Advanced Functional Verification for Automotive System on a Chip

Jaein Hong, Jieun Jeong, Namyoung Kim, Hongkyu Kim, Sungcheol Park
Samsung Electronics, 1-1, Samsungjeonja-ro, Hwaseong-si, Gyeonggi-do 18488, Korea

Sangjun Mun
Cadence Design Systems, 2655 Seely Avenue, San Jose, CA 95134, USA

***Abstract*-Rapidly growing automotive semiconductor industry and the market requirements for advanced driver assistance system (ADAS) are leading a more complex and larger size automotive system on a chip (SoC) development. This paper studies on system level functional verification for automotive SoC. Specifically, it explains the automotive features that impose huge challenges on system level verification in terms of completeness and efficiency, and presents advanced verification approaches to solve these challenges. Firstly, complex power scheme in the automotive SoC is discussed as verification completeness challenge and the following approaches are provided as a solution: i) automatic coverage generation based on UPF, specification of power intents; ii) extension of the functional coverage with system control events; iii) assertion based low-power feature debugging methods. These approaches enhance metric driven low-power design verification and unveil undetected design bugs. Identified issues through proposed approaches are summarized in this paper so that it could be of practical help. Secondly, for self-diagnosis features that guarantee high system reliability of automotive SoC, limitations of existing simulation approaches when running usage scenarios are explained in terms of verification efficiency challenges. To deal with these challenges, new testbench architecture and test platform for efficient use of emulators are proposed. The proposed platform significantly reduces the TAT, thus confirming that tests difficult to validate earlier, can now be validated at the pre-silicon phase.**

I. INTRODUCTION

The automotive applications that support advanced driver assistance system (ADAS) demand i) high performance computing units such as machine-learning engines, CPU cores, and GPU cores; ii) separate units of safety and security power islands; and iii) self-diagnosis and recovery design. These functional requirements increase not only the number of integrated components, but also the diversity of use case scenarios. For example, the automotive system requires a completely isolated island for security and safety applications and it drives more segmented power modes. As the complexity increases with an increase in the power mode, the possible combination of power events and functional operations increase indefinitely. It is important to cover this complexity in the system-level verification. To improve verification completeness, metric driven verification methodology is developed and functional coverage is mainly used as the metrics. However, existing methodologies of defining the coverage for the power-control logic and the implemented power feature are insufficient to cover all possible combination of usage cases of the automotive SoC, especially using a complex power scheme. The automotive system must also guarantee a high system reliability, and it is important to diagnose all possible memory instances and logic blocks, during operation. In traditional AP SoC verification, diagnosis and recovery are mainly considered as wafer-level circuit testing and therefore, corresponding functional verification methodology is not developed, and functional simulation is mainly performed to verify that the test mode is set up correctly for circuit testing. Because automotive SoC supports the system diagnosis and recovery in function mode, it is important to deploy use case scenarios as a part of the SoC verification process. However, owing to the size of the test pattern and the longer simulation times, functional simulation tests are difficult to perform unlike wafer-level tests. This results in increased verification efforts that guarantee the system quality alongside an increase in the associated verification time, thus imposing a huge challenge on system-level verification in terms of completeness and efficiency.

This paper presents solutions for such verification challenges and proposes advanced approaches. The rest of this paper details them and it is organized as follows. Section II introduces the complexity challenges of automotive system and describes the limitations of the existing verification methodologies especially for low-power design. Detailed solutions to overcome these limitations are offered using practical example codes. Section III explains self-diagnosis and recovery design verification challenges in terms of efficiency. After typical use case scenarios and conventional test methods are explained, emulation based test platform is presented for built-in self-test (BIST). Finally, Section IV summarizes the approaches and achievements.

II. Ensuring Verification Completeness

A complex low-power design poses one of the biggest challenges for verification completeness [1]. The existing low-power verification implements power-aware simulation to ensure that each low-power feature such as power supply shut-off, dynamic voltage frequency scaling, power gating, retention, and isolation is correctly implemented in the power intent. From a completeness perspective, it brings functional coverage that defines the power states of on-chip components and the transitions between them, to a verification closure [2]. Fig. 1 shows an example of an automotive SoC that composed of CPU, graphics, computing engines, IOs, etc., as well as a low-power design that features multiple voltage rails and power gating switches. As the SoC is partitioned into segmented power domains and has analog components such as PLL, ADC, and PHYs that need separate supply voltage, many power rails are implemented for individual power control. Each power domain can have various power modes, and the power states and possible state transitions are manifold. Table I below describes examples of power modes in automotive systems, their power mode transitions, and possible power gating conditions in each power mode.

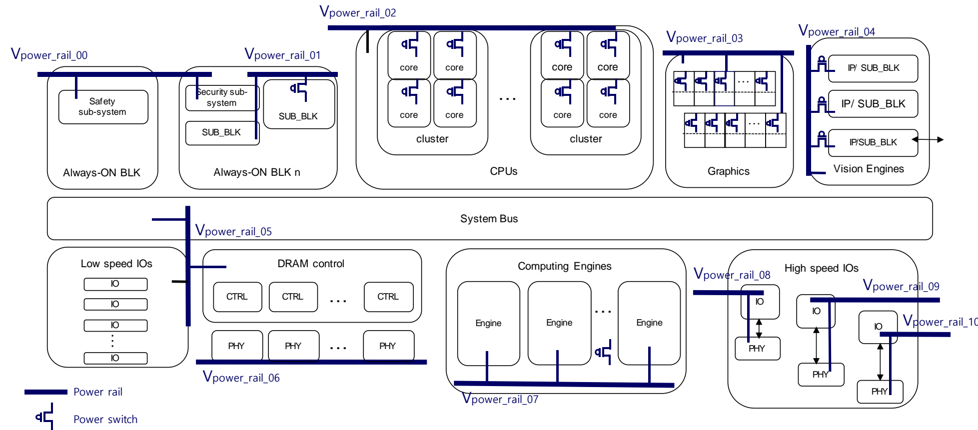

Figure 1. Example of automotive SoC power scheme including power rails and power switches

The power mode is defined based on the actual system operation, and for each power mode, the needed power rail is determined. Table I-(a) is a sample table that defines multiple power modes and conditions for each power rails are specified for each power mode. A power mode can be switched to another mode based on usage, and Table I-(b) represents power mode transitions. Additionally, power control events such as power gating, retention, and isolation can occur in each power modes. Based on the design, examples of the status of blocks/IPs that can be operated in each power mode are organized in Table I-(c). Power mode definition, all possible combinations of power mode transitions and power events must be fully covered in system level low-power verification.

TABLE I.
Example of Power Modes, Power Transitions, and status of block/IP according to power mode for Automotive SoC

(a) Power mode define

Power modes	Vpower rail_00	Vpower rail_01	Vpower rail_02	Vpower rail_03	...	Vpower rail_n
AON_0 mode	ON	OFF	OFF	OFF	...	OFF
AON_1 mode	ON	ON	OFF	OFF	...	OFF
...	...	...	...	...	...	...
AON_n mode	ON	ON	ON	OFF	...	OFF
Usage_0 mode	ON	ON	ON	ON	...	OFF
Usage_1 mode	ON	ON	ON	ON	...	OFF
...	...	...	...	...	...	...
Usage_n mode	ON	ON	ON	ON	...	OFF
ALL ON mode	ON	ON	ON	ON	...	ON

(b) Power mode transitions

Current Power Mode	Next Power Mode
AON_0 mode	{AON_0~3, Usage_0}
AON_1 mode	{AON_0~n, Usage_0~1}
...	...
AON_n mode	{AON, SOC_2~5}
Usage_0 mode	{AON_0, Usage_1~n}
Usage_1 mode	{AON_n, Usage_2~n}
...	...
Usage_n mode	{AON, SOC_2~8}
ALL ON mode	{*} all of possible power modes

(c) Status of Blocks/IPs according to power mode

Power modes	BLK_A	BLK_B	BLK_C	CPU	Graphics	BLK_D	BLK_E	...	BLK_XX
AON_0 mode	ON(possibly gating)	OFF	OFF	OFF	OFF	OFF	OFF	...	OFF
AON_1 mode	ON	ON(possibly gating)	OFF	OFF	OFF	OFF	OFF	...	OFF
...	...	...	...	...	...	...	...	...	...
AON_n mode	ON	OFF	ON	OFF	OFF	OFF	OFF	...	OFF
Usage_0 mode	ON	ON(possibly gating)	OFF	ON(possibly gating)	OFF	ON	OFF	...	OFF
Usage_1 mode	ON	ON(possibly gating)	ON	ON(possibly gating)	ON(possibly gating)	OFF	ON(possibly gating)	...	OFF
...	...	...	...	...	...	...	...	...	...
Usage_n mode	ON	ON(possibly gating)	OFF	ON(possibly gating)	ON(possibly gating)	ON	OFF	...	OFF
ALL ON mode	ON	ON	ON	ON(possibly gating)	ON(possibly gating)	ON	ON(possibly gating)	...	ON

Each table in Table I can be translated as a single coverage group and all conditions listed in the table can be implemented as coverage bins to confirm the completeness of power feature verification. However, manually defining coverage bins could lead to errors and any missing coverage bin may result in silicon malfunction. In addition, coverage closure toward simple power states and transitions are insufficient because of the complexity of the low-power design. Issues can occur due to a combination of power-related sequences, including reset and LPI handshaking combined with power-state changes, and not due to a simple power-mode transition. Finally, power-aware simulation consumes longer time than logic simulation and the results are hard to debug because many power-related components such as isolation cells and level shifters are not visible in the RTL code. The latter half of this section describes improved low-power verification approaches to address such hurdles.

A. Automatic generation of power coverage based on UPF

Power-aware simulator automatically generates power coverage based on UPF by adding coverage options. Using Cadence Logic Simulator, the coverage models for the following low-power items are provided by adding LPS options:

- Supply net states
- Supply set sim-states
- Power switch
- Power state table (PST)
- Control signals of isolation and retention

As part of test completeness, it is important to analyze coverage data of these low power items to ensure that they are completely exercised in power tests, and any illegal conditions are detected [3]. Uncovered coverage bins can provide clues for missing test scenarios.

B. Extending functional coverage bins to combine system-level power management events with a set of system operations

Tool-generated coverage metrics can help low-power verification. However, it is difficult to confirm if a test scenario including simultaneous or sequential power events is verified, or a test scenario in which multiple power events and system events combined is verified. Therefore, we extended functional coverage bins to combine system-level power management events with a set of system operations that access and change the state of power-control signals. For example, power state transition in the middle of a system operating mode change; power switch on and off during interrupts and watchdog fire; DRAM retention during system reset; and so on.

Fig. 2 is a sample code snippet that combines system reset with power state. Automotive SoC has multiple system resets with different purposes and reset regions. The coverage metric combining power event and system reset event can identify design errors in the Power Control Module.

```
cp_system_reset_power_mode: coverpoint system_power_state
iff($rose(system_reset)){
  bins power_state_b[] = {ALL_ON, Usage_1, Usage_n};
}
```

Figure 2. Example of coverage code combining system reset with power state

Fig. 3 describes a sample coverage metric for interrupt handling and power mode. Interrupt handling can be an example of a system operation since some interrupt signals may pass across many power domains. In addition, wrong clamp value of isolation can cause unexpected interrupts or a system malfunction. Cross coverage of power state transition and interrupt handling enables verification engineer to find bugs or identify missing interrupt test scenarios.

```
cr_intr_during_power_state_transition: cross cp_interrupt_00,
cp_system_power_state_transition_AON_0;
```

Figure 3. Example of coverage code for power state transition during interrupt handling

The SoC consists of multiple power domains. Multiple IPs in each power domain continuously communicate with IPs in other power domains. LPI handshaking of IPs in a power domain must be completed before the power domain is turned off to ensure that there are no remaining transactions. If a missing LPI handshaking exists or the order of

LPI handshaking is incorrect, the system may malfunction. For example, if the LPI handshaking of an asynchronous bridge is missing, or the LPI handshaking of an IP and the LPI handshaking of the asynchronous bridge are completed in wrong order, a system error may occur during communication with another power domain, after the current domain power is turned on again. Therefore, it is necessary to verify that all LPI handshaking in the SoC is performed correctly by implementing the functional coverage model. The sample code snippet that combines LPI handshaking and power states is shown in Fig. 4.

```
cp_qch_state: coverpoint qch_state {
  bins qch_state_b[] = {Q_RUN, Q_REQUEST, Q_STOPPED, Q_EXIT, Q_DENIED,
Q_CONTINUE};
}
cp_qch_state_transition: coverpoint qch_state transition: coverpoint
qch_state {
  bins Q_RUN_to_Q_REQUEST     = (Q_RUN => Q_REQUEST);
  bins Q_REQUEST_to_Q_STOPPED = (Q_REQUEST => Q_STOPPED);
  bins Q_STOPPED_to_Q_EXIT    = (Q_STOPPED => Q_EXIT);
  bins Q_EXIT_to_Q_RUN        = (Q_EXIT => Q_RUN);
  bins Q_REQUEST_to_Q_DENIED  = (Q_REQUEST => Q_DENIED);
  bins Q_DENIED_to_Q_CONTINUE = (Q_DENIED => Q_CONTINUE);
  bins Q_CONTINUE_to_Q_RUN    = (Q_CONTINUE => Q_RUN);
}

cr_qch_state_and_power_state: cross cp_qch_state, cp_system_power_state {
   ignore_bins AON_0 = binsof(cp_system_power_state) intersect {AON_0};
}
cr_qch_transition_and_power_state: cross cp_qch_state_transition,
cp_system_power_state;
```

Figure 4. Example coverage code of ARM LPI Q-channel

C. Enhancing debug capabilities for low-power features using assertions

During power-aware simulations, objects inside a shut-off domain are corrupted with generated X values. Proactively, identifying X problems in real silicon, necessitates X-propagation verification in power-aware simulation. However, power-aware X-propagation simulation is time consuming and tedious for bug identification. Using assertions not only helps the verification engineer to easily identify an X-problem in a non-X-propagation simulation – before running power-aware X-propagation simulation – but also enables faster power-issue detection without waveform analysis. Following items can be written as assertions for low power verification:

- Power switch/isolation/retention control signals must not be X or high-z
- Initial value of power switch/isolation/retention control signals
- Relation between isolation control signal and power domain shutoff: Isolation signals should not toggle during power off. Isolation signals should be enabled before power off.
- Retention: Signals for save and restore should not toggle during power off. The retention save process should be done before power off and retention restore.
- Power on/off sequence: The order of power supply, switch, isolation, retention, and reset controls must follow the specification.

In addition, Cadence SimVision™ Debug derives power objects that exist in the power intent such as power domain states and isolation conditions [4]. This method supports full tracing for both power objects and system events, thus shortens the debug time.

D. Results

The proposed approaches greatly increase the number of coverage metrics and assertions for low-power feature. Critical design errors were easily identified at the pre-silicon verification phase, and a low-power bug-free silicon was achieved. Owing to target design confidentiality, the contents of the implemented metrics and design errors cannot be described in detail. However, items that likely cause system malfunction issues, which were difficult to find with the existing low-power verification methodologies, can be categorized as follows:

- Wrong sequence of retention save/restore and power switch signals

- Missing isolation between analog components and digital-logic power domains.
- Wrong isolation values
- Missing LPI handshaking
- Incorrect order of LPI handshaking for IP and asynchronous bridge.
- Power Control Module errors – incorrect power control signal sequencing
- Missing test scenarios that combine real IP operations and power mode transitions

III. Increasing Verification Efficiency

The extension of automotive systems to fulfill the safety requirements has created a new challenge for verification efficiency because verification must be completed before the system is realized in silicon. On-chip self-diagnosis and recovery logic to expedite the in-field circuit testing that relies on built-in self-test (BIST) mechanisms is an example. The BIST circuits can be enabled both at system power-on – power-on self-test (POST) and periodically in the middle of an assigned mission execution – in-system test (IST) [5]. BIST covers on-chip memory instances and logic blocks – memory BIST (MBIST) and logic BIST (LBIST). As the memory size and the number of memory instances increase, the MBIST proportionally increases and gets complicated. The complication can be translated as the explosion of verification entities and an infinite simulation TAT. The LBIST imposes a more complex burden because it uses built-on scan chain [6]. The scan chain is invisible at RTL level and can only be verified at gate level after scan insertion. This results in longer simulation time than RTL simulation.

Fig. 5 illustrates an example sequence of full BIST operation phases including POST and IST. When the system is powered up for the first time and cold reset is released, POST is operated for all memory instances to guarantee system reliability. The system bootloader must set the proper clocks for POST operation and trigger the POST start signal to read dedicated POST ROM codes – test instructions and patterns stored on ROM. After POST operation completes, the system enters mission mode in which all functional operations run. During mission mode, system-monitoring functions such as temperature measurement or voltage droop detection run simultaneously with the system function operation to check for any system malfunction. In the event of a system malfunction or when the system manager wishes to trigger IST, system enters an idle state and starts IST operation to diagnose and recover both memory instances and logic blocks. Before and after IST, the system manager enables and disables isolation settings to avoid propagating any garbage values during the IST and to guarantee the system operation after IST. After IST completes, the system enters another mission mode, and repeats IST periodically until power off. Although the operation of each phase is simple, larger BIST patterns and longer simulation times contribute to making system level verification difficult.

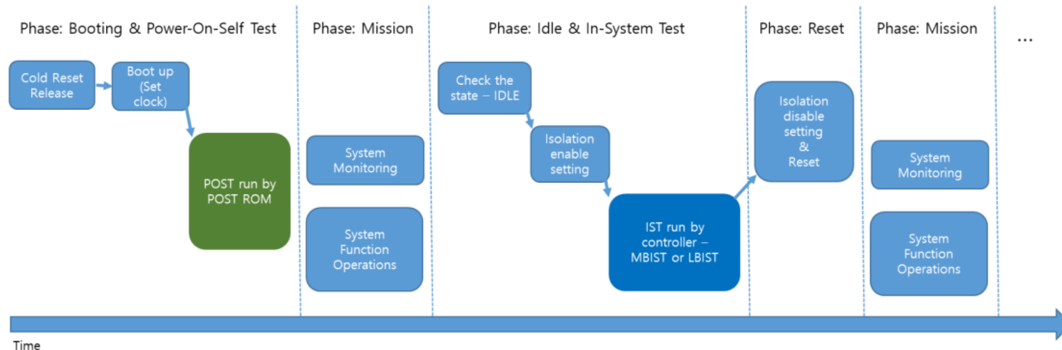

Figure 5. Example sequence of full BIST operation

A conventional method of reducing verification TAT and performing BIST operation test is to cut down the complete BIST pattern and usage scenario – create partial patterns and only check the feasibility of BIST operation flow with limited simulation. Fig.6 shows the difference between the full pattern and the partial pattern of memory BIST as an example. While the full pattern covers all the memory instances and BIST controller logics, the partial pattern covers the subset of memory instance that includes all BIST controller logics. Partial pattern enables BIST operation simulation for all BIST controllers with a lower memory coverage, thus enabling BIST operation feasibility and identification of critical errors in the BIST controller. However, this approach can ignore design errors such as a misconnection between the BIST logic and memory instance. In addition, there is a limit to reduce the size of partial pattern, and simulation time overhead remains high as long as all BIST controllers are covered, thus making it tedious to simulate the full BIST operation test.

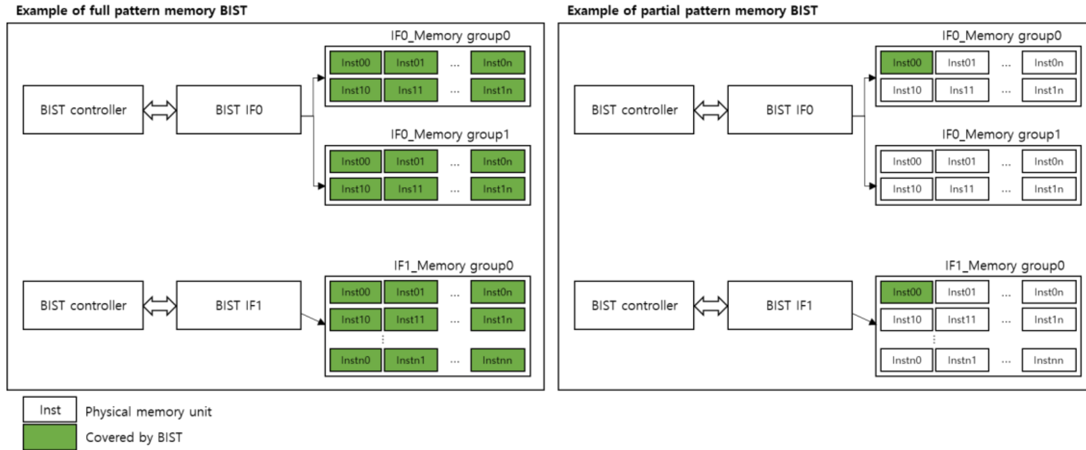

Figure 6. Example of memory BIST patterns (full pattern vs. partial pattern)

Fig.7 depicts the conventional simulation test sequences that individually cover the POST, mission, and IST operation. In POST only operation test, POST controller reads a simulation ROM code that includes partial patterns, real ROM code is replaced with the simulation ROM code to shorten the simulation time. Mission mode operation is separately covered including isolation setting that is a part of full BIST operation. IST operation is similarly covered to POST using the partial pattern. The complete IST operation sequence is manipulated to skip the POST and mission mode to reduce the simulation time. After POST or IST, simple register access test is performed to confirm only the system healthiness whether it can start with clock or function register setting instead of mission mode operation. This approach has limitations in confirming the real BIST use-case scenarios, thus mandating the need for an improved verification method to reduce the simulation TAT and increase coverage at the same time.

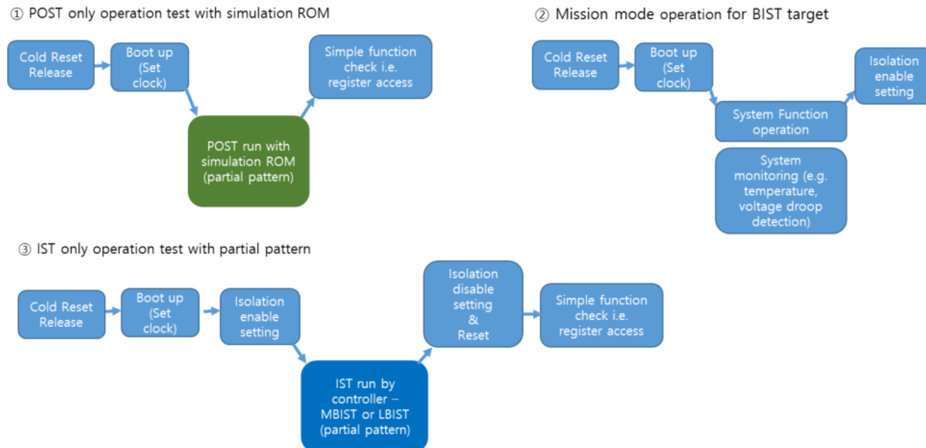

Figure 7. Example of simulation test sequence for POST, mission, and IST operation

Using emulator to accelerate simulation is ideal when issues such as limited emulator capacity, low debugging capabilities, compilation dependencies, and possibility of RTL misinterpretation due to self-synthesize mechanism are resolved. Because of the emulator capacity limitation, DUT is cut down for uploading emulator hardware and corresponding emulation testbench is developed. Fig. 8 shows an example of conventional emulation testbench that stubs-out unrelated blocks from the test target to save emulator capacity. According to the test, block-wise RTL is replaced by a black box and target RTL is loaded in hardware emulator. This existing emulation testbench is useful for increasing the BIST pattern coverage in individual POST or IST for memory. However, it is not suitable for LBIST and the full BIST operation test, since LBIST is implemented in built-on scan chain that is only visible at netlist.

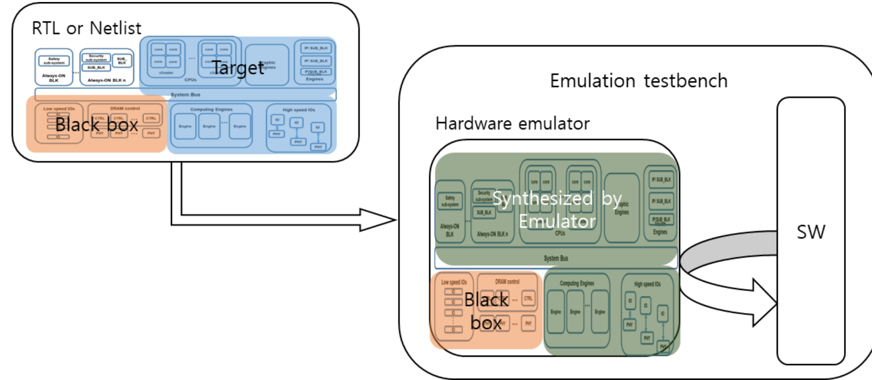

Figure 8. Example of conventional emulation testbench

An easy way of performing LBIST is to change the RTL used in emulation testbench to the netlist. Because netlist consumes more emulator hardware capacity, the more efficient use of emulator is needed to perform the same test that was completed with RTL. The netlist-based DUT testbench may behave differently and software test migration may require to run the existing tests as it is. In particular, in the case of a target designed to fit the netlist like LBIST, the test result may vary depending on the interpretation of the emulator. So, false failures can occur during diagnosis and recovery processing, and it can cause unwanted iterations of compilation and emulation test. To deal with these problems, a new emulation testbench is presented for referencing netlist based DUT.

Fig. 9 shows the proposed emulation testbench that reconfigure the target DUT as RTL, original netlist, and black boxes. Most parts of the DUT are aimed to configure as RTL and black boxes for less consumption of emulator hardware capacity, and for minimizing the software test migration efforts. A few parts of DUT that includes target diagnosis and recovery scheme only configures as netlist in the proposed testbench. Additionally, Cadence Palladium® Z1 platform supports loading the synthesized netlist as it is, and it helps to avoid unnecessary iterations due to the netlist re-synthesize at emulator hardware.

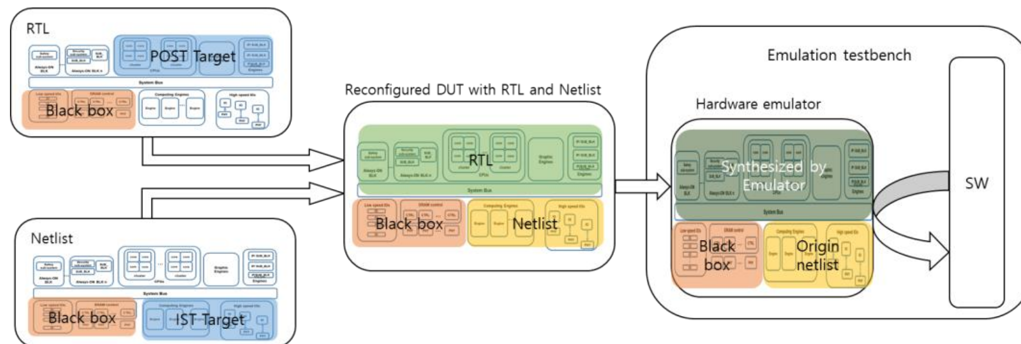

Figure 9. Proposed emulation testbench

Using the proposed emulation testbench, full pattern BIST operation tests are able to run at pre-silicon functional verification phase. Table II shows the results of comparing the BIST pattern coverage and run-time between simulation and emulation. The POST code is restored on-chip ROM to expedite during power up. In the conventional simulation, nearly 10% coverage pattern ROM is used and the time takes around three days. However, using the emulator, the complete pattern is exercised using real ROM code and the test is completed within two hours. For the IST, the conventional MBIST simulation used a 26% coverage vector and consumed over six days; however in emulation, 100% coverage vector completed in five hours. For 16K random LBIST pattern, emulation test consumed 48 hours whereas the same vector simulation failed to complete despite running for more than two months.

Additionally, using the proposed emulator testbench, the complete BIST sequence is able to confirm as real usage case scenario. Each functional sequence that individually confirmed is merged, and perform seamless operation, from the cold reset release to mission mode operation as depicted in Fig. 5. As the Cadence Palladium® Z1 platform enables users to probe any native RTL signal during run-time, without re-compiling the design [7], it increases the debugging capability on emulator and helps to reduce the real usage scenario creation and validation time.

TABLE II.
BIST PATTERN COVERAGE AND RUN-TIME: SIMULATION VS. EMULATION

	Simulation		Emulation	
	Pattern coverage	Run-time [hrs]	Pattern coverage	Run-time[hrs]
POST ROM	10%	72	100%	2
IST MBIST	26%	145	100%	5
IST LBIST	1-pattern	NA	16384-pattern	48

IV. CONCLUSION

In this paper, advanced verification approaches to overcome the completeness and efficiency challenges of the automotive SoC are provided. First, an improved metric driven verification is explained to enhance verification completeness. This is achieved by generating metrics and assertions from specifications and combining them with the system-operating modes. Next, a new testbench for efficient use of emulator is proposed to accelerate simulation to verify the full BIST operation using netlist, which is impossible in the conventional method. These proposed methods enable identifying corner cases and design issues that are normally undetectable during pre-silicon validation phase, thus achieving a one-time correct silicon development for a billion-gate automotive SoC.

ACKNOWLEDGMENT

We would like to thank Mijung Noh, Corporate Vice-President, Samsung Electronics, for the support and encouragement in publication of our result.

REFERENCES

[1] Ashok B. Mehta, “ASIC/SoC Functional Design Verification”, Springer, 2018.
[2] P. Khondkar, P. Yeung, et al., “Low Power Coverage: The Missing Piece in Dynamic Simulation”, DVCon US 2018.
[3] Cadence Design Systems, Inc., “Low-Power Simulation Guide (IEEE 1801)”
[4] Cadence Design Systems, Inc., “Debugging Low Power with SimVision”
[5] D. Sargsyan, "ISO 26262 compliant memory BIST architecture," 2017 Computer Science and Information Technologies (CSIT), 2017, pp. 78-82.
[6] M. Cogswell, D. Pearl, J. Sage and A. Troidl, "Test structure verification of logical BIST: problems and solutions," Proceedings International Test Conference 2000 (IEEE Cat. No.00CH37159), 2000, pp. 123-130.
[7] Cadence Design Systems, Inc., “Palladium Z1 Enterprise Emulation Platform Datasheet”