2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

An Accelerated System Level CPU Verification through Simulation-Emulation Co-Existence

Ruchi Misra, Samridh Deva, Sai Krishna Pallekonda, Alok Kumar, Garima Srivastava,
Samsung Semiconductors India R&D, Bangalore,
ruchi.misra@samsung.com, samridh.deva@samsung.com , p.saikrishna@samsung.com ,
kumar.alok@samsung.com, s.garima@samsung.com

Youngsik Kim, Seonil Brian Choi,
Samsung Electronics, Korea,

ys31.kim@samsung.com, seonilb.choi@samsung.com

***Abstract*— The most significant hurdle currently for design verification is the very long simulation time for functional and low-power scenarios of complex designs at the SoC level. This huge simulation time arises due to the addition of multiple heterogeneous processor cores, graphics processors, accelerators, several peripherals, DDR, etc., and makes iterative debugging a time-consuming and sometimes inefficient task. While simulation is suitable for a software arrangement that tries to mimic all the actual conditions, emulators are one step closer to this aim, with almost the whole target device virtualized. Hardware Emulation is the technique of prototyping real SoC design and accelerating the speed of design execution. The methodology presented in this paper has the potential to be scaled to various platforms across different designs like Automotive SoCs, Mobile SoCs, and Wearable SoCs, along with its uniqueness and complexity. In order to uncover bugs related to Low-Power functionalities, testing involves traversing through multiple Power-up, and Power-down sequences, and hence these scenarios are best suited for testing on emulators. Some of the other long-running cases that could potentially reap the most benefit are IO Coherent scenarios which involve real RTL of GPU or other Multimedia blocks along with real CPUs. Hence, this paper targets bridging the gap between Design and Time-to-market using co-existing simulation and emulation methodology and walks through some novel approaches that cover stress areas of functional verification.**

I. INTRODUCTION

Hardware-assisted verification, or emulation, delivers the capacity and performance for extremely fast, full System-on-Chip testing. Emulation enables testing of a large number of long-running test cases in a fraction of the time consumed by simulation. In doing that, it also allows more design requirements to be covered while more bugs are uncovered. For the purpose of this paper, we have done extensive testing of functional scenarios spanning from Coherency to Dynamic voltage and Frequency Scaling to Power Aware runs of various low power scenarios which are very critical for a competitive SoC in these times. Since Emulators are running at actual clock speed and each of its hardware components (FPGA, on-chip memory, and off-chip memory) are running at megahertz speed, the throughput of design becomes really high, and the overall verification life cycle is sped up.

Figure 1 below shows a high-capacity system emulator with the hardware blocks associated with it. In the Server, the Design Under Test (DUT) is mapped onto one or several FPGAs and memory chips. The test environment is mapped onto dedicated FPGAs (interface FPGAs) which implement the Hardware Test Bench. The Emulator Server unit is connected to the host PC through a high-speed interconnection board.

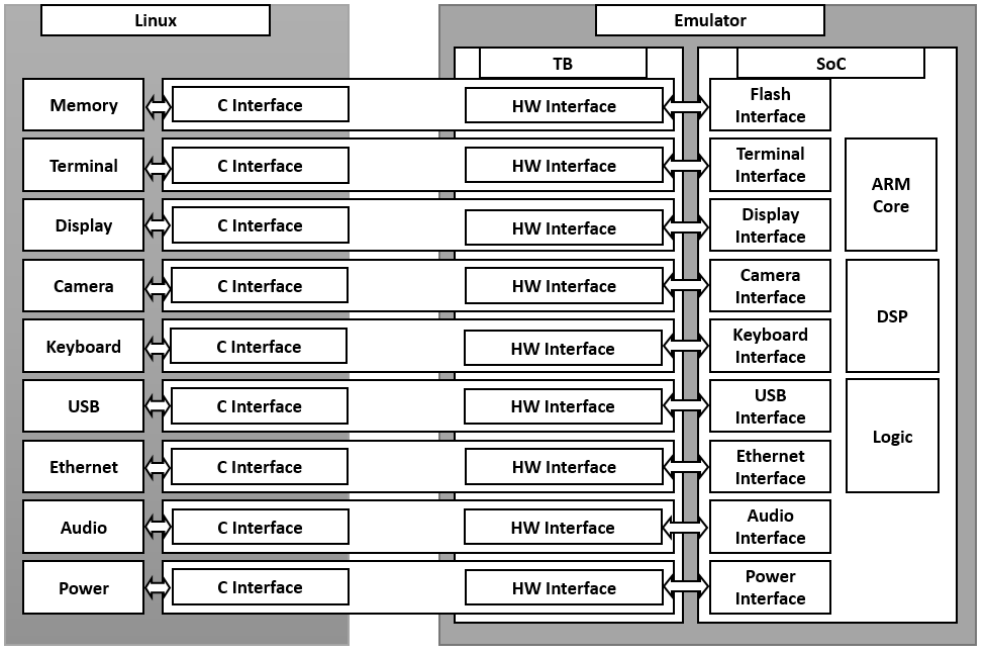


Figure 1 : Hardware of Emulation Platform

II. EMULATOR COMPILATION

The compiler as depicted in Figure 2 starts with the original Register-Transfer Level (RTL) code of the design and automates all of the necessary steps to synthesize and map the design for a given emulator hardware configuration. The emulator compilation has 4-stages: Front-end, system-level back compilation, Core-level backend compilation and FPGA Place and Route. The first stage Synthesizes RTL source files to generate gate-level netlists. System-level back-end compilation splits extra-large netlists into medium sized netlists and performs system-level place and route and static timing analysis generating the appropriate databases for runtime environment. The compilation maps medium-sized netlists onto Emulator server and FPGA Place and Route creates the binary files for the FPGAs. The choices when creating the compilation project are important because a correct compilation project makes design verification easier. An optimized compilation balances compilation efficiency and emulation performance.

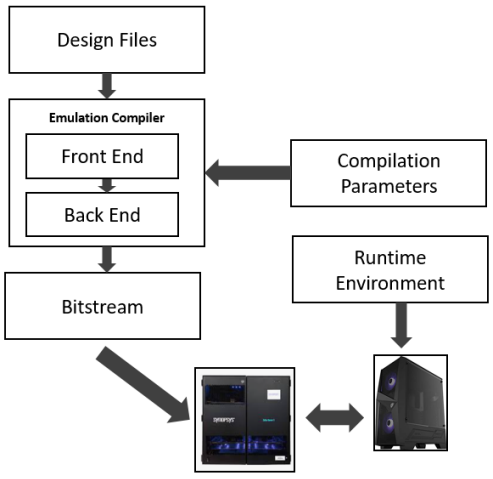


Figure 2 : Emulator Compilation Flow

III. RELATED WORK

Power-aware Verification of SoCs has been around for over a decade ever since IEEE released the UPF standard. Early works such as [1],[2],[3] laid the foundation for simulation-based power-aware verification of Systems-On-Chip, since then, advancements were made by industry partners to streamline UPF verification flows. Today, we not only have multi-core SoC designs but also multi-cluster SoCs with dual architecture processors

implemented on the same die. Needless to say, this increase in complexity due to SoC architecture evolution has seeped into the requirements of power-aware verification and is fast becoming the long-pole of the verification lifecycle. To showcase the results obtained in this paper, we divided the power-aware verification requirements into a number of test categories and successfully accelerated the power-aware verification lifecycle with the use of emulators by incorporating UPF design libraries and power domain control into the design images used for emulation-based verification. This has resulted in up to 40x savings in time and resources.

IV. FUNDAMENTAL METHODOLOGY

A. Test Planning and Test Priority

A diligently made test-plan gives greater predictability, more aggressive innovation and handles late-stage spec changes. Keeping this in perspective, we prepared our test-plan in such a way that the critical and long running functionalities like low-power scenarios, random stress traffic etc. are targeted to run on Emulation platform in order to give faster results. Due to certain limitations in emulation such as missing checkers or difficulty in debug, such cases could be recreated on simulation easily. During the verification cycle, the test plan will usually go through several reviews and needs to be approved prior to execution.

After the test plan is completed, it would go through test prioritization. Setting the test priorities is not really a simple task and great responsibility lies on getting the priorities right since the priorities set the order of the test plan execution and debug. Our aim is to categorize the tests and their priority such that verification milestones are not blocked due to the run time of some of the important scenarios.

B. Migration to Emulation

Initially few basic sanity tests are run in emulation and it is carried out via a step-by-step process where the set-up that needs to be done before the actual emulation run is brought up. Once the scenarios are identified as emulation closure items, their C image files are generated after compile similar to simulation of C code. As per the steps shown in Fig.3, the verification engineer is needed to identify which functional scenario needs to run in which environment. Once identified, a variety of mixed functionalities are run on both simulation and emulation for comparison of the run time. Starting with the desired scenarios which need to be run, we compile the C test and generate the image file which can be either run on full image or base image depending on the blocks involved. Using certain scripts to generate or separate the binary file with design data and using .tcl file for rest of the setup changes, the run is launched on Emulation Platform.

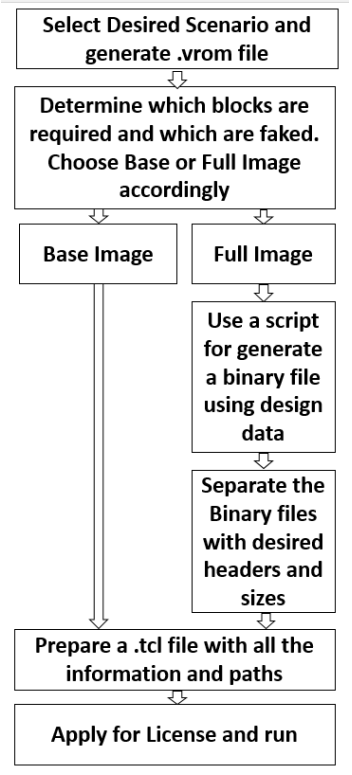


Figure 3 : Basic Steps of migrating a simulation run to emulator

C. Enhancements Done

In addition to migrating the complex scenarios of SoCs to Emulation, this paper also targets to automate the process of migration in order to improve the efficiency. As shown in Fig. 3, the separation of binary files in case of full image, application for license and the start and stop of waveform dump, all require involvement of the user. All these steps are automated to make it into a single input, single command scheme.

V. COMPLEX COHERENCY SCENARIO ON EMULATOR

Understanding the fact that the processor vendor may not provide coherent connection mechanism for multiple clusters and system level Cache coherency is fundamentally had to verify, we tried to create a complex coherency scenario at SoC level since we are connecting multiple clusters together and each core in each cluster will have their own private L1, private/shared L2 caches and shared L3 caches. In addition to this, there can be a system level cache or last level cache in the system, hence adding thorough requirement for coherency testing at full-chip.

We successfully created a stress scenario mixing varying traffic such as DVM, exclusive accesses, IO coherent transactions etc which is an ideal candidate for emulator run. We also varied page table properties and memory ordering in these scenarios and created multiple cache-line evictions.

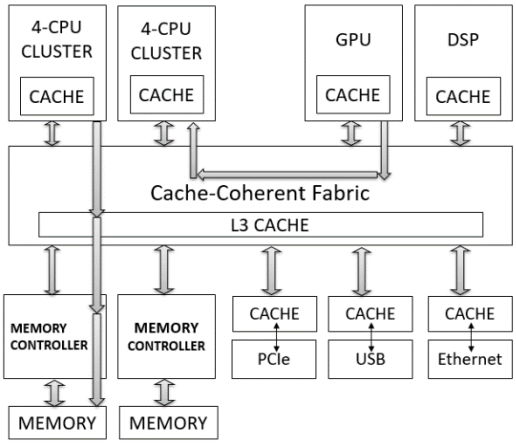


Figure 4 : Typical Complex SoC and a Complex Coherency Scenario

VI. DYNAMIC VOLTAGE AND FREQUENCY SCALING SCENARIO

We all know that DVFS provides ways to reduce power consumption of chips on the fly. Different design Blocks at SoC work on different frequency and verification team has to traverse through all the voltages and frequencies for testing purpose. But at the same time its important to note that testing/scaling of frequencies for memory interface, cores, coherent interconnects etc. takes huge simulation time. Hence we created and ran this scenario on emulator platform in order to gain significant confidence in our DVFS tests.

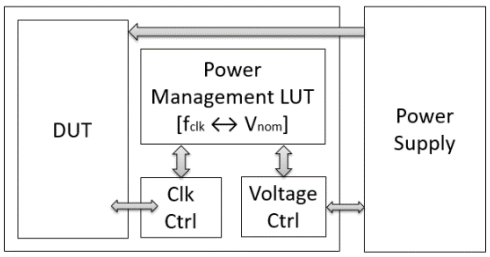


Figure 5 : Dynamic Voltage and Frequency

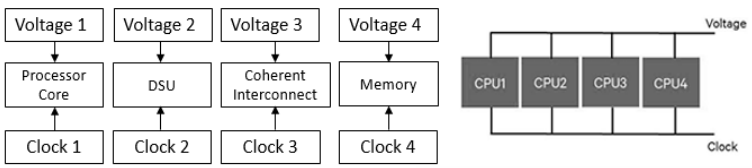


Figure 6 : Different blocks of SoC operating on different Voltage/Clock

VII. LOW POWER SCENARIOS

We tried to run various low power scenarios at Emulation level like Core/Cluster Clock Gating schemes, Memory Retention schemes as well as complete power down and sleep scenarios for various blocks of SoC. The Figure 7 on the right shows different processor states possible and the waveform in Figure 8 shows a typical Power Down scenario for Cores and Cluster which can be verified using emulator in an efficient manner. These low power scenarios are also run in Power-Aware setup with UPF incorporated. The entire methodology to enable Power-Aware runs is explained in upcoming sections of this paper.

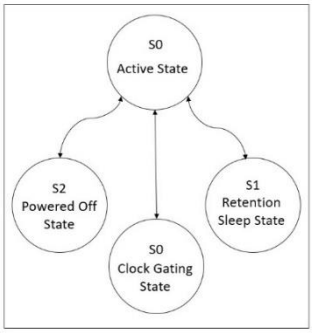


Figure 7 : Power State transitions

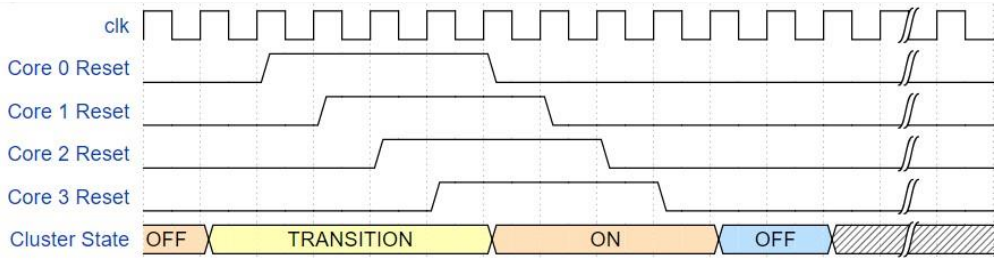


Figure 8 : Power Down modes for Cores and Cluster

VIII. UPF ENABLEMENT AND COMPILATION FLOW

A. Enabling UPF based Power Aware Runs on Emulator

This methodology ensures that all synthesizable design components are re-used between PA Simulation and PA Emulation Design Targets thus enabling consistency of testing between PA Simulation and PA Emulation test environments. This means now we can use both platforms to complement each other in the Verification lifecycle with a high degree of confidence. Further in the paper, we shall see how we have efficiently offloaded long running power aware scenarios to the PA Emulation environment to bring down the testing time to a fraction of what it was in the PA Simulation environment.

The topology of the power network is described by the UPF script for the design and is an input to the compiler. the emulation compiler chosen supports the following standards of UPF (Unified Power Format) versions:

- UPF 1.0 and UPF 2.0 (IEEE 1801-2009 standard)
- Limited set of commands from UPF 2.1 (IEEE 1801-2013 standard)

The compiler chosen also supports both hierarchical and non-hierarchical UPF, hence inside of the UPF files, hierarchical paths can be declared and applied.

```
#settign design attributes such as nwell and pwell bias
set_design_attributes -elements {.} -attribute enable_bias true

#create top supply ports
create_supply_port VDD      -direction in
create_supply_port VSS      -direction in
create_supply_port VVDD_blk -direction in

#power domain
create_power_domain pd_blk  -elements {.}

#create supply nets at top level
create_supply_net VDD
create_supply_net VSS
create_supply_net VVDD_blk -resolve parallel

#create supply nets at child level
create_supply_set pd_blk_primary -function {power VVDD_blk} -function {ground VSS} -function {nwell VDD} -function {pwell VSS}
create_supply_set pd_blk_aon     -function {power VDD}      -function {ground VSS} -function {nwell VDD} -function {pwell VSS}

create_power_domain pd_blk             \
-supply {primary pd_blk_primary}       \
-supply {default_isolation pd_blk_aon} \
-supply {default_retention pd_blk_aon} \

#create power switches
create_power_switch pgsw_blk                       \
-domain             pd_blk                         \
-input_supply_port  {VDDG VDD}                     \
-output_supply_port {VDD VVDD_blk}                 \
-control_port       {SLEEP PWRDWN_BLK}             \
-ack_port           {SLEEPOUT PGTESTOUT_BLK {SLEEP}}\
-on_state           {on_pgsw_blk VDDG {!SLEEP}}    \
-off_state          {off_pgsw_blk {SLEEP}}         \
```

Figure 9 : A Sample UPF code describing power intent of the design

B. Compiling a PA Emulation Design Target

The first step is to do Makefile changes and identifying compilation options that need to change for running low-power scenarios. While compiling, we need to fix any errors that may come due to any power related ports for any IP instances. We also need to ensure the availability of complete list of UPF libraries to be included. Next step is to use the available utility to convert present libraries to Emulation compatible libraries. After that we can run lib to DB conversion script and convert libraries to the format needed. After all this, once we fix all the UPF compilation issues related to version or any specific tool option, we can run our basic single/multi core power down scenarios to verify that the intent of the test has been achieved.

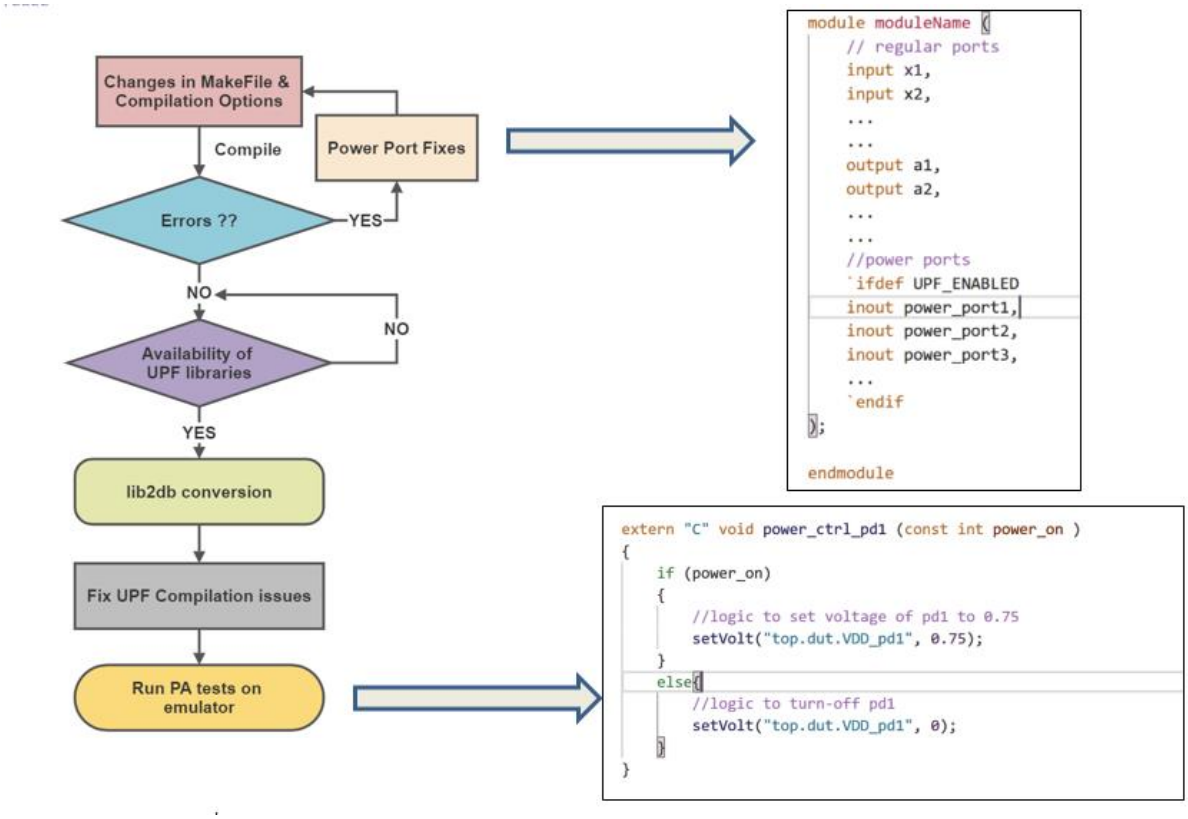


Figure 10 : Methodology of running PA runs on Emulator

C. Power Aware Compilation Flow on Emulator

As shown in the Figure 11 below, both UPF files and RTL files are fed to the EDA tool which understands the power intent of the design through UPF files and functional intent from design files. After the process of analysis, elaboration and splitting, and then synthesis, EDIF files are produced which is a vendor-neutral file format used to store netlists.

The power-aware emulation model compilation flow presents us with the following advantages:

- Common semantics and analysis between simulation and emulation models/targets.
- Similar UPF constructs are supported between simulation and emulation environments.
- Early error flagging capabilities of the simulator tool are leveraged.
- Precedence Rules and implementation carried over from simulation target.
- Debug continuity across PA-Simulation and PA-Emulation.

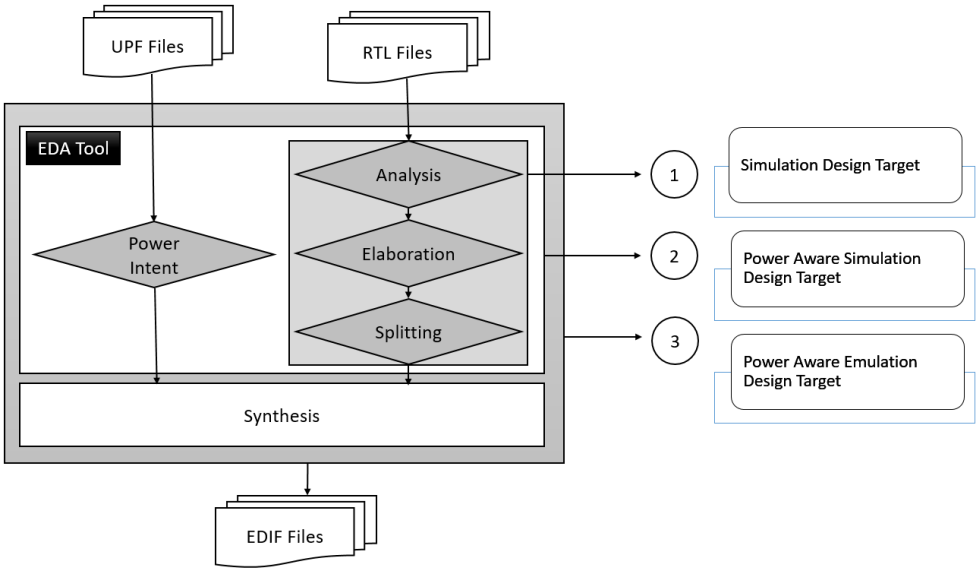


Figure 11 : Unified Compilation flow for Power Aware Run

D. Important Commands and APIs used for UPF enablement

Figure 12 below illustrates the important UPF commands that are leveraged from PA-Sim target's power intent.

UPF Command	Description
create_power_domain	used to declare power domains
create_supply_port	used to declare power supply ports at top level
create_supply_net	declares nets to deliver power from supply ports
connect_supply_net	connects supply nets to ports and other supplynets
create_supply_set	creates supply nets at child level
create_power_switch	used to declare various power switches in the design
set_isolation	sets the isolation strategy of ports for a given power domain
set_retention	sets the retention strategy of ports for a given power domain
set_design_attributes	used to set various power attributes of design at top level for each block
set_port_attributes	used to define port behaviour and attributes like isolation, retention etc.

Figure 12 : List of important UPF Commands and their usage

The Emulator also provides a set of C++ APIs through the Class "PowerMgt" that seamlessly enables Power Aware Verification on emulator. In the below table, some of the important members of the PowerMgt Class are discussed.

C++ methods	Description
Init	starts power aware verification
Enable	enables power aware verification
IsPowerManagementEnabled	checks if the design is compiled to support power aware verification
getListOfDomains	Returns a list of all power domains declared in the Power Management UPF script
getPowerDomainState	Returns the state of a power domain
isRetentionStrategyEnabled	returns info about retention strategy
enableRetentionStrategy	enables a retention strategy
disableRetentionStrategy	disables a retention strategy
setPowerDomainOn	switches a power domain ON
setPowerDomainOff	switches a power domain OFF
setPowerDomainState	switches a power domain to the given state
releasePowerDomain	releases testbench control of the power domain
setForceMode	defines behaviour of forces regarding power domain states

Figure 13 : Important C++ API's used during PA run

IX. COMMON ERRORS FACED AND THEIR SOLUTION

A. *Clash of Rules and Design Attributes amongst various IPs:*

One of the most frequent errors that we faced during model compilation is the enabling/disabling of power well biases varying from one IP to another and clashing with the power bias enabled/disabled rule at the SoC level. To resolve this, we had set enable_bias to false for all IPs and the SoC top.

```
Error- [UPF_ATTR_VALUE_MISMATCH] Attribute value different from parent
/disk/path/to/ip/A/ipA_top.upf, 20
Attribute 'enable_bias' has been specified in an ancestor hierarchy as
'false'. Cannot specify attribute as 'true' in scope
```

Figure 14 : Most frequent error during model compilation

B. *Corrupt cell libraries:*

Cell libraries for various IPs have vastly different release schedules and hence during integration, sometimes we end up with stale and corrupt lib paths which need to be updated manually.

```
Error- [CFCILFBI] Cannot find cell in liblist
/disk/path/to/ip_A/design/files/RTL/ip_A_prj_<size>_ram_<attr>.v, 162
Cell 'ip_A_<process>_<size>_rtl_top' cannot be found in liblist for
binding instance
`top.dut.<ip_A_soc_hierarchy>.cell_name
Liblist: WORKLIB_ip_A DEFAULT
```

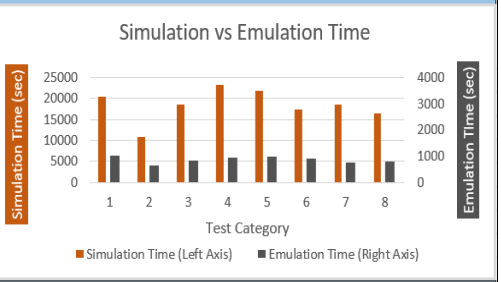
Figure 15 : Error due to Corrupt Cell Libraries

X. RESULTS AND CONCLUSION

1. This paper demonstrates how multiple long-running scenarios can be smoothly ported onto an emulation platform and is much easier and faster than simulation.
2. The experiments were carried out for several functional and low power scenarios along with simulation for the same scenario on same design drop for comparison.
3. The run times were recorded and analyzed. The corresponding plots are shown in Figure 16 as an indicator of the results.
4. By using this method, we significantly reduced the number of iterations needed, debug time, and the verification schedule. Current results are indicative and based on two emulator platforms from vendors like Cadence and Synopsys however with latest version platforms, the benefits observed are further more.
5. We also narrowed down the areas of critical bugs since we specifically aim to find relevant bugs early through this environment and not the complete coverage.
6. Further enhancing the process by adding automation, verification engineer can maneuver between the two environments very easily as per the requirement.
7. The results achieved in some of the Exynos Mobile SoCs and Automotive SoCs were demonstrative of the fact that we saved at least 50X time in closure of certain critical long pole features.

CADENCE PLATFORM

Test Category	No. of clock cycles Millions	RTL Simulation Time (sec)	Emulation Run Time (sec)	Speedup %
Category 1	8.641	20529	1020	95.0314
Category 2	5.182	10715	645	93.9804
Category 3	6.526	18654	840	95.4969
Category 4	7.951	23279	960	95.8761
Category 5	7.375	21782	971	95.5422
Category 6	7.250	17458	911	94.7818
Category 7	6.404	18580	765	95.8827
Category 8	5.530	16381	785	95.2079



SYNOPSYS PLATFORM

Test Category	No. of clock cycles Millions	RTL Simulation Time (sec)	Emulation Run Time (sec)	Speedup %
Category 1	9.199	20492	307	98.5019
Category 2	9.056	23235	288	98.7605
Category 3	7.374	18065	322	98.2175
Category 4	8.641	20529	331	98.3876
Category 5	6.526	18654	268	98.5633
Category 6	7.951	23279	275	98.8187
Category 7	7.375	21782	273	98.7467
Category 8	7.374	18432	275	98.5080

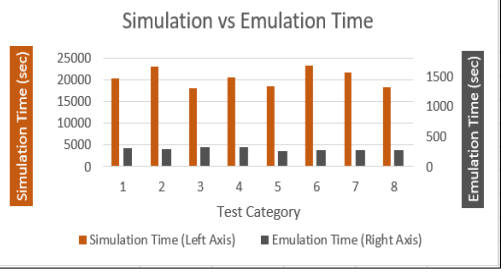


Figure 16 : Non-Power Results Across Two Platforms

Test Category	RTL Simulation Time (sec)	Emulation Run Time (sec)	Speedup %
Category 1	116877	2417	97.9320
Category 2	119090	2733	97.7051
Category 3	70703	4027	94.3043
Category 4	267216	7782	97.0877

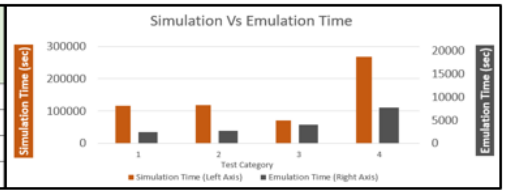


Figure 17 : Power Aware Run Results on Synopsys Platform

I. REFERENCES

[1] F. Bembaron, R. Mukherjee, A. Srivastava, “Low Power Verification Methodology using UPF”, in Conference on Electronics Systems Design and Verification Solutions, DVCON 2009

[2] C. Trummer, C.M Kirchsteiger et.al., “Simulation-based power aware verification of systems-on-chip designs using UPF IEEE 1801”, NORCHIP 2009

[3] J. Liu, M. Hong, B.H. Lee, J. Choi, H. Won, K. Choi, H. Vardhan, A. Kher, “Low power verification with UPF: Principle and Practice”, in Conference on Electronics Systems Design and Verification Solutions, DVCON 2010