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

A NOVEL AND EFFICIENT METHODOLOGY TO EXPEDITE COMPLEX SOC DV CLOSURE BY LEVERAGING MODULARLY ARCHITECTURED SCALABLE ENVIRONMENT

Vinay Swargam, Senior Staff Engineer, Samsung Semiconductors India Research(SSIR), Bangalore, India (vinay.s9@samsung.com)

Guttapalem Yatisha, Associate Staff Engineer, Samsung Semiconductors India Research, Bangalore, India (g.yatisha@samsung.com)

Ayush Agrawal, Senior Engineer, Samsung Semiconductors India Research, Bangalore, India (ayush11.a@samsung.com)

Sriram Kazhiyur Soundarrajan, Associate Director, Samsung Semiconductors India Research, Bangalore, India (sriram.k.s@samsung.com)

Somasunder Kattepura Sreenath, Director, Samsung Semiconductors India Research, Bangalore, India (soma.ks@samsung.com)

***Abstract*— Complexity of System-on-Chip is increasing exponentially with the advancements in artificial intelligence, machine learning and IoT technologies. With the diminishing dependency on the software to take control of the hardware, the complexity of the hardware is increasing further. System on Chip (SoC) designs incorporate more and more Intellectual Property(IP) to cater the ever growing market requirements. These complex requirements culminate in increasing the run time of simulations by many folds. With time to market being a significant factor, any delay due to run time or reruns need to be eliminated. One of the most effective solutions to scale down the simulation run times is by targeting the tests and coverage at sub system level and at the same time there is a need to bridge the gap between what is covered at sub system and SoC. This paper canvasses a strategy for resolving and reusing components built for Peripheral Component Interconnect Express(PCIe) and Universal Serial Bus(USB) at lower abstract level and how it is leveraged to close the design verification at higher abstract level like SoC. The concept of using sub system environment has been in existence for long, but how to intelligently plan for the modular component hook up and how to visualize the reuse of these components at different higher abstract level with minimal update to drastically reduce the development and simulation time is something which resulted in saving run time by 59x and man months by at least 3 months. Upon quantification of computing cost, more than 26K hours of run time and 31K precious license cost were saved per sub system.**

Keywords: Subsystem, SoC, DSNR (Dynamic Save and Restore), CNR (Capture and Replay), PCIe, USB, Simulation Speed, Coverage

I. INTRODUCTION

The SoC in the current context consisted of more than 12 complex blocks with intense cross block interactions along with 1 lane GEN3 PCIe Root Complex, 2 lane GEN4 Endpoint and 1 lane USB 3.1. The past experience of these protocols in an SoC half complex as this had average run times of more than 10hrs. The need to close the verification activities in given timelines and with available resources kindled us to brainstorm on a novel approach right from planning stages to close the verification activities by dividing the activities between sub system and SoC. Typically sub system verification is carried out by IP teams, but the visualization and concept idea drove the SoC team to own up sub system verification for the first time. Basic Data path, Core features, Clocking and power scheme tests, GPIO connection and interrupt tests and toggle coverage requires the tests to be run at SoC level and its inevitable. Code coverage, checkers, monitors, scoreboard, drivers, pipe monitors, Analog - Digital interface cross over checks are few of the components that are independent of SoC environment which were planned targets for sub system. Considering the timelines and need to optimize resource allocation and effective use of the same, we enabled the Sub system environment to be directly scalable for both upper/further lower abstraction levels by splitting the sequence calling/integration at module level. Data path bring up at SoC level is one of the most challenging activity during the verification process. With modular sub system and approaches like DSNR and CNR data path bring-up was completed in a span of two days. The detection of bugs at later stages of

SoC can be very painful resulting in missing tape out timelines but with reusable subsystem (figure-1) already working, we were able to catch more than 17 critical bugs even before the data path was working at SoC level.

II. VERIFICATION METHODOLOGY

A. Subsystem Reuse at SoC

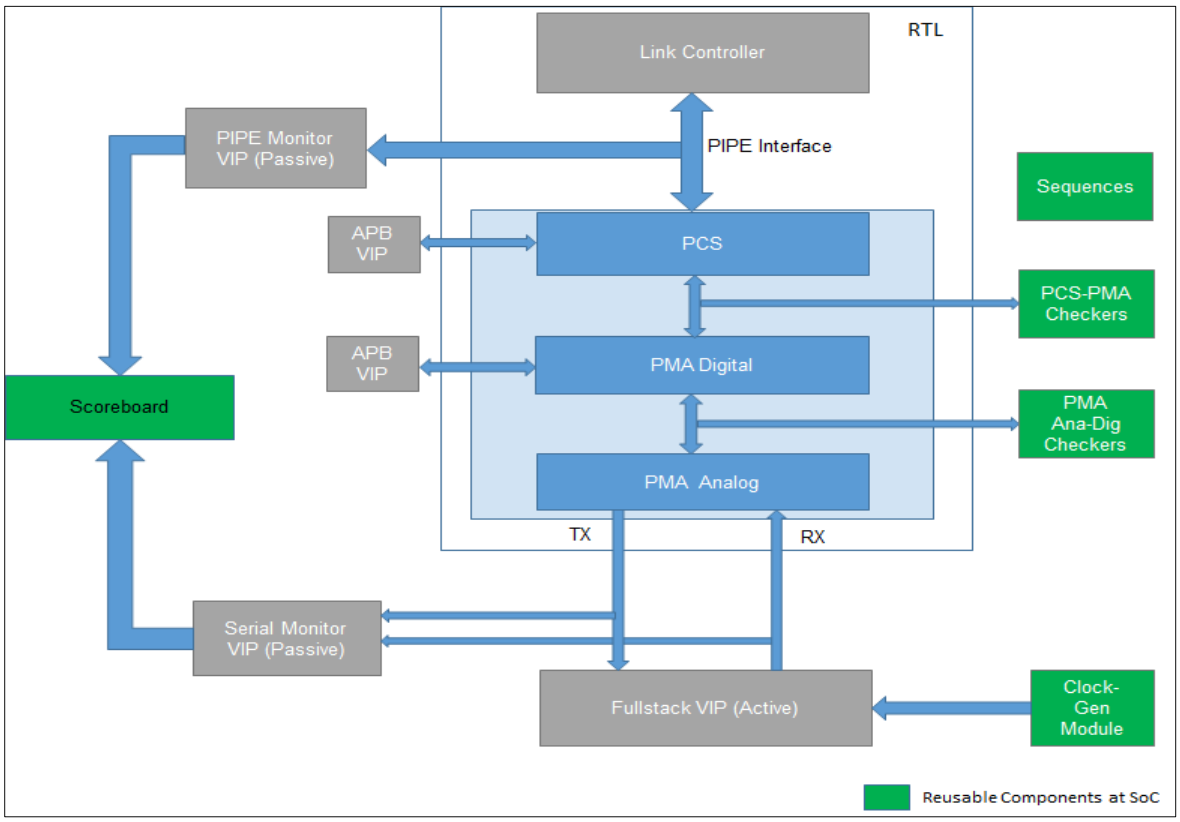

Figure 1. Subsystem Verification Environment

The challenges at SoC verification increase not just due to complexity of design but a need to run the simulations on different flavors to gain the confidence. The run time impact on multiple flavors like Power Aware(PA), GLS, PA GLS/MPNET flavor of simulation is manifold as compared to pure RTL run time.

Type of Simulation	Simulation Speed at Subsystem(hours)	Simulation speed at SOC (hours)
RTL	0.50	18
PA	0.70	27
GLS (Unit Delay)	0.91	32-45
GLS with SDF	NA	54
GLS with MPNET	NA	63

Table 1. Simulation Speeds at SoC vs Subsystem

To completely leverage the effort put on sub system at SoC requires a good planning and visualizing the environments at both levels and developing the sequences, monitors, drivers, scoreboard and test cases. The project and environment planning were carried out in 5 stages as shown in figure-2.

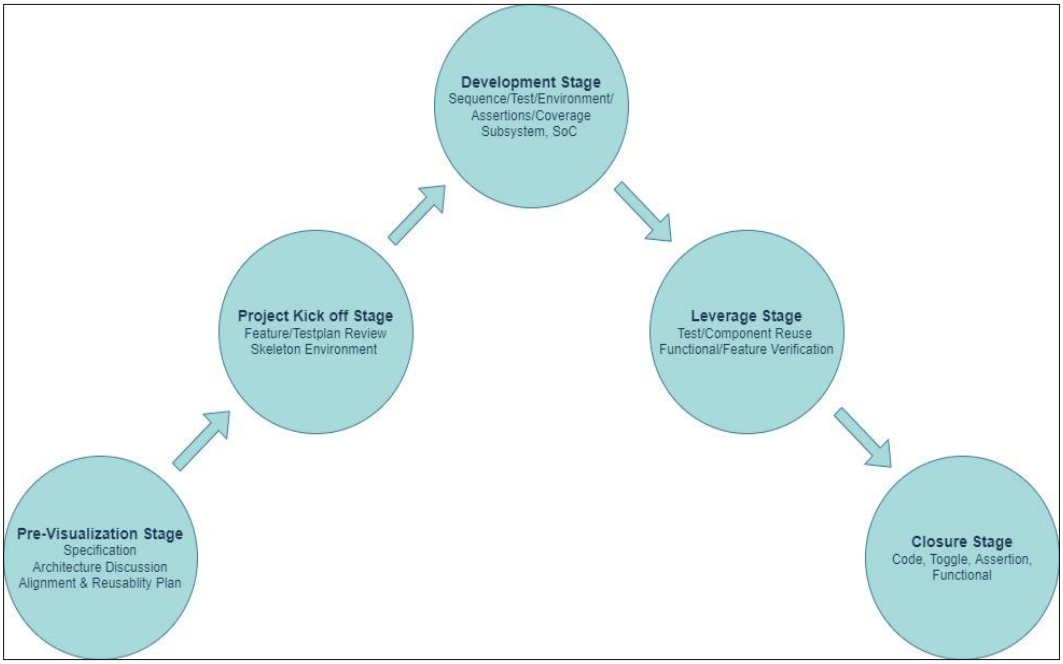

Figure 2. Project Execution Flow

Pre Visualization Stage is where the alignment and reusability plan including the tests planned at different abstract levels were finalized along with different builds and parameterization. In the Project kick off stage skeleton environment is created which includes environment specific classes, packages and configurations. Development stage was accelerated on subsystem environment to provide the head start for SoC. At this stage SoC environment is also developed in parallel to ensure when sub system environment works there is no further delay at SoC abstract level. Leverage Stage is where we have subsystem data path working and SoC is reusing the sub system components to get data path up. This stage extends till all tests at both abstract levels are completely verified. This is one of the most agile stages as changes w.r.t project specification and feature modifications are expected and needs to be accommodated to be a customer friendly foundry business.

Closure stage is when coverage closure activities are taken to closure at both sub system (Functional, Code – Toggle, Branch, Expression, FSM) and SoC (Toggle coverage at sub system wrapper). Metrics driven verification always ensures the gaps based on the test plan identified are caught during the development phase. The tests were identified based on the divide and conquer strategy resulted in effectively reducing SoC level runs to 46 test and 2634 tests for Subsystem to cover core and coverage specific scenarios. Ranking is one of the most efficient methodology deployed during coverage closure which eliminates redundant tests and identify tests contributing most to the coverage. Based on ranking of multiple regressions we found the subset of 1796 tests for achieving coverage closure at Subsystem.

ENV Type	SOC without Subsystem	SoC With Subsystem	
		SOC	Subsystem
Average Runtime of a test (hours)	14	14	0.56
Total no of Tests	2680	46	2634
Regression Time (hours)	1251	21	49
No of Iterations during SoC Cycle	15-22		6
Net Run Time (Max) (hours)	27522	462	294
x factor of Simulation Time	59x	x	0.01x

Table 2. SoC vs Subsystem Runtimes

Adoption of structured Subsystem Verification methodology provides significant productivity boosts to SoC Verification. Based on comparative study, our approach resulted in saving 26766(97.25%) hours of simulation, 48621 (98.9%) licenses and very critical CPU slots for a single subsystem with 59x improved sim time. The net code leveraging stood at 78% that required no modification. Extrapolating this to other sub systems would mean net savings of 41773 hours of simulation time.

B. DSNR – Single/Multi Snapshot

Dynamic save and restore provides a mechanism to create a snapshot of a simulation at a specific point in time (Save) which can be replayed for subsequent simulations (Restore). Typically, in many verification environments same configuration cycles are re-used across different test-cases. These cycles might involve writing and reading from different configuration and status registers, loading program memories, memory initializations and other similar tasks to set up a DUT for the targeted stimulus. In many of these environments, the time taken during these configuration cycles are very long. Also, there is a lot of redundancy as the verification engineers have to run the same set of verified configuration cycles for different testcases leading to a loss in productivity. This is especially true for complex verification environments with multiple interfaces which require different components to be configured. Whereas DSNR methodology provides an option of saving the state of the design and the testbench at a particular point in time. We can restore the simulation to the same state and continue from there. This can be done by adding appropriate built-in system calls from the Verilog code. Save and restore allows a session to be restarted without incurring the burden of reloading the design. This results in a significant performance boost on subsequent design reloads. Another benefit comes from the ability to fork a session for exploring several different scenarios. In this model, we can load the design, save the session, make copies of the session, and then for each variation, restore a copy and continue with the tests.

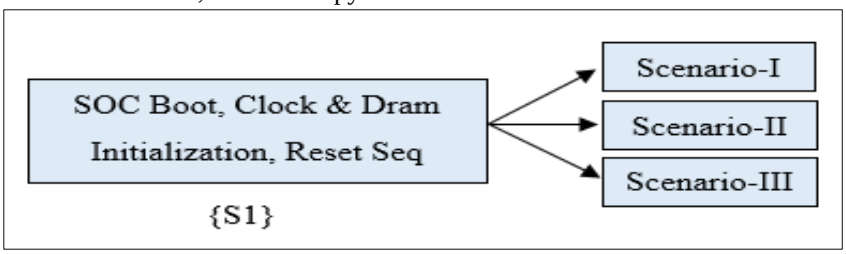

Figure 3. Traditional Single-Snapshot DSNR

Based on the experience from previous projects with traditional DSNR approach, snapshot load time during restart has increased exponentially with increase in design complexity. With the help of proposed approach as shown in figure-4 we can have control on snapshot load time by having multiple snapshots like SoC Boot, Initializations, Reset sequence and IP Specific linkup. Once a stage is successful the snapshot will be removed based on decision logic to minimize the loading effect on the simulator.

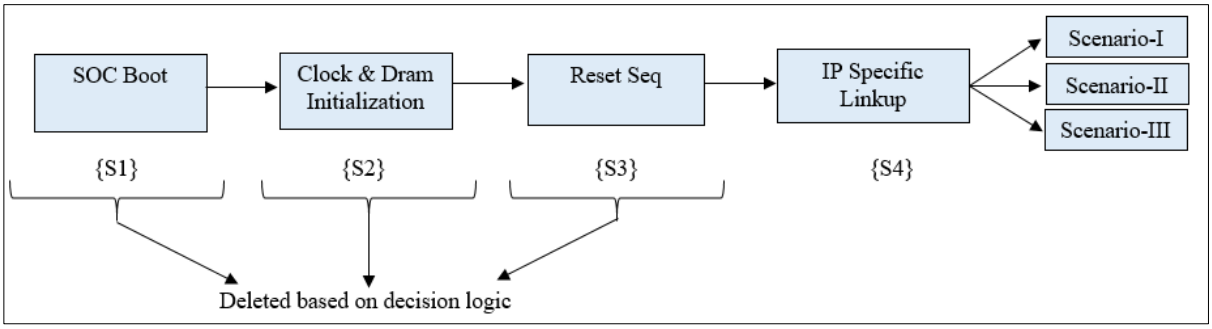

Figure 4. Proposed Multi-Snapshot DSNR

At each stage a snapshot is created by isolating the operation as shown in figure-4. S1 will act as starting point for S2 and S2 for S3 and so on. Once a snapshot is created the decision logic will decide whether to delete or retain the snapshot based the input constraint file and processing the log file.

DSNR Approach	Snap Size (GB)	Load Time (min)	x factor of simulator load time	SoC Regression Load time (in hours) (Considering SoC suite of 8320 tests)
Single-snap + Retest	13	56	x	7765
Multi-Snap + Retest	1.5	6.46	0.11x	895.7

Table 3. Load Time of Single vs Multi Snapshot

With the help of proposed approach 6869.3 hours of simulation and very critical CPU slots are saved. Also by making use of multiple snapshot method link bring-up time has drastically reduced as only failed snapshot need to be rerun. This approach can further be extended to have user configurable snapshots which is done based on time/size.

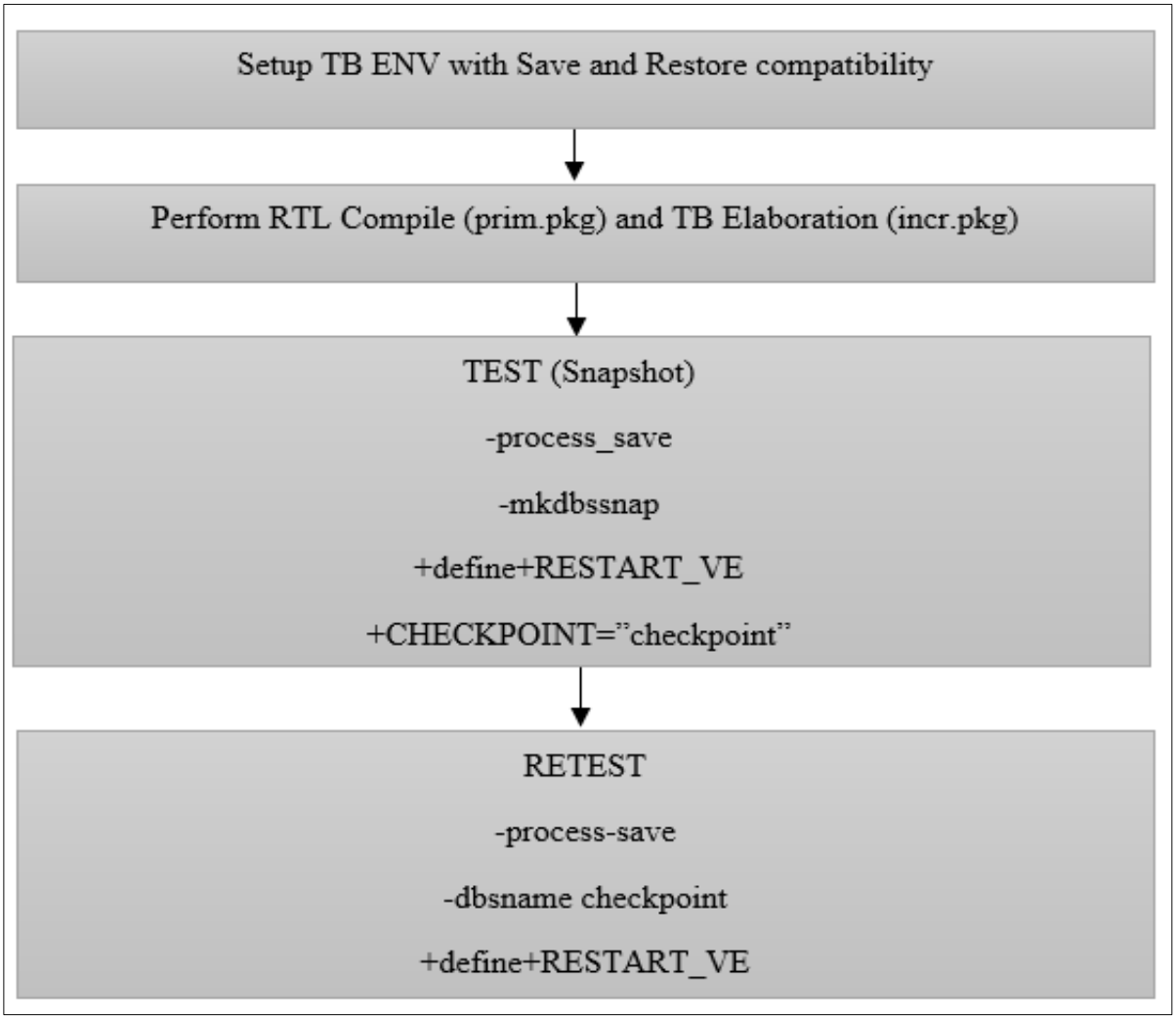

Figure 5. DSNR Flow

C. CNR

Capture and Replay is a faster and easier method to debug functionality/performance issues in a complex SoC design and reduces turnaround time of fixes. In this method a part of the erroneous design behavior is captured on one label of the design and replayed on other label by specifying the target scope. As only stipulated stimulus is replayed, simulation runtime saved is huge and debug is faster. Typically, GLS and PA simulations are run towards end of the project due to which bugs are detected at later stage of verification cycle. To minimize the bug hunting time and achieve faster SoC linkup below approaches are used

a. Capture stimulus at subsystem and replay at SoC
b. Capture stimulus at RTL and replay in zero delay GLS/Power Aware Simulations
c. Replay only part of captured stimulus

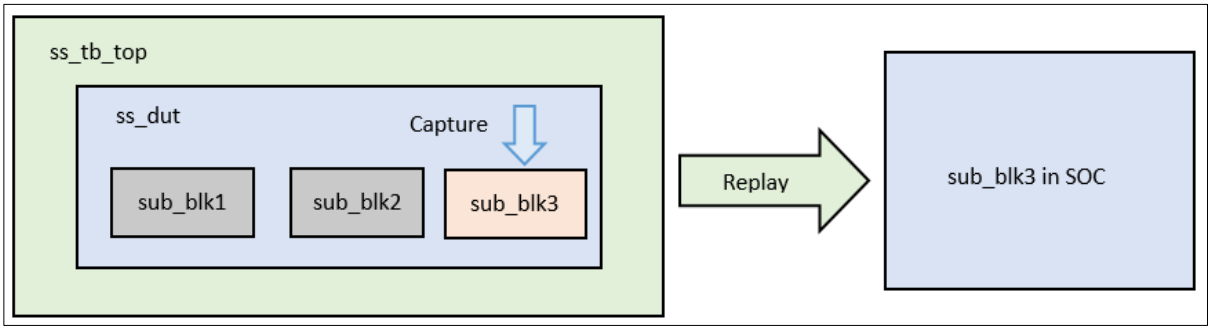

Figure 6. Capture part of subsystem (sub_blk3) and Replay at SoC

The scope of design at which replay need to be performed is captured first using the database option and the signals are probed using database and probe commands as shown in figure-7. Create a replay configuration file with database details, source/destination scope, sampling clock, force values to deposit and replay check output to create a log file with list of mismatches in the target scope. Replay database is created by loading the configuration file and probing the destination hierarchy.

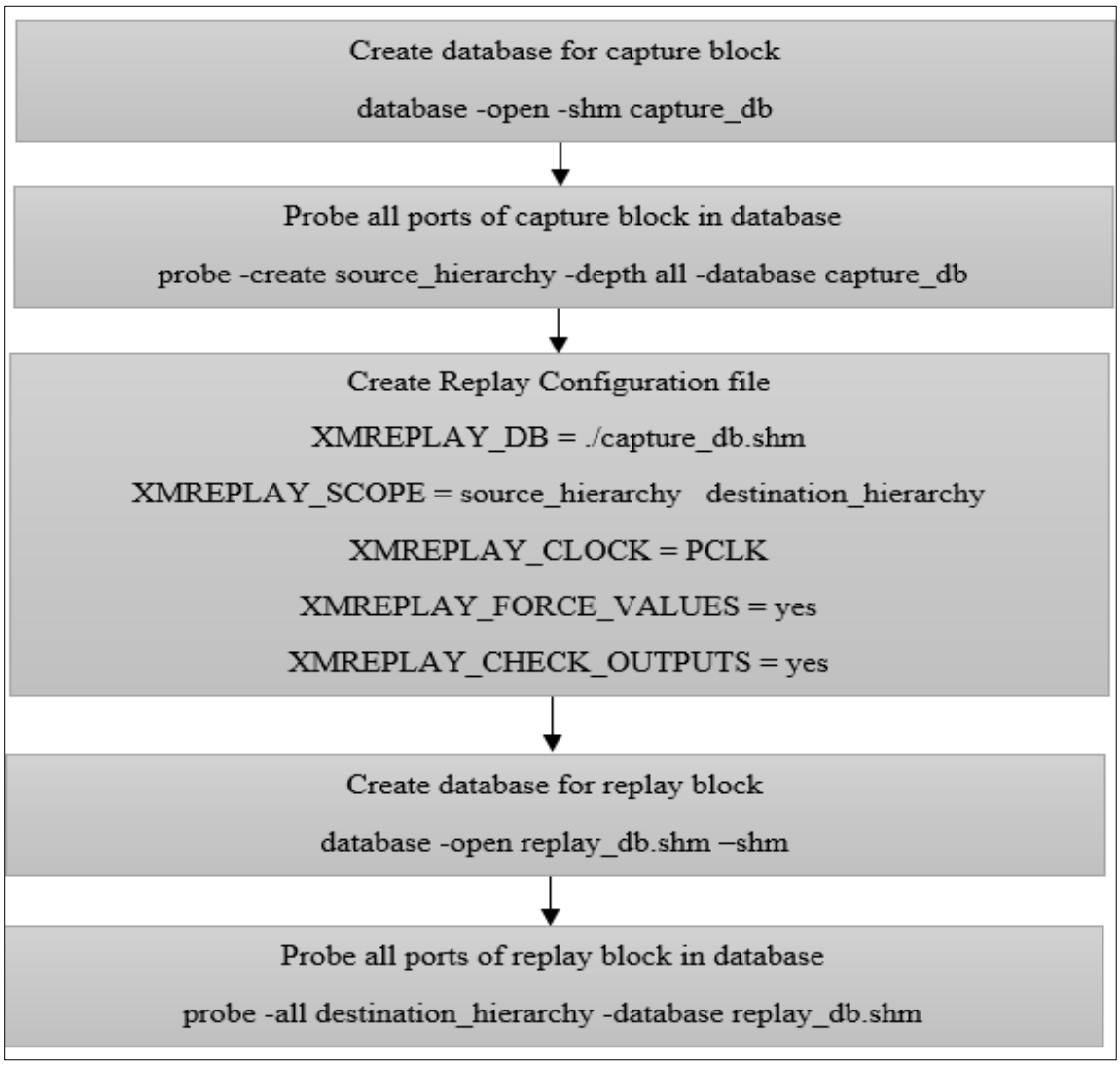

Figure 7. Capture and Replay Flow

III. CONCLUSION

The approach of subsystem reuse at SoC and CNR that was meticulously planned and executed has resulted in overall improvement in data path bring up time and early detection of 17 bugs, saving 97.25% of simulation time with 59x improvement in simulation speed and 98.9% of licenses as compared to conventional methodology thereby saving man months by at least 3 months. Multi snapshot DSNR approach has helped in saving 6869.3 hours of simulation time with 88.4% improvement in snapshot loading time.

This generic architecture can be extended to other low speed and High Speed protocols like CSI DSI, MIF, I2C, I3C. Evaluation of current architecture and feasibility for mimicking the similar approach on other IPs/Sub Systems are ongoing and will be published in future.

REFERENCES

[1] 20484428:Dynamic Test Loading with Xcelium (RAK)

[2] D. Geist, G. Biran, T. Arons, M. Slavkin, Y. Nustov, M. Farkas, K. Holtz, A. Long, D. King, S. Barret, “A Methodology For the Verification of a ‘System on Chip’,” DAC, 1999.