Innovative 4-State Logic Emulation for Power-aware Verification

Kyoungmin Park[1], Brad Budlong[2], Hyundon Kim[1], Danny Yi[2], Jaehunn Lee[1], Chulmin Kim[1], Jaemin Choi[1], Kijung Yoo[1], Gibs Lee[2], Youngsik Kim[1], Yogesh Goel[2], Seonil Brian Choi[1]

[1]Design Verification Team, SLSI division, Samsung Electronics, Hwaseong, Korea
[2]Cadence Design Systems, Inc.
(km8228.park@samsung.com)

***Abstract*-This paper introduces an innovative 4-State logic ('0', '1', 'X', 'Z') emulation technology and illustrates it's application to an actual mobile-AP SOC for power-aware verification, marking a first in the verification industry. This 4-State emulation is highly useful for identifying power intent bugs and simplifies the debugging of 'X' sources. The improvements in emulation performance significantly enhance the overall verification turnaround time.**

Additionally, this paper presents unknown ('X' value) sources which are implemented in the emulator compared to the simulator, and outlines key requirements for the successful adoption of 4-State emulation. The paper also compares the results of the 4-State emulation experiment with traditional simulation-based verification and shows how much faster it is than 4-State logic simulation. This paper shares experimental data on the additional resources required for 4-State logic emulation compared to 2-State logic emulation.

I. INTRODUCTION

Power-aware verification, including UPF, typically takes more execution time than standard logic verification. Therefore, to improve the power-aware verification turnaround time, emulation could be a beneficial solution since it is faster than simulation and most emulator vendors support UPF.

Power-aware simulation depends on X propagation to display the effects of power. Using this, designers can identify bugs related to a missing isolation cell or any enable signal (e.g., isolation enable) which is not active and could cause the isolation cell to malfunction [1].

Furthermore, a significant debug issue in power-aware simulation is identifying the root cause of unknown (X's) values on various signals. There can be many reasons why X values appear on signals in power-aware simulations, ranging from incorrect UPF specification (missing isolation/level-shifting retention cells or improper power domain partitioning) [2].

However, traditional emulation methodology only supports 2-State logic ('0' and '1'), so 'X' and 'Z' values are not supported, even in power-aware verification. With this limitation, we might overlook some power isolation issues in power-aware emulation because a '0' or '1' value is propagated instead of an 'X' when using 2-State logic emulation. Sometimes, this non-intuitive emulation method makes debugging more difficult compared with an RTL logic simulator.

For these reasons, logic verification engineers use emulation only for power sequence checking, not for their full power-aware verification. Traditional emulator doesn't replace logic simulation in the low power verification area, even though emulation could reduce their verification turnaround time.

However, the size of mobile and automotive SoCs is increasing, and the development period is getting shorter. Even the number of power domains is rising as chip architects incorporate finer-grained control into chips and systems, adding significantly to the complexity of the overall design and verification effort [3]. Therefore, logic simulation is insufficient in this increasingly complex and challenging SOC verification situation.

II. 4-State Logic Emulation

In light of this challenging SoC verification situation, my team and emulator vendors have been discussing the development of 4-State logic emulation to improve power-aware verification for the past 10 years.

Finally, last year, 4-State emulation became possible. This paper describes how the 4-State logic emulation was developed, including the historical backgrounds. It shows the unknown ('X' value) sources that are implemented in the emulator in comparison with the simulator. The evaluation process with the flagship mobile SOCs is introduced, along with some requirements for successful 4-State logic emulation adoption drawn from our experiences. For result analysis, we share a runtime and compile time comparison with the logic simulation. Finally, a hardware resource comparison with the 2-State logic emulation is shown based on the experimental data.

A. Innovative 4-State Logic Emulator

Hardware emulation has long been implemented with 2-state logic, while the de facto standard for software-based simulation is 4-state logic. The use of 2-state logic for emulation was an accepted tradeoff for the performance gains that emulation offers, and it was thought impractical to implement 4-state logic in hardware.

The increasing adoption of low-power designs with increasing size and complexity is challenging the accepted idea that 2-state emulation is good enough. With the novel implementation of native 3-state (‘0’, ‘1’, ‘X’) computation in the Palladium Z2 processor along with additional compiler logic transformations to add ‘Z’ support, 4-state hardware emulation is now practical to implement and well suited for low-power verification.

The introduction of ‘X’ due to low-power handling is the primary focus of this paper, but there are multiple possible sources for ‘X’ in the simulation of a design. TABLE I shows the sources of ‘X’ from simulation and how they are handled in 4-state emulation with Palladium Z2.

TABLE I
4-State Emulation X-Handling

‘X’ Source	Simulation	4-State Emulation	Description
Power down event	X	X	This is the primary initial application for 4-state emulation.
Propagation	X	X	‘X’ values will propagate through the design. Simulators can support multiple models (LRM, CAT, FOX). Palladium 4-state emulation supports a localized CAT model based on the mapping of logic to the emulator.
Uninitialized signal	X	X	Uninitialized signals can either retain their ‘X’ value or can automatically be converted to 0 or 1 when swapping into emulation from simulation.
Uninitialized memory	X	X	Memories support 0, 1 and X, and can be left at their default ‘X’ value, loaded from a file, or set to a value when swapping into emulation from simulation.
Injection from RTL	X	X	An ‘X’ value can be explicitly assigned in RTL source. This can be used to represent an illegal condition such as a default condition of a case statement.
Force from TCL	X	X	The force command supports 0, 1 and X.
Port from simulation	NA	X	‘X’ is supported across the simulator / emulator boundary.
DPI argument	X	X	4-state types are supported with the standard Verilog aval/bval encoding.
Multi-driven net	X	X	Net resolution is done similarly to the Verilog simulator implementation, resulting in an ‘X’ if multiple driven sources have conflicting values.
Undriven net	X	X	Undriven nets have a ‘Z’ value and the ‘Z’ value is converted to an ‘X’ when used as a source to downstream logic.
Timing violation	X	NA	Emulation is done with a zero-delay model, so timing violations are not applicable.

The usage methodology for 4-state emulation retains the same features, compile flow and run-time operation as 2-state emulation. In the same way as all emulation flows, 4-state emulation operates on a synthesized representation of the design. The Palladium Z2 processor hardware introduces the ability for the processors to work individually to emulate 2-state operations, or in pairs to perform the logically equivalent functionality with 4-state logic. Due to the usage of a processor pair, the emulation resources required for 4-state calculations must be at least two times that of 2-state calculations.

B. Evaluation with mobile-AP

This paper describes the evaluation journey and the issues we encountered during the evaluation period, then summarizes the requirements for successful 4-State logic emulation adoption.

Firstly, the capabilities of 4-State logic emulation with various 'X' situations were confirmed by checking the state corruption due to UPF power down, X propagation, and uninitialized signals through an example design.

Figure 1 and Figure 2 are snapshots from the 4-State emulation waveforms. Figure 1 demonstrates the state corruption due to UPF power down and the 'X' state for the initial values until reset. The waveform accurately shows the unknown state, making it helpful for debugging power intention with UPF.

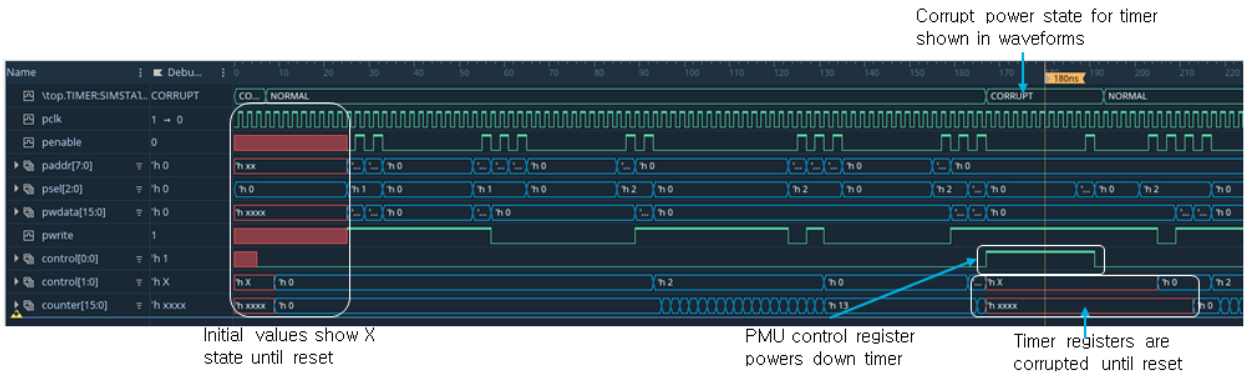

Figure 1. State corruption due to UPF power down

Figure 2 displays X propagation for small 'counter' logic. From the beginning, forced X values to 'counter[3]' and then X values are propagated to the upper bits. 4-State emulation also supports X propagation situations. This feature will assist in finding the root cause of 'X' through the waveform.

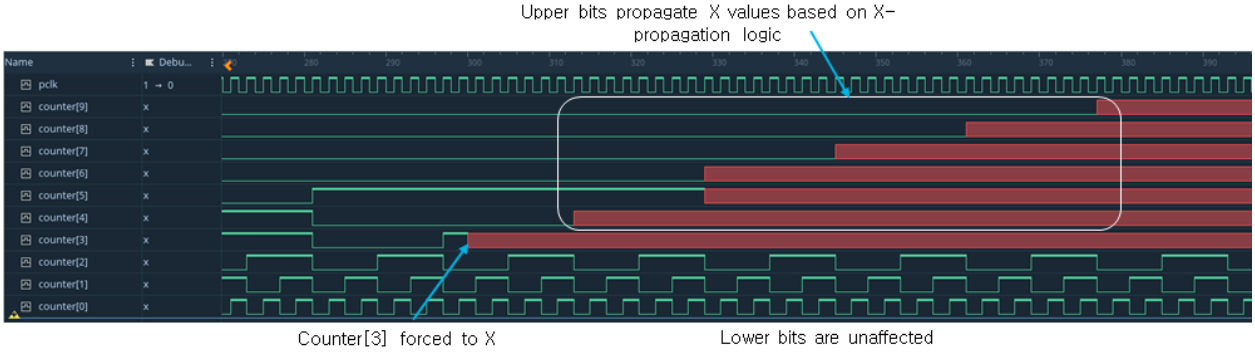

Figure 2. X Propagation

Then the main evaluation with the full mobile-AP SoC design was started. The target design has over 20 different power domains and the target evaluation scenarios are system power gating cases.

Based on the evaluation, the following requirements are needed for successful 4-State logic emulation.

1) *Power aware synthesizable (for emulation) PHY model*
   - All power features should be implemented fully in the model including power pins and the model should be working with UPF at the same level as the simulation model. In most cases, PHY models for simulation have many fully featured power aware models, but most of them are not synthesizable code, so they are not for emulation environment.
   - For example, in the DRAM PHY model, there were issues with DRAM access because all power features were not implemented exactly, and this occurred in a system power gating scenario where the DRAM controller was powered off. After waking up from the system off, DRAM access was not possible, which was an issue caused by the lack of retention handling in the DRAM's RESET.

2) *PAD MUX implementation for taking care of high 'Z' and 'X' values*
   - For example, bidirectional PAD signals should be considered with 'Z' value in 4-State emulation. The following 'i' assignment RTL coding style is working properly in 2-State emulation environment due to 2-State emulation doesn't care high 'Z' cases, but it might not work properly in 4-State emulation which supports high 'Z' cases. It should be fixed to support high 'Z' cases like 'ii'.

     i. assign PAD = (unknown === 1'bx) ? 1'bx : PAD_i; // might not working properly

     ii. assign PAD = (unknown === 1'bx) ? 1'bx : (PAD_i === 1'bz) ? 1'bz : PAD_i; // it is synthesized correctly

3) *Removal of Dynamic RTL(emulation debug feature)*
   - Dynamic RTL is one of debug feature in emulation, dynamic RTL model is not included in static compile time, but it can be added to just before runtime, so it is useful to implement debugging modules without additional compilation time. But dynamic RTL is not supported with 4-State emulation, so any monitor or debugging logic which is implemented with dynamic RTL, needs to move to a static RTL implementation.

4) *Handling pull-up/pull-down for 'Z' state net*

5) *Initialization un-driven nets in both a test-bench and an emulation model*

6) *SRAM/ROM code preloading*
   - Preload to avoid unexpected 'X' propagation after accessing an un-initialized memory region. In a 2-State emulation environment, emulation rarely stops because the memory output value is 0 or 1 even if the memory is not initialized. However, in a 4-state environment, if the memory is not initialized, the memory output value 'X' appears and this value is propagated, making debugging very difficult.

C. Results

The result shows how much faster 4-State logic emulation is than logic simulation in terms of performance, and also how much more accurate it is than the existing 2-State logic emulation in terms of power-aware verification with full support for power-aware features using UPF.

First, let's look at it from a performance perspective. Figure 3 shows the run time and compilation time comparison for two different SOC design (Design A and Design B).

The emulation compile time for Design A is 16% more than simulator, but the run time performance is improves by 115 times.

Design B is a larger chip and target power scenario is more complicated and longer than that of Design A. The compile time gap is wider but the run time gain is smaller than Design A. This result appears to be due to physical factors such as hardware emulator resources and emulator driver clock frequency.

In the full-chip SoC power-aware simulation, most power scenarios take over one or two days of simulation time (wall clock). However, this innovative 4-State logic emulation allows for results 30~115 times faster than simulation, depending on design size and test scenario.

Regarding compilation time, emulator compilation has traditionally taken longer than simulation because hardware-based emulators require design synthesis to run. Thus, the compilation time takes 1.16~1.7 times longer than that of the simulator.

Since each power scenario typically runs more than ten times to complete verification, these performance improvements significantly help to enhance overall verification turnaround time.

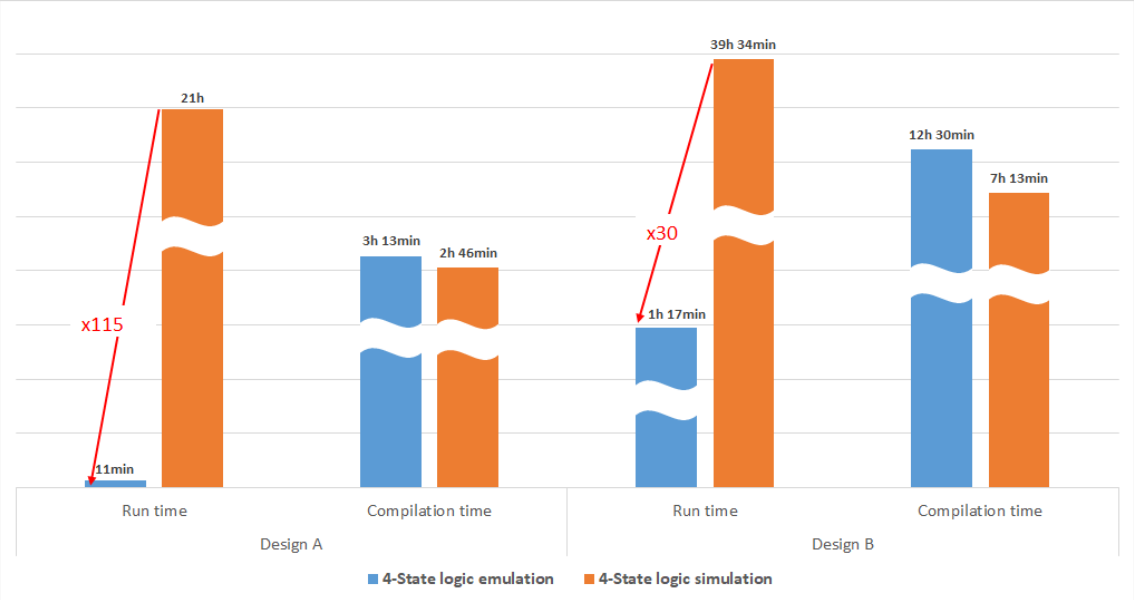

Figure 3. Run and Compilation time

In terms of accuracy of power aware verification, this paper shares two experiences.

First, there was an issue with 'X' occurring after power down in a ROM that always needs to be powered on, which was actually caused by a missing liberty cell for the ROM, which did not properly reflect the PG pin information and was connected to wrong power supply net, causing the power to actually go down and the ROM to fail to keep data, resulting in an 'X' appearing at the ROM's output and propagating.

Another one is the power switch control signal seen as X in corrupt power domain. In this case, the tool interpreted the wrong interpretation of set_port_attributes -driver_supply/-receiver_supply described in the UPF, leading to the power switch signal being treated as 'X'. Both of these phenomena did not occur in 2-State emulation.

D. Resource Analysis

In this section, this paper presents experimental data on the additional resources required for 4-State logic emulation compared to 2-State logic emulation. The primary factors we tracked are compiled capacity, compile time, and performance. TABLE II employs ratios relative to a base 2-State without UPF configuration to clearly illustrate the impact.

'Emulation capacity' refers to the volume of resources utilized by the emulator hardware. 4-State emulation requires approximately 2.3 times more resources than 2-State emulation.

Adding UPF increases overall resource usage by approximately 3.2 times compared to the default configuration. However, this is only 2.3 times compared to the 2-State UPF configuration. The extra resources needed, along with the additional processing of UPF constructs, results in an extended compile time.

The initial emulation speed is influenced by both the addition of UPF and the introduction of 4-State logic.

TABLE II
RESOURCE ANALYSIS

Mode	Relative Cost	2-State w/o UPF	2-State w/ UPF	4-State w/o UPF	4-State w/ UPF
Emulation capacity	vs. 2-State	1	1.4x	2.3x	3.2x
	vs. 2-State w/ UPF		1		2.3x
Compile time	vs. 2-State	1	1.5x	1.5x	2.1x
	vs. 2-State w/ UPF		1		1.4x
Performance	vs. 2-State	1	0.75x	0.9x	0.59x
	vs. 2-State w/ UPF		1		0.81x

The results of this experiment represent initial data, without any optimization applied. Precise profiling for capacity, compile time and performance will be planned to optimize each factor. It may be beneficial to use a hybrid form from where the verification target block is set to 4-state and the remaining blocks are set to 2-state.

In addition, further work will include checking waveform dump time and conversion time, both essential for debugging, in this experiment. Traditionally, it takes a long time to dump the waveform in emulation and convert it to the format used in debugging, which presents a huddle in the debugging turnaround time. By selecting useful features related to these waveforms not only in a 4-state environment but also in a 2-state environment, we should be able to shorten the overall verification time.

III. CONCLUSION AND FURTHER WORK

This paper introduces the innovative 4-State logic emulation technology for power-aware verification and presents the evaluation result through an actual mobile-AP SOC project for the first time in the verification industry. The 4-State emulation is highly useful for identifying power intent bugs, such as missing isolation cells, and makes it simplifies the debugging of ‘X’ sources.

This paper demonstrates which ‘X’ sources are implemented in 4-State emulation, which is almost equivalent to 4-State logic simulation. Therefore, most power scenarios can be run in this 4-State emulation environment and verification turnaround time(TAT) will be significantly improved.

Drawing from evaluation experience, this paper presents key requirements for the successful adoption of 4-State emulation. Most of these requirements revolve around a full-featured power-aware model for emulation that takes into account the ‘X’ and ‘Z’ of the PHY and PAD models. Additionally, care must be taken with un-driven nets, pull-up/pull-down handling, and memory initialization.

Experimental results reveal that 4-State emulation is significantly faster than simulation, approximately 30~115 times faster, depending on design size and scenario complexity. This resource analysis can be very beneficial to anyone planning to adopt the 4-State emulation methodology.

This paper has so far focused on implementing a 4-State logic emulation environment and low power verification. Therefore, in this experiment, compilation time and performance analysis have not yet been conducted. As this paper shows in the resource analysis section, only experimental data for compilation time and performance are shared. Afterward, detailed profiling for each resource item will be planned, and then an optimization approach will be properly applied to improve compilation and performance.

For further work, an in-depth analysis of debugging features for power intent, including waveform, will likely be necessary. To provide an environment almost identical to the one verification engineers previously used in the simulation environment, various convenience features must be supported.

ACKNOWLEDGMENT

We would like to express our appreciation to our team members who assisted with the debugging, especially in finding the 'X' root causes, and to the PHY modeling team who implemented the synthesizable emulation PHY models. We would also like to thank the Cadence members who collaborated to enable this methodology.

REFERENCES

[1] Himanshu Bhatt and Shreedhar Ramachandra, "Efficient Low Power Verification & Debug Methodology Using Power-Aware Simulation", Semiconductor Engineering, December 2018

[2] Durgesh Prasad, Madhur Bhargava, Jitesh Bansal, and Chuck Seeley, "Debug Challenges in Low-Power Design and Verification", DVCon US, 2015

[3] Ann Mutschler, "Power Domain Implementation Challenges Escalate", Semiconductor Engineering, June 2022