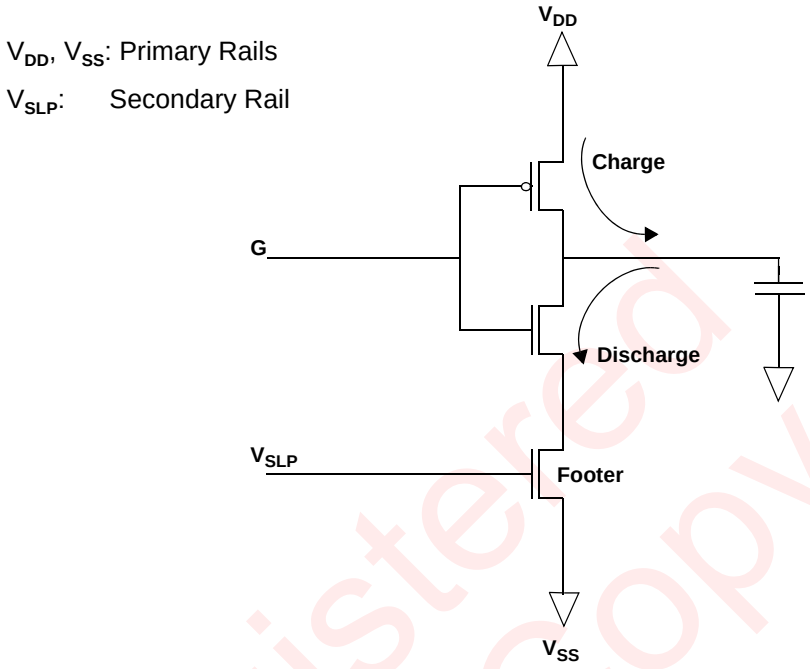


Figure 2-1. Rails

2.1.5 VDD AND VSS

Vdd and Vss are the *power* and *ground* rails of a CMOS cell as in the existing non-multi-voltage cells. At the risk of sounding repetitious, it is also important to note that a Logic 1 is charged to the level of Vdd and a Logic 0 is discharged to the level of the Vss. Less obvious, but equally important is the fact that the effects of any secondary rails applied to the CMOS element are only in relation to the values of Vdd and Vss. This property is critical in every aspect of power management. Also note that in most cases, Vdd/Vss are the only driving rails of the CMOS cells used in digital logic.

2.1.6 HEADER AND FOOTER CELLS

High threshold voltage (Vt) transistors are used to “switch” or “gate” the connection between the primary rails and the PMOS/NMOS elements that implement logic, as shown in Figure 2-2. (This leads to the term power gating, which will be discussed later.) These High Vt transistors are called “Headers” when inserted between Vdd and

the PMOS elements; they are called “Footers” when they are in the path between Vss and NMOS elements. Except for some special cases, Headers are comprised of PMOS transistors, and Footers are comprised of NMOS transistors.

In theory, a single ideal Header or Footer cell is sufficient for any amount of logic that is fed off of the primary rails. However, in practice it is common to use multiple parallel transistors, which are often staged in time by a control mechanism. (Read: target of verification!). Many practical issues surround the use of Header/Footer cells, mostly tied to layout topology. The *Low Power Methodology Manual* [2] offers more information on this aspect. One important property to note at this time however is that based on the voltage applied to the gate of the Header/Footer (relative to the Vdd/Vss levels), the logic elements are either connected to Vdd/Vss or cut off from them. In the latter state, it amounts to shutting down these elements. Later in this chapter, in the Power Gating section, we will discuss the verification oriented aspects of using headers and footers.

Furthermore, the gate of the Header/Footer may be connected either to a rail or a logic signal. This distinction is really for the user’s design flows and a matter of how the on/off control to these gates is generated. The response of the Header/Footer is always to the relative voltage levels and must be considered mixed signal behavior. Note that Header/Footer cells are themselves simple voltage regulators with just an On/Off function.

2.1.7 VIRTUAL VDD/VSS

When a Header/Footer network is used, the “output” of these cells is the one that physically feeds the logic elements. As long as the gate is fully on, the output tracks the Vdd/Vss levels; otherwise, it is floating. This node is called the Virtual Vdd/Vss, as indicated by VV_{DD} and VV_{SS} in Figure 2-2. Virtual rails are also called switched rails (or switched power and ground).

In practice, most designs use either headers or footers but not both. Hence, it is common to find either Virtual Vdd or Virtual Vss, not both.

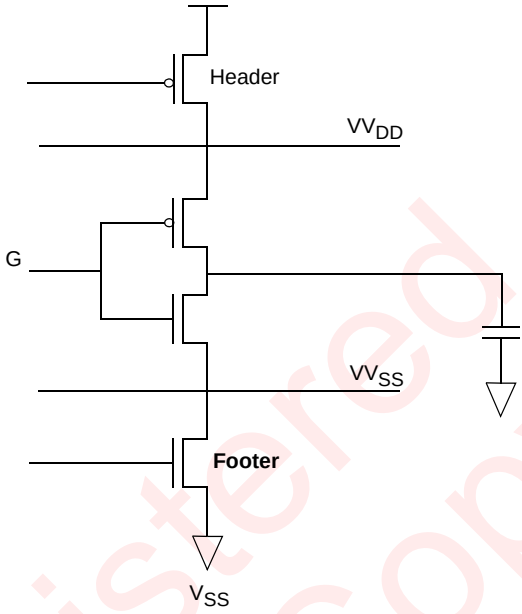


Figure 2-2. Header and Footer

2.1.8 Retention Cells

When the power supply to volatile memory or sequential logic elements (such as flip-flops and latches, register files, etc.) is turned off, the state stored in them is lost. When power is restored, the state of all the sequential elements is reset. Any execution of software or continuation of execution from which items were left off is not possible. This may be architecturally undesirable in many circumstances: retention is a phenomenon that is used to restore the context of sequential elements. Frequently, retention is accomplished by providing circuitry at the leaf level of implementation, such as the register and latch elements. Conceptually, an additional latch is provided whose power is not turned off (and hence state is not lost). Often, the power to the latch (aptly called the “shadow element”), is supplied by a secondary rail. This rail remains powered on while the primary rail is turned off.

There are numerous ways to accomplish retention of state and subsequently, with many deep implications of retention all across IC design flows. The topic is enough to

warrant a chapter to itself. Chapter 4, “State Retention” takes an in depth look at retention.

A retention cell with two rails, one for the primary and the other for retention elements is shown in Figure 2-3 below.

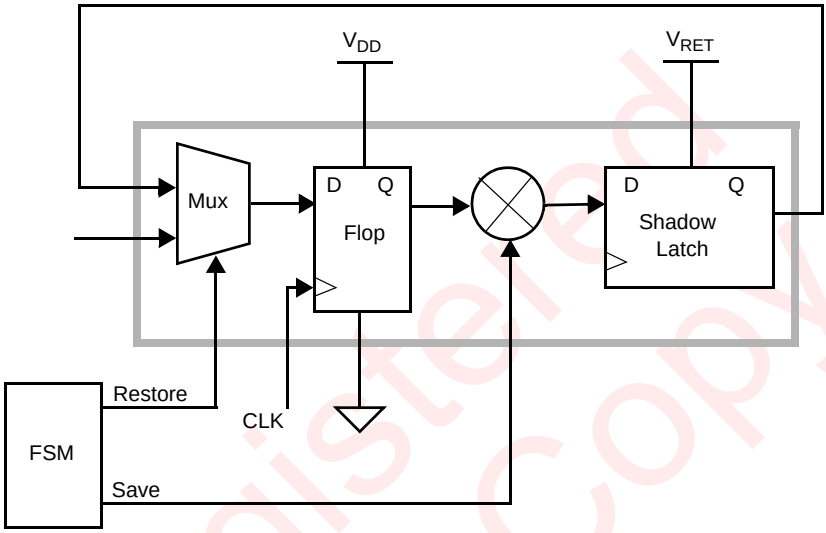


Figure 2-3. Retention Cell with Dual Power Rails

A retention element that uses the same Vdd for main and shadow elements is shown in Figure 2-4 below. In this scheme, only the main storage element is power gated. The shadow element remains powered up. Note that these are the main conceptual styles of retention. In practice many schemes for retention exist, based on the actual implementation scheme used.

Note that the retention rail itself charges and discharges nodes in a cell, but not the final outputs and hence is not considered a driving rail of a power domain.

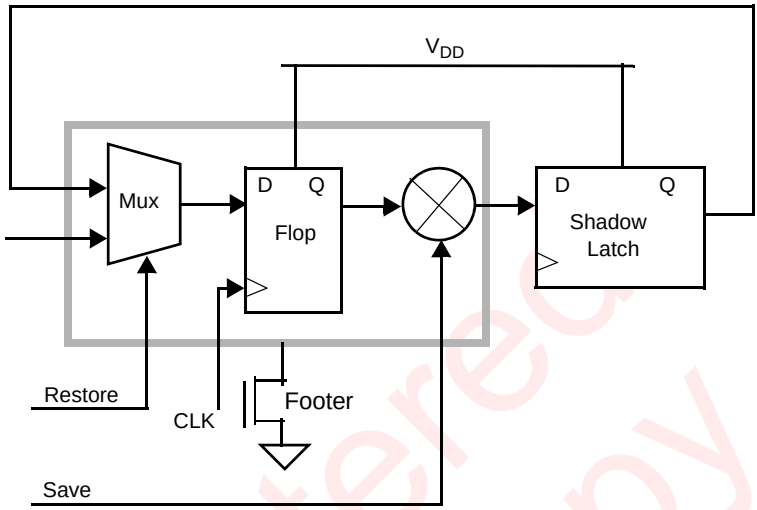


Figure 2-4. Single Rail Retention Cell with Power Gating

2.1.9 BULK/BODY TERMINAL

The substrate or well connection of a MOS transistor is its 4th terminal, usually referred to as bulk or body terminal. In many standard cell implementations, this is not a control terminal; it is merely connected to Vdd for PMOS and to Vss for NMOS internally. However, in power management, this terminal provides an important handle on leakage control. The phenomenon exploited is the fact that the threshold voltage of the transistor is a function of the potential difference between source and bulk terminal. The threshold voltage can be reduced, increasing the leakage as well as maximum frequency possible, but it can also be increased, thereby decreasing leakage and slowing the logic down. It is the latter phenomenon that is typically utilized.

At the time of this book's original draft, no EDA standards exist for modeling supply voltage or body terminal voltage (also called well voltage). Hence, it is very difficult to arrive at safe timing models for varying threshold voltage. As such, this technique is mostly used to realize a low leakage standby state.

2.1.10 ISLAND

A set of logic elements such as hierarchical modules, leaf cells or a group of HDL/ESL statements with common rail connections. These elements are electrically identically controlled. The typical sets of connections possible to an island are Vdd/Vss, Retention, Sleep, Header/Footer and Bulk connections. The notion of Island is that an electrically identical set of design elements form a “unit” of control in power management: this definition of an island is verification oriented. An island in physical terms may be spread across many logical hierarchy components and/or physical regions.

Author’s note: Many reviewers have pointed out the confusion between island and domain (below). They are regrettably used interchangeably, but especially on ICs that go beyond Vdd control, it is useful to make an electrical distinction.

2.1.11 WELL

A Well is a set of cells with a common body/bulk connection. Given that there are two body connections in CMOS, they need to be identical for both PMOS and NMOS connections. The term Well refers to cells by grouping them according to their bulk connections. This grouping has implications throughout the physical design flow. Strictly speaking, some bulk connections are substrate connections as opposed to being in a “well,” but it is useful to think of all the elements with a common bulk connection as belonging to a particular class of control.

2.1.12 DOMAIN

Domain is the drain of the driver.

Domain is the most abused term, often confused for islands; domain indicates which primary rails are driving the signal. *This is probably the single most important definition of all from an electrical point of view.* Note that the domain of a signal determines its condition of being in shutdown or when the signal is received in another domain, the need for adjusting its voltage levels to the receiver.

An increasing term commonly being used is power domain: this is more or less analogous to what we defined as an island. In fact, the word island is often used only when an island is power gated with headers and footers (thus creating an island!).

As most users will take note, multiple islands could have the same driving rails and be part of the same domain. For example, the difference may be which retention supply they use for a secondary rail. Similarly, multiple domains could have common body connections and be part of the same well, but still be different islands electrically. In that sense, the definition of island is the most granular, since any variation in rail based control is accounted for.

2.1.13 Always ON Regions

In physical implementation, buffering of signals is required when traversing long routing lengths. Even conceptually, it helps to think of a general placeholder domain for logic that is *not* subject to power management. This is referred to as the Always On domain. Often, this is also battery domain logic, which remains On even when the rest of the chip is turned off.

Always On domains have different implications for verification and implementation. In the former, they represent additional correctness checks and corner cases for verification such as a drained battery. In the latter, they represent useful areas which can ease constraints on the implementation.

For verification engineers, it is best to take the concept of Always On with a grain of salt. In reality, there may not exist an Always n domain. External power may be turned off and the battery may be completely drained. Many bugs lurk around this corner case, preventing a wake-up/system bringup.

2.1.14 Spatial Crossing (Crossover)

Spatial crossing is a signal that is sourced in one island (not a domain) and has (a) destination(s) in another. Note—this is a source destination pairing on an island basis. If an signal has fanouts in multiple electrically disparate islands, they (may) behave differently at each receiver or the receiver may react differently. Irrespective of logical and physical hierarchy, all the destinations in an island from the same source can be considered as the same crossover.

2.1.15 Temporal Variation

Temporal variation is the variation of a rail over time. The rails of an island, when varied over time, result in different electrical conditions or “states” of the island and

the chip. In the case of power gating, temporal variation in an island can also be achieved by varying the power gating control bit.

2.1.16 Multiple Voltage State or Power State

The Multiple Voltage State or the Power State is the set of all rail values present in the system at any given point of time; this is the instantaneous complete picture of spatial and temporal variations in the chip. The state of the system at any point is of course based on the state of all of its islands, which in turn for each island is a function of its rail values. As temporal variation in rails is accomplished by the control mechanism, the state of the islands varies. For example, we described earlier in this section how the island can be turned “off” by exercising control over the gate of these cells.

Generically, an island is either in shutdown state or turned on at a particular voltage level of its rails, especially the primary rails.

2.1.17 Protection Circuit

A Protection Circuit is a cell that is used on a spatial crossing, to electrically protect the receiving island or to maintain the signal value in the receiving island.

2.1.18 Isolation

Isolation is a technique to protect a receiving island that is active from a signal originating in an island that is turned off. The temporal variation of islands from On to Off and vice versa implies this isolation is a gating element, controlled in conjunction with the state of the source island. As innocuous as it sounds, isolation is a common source of functional errors, thereby implying that it will be a significant target of our verification efforts.

In terms of logic gates, AND/NAND/OR/NOR types are typical choices for isolation gates. Latches may also be used, but come with additional verification challenges, among other additional costs such as area/delay. Isolation gates come with a whole host of “correctness” requirements: from their placement spatially to the isolation control temporally.

An example is shown below in Figure 2-5, marked by the gate between domains V3 and V4 as Isolation.

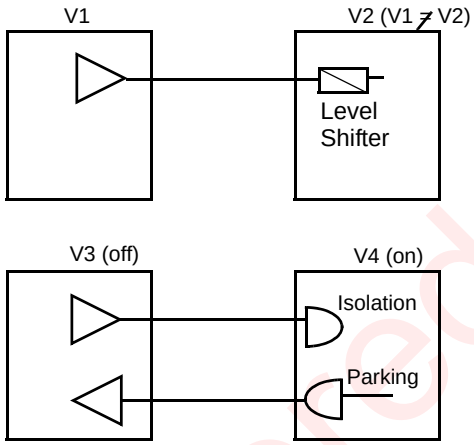


Figure 2-5. Isolation and Level Shifting

2.1.19 Input Isolation (Parking)

Input isolation is a technique to arrest input toggles in an island. This is especially useful if the island is being put to a shutdown state or other “standby” architectures that are described in the next section. Inputs can be isolated to Logic 0 or Logic 1. Input isolation, when used for islands in shutdown, requires the isolation device to be NOT in the shutdown island, rather in something that is ON and this may be done only to Logic 0, to avoid an inadvertent connection to ground through a pass transistor (known as “sneak path”). Input isolation is not always required. Often having CMOS gate first stages (as opposed to pass transistors) is sufficient. Input isolation is however highly desirable to lower power consumption in the case of a net that drives a lot of load capacitance in the shutdown domain, such as a clock/reset signal. Input isolation may be essential in the case of standby architectures.

An example is shown above in Figure 2-5, marked by the gate between domains V4 and V3 as Parking.

2.1.20 Level Shifting

Level Shifting is a technique to convert a signal driven by one set of primary rails (and hence referenced as Logic 0 and Logic 1 to them) to another set of primary rails

on a spatial crossing. Typically, level shifters are either high voltage to low voltage or vice versa, although “auto level shifters” are also common: they shift both high to low and low to high. An example is shown in Figure 2-5, marked by the gate between domains V1 and V2 as Level Shifter.

An important class of level shifter is the Enabled Level Shifter, which is the combination of an isolation device and level shifting.

The reader may wonder what really is behind “low to high” and “high to low.” The hidden implication here is that we are referring to the Vdd levels of the driver and receiver. This is a convenience that we can get away with: most ICs we implement today are common ground and have a common Vss level. Therefore, the level shifting of Logic 0 is not needed, except in some special situations.

Why do we need level shifters in the first place? Because Logic 1 is charged up to the level of the Vdd of the driver. This may be either inadequate to represent a Logic 1 at the receiver based in its Vdd level or even if it is, it may cause excess current consumption. (Remember the CMOS transfer curve?)

One of the other common questions in many engineers’ minds is why then, is high to low shifting needed? The driver’s Vdd, surely, is sufficient to represent Logic 1. The answer is that over driving the gate of the receiver can result in quite a bit of additional current consumption and in some extreme cases even cause a Logic 0 at the driver to be perceived as a Logic 1 on the receiving end. Sometimes, the transistors of the receiving domain may not be even rated for the higher voltage requiring a high to low conversion, or risking damage to the oxide layer in the receiving gates. As we migrate to lower process geometries, gate leakage is an increasing component. Hence, the cost of not converting high to low rises with smaller geometries.

2.1.21 Power State Table

A Power State Table is a listing of “voltage modes.” Each entry is a combination of voltages that could be applied to the chip. Note that unlike a state register, there is often no physical equivalent for a power state table; its entries are real numbers.

The Power State Table can be used to automate isolation/level shifting requirements. For example, if a crossover exists from an island that has an off state to another in on state (in any entry of the state table), it can be inferred that at isolation device is needed and must be activated in the corresponding state. Similarly, if a crossover is subject to a voltage difference between source and receiver, it may require a level shifter.

One of the most fundamental aspects of a Power State Table is the “All Off” state. Designs without the all off state listed risk missing states in the power up and down sequence as well and could also be missing isolation/level shifting inference.

2.1.22 STATE TRANSITIONS

Power state changes imply that there will be transitions in voltage applied to move between states. These transitions may occur because of either logical events or voltage events, or both. For example, the arrival of an interrupt (logical event) may cause the device to put a certain island into standby or shutdown states (voltage event). On the other hand when the voltage is sensed to be a certain level, it may trigger a reset or PLL event.

State transitions between power states cannot be arbitrary. Islands must follow a strict order of power up and power down to keep the device safe at all times and not violate the electrical integrity of devices placed on the chip. One of these constraints is that the voltage relationship factored at a level shifter may not be reversed. Another example is that multiple rails changing between states could cause electrical integrity failures. This is used as a metric of the “safety” of the Power State Table.

Rule 2.1 — *Safe Graph Rule:* Each State in the Power State Table must have a transition possible to at least one other state to which it differs in only one voltage level.

It must be possible to traverse the entire power management state scheme varying only one voltage rail at a time. Designs that violate this rule will need additional electrical validation in a (power) noise analysis tool.

2.1.23 STATE SEQUENCES

State Transitions are rarely ever done in isolation. Real systems react to commands or events (or lack thereof) and move from one “mode” to another. While each mode itself could represent a power state in itself, often, a mode or migrating to a mode involves several intermediate steps, with both logical and voltage events. Sequences combine both logical and voltage states/events and could be long, software/hardware driven protocols.

2.1.24 PMU (POWER MANAGEMENT UNIT, POWER CONTROLLER)

The Power Management Unit (PMU) is the functional block that makes the temporal variation of voltages occur. Its function is to sense the state of the system or IC and ensure that the device makes a transition to the appropriate system. Power Management Units don't just control voltages, they make sure that clocks, resets, retention controls etc. all act in conjunction. They monitor the various blocks and assert the appropriate controls. They are often a combination of hardware and software based control and observation.

2.2 DESIGN STYLES FOR MULTI-VOLTAGE

The previous section described the basic elements of multi-voltage control. Various design styles are possible by using the basic elements and controlling them appropriately with a power management unit. To reiterate what we discussed in Chapter 1, "Introduction", there are only a few basic architectures:

- Turn off blocks that are not in use or put them into a "standby" low power state
- Use blocks at the lowest possible (effective) voltage when needed

2.2.1 SHUTDOWN

This is a state in which the Vdd to a domain or an island is turned *Off*, not necessarily to zero volts. The voltage regulator typically resides off-chip, but it is increasingly common to achieve this with an on-die regulator or a power switch. (Power gating will be discussed later in this chapter.) Although in current usage, shutdown has become synonymous with power gating, it can be accomplished by direct control of the voltage source as well, not just power switches.

Shutdown works primarily as a leakage reduction technique. As we discussed before, a shutdown of a block implies that all state is lost and needs to be either reset or restored upon wakeup. Further, shutting down a domain also implies that certain amount of pre- and post shutdown control of clocks, isolation control signals etc., is required.

2.2.2 STANDBY

In general, Standby is a low power state, in which a quick wakeup is expected. State retention in the memory elements is essential. Typically, clocks are gated, but PLLs are not disabled. However, there could be multiple grades of Standby, turning off more circuitry as time passes. The overall Standby scheme is one that involves a process of gradually turning off (on) clocks, PLLs, and voltages. In this context, we focus on the use of voltage control. Various techniques are used for Standby.

2.2.2.1 CLOCK GATED STANDBY

Clock Gated Standby is a mode in which no Vdd control exercised. The clock network is gated, and that in itself saves a lot of dynamic power. The PLL may be On or Off, depending on the desired resume time. While there is no Vdd control, this mode serves as a preparation for it.

Clock gating can be combined with two voltage control techniques to reduce leakage. Low Vdd standby and Back Bias, which are described below.

2.2.2.2 BACK BIAS

Back Bias, also called Reverse Bias as shown in Figure 2-6, raises the threshold voltage of the island, which in turn reduces the leakage. State in registers is not lost. The effective frequency of operation is reduced; this technique is typically applied for standby mode. Blocks with large RAMs and register files are especially amenable to this technique. Note—a separate bulk terminal grid is needed in the layout.

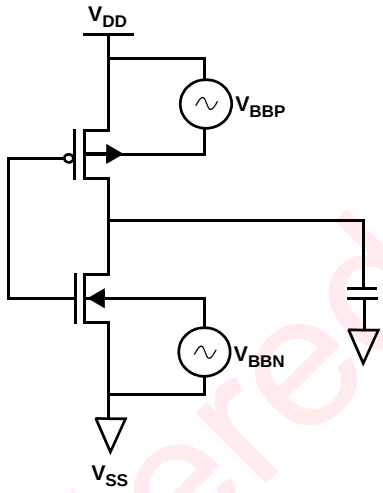


Figure 2-6. Back Bias

2.2.2.3 Low Vdd Standby

Low Vdd Standby is a state in which the Vdd is lowered to just enough value for the memory elements to not lose state. Inevitably, the clock must be gated down for this situation, because there (typically) is not sufficient drive to do write operations at this voltage level. Note—the outputs of a block in low Vdd Standby need level shifting, which they do not need in the case of back bias. However, there is no need to layout another grid to the bulk terminals.

2.2.2.4 Output Parking

Output Parking is typically used in Low Vdd Standby state. This prevents the need for a level shifter *at the destination* in a common ground system; output parking is at Logic 0. It is accomplished by inserting an AND/NOR gate on the output path out of the block in Low Vdd Standby.

2.2.3 Sleep/Power Gating

Power Gating is a form of Shutdown and is increasingly referred to as Power Shut Off or MTCMOS. A Header or Footer are applied to the block of logic. The Header and/or Footer are controlled with either a zero (Signal based) or negative Vgs (Gate

Source Voltage) to cut off the cells from Vdd and Vss. The Header and Footer may be shared between cells, rows, or even whole areas of the die. Often, multiple Header/Footer transistors are used in parallel, to ensure there is sufficient current delivery strength and capacity to withstand fluctuations in current.

A common approach is using logic signals to control the Header/Footer transistors, although charge pumps provide better leakage reduction. However, using a logic signal relieves the need to use both charge pumps on die as well as route a separate rail.

2.2.4 Retention

Retention is a combined variation of Standby and Shutdown: the block is shutdown achieving low leakage, but state is not lost. There are many ways to accomplish this, all the way from using software based schemes to leaf cell implementations in the library. The generic aspect in every scheme however is utilizing a save operation prior to shutdown and a restore after wake-up, both of which are carefully orchestrated in control sequence. *Design for Retention: Strategies and Case Studies* [5], a paper co-authored by one of the authors, David Flynn, is an excellent source of information on this topic.

2.2.5 Dynamic Voltage Scaling (DVS)

Dynamic Voltage Scaling is a technique in which the rails of an island are varied, especially, the Vdd rail, to achieve various power/energy targets. Strictly speaking, this should be Dynamic Voltage Frequency Scaling (DVFS), since frequency is usually scaled with Voltage.

2.2.6 Discrete/Continuous DVS

- If the cells of an island are not operational, i.e., the clock is gated down during a shift in the DVS operating point, the mechanism is discrete DVS.
- If the cells continue to operate, the mechanism is continuous DVS.

Note—DVS need not be pure Vdd scaling. Techniques such as back bias or forward bias may be used to achieve different frequencies.

2.2.7 Discrete Vs Continuous Voltage Scaling

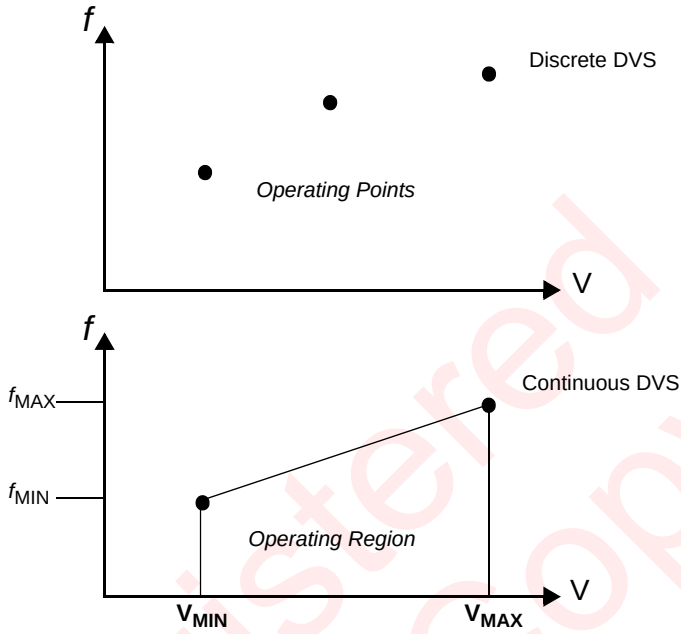


Figure 2-7. Voltage Scaling—Discrete vs. Continuous

2.3 Conclusion

Given the various choices available in design style, what determines the architectural selection? This is quite a tricky question to resolve, especially when most of the operations of SoCs are software driven. In general, a few principles apply, which are described below:

- Leakage reduction is primarily achieved by shutdown or standby. About 5–10 years ago, the choice was to use external voltage regulator shutdown or to use Low Vdd Standby. However, Power Gating and Retention are the most commonly used techniques today (for various reasons beyond the scope of this book).
- Multi-media applications which are dynamic power dependent are still heavily reliant on clock gating and increasingly DVFS.
- Large memories and register files choose to use back bias for leakage reduction, although the use of this technique is becoming popular with certain applications for random logic as well.

In summary, multiple techniques are available to the user for effective power management using voltage variation in spatial domains and time or both. However, each technique as adopted and implemented brings forth new challenges in achieving functional correctness. The next chapter illustrates some of the things that could go wrong, ranging from very simple to extremely complex.

CHAPTER 3 POWER MANAGEMENT BUGS

ABSTRACT

The various causes and effects of power management bugs are studied using examples. A high level classification of bugs is also made. Appropriate rules and recommendations are discussed along with the bug examples.

3.1 INTRODUCTION

In this chapter, we describe various types of bugs in multi-voltage designs that occur in various power management schemes. By no means is this meant to be a comprehensive list. Our attempt, however, is to show the various sources of error and caution to the user that these errors require a voltage aware detection strategy, and guide the formulation of a test plan. We also attempt to alert the user that apart from detection, the debug of such failure conditions also involves a mindshift from simple boolean to voltage-aware boolean thinking.

Broadly speaking, we can classify bugs to be found in Power Management under the following causes:

- Structural errors
- Control Sequence errors
- Architectural errors

However, power management bugs are also characterizable by their effects: not all bugs are errors in the observable “1/0” functionality of the chip/system. They can be classified as follows:

- Observable functional errors
- Internal logic corruption
- Potential device failure
- Excess current consumption
- Architectural failure: failing to meet a system requirement such as power budget

For example, if a chip works as intended, but the power consumption is higher than the specification, this can be attributed to an error in the power management architecture itself. However, it is not the type of bug that can be found in traditional silicon debug and characterization. Detection of this behavior may require a certain level of system integration and application stimulus to be applied. In this chapter, we study both the causes and effects of power management bugs through examples.

3.2 Structural Errors

These are errors caused by a design structure. This class of errors is mostly detectable by pure static analysis, *assuming there is a specification to check against*. In general, these errors can be rectified by changing the design structure. While the emphasis on the existence of a specification seems trivial, one must remember that at the time of this book’s initial edition, power intent specification languages were in their infancy of discussion and standardization. It is not necessary that structural errors occur in only the RTL or the netlist stages. They could occur in any phase of the design. However, the end result is often the same. A structural error results in electrically bad connections, leading to functional bugs, device breakdown or excessive power consumption.

RTL structural errors often arise out of coding practices that are not cognizant of multi-voltage errors or often the result of integrating Intellectual Property (IP). In our experience, at the RTL and SoC integration stage, errors are also caused by improper specification of the power management scheme.

Another common cause of structural errors in RTL is a methodology in which the protection circuits have been fully or partially inserted into RTL, especially in or around an IP being integrated into the chip. At this stage, the designers derive the benefit of dynamic verification on control signals directly hooked up to isolation

devices etc. This verifies their control sequence at RTL instead of having to wait until the netlist stage.Many structural errors occur as a violation of this fundamental rule:

Rule 3.1a — A spatial crossing must be protected with the appropriate circuit at all times.

Rule 3.1b — If an enable signal exists, the protection circuit must be suitably enabled or disabled from its function at all times.

There are various types of protection circuits used in current practice. In chapter 2, we defined two generic cells: Isolation and Level Shifters. In the following subsections, we take a deep look at the issues that could arise out of a violation of Rule 3.1.

3.2.1 Isolation and Related Bugs

Power gating and external VDD shutdown are the most commonly used low power techniques today. Therefore, isolation devices are used quite often. It is therefore, no surprise that many bugs related to isolation devices occur in the design. This subsection gives a few examples.

3.2.1.1 Missing Isolation

The following is a simple example of Rule 3.1a. When a source domain such as the On/Off domain in Figure 3-1 goes into the off state, an isolation device must be placed in the path between any source signals from the off domain to any fanouts in the on or standby state domains. In the absence of isolation, a floating net will directly feed the CMOS gate or diffusion terminal in the on domain, causing the following effects: logic corruption in the subsequent stages of the on domain, excessive power consumption in the on domain and in extreme cases, a breakdown of the device itself. Missing Isolation (or level shifter) is one of the most common bugs encountered in low power design. Fortunately, it often takes only a structural check to detect this. Avoidance of this bug has also become easier as design teams transition from manual insertion at RTL to automated insertion in netlist stage through the use of policies in the power intent description.

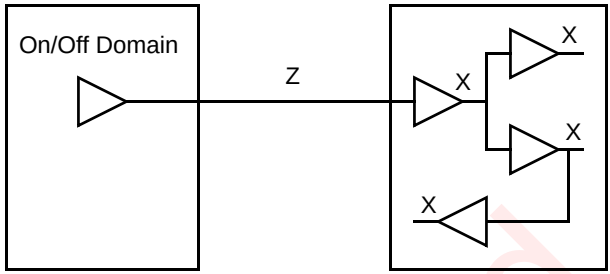


Figure 3-1. Missing Isolation

3.2.1.2 Incorrect Isolation Polarity

By default, most signals are isolated to Logic 0. However, if the signal being isolated is an active low signal, then the isolation polarity must be Logic 1. As shown in Figure 3-2, it would be incorrect to isolate an active low signal named reset_N to Logic 0, since it would place all of its fanout in reset state for the duration of isolation. Similarly, many protocols, such as PCI and USB, are implemented with active low request lines for bus access. Isolating them to Logic 0 causes spurious grants, degradation in bus performance, and frequently deadlock conditions.

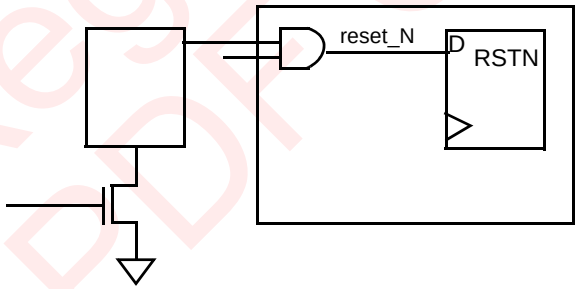


Figure 3-2. Isolation—Wrong Polarity

3.2.1.3 Incorrect Isolation Enable Polarity

In this case as shown in Figure 3-3, the isolation enable polarity is incorrect. This can be caused either by an incorrect connection to the Power Management Unit (PMU) or by implementation error in RTL/Netlist. The effect is that the isolation device is not

functional when it is needed—in fact, it may turn the other way around. Isolation may kick in when not needed and be dysfunctional when actually needed.

Note that this can be a control error as well. We will see another flavor of this bug later in the chapter.

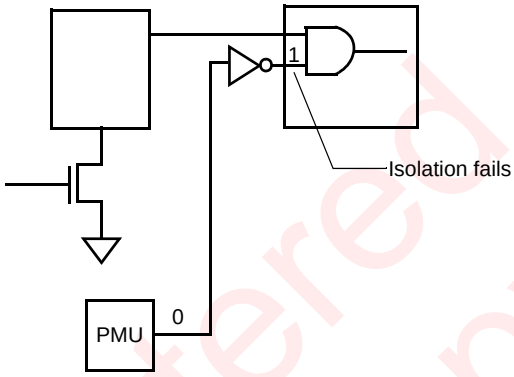


Figure 3-3. Isolation Enable—Wrong Polarity

3.2.1.4 Incorrect Isolation Gate Type

Consider Figure 3-4. In this case, the isolation device exists and suitable control is exercised by the Power Management Unit. However, the isolation gate placed is an OR gate instead of an AND gate. Hence the gate does not respond to the Isolation enable in the desired manner. This bug was often caused by methodologies which manually inserted isolation gates into the RTL. However, with the advent of automated tools, the bug has often transformed into Incorrect Isolation Polarity or Incorrect Enable Polarity.

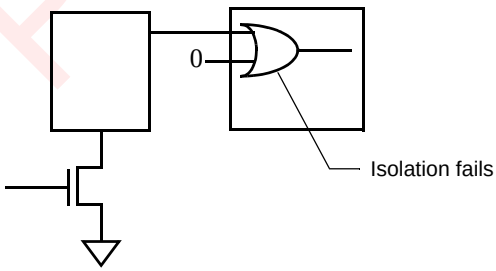


Figure 3-4. Incorrect Isolation Gate Type.

3.2.1.5 Redundant Isolation

Often, the problem with isolation is that it is applied where it is NOT needed. Such isolation is redundant and is extremely detrimental to functionality. There are various flavors of redundant isolation: two dangerous ones are described below.

In Figure 3-5, consider the spatial crossing (crossover) between Domain 1 and Domain 2. No isolation device is needed: inserting one is a waste of area and causes a timing penalty. Even worse, when Domain 3 is switched off, Signal A is needlessly tied to an isolated value of Logic 0, thereby “freezing” Signal A as long as Domain 3 is off. Any changes on Signal A are lost in this period. Redundant isolation devices can hence cause very serious functional errors by “killing” or “freezing” logic signals inappropriately. This bug can be caught statically. Note that only the isolation device on Signal A is redundant in this case. A variation of this bug can be caused by control error as we see in the section Control/Sequence Errors- this will be a situation where the required isolation gate on Signal B is rendered redundant.

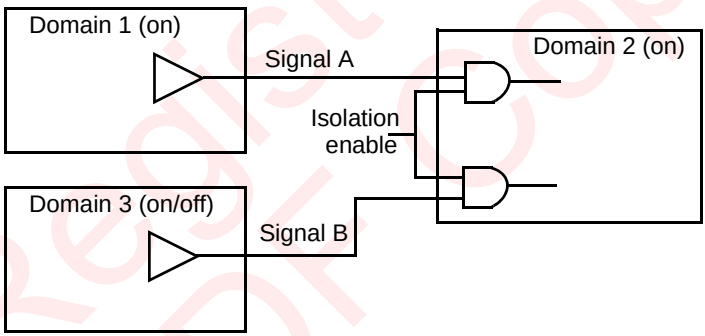


Figure 3-5. Redundant Isolation

3.2.1.6 Gated Latch: Undefined Last Known Good State

In most of the prior discussion, we have subtly alluded to isolation devices as logic primitives such as AND/OR or their inverted versions. However, latch based isolation is fairly common, even if not as often used for the additional area penalty it incurs. latch based isolation adds a new dimension to the spatial crossing. As opposed to gating down a fixed logic value to the isolated signal, the latch device retains the Last Known Good State (LKGS) of the signal to be isolated. The reader can probably see where this leads. There is no fixed “isolation value”: When the source island is powered off, the destination may see either a logic 0 or logic 1 as isolation output.

Hence, the burden of verifying functionality in both cases (sigh! more work!) falls upon the verification engineer. This can quickly extend into complex tests once many latch based isolation instances are used; hence, designers have avoided this strategy unless it is necessary.

LKGS also presents an often unnoticed, hard to debug but severe hazard to the power up sequence. When the chip wakes up from an all off state (cold start), the LKGS value is unknown i.e., logic "X." If the receiving domain in which the latch isolation element is present wakes up before the source element, then the wake up process must be able to proceed without any dependency on the unknown last known good state(!). One easy workaround to this issue is to reset the latch at power up; however, many current implementations (regrettably) do not support the use of resets in isolation latches. Thus, latch based isolation must be covered in verification for a state where the LKGS is not merely 0 or 1, but also for logic X.

Rule 3.1c — Latch based isolation devices must be tested for wakeup from the all-off state for the entire power up and down sequence.

Recommendation 3.1d — Use appropriate reset for latch based isolation devices.

3.2.1.7 Ungated Latch—Unknown States and Loss of Control

Designs with bus implementations have long used keeper devices: inverter loops that maintain state without any latch enable. Some designers have extended this practice to their use as isolation devices. When used as isolation, these devices suffer from the same issues as gated latches in the previous example. They need to be validated for the last known state being logic 0 or logic 1. In addition they need to be tested for undefined state when waking up from the fully powered down state. Many bugs also arise from the fact that an ungated latch is extremely hard to reset, especially when the primary driver of their net is powered down.

It is also an error to have a different voltage level driving the keeper than the source domain. Imagine a situation where the On/Off domain is at 1.2V, but the keeper is maintained at 0.7V, presumably to "save power." This leads to quite a few problems—electrical and functional. There may excess power consumption when the source domain is On. There may be logical corruption when the source domain is Off, because the keeper is not able to provide sufficient voltage level for the logic value.

Extending this concept further, as a domain shuts down, the ungated latch is exposed to the intermediate logic/voltage values on the output signal and may become

corrupted, thereby destroying the very premise of isolation. Needless to say, these devices are not commonly used.

Mere static verification to check the presence of isolation is not sufficient for this type of isolation. Fairly comprehensive vectors need to be applied to verify that these design styles work. However, these have been classified as structural errors because they are primarily caused by a choice of design structure.

3.2.1.8 PULLUPS/PULLDOWNS—EXCESS CURRENT CONSUMPTION

Most electrical integrity rule sets have outlawed these devices practically. Nevertheless, their use as isolation devices is found occasionally. While logically the same as AND/OR based isolation, difficulties in testability, silicon debug, characterization have limited these devices. They experience similar bug profiles as AND/OR isolation types.

Additionally, ungated latches, pullups and pulldowns represent one major problem: the isolation, if turned on before shutdown (as is usually expected) can cause an enormous amount of short circuit current in the time to actually shutdown. This of course happens, if the existing logic level is opposite to that of the pullup/pull down device. For example, if a signal level is logic 1 and a pull down device is enabled, then there is a excess current consumption from the driver of the signal through the pulldown device. Likewise, as the domain wakes up, the output signals of the domain may conflict with the pullup/pulldown device, causing a spike in current consumption.

Recommendation 3-2 — Do not use ungated latch or pull up/pull down type of isolation.

Before we conclude the subsection on isolation, we would like to present the reader with a quandary. Can an isolation device be present in an intermediate domain, i.e., connected to the power rails of an intermediate power domain? The answer seems to be obvious—yes, as long as the isolation device is also on when its receiving domain is on. Many modern implementation tools use this logic. However, the hazard of doing so is the situation when both source/destination domains are off, but the isolation device remains on. This situation represents a potential hazard in the form excessive current consumption. This occurs when a sneak path (described in “Input Isolation (Parking)” on page 37), forms as a result of this connection/state combination. Luckily, this situation can be detected statically.

3.2.2 Level Shifting and Related Bugs

A level shifter must exist on a spatial crossing when there is a disparity in voltage levels between its source and destination. As the user might recall from Chapter 2, a Logic 1 is referenced with respect to the Vdd rail of the domain from which the signal originates. The driven net is charged up to a voltage equal to the Vdd. This works well within the domain. However, when crossed into another domain, the voltage level on the net may not be adequate to be recognized as a Logic 1. Even if it is, it may place the receiving device(s) in a region of the CMOS curve that is consuming enormous current.

Failure to insert a level shifter results in either corruption or logic levels in the receiving domain or excess current consumption or both. Logic level corruption usually propagates as a functional failure downstream. Excess current consumption can occur in the form of short circuit current in the case of missing low voltage to high voltage level shifters and excessive gate leakage

Rule 3.2a — A level shifter must be used when there is a difference greater than a number determined by the technology library, between source and destination Vdd levels.

3.2.2.1 Level Shifter Out of Range

As shown in Figure 3-6, it is not sufficient to connect a level shifter. One must pick a level shifter that can handle the voltages applied to it. In this figure, the level shifter specification requires Vs (the source voltage) to be between 0.8V and 1.2V. The applied source voltage of 0.7V is lower than the minimum required specification of 0.8V as source voltage. Note that library formats such as Liberty now have attributes that have specifications for voltage ranges, which can hence be checked statically by tools. The error is mostly caused by manual insertion. Automated tools that synthesize power intent often do not have this bug.

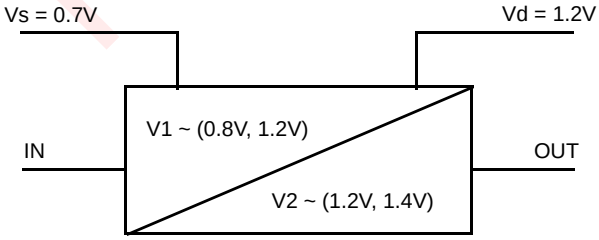


Figure 3-6. Level Shifting device out of range

Rule 3.2b — Tolerance of the Vdd levels must be taken into consideration for level shifting determination.

3.2.2.2 INCORRECT DOMAIN OF LEVEL SHIFTER

A level shifter must be connected to the appropriate rails on the source and destination side, so as not to violate its purpose itself, as shown in Figure 3-7. In this example, the level shifter is placed in intermediate domain V1 which has a level of 1.0V. This means that another level shifter is required when the output of this level shifter is connected to fanout in the power domain supplied by V3. Thus two level shifters are required instead of one, increasing area, power and delay. The placement of level shifters in intermediate islands is fine as long as it does not require another level shifter and isolation device by doing so.

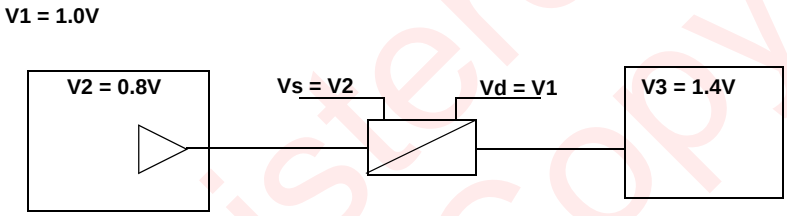


Figure 3-7. Level Shifter in Wrong Domain

While it is easy to conclude that level shifting needs are entirely statically detectable, they depend on the ability of the user to predict the power state table. The task of defining a power state table gets very cumbersome and error prone as the number of islands and operating voltage points increases: sometimes it runs into the hundreds and thousands. Current power intent specifications also do not make it easy to express designs with a large number of power domains in terms of a state table. For example, a design with seven domains has about 128 states just turning domains on and off, not including various variable voltage states as well as transient conditions. Hence, there is a degree of dynamic verification needed across relevant modes and corner conditions for level shifter verification.

Rule 3.2c — Level Shifting must also be determined by transient conditions on voltage rails across each source/destination pair.

Rule 3.2c can be predicted statically to a certain degree. However, it is essential to simulate, such a set of conditions to obtain coverage dynamically. Note that voltage rails often take several microseconds and sometimes in the order of milliseconds to execute transitions: violating level shifting rules on transitions therefore exposes

many devices to high current draw conditions for fairly long periods of time. This point is illustrated in “Logic Corruption” on page 64.

3.2.3 Other Structural Errors

We certainly do not want to leave the user with the impression that structual errors are all related to Isolation and Level Shifting! Other structual errors are possible but not often within the purvey of front end logical verification.

For example, consider a power switch structure that causes excessive in-rush current (di/dt effects). This error needs to be detected by a post layout electrical integrity analysis. Unchecked, this leads to a collapse of the power rails and logic levels.

Errors are also possible within individual standard cells by erroneous transistor structures. These require static transistor level analysis. Increasingly these tools are available in the market for use at the netlist level.

Hard macro internals form another source of structural errors. In our experience, some of these are caused solely due to the fact that there was no formal specification method to communicate the internal power domain partitions of the hard macro or the pin level partitions. For example, consider an IO-pad cell that has 1.8V and 1.2V domains. If the 1.2V domain can be internally shutoff, but has built in isolation, then it is an error to connect isolation devices externally as well. Fortunately, library specification formats have evolved to represent all these attributes, leading to automated tool support for static detection of such errors.

3.3 Control/Sequence Errors

This category is by far the most frequent source of functional errors. Errors often occur because the control of power management events is not exercised appropriately. Power management events include both movements in voltage as well as changes in logical signals such as isolation enables, resets, clock gating signals etc. In many cases, these bugs rise from the design of the power management unit itself. Most of today’s designs combine IP blocks from various sources, including inherited blocks from previous generations and third parties. The power management unit is therefore a major aspect of the integration, and its thorough verification is central to avoiding bugs.

Historically, many designs that adopted multi-voltage low power techniques had mostly structural errors caused by lack of automated tool flows. Now that we are well into solving that problem with the current generation of tools, getting the control right is the next major hurdle.

3.3.1 ISOLATION CONTROL ERRORS

The mere presence of isolation devices is not sufficient. The isolation device must be exercised on its control input to be actively isolating when there is a shutdown condition. This condition cannot be verified structurally: it needs temporal verification either by dynamic simulation or other techniques, such as property checking.

Author's note: This section has a different format because we are describing Control/ Sequence errors. As a precursor to later verification, we are introducing how the testbench/test vector must be built to catch the bug.

3.3.1.1 INCORRECT ISOLATION ENABLE TIMING

Description — In this bug, which is extremely common, the source island is shutdown, but the receiver's isolation gate is not enabled. This causes an unknown value at the output of the isolation gate. Refer to Figure 3-8 below. When Signal A transitions from Logic 1 to Logic 0, the footer transistor is cut off. However, Signal B, the isolation enable, arrives much later. This causes an unknown logic value and excess current consumption in the isolation gate. Worse yet, the unknown logic value at the output of the isolation gate may not be observed in the logic regression and end up detected only on silicon, as high power consumption. The methodology recommended in this book seeks to avoid this first by requiring coverage of such elements and second by recommending assertions around each isolation device to test for such a condition.

As shown in Figure 3-8, this bug is likely when an On/Off block such as a power gated domain feeds an source that is On. An isolation device exists, but must be tested to verify that the isolation enable timing is incorrect.

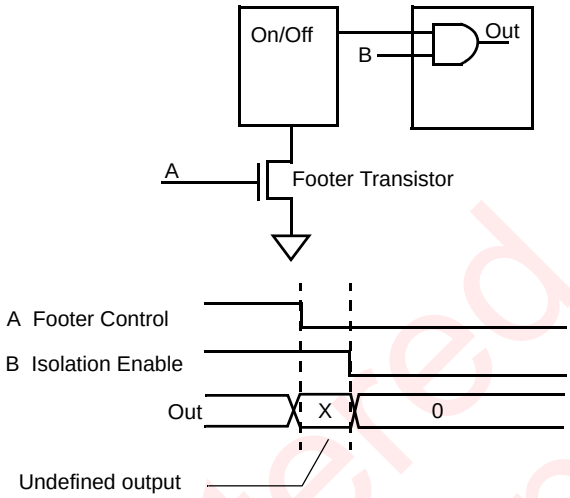


Figure 3-8. Incorrect Isolation Enable Timing

Testbench — The testbench should exercise the voltage vector to make the source logic go to shutdown and wake-up. Also, the isolation signal needs to toggle accordingly, as exercised by the PMU.

Verification — It must be verified that no logic corruption propagates into the On island through the isolation gate. This must be tested at each isolation device. An assertion must be activated at each isolation point to ensure that there is a sufficient timing window between the activation of isolation and shutdown (and vice versa).

Principles to avoid this bug —

Author's note—these principles and others have been captured as rules. They are included here to capture the perspective of the original contributors.

- Isolation Enable with the correct polarity must occur *before* Power Gating happens at *each* crossover protected by an Isolation device.
- An isolation device of appropriate device type is needed every time a signal crosses over from an island that can be off to an island that is on. Similar rules *may* apply to a signal crossing from an On island to an Off island.
- Isolation level of the signal must be set to an inactive level, so as to "kill" the signal.
- Isolation devices must be hooked up to the correct (On) voltage rails.

Guideline 3.4a — Crossovers and Isolation enables are essential coverage points.

Guideline 3.4b — Verify that isolation levels are inactive. For example, a fan can't be tied to “on” level, an arbiter request cannot be tied to a logic level that indicates active request.

Guideline 3.4c — Signals with isolation should be level sensitive, not edge sensitive. The very act of isolating and releasing may cause an edge.

3.3.1.2 Redundant Isolation

Consider a case as in Figure 3-9 when the isolation_enable is turned on when Domain 3 is on. Signal B is now “frozen” when the source domain is still on. However, this is a tricky situation. Most power management protocols require that isolation be turned on prior to shutdown. For example Domain 3 is still on and requires isolation to be turned off well after the Domain is powered up. The temporal detection of this situation is much harder than a simple assertion based check. One good indication of activity is a wiggling clock to Domain 3. The problem can be made easier by adopting a protocol where the activation of isolation should follow disabling the clock and follow the reverse sequence during power up. After power up, it is also feasible to use the reset signal as an indication to see if isolation can be removed.

Rule 3-4 — Redundant activation of isolation must be checked by appropriate assertions.

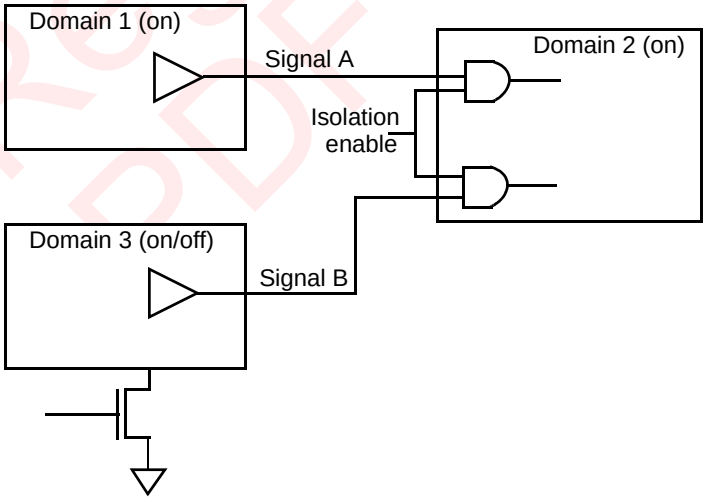


Figure 3-9. Redundant Isolation

3.3.1.3 Memory Corruption in Standby (Unsafe Write)

Description — Memory elements such as registers, latches and RAMs in the design can be taken to an optimum lower level of voltage (Low Vdd Standby) and hence significantly reduce leakage. This way, they retain the value but will not be able to drive or accept any value to be written to them. If a clock wiggles at the input of standby element, it may so happen that the data it is retaining gets corrupted. This is a functional bug. As the memory element wakes up, it may contain corrupted bits that can cause a design malfunction.

Testbench — In this situation, a flip-flop as shown in Figure 3-10 is considered a memory element powered by Vdd. To put this in standby mode, Vdd is reduced to the standby voltage through a voltage regulator.The testbench needs a voltage regulator model to change the voltage applied to memory element; in this case, a flip flop. The voltage regulator takes the voltage value to standby levels. The clock is toggled when the flip flop enters standby state.

Verification — Conventional simulators do not have the mechanism to identify voltage or a standby mode: a power aware simulator needs to be used. Once the voltage is at standby, if clock toggles it will corrupt the value retained in the memory element. Memory is corrupted because this is an *unsafe write*. An automatic assertion failure is required with information to debug the offending write and to ensure that the failure is not missed.

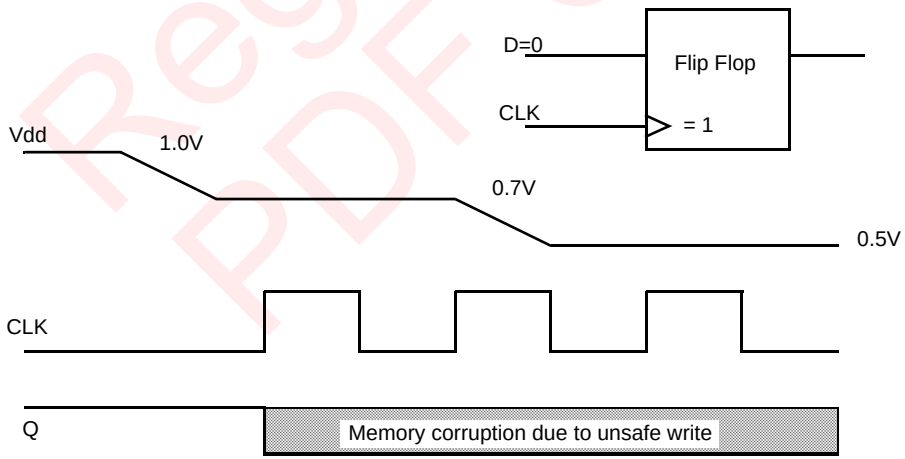


Figure 3-10. Unsafe Write

Rule 3.5a — Do *not* wiggle any pin, especially clocks, read/write pins when in standby. (Resets are usually supported at this voltage.)

Rule 3.5b — A clock gating device must be present and gated to inactive clock level in this situation.

3.3.1.4 Save and Restore Sequence

Description — Retention methodology typically requires special control signals, such as save/restore, to be generated by a power management unit. The sequence for this power save mode should be typically, save-power gate-wakeup-restore. Though the sequence is followed, premature scheduling of these signals will corrupt the restored value and design will wake up in unknown state. This will cause the design to malfunction.

Testbench — As shown in Figure 3-11, the stimulus applied needs to put the functional block (a DMA block in this case) into shutdown and wakeup with the save-restore signals exercised appropriately. The two registers inside the power gated DMA, regA and regB are retention registers. In this case, the restore operation for regA happens before full wakeup of its supply voltage, corrupting the restore value. The restore for regB follows the correct sequence, hence avoids corruption.

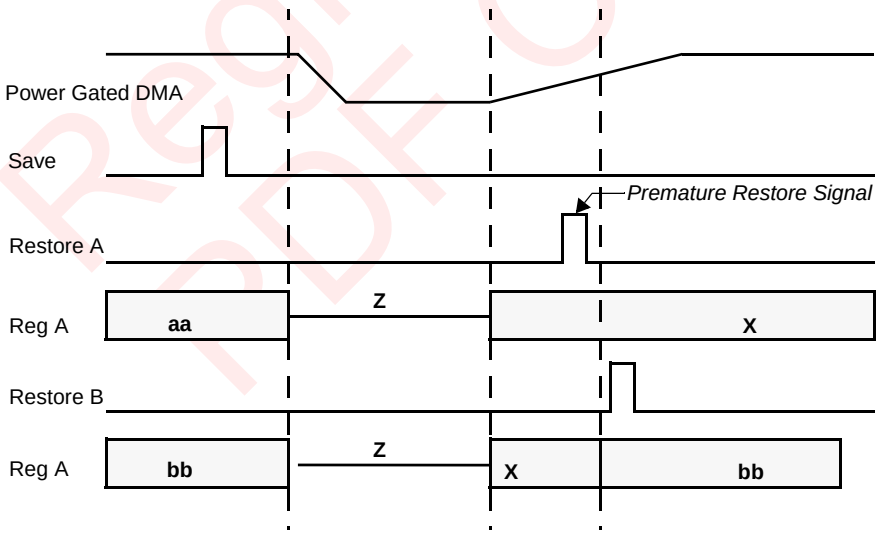


Figure 3-11. Premature Restore Error

Verification — A corruption in the register value must be manifested by the simulator. In addition, however, there must be adequate coverage (and observation) of the restored registers after the wakeup event. Assertions are hence needed to guard every retention element against such corruption.

3.3.1.5 Power Wastage due to Control Error

Description — Once a block is power gated there is no need to wiggle any inputs to this block. Any input wiggling at the inputs needlessly toggles the capacitance of the loads and hence wastes power. Especially so for high fanout and high toggle rate signals, such as clocks, toggles during shutdown states as shown below in Figure 3-12, can waste a lot of power. These toggles can be cut at the source to save power. Conventional low power verification, such as X-injection, will not be able to locate such problems dynamically.

Testbench — The testbench needs to first put a power gated island in OFF state. While the island is power gated, the clock must be checked for any toggles that cause power wastage. This is a very difficult bug to catch, since the functionality of the device is not affected.

Verification — This bug shows no functional impact in the simulation. Hence, only a set of carefully applied assertions can detect this occurrence. Note that the mere static detection of clock gating elements is insufficient.

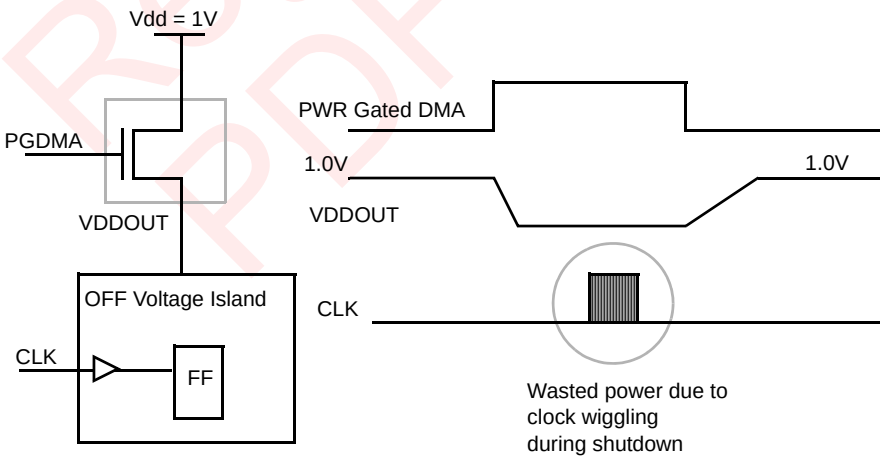


Figure 3-12. Power Wastage

Rule 3.6a — Clocks, resets and other high fanout nets in off islands must be gated inactive when the island is powered off.

- This sounds like a structural check, but the mere presence of clock gating does not help. An assertion must be kicked in to make sure the gating is actually activated.

Rule 3.6b — A transistor level check must be done to waive this violation.

- The first stage in the off island is not allowed be a pass transistor or a diffusion connection. A pass transistor may breakdown under these conditions.

3.3.2 Logic Corruption

Description — Most physical libraries come up with ranges of voltages where a level shifter is not required. In this case, the specification is such that any difference less than 150mV of Source-Receiver driving voltages does not need a level shifter.

Many designers look at the power state table combinations to determine whether a level shifter is needed. In the test case as shown in Figure 3-13 below, the two states possible for the domains are (1.2V, 1.1V) and (0.9, 0.8V) as power states. If one were to statically analyze this, it can be concluded that a level shifter is not needed between these domains. However, when the design transitions from one power state to another, the voltages may not maintain the difference to be under 150mV, hence violating the level shifter rule. This kind of error is purely dynamic and can be caught by only by targeted testing or exhaustive random testing. Some amount of static analysis of ramp rates is possible, but that may not include the dynamic events that influence the voltage ramp.

The design may or may not display functional failure; however, electrical failure, timing variations, and excess power consumption are likely. The damage entirely depends on the difference in voltages and whether any transistor structures are exposed to unsafe conditions. The source of error is obviously in control: multiple rails moving in value at the same time cause a lot of electrical disruption. A CMOS device with its source/destination Vdd rails moving in value at the same time could behave quite unpredictably. This can be avoided by scheduling the voltage changes in a safe manner.

Testbench — The test case must be written with worst case voltage transitions in consideration for each source-receiver direction. The maximum disparity between voltage levels between sources and receivers must be tested for correct functionality.

Verification — It is not standard for libraries to export a variable that indicates logic tolerance levels (much like Vil/Vih in board level design). One facet of this bug is that multiple rails are transitioning. Assertions are needed to guard against such a situation, and can be written easily; however, that blows up in size as the number of rails grows. Static analysis can be performed on the power state table to analyze this possibility. However, note that dynamic simulation does not always comply with the state table or follow state transitions in a predictable manner. Designs do land into illegal states and transitions!

Rules — No more than one rail should transition at the same time. A waiver can be granted if there are no crossovers between the islands controlled by these rails. However, this is for functional verification only. Electrical integrity verification must still be done when such a situation exists.

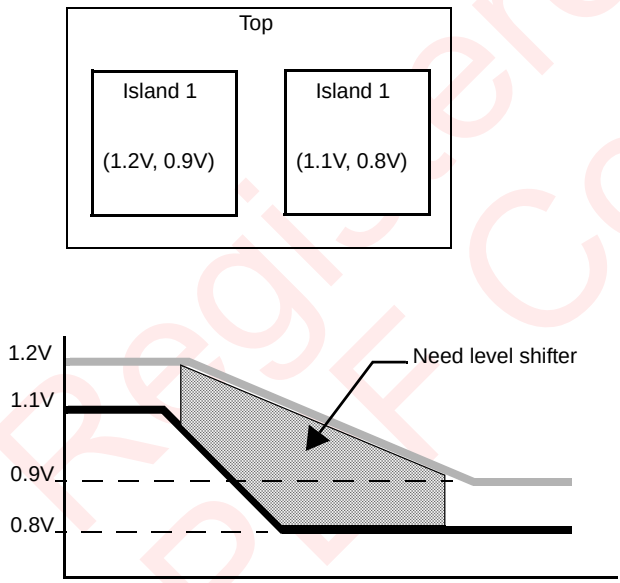


Figure 3-13. Dynamic Level Shifting Error

In summary, the errors possible are many, once the power management scheme is inserted into functional logic. The verification engineer must scope out the tests required and proceed accordingly. In every case, however, there is no substitute for a sufficient test harness (Chapter 5, “Multi-Voltage Testbench Architecture”), well planned test methodology (Chapter 6, “Multi-Voltage Verification”) and well managed coverage and assertions (Chapter 6). However, before we proceed down that path, we need to cover an important new source of control and sequence bugs, state

retention, which as we have mentioned is an entirely new semantic to current languages. That is the topic of discussion in the next chapter.

3.4 Architectural Errors

This category typically shows up as a problem in system integration or operation. Despite the chip being fully functional, according to "specification," a real life system that integrates the chip may not yield the desired results.

Consider the case in which an idle block that is not being used should be turned off. In this case, the functionality of the chip is unaffected, but the power consumption may be higher than expected, and then drain the system's battery further or raise its standby current draw. While this is not a strictly functional error, it is an error in the partition scheme of power domains. Similarly, if there is an erroneous policy to schedule the voltage, the device may spend an inordinate amount of time in voltage transitions, thus expending more energy than intended—such a bug is hard to find and debug.

On the relatively simpler side, architectural bugs that put a device resource in inoperable states such as shutdown and standby are easier to find. For example, if the cache is not powered on when the processor is on, this is an error in power sequence and/or policy, but can be found as a functional error.

Architectural bugs are many in their nature and unfortunately, they do not show up until the system is actually running software in its finally integrated configuration. They may show up in some configurations, as when a certain wake-up pin is used, but not in others.

The detection of architectural bugs needs a very good understanding of the end system integration and use model in software, followed of course by a specification system into the automation tools for analysis. This is easier said than done, but we will discuss how to do this to a very high degree of coverage in the test plan section of Chapter 6.

3.4.1 Power Gating Collapse

Description — MOS transistors are dependent on their gate-source voltage difference to determine whether they are "on" or "off," also known as "conducting" or "non-conducting." This mechanism is used in power gating. The gate voltage is

such that the transistor is non-conducting. For example, as shown in Figure 3-13 below, this is done by issuing logic “1” to the gate, which charges it up to the Vdd level of the driver. In general, this is the same voltage level as the power switch. The gate-source difference kicks in to turn the transistor off. However, when the Vdd of the power gating signal's driver “dips,” the off island makes an “on” excursion and come back. On the other side, the power gate can also become more resistive. Similarly, an on island with a footer can suddenly become more resistive or make an on excursion.

This phenomena is quite dangerous, as it will lead to a current spike and a further collapse of rails. Although the profile looks like a power integrity issue and may well be caused by bad implementation of the power grid, frequently the cause is an over-scheduling of power state changes instead of staggering them in time. This in turn causes fluctuations in the power supply. The PMU must take the stability of the power supplies into account before moving rails in voltage value.

Testbench — It is hard to write a test for this. However, the following conditions must be satisfied:

- The threshold voltage of the power switch needs to be declared for all power gated values.
- The “dip” in voltages especially when islands turn on/off must be modeled.
- The worst case situation is when a maximum number of rails change at the same time. The test case must try to create a meaningful stress on the SoC to reflect this.

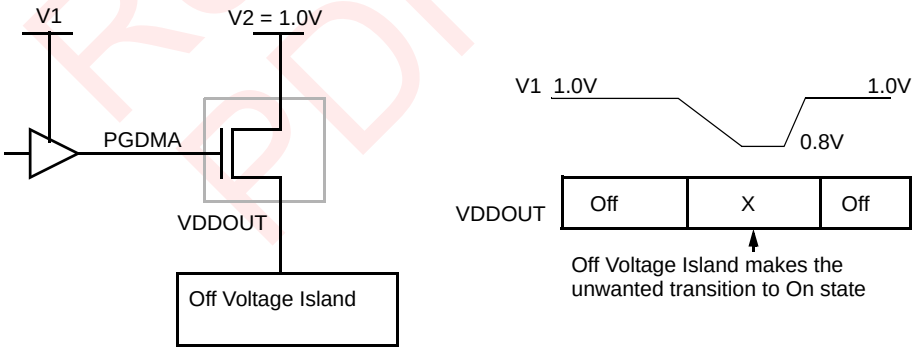


Figure 3-14. Power Gating Collapse

3.4.2 MEMORY CORRUPTION IN STANDBY

Description — Memory elements in the design can be taken to an optimum lower level of voltage so that they retain the value but will not be able to drive or accept any value to be written to them. This state can be called as a low Vdd standby mode. This memory element cannot be put below the minimum standby voltage value required by the technology and circuit implementation as the contents would be corrupted invalid. Therefore, it is very important to ensure the driving voltage does not go below the minimum standby value specified by the memory element provider.

Testbench — Here, a flip flop, (for example, as shown below in Figure 3-15) is considered as a memory element powered by Vdd. To put this in standby mode, Vdd is reduced to the standby voltage through a voltage regulator. The testbench needs to have a voltage regulator model to change the voltage applied to memory element: in this case, a flip flop. The voltage regulator takes the voltage value to the minimum standby level and then lower (usually in error). This parameter is called *Standby Voltage*, (the technology library should include this parameter, it is usually present in the data sheet of memories and register files). However, the simulation mechanism must have both a corruption model and an assertion mechanism to detect and debug such a failure.

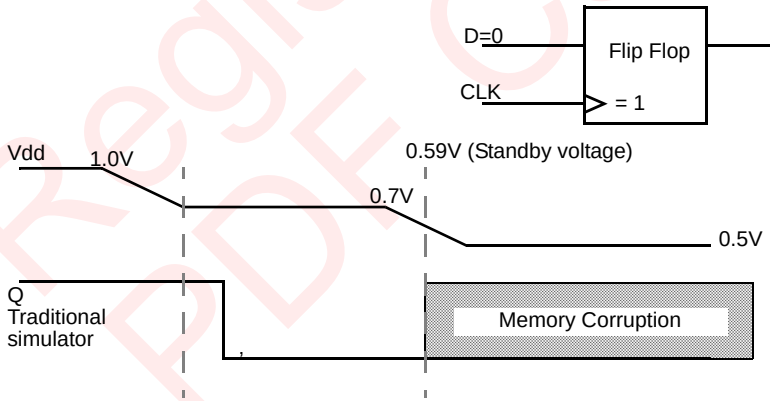


Figure 3-15. Memory Corruption due to Insufficient Voltage

Rule 3.7a — The Voltage ID (VID) codes of a voltage regulator must have assertions on them to detect the scheduling of voltage.

Rule 3.7b — Memory IP datasheets or technology files must specify the minimum standby voltage

3.4.3 Modeling External Components and Software

A change in power state is complicated by the asynchronous, electrical behavior of mixed signal devices such as voltage regulators and power switches. A verification system that does not model these effects frequently misses the bugs involved in closed loop control systems. Further, it is important to note that resets and clocks are governed by these asynchronous signals. In other words, the operation of synchronous systems is “started” and “stopped” by the asynchronous events in a power state sequence.

From another perspective, the power management unit or controller monitors receives stimulus from synchronous systems such as a SoC block. In this case, software writes to a register to change power state, such as to transition an island from 1.0V to 1.2V. The signals for clock gating may be exercised first, presumably according to a synchronous mechanism, and then the voltage regulator itself is sent a signal, indicating the new voltage desired.

However, the transition of the voltage and hence the arrival of any status signal that indicates so is an asynchronous signal, it also takes a much longer time compared to normal frequency of clock operation in today’s ICs. Any verification that does not include the response of the voltage regulator is likely to miss any errors in control that happen around this event. The bug examples in this section illustrate this issue. However, for now, we impose a generic rule that the behavior of the voltage source must itself be modeled.

Rule 3.8 — The behavior of a voltage source must be modeled in an electrically accurate manner.

In the Testbench section (“Testbench Components” on page 89), we go deeper into how a testbench must be set up with models for voltage source behavior. In this chapter, we continue to now discuss the various types of errors that lead to our setting up a testbench in a manner designed to catch these errors.

Another aspect of control/sequence errors is the concurrency of subsystems or events. For example, a system may receive both a thermal interrupt indicating a high temperature and a new application requiring additional processing power at the same time. These unrelated asynchronous events must be processed by a power management unit in a consistent manner across various corner cases. In a modern SoC design, the verification engineer must consider the possibilities to which a

system may be exposed. In Chapter 6 and Chapter 7, we discuss how to address this in a constrained random testing methodology.

Software control is another critical aspect of sequencing. Monitoring various bits by software and hardware, loading any relevant software routines/threads and writing software controlled bits/command sequences are aspects that must be thoroughly verified. This, in essence, is core to the functionality of the chip, which could be further complicated by the presence of multiple processors executing various threads in the system.

Rule 3.9 — The testbench and test cases must include a provision to model software observation and control of power management.

3.5 CONCLUSION

In conclusion, the path to low power design is paved with many errors that are not conventional logic failures. They are caused by a variety of reasons—by design structure, by faulty control, or faulty architecture. Debug can be hard, especially since a majority of the bugs just end up consuming a lot of power. A combination of static checks, testbench components and focused test vectors is the best way to detect these errors early in the design.

CHAPTER 4 STATE RETENTION

ABSTRACT

This chapter will present an in depth view of state retention—the various architectures possible, their trade-offs and verification challenges. Selective and partial retention schemes are then introduced and the increased problems for verification are described.

4.1 INTRODUCTION

As previously noted in Chapter 1, “Introduction” and Chapter 2, “Multi-Voltage Basics”, one of the most effective means to battle leakage is to turn off idle blocks. In earlier generations of process technology, clock gating was used, which was quite effective at controlling wasted dynamic power, but not useful from a leakage control point of view. However, a clock gated block could be resumed without any loss of state—that is no longer the case when the power to a block is turned off. In that case, all state is lost and has to be reset and or restored to return to the original “state” of the system or block.

The loss of state has been the architect's nemesis for a long time. Traditional techniques such as back bias and low Vdd standby have been alternates to a full shutdown. However, low Vdd standby is increasingly becoming unfeasible as process nodes shrink and back bias is not yet supported in all libraries. Moreover, neither technique is a full reduction of leakage, which leaves shutdown as the most effective method to reduce leakage.

Power gating has especially reduced the latency of wakeup as compared to a full external shutdown, which has put further pressure on engineers to use it to quickly resume operation. Power gating reduces the leakage of both logic and registers, but typically introduces an energy cost function, i.e., restart of the sub-system. All the registers in the power-gated subsystem need to be reset from their unknown state at power-up once the power and clock networks have stabilized. There is typically an energy cost in “re-booting” the power-gated subsystem, and a real-time service latency impact. For example, a processing subsystem that services a time-critical interrupt in an SoC design will now have additional latency for being “waken-up” by the interrupt, which must not only restore the processor so that it can service the interrupt, but then handle the interrupt itself.

Architecturally, retention has been around for a long time. Systems and subsystems that undergo shutdown always save context to memory, volatile or non-volatile. This is the operation of the “hibernate” state of a laptop or the channel/volume setting of a television set. In such cases, the shutdown (loss of state) is preceded by a protocol to save the context and then restore the state.

State retention within the IC, however, is an entirely new design paradigm. From the viewpoint of IC implementation, the retention scheme has “penetrated” the leaf level cell. While this scheme has allowed a fast restoration of state, the challenges of architecting the scheme, then verifying and testing the scheme have increased dramatically. Moreover, the cost of state retention that predominantly affected latency and only minimally affected area and timing has shifted so it now affects area to a much greater degree.

4.2 State Retention Approaches

The basic principles of retaining state with hardware and software schemes are described in this section. However, later on in the next section, we will focus exclusively on the verification challenges of “state retention with power gating/ balloon latches/shadow latches”—a retention style that is the dominant trend and one that needs significant changes to verification methodology.

Hardware approaches to IP subsystems are attractive in which the state save and restore functionalities can be added almost transparently. If no changes are required to the third party operating system (OS) or firmware, then the overall project and product time scales will not be impacted. Software approaches that specifically add support for saving and restoring system or task-level state, however, may be required in more complex systems, and may impact the OS kernel or device drivers.

4.2.1 HARDWARE APPROACHES

Edge-clocked RTL design can in fact be powered down and back up again in between active clock edges, provided the register state is preserved precisely. All the combinatorial logic between register stages simply re-evaluates state inputs and regenerates valid outputs, provided sufficient time is given to allow power supplies to re-stabilize and timing constraints are honored.

The basic approaches to providing the illusion of persistent logic state in hardware are as follows:

- Provide distributed state retention within special-purpose extra latch cells
- Sequenced state check-pointing to on- or off-chip memory (re-using scan chains)
- Ad-hoc schemes that “freeze” the clocks and maintain state at reduced voltage (also known as low Vdd standby or “drowsy” state retention)
- State-retention registers that include a third low–leakage latch in addition to the standard

The master/slave latch provides a useful abstraction for implementing state retention**.** The state retention latch (or “balloon” as is often described in the literature) supports an independent backup power supply and some form of sample-and-hold function with voltage isolation. The RTL is synthesized with such registers in a standard flow. Additional control mechanisms are then added to allow the sequencing of saving state (before power down of the logic and the master/slave portion of the basic register) and restoring state once primary power is stable. Optimal RTL for synthesis is highly amenable to implementation with retention flops. However, there is a need to add an external retention signal control sequencer and provide some extra validation sequence testing for entry to and exit from retention. Such retention registers are very fast and efficient to save and restore state, but they incur additional area cost for every such register because they need back up power to maintain latch structure.

Manufacturing test support for synthesized designs is efficiently supported by implementation flows that substitute standard flip-flops with scan-flops that can subsequently be hooked up for efficient Automatic Test Pattern Generation (ATPG) testing. With care, it is possible to re-use these manufacturing scan chains in a form of a shift-register based approach where one can store a functional state using the scan enable control, provided the state is carefully shifted back in before resuming normal circuit operation.

The area costs are minimal compared to distributed retention registers, even with kilobits of storage in some form of a lower-leakage SRAM bank somewhere in the

system. However, there is a state-value-dependent dynamic energy cost in shifting the register contents out and then in. Because of this, care is required not to exceed simultaneous switching IR-drop requirements. System control complexity is also required, associated with the fact that scan-enable control of scan chains typically overrides wait states and clock gating control. The real-time impact is dependent on scan chain length. The more parallel shorted scan chains them better; however, the minimal area cost may be compelling in some situations.

In a voltage scaling environment, there is a third approach that can provide reduced leakage state retention. Provided that the clocks and resets can be parked in an inactive condition and the outputs are clamped when the subsystem is put to sleep, then the standard synthesized circuitry can have the voltage rail lowered to a point that the register latches still hold state (low Vdd standby). However, the registers used in the implementation must be designed to have low retention supply voltage requirements. This approach can be deployed on an independently scalable power rail domain (typically this requires an independent power regulator which has an end-product cost implications) and will have the lowest area impact. However, the real-time cost is dependent on the voltage-scaling ramp times and the leakage savings are harder to quantify compared to on-off power gating.

In all of the above hardware approaches, full state retention is the easiest starting point.

4.2.2 Software Approaches

For designs with integrated Intellectual Property (IP), an attractive alternative is adding an explicit software application programming interface (API) to allow the user to read the context state and saving it to memory. The existing hardware reset mechanism is used to re-initialize the IP state after re-powering and then a software flag is used to indicate whether this is to be treated as a warm restart, in which case the saved state is to be re-written back.

The underlying IP block is required to support reading and writing of key architectural states under program control. For security reasons, this full state access might only be made available via a specific privileged access region or protocol. This would prevent accidental or malicious reconfiguration of the subsystem. The amount of state to be saved and restored can be tailored to the architectural defined state or to a consistent subset of the defined programmer’s model register state. The save and restore API functions then become part of the IP validation model and deliverables. The real-time cost is impacted by the need to read and copy state; the energy costs are highly state dependent.

For a cached microprocessor, there are additional complications to consider. Saving and restoring a state may pollute part of the cache (which has an indirect energy cost to rebuild the cache on resuming operation) or have a more significant real-time impact if the code is run uncached (to minimize displacement of cache contents).

The main issue, however, is the impact on project or product time schedules. Such impact can occur if the operating system or device drivers need to be enhanced and verified with extra API support for retention.

4.3 State Retention Registers

One of the key points to note about state retention registers is that these are actual library elements that accomplish an important architectural task. This requires the following considerations, once the actual circuit elements are provided by a library designer.

- Specify which registers are retained
- Specify the power supplies (primary and backup) to the registers
- Control the save/restore sequence especially as interleaved with clock, reset, scan and power supply functions
- Measure coverage
- Write and measure the appropriate assertions to the retention element
- Verify the design after detailed implementation (and at intermediate stages)

However, this is a minimal verification list. Retention has an impact on implementation flows as well, starting with register selection, placement constraints, library cell design and characterization (and the representation of that characterization), test, and formal verification methods.

In micro-architecture, state retention poses quite a few challenges. We have traditionally used to a model of a register whose pseudo-code can be roughly represented as follows:

```
always @ (posedge clk or negedge reset_n)
    if (! reset_n) q <=0;
    else q <= d;
```

This model and its variants have well understood mechanisms that are technology independent. With the advent of state retention, we are looking for additional

behavior to be captured according to the following: save, restore, power up, and power down. Semantically, an additional bit has been added to the device that must be accurately represented in simulation behavior. Micro-architecturally, the high level choices are:

- Saves/restores are completely asynchronous and have certain mutex (mutual exclusion) periods with clocks/resets
- Saves/restores are completely synchronous and have timing with respect to clocks
- Save/restore are aligned to a particular phase or edge of the clock dependent on a relationship to the active edge of the register
- Save/restores are done with a single signal or dual signals with polarity constraints
- Save/restores have relationships with respect to validity of power to the registers
- Mechanisms provided to reset just the main output of the register vs. flushing the entire retention content

Note that the power management unit is responsible for generating the control signals for save/restore in a timely manner. However, the generation of these signals needs to be aligned to the exact protocol for retention control. Ideally, adding state retention to a design would become as easy as supporting clock gating thanks to the wide-scale availability of standardized clock gating latch elements in libraries. However, the control aspects require some basic control sequencing specific to the style of retention register and to the exact form of power gating employed. For the RTL engineer of the underlying subsystem who designs for transparency, a primary requirement is that the retention control must have “priority” over the clock and set/reset functions of the underlying register.

This is important because the high fan-out clock and initialization networks are typically leaky structures that benefit from power gating, rather than having to keep them active at all times. Additionally, the synthesizable RTL coding needs to have the existing asynchronous and synchronous reset, clock and enable terms maintained exactly in terms of priority before an overriding retention entry/exit scheme is superimposed.

For example, consider the single signal state retention example here vs. a dual signal retention scheme as shown below in Figure 4-1 and Figure 4-2.

The register captures the state at the falling edge of NRETAIN and restores the value at the other edge. While NRETAIN is asserted low, the reset and clock are ignored and can go to X-values in simulation without corrupting the retained state. NRESET

is shown asserted explicitly as well to support any non-retention state that must be re-initialized after re-powering.

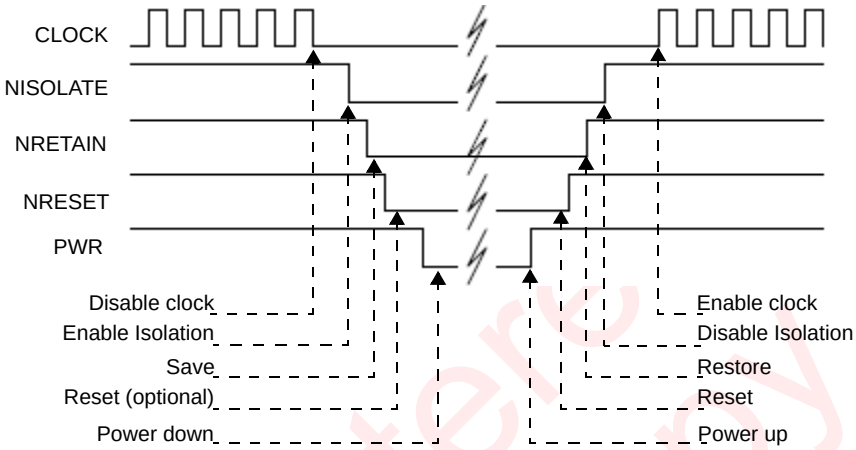


Figure 4-1. Single Signal Retention

In a dual signal retention scheme, two separate signals are routed for save/restore. While this incurs the additional penalty of a signal to be routed, there are advantages in terms of clean generation of multiple retention signals: various registers in a module can be saved/restored in stages as well as multiple restores of the same context that can be undertaken with this scheme, especially as an aid for debug.

Irrespective of the scheme of control used, a few basic generics remain unchanged:

- Save must be done in a clean way before power down; the clock must be disabled to ensure that no further writes occur
- Restore must wait for complete stability after power up and not clash with regular writes/resets

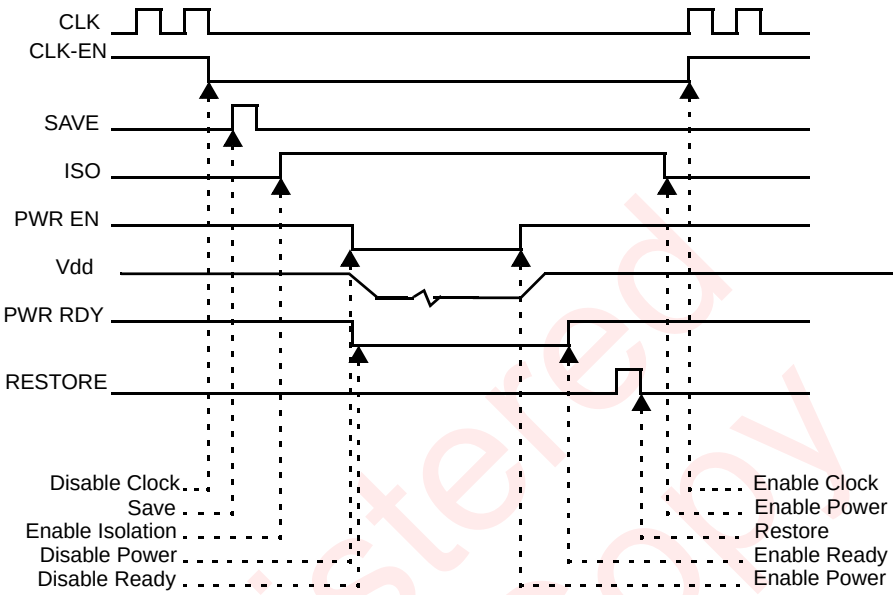


Figure 4-2. Dual Signal Retention

4.3.1 SELECTIVE RETENTION

Selective retention is a design technique where different subsets of retained state are restored (and potentially saved) at different times, using different control signals/ sequences. In current practice, it is more likely to see common save signals, but varying restore signals. One other aspect of retention is an identification of *What* is to be retained and restored, and *How* and *When*. When a subset of state registers is selected as the target of save and restore operations as opposed to the whole set, then it is a partial state retention. In this case, the registers that are not retained are reset.

Selective retention can coexist with partial retention and is also independent of which hardware/software approaches are used. This section goes deeper into selective retention, while the next section covers partial retention.

This idea is both powerful and hazardous. Architects get finer control on the state or context being saved/restored and get a further handle on latency, so this technique may be essential. A power domain may have registers of multiple clock domains or perform a staggered wakeup of subdomains, thus requiring separation in the restore

signals. In some cases, this may be essential: the wakeup of certain elements is necessary to set further conditional execution of code downstream. Selective retention also allows for better absorption of latency. An essential subset of registers can be restored to let normal operations proceed, while the rest are either streamed in or restored at a later time.

In all applications of selective retention, there is quite a bit of extra work for the verification engineer—obviously, such fine control implies that there is ample scope for bugs to creep in! To begin with, let's examine a relatively simple selective retention as shown below in Figure 4-3. The power domain has two sets of registers, A and B. Registers A are saved/restored by Save_A and Restore_A. Registers B are saved/restored by Save_B and Restore_B.

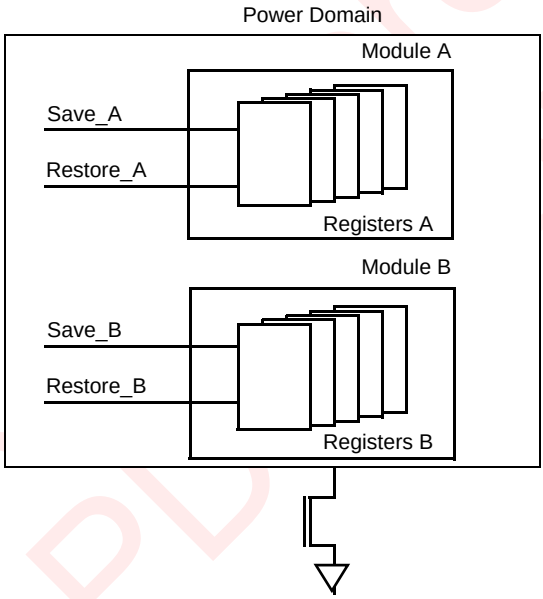


Figure 4-3. Selective Retention

Once such a partition is made, we can impose a temporal sequence as in the following pseudocode:

Go to Shutdown —

Save Registers_A;

Clock_Gating for Registers A;

Run some operations;
Save Registers_B;
Clock_Gating Registers B;
Go to Shutdown; (power gating enabled)
// time..time..
Wakeup power domain;
Ungate clock to Registers A;
Restore A;
Run some operations/checks;
Ungate clock to Registers B;
Restore B;

So, what could go wrong? First, we must recognize that this is a variant of the partial retention problem. Are the registers appropriately partitioned into the correct retention subsets? In other words, for each register in Registers A and B, how do we determine that the partitioning is appropriate? What tests would be able to establish this? Naturally, the same problem morphs at the netlist level into whether each register is connected to the appropriate save/restore signal physically.

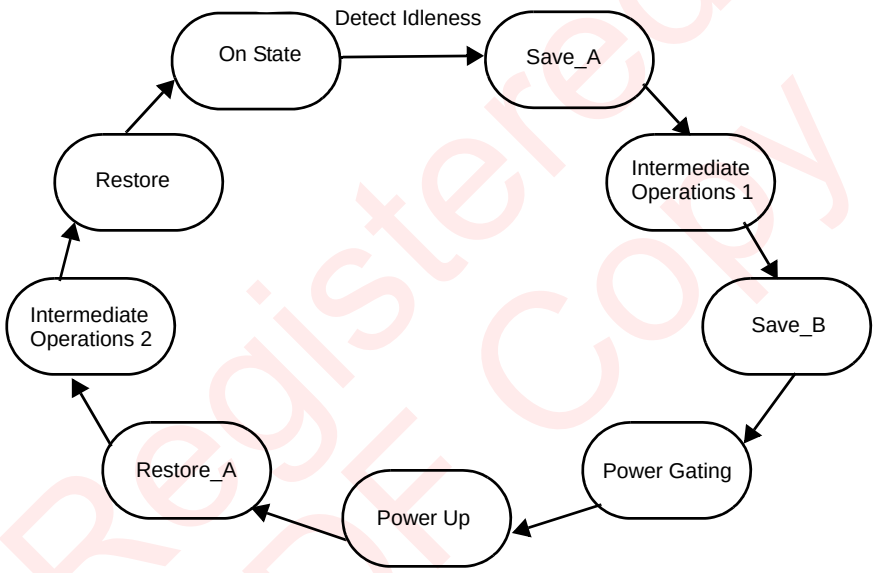
Rule 4.1 — Partitioning of selective retention must be verified by appropriate register tests.

Further, we need to establish that the control signals are indeed exercised in the appropriate order and that the functionality of the device after wakeup is unhindered by the restore sequence. Perhaps more subtle is that operations inserted between the subset saves/restores (between A and B in this case) do not cause problems. Some of these aspects, for example, that a processor can restore state arbitrarily and continue along its execution path for billions of cycles can be quite hard to prove. One must really seek to create smart tests to “poke” at the contents of the registers after restore to ensure that functionality is indeed maintained.Selective retention must be tested at least by a walk through of the desired control sequence.

Recommendation 4.2 — Post-restore coverage must be measured as the number of registers used in operations after restore.

Naturally, the above discussion leads to an examination of what kind of coverage metrics are needed. At the least, a walk through of the finite state machine shown below in Figure 4-4 is needed. However, the real test of such a scheme is in testing the save/restore mechanism under various contexts, prior to shutdown and subsequent to

wakeup. In that sense, this problem is the same as the generic retention scheme. What makes it more complex is the possibilities that could occur in the window between the saves/restores. Test plans must therefore aim at potentially disruptive events in this critical window. One of the subtle, but important aspects of this window is to prove that in the window between the selective save operations, there is no pertinent change to the registers already saved. Likewise, on the selective restore side, there must be no dependence on the registers yet to be restored.



Intermediate Operations 1: *Test for disruption*— test for changes to saved set A.
Intermediate Operations 2: Test for dependencies on unrestored B.

Figure 4-4. Conceptual State Machine for Selective Retention

Rule 4.3 — The intermediate period between saves/restores must be tested to check for changes to saved registers and dependencies on unrestored registers.

Last but not least, is the problem of formal verification. Connections to selective retention registers must be proved to be equivalent between RTL and netlist versions. In this context, it is pertinent to discuss coding practices—the risk of human error is greatly magnified by partitioning selective retention blocks within a module of logical hierarchy. Power intent files reside on the side: in the process of identifying and

creating the power intent files, human error is as likely to be a cause of problems as anything else. Avoidance is the best strategy here.

Recommendation 4.4 — Selective retention must be applied at the module level; partitions within a module must be avoided.

4.3.2 PARTIAL STATE RETENTION

In the preceding discussion regarding the low-level verification of retention protocols, we intentionally ignored a few aspects of architecting the retention scheme itself.

- Will the device work seamlessly once the context is restored? What variations in hardware and software behavior are possible?
- Is the set of registers being retained comprehensive enough? Is it optimal enough?

Regarding the first aspect, one can reasonably conclude that it is the target of directed verification to cover all the known modes of operations and eventualities in terms of coverage**.**

However, the latter is trickier. Given the high cost of retention elements in area, architects try hard to retain minimal context, which has led to the birth of a concept called partial state retention. Partial state retention is similar to but not the same as the selective retention that we discussed earlier. Partial state retention sounds attractive since the area cost of retention can be scaled down in proportion to the state being retained. Because every retention register contributes additional leakage power, partial state retention should result in lower standby power, and the high-fan-out buffering of retention controls should also be reduced. However, partial state retention is more complex to design and verify than full-state retention.

Taking the case of a microprocessor core, there typically are certain states (such as register banks, processor status flags, and mode information) that are visible to the programmer. This must be preserved from the software perspective by any hardware state retention scheme. There is also a specific micro-architecture state (such as pre-fetch buffering or branch predictors), which provides more efficient dynamic execution behavior at the expense of higher leakage when halted. If this hardware/software accelerator state is not saved, then the micro-architected hardware reload costs in terms of energy and time can be critical in area and power-sensitive designs.

Any state that is not retained but power-gated will have an arbitrary state when re-powered, and will typically have to be re-initialized explicitly. The verification state-space grows massively because of the interaction between retained and non-retained

states. This impacts not only the RTL view, but also implementation optimizations such as clock gating, which are built on the premise that state is globally persistent.

For example, the all-register state that is factored into the cone of logic of a further control state term needs to be analyzed to ensure that it is deadlock-free or non-state-corrupting. This is typically not viable for complex generic designs of any real complexity.

The reader may notice that there are many parallels between selective retention and partial retention. In the latter, the selectiveness is really a partition between a reset domain and a restore domain after wakeup from shutdown.

It is desirable to ensure that the retention aspects of a design do not break standard testability and ATPG flows. This involves ensuring that technology-dependent save and restore functionality is transparent to standard logic test. To facilitate this, the save and restore mechanisms must be made controllable in test-mode in a similar way as for clocks and resets.

It is desirable that the retention controls are externally controllable in test mode. This is in the same way that clocks and resets are ideally multiplexed from external pins, and test-mode multiplex controls are coded into the top-level SoC. If external pins are not available, then test-mode overrides (multiplexers) need to be explicitly coded in the RTL design. This coding needs to force the retention controls to an inactive state so that the standard scan tests will run properly. Functional test vectors are then required to exercise the retention control sequencing to ensure that it is correctly connected to the controlled subsystems.

In summary, retention registers may have different system control, however, the RTL design should be independent of the specific implementation of a targeted retention register. There are many variants of behavior that need to be understood at the RTL verification stage. Contrary to earlier practice, state retention needs to be aware of exact target library protocol fairly early in the verification process to ensure accurate verification.

4.4 Architectural Aspects of Retention and Verification

4.4.1 Resets and Initialization

To support partial retention schemes, it is the authors' views that explicit reset networks should be architected into the design for every retention domain. It then becomes possible to distinguish between power-on reset and restart conditions that are initiated explicitly from an SoC-level power control sequencer.

4.4.2 Verification State Space Explosion

Verification of systems with partially retained states is always going to be a challenge. RTL and gate-level representations assume that state is persistent, or they provide explicit support through cyclic redundancy coding schemes. These coding schemes regenerate a consistent state, or they indicate error conditions and require system-level intervention or re-initialization. With partial retention schemes, it becomes a designer's responsibility to ensure that mechanisms to cleanly flush and restart functional units are specified and architected. This is particularly true when a pipeline operation over a number of cycles is concerned. For microprocessors, such functionality may well be designed in and needs to be harnessed by the state retention control sequencer. However, without detailed knowledge of the design, it is very difficult to determine whether a part of circuit can be independently reset without deadlocking or corrupting the retained state.

In summary, full state-retention is attractive from the verification perspective. Partial state retention requires detailed design knowledge, and extensive verification will be required to thoroughly validate that the retained state is not masked or corrupted for all legal retained state values and for re-initialized control implementations.

4.4.3 Interaction of Retention with Clock Gating

Clock gating is a technique that aims to reduce clock power by gating a latched enable term with the clock waveform. Inherently, clock gating terms have some form of transparent latch structure, which is unlikely to have retention support. In a full state retention design, all the state terms that are factored into a clock gate enable must re-evaluate to the same enable logic value. However, in a partial state retention design,

the full cone of fan-in logic state values has to be guaranteed to result in the correct or safe value, which enables or inhibits the first clock after the retained state is restored.

The clock gate insertion and optimization tools are based on a static analysis of the clock enable terms. For example, if the rising and falling edges of a clock are both used in a design, then it may be impossible to arrange a clock level. This leaves the latch enables open, which can re-evaluate the clock gating terms afresh after a restored state for one or another clock phase.In fact, the latter concern is true for full state retention designs—this is another good reason why single-edge clocking is strongly encouraged.

4.4.4 RECOMMENDATIONS

Recommendation 4.5 — Unless a design has been architected for partial state retention explicitly, then don't go there! Full state retention is strongly recommended for both implementation and verification flows.

Recommendation 4.6 — Retained and non-retained states should have explicit independent reset networks in the RTL design. This allows functional simulation testing before state retention implementation, and gives the verification tools clear visibility as to which reset terms factor into state registers unambiguously.

Recommendation 4.7 — Only use a single edge of the clock to ensure the clock gating latch state can always be re-evaluated correctly.

4.5 CONCLUSION

In conclusion, there is no simple single answer as to which approach to state retention is the most effective.

For each and every design, one must understand the following:

- The sleep/wake-up activity profile, which for processors is primarily understood from the operating system scheduler, queues and active device drivers.
- The technology node.
- Significant leakage power savings are not limited to the fast leaky processes—it also shows valuable savings in the low-leakage versions where gate leakage is a larger contribution.

- The thermal profile. The most effective leakage reduction technique at any point in time depends on the temperature of the die, which in turn depends on the dynamic behavior the on-chip functional subsystems (including unrelated neighboring IP blocks).
- The most suitable corners for optimizing the implementation. Timing always has to be signed off to guarantee meeting worst case conditions, but sleep states require optimization at realistic temperatures for a typical silicon process.
- The libraries and characterization views. Understanding that optimizing a design for the worst case leakage corner (highest voltage and temperature) is not necessarily going to give the best solution. The consumer will ultimately judge how good the standby life of a product is relative to competitive products.
- State retention is very important because third party software or long-term persistent state is involved. The cost/benefit analysis of which state retention approach to select is an issue for the system designer. Provided that suitable library IP and design views are available, the baseline RTL design should be amenable to overlaying state retention schemes so long as clean synthesis coding styles have been observed and total state retention is acceptable.
- Partial state retention may appear attractive from an area perspective. Typically, this will require more invasive work on the subsystem RTL in terms of hierarchical partitioning and explicit independent reset networks for architectural (retained) and re-initialized (non-retained) states, as well as a verification approach that can validate the state partitioning.
- It is useful to understand where state retention is not sensible or worthwhile. In systems that are primarily data-flow driven, such as graphics or DSP pipelines, and where the processing engine primarily generates outputs from memory-based data and coefficients, then optimizing these units for performance may be the best choice. These units can then simply be power gated to save leakage power when not in use. At power up, a standard power-on reset can be employed to return them to a functional state. However, from a system design perspective, this is actually a specific case of partial state retention. Some state has to be maintained at the system level to manage the power-gated sub-system.

Overall, retention architectures remain a challenge to the architecture selection process and verification of the selected scheme. In Chapter 5, “Multi-Voltage Testbench Architecture” and Chapter 6, “Multi-Voltage Verification”, we will discuss how to set up adequate test coverage for retention.

CHAPTER 5 MULTI-VOLTAGE TESTBENCH ARCHITECTURE

ABSTRACT

In this chapter, the formation or migration to a multi-voltage testbench is discussed. The various testbench components are also identified and discussed. This will cover coding guidelines, power intent and library modeling aspects as well. Overall preparation for the verification process is the focus of this chapter.

5.1 INTRODUCTION

In the following pages, we focus on the formulation of a testbench architecture for a multi-voltage design: especially on the methodology of migrating from a non-multi-voltage environment. The primary objective of the testbench is to have the infrastructure provide effective and comprehensive testing of the multi-voltage feature set.

In the process of setting up and/or migrating to a multi-voltage test setup, many issues such coding practices, modeling of various elements, file formats and others enter the picture and we will discuss these in detail.

In subsequent chapters we will cover the actual usage of the test setup: to generate the required coverage, assertions and related topics.

5.2 TESTBENCH STRUCTURE

The essential components of the power management control system are illustrated in Figure 5-1.

- PMU (power management unit): typically an RTL block or set of RTL blocks that may interact with software
- SoC functions: effectively “controllees” that are monitored and managed
- Block-level circuitry: level shifters, isolation devices and retention cells.
- Mixed-signal circuitry: power switches, voltage regulators (VR), battery, and other components.
- Asynchronous, mixed-signal sourced logic signals: power on reset (POR), and Non-Volatile Memory (NVM)/One Time Programmable (OTP) memories.
- Software: code that executes on the CPU(s) often performing the function of power management

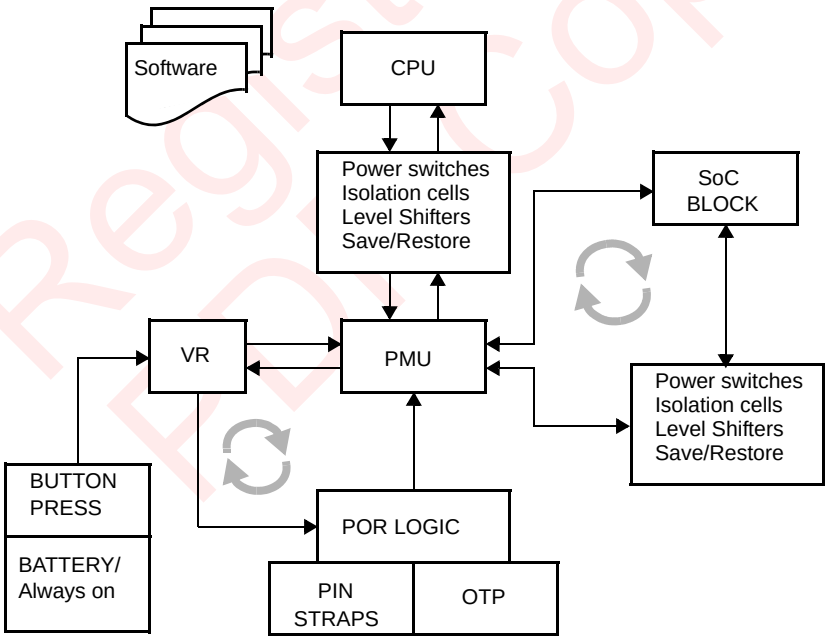


Figure 5-1. Structure—Typical Power Management System

Consequently, the testbench formed to verify such a system must have a corresponding structure. Figure 5-2 identifies the mapping of this structure to a verification system. Note that there may be regular non-power related testbench structures and DUT entities, all integrated into one setup.

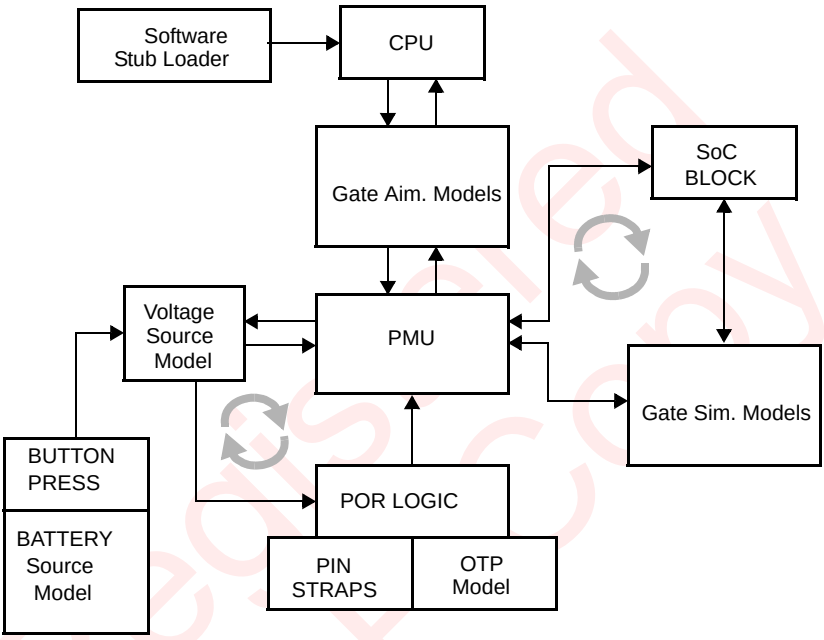


Figure 5-2. Typical Verification Testbench Structure for Power Management

5.3 TESTBENCH COMPONENTS

Let's take a deeper look into the various components illustrated in Figure 5-2.

5.3.1 SOFTWARE STUB LOADER

This is a testbench stimulus generator that mimics the fetching of instructions to the CPU. Most of the SoC testbenches today have such a harness. However, the difference with low-power testbenches is that the firmware that exercises power management must be appropriately covered.

Some of the typical routines that need to be tested are as follows:

- Boot and initialization routines, especially those that are part of system power up and power down
- Load prediction, detection and voltage scheduling routines
- Interrupt service routines (related to power management)
- Timer/status bit monitoring routines

It is not sufficient to merely have the firmware present and jammed in! For example, some tests force the execution of specific code stubs that turn power domains on and off or insert the device into an appropriate state. While this approach has some merit, it does not really exercise the power management control loop. It also suffers from a drawback, namely that the CPU that executes the software and the memory interface/storage may themselves be in some low-power or standby state. It is best to verify the overall control system: the hardware/software that triggers the transition to another state and the software that executes and monitors it.

Rule 5.1 — Power management software must be tested with the control loop that triggers it.

The other critical aspect of software testing is the verification of situations where conflicting resource requirements or power requirements occur. For example, a low battery situation might shut down the device or put it in standby state, but a phone call or chat message might come in that demands the user's attention. Such a situation, where the shutdown sequence is not complete, but conflicting demands abort the shutdown, is described briefly in the paper: *Challenges of Multi-Voltage verification on a complex low-power design, SNUG San Jose 2008* [6].

In designs with multiple processors, it is possible that software execution or hardware events in a subsystem trigger events in a different part of the system hierarchy. A common example of this would be plugging a digital multimedia device into a computer through a USB port. Such systems tend to be quite complex and can cause deadlock at the system level.

In general, it is difficult to measure meaningful coverage on power management software routines. However, we can use the registers used by software as meaningful coverage elements. A VMM application such as RAL can then be used to manage this. It has the added advantage of being able to generate random stimuli and stress tests to simulate various conditions. RAL also makes it easy to manage subsystems. The register space is already hierarchical in the DUT. It is also important to recognize branches in software and measure their coverage.

5.3.2 CPU

In most SoC verification processes, the CPU resides in the SoC itself. An RTL model is usually integrated into the DUT. Problems arise, however, when a C or other precompiled model is used as a plug-in to the simulation. This is usually done to improve simulation performance. However, such simulation models typically have two drawbacks. They are cycle accurate and hence, cannot easily respond to the asynchronous events in a power management sequence. Hence, they may not reflect the effects of a power management well. It is also difficult in current power specifications to partition inside a C model. Many CPU cores today shut down all but the cache, which is put in a low Vdd standby state. Such a partition or behavior cannot be easily reflected in the simulation model.

For example, imagine a control register in the CPU that is not appropriately reset after a power down and wakeup. The C model will not display this behavior and continue execution, whereas the RTL simulation will correctly stall based on the corrupted registers.

Rule 5.2 — Behavioral models not covered by multi-voltage semantics must be modified accordingly to respond to power management events.

Recommendation 5.3 — Use RTL models for components inside the DUT for power management tests.

5.3.3 Simulation Models

Cells like isolation elements, level shifters, power switches, and retention elements are not necessarily present in the RTL. Power intent standards, such as IEEE (P) 1801, allow users to specify these elements in side files. When the design undergoes simulation, appropriate semantics must be applied to ensure accuracy of results. While this is itself not a problem, the verification process needs to account for differences between such semantics and the actual behavior of the technology library. This is especially pronounced in the case of retention cells. Equally troubling is the behavior of level shifters and power switches, which are essentially mixed-signal circuits. Isolation cells are usually not an issue. However, some power intent formats may not define resets on latch-based isolation (LKGS). Isolation cells with esoteric features such as multiple enable controls and test mode overrides may also need better modeling and coverage at RTL simulation.

One of the most significant simulation models is for the voltage regulator (VR) and this can be used equally effectively for power switches. Both are voltage sources which respond to control by the PMU. Figure 5-3 shows a generic voltage source model, along with essential parameters listed below.

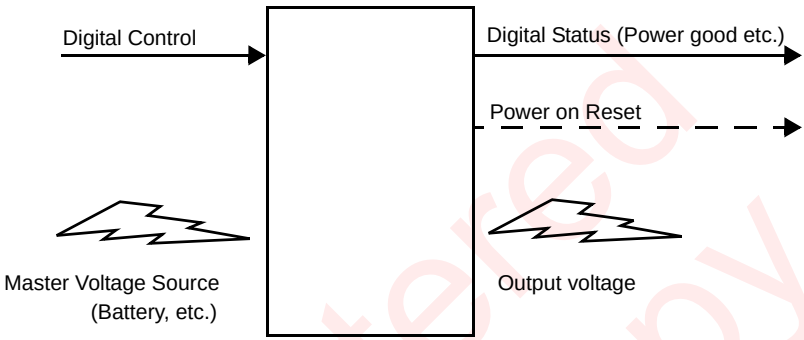


Figure 5-3. Generic Voltage Source Model

Note that both the master voltage source and the output voltage are real numbers, continuous in their variation. Depending on how the voltage source model is built and the languages used, the instantiation of this model varies. However, the key aspect to note is that digital control needs to be sourced to some signal not dependent on the output voltage. The digital status and power on reset need to be referenced to the output voltage.

The parameters/inputs needed for this model are the following:

- Conversion function from voltage in to output voltage
- Trigger point for in time and voltage for digital status signals
- Digital control interpretation and timing
- Simulation step in terms of voltage and time to produce a continuous variation effect.

This modeling of voltage sources is quite convenient. It allows you to hook up a master source like the battery or other master voltage source while representing cascading DC-DC regulators or power switches. It also allows for the response, both digital and analog, to be integrated into coverage metrics directly in response to the digital controls from the PMUs.

The conversion function of the regulator is quite important. One might visualize the conversion function as something simple:

```
// voltage out function
if (0.5<=Vmaster < 3.0) Vout = Vmaster * 0.5;
else if (3.0<=Vmaster<=3.6) Vout = 1.8;
else (message: Vmaster is out of range)
// simple function
```

However, more real life complexity can be added:

- Back annotating delays in voltage development from actual parameters
- Piece wise linear approximations of voltage development (or degradation) curves, esp. post layout
- Injecting droops/spikes in response to specific stress conditions; for e.g., the master voltage droops, say, 12% when a low battery indicator is received or Vout droops when a large domain is suddenly turned on
- Injecting random variations to reflect actual voltage tolerances, such as producing a random Vout between 1.62 and 1.98 V for 1.8V with a specified 10% tolerance as opposed to a constant 1.8V.

An interesting application of such voltage source models is to simulate their activation of non-volatile memory components such as configuration ROMs, laser fused bits or memory repair bits. Typically, these bits are valid as the Vout is developing and often sampled to wake up the chip or block in certain configurations. For example, a memory repair bit may activate one set of address decoder lines as opposed to another when programmed. The chip or block may wake up in a 16KB cache configuration instead of a 32KB configuration. Often these bits do not need special simulation models, unless the RTL instantiates special macros which then require a simulation model underneath. In most cases, it is sufficient to simulate the control system by appropriately activating the bits along the voltage development path (equivalent to re-initalizing these bits in simulation) and verifying that the chip/block indeed lands in the desired configuration.

In the online examples provided with this book for downloading, the readers can find examples of generic voltage source models' source code. These are intended as a starting point to illustrate how closed loop control of voltage sources is modeled.

5.4 Coding Guidelines

As can be expected, the impact of power management can be felt on how code is written as well, both for the DUT and testbench. This section contains coding issues and guidelines for low power designs. These are usually encountered when migrating either existing code or coding rules to low-power designs. They involve both testbench and DUT code.

5.4.1 X-Detection

Many testbenches are written to detect an X (logic level) on various critical signals and to give error messages or even at times, abruptly end the testcase with a fail status. This is in conflict with low-power design practices, which rely on X and Z corruption to reflect logic values in shutdown. Modifying such X-detection routines to account for shutdown is one of the most commonly seen changes of current coding practices in testbenches.

5.4.2 X-Propagation

RTL code is sometimes built for 2-state simulation and may not propagate X's correctly. If the simulation semantic corrupts a register, say on improper restore, the X logic value placed in the register may never be observed by the testcase. Further, X propagation may not occur in RTL, but gate-level simulation may yield a different result. Fortunately, most of the structures that arrest X-propagation can be detected by linting tools.

Recommendation 5.4 — Constructs that inhibit propagation of X logic values should be avoided in RTL.

Recommendation 5.5 — Corruption in simulation may not be observable in simulation results and assertion failures must be used to detect such situations.

5.4.3 Hardwired Constants

For starters, consider the ubiquitous 1'b1 and 1'b0 constants that are used all through the RTL. This was functional in the days when the entire chip (or at least the core) had a single supply voltage and a single ground. However in today's multi-voltage designs, there is no such thing as a single Vdd or single ground connection.

In addition, rails such as back bias nets or retention supplies may not even drive 1s and 0s. They may be arbitrary voltage values that not equal the Vdd/Vss value, which makes it questionable for them to be declared as supply1/0 nets.

Note that emerging standards, such as the IEEE (P) 1801, Unified Power Format (UPF) define power nets/rails and a type/value can be assigned to them. This alleviates some of the difficulty in analyzing the power nets and their connections, but the burden of avoiding hardwired constants is still with the RTL designer.

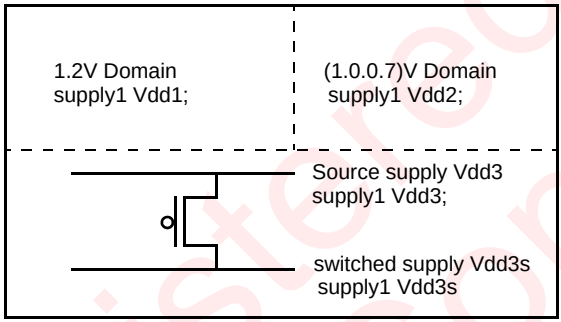


Figure 5-4. Multiple supply1 Nets

In most cases, the answer seems to connect to the local Vdd/Vss of the standard cell such as the supply1 declarations above. That may well work most of the time, especially in static multi-domain designs. However, in the case of power gated domains, where both the source Vdd and the switched Vdd are considered supply1 as in Figure 5-4, it would be legal to connect either Vdd to a 1'b1 connection, but not necessarily correct.

Additionally, consider the case where the constant is connected across from another domain in the design. *Placement and Routing* tools especially grapple with this issue, in their "flat" view of the design. The worst of these is the case in which the parent module is in one power domain and then instantiates constants in the port map of a module that is partitioned to another domain.

Note that synthesis and physical synthesis optimize constants away. This may no longer be valid for some multi-voltage designs. This situation has to be treated differently depending on whether the constant is local and subject to being turned off and whether there is any interaction with other domains.

One solution that will work is the creation of TIE_HI_VDDx or TIE_LO_VSSx type of nets. This will force the RTL designers to explicitly identify constants and think the implications through. This will also serve as an unambiguous guide to the verification and implementation tools.

To summarize this section, the following are a few basic rules for multi-voltage low-power designs: Rule 5.5 and Rule 5.6.

Rule 5.5 — Instead of hardwired constants, use TIE_HI_<NAME>or TIE_LO_<NAME> to explicitly identify the intended connection.

Rule 5.6 — Make sure that constants do not cross domain boundaries. Their behavior will need to be comprehensively analyzed across all sources: destination state combinations if they do. An additional level shifter may also be wastefully needed if this is done.

5.4.4 Expressions in Port Maps and Side Files

This refers to the practice of expressions contained in port maps. For example, consider the code contained in the following stub:

```
myreference Inst_myref (.input_pin(sigA && sigB)// ..
```

This is perfectly valid Verilog code, even if it is not good coding style. However, consider the situation where `Inst_myref` is partitioned into a different power domain than its parent. This leaves us with the mystery of where the expression in the port map belongs, how level shifting and/or isolation is to be applied. In conventional methodology, this will most likely be a synthesized gate at the parent level, However, consider a further version of the code stub above.

```
myreference Inst_myref (.input_pin(sigA && sigB),// ..

                        .output_pin1(sigA),

                        .output_pin2(sigB), //..
```

sigA and sigB are actually outputs of Inst_myref. The convention that the synthesized gate is placed in the parent is less justifiable in this case and outright complex to resolve.

Recommendation 5.7 — Avoid port map expressions at power domain boundaries. They are likely to cause improper specification and hence difficult to verify.

Likewise, expressions in power intent side files are extremely hazardous. They may not be synthesized correctly or verified/covered as needed.

5.4.5 Flip Flop at First Stage

There are designs in which the first stage of logic is a storage element. This used to help timing, but does not work very well if the power domain containing the logic is turned off and the sender of the data is still powered-on. In fact, it could be outright dangerous if the first stage of a flip-flop is a pass transistor.

Consider the case where the eventual target library has a pass transistor connection at its first stage (D input or scan input) as shown in Figure 5-5. When a live/on domain drives this connection when the domain with the first stage flip-flop is off, then there could be a sneak path for current, because the state of the gate is unknown. In rare and extreme cases, this can cause device breakdown, but the normal symptom is power wastage.

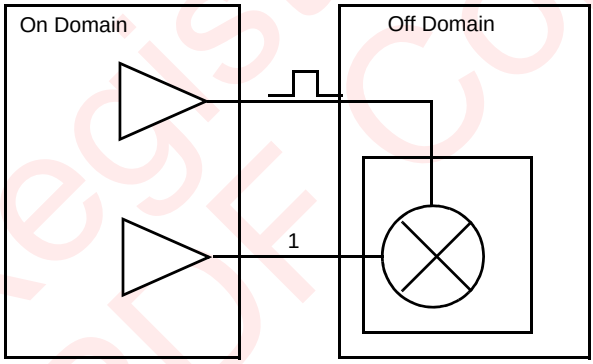


Figure 5-5. Sneak path hazard

The state of the gate of the pass transistor depends on the clock condition. If the clock to the domain is wiggling, it potentially connects to a lot of first stage CMOS gates. This indicates a lot of capacitance is wiggling even though the domain is off. If the external clock directly drives them, this could keep opening the pass transistors described in the earlier paragraph. This situation is a pure waste of power and must be avoided. The following are a few rules for coding IP blocks or hierarchical modules.

Rule 5.8 — Do not use first stage flip-flops if the domain is going to be turned off, unless input isolation is used.. Verification tools must ensure that this is the case.

Rule 5.9 — Verify that gate clocks are gated down to first stage inactive if the domain is going to be turned off. First stage inactive means that it must be either a cmos gate connection or the pass transistor it hooks up to must be closed.

Rule 5.10 — Verify that elements with first stage pass transistors are not used at the domain boundaries.

5.4.6 MONITORS/ASSERTIONS

It is customary to write testbench code to monitor various functions in the code. Similarly, assertions may be written either at the testbench level or deep in the code. Unfortunately, most of these assertions or monitors may have been written without planning for a multi-voltage architecture. The verification engineer is likely to encounter many tricky situations when migrating such a testbench/environment to the multi-voltage world.

First, consider monitor statements that directly access names nets hierarchically (which is bad coding to begin with). If the domain goes to shutdown, these nets may be assigned to Z or X, throwing off code written in the monitor.

Similarly, assertions may not factor in shutdown conditions. It is not as simple as factoring in X and Z values in the code. The reality is that a power state transition such as shutdown goes through a number of pre- and post-shutdown management events such as clock gating, multiple resets, and retention/restore sequences. The monitor or assertion set needs to stall or account for these transitionary states. In fact, NEW assertion and monitor code may be needed to factor in the power state tables.

Broadly speaking, the change in monitor/assertion code is that there may be code that is always monitoring the block; code that is off when the block is off and code that is on when the block is off and any further code to monitor transitionary states.

Extending this concept further, there may be force statements at the testbench level that make cross module references. These are especially done to set up pin strap options, device ID bits, etc. These force statements can conflict with any assignments the simulator is doing, especially in shutdown and retention situations. Even without low-power design, using cross module force statements should never be implemented. Low-power design adds further twists to the usage of this construct.

5.4.7 INITIALIZATION

Almost every testbench infrastructure utilizes initial blocks. Often, initial statements are used to load memories, set constants, and set finish times/stop times.

In the case that the initial block (along with a construct like readmem) is used for a block that is off by default and wakes up only later, any initializations must be deferred until the actual wake-up. Similarly, for a block that can be turned off, any memory initialization in it must be repeated after every power up. In addition, such an initialization needs to be sensitive to any handshake with power sensing and reset signals applied to the block. This handshake is often a source of bugs, so it is best to avoid such readmem based initializations. At least a few tests must cover the actual hardware-based initialization sequence.

On the contrary, registers such as non-volatile memory bits, laser fuse bits, and one time programmable bits need to NOT be corrupted by shutdown. Unfortunately, current HDLs do not provide for a simulation semantic to such bits in the first place. In the era of low power design, recognizing and supporting such bits is essential to accurate verification. Note that these bits do not wake up instantly. There is a point of activation along the rising power rail as it turns on. Also, the protocol often involves power-good and reset signals to latch these bits, adding further complexity to how this mechanism can be verified.

Extending this concept further, any PLI routines that form behavioral models or collect data, including debug/coverage routines, need to be aware of the shutdown conditions. For example, a CPU simulation model may be built in C and hidden inside a wrapper. A shutdown of the CPU may completely escape such a model. In fact, such a model may not just shut down accurately, it may also wake up or reset incorrectly as well.

5.4.8 STATE RETENTION

State Retention is altogether a new semantic that is being applied to sequential elements. Consider an sequential element such as a flip-flop being assigned to be a retention element: In this case, the flip-flop is probably coded in Verilog as an always at the *posedge* or *negedge* of the clock construct. However, the intended behavior is that when the domain goes to shutdown, there are additional save/restore signals hooked up to the actual sequential element that retain and restore the value of the bit.

There are numerous implementations of retention elements available. These change the protocol that is followed for save/restore and further impact the behavior of clock and reset (and scan in some cases). The same RTL may have to be simulated differently, depending on the actual behavior of the element being used in the context of instantiation. For example, a reset may clear the output of the flip-flop but not the retention element. Also, there may be a special reset pin needed to flush the retention element itself.

Another complexity is that the original RTL does not have save/restore pins instantiated locally in the first place. This implies that such a “connection” is done by the power intent file on the side. While this is extremely convenient and useful for the overall flow, RTL and gate-level simulation results may vary, based on how the save/restore signals are connected in the netlist.

5.4.9 SYNCHRONIZERS

It is common practice to synchronize asynchronous signals while crossing domains. Power management control loops, however, involve many asynchronous signals whose state is relevant to the PMU. While synchronizers can still be used, it must be recognized that there may be additional isolation latches in the path, which makes the synchronizers somewhat redundant. Furthermore, the design may enter a deadlock state by gating the clock to the synchronizers, while waiting for a wake-up event, which never makes it past the gated synchronizers.

5.4.10 PROTECTION CELL NAMING

Often, isolation cells are simply AND/OR gates or their inverted versions. However, it becomes difficult to distinguish between regular logic and isolation cells, especially to detect redundant isolation insertions. Signals at domain boundaries have less ambiguity with respect to isolation gates. Not detecting redundant gates can be quite dangerous functionally. Hence, it is best to have special isolation cells or create wrappers around the basic gates in case they are used for isolation. The wrappers enable static detection of redundant gates and form easily identifiable coverage points.

5.4.11 Activation of shutdown code

This is not exactly a coding guideline but a language semantic of which to be wary. Imagine a stub of Verilog code as shown below, which implements a combinatorial equation. If the `always` block resides in a shutdown domain and signal `inputsig` is sourced from an On domain, simulation results can be inaccurate. Consider the situation when `inputsig` changes from 0 to 1 when the block containing the expression is in shutdown state. A value of logic X is driven on to `outputsig` for the shutdown period, but once shutdown is removed and the block is woken up, the normal re-evaluation of logic is not defined in traditional HDLs. The issue can be worked around by either simulator support and/or an assertion indicating an input toggle while in shutdown mode.

```
always @(xinputsig)
youtputsig = xinputsig;
```

Similarly, Verilog does not natively allow a mechanism to code the complete behavior of an asynchronous reset. A reset operation can only be triggered on an edge on the reset signal. This causes a problem in a the power management scheme, which asserts reset before powering up an Off block so the block wakes up in a reset functional state. The edge of reset is masked as the logic is in Off state. On wakeup, the reset is already low but the Verilog description does not initiate a re-evaluation of the registers in that domain. Avoiding usage of asynchronous reset is not possible but users must ensure that their low-power verification solution handles asynchronous reset properly.

5.5 Library Modeling for Low Power

As one can imagine, traditional representations for libraries will need to be updated for low-power designs. These updates are quite intertwined with both implementation and verification. Hence, both sets of tools need to use the same information consistently across the entire design flow.

We can primarily identify two major areas where libraries need to change. One is the addition of power management cells such as isolation cells and level shifters into the library. The second is the modification of existing standard cells to accommodate the fact that designs now have multiple voltage rails.

5.5.1 Power Management Cells

Cells like isolation cells look deceptively like AND/OR gates or other standard logic. Level shifters may be misconstrued as yet another buffer. However, the design and overhead of these cells tends to be quite different. And then there are elements such as power switches which are entirely new. Not only do these cells need to be added, there needs to be a suitable identification of these cells in the library files. For example, the isolation cell may have an “is_isolation” attribute or similar to be set to true.

Further, the pins of such cells may need special identification. Imagine for example that the isolation cell has one of its inputs protected with a high Vt implant to better withstand the fluctuations of a floating input from the off island. This implies that the data to be connected must hook up with this particular pin and not be swapped with the isolation enable, though one might be tempted to think of the cell as a logically symmetrical AND/OR gate. Such a cell property needs identification. Tools hence need to understand this property and check for its correct realization in the design.

Another example is when the isolation enable is inverted in the cell. Not only should we not connect the data input to this pin (thereby causing an un-isolated input internal to the cell), but also, this information must be factored into synthesis, static and verification.

Cells with multiple rails such as power switches, level shifters, charge pumps etc. need strong identification of source side and output side voltage rails. Logically both networks may be represented as “supply 1,” but there is an immense physical difference.

It is also typical to enclose a function attribute into traditional representations. Power management cells make this complex, because sometimes the function may be quite mixed-signal-like in its behavior. It must be recognized therefore, that certain cells may not have the right functional model. Further, their simulation models must be built with care. It is a matter of debate whether the power and ground rails of a cell must be included in the simulation models. While the answer seems to obviously be yes in the case of standard logic cells, it is less clear in the case of power management cells. This is because these functions, by design, are complex, and expected to vary with voltage dynamically. Further, existing simulation models are represented in the Verilog language, which is inadequate to represent mixed-signal behavior. At this time, various standards have been proposed to the address this issue, since the ability to write such functions is essential.

The dynamic voltage-dependent behavior of these cells leads to another issue: how the timing arcs are represented. There are new sets of timing arcs to be standardized in the first place. These need to be characterized for and exported in a standard library format.

While all of the prior discussion looks like it is primarily focused on static verification, consider the example earlier where the data pin has different properties compared to the isolation enable. In this case, any assertions being written to compare the relative timing of these signals must be applied at the appropriate pins—this can be quite a task in a large design!

Retention cells represent another conundrum: how are save/restore relationships to be modeled? Given the vast variance in retention schemes from one library element to another, how are these to be represented in a common format that yields to consistent representation? Library vendors today have begun exporting retention cells with many new attributes and functions, which must be checked by both static and dynamic verification.

5.5.2 Standard Logic Cells

It is not immediately obvious that standard logic elements like buffers and NAND gates need to be changed because the design is made with multiple voltage rails. Changes on this front are primarily on the representation—the cells always had power and ground rails hooked up to them.

One motivation to include the power rails to the cells in the representation is the fact that they need to be explicitly hooked up to one of many rail networks and that connection must be verified. Therefore, a verification process must independently infer power network connectivity from certain standard attributes. The introduction of back-bias is another such additional item: back bias pins come with additional routing overhead, apart from the fact that the pin itself is now additionally present in the cell representation.

The other factor is that delay and power characteristics of cells change with voltage. This means that multiple sets of library data may be needed and factored into the analysis of various design stages.

Simulation models need to now account for the fact that the voltage applied to the cell could change: at this time, most models account only for on/off behavior. It is up to various simulation tools to impose voltage dependent behavior on these cell models beyond on and off.

5.5.3 Custom Macros

Custom cells like I/O pads, memories and analog blocks pose some unique problems in multi-voltage flows. These cells typically get connected to multiple power rails, without the notion of a driving rail or primary supply. Within these cells, the rails may actually drive different domains or be non-functional reference rails, such as a reference analog power supply. Further, any digital inputs may need to be referenced to a certain power supply. Likewise, digital outputs may be referenced to a set of driving rails. Inputs may already be protected, i.e., contain level shifters and isolation cells in expectation of certain states. All in all, this is a problematic area with ad hoc representations at this time, but standards are emerging for pin-level attributes and simulation behavior.

In terms of simulation behavior, note that the generic voltage source model described earlier in this chapter can easily be extended to represent such cells. The key however, is the validity of such models; they must be made along with the design and export process of the macros.

With custom macros, an additional source of trouble are restrictions on their temporal behavior. Consider for example, a memory with a built-in back-bias mechanism. The external world merely asserts a logic 1 on a standby pin as shown in Figure 5-6 below.

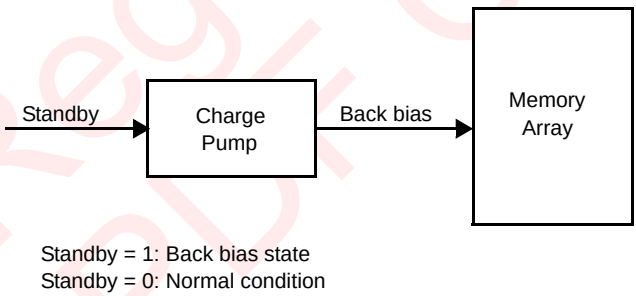


Figure 5-6. Memory Macro with Built-In Back Bias Features

Such a macro needs to reflect the true constraints of multi-voltage behavior in its integration, though such a cell is not connected to multiple rails. Such a cell might come with a variety of implementation restrictions as well, such as power grid requirements, standby pin timing arcs, and clock gating conditions. Unfortunately, such properties are mostly expressed ad-hoc. From a temporal perspective, we can expect to see a restriction on activity while the standby pin is asserted and for a

certain period after its de-assertion. An IP exporter must therefore ensure that all such restrictions are adequately reflected to the end user.

Rule 5.11 — A block of IP must be verified to be compliant to its original design properties, even if the integration is not in violation of top-level power intent.

For example, consider the memory in Figure 5-6. If the original design prohibits power gating of this block when standby is on, such a “state” restriction may not be reflected in the power intent (legal states) of the top level integration. However, a mere coverage of power states and transitions will not be adequate to verify this aspect of the design.

Recommendation 5.12 — An IP exporter must provide adequate assertions and coverage points to the end-user to verify low-power states and functionality.

5.6 CONCLUSION

In a nutshell, multi-voltage design brings about significant changes to the way libraries are represented modeled and used. New library attributes and standards [3], [4] already reflect these changes: the key however, is what we mentioned earlier. Consistent interpretation and comprehensive testing of these new attributes is essential for both design and verification processes.

As the user can see, formulating a multi-voltage test harness can be quite a transition. A well planned testbench architecture and migration is essential for the success of verification, which is the subject of Chapter 6, “Multi-Voltage Verification” and Chapter 7, “Dynamic Verification”.

CHAPTER 6 MULTI-VOLTAGE VERIFICATION

6.1 ABSTRACT

This chapter takes a detailed look at both static and dynamic verification. We cover static verification first as part of the flow and move onto dynamic verification. The flow at various design stages is also discussed.

6.2 INTRODUCTION

In the previous chapter, we looked at the preparation for verification at various levels of abstraction from testbench and RTL to post layout. In this chapter, we cover the basic verification process and flow, including both static and dynamic verification. Chapter 7, “Dynamic Verification” focuses deeper into the area of dynamic verification. While an immense amount of preparation and infrastructure is needed for dynamic verification, in power managed designs, a good amount of static verification is needed to make sure that bugs that can be detected without the effort of running vectors. So, the question is, what exactly are the goals of verifying power management?

This goal has gone through quite a bit of evolution, from a verification standpoint. Given the emergence of voltage-aware logic analysis, it is now possible for verification engineers to make sure that the DUT works as intended once plugged into a system. In Chapter 5, “Multi-Voltage Testbench Architecture”, all the effort was directed at ensuring that we have a test harness that actually mimics the system setup as well as the electrical effects brought about by voltage variations.

So, what exactly do we do to make sure that the DUT works as intended? First and foremost, we check that the design is indeed structurally connected correctly. This task involves numerous checks, but that can be done statically. Then, we proceed to verify that the power management unit functions as intended. However, that is only the first order of business. What we are really after is to ensure that the DUT works in all power states and can execute all power transitions and sequences as intended. This is a complex task. While it is quite difficult in current technology to prove that the DUT actually saves power, we have the additional burden of proving that the design *never* enters an electrically unsafe state. An IC that is functionally correct, but consumes excess power—even burns out at times, is not very useful.

Rule 6.1 — Verification must first focus on the electrical safety of the design.

Broadly speaking, as the SoC operates, we need to verify that it is functional in all states and can execute all the intended transitions and sequences. Furthermore, we need to determine if there are any unsafe electrical situations or excessive current consumption scenarios as early as possible. The verification engineer's challenge is to ensure that these goals translate into effective coverage metrics, directed and random tests, and assertions.

Although we tend to think of static and dynamic verification as two separate activities, it helps to think of them together when it comes to coverage of the problem at hand. Static verification can be used to profile the design, which can then be subject to further dynamic tests. This is especially true when static tests are used to detect temporal bugs as opposed to purely structural ones. Equally beneficial is a flow where dynamic verification results or assertions related to them are used in formal analysis to find errors.

However, most IC design groups today are organized into verification and implementation teams. Static verification is typically run by implementation teams, as new netlist generation is done at various points in the flow. This practice no longer works with multi-voltage verification. A team that performs dynamic verification alone may spend a lot of time detecting and debugging errors that could have been detected statically.

Recommendation 6.2 — Any errors that can be detected statically must be fixed before dynamic verification is performed. Dynamic verification must account for any errors that are not found by static checks.

Recommendation 6.2 is not a Rule, as much as it should be. This is for a practical reason: team organization may vary, and parallelism may be desired by some teams. However, the wisdom of launching a regression can be questioned when errors

detectable statically are present, unless of course, the dynamic tests have intelligently focused their attention on other problems.

6.3 STATIC VERIFICATION

The first thing to remember about static verification is that it is mainly for structural violations, but not necessarily at the gate level—this is a common misconception. Another interesting aspect of static verification is that it is not necessarily the power intent that has to be fixed all the time. Static verification is really a cross product of three entities: design structure, power intent and library elements. Irrespective of the level of abstraction, static verification takes in these three components. Appendix B lists possible static checks comprehensively. In this chapter, we focus on the process.

6.3.1 RTL STATIC VERIFICATION

Legacy flows involved the insertion of protection cells in RTL code by script or hand. In current power intent methodology, the protection cells are overlaid in a side file. In either case, the existing (or intention to insert) protection cells where needed must be verified to comply with “Rule 3.1a” on page 49, which states that all spatial crossings must be suitably protected.

This stage of the design process can also be used to verify that the power intent is complete and consistent. For example, a power intent file that does not partition certain design modules into power domains or has incorrect library element selection commands.

Most of the dynamic verification happens in RTL at this time. Hence, this stage of design is a great time to do formal analysis, especially in conjunction with simulation. Even without any such interaction, it is possible to analyze the architectural aspects of power intent to look for violations, such as not having an all off state or transitions that require too many rails to transition. In some commercial implementations of static tools, temporal ordering of islands can be used to derive legal state tables or vice versa.

In the era of extensive IP integration, there is another useful aspect of RTL static verification. Dependencies on control signals that emanate from the off or standby state sources in destinations that are On can be detected by static checks, though these bugs are temporal in nature. Clocks, resets, power gating controls, and isolation controls form the essential list to check. However, each DUT is unique and needs

specific signals to be checked. Consider, for example, Figure 6-1. There is a dependence on signal I/O EN to send a PWR EN to the Always On block, which in turn sends a wakeup signal to the On/Off block. However, this puts I/O EN in an unknown state when its source block is turned off. Note that inserting an isolation device does not necessarily solve the problem. It has to be isolated to the appropriate value and since a a bi-directional pad is shown, make sure there are no hazards with the direction set by isolation.

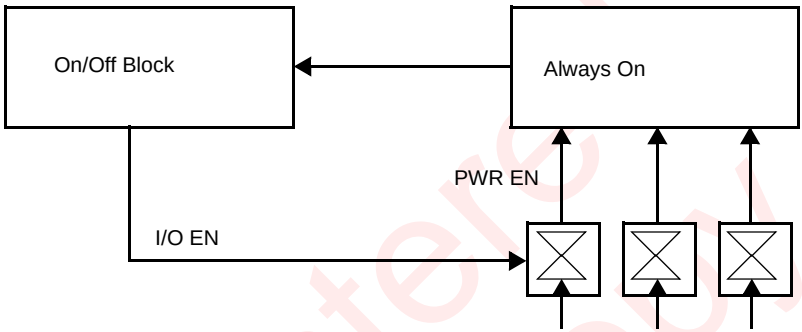


Figure 6-1. Control Dependency of Off Island

Not all control dependencies are this simple nor are all such dependencies direct functional errors. Even in the simple cases, a signal such as I/O EN in Figure 6-1 needs to be either profiled automatically by tools and/or by the verification engineer. Some of the commercial tools today are capable of automatically identifying these dependencies. However, the complexity and sometimes hidden or encrypted model of IP integration may elude automated analysis.

Sequential dependence of software register-based dependence is even more complex to detect. These can be identified by rigorous identification and testing of the power modes in which these signals are not valid. For example, perhaps a read is made to an address that resides in an off part of the DUT. The isolated values are returned, thereby returning an incorrect execution downstream. This situation is in fact, quite critical. There is no runtime probe that tells the DUT that a particular block is off. It entirely depends on the software remembering context and/or probing some other elements in hardware, such as power switch controls and other devices.

Rule 6.3 — The testbench must have assertions to protect against transactions with a block that is in off or standby mode.

Rule 6.3a — Software addressable registers known to be in on/off islands must have assertions to verify that access happens only when they are in the on state.

The reader will also notice that such a dependency is not necessarily an error. Perhaps logic exists to override the offending signal in the mode where its source is shutdown. However, for verification, this needs some tests or formal/property analysis.

Recommendation 6.4 — Identify critical control signals that originate in On/Off blocks and verify that there is no dependency on them to transition out of the state in which their source is off.

RTL static verification takes on more meaning as the number of islands and/or the size of the design grows, making the power intent specification and dynamic coverage much harder.

6.3.2 Gate Level Static Verification

In most design flows, the insertion of isolation gates, power switch structures etc. is done at the netlist stage. Hence, a whole range of static checks become essential as netlist structure is repeatedly transformed through synthesis, floor planning, power switch insertion, scan chain formation, clock tree synthesis, buffer tree insertion and timing fixes. Each iteration of the netlist transforms the design and its structure, thus requiring verification, at least statically if not dynamically after each transformation.

Gate-level static verification also depends on the availability of accurate library models. In the case of static verification, this information is present in the Liberty format, as we discussed in Chapter 5, “Multi-Voltage Testbench Architecture”. Gate level simulation also depends on the availability of cell level models that comprehend changes in voltage rails.

The other aspect of gate-level verification is the insertion of I/O cells such as pads. These typically have multiple domains within them and are sometimes part of the power network. Adding to the complexity, some cells come with built-in level shifters and isolation cells. Power intent formats vary widely in their range of expression, their ability to communicate hierarchically, and to impose rules on the overall integration. Overall, this is an area of extreme caution for the user. Even if complete automation was available, the amount of setup by the user is quite high.

In designs that rely on external power supplies, it becomes impossible to perform verification without testing the partitions of the I/O structure along the power rails and to account for any sequencing of power rails from the outside world. For such designs, I/O cells become part of essential coverage.

A quick glance at what static checks are needed can be found in Figure 6-2. A more exhaustive list is found in Appendix B.

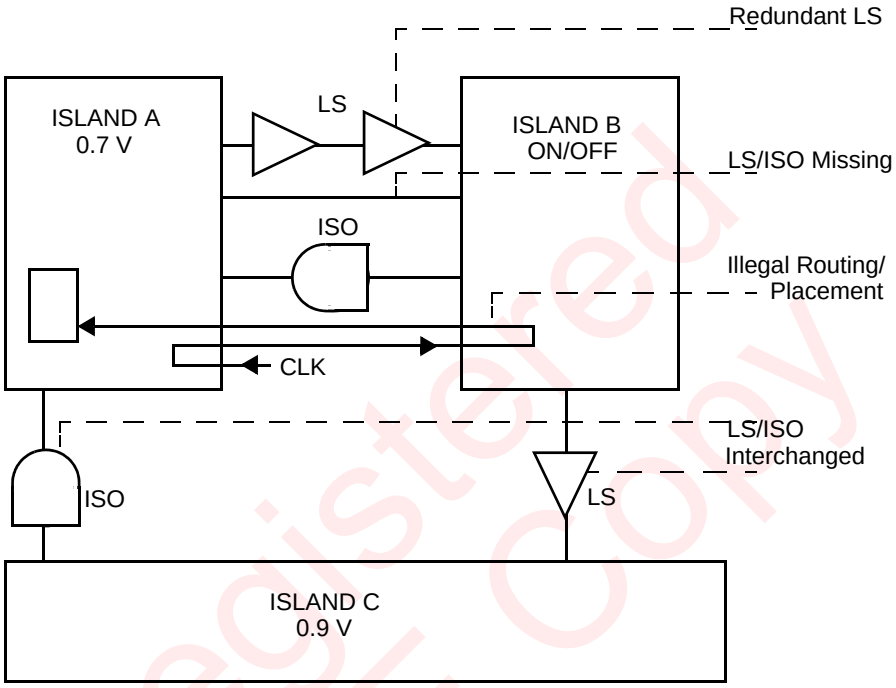


Figure 6-2. Static Checks at a Glance.

The list in Appendix B is exhaustive, but not necessarily universally applicable or complete. Specific design structures and IP require their own checks and need to be implemented for sign-off. However, it is interesting to note that all static structural checks emanate from a few simple requirements. A spatial crossing must be in an electrically safe state at all times, a power structure must be electrically viable at all times, and the design must not consume power in excess of what is needed.

It is increasingly common for users to adopt a methodology of reverse transformation from a power/ground connected netlist view to the power intent (spatial and partially temporal in nature). This exercise is helpful especially when hardened IP is integrated and enables robust checking in the back end.

So far, we have not broached the topic of power structure viability except in Chapter 1, “Introduction”. This brings us to the topic of a sign-off process for multi-voltage low power designs. As we mentioned in Chapter 1, power is as much a

delivery and reliability problem as it is a density and leakage problem. On the delivery front especially, the ability of the power structures to handle peak current loads and current fluctuations is essential. Such checks must be part of an overall sign-off process.

Last but not least, is the topic of equivalence. A reference design and implementation may be equal when all islands are on and equal, but not necessarily equivalent as voltage changes are applied. Consider Figure 6-3: the multiplier block is moved from *Domain 1* to an on/off domain in the implementation. An equivalence check that ignores the power down state of Domain 2 will not detect the error in the implementation. Even if one were to perform equivalence checks in the power down state, the isolation enable in the implementation tied to constant “1” or inactive level for formal verification will not detect the error. An active isolation value needs to be applied to detect the error.

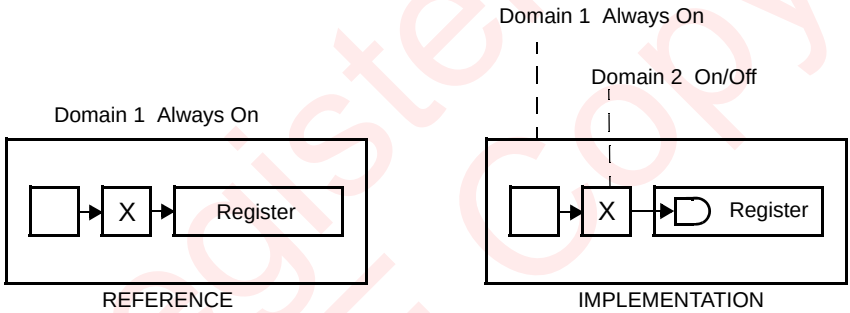


Figure 6-3. Spurious Domain Connection

Note—the design as implemented will pass a purely static electrical check on the spatial crossings. Hence, a comparison to the original design is needed to ensure that the original architecture is preserved. Overall, across all the multi-voltage design styles, equivalence issues are indeed a tricky problem. In recent times, commercial solutions to solve this important problem are increasingly available. We conclude this subsection with an essential sign-off rule.

Rule 6.5 — The implementation must be checked both as a stand alone as well as for equivalence to the original reference design across all power states.

6.4 DYNAMIC VERIFICATION

The first objective of dynamic verification is to exercise the power state table. Assuming that static verification yields a clean result, we can assume that in a steady multi-voltage state, there are no further obvious electrically hazardous conditions. Corner cases may well exist that need to be uncovered by dynamic verification. However, before we get there we have some basic functionality to verify.

For example, consider the situation from Figure 6-3, where it is not the implementation that has an error; rather, that a multiplier is incorrectly partitioned into a domain that can be turned off. However, the multiplier is potentially needed as a resource in one of the modes, which happens to unintentionally turn off the multiplier. If there are no tests that observe the output of the multiplier in this mode, then the error is unlikely to be detected. Note that the design could be completely correct structurally and yet encounter this error.

The reader might conclude that this is a trivial problem: the multiplier outputs would all be isolated values and the result, once used in the logic downstream, will definitely yield the error. However, consider the situation where the number of power domains is seven: each domain can be On/Off. That yields about 128 possible legal states. The partitioning is not at the multiplier level. It is possibly at the level of an IP block such as a processor core. The error in partitioning the multiplier is caught iff there are tests for the multiplier in every mode where the processor core is on. Even with a small number of power domains, the problem at this granularity becomes quite a monster. We need to arrive at exhaustive coverage in each mode of the design through random testing methods.

Recommendation 6.6 — Each power state needs to be tested exhaustively by covering all the major micro-architectural elements in that state.

Recommendation 6.7 — In each power state, coverage of all logic that is on should be as close to complete as possible.

The opposite situation is equally troublesome: verifying that resources not needed in a given mode are actually turned off. For example, consider the situation where the multiplier is in a domain that is on while the processor core is turned off. This is quite a hard problem to detect because the multiplier inputs are isolated in this state and hence controllability becomes practically impossible. However, Recommendation 6.7 will yield the offending block as an uncovered element and the debug process should result in an architectural analysis. Therein lies the lesson: power management failure conditions tend to be difficult to debug and may involve a re-examination of the architecture.

Moving further beyond power states, state transitions are the next target of verification. The ability of the PMU to sense the need for a state transition, assert and de-assert the appropriate control signals relevant to the transition, and signal a completion of transition and resumption of execution need to be well tested.

The verification of transitions is complicated by the following factors:

- The transition may be aborted and a safe return to the original or other state may be required [6].
- Electrically unsafe conditions such as level shifters being required where they would not be or level shifters going out of input/output voltage range may occur.
- The power integrity of the design may be stressed by rush currents or by the fact that many voltage rails are changing at the same time, causing a lot of noise on the power rails.

In terms of state transitions and sequences, a must have for verification coverage is the set of possible sequences for wakeup and shutdown to an all-off state, such as the bring up sequence. A number of complex tasks happen in this set of states and transitions often involve asynchronous and mixed-signal events. For example, the power on reset of the chip or parts of it may be triggered by the supply voltage passing a certain trip point. This in turn may latch configuration bits, device status bits and others before proceeding to full power up or aborting the power up. This is also a place where system-level deadlocks are likely. Hence, tests must be directed at this part of the power management state space, not just focus on functional states/ modes.

Hence, the verification of transitions needs to comply with the following rules, apart from the obvious one that all possible state transitions must be covered.

Rule 6.8 — Transitions must be tested for abort signals if applicable.

Rule 6.9 — Assertions to guard against multiple rail changes at the same time must be present.

Recommendation 6.10 — Level shifter range violations must be guarded against by writing appropriate assertions at each spatial crossing across voltage domain boundaries.

Transitions can also occur for multiple reasons and perhaps conflict with each other. For example, an incoming phone call may direct the CPU to operate at 1.2V whereas a camera “click” in progress may be operating the CPU at 1.4V.

Rule 6.10 — Power states must be tested for conflicting transition inputs and priority of transitions must be resolved as architecturally intended.

However, in the case of the above example with a phone call there is a camera conflict. Consider the situation in which the phone call is indeed the priority, but the camera data is not discarded. The process is merely sent into the background. This implies that once the phone call is done, there is a need to restore the camera mode and execute accordingly. This now brings us to sequences. Even a small design with few power states, if likely to see numerous sequences, particularly with logic elements saving relevant context in various states.

Thorough coverage of sequences is not possible without resorting to some random stimulus techniques. Further, sequences should represent real life usage. In modern SoCs, coverage of sequences is best accomplished by applying as many software tests as possible. In addition, one must apply random stimuli such as interrupts, critical failure conditions, timer/counter triggers to ensure that in every case, the DUT stays within defined power states and without deadlock conditions.

As the number of power domains and states increases, this becomes an exponentially increasing coverage space. Such designs do not lend themselves to ordinary coverage metrics. One must necessarily think of the design as a hierarchy of connected systems and target verification at the interaction between such connected systems and within each subsystem. We must also try to prune the coverage required by being selective about which metrics are relevant to verification.

6.4.1 Impact of Design Style—Architecture and Micro-Architecture

One of the unique aspects of power management is that the design style chosen makes a significant impact on the bug types and hence the verification strategy. For example, a power gated design is susceptible to isolation errors, rush current/voltage scheduling errors or reset errors, but does not have level shifting or memory corruption issues. Likewise, a design that uses power gated state retention is unlikely to have a logic conversion error.

Apart from the generics of walking through the state table, transitions and sequences, we need to focus verification on the micro-architectural implementation of control as well. If we turned our attention to the design structure, islands, power gating enables, voltage ID codes, retention controls, isolation enables, and charge pumps, we can ask ourselves what the coverage metrics look like from this point of view. In general, it is

not sufficient to walk through the power state table and transitions. In fact, this becomes prohibitive as the number of islands and hence state combinations grows.

However, when one focuses on the design elements, a more manageable coverage metric emerges. We still need to watch for illegal states and transitions, but it would help to know how many times an island has been shutdown. Sometimes, this might yield a vector that exercises all power states, but never shuts down a particular island, thereby indicating an error in the state table or the partition.

Rule 6.11 — Coverage must be measured on design elements such as islands and power management controls, apart from the power state table, transitions and sequences.

Focusing further on control, one must take into consideration special effects such as selective retention, staggered power switch control, split isolation structures, and latch-based isolation, and test them at the appropriate points in the power management sequence. For example, consider the example of selective isolation enable as shown in Figure 6-4. It is important not only to cover Iso_en1 and Iso_en2, but also ensure that any sequence between them is honored. It is important also to test the interval between the isolation enables and disables to see if the design's functionality is maintained.

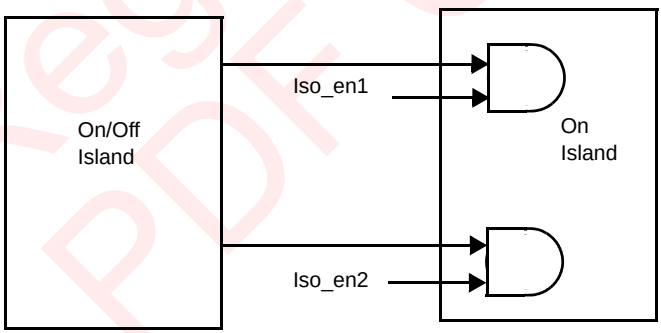


Figure 6-4. Selective Isolation

Overall, micro-architectural coverage, while essential, is quite design specific. The test plan hence needs to have a dedicated section aimed at these design elements. Focusing on the design elements yields another extremely beneficial result. Critical control signals that are essential to the power management scheme will be defined

rigorously in the process, which can be statically verified as described in “Recommendation 6.6” on page 114.

6.5 HIERARCHICAL POWER MANAGEMENT

In most current systems, the design is already organized as a collection of resources and subsystems, such as the memory subsystem, the video subsystem, the graphics rendering engine, and the analog subsystem. The implication here is that each of these functions presents a view (say a control interface) of power management to its master controller, which in turn presents a view of functions and interface to its master controller. This interface need not necessarily be a power management view, it works well for any functionality as well, such as a DMA transfer.

Thankfully, while most of today’s systems organize themselves in such a fashion for reasons other than power management, the boundaries of such subsystems form natural boundaries for voltage control.

We can now view such a system as a bunch of finite state machines linked to one another with a defined pecking order. This order is quite relevant. We cannot have a situation where the master controller of a device/subsystem is off, but the device is itself in the on state.

Some readers may be thinking, “*Wait! My keyboard subsystem recognizes the power button or lid opening on my laptop and wakes up the entire system. So, it is not an error for a device to be on while its master is off*”. While this is an excellent observation of most real life systems, the subtlety in the master-slave relationship is that the power rails may follow a different hierarchy. It is a hierarchy of voltage rails.

In the above example, while the input devices as a class are controlled by some master for their functionality, a power domain view of the keyboard may show the power and lid inputs being partitioned into a separate always on domain. Further vexing the reader perhaps, is the fact that the functionality of the power button and lid can be configured by the user, once the system is up and running, for power management options and one option may appear to change the hierarchy. For example: the lid input maybe programmed not to cause any state change. Note that this may not necessarily change the hierarchy of power domains; it may be changing the system's response to an input.

This brings us to an important rule of hierarchical power management.

Rule 6.12 — The default, unconfigured functionality of configurable power management hierarchy must be tested.

Rule 6.12a — The programmed/configured functionality of configurable power management hierarchy must be tested.

One of the hidden aspects of hierarchical power management is that once the rails / domains are ordered as masters/slaves in a tree, the state table of the system is quite amenable to derivation. We can further derive transitive relationships or disjoint properties between the power domains, thereby reducing the need to cover modes that may not be realistic.

An excellent example of hierarchical power management can be found in[16] with additional background in the following references: [3], [16], [30].

We now return to the basic aspect of verification: writing tests and measuring coverage. This is the topic of the next chapter.

CHAPTER 7 DYNAMIC VERIFICATION

ABSTRACT

This chapter focuses on Dynamic verification, coverage and assertions. The various aspects of power intent and power management state space are discussed from a coverage point of view.

7.1 INTRODUCTION

Static verification techniques are very powerful because they are deterministic and exhaustive. However, they are limited in scope because of the inherent complexity and heterogeneousness of a complete power management system. Today, as linting and formal verification cannot find all of the functional bugs in your design, it is not possible to guarantee the correctness of the power management aspect of your design using only static verification techniques.

Fortunately, the same technique used to augment the static functional verification of your design can be used to augment the dynamic verification of the power features of your design. Using the appropriate models and verification environments, simulation can be used to verify those power intent aspects of your design that remain unverifiable under static techniques.

Unfortunately, the simulation techniques for verifying power intent suffer the same limitations as the simulation techniques for verifying functionality; you must exercise a bug and observe its effect to identify it. This chapter presents techniques and

guidelines for maximizing the likelihood that bugs, unidentifiable through static means, will be uncovered.

SystemVerilog simulators are, by default, digital simulators. They only know four logic states: zero, one, high-impedance, and unknown. Furthermore, they are only concerned with data signals, not supply voltages. In power-management features, functional errors are often related to the analog nature of the power supplies and external control signals. It is therefore necessary to use a power-aware simulator, which can accurately represent and simulate these effects, to be able to identify many of the bugs that are targeted by the methodology described in this chapter.

7.2 VERIFICATION PLANNING

This section describes the additional planning you must do to verify the power-saving functionality of your design. As with verifying any other feature, the verification of the power-saving features must be planned.

Recommendation 7-1 — Power domains should be verified independently first.

It is the combination of multiple power states of independent power domains that creates the functional complexity of a low-power design.

If the functional correctness of a power domain can be verified on its own, then only system level verification is required. In other words, verify that the power domain can be put in its various states; the functional correctness will be implied This method is much easier than trying to verify the functional correctness of multiple independent power domains at the system level. Fortunately, power domains often correspond to design units that are functionally verified independently before being combined into a final design.

Often, a power domain will be composed of a design unit whose original architecture did not factor in low power related structures or effects in time. Such a design needs to be retro-fitted with power switches, protection circuits, retention cells etc., and then verified with these elements present. In both cases, a stand-alone verification environment for the all-ON functionality should (eventually) exist. That verification environment can be leveraged and then augmented to verify the power-saving capabilities of the power domain.

If a stand-alone verification environment is not available for individual power domains, it would not be viable to individually exercise a power domain in a

multiple-domain design. Domain ordering rules may require that some other domain be turned off before turning off the domain of interest. Verifying that the functionality of the targeted power domain responds as expected in the various power states may be difficult if its interfaces are not externally visible, or if the interfaces cannot be observed or driven through the surrounding blocks.

Rule 7.2 — The All-ON functionality of the design should be verified first.

The verification of the power intent is to ensure that the design operates correctly while portions (or all) of it are powered down and then powered up. It is best to ensure that the basic functionality of the design is correct with none of the power-saving features enabled. This method will facilitate identifying the causes of failures during the power-related verification; any failures will be caused by the power management features, not the functionality of the design.

Obviously, several power-management bugs may be found and fixed in the process of verifying the all-ON functionality of the design. But these bugs all lie on the path to the “all ON” power state. It is necessary to plan the verification of the power states that have not been visited, and the power state transitions that have not been taken.

7.2.1 Response Checking

Traditionally, the response checking mechanism has assumed that the design was continuously performing all of its functions. However, in a design with power-saving features, some of that functionality may be turned off or suspended. The response checking mechanism, most frequently a scoreboard, must take these additional modes into account.

Rule 7.3 — A power-aware simulator shall be used to verify the correctness of a powered-down domain.

A powered-down domain is rendered inactive through the removal of its power source. Power sources are not part of the functional description of a RTL design. Therefore, there is no way to model the loss of state (if retention is not used) and activity created by the power down.

A power-aware simulator will invalidate the portion of the device under test (DUT) that corresponds to the powered-down island, and prevent any simulation from occurring. The lack of response must still be verified to make sure the powered-down domain includes all of the target functionality.

Rule 7.4 — The response-checking mechanism shall be notified of the start and end of any power state changes.

The response-checking mechanism can use these notifications to appropriately modify its expected response according to the new state of the design. It is important that both the start and end of a transition are indicated. The process of transitioning from one stead-state mode to another is not instantaneous; it usually involves a lengthy firmware-driven protocol. The transition process will affect the expected response from the time it starts to the time is completed.

If using the Power State Manager defined in Appendix A, the notification service interface is found in the *vmm_lp_design::notify* class property can be used to that effect.

Recommendation 7.5 — The exact timing of the effect of a power state transition should not be predicted.

The process of transitioning a design from one steady-state mode to another may involve transitioning many power domains from one state to another. In the presence of in-process transaction, it will be difficult to accurately identify which transaction will be aborted and which one will be successfully completed.

For example, the design in Figure 7-1 shows three domains pipelined to process transactions in sequence. A typical scoreboard for such a design would be to predict the final responses, and to queue them in expected order of arrival, ready to be compared against the observed response, when it is observed. In contrast, the design pipelines its functionality. Should a power-mode transition power down domain “A” first, it will be difficult to identify which transactions were currently being processed by that domain, as well as which transactions were currently being processed by the other domains. It will no longer produce a response.

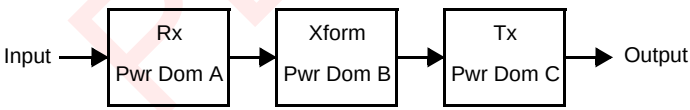


Figure 7-1. Pipelined Power Domains

It will be easier to indicate to the scoreboard that all of the currently expected responses *might* be lost, as illustrated in Figure 7-2. The comparison function would then continue to successfully compare observed responses against expectations for

those transactions that turn out not to have been affected by the transition. Further stimuli should be ignored while the design is in a power-down mode.

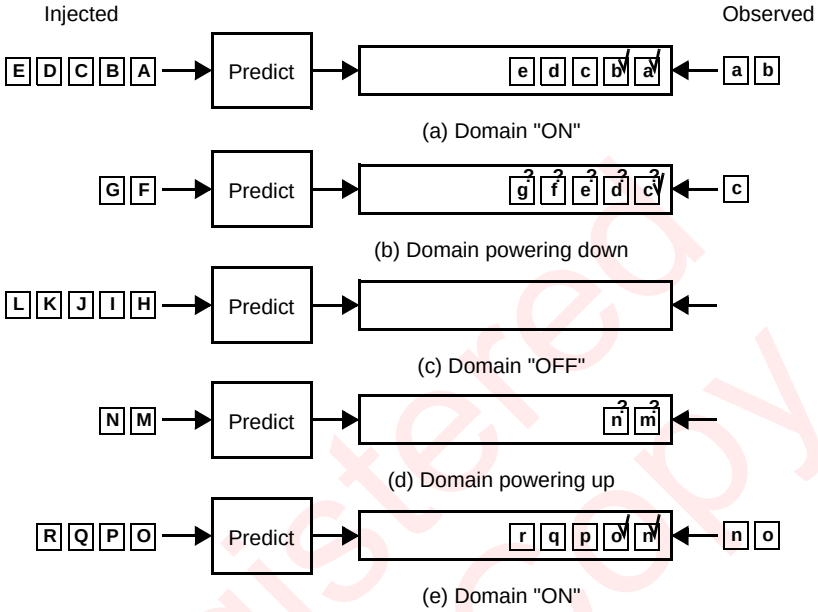


Figure 7-2. “Fuzzy” Scoreboard

The process is reversed for a power-mode transition in the opposite direction. As soon as the start of the transition is indicated, stimulus is once again accepted by the scoreboard, but continuously marked as “possibly” lost. Once the transition is indicated as complete, further expected responses are marked as “definitely” expected. During this transition process, the design will eventually start to produce responses at an unspecified point in the transaction flow. At that time, the comparison function will find that the expected response does not match the observed response. This is because the expected response corresponds to a transaction lost by the power-mode transition. The comparison function should then look ahead in the “possibly lost” expected response sequence for an expected response matching the observed response. Once found, all intermediate transactions can safely be assumed lost and all subsequent transactions must then be observed.

The *`vmm_sb_ds::expect_with_losses()`* method can be used to implement such a comparison function.

Rule 7.6 — Functional correctness shall be confirmed after each transition.

The functional consequences of a power state transition may not be immediately visible. The self-checking mechanism must indicate if the functional consequences have been observed and verified to be correct. A test case can then use this indication that it is safe to proceed. Otherwise, the absence of error messages may cause a lack of observable output with an absence of errors.

Rule 7.7 — The self-checking structure shall use a vmm-consensus instance to confirm observed correctness.

A power transition may affect several functional areas, each verified by a different portion of the self-checking structure.

Use a single `vmm_consensus` instance to have the entire self-checking structure report that it has positively confirmed functional correctness. This allows an arbitrary number of threads and checks to independently confirm the functional correctness.

This will also allow the composition of this self-checking structure with others that follow this guideline in a system-level verification environment.

7.2.2 External Controls Verification

The All-ON verification is concerned with verifying the end-to-end functionality of the design. To speed up the execution of the large number of required tests, the power-on-reset sequence is designed to meet the requirements of the simulator, not accurately modeling the physical behavior and timing of a power-on reset circuit.

The functionality of the design must be verified with a proper model of the power-on reset circuit. Chapter 5, “Multi-Voltage Testbench Architecture”provides guidelines for writing such a model, and how to write a suitable power-on reset test case.

It is also possible for the design to depend on the initial state of the simulation. For example, any *bit*-type or *real*-type variable is implicitly initialized to zero, not unknowns. It is necessary to ensure that the design is properly reset when the external power is gated and the external reset signal is asserted.

The reset functionality must be verified by applying the reset signal after the design has been functionally operating for a while. Then, ensure that it can be restarted into a functionally-correct state. Similarly, the effect of external power gating must be verified by removing then restoring the external power to the design (followed by a reset signal) after the design has been functionally operating for a while. Chapter 5,

"Multi-Voltage Testbench Architecture" provides guidelines on implementing such test cases.

The design should also be verified under clock-gating conditions. For each externally-controllable clock that could be gated, its clock source must be "gated" by holding it at a constant logic "0." This must be done after the design has been functionally operating for a while, and with valid data flowing through the "gated" clock domain. The clock must remain "gated" while valid stimulus is applied to the design. The clock must eventually be restored to its normal operation, with valid stimulus. This verifies that the clock domain will correctly ignore inputs when gated and resume normal processing, without data losses, once the clock signal resumes.

Note that verifying external power gating requires a power-aware simulator.

7.2.3 Power States

For each power domain, it is necessary to visit all of its power states and intermediate logical states, as well as all of the transitions between those states. The transitions through the power state space should be taken randomly to increase the probability that any problems related to the timing of a transition or the path taken through the power state space are uncovered. The VMM Power State Manager can be used to generate random power state transitions.

A coverage model should be use to confirm that all power and logical states, as well as all transitions have been taken. This coverage model may be automatically available in the power-aware simulator. For example, Synopsys's MVSIM can create a functional coverage model of the power and logical state machines, based on the description of those power and logical states in the power intent file.

Suggestion 7.8 — The power state functional coverage model may need to include power state paths.

It may be possible for the power state transitions to take different paths through the power state space. For example, there may be more than one way to reach a stand-by mode, or there may be different logical states that lead to the same power state.

In such a case, it may be important to ensure that the functionality of the design remains correct for all paths through the power state space. Path coverage may need to be included in the power state functional coverage model.

7.2.4 State Retention

If a power domain includes state retention, it is necessary to verify that sufficient state information is retained and properly restored.

Rule 7.9 — State retention power states shall be visited without going through a reset.

Visiting a state retention power state is not enough. It must be visited from a powered-on state and back to a powered-on state, without going through a power state where the state of the domain is reset. Otherwise, the functional correctness of the state retention could not be verified, as it would be indistinguishable from a simple power-up from reset transition.

Rule 7.10 — Every retention cell shall retain a non-reset value.

To ensure that the state of the domain is properly retained, it is important to verify that an active state being retained. The functional coverage model must ensure that each retention cell has been put in the retention state with a value different from its reset value. Otherwise, the retention state would be indistinguishable from a simple power-up from reset transition.

7.2.5 Dynamic Frequency Scaling

Rule 7.11 — The frequency of the scaled clock shall be verified.

The functional response of a domain subjected to dynamic frequency scaling does not change, only its relative performance. It would be challenging to measure the performance of a domain. Any performance-related error could only be reported after enough samples had been collected, and thus disconnected in time with the cause of the error. It is much easier to verify the functional correctness of the domain, regardless of its current clock frequency setting, and to measure the frequency of its clock. Any error will be reported immediately, close to the time and space of the cause of the error.

7.3 All-On Verification

The “All-On” functionality is the functionality of the design when all of its components are fully powered up. This is the functionality the design would have if it did not have any power-saving features. The verification methodology described in the *Verification Methodology Manual for SystemVerilog* [1] implicitly assumed an All-On design.

Rule 7.12 — The All-On verification environment shall be designed to support the subsequent power verification.

The verification of the power intent must leverage the basic functional verification environment. It can use the same stimulus and response checking mechanism to identify a functional failure caused by the power management. Power-related test cases should be seen as test cases requiring the introduction of power-management stimulus, as no different than other corner-case functional tests.

The *vmm_env* base class has been extended to help implement an All-On verification environment that can be leveraged for verifying the power-intent of the design. The following guidelines describe how these extensions should be used.

Rule 7.13 — The *vmm_env::rst_dut()* method shall not be extended.

The All-On verification environment needs a very simple power-on reset model that is designed to initialize the model of the DUT to its reset state. But power-related tests will need to provide a more accurate model of the power-on reset sequence. It must thus be possible to modify the power-up reset model in the power-related test. If the power-on reset sequence is implemented directly in this method, it will be impossible to supersede it as any subsequent attempt to extend it to modify it will have to call *super.rst_dut()*.

Rule 7.14 — The power-on reset sequence shall be modeled in an extension of the *vmm_env::power_on_reset()* method.

The *vmm_env::power_on_reset()* virtual method is invoked by the default implementation of the *vmm_env::rst_dut()* method. Any user-defined extension need not call *super.power_on_reset()*, thus making it possible to completely supersede the simple power-on reset sequence for the All-ON tests with a more complex one for the power-related tests.

Rule 7.15 — An active external hardware reset sequence shall be modeled in an extension of the *vmm_env::hw_reset()* method.

This hardware reset sequence may be used by functional tests to reset the DUT once it has been powered up. Unlike the power-on reset sequence, the clock signals are active and running and the reset signals are well-defined with clean edges.

Rule 7.16 — The DUT shall be brought into the All-ON state in an extension of the *vmm_env::power_up()* method.

This method is invoked at the beginning of the *vmm_env::cfg_dut()* step. It must be defined to perform whatever operation is necessary to bring the design in the

All-ON state. This may involve asserting clock enable signals, releasing reset signals and turning on power switches.

The objective of this method is to bring the design in the All-ON state as quickly as possible, to be ready to perform the functional verification of the design. Subsequent power-related tests are likely to further extend the method to put the design in a partially-ON state, or leave it in a dormant state altogether. This depends on the initial state of the DUT required by the test.

Recommendation 7.17 — If using a register abstraction layer, registers in different power domains should not be grouped together.

If host-accessible registers are located in different power domains, they may not be accessible until their respective power domain has been turned on. By grouping registers according to their power domain, it will be easier to verify the registers, one power domain at a time.

For example, when using the VMM Register Abstraction Layer, each power domain should be specified as a separate block. It is then possible to use the pre-defined block-level register verification routines to verify the registers in each domain, as each domain is turned on.

7.4 MODELS

The power management features included in the design must be appropriately modeled to determine their functional correctness. Some of these features can be modeled entirely as synthesizable RTL code, such as registers controlling power switches. Some must be carefully modeled to avoid introducing unexpected side effects. Other features must be modeled using augmented semantics to take into account the analog effects of voltage variations in power managed circuits.

The following section provides guidelines for modeling the power-related functionality in a design.

Rule 7.18 — Clock gating models shall not use delays nor non-blocking assignments.

The non-blocking assignment on clocked signals is the cornerstone of RTL modeling in SystemVerilog to accurately model the behavior of synchronous circuits. It relies on an infinitesimal delay between the edge of the clock, and updating the sampled signals to reliably resolve potential race conditions. However, this approach relies on having *all* of the clock signals occur before that infinitesimal delay. Should the

nonpolluting assignment—or worse yet, a real delay—be used to create those clock signals as shown in Example 7-1 below, race conditions will reappear.

Example 7-1. Improperly modeled clock gating

```
always @(posedge uclk or negedge rstn)
begin
   if (!rstn) clk_en <= 1'b0;
   else       clk_en <= ...
end

always @ (uclk or clk_en) clk <= uclk & clk_en;
```

When modeling glock gating circuits, continuous assignments, primitives or blocking assignments must be used, as shown in the following Example 7-2.

Example 7-2. Properly modeled clock gating

```
always @(posedge uclk or negedge rstn)
begin
   if (!rstn) clk_en <= 1'b0;
   else       clk_en <= ...
end

assign clk1 = uclk & clk_en;
and(clk2, uclk, clk_en);
always @ (uclk or clk_en) clk3 = uclk & clk_en;
```

Rule 7.19 — Asynchronous reset signals shall remain asserted for at least one active clock edge.

The RTL coding style for asynchronous reset, as shown above in Example 7-1, is not behaviorally accurate. It requires an active edge to occur on either the clock or reset signals for the reset to occur. This implies that unlike real hardware, islands waking up with their asynchronous reset input asserted will *not* wake-up in a reset state. It is therefore necessary to wait until an active clock edge has occurred for the state of the island to be properly reset in the simulation.

An alternative approach could be to cause an active edge on the asynchronous reset signal. For example, driving the reset signal to “X” before asserting it again, but that approach is only feasible if the asynchronous reset input is externally controllable. It will not be possible to cause an extraneous active edge on any reset signal generated internally.

Note that there are ways to properly model the level-sensitive nature of asynchronous reset signals in SystemVerilog. The problem is that they are unlikely to be identified as such by synthesis tools.

7.5 DIRECTED TESTS

Some tests make use of random stimulus, but their primary objective is still accomplished through directly implemented stimulus. These test cases use the random stimulus simply as a mean to create background stimulus to verify that the functional correctness of other portions of the design is maintained.

7.5.1 POWER-ON RESET TEST

The first directed test that must be implemented is the power-on reset test. In all other simulations, the initial hardware reset sequence is applied instantaneously because it is the initial state after reset that is of interest. But in the power-on reset test, it is the analog nature of the power-on reset sequence that is of interest. Ensure that the initial state after reset, that we are relying upon for all the other tests, is correctly reached.

Rule 7.20 — The *`vmm_env::power_on_reset()`* task shall be overloaded for the power-on reset test.

It is necessary to replace the instantaneous hardware reset stimulus used by all other tests with a more accurate model of the power-on reset sequence. This is done by creating a test-specific extension of the verification environment.

Example 7-3. Overloaded power-on reset sequence

```
class por_env extends tb_env;
   virtual task power_on_reset;
      ...
   endtask
endclass
```

Rule 7.21 — A short broad-spectrum random test shall be used.

The power-on reset test is only concerned with the verification of the power-on-reset sequence and the initial state it forces on the design. Running a short but broad-spectrum test after the power-on reset sequence has been applied will verify that the initial state of the design was appropriately set.

If the verification environment implements an unconstrained random test by default, the power-on reset test simply needs to limit the duration of this default test, as shown in Example 7-4.

Example 7-4. Power-on Reset Test on Random Environment

```
program por_test;

por_env env = new;
```

```
initial
begin
   env.gen_cfg();
   env.run_for = 10;
   env.run();
end
endprogram
```

7.5.2 HARDWARE RESET TEST

The second directed test that must be implemented is the hardware reset test. In all other simulations, the hardware reset sequence is applied at the beginning of the simulation to put the design into a known initial state. However, it is important to verify that the design can return into a functional state after having been forcibly reset. It also verifies that the initial state of the design does not depend on initial simulator values

The main challenge in implementing such a test case is that the simulation sequence defined in a VMM environment cannot be violated. It is not possible to invoke the *vmm_env::reset_dut()* method multiple times during a simulation. It is possible to invoke the *vmm_env::hw_reset()* task but performing a hardware reset requires the subsequent resetting all transactors and the response checking structure, reconfiguring the design, then restarting all transactors, which would effectively duplicate what is already implemented in the various simulation steps.

The following guidelines show how to make use of the new *vmm_env::restart_test()* method.

Rule 7.22 — The self-checking structure shall be resettable.

When the device is reset, all pending stimulus will be wiped out. It is necessary to be able to similarly wipe out the response checking structure of any pending expected response.

Rule 7.23 — The virtual *vmm_env::reset_env()* task shall be implemented for a *FIRM* reset.

This method specifies how the various transactors and other verification components in the verification environment are reset so they can be restarted again. Note that after a *FIRM* reset, the environment will be restarted at the *vmm_env::reset_dut()* step and thus the *vmm_env::build()* step will not be executed again. It is thus necessary to make sure that any required callback registration is not cleared while resetting a component.

Rule 7.24 — A broad-spectrum random test shall be used.

It is necessary to use a test case that will move all design state information as far away from its initial state as possible. Generally, a broad-spectrum random test, run for long enough, should be sufficient.

However, it may be necessary to repeat the hardware reset test with multiple tests if it is not possible to exercise the entire design with a single test.

Rule 7.25 — Only threads started after the *vmm_env::build()* step shall be aborted before calling *vmm_env::restart_test().*

At a suitable point in time, during the execution of the broad-spectrum test, the *vmm_env::restart_test()* method needs to be called to abort the running test and reset the verification environment and then the test must be restarted. For the execution of the test to resume correctly, any thread started before the restart point (in this case, the *vmm_env::reset_dut()* step), must still be running. It is very important to use the *disable* statement to abort the execution of the test, but have an effect limited to threads started after that step.

Example 7-6. Hardware Reset Test on Random Environment

```
program hwr_test;

tb_env env = new;

initial
begin
   env.build();
   fork: test case
      env.run();
   join_none
   // Wait for the test to be in full progress
   ...
   disable test case;
   env.restart_test();
   env.run();
end
endprogram
```

7.6 Power Management Software

In today's large system-on-a-chip (SoC), a significant portion of the power management system is implemented in the embedded or application software running on top of it. For example, it is the software running on a cell phone that will be responsible for powering up the digital camera whenever the user selects the photo-

taking mode on his or her cell phone, and to conversely power it down when the photo-taking mode is exited. It is the power management firmware (PMFW) that is responsible for the timely and correct transitions of various islands in the design to (set) different power states.

Rule 7.26 — The power management firmware code shall be verified.

As the power management firmware is an integral part of the overall power management system of your design, it is important that is be verified accordingly: as an integral part of the functionality under verification. This implies that the *actual* firmware code that will be ultimately loaded in the device must be verified. If the power management firmware is replaced by a model of its behavior, for example a mocked-up interrupt service routine implemented in SystemVerilog accessing the registers through a bus-functional model of the processor, a critical component of the power management system will not have been verified. How can you ensure that the actual firmware will exhibit the same behavior?

Running actual firmware code raises the spectre of simulations dragged down to a virtual halt by the object code executing on an RTL or instruction-set model of the processor running the compiled firmware code. Fortunately, a suitable methodology, as the one described in the remainder of this section, can alleviate this run-time issue and provide functional verification without sacrificing run-time performance.

Rule 7.27 — Once verified, the power management firmware code shall not be modified.

As soon as something is modified, it can no longer be considered as “verified.” Any change may have introduced a functional error. The exact same firmware code that will be compiled and then run on the target application processor must be verified as-is, without any modifications.

One method to verify the unmodified firmware code is to compile it and run it on an instruction-set simulator of the target process. However, this approach causes a significant slow-down of the simulation for two reasons. First, the model of the processor increases the complexity and size of the simulation, thus requiring more events and models to be processed. Second, the execution of object code is more often than not concerned with executing instructions that do not directly interact with the design that is the primary focus of the verification.

Rule 7.28 — A register-level co-simulation API should be used.

The only functionally significant interactions between the power management firmware and the design it controls are the read and write access to the appropriate control registers. A register-level co-simulation API allows the algorithmic and

decision-making portion of the firmware to be executed natively on a workstation and interact with the simulation only when control registers must be read or written.

For example, the VMM Register Abstraction Layer (RAL) provides a C/C++ interface that allows C/C++ code running on a computer to read and write software-accessible registers in a simulated design in a SystemVerilog simulator. The C/C++ code is executed by the host computer, not a model of a processor in a simulator. The simulation of the design is not slowed down. The interface between the software running on the host processor and the simulated design is provided by RAL and is designed to minimize the run-time impact of the software interaction with the design through a bus-functional model of the processor.

Chapter 17 and Appendix C of the *VMM Register Abstraction Layer User Guide* details the C API provided by the VMM Register Abstraction Layer. Example 7-7 shows how it can be used to read registers.

Example 7-7. Setting a bit through RAL

```
int pwr;
ral_read_PWR_CTRL_in_dut(dut, &pwr);
pwr |= 0x0040;
ral_write_PWR_CTRL_in_dut(dut, &pwr);
```

Rule 7.29 — A pure C implementation of the register-level co-simulation API shall be used to compile the firmware on the target processor.

Ultimately, the power management firmware must be compiled to be executed on the actual processor in the end product. Yet, it must not be modified after having been verified. This implies that the co-simulation API must have an alternative C/C++ implementation that can be compiled on the target simulator.

For example, the VMM RAL provides an alternative pure C implementation of the register access API that can be used to compile the firmware for execution on the actual design. As illustrated in Figure 7-3, the pure C implementation is generated from the same register specification as the co-simulation implementation and provides the exact same functionality. This allows the same code to be executed in a co-simulation environment and on the actual design without any modifications.

.

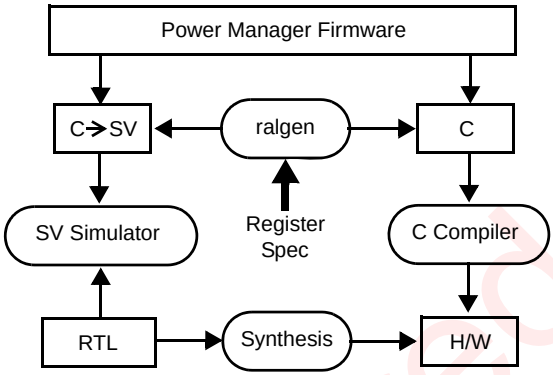


Figure 7-3. RAL-Generated APIs

Rule 7.30 — All hardware-to-firmware event notifications shall be via interrupts.

There are two methods for firmware to be notified of a significant event in the hardware, such as the acknowledgement that a power rail has been turned on or off. The firmware can be asynchronously notified via an interrupt signal that must then be serviced, or the firmware can continuously poll the hardware looking for significant events.

The former method is more complex to handle, but the latter method consumes more power as the processor must continuously fetch, decode, and execute instructions. Many of the signals that make up the on-chip bus continuously transition, forcing the slaves to continuously answer to the polled addresses.

An interrupt-based power-management firmware will also be more efficient to verify in a co-simulation environment as it will only execute when required. This would allow the hardware simulation to proceed as fast as possible. Should a polling-based implementation be used, the hardware simulation could only advance during the read cycles, as it is the only time in which simulation could be allowed to proceed.

Rule 7.31 — All power-related interrupt status bits shall be OR'd into a single power management interrupt status bit.

There are always more conditions that can cause an interrupt than there are interrupt signals. To circumvent this limitation, interrupt status bits are OR'd (after having been optionally masked) into a final interrupt signal. The interrupt service routine needs to read the interrupt status register to identify, and thus properly service, the cause of the interrupt. This mechanism can be recursively applied to create a tree of

interrupt status bits. As shown in Figure 7-4, a branch of the interrupt tree must be dedicated to power management interrupts.

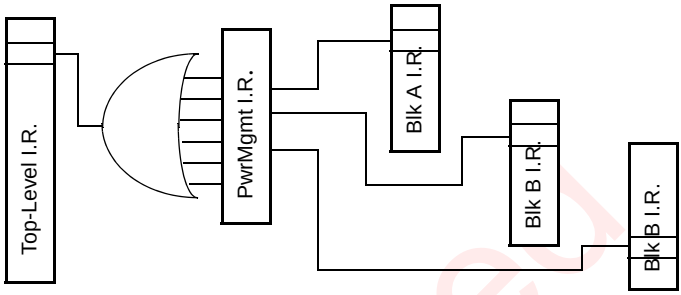


Figure 7-4. Power Management Interrupt Structure

This approach lends itself to writing modular interrupt service routines, with a function designed to service a particular reported interrupt status.

Rule 7.23 — The power management interrupt service routine shall be encapsulated into a single function.

By collecting all of the power-related interrupt status bits into a single power management interrupt signal, a single function can be used to analyze the cause of the power management interrupt: query the power-related interrupt status bits in all known power status registers and appropriately service it.

This software structure makes it easy to invoke the power management firmware, and only the power management firmware, whenever required by the design or desired by the testbench.

Example 7-8. Power Management Firmware Function

```
int
dut_pwr_mgmt(size_t dut)
{
   ...
}
```

7.7 CONCLUSION

This chapter described guidelines for implementing a VMM-based verification environment to dynamically verify the power-management features of a design. To ease that task, it leveraged the existing functional verification infrastructure and pre-defined functionality specified in Appendix A.

This chapter focused on how to implement a verification environment and the dynamic testcases on top of it. It did not cover the specific testcases that need to be executed, other than the basic power-up tests. That is because the tests that need to be executed depend on the design to be verified and the low-power architecture it uses. These tests should have been identified during the verification planning stage, as described in Chapter 6. They should aid to exercise all potential failures outlined in Chapter 3 that cannot be verified statically with your existing toolset.

Taken together—the guidelines outlined in this chapter and the tests planning outlined in earlier chapters—will ensure that the power-management features of the DUT are operating as expected.

CHAPTER 8 RULES AND GUIDELINES

8.1 SUMMARY OF RULES AND GUIDELINES

Following is a summary of rules, guidelines and recommendations.

CHAPTER 2 — RULES AND GUIDELINES

Rule 2.1—Safe Graph Rule:Each State in the Power State Table must have a transition possible to at least one other state to which it differs in only one voltage level.

CHAPTER 3 — POWER MANAGEMENT

Rule 3.1a — A spatial crossing must be protected with the appropriate circuit at all times.

Rule 3.1b — If an enable signal exists, the protection circuit must be suitably enabled or disabled from its function at all times.

Rule 3.1c — Latch based isolation devices must be tested for wakeup from the all-off state for the entire power up and down sequence.

Recommendation 3.1d — Use appropriate reset for latch based isolation devices.

Recommendation 3-2 — Do not use ungated latch or pull up /pull down type of isolation.

Rule 3.2a — A level shifter must be used when there is a difference greater than a number determined by the technology library, between Source and Destination Vdd levels.

Rule 3.2b — Tolerance of the Vdd levels must be taken into consideration for level shifting determination.

Rule 3.2c — Level Shifting must also be determined by transient conditions on voltage rails across each source/destination pair.

Guideline 3.3a — Crossovers and Isolation enables are essential coverage points.

Guideline 3.3b — Verify that isolation levels are inactive. For example, a fan can't be tied to “on” level, an arbiter request cannot be tied to a logic level that indicates active request.

Guideline 3.3c — Signals with isolation should be level sensitive, not edge sensitive. The very act of isolating and releasing may cause an edge.

Rule 3-4 — Redundant activation of isolation must be checked by appropriate assertions.

Rule 3.5a — Do *not* wiggle any pin, especially clocks, read/write pins when in standby. (Resets are usually supported at this voltage.)

Rule 3.5b — A clock gating device must be present and gated to inactive clock level in this situation.

Rule 3.6a — Clocks, resets and other high fanout nets in off islands must be gated inactive when the island is powered off.

Rule 3.6b — A transistor level check must be done to waive this violation.

Rule 3.7a — The Voltage ID (VID) codes of a voltage regulator must have assertions on them to detect the scheduling of voltage.

Rule 3.7b — Memory IP datasheets or technology files must specify the minimum standby voltage

Rule 3.8 — The behavior of a voltage source must be modeled in an electrically accurate manner.

Rule 3.9 — The testbench and test cases must include a provision to model software observation and control of power management.

CHAPTER 4 — STATE RETENTION

Rule 4.1 — Partitioning of selective retention must be verified by appropriate register tests.

Recommendation 4.2 — Post-restore coverage must be measured as the number of registers 'used' in operations after restore.

Rule 4.3 — The intermediate period between saves/restores must be tested to check for changes to saved registers and dependencies on unrestored registers.

Recommendation 4.4 — Selective retention must be applied at the module level; partitions within a module must be avoided.

Recommendation 4.5 — Unless a design has been architected for partial state retention explicitly, then don’t go there! Full state retention is strongly recommended for both implementation and verification flows.

Recommendation 4.6 — Retained and non-retained states should have explicit independent reset networks in the RTL design. This allows functional simulation testing before state retention implementation, and gives the verification tools clear visibility as to which reset terms factor into state registers unambiguously.

Rule 4.7 — Only use a single edge of the clock to ensure the clock gating latch state can always be re-evaluated correctly.

CHAPTER 5 — MULTI-VOLTAGE TESTBENCH ARCHITECTURE AND PREPARATION

Rule 5.1 — Power management software must be tested with the control loop that triggers it.

Rule 5.2 — Behavioral models not covered by multi-voltage semantics must be modified accordingly to respond to power management events.

Recommendation 5.3 — Use RTL models for components inside the DUT for power management tests.

Recommendation 5.4 — Constructs that inhibit propagation of X logic values should be avoided in RTL.

Recommendation 5.5 — Corruption in simulation may not be observed in simulation results and assertion failures must be used to detect such situations.

Rule 5.5 — Instead of hardwired constants, use TIE_HI_<NAME>or TIE_LO_<NAME> to explicitly identify the intended connection.

Rule 5.6 — Make sure that constants do not cross domain boundaries. Their behavior will need to be comprehensively analyzed across all sources: destination state combinations if they do so. An additional level shifter may also be (wastefully) needed if this is done.

Recommendation 5.7 — Avoid port map expressions at power domain boundaries, they are likely to cause improper specification and hence difficult to verify.

Rule 5.8 — Do not use first stage flip-flops if the domain is going to be turned off, unless input isolation is used.. Verification tools must ensure that this is the case.

Rule 5.9 — Verify that gate clocks are gated down to first stage inactive if the domain is going to be turned off.First stage inactive means that it must be either a cmos gate connection or the pass transistor it hooks up to must be closed.

Rule 5.10 — Verify that elements with first stage pass transistors are not used at the domain boundaries.

Rule 5.11 — A block of IP must be verified to be compliant to its original design properties, even if the integration is not in violation of top level power intent.

Recommendation 5.12 — An IP exporter must provide adequate assertions and coverage points to the end-user to verify low power states and functionality.

CHAPTER 6 — MULTI-VOLTAGE VERIFICATION

Rule 6.1 — Verification must first focus on electrical safety of the design.

Recommendation 6.2 — Any errors that can be detected statically must be fixed before dynamic verification is performed. Dynamic verification must account for any errors that are not fixed.

Rule 6.3 — The testbench must have assertions to protect against transactions with a block that is in off or standby mode.

Rule 6.3a — Software addressable registers known to be in on/off islands must have assertions to verify that access happens only when they are in the on state.

Recommendation 6.4 — Identify critical control signals that originate in On/Off blocks and verify that there is no dependency on them to transition out of the state in which their source is off.

Rule 6.5 — The implementation must be checked both as a stand alone as well as for equivalence to the original reference design across all power states.

Recommendation 6.4 — Each power state needs to be tested exhaustively by covering all the major micro-architectural elements in that state.

Suggestion 6.5 — In each power state, coverage of all logic that is on should be as close to complete as possible.

Recommendation 6.6 — Each power state needs to be tested exhaustively by covering all the major micro-architectural elements in that state.

Recommendation 6.7 — In each power state, coverage of all logic that is on should be as close to complete as possible.

Rule 6.8 — Transitions must be tested for abort signals if applicable.

Rule 6.9 — Assertions to guard against multiple rail changes at the same time must be present.

Recommendation 6.9 — Level shifter range violations must be guarded against by writing appropriate assertions at each spatial crossing across voltage domain boundaries.

Rule 6.10 — Power states must be tested for conflicting transition inputs and priority of transitions must be resolved as architecturally intended.

Rule 6.11 — Coverage must be measured on design elements such as islands and power management controls, apart from the power state table, transitions and sequences.

Rule 6.12 — The default, unconfigured functionality of configurable power management hierarchy must be tested.

Rule 6.12a — The programmed/configured functionality of configurable power management hierarchy must be tested.

CHAPTER 7 — DYNAMIC VERIFICATION

Recommendation 7.1 — Power domains shall be verified independently first.

Rule 7.2 — The All-ON functionality of the design shall be verified first.

Rule 7.3 — A power-aware simulator shall be used to verify the correctness of a powered-down domain.

Rule 7.4 — The response-checking mechanism shall be notified of the start and end of any power state changes.

Recommendation 7.5 — The exact timing of the effect of a power state transition should not be predicted.

Rule 7.6 — Functional correctness shall be confirmed after each transition.

Rule 7.7 — The self-checking structure shall use a vmm-consensus instance to confirm observed correctness.

Suggestion 7.8 — The power state functional coverage model may need to include power state paths.

Rule 7.9 — State retention power states shall be visited without going through a reset.

Rule 7.10 — Every retention cell shall retain a non-reset value.

Rule 7.11 — The frequency of the scaled clock shall be verified.

Rule 7.12 — The All-On verification environment shall be designed to support the subsequent power verification.

Rule 7.13 — The *vmm_env::rst_dut()* method shall not be extended.

Rule 7.14 — The power-on reset sequence shall be modeled in an extension of the *vmm_env::power_on_reset()* method

Rule 7.15 — An active external hardware reset sequence shall be modeled in an extension of the *vmm_env::hw_reset()* method.

Rule 7.16 — The DUT shall be brought into the All-ON state in an extension of the vmm_env::power_up() method.

Recommendation 7.17 — If using a register abstraction layer, registers in different power domains should not be grouped together.

Rule 7.18 — Clock gating models shall not use delays nor non-blocking assignments.

Rule 7.19 — Asynchronous reset signals shall remain asserted for at least one active clock edge.

Rule 7.20 — The body of the vmm_env::reset_dut() task shall be implemented in a virtual task.

Rule 7.21 — A short broad-spectrum random test shall be used.

Rule 7.22 — The self-checking structure shall be resettable.

When the device is reset, all pending stimulus will be wiped out. It is necessary to be able to similarly wipe out the response checking structure of any pending expected response.

Rule 7.23 — The virtual *vmm_env::reset_env()* task shall be implemented for a *FIRM* reset.

Rule 7.24 — A broad-spectrum random test shall be used.

Rule 7.25 — Only threads started after the *vmm_env::build()* step shall be aborted before calling *vmm_env::restart_test().*

Rule 7.26 — The power management firmware code shall be verified.

Rule 7.27 — Once verified, the power management firmware code shall not be modified.

Rule 7.28 — A register-level co-simulation API should be used.

Rule 7.29 — A pure C implementation of the register-level co-simulation API shall be used to compile the firmware on the target processor.

Rule 7.30 — All hardware-to-firmware event notifications shall be via interrupts.

Rule 7.32 — All power-related interrupt status bits shall be OR'd into a single power management interrupt status bit.

Rule 7.23 — The power management interrupt service routine shall be encapsulated into a single function.

APPENDIX A VMM-LP BASE CLASS AND APPLICATION PACKAGE

This appendix specifies the detailed behavior of a set of base and utility classes that can be used to implement the VMM-LP methodology described in this book. These classes are specified as additions to the VMM Standard Library, as described in Appendix A of the *Verification Methodology Manual for SystemVerilog*, and in the VMM Application Packages. The actual implementation of these classes is left to each tool provider. Chapter 7, “Dynamic Verification” provides detailed guidelines on how to use these classes.

At this time, only the classes that differ from or are not included in the specification in Appendix A of the Verification Methodology Manual for SystemVerilog, in Appendix A of the VMM Standard Library User’s Guide, or in the VMM Register Abstraction Layer User’s Guide are included here.

A.1 RALF CONSTRUCT SUMMARY

A.1.1 REGISTER

The following properties are added to the register RALF specification.

Properties

```
[attributes {
<name> <value>[, ...]
}]
```

Specifies a value for the specified user-defined attribute. Multiple attributes may be specified by separating each attribute-value pair with a comma. If the value contains white spaces, it must be included between double-quotes.

Example A-1 Attribute specification for a register

```
register R {
...
attributes {
    NO_RAL_TESTS 1,
    RETAIN       1
}
}
```

A.1.2 MEMORY

The following properties are added to the memory RALF specification.

Properties

```
[attributes {
<name> <value>[, ...]
}]
```

Specifies a value for the specified user-defined attribute. Multiple attributes may be specified by separating each attribute-value pair with a comma. If the value contains white spaces, it must be included between double-quotes.

A.1.3 BLOCK

The following properties are added to the block RALF specification.

Properties

```
[attributes {
<name> <value>[, ...]
}]
```

Specifies a value for the specified user-defined attribute. Multiple attributes may be specified by separating each attribute-value pair with a comma. If the value contains white spaces, it must be included between double-quotes.

A.1.4 SYSTEM

The following properties are added to the system RALF specification.

Properties

```
[attributes {
<name> <value>[, ...]
}]
```

Specifies a value for the specified user-defined attribute. Multiple attributes may be specified by separating each attribute-value pair with a comma. If the value contains white spaces, it must be included between double-quotes.

A.2 VMM-LP CLASS LIBRARY SPECIFICATION

This section specifies the detailed behavior of a set of base and utility classes that can be used to implement the VMM-LP methodology described in the *VMM Low Power Verification Methodology* book. They are specified as additions to the VMM Standard Library, as described in Appendix A of the *Verification Methodology Manual for SystemVerilog*, Appendix A of the *VMM Standard Library User's Guide*, and to the *VMM Register Abstraction Layer User's Guide*.

The classes are documented in alphabetical order. The methods in each class are documented in a logical order, where methods that accomplish similar results are documented sequentially. A summary of all available methods with cross references

to the page where their detailed documentation can be found is provided at the beginning of each class specification.

At this time, only the classes that differ from or are not included in the specification in Appendix A of the *Verification Methodology Manual for SystemVerilog*, Appendix A of the *VMM Standard Library User's Guide*, or the *VMM Register Abstraction Layer User's Guide* are included here.

VMM-LP Library Class Summary:

A.2.1 VMM_ENV

The following members are added to or modified in the vmm_env class.

Summary:

A.2.1.1 VMM_ENV::HW_RESET()

Hardware reset sequence.

SystemVerilog

```
virtual task hw_reset();
```

Description

Perform a hardware reset sequence on an active design. This method differs from the method in that it assumes that all clocks and power signals are active and that the design will quickly respond to an assertion of the hardware reset signal.

This method may be repeatedly invoked to reset the design under verification.

By default, this method does nothing. It must be implemented for every environment to perform a rapid hardware reset of the entire DUT.

A.2.1.2 VMM_ENV::POWER_ON_RESET()

Power-on reset sequence.

SystemVerilog

```
virtual protected task power_on_reset();
```

Description

Perform the initial power-on reset sequence, including enabling clock sources. This method is automatically called by the *`vmm_env::reset_dut()`* method and must not be invoked directly.

This method differs from the method in that it assumes that all clocks and power signals are off and that the analog nature of the assertion of the hardware reset signal during power-up are to be accurately modelled.

The default implementation of this method calls the method. This default functionality should be sufficient when not using a power-aware simulator.

A.2.1.3 VMM_ENV::RESET_DUT()

Reset simulation step.

SystemVerilog

```
virtual task reset_dut();
```

Description

This method implements the DUT reset simulation step in the test simulation sequence.

This method invokes the task and should not be otherwise overloaded.

This method may be directly invoked by a testcase to move the state of the simulation past the DUT reset stage. Once this method returns, the design is in the default initialization state and should be ready to be configured.

A.2.1.4 VMM_ENV::POWER_UP()

Power-up the design under verification.

SystemVerilog

```
virtual task power_up();
```

Description

Power up the design to execute the test. This method is automatically called by the method and should not be invoked directly.

The default implementation of this method is empty. If the design under verification is initialized in a powered-down state after the execution of the method, this method must be implemented to put the design in the required active state.

This first-level implementation of this method should put the design in an all-ON or fully powered up state to allow any functional test to execute without having to worry about missing functionality because the corresponding design implementation was powered down.

Second-level implementations of this method can be used to put the design in a test-specific initial power state.

A.2.1.5 VMM_ENV::CFG_DUT()

DUT configuration simulation step.

SystemVerilog

```
virtual task cfg_dut();
```

Description

This method implements the DUT configuration simulation step in the test simulation sequence.

This method ensures that the has been invoked then invokes the task. Any user-defined implementation of this method must first invoke *super.cfg_dut()*.

This method may be directly invoked by a testcase to move the state of the simulation past the DUT configuration stage. Once this method returns, the design is in a known configured state and should be ready to accept stimulus.

A.2.1.6 VMM_ENV::RESET_DUT()

Reset the simulation environment.

SystemVerilog

```
virtual protected task reset_env();
```

Description

This method specifies how the environment is reset to the state before the *vmm_env::reset_dut()* simulation step is called. This method is automatically called when the simulation sequence is restarted using *vmm_env::restart(vmm_env::FIRM)*.

By default, this method is empty. It must be appropriately implemented for each environment to allow the a testcase to be interrupted by the unexpected occurrence of hardware reset and properly restarted. Care must be taken that all callback registrations and threads started after the end of the *vmm_env::build()* step are properly unregistered and killed respectively.All transactors must be similarly reset using *vmm_xactor::reset_xactor()*, all channels must be flushed using *vmm_channel::flush()* and all scoreboards must be wiped.

A.2.2 VMM_LP_DESIGN

This class implements a power state management service that mirrors the current power state of the design under verification. It provides an interface for transitioning between power states. It also maintains power state dependencies to ensure power transition sequences are not violated.

A single instance of this class manages and tracks the power state in a design. The various power domains in the design are defined. The various power states of each power domains are similarly defined, as well as the routines that allow the design to transition one of its power domain to a specific power state.

Power state dependencies may also be defined. For example, it may be required for power domain A to be ON to be able to transition power domain B. Power state dependencies can be checked for (requesting a transition on power domain B while

power domain A is OFF would report a run-time error) or enforced (the same request would implicitly cause a request for power domain A to be turned ON beforehand).

A design composed of separate designs, each with their own power domains and states can be described by composing multiple instances of this class.

Summary:

A.2.2.1 VMM_LP_DESIGN::NEW()

Create an instance of a power-state manager service.

SystemVerilog

```
function new(string name);
```

Description

Create an instance of the power state management service for a design with the specified name.

The specified name will be used as the instance name of the message service interface instance for this power state management service instance (refer to “vmm_lp_design::log” on page 157).

A.2.2.2 VMM_LP_DESIGN::LOG

Message service interface instance.

SystemVerilog

```
vmm_log log;
```

Description

Message service interface instance for this power state management service instance.

The name of the message service interface is *Power State Manager*. Its instance name is the name of the power state management service instance.

A.2.2.3 VMM_LP_DESIGN::NOTIFY

Notification service interface instance.

SystemVerilog

```
vmm_notify notify;
```

Description

Event notification interface instance for the power state management service instance.

The and notifications, are pre-defined for all instances of the power state management service. Notifications are also automatically defined for each power

domain that is defined using the method. These domain notifications are indicated when the power domain changes state.

A.2.2.4 VMM_LP_DESIGN::STATE_CHANGE

Notification of a power state change in the design.

SystemVerilog

```
typedef enum {STATE_CHANGE};
```

Description

Pre-defined notification indicating that a power state change is either starting or has just completed. The status of the notification is an instance of the class describing the state change.

This notification is automatically indicated whenever the method is used to transition from one power state to another. The state of this notification should not be directly modified.

A.2.2.5 VMM_LP_DESIGN::EXTERNAL_SUPPLY

Notification of the state of the external supply.

SystemVerilog

```
typedef enum {EXTERNAL_SUPPLY};
```

Description

Pre-defined ON_OFF notification indicate the current status of the external supply of the design, as specified by the method.

The state of this notification should not be directly modified.

A.2.2.6 VMM_LP_DESIGN::DEFINE_DOMAIN()

Define a power domain.

SystemVerilog

```
function bit define_domain(string name,
string states[])
```

Description

Define a power domain with the specified name and specified state names. One of the power state *must* be named ON. The first power state is assumed to be the name of the initial state of the domain after a hardware reset or power-on reset. The name of a domain may not contain the'.' character.

If a state name is specified between forward slashes, the slashes are stripped from the state name and the state is interpreted as being an intermediate logical state. Intermediate logical states are transient states involved in transitioning the domain from one steady-state power state to another.

If a state name ends with an exclamation mark, it specifies that the functionality of the domain is active when in that state. The ON state is assumed to be an active state and the presence of the exclamation mark is not necessary. The exclamation mark must be specified inside the forward slashes of an active intermediate logical state. The exclamation mark is not part of the final state name.

Returns TRUE if the domain could be defined as specified. Returns FALSE if there was an error in the domain specification.

Whenever a power domain is defined, a corresponding notification is also defined in the notification service interface. The identifier for the notification corresponding to the defined domain can be obtained by using the method.

Examples

Example A-2

```
pwr_mgr.define_domain("IO", '{"off", "ON*", "/ramp*/
", "Idle*", "stdby", "/retain/", "/restore/"});
```

A.2.2.7 VMM_LP_DESIGN::DEFINE_MODE()

Define a power mode.

SystemVerilog

```
function bit define_mode(string name,
string states[])
```

Description

Define a combination of power states as a steady-state operating mode with the specified name. A power state is specified using the “domain/state” format, in which *domain* is the name of the power domain and *state* is the name of a power state of the power domain. All of the domains in the power state list must be in the specified power state for the design to be considered in the specified mode. The name of a power mode may not contain a ’+’ or ’.’ character.

Two modes are pre-defined. The INITIAL mode is composed of the initial power state of all power domains, as defined by the method. The ALL-ON mode is composed of the ON power state of all power domains.

Returns TRUE if the power mode could be defined as specified. Returns FALSE if there was an error in the power mode specification.

A.2.2.8 VMM_LP_DESIGN::DEFINE_SUBDESIGN()

Define a hierarchical power managed design.

SystemVerilog

```
function bit define_subdesign(string name,
 subdesign,
bit shared_external_supply = 1)
```

Description

Include the specified power state management service for a design under this power state management service under the specified name. This can be used to assemble and configure the power state management service for a hierarchically power-managed design.

Once a power state management service has been defined as a sub-design of another power state management service, all of its power domains and power modes are available to the higher-level power state management service. A power domain or power mode in a sub-design is specified by prefixing it with the name of the sub-design and a ’.’ character. For example, the power domain DSP in the sub-design VIDEO would be specified as VIDEO.DSP.

If the `shared_external_supply` argument is TRUE, the external supply of the sub-design is the same as the external supply of the higher-level design. Turning the external supply of the higher-level design on or off via the method will implicitly

turn the external power supply of the sub-design on or off. If it is FALSE, the external power supply of the sub-design is assumed to be controlled by the higher-level design. It will be necessary for the power state transition code implemented in the (see page 170) methods to properly specify that the external supply of the sub-design is turned on or off.

A.2.2.9 VMM_LP_DESIGN::HOW_TO()

Define how to change power state.

SystemVerilog

```
function bit how_to(
string       domain,
string       from_state,
string       to_state,
             goto,
string       requires[])
```

Description

Specifies how to transition a power domain from a specific power state to another power state. The *from* state is specified as a regular expression matching the name of the power state from which the specified *to* state can be transitioned to.

A list of prerequisite power states for other domains may be specified as well. A prerequisite power state is specified using the *domain/state* format, in which *domain* is a regular expression matching the name of the prerequisite power domain, and *state* is a regular expression matching the name of the prerequisite state name. All of the matching domains in the prerequisite list must be in a matching prerequisite state for the transition to be possible. If no prerequisites are specified, an implicit prerequisite of "./." is assumed, i.e. the power domain can be transitioned to the specified power state regardless of the states of the other power domains.

Returns TRUE if the specified domain and states and all of the prerequisites match at least one domain and state. Returns FALSE if an unknown domain or state was specified.

A.2.2.10 VMM_LP_DESIGN::CHECK_CONFIG()

Check the configuration of the power state manager.

SystemVerilog

```
function bit check_config()
```

Description

Verify the integrity and correctness of the configuration of the power state manager. Returns TRUE of the configuration was found to be correct and complete. This method is implicitly called whenever any method modifying the power state of the design (such as , , , or) if first called.

Once this method has been called and the configuration of the power state manager has been confirmed as valid, the configuration of the power state manager is frozen and can no longer be modified.

A.2.2.11 VMM_LP_DESIGN::GET_DOMAINS()

Get the defined power domains.

SystemVerilog

```
function void get_domains(ref string names[])
```

Description

Replace the content of the array with the names of all defined power domains in this instance of the power management service.

The order in which domains are returned in the array is unspecified.

A.2.2.12 VMM_LP_DESIGN::GET_STATES()

Get the defined power states.

SystemVerilog

```
function void get_states(string domain,
ref string names[])
```

Description

Replace the content of the array with the names of all defined power states in the specified power domain in this instance of the power management service.

The order in which states are returned in the array is unspecified, except that the first state is the initial power state of the domain after a hardware reset.

If an invalid domain name is specified, an error message is issued and an empty array of state names is returned.

A.2.2.13 VMM_LP_DESIGN::GET_MODES()

Return defined power mode.

SystemVerilog

```
function void get_modes(ref string modes[])
```

Description

Replace the content of the array with the names of all defined power modes in this instance of the power management service.

The order in which modes are returned in the array is unspecified.

A.2.2.14 VMM_LP_DESIGN::NOTIFICATION_ID()

Get the notification identifier for a power domain.

SystemVerilog

```
function int notification_id(string domain = "")
```

Description

Return the notification identification for the state-change notification corresponding to the specified power domain. It is the identifier of the notification automatically defined in for every domain defined with .

The notification is an ON_OFF notification that is reset whenever the domain exits a state and is indicated whenever the domain enters into a new state. The notification is reset when the domain is between power states, and indicated when the domain is in a well-defined power state.

The notification is initially reset until the state of the power domain is explicitly defined, usually by specifying that the design has been reset using or that the external power has been turned on using .

If the specified domain does not exist, an error message is issued and an invalid notification identifier is returned. If no domain name is specified, the state-change notification identifier state for the external power supply for the entire design is returned (i.e. the value of).

A.2.2.15 VMM_LP_DESIGN::EXTERNAL_POWER()

Specify the state of the external supply.

SystemVerilog

```
function void ext_power(vmm_lp::state_e on_off);
```

Description

Specifies the current state of the external power supply as either *vmm_lp::ON* or *vmm_lp::OFF*. The design starts with the external power supply implicitly specified as OFF. The state of the external supply is reflected by the notification.

If the external supply is specified as ON when the external supply was previously OFF, the state of all defined power domains is set to their initial state as defined by the method and can be subsequently changed.

If the external supply is specified as OFF when the external supply was previously ON, the state of all power domains is forced to the OFF state, even if a state of that name has not been explicitly defined for a power domain.

Repeatedly specifying the same state of the external supply has no additional effect on the state of the power domains in the design.

A.2.2.16 VMM_LP_DESIGN::HW_RESET()

Specifies the design has been reset.

SystemVerilog

```
function void hw_reset();
```

Description

Indicates to the power state management service that the design has been externally reset. Resets the power state of all defined domains to their respective initial power state, as defined by the method.

A.2.2.17 VMM_LP_DESIGN::PSDISPLAY()

Print the power state of the design.

SystemVerilog

```
function string psdisplay(string prefix = "");
```

Description

Return a human-readable description of the current power state of the design. Each line of the description is prefixed with the specified prefix.

A.2.2.18 VMM_LP_DESIGN::WHERE()

Get the current power state of a domain.

SystemVerilog

```
function string where(string domain = "");
```

Description

Return the name of the current power state of the specified power domain. If the design is currently transitioning from one state to another, a string with the format *from_state->to_state* is returned.

If the specified domain does not exist, an error message is issued and an empty string is returned.

If no domain name is specified, the state of the external power supply for the entire design, as last specified by the method, is returned (i.e., ON or OFF).

A.2.2.19 VMM_LP_DESIGN::IS_IN()

Check the current power state of a domain.

SystemVerilog

```
function bit is_in(string domain,
string state)
```

Description

Check if the specified power domain is in the specified state. Returns TRUE if the specified domain is in the specified state. If the domain is specified as the empty string (i.e. ""), then the state of the external power supply is checked and the only valid states that can be specified are ON or OFF.

An error message is issued if the specified domain does not exist or if the specified state does not exist in the specified domain.

A.2.2.20 VMM_LP_DESIGN::IS_IN_TRANSITION()

Get if a domain is in transition between two states.

SystemVerilog

```
function bit is_in_transition(string domain)
```

OpenVera

Not supported.

Description

Check if the specified power domain is currently transitioning between two states. Returns TRUE if the specified domain is currently transitioning.

An error message is issued if the specified domain does not exist.

A.2.2.21 VMM_LP_DESIGN::IS_ACTIVE()

Check if the specified domain is currently in an active state.

SystemVerilog

```
function bit is_active(string domain)
```

Description

Check if the specified power domain is in a state that was specified as an active state (using a '*' suffix). Returns TRUE if the specified domain is in an active state.

An error message is issued if the specified domain does not exist or is specified as the empty string ("").

A.2.2.22 VMM_LP_DESIGN::IS_TRANSIENT()

Check if the specified domain is currently in a transient logical state.

SystemVerilog

```
function bit is_transient(string domain)
```

Description

Check if the specified power domain is in a state that was specified as an intermediate logical state (between forward slashes). Returns TRUE if the specified domain is in a transient state.

An error message is issued if the specified domain does not exist or is specified as the empty string ("").

A.2.2.23 VMM_LP_DESIGN::IN_MODE()

Return current power mode.

SystemVerilog

```
function string in_mode()
```

Description

Return the name of the power mode the design is currently in. If the design is not currently in a steady-state power-mode, an empty string is returned.

If the design is in more than one mode at the same time (a possibility if two modes are defined by the states of different power domains), the name is returned as the names of the individual power modes separated with a '+' character.

A.2.2.24 VMM_LP_DESIGN::GOTO_STATE()

Go to the specified power state.

SystemVerilog

```
task goto_state(
ref bit   ok,
input stringdomain,
input stringstate
input bit require = 1)
```

Description

Transition the specified power domain to the specified power state. Set `ok` to TRUE if the transition was successful.

If the *require* argument is TRUE, the necessary power state transitions to bring any required power domain to their prerequisite power state are implicitly performed beforehand. If the *require* argument is FALSE, and any required power domain is not at the prerequisite power state, an error is reported.

The notification associated with the power domain will be indicated once the transition completes.

A.2.2.25 VMM_LP_DESIGN::WANDER()

Perform a random power state transition.

SystemVerilog

```
task wander(
ref bit          ok,
ref              transition,
input bit        blindly = 0)
```

Description

Randomly pick a power state transition that can be made based on the current power state of the design and perform the transition. The status of the transition is returned in `ok`. The description of the power state transition chosen is assigned to `transition` immediately after being randomly picked.

If *blindly* is FALSE, power state transitions that have not yet been taken will be picked in preference to those transitions that have already been taken. If *blindly* is

TRUE, all possible power state transitions, whether already taken or not, will have an equal probability of being picked.

If a domain is currently in an intermediate logical state, a transition out of that state will be favored over transitioning a domain out of a steady-state power state.

A.2.2.26 VMM_LP_DESIGN::GOTO_MODE()

Go to the specified power mode.

SystemVerilog

```
task goto_mode(
ref bit   ok,
input stringmode)
```

Description

Put the design in the specified power mode by transitioning the necessary power domains to the respective required power state. Set `ok` to TRUE if the transition was successful.

A.2.3 VMM_LP_TRANSITION

An instance of this class specifies how to transition a power domain to a specific power state.

It is a virtual class that must be extended to be used. Each extension specifies how to transition one power domain from one power state to another.

Summary:

A.2.3.1 VMM_LP_TRANSITION::LOG

Message service interface instance.

SystemVerilog

```
protected vmm_log log;
```

Description

Message service interface instance of the power state management service instance where this power state transition descriptor is used.

This reference to the message service interface of the power state management service instance can be used to issue messages during the execution of a power state transition.

A.2.3.2 VMM_LP_TRANSITION::GOTO()

Perform this power state transition.

SystemVerilog

```
pure virtual protected task goto(ref bit ok)
```

Description

This method specifies how to transition a specific power domain from a specific power state to another power state. This method must be overloaded to implement whatever steps are necessary to cause and complete the transition.

The implementation of this method must set `ok` to TRUE if the transition was successful. Otherwise, it must set `ok` to FALSE.

The default implementation of this method issues a message that the state transition has not be implemented. Therefore, any user-defined extension of this method *must not* call the default implementation using *super.goto()*.

A.2.3.3 VMM_LP_TRANSITION::GET_LP_DESIGN()

Get the power state manager managing the transition.

SystemVerilog

```
function  get_lp_design()
```

Description

This method returns the power state management service instance that this instance of this class has been associated with using the method.

Returns *null* if this class instance has not been associated with a power domain.

A.2.3.4 VMM_LP_TRANSITION::GET_DOMAIN()

Get the domain being transitioned.

SystemVerilog

```
function string get_domain()
```

Description

This method returns the name of the domain this instance of this class has been associated with using the method.

Returns the empty string if this class instance has not been associated with a power domain.

A.2.3.5 VMM_LP_TRANSITION::GET_FROM_STATE()

Get the state being transitioned from. **SystemVerilog**

```
function string get_from_state();
```

Description

This method returns the name of the source power state this instance of this class has been associated with using the method.

Returns the empty string if this class instance has not been associated with a power domain.

A.2.3.6 VMM_LP_TRANSITION::GET_TO_STATE()

Get the state being transitioned to.

SystemVerilog

```
function string get_to_state();
```

Description

This method returns the name of the destination power state this instance of this class has been associated with using the method.

Returns the empty string if this class instance has not been associated with a power domain.

A.2.3.7 VMM_LP_TRANSITION::IS_DONE()

Check if the state transition is complete.

SystemVerilog

```
function bit is_done();
```

Description

This method returns TRUE if the power state transition is complete and the domain is in its destination power state. Returns FALSE if the power state transition is currently in progress or if the domain is in a different power state.

This method is useful to differentiate the indication of the notification at the start of a power state transition (FALSE) from the indication at the completion of a power state transition (TRUE).

A.3 RAL

The following methods and classes are added to the RAL Application package.

Summary:

A.3.1 VMM_RAL_BLOCK_OR_SYS::SET_ATTRIBUTE()

Set an attribute for a block or system.

SystemVerilog

```
virtual function void set_attribute(string name,
string value);
```

Description

Set the specified attribute to the specified value for this block or system. If the value is specified as "", the specified attribute is deleted.

A warning is issued if an existing attribute is modified.

Attribute names are case sensitive.

A.3.2 VMM_RAL_BLOCK_OR_SYS::GET_ATTRIBUTE()

Get an attribute from a block or system.

SystemVerilog

```
virtual function string get_attribute(string name,
bit inherited = 1);
```

Description

Get the value of the specified attribute for this block or system. If the attribute does not exist, "" is returned.

If the *inherited* argument is specifed as TRUE, the value of the attribute is inherited from the nearest enclosing system if it is not specified for this block or system. If it is specified as FALSE, the value "" is returned if it does not exist in the this block or system.

Attribute names are case sensitive.

A.3.3 VMM_RAL_BLOCK_OR_SYS::GET_ALL_ATTRIBUTES()

Get all attributes for a block or system.

SystemVerilog

```
virtual function void get_all_attributes(
ref string names[],
input bit inherited = 1);
```

Description

Return an array filled with the name of the attributes defined for this block or system.

If the inherited argument is specified as TRUE, the value of all attributes inherited from the enclosing system(s) is included. If the argument is specified as FALSE, only the attributed defined for this block or system is returned.

The order in which attribute names are returned is not specified.

A.3.4 VMM_RAL_BLOCK_OR_SYS::POWER_DOWN()

Specify that a block or system is powered down.

SystemVerilog

```
virtual function void power_down(bit retain = 0);
```

Description

Specify that this block or all blocks in this system has been put in a power-saving state. A read or write access to any register or memory inside the powered-down block will result in a run-time error message and a *vmm_ral::ERROR* status code.

If the *retain* argument is TRUE, the mirrored value of registers with an inherited non-zero RETAIN attribute value will be maintained and restored when the block is powered back up using the method (see page 175). If the *retain* argument is FALSE, the mirrored value of registers will be set to the reset value when the block or system is powered back up.

A powered-down block with retention enabled can be further powered down with retention disabled.

A.3.5 VMM_RAL_BLOCK_OR_SYS::POWER_UP()

Specify that blocks are powered back up.

SystemVerilog

```
virtual function void power_up(string power_domains
= "");
```

Description

Specify that the block or blocks in the system and the memories within them with an inherited POWER_DOMAIN attribute value that matches the specified power domain regular expression have been restored to a powered-up state. If the power domain is specified as "", then the block or blocks in the system and any memory within them are powered up regardless of the POWER_DOMAIN attribute value.

If a block is powered down using the method with a *retain* argument specified as TRUE, the mirrored value of registers with an inherited non-zero RETAIN attribute

value is restored. Otherwise, the mirrored value of registers is set to the specified reset value. By default, a block or system is powered-up.

A.3.6 VMM_RAL_MEM::SET_ATTRIBUTE()

Set an attribute for a memory.

SystemVerilog

```
virtual function void set_attribute(string name,
string value);
```

Description

Set the specified attribute to the specified value for this memory. If the value is specified as "", the specified attribute is deleted.

A warning is issued if an existing attribute is modified.

Attribute names are case sensitive.

A.3.7 VMM_RAL_MEM::GET_ATTRIBUTE()

Get an attribute for a memory.

SystemVerilog

```
virtual function string get_attribute(string name,
bit inherited = 1);
```

Description

Get the value of the specified attribute for this memory. If the attribute does not exists, "" is returned.

If the *inherited* argument is specified as TRUE, the value of the attribute is inherited from the nearest enclosing block or system if it is not specified for this memory. If it is specified as FALSE, the value "" is returned if it does not exists in the this memory.

Attribute names are case sensitive.

A.3.8 VMM_RAL_MEM::GET_ALL_ATTRIBUTES()

Get all attributes for a memory.

SystemVerilog

```
virtual function void get_all_attributes(
ref string names[],
input bit inherited = 1);
```

Description

Return an array filled with the name of the attributes defined for this memory.

If the *inherited* argument is specified as TRUE, the value of all attributes inherited from the enclosing block and system(s) is included. If the argument is specified as FALSE, only the attributed defined for this memory are returned.

The order in which attribute names are returned is not specified.

A.3.9 VMM_RAL_MEM::POWER_DOWN()

Specify that a memory is powered down.

SystemVerilog

```
virtual function void power_down();
```

Description

Specify that this memory has been put in a power-saving state. A read or write access to any location inside the memory will result in a run-time error message and a *vmm_ral::ERROR* status code.

A.3.10 VMM_RAL_MEM::POWER_UP()

Specify that a memory is powered back up.

SystemVerilog

```
virtual function void power_up();
```

Description

Specify that this memory has been a restored to a powered-up state.

A.3.11 VMM_RAL_REG::GET_RESET()

Get the reset value for a register.

SystemVerilog

```
virtual function bit [63:0] get_reset(
vmm_ral::reset_e kind = vmm_ral::HARD);
```

Description

Return the specified reset value for the register.

A.3.12 VMM_RAL_REG::SET_ATTRIBUTE()

Set an attribute for a register.

SystemVerilog

```
virtual function void set_attribute(string name,
string value);
```

Description

Set the specified attribute to the specified value for this register. If the value is specified as "", the specified attribute is deleted.

A warning is issued if an existing attribute is modified.

Attribute names are case sensitive.

A.3.13 VMM_RAL_REG::GET_ATTRIBUTE()

Get an attribute for a register.

SystemVerilog

```
virtual function string get_attribute(string name,
bit inherited = 1);
```

Description

Get the value of the specified attribute for this register. If the attribute does not exist, "" is returned.

If the *inherited* argument is specified as TRUE, the value of the attribute is inherited from the nearest enclosing block or system if it is not specified for this register. If it is specified as FALSE, the value "" is returned if it does not exists in the this register.

Attribute names are case sensitive.

A.3.14 VMM_RAL_REG::GET_ALL_ATTRIBUTES()

Get all attributes for a register.

SystemVerilog

```
virtual function void get_all_attributes(
ref string names[],
input bit inherited = 1);
```

Description

Return an array filled with the name of the attributes defined for this register.

If the *inherited* argument is specified as TRUE, the value of all attributes inherited from the enclosing block and system(s) is included. If the argument is specified as FALSE, only the attributed defined for this register are returned.

The order in which attribute names are returned is not specified.

A.3.15 VMM_RAL_TESTS::BIT_BASH()

Verify the bits in a registers.

SystemVerilog

```
static task bit_bash(vmm_ral_block blk,
string domain,
vmm_log log);
```

Description

Exercise every bit in the registers found in the specified domain in the specified block to ensure they behaves as specified. Bits in fields with a mode specified as USER or OTHER are not verified. If the domain is specified as "", all registers are exercised. Registers with an attribute named NO_BIT_BASH_TEST or NO_RAL_TESTS are skipped.

This method must be invoked only when the block has been fully powered-up and in the reset state. The state of the block must remain idle and stable so that the register values do no change from their initial states.

A.3.16 VMM_RAL_TESTS::HW_RESET()

Verify initial reset values of registers.

SystemVerilog

```
static task hw_reset(vmm_ral_block blk,
string domain,
vmm_log log);
```

Description

Read every register in the specified domains of the specified block and verify that the value read corresponds to the specified reset value. If the domain is specified as "", all registers are exercised. Registers with an attribute named NO_HW_RESET_TEST or NO_RAL_TESTS are skipped.

This method must be invoked only when the block has been fully powered-up and in the reset state. The state of the block must remain idle and stable so the register values do no change from their initial states.

A.3.17 VMM_RAL_TESTS::MEM_ACCESS()

Verify the operation of memory access.

SystemVerilog

```
static task mem_access(vmm_ral_block blk,
vmm_log log);
```

Description

Write every memory location through every available front-door domain and read the corresponding value through the backdoor and vice-versa. Memories with no backdoor access defined or with an attribute named NO_MEM_ACCESS_TEST or NO_RAL_TESTS are skipped.

This method must be invoked only when the block has been fully powered-up and in the reset state. The state of the block must remain idle and stable so that the memory locations do no change from their initial states.

A.3.18 VMM_RAL_TESTS::MEM_WALK()

Verify access to memory.

SystemVerilog

```
static task mem_walk(vmm_ral_block blk,
string domain,
vmm_log log);
```

Description

Write and then read back every location in every memory in the specified domains of the specified block, and verify that the value read corresponds to the previously-written value. If the domain is specified as "", all memories are exercised. Memories with an attribute named NO_MEM_WALK_TEST or NO_RAL_TESTS are skipped.

This method must be invoked only when the block has been fully powered-up. The content of the memories must remain unchanged and stable so that the values do not change from their previously-written states.

A.3.19 VMM_RAL_TESTS::REG_ACCESS()

Verify the operation of register access.

SystemVerilog

```
static task reg_access(vmm_ral_block blk,
vmm_log log);
```

Description

Write every register through every available front-door domain and read the corresponding value through the backdoor and vice-versa. Registers with no backdoor access defined or with an attribute named NO_REG_ACCESS_TEST or NO_RAL_TESTS are skipped.

This method must be invoked only when the block has been fully powered-up and in the reset state. The state of the block must remain idle and stable so that the register values do no change from their initial states.

A.3.20 VMM_RAL_TESTS::SHARED_ACCESS()

Verify the performance of shared access.

SystemVerilog

```
static task shared_access(vmm_ral_block blk,
vmm_log log);
```

Description

Write every shared register and shared memory locations through every available front-door domain and read the corresponding value through every other available front door (and backdoor if available) and vice-versa. Registers and memories that are not shared across multiple domains or with an attribute named NO_SHARED_ACCESS_TEST or NO_RAL_TESTS are skipped.

This method must be invoked only when the block has been fully powered-up and in the reset state. The state of the block must remain idle and stable so the register values and memory locations do no change from their initial states.

APPENDIX B STATIC CHECKS

The following list is an example snap shot. It changes with design style, flow and libraries. The intent is for this to be used as a generic starting point to formulate a sign off check list. The design stage at which to apply the checks is also identified.

Note—RTL checks apply in the case where the elements in the checklist are instantiated in the source code. They also apply when the power intent commands are checked.

B.1 ISOLATION CHECKS

Missing Isolation Cell—RTL, Netlist, PG-Netlist

Incorrect Isolation Cell type—RTL, Netlist, PG-Netlist

Incorrect Isolation Cell output polarity—RTL, Netlist, PG-Netlist

Redundant Isolation Cell on a crossover signal—RTL, Netlist, PG-Netlist

Redundant Isolation Cell used inside an island—RTL, Netlist, PG-Netlist

Incorrect power-pin connectivity—PG-Netlist

Incorrect ground-pin connectivity—PG-Netlist

Incorrect isolation enable signal—RTL, Netlist, PG-Netlist

Incorrect isolation enable signal polarity—RTL, Netlist, PG-Netlist

Incorrect Isolation Cell functional implementation—RTL, Netlist, PG-Netlist

Incorrect protection cell (e.g., Isolation instead of Level Shifter)—RTL, Netlist, PG-Netlist

Incorrect isolation enable signal network (path), e.g., going through an Off island—RTL, Netlist, PG-Netlist

All the above checks for input isolation—RTL, Netlist, PG-Netlist

B.2 Level Shifters (LS)

Missing LS—RTL, Netlist, PG-Netlist

Incorrect LS cell type—RTL, Netlist, PG-Netlist

Incorrect LS voltage range—RTL, Netlist, PG-Netlist

Redundant LS on a crossover signal—RTL, Netlist, PG-Netlist

Redundant LS inside an island—RTL, Netlist, PG-Netlist

Incorrect power connectivity of LS- PG-Netlist

B.3 Enabled Level Shifters (ELS)

Missing Isolation Cell—RTL, Netlist, PG-Netlist

Incorrect Isolation Cell type—RTL, Netlist, PG-Netlist

Incorrect Isolation Cell output polarity—RTL, Netlist, PG-Netlist

Redundant Isolation Cell on a crossover signal—RTL, Netlist, PG-Netlist

Redundant Isolation Cell used inside an island—RTL, Netlist, PG-Netlist

Incorrect isolation enable signal—RTL, Netlist, PG-Netlist

Incorrect isolation enable signal network (path), e.g., going through an Off island—RTL, Netlist, PG-Netlist

Incorrect isolation enable signal polarity—RTL, Netlist, PG-Netlist

Incorrect Isolation Cell functional implementation—RTL, Netlist, PG-Netlist

Incorrect protection cell (e.g., Isolation instead of LS)—RTL, Netlist, PG-Netlist

Missing LS—RTL, Netlist, PG-Netlist

Incorrect LS cell type—RTL, Netlist, PG-Netlist

Incorrect LS voltage range—RTL, Netlist, PG-Netlist

Redundant LS on a crossover signal—RTL, Netlist, PG-Netlist

Redundant LS inside an island—RTL, Netlist, PG-Netlist

Incorrect power connectivity of ELS (VDD1, VDD2, VSS)—PG-Netlist

B.4 ISLAND ORDERING CHECKS

In general, a control/critical signal driven to an ON island by an OFF island in one of the power states. Qualifying signals are mentioned below:

Clock—RTL, Netlist, PG-Netlist

Clock Enable—RTL, Netlist, PG-Netlist

Reset—RTL, Netlist, PG-Netlist

Scan Enable—RTL, Netlist, PG-Netlist

Isolation Enable—RTL, Netlist, PG-Netlist

Power Enable—RTL, Netlist, PG-Netlist

Power OK—RTL, Netlist, PG-Netlist

Retention save signal—RTL, Netlist, PG-Netlist

Retention restore signal—RTL, Netlist, PG-Netlist

User specified signals—RTL, Netlist, PG-Netlist

B.5 Retention Cells

Redundant retention cells (More cells than specified in power intent)- Netlist, PG-Netlist

Incorrect save signal connectivity—Netlist, PG-Netlist

Incorrect save signal polarity—Netlist, PG-Netlist

Incorrect restore signal connectivity- Netlist, PG-Netlist

Incorrect restore signal polarity—Netlist, PG-Netlist

Incorrect power enable signal (if applicable)—Netlist, PG-Netlist

Incorrect power enable signal polarity (if applicable)—Netlist, PG-Netlist

Incorrect primary power connectivity—PG-Netlist

Incorrect back-up power connectivity—PG-Netlist

Incorrect ground connectivity—PG-Netlist

Incorrect isolation enable signal network (path), e.g., going through an Off island—RTL, Netlist, PG-Netlist

B.6 POWER SWITCH

Incorrect partition of a power-switch in power intent—Netlist, PG-Netlist

Incorrect connectivity of power-rail coming into the power-switch—PG-Netlist

Incorrect connectivity from power switch to cells in the power domain—PG-Netlist

Incorrect power-enable signal connectivity to power switch—Netlist, PG-Netlist

Incorrect power-ack signal connectivity between daisy-chained switches—Netlist, PG-Netlist

Incorrect polarity of power switch control signals—RTL, Netlist, PG-Netlist

Incorrect power switch enable signal network(path)—RTL, Netlist, PG-Netlist

B.7 ALWAYS-ON CELLS

Power-pin connectivity of always-on cells—PG-Netlist

Floating/undriven inputs to always-on cells—PG-Netlist

(Isolation/Level shifting requirements apply as well to Always On domains)

APPENDIX C REFERENCES AND RECOMMENDED READING

List of References and Recommended Reading:

[1] — Bergeron, J et al; *Verification Methodology Manual for System Verilog*, Springer, 2005, ISBN 0-387-25538-9

[2] — Keating M., Flynn D., Aitken R., Gibbons A., Shi K., *Low Power Methodology Manual for System-on-Chip Design*, Springer 2007, ISBN 978-0-387-71818-7

[3] — Synopsys, *Power Management Verification User Guide (for MVSIM, MVRC).*

[4] — Accellera, IEEE-P1801 http://www.accellera.org/activities/p1801_upf/

[5] — Flynn D., Gibbons A., “Design for Retention: Strategies and Case Studies,” SNUG San Jose 2008

[6] — Ram S., Tiwari, P., Thiriveedhi, A. “Challenges of Multi-Voltage Verification,” SNUG San Jose 2008

[7] — Bhairi P., et al, “Addressing the challenges in full chip power aware functional verification with MVSIM,” SNUG India 2008

[8] — Narasimhan, P. et al, “Voltage Aware Static Rule Checks for Power Managed Designs,” SNUG India 2008

[9] — Mukhopadhyay A. et al, “Establishing a methodology for early validation of multi-voltage RTL designs,” SNUG India 2008.

[10] — Kim N., Austin T., Blaauw D., Mudge T., Flautner K., Hu J., Irwin M., Kandemir M., Narayanan V., “Leakage current: Moore's law meets static power” IEEE Computer Vol. 36, Issue 12, 2003

[11] — Mutoh S. et al. “A 1v multi-threshold voltage CMOS DSP with an efficient power management technique for mobile phone applications” ISSCC1996, pages 168–169, 1996.

[12] — Zyuban V., Kosonocky S., “Low Power Integrated Scan-Retention Mechanism,” Proceedings of the International Symposium on Low Power Electronics and Design, August 2002, pp. 98 –102.

[13] — van der Meer, P.R. et al. “Low Power Deep Sub-Micron CMOS Logic – Sub threshold Current Reduction,” Springer /KAP 2004, ISBN 1-4020-2848-2A (Chapter 7 in particular, pages 106-111)

[14] — Biggs, J. Gibbons, A. “Aggressive Leakage Management in ARM-based Systems” SNUG Boston 2006

[15] — Flynn, D. Flautner, K. Patel, D. Roberts D. “IEM926: An Energy Efficient SoC with Dynamic Voltage Scaling” DATE 2004

[16] — Hattori T. et al, “Hierarchical power distribution and power management scheme for a single chip mobile processor,” Proceedings of the 43rd annual conference on Design automation (DAC), 2006, pp292-295.

[17] — Flautner K., Kim N.S., Martin S., Blaauw D., Mudge T.N, “Drowsy Caches: Simple Techniques for Reducing Leakage Power. ISCA 2002: 148-157

[18] — Allsup C., “Design for Low-Power Manufacturing Test,” EDA Designing, March 18, 2008.

[19] — Demler M.,”Power-Sensitive 65nm Designs Increase the Need for Transistor-Level Verification,,”” EDA DesignLine, 8/27/07

[20] — Dalkowski K., “Advanced Multi-Voltage Design Implementation” Elektroniktidningen, August 2007

[21] — Shi K., Lin Z., Jiang Y., “Practical Power Network Synthesis of Power Gating Designs,” EDA DesignLine, June 5, 2007

[22] — White M., “A Comprehensive and Effective Power Management Approach for Advanced System-on-Chip Designs,” Power Systems Design Europe, January/ February 2008

[23] — Jadcherla S., “Off by Design Architectures Curb Energy Waste” SCD source, March 25, 2008

[24] — Jadcherla S., “A ticking time bomb?” Chip Design, June 2008

[25] — Balachandran K., “Boost verification accuracy with low-power assertions,” EE Times, July 28, 2008

[26] — Balachandran K., “Static Checks for Power Management at RTL—Is this a case of “a stitch in time saves nine?,”” EDA Design Line, May 20, 2008

[27] — Balachandran K., “Voltage-aware simulation: No longer a fad, but a must for low-power designers “, EDN May 14, 2008.

[28] — Shirrmeister F., Thoen F., “Software development on virtual platforms Speeding time to market for low-power devices,” Electronic Products, August 2008

[29] — Schirrmeister F., “Software Driven Low Power Optimization for ARM Based Mobile Architectures,” ARM Developer's Conference, 2008

[30] — ACPI—Advanced Configuration & Power Interface, http://www.acpi.info

[31] — Krishna Balachandran, “Cover low-power design with constant analysis,” EE Times Online, October 27, 2008

[32] — Energy Star Program, http://www.energystar.gov

[33] — VMMCentral, http://www.vmmcentral.org

APPENDIX D NOTES

APPENDIX E ABOUT THE AUTHORS

E.1 AUTHORS

Following are the biographies of the authors of this book.

SRIKANTH JADCHERLA

Group Director of R&D, Verification Group
Synopsys, Inc.

Srikanth Jadcherla came to Synopsys as part of the ArchPro acquisition, where he was founder and CTO. Prior to ArchPro, Jadcherla was an IC designer and architect at companies such as WSI, Intel, Jasmine and Synopsys. He is a veteran of low power designs and pioneer of many energy efficiency techniques and principles. Jadcherla received an Intel Achievement Award for his work on low power and is the author of 12 patents. He is an honorary green evangelist and technical advisor to various companies ranging from solar energy suppliers to real estate developers. Recently, he has been advocating new paradigms in energy efficient design in semiconductor systems worldwide from both the supply and demand side of energy consumption.

Mr. Jadcherla holds a bachelor's degree in electrical engineering from IIT-Madras in India, and a master's degree in computational science and engineering from the University of California, Santa Barbara.

JANICK BERGERON

Synopsys Fellow

Janick Bergeron is a Fellow at Synopsys Inc. responsible for the development and specification of the functional verification methodology to be supported by their digital simulation products. He is the author of the best-selling “Verification Methodology Manual for SystemVerilog” and of the “Writing Testbenches” book series. Both are the first industry references on modern functional verification techniques and methodologies.

Mr. Bergeron holds a Bachelors degree in engineering from the Universite du Quebec a Chicoutimi, a Master of Applied Sciences in Electrical Engineering from the University of Waterloo VLSI program, and an MBA from the University of Oregon through the Oregon Executive MBA program.

YOSHIO INOUE

Chief Engineer
Renesas Technology Corp.

Yoshio Inoue is Chief Engineer of Design Technology Div. at Renesas Technology Corp., which was formed through the merger of semiconductor operations of Hitachi and Mitsubishi on April 1st, 2003.

He graduated BS degree at Tokyo Denki University. He joined Mitsubishi Electric as a gate array design engineer in 1984. Since 1989 he has been involved in advanced EDA design methodology development and EDA design systems to support the US's high-speed, high-complexity SoC designs.

When Renesas was formed, he focused more on RTL prototyping technology for Japanese customer's, and US customer's designs. Now he has expanded his area of focus to ultra low power design methodology such as application processors for cellular phones and is a pioneer in the area of hierarchical power management.

PROFESSOR DAVID FLYNN

ARM R&D Fellow
Visiting Professor, Southampton University

David Flynn, a Fellow in R&D at ARM Ltd., has been with the company since 1991, specializing in System-on-Chip IP deployment and methodology. He is the original architect behind the ARM® synthesizable CPU family and the AMBA® on-chip interconnect standard. His current research focus is low-power system-level design. He holds a number of patents in on-chip bus, low power and embedded processing sub-system design and holds a BSc in Computer Science from Hatfield Polytechnic, UK and a Doctorate in Electronic Engineering from Loughborough University, UK. He is currently Visiting Professor with the Electronics and Computer Science Department at Southampton University, UK.

E.2 ACKNOWLEDGEMENTS

The authors would like to thank many of our esteemed colleagues in our organizations and in our customer/partner companies for their valuable support, reviews, collaboration in bringing the book out. Some contributing authors have generously sent in material and narrated their experiences, which has enriched our content and we would like to thank them deeply for this effort.

A special acknowledgement goes out to Krishna Balachandran for making sure the contracts are all taken care of and for his extraordinary efforts in making this effort ready to launch.

Ghassan Khoory has been instrumental in keeping us paced and for enabling us in every possible way.

John Goodenough, Bryan Dickman and Joe Convey from ARM, Phil Dworsky, Alan Gibbons and Phil Morris from Synopsys for making our collaboration happen in time for this exercise and for their valuable technical perspective.

Our message has reached many people even prior to its formal publication and that is thanks to the efforts of Swami Venkat.

Manoj Gandhi, SVP and GM of the Verification Group at Synopsys has been a tremendous source of support and vision through out this process—without which an effort like this would not have been possible.

The Low Power Verification team at Synopsys led by Vinay Srinivas makes the software that supports this methodology happen and we are fortunate to have minds like Arturo Salz, Prapanna Tiwari, Yoichiro Iida, Prognya Khondkar, Puneet Jethalia, Sesha Sai Kumar, Debabrata Bagchi, Lalit Sharma, Harsh Chilwal, Vikas Gautam,

Sathyam Pattanam, Nadeem Kalil, Marc Edwards, Shankar Hemmady and their teams committed to being the leaders in this field. We have also been helped by the broader Low Power team at Synopsys across all business units—especially Mike Keating, Godwin Maben, George Zafiropoulos, Tom Borgstrom, Takaaki Akashi, Hitoshi Kurosaka, Josefina Hobbs, Jim Sproch, Larry Vivolo, Tom Chau and many others—our gratitude goes out to them.

We would like to thank Sesha Sai Kumar, Prapanna Tiwari, Vikram Malik and Ajay Thiriveedhi for helping out with testcase preparation, contributing material and training material. In addition, Vikram has patiently reviewed the document and helped out with the editing process. Mike Donlin and Edgar D'Souza(Collabis) deserve our thanks for editing what started out as fairly unreadable text!

We would like to additionally thank the following reviewers for their inputs either written or in oral discussions: Kelly Larson (Mediatek Wireless), Nobuyuki Nishiguchi (STARC), Hiroyuki Mori (STARC), Jianfeng Liu (Samsung), Ying-Chih Yang (Sunplus), David Wheelock (Seagate Technology), Brian Bailey (Brian Bailey Consulting), Sree Reddy (Nvidia), Kiran Puttegowda(Broadcom), Hillel Miller(Freescale), John Wei (Alchip), Yao Cong (Huwaei), Lily Jiang (Spreadtrum), Santosh Madathil (Wipro), Summer Yang (Realtek), Gary Delp (LSI Logic), Anand Raghunathan (Purdue University, ex-NEC Labs), John Biggs (ARM), Alan Hunter (ARM), Srivats Ravi (TI), Magdy Abadir(Freescale), Scott Runner (Qualcomm), Pidugu Narayana (Cypress), Partha Narasimhan (Cypress), Jerry Vauk (AMD), J.L.Gray (Verilab), Harunobu Miyashita (Fuji Xerox), Mutsumi Namba (Ricoh), Nobuyoshi Nakajima (Sony), Michael Hsieh (Sun), O. Kim (Silicon Image), Jim Crocker (Paradigm Works), David Hui (AMD), Phil McCoy (MIPS), Minaki So (Panasonic), Tomotoshi Nakamura (TJSys), Yuzo Kubota (Verifore), Bryan Dickman (ARM), Elango Rajasekharan (Connexion Semiconductor), Ed Huijbregts (Magma), Alan Gibbons (Synopsys), Will Chen (Synopsys).

The Marcom team at Synopsys, especially Josh Perkel, Paula Sterling, Sheryl Gulizia, Janet Berkman, Mital Poddar Lisa Rivera and Amy Timpe deserve many thanks for their unwavering and patient support to enable the book to go out.

Last but not the least, Rena C. Ayeras has done a fantastic job of cleaning up our follies in publishing. Many thanks to her for making this happen on time.

INDEX