Saving and Restoring Simulation Methodology using UVM Factory Overriding to Reduce Simulation Turnaround Time

Ahhyung Shin, Yungi Um, Youngsik Kim, Seonil Brian Choi
Samsung Electronics Co., Ltd., Seoul, Korea
(ah0403.shin@samsung.com)

***Abstract*-Since the size and the complexity of SOC designs increase significantly, it is also getting hard to verify the designs in time without hurting the entire production schedule-. To reduce simulation turnaround time (TAT), a lot of technologies and methodologies have been being developed in many studies. However, these solutions are able to have technical limitations or financial limitations. When we analyze all test scenarios in SOC verification, some initial setup sequences are repeated over and over again at all subsequent tests [1]. This paper analyzes test scenarios from a macroscopic perspective and proposes new simulation methodology reducing simulation TAT using Universal Verification Methodology (UVM) factory override mechanism. This new methodology introduces advanced save, modify and restore methodology that is different from typical save and restore mechanism.**

I. INTRODUCTION

As a size and a complexity of SOC design has been increasing, verification challenges for coverage and turnaround time(TAT) are also increasing drastically. To resolve these verification challenges, sufficient test cases and sufficient time to test them should be secured. However, since the development period is hardly limited, it is very difficult to secure more verification time. Therefore, it is difficult to perform enough tests within a short specified period, and it is even more difficult to proceed with error-free verification in all SoC projects. We are sure that we can perform enough tests only if the simulation time for the test is drastically reduced. To reduce simulation time, a lot of technologies and methodologies have been studied in many previous papers. Improving the performance of simulator itself and using emulator are one of examples. However, these approaches have some limitation, respectively. Simulator performance has been improved approximately 10% per a year, which is a way behind than the speed of SOC design size increase. An emulator requires huge financial investment and yet gets maximum performance improvement only for long runtime test scenarios. Therefore, a study for a new methodology which can reduce simulation TAT without any HW investments is required. This paper proposes the new methodology called as "Save and Restore (**SnR**)" that can be applied to every test scenario by analyzing test scenarios and trying to reduce simulation TAT.

According to analysis of SoC-level test scenarios, there is a basic and commonly used test sequences such as booting or initialization for SOC operation, and these sequences must be performed for a very long time before testing the target block to verify. Furthermore, these identical common sequences are typically repeated for all subsequent tests and it occupies a very large portion of all test sequences [1]. In other words, even if verification engineers who are only interested in their actual test sequences should execute this common sequences in every time, and this common sequence takes relatively longer time than test sequences when size and complexity of SoC is bigger. Furthermore, whenever verification engineers modify test sequences, engineers have to start simulation always from the common sequence. This iteration work is the main reason to increase verification TAT.

The concept of SnR has been present from the earliest versions of RTL simulators and has been being developed [2]. However, a lot of challenges remain like it is only optimized to regression simulation that is progressed with almost fixed testbench. Typical SnR methodologies are not effective in situations where testbench needs to be constantly modified during development. This paper proposes a new SnR methodology to reduce simulation TAT when the test sequence was modified. The proposed methodology adopts a UVM Factory override mechanism. Since most verification environments of SOC designs are being performed on UVM environments in nowadays, this is the easiest way to reduce verification TAT in UVM environment with very low cost.

II. BACKGROUND

Designers of complex designs may go through a common initial setup phase for all their tests. This setup phase could be either executing the same sequence of simulation steps that programming their design to reach the same initialization or reset state [1]. SOC booting sequences that initializing CMU, PMU and PHY initializing sequences can be examples. These common sequences are repeated over and over again for all test scenarios in SOC verification. These identical and repetitive processes are the main reason that increases simulation TAT.

The basic concept of SnR methodology is to specify a specific point during simulation and then skip the iterative simulation sequences using the point as the starting point of the next simulations. In general, all states of the design and the testbench at the specified point are stored to a file. Then these states are re-loaded from the file and the simulation continued from that point [2]. And so far, many challenges are being developed from this basic idea.

A. (Typical) Saving and Restoring Simulation Based on Simulation Time

The most basic and simple SnR Methodology is a time-based SnR simulation. Verification engineers specify the point with a simulation time when the engineers want to save the state of design under-test (DUT) and testbench to a file using simulator command and re-load it. This method has a limitation that the verification engineers cannot know exact point where the simulation state is captured in test scenarios. In other words, it is difficult to align a simulation time and instruction sequence.

B. (Typical) Saving and Restoring Simulation with DPI-C and UVM Factory Override Mechanism

A more advanced scheme of SnR Methodology is simulation which uses DPI-C function and Tcl command that performs UVM factory override mechanism. This method saves the state of DUT and testbench to a file through DPI-C function at the save point that is set by the verification engineer. When re-loading the saved file, the simulator executes a Tcl command to override new UVM sequences those will be executed after the saved point. However, the new UVM sequences which will override at restoring simulation are compiled and elaborated when the simulation is saved to a file. Therefore there is a limitation that verification engineers cannot modify the new UVM sequences after the saved file is written.

III. PROPOSED APPROACH

Existing SnR methodologies have limitations as described in the preceding section. It is difficult to align the saving point with the sequence of the testbench correctly, or it is difficult to change the UVM sequence after the saving point is specified. Also, since several points cannot be specified, several versions of the SnR environment must be built. In this paper, a method of designating a multi-saving points and overriding the UVM Factory dynamically is proposed to solve this limitation.

When simulation reaches the saving point that is set inside of the test scenario, all states of DUT and testbench are written at a file that can be called as “a snapshot”. After the snapshot is created, restoring simulation can be started. The UVM factory overriding is performed in the restoring simulation. Restoring simulation proceeds in the following order:

1) *New input file lists are compiled and elaborated*
2) *A Tcl command of simulator performs a UVM factory override*
3) *Start restoring simulation from the save point that is set inside of the test scenario*

As simulator compiles and elaborates the new input file lists when starts restoring simulation, the new UVM test sequences which is used in restoring simulation should be modifiable.

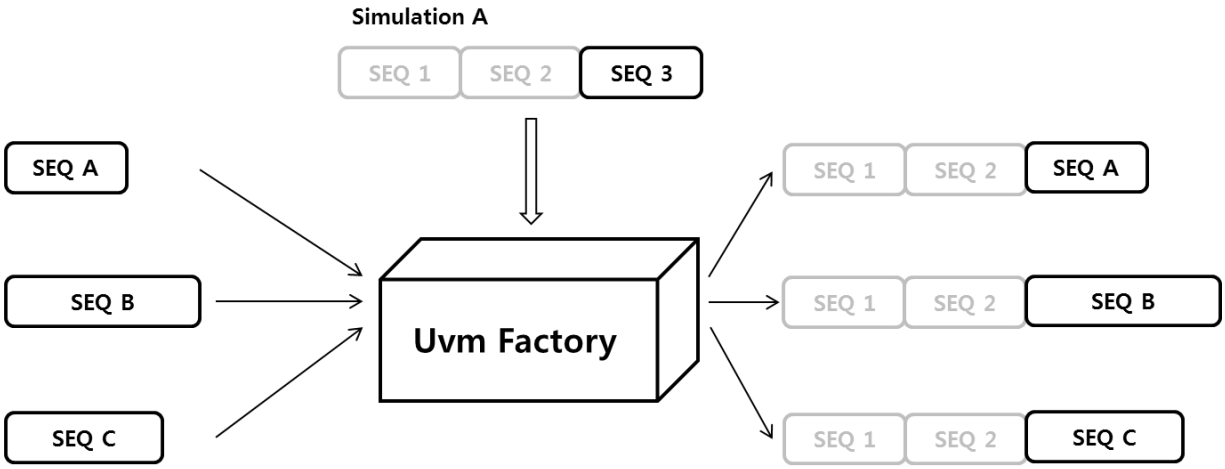

Figure 1. Basic Concept of UVM Factory Override Mechanism

A. UVM Factory Override Mechanism

The UVM Class Library provides various utilities to simplify development and use of verification environments. These utilities support a flexible verification environment construction called as UVM factory. The factory allows verification engineers to substitute the verification component without having to provide a derived version of the parent component as well. Using the UVM built-in a factory reduces the effort of creating an advanced factory or implementing factory methods in class definitions [3]. In the SnR methodology workflow, the restoring simulation can proceed after the snapshot is created while the saving simulation and the restoring simulation perform the UVM factory overriding. When the restoring simulation starts test, the restoring simulation performs the UVM sequence overriding using the UVM factory override mechanism. Through these processes, one saved snapshot can be reused by a lot of test sequences.

To successfully implement the UVM sequence overriding for the SnR methodology, the testbench must meet all of the following conditions:

1) A test sequence being overridden in the saving simulation and a test sequence to override in the restoring simulation must be registered to the UVM factory using UVM macro.
2) Since the UVM sequence overriding by the Tcl command is based on type overriding, the test sequence being overridden in the saving simulation and the test sequence to override in the restoring simulation must be the same type.
3) There should be a UVM sequence that is executed after the point is specified. In other words, there should be a test sequence being overridden in saving simulation after the save point where making the save snapshot is proceeding.

B. SystemVerilog Package

SystemVerilog supports separate compilation by units. The compilation-unit scope is treated as a top-level design unit. And SystemVerilog package is one of the items that compilation-unit scope of SystemVerilog can contain. SystemVerilog packages provide an additional mechanism for sharing parameters, data, type, task, function, sequence, properties and checker declarations among multiple SystemVerilog modules, interfaces, programs, and checkers [4]. If the testbench is described in the SystemVerilog packages, the dynamic SnR method is possible. To make modifiable test sequence that will be executed in the restoring simulation, the restoring simulation test sequence should be packaged with SystemVerilog packages. The restoring simulation proceeds following steps:

1) The test sequence to override in the restoring simulation must be located inside of a SystemVerilog package.
2) The restoring simulation receives the file that includes the SystemVerilog package of the test sequence to override as an input.
3) The simulator compiles and elaborates the input file and starts the restoring simulation.

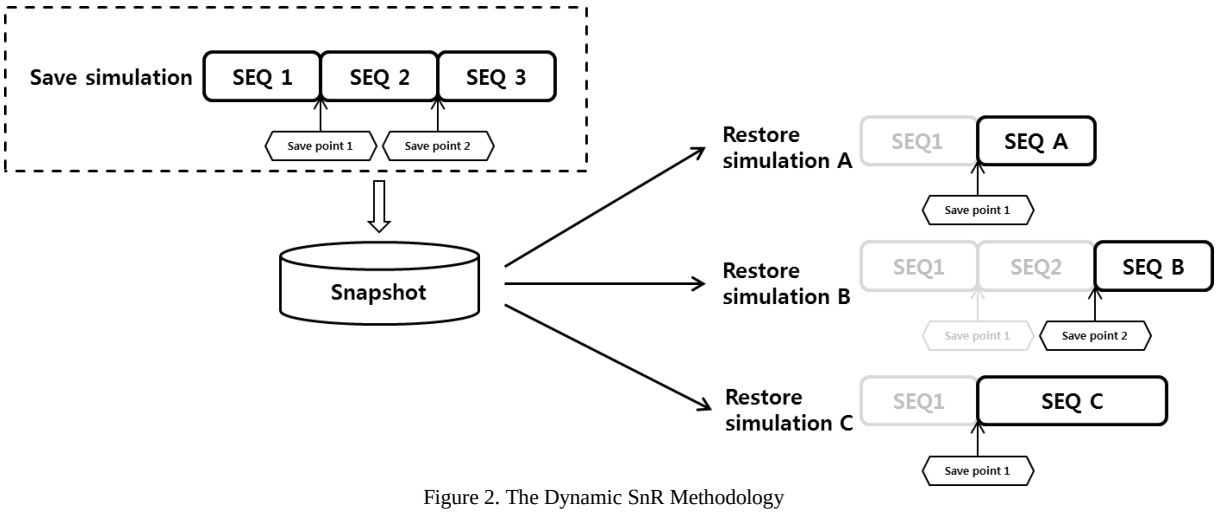

Figure 2. The Dynamic SnR Methodology

IV. PRACTICAL SNR METHODOLOGY

Figure 2 shows the basic concept of the SnR Methodology. Verification engineers can set multiple save points inside of the saving simulation testbench between the UVM sequences. In restoring simulation, the verification engineer can choose one save point and start restoring simulation from the save point.

A. Practical SnR Methodology Workflow

This paper proposes a practical workflow on how to implement the proposed SnR methodology. Figure 3 shows comparison of two verification workflows, one is the normal verification workflow without SnR methodology and the other is the proposed advanced verification workflow with SnR methodology.

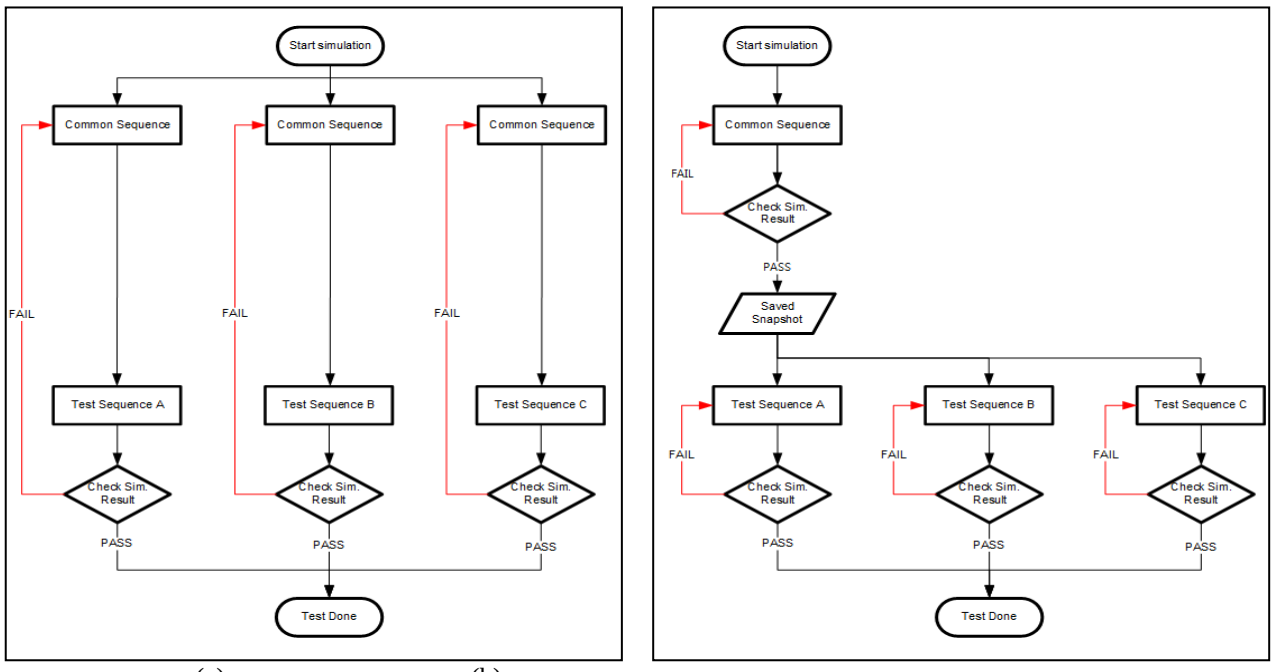

Figure 3. Proposed Workflows with SnR Methodology.

The red lines in Figure 3 are the simulation TAT that a verification engineer usually experiences if testbench developing is on-going. Figure 3 (a) shows the normal verification workflow without the SnR methodology. In Figure 3 (a), verification engineers have to simulate both of common sequence and test sequence for each test scenario. And if the simulation is failed, the verification engineer debugs errors and starts the simulation from a common sequence. On the other hand, Figure 3 (b) shows a verification workflow with the SnR methodology. A verification engineer executes a common sequence test at the first step in the case of verification with the SnR methodology. If the common sequence test does not have problem, a snapshot file that has all states of DUT and testbench is generated at save point. And the verification engineer starts multiple test scenarios with the snapshot. If a test sequence is failed, then the engineer could debug and simulate the test sequence from the checkpoint without executing a common sequence again.

B. Practical SnR Methodology with UVM Codes

This paper proposes practical UVM codes along with workflow. The following examples describe simple SnR testbench with "common_sequence" for the saving simulation and "test_seqeuence" for the restoring simulation.

1) Saving Simulation

<saving_test.sv>

```
class saving_test extend uvm_test;
  …
  task run_phase();
    common_sequence.start();     // The common sequence that will be saved
    do_save("check_point");      // Save a snapshot at this point
    base_sequence.start();       // A sequence being overridden at restoring simulation
  endtask: run_phase
  …
endclass : saving_test
```

<do_save.c>

```
#include "cfclib.h"
#include <stdio.h>
Void do_save(char* snapshot)
  { char cmd[1024];
    sprint(cmd, "stop -delta 1 -delbreak 1 -execute
    {save -overwrite %s} -silent -continue", snapshot);
    cfcExecuteCommand(cmd);
}
```

<run script>

```
%> xrun … \
        -mkdynamicsnap \
        -process_save \
        +CHECKPOINT="check_point"
```

Figure 4. Saving Simulation and do_save() function

Figure 4 shows command codes for a saving simulation those are related with execution of a common sequence and saving a snapshot at the save point called as "check_point". To save all states of DUT and testbench, the verification engineers use a function, "do_save()". The function "do_save" is a DPI-C function that stores the states of DUT and testbench. Whenever the "do_save()" is called at the saving simulation, a snapshot including states of DUT and testbench is created.

2) Restoring Simulation

```
<test_pkg.sv>

package test_pkg;
                        / New package file that includes test sequence
  …
  class test_sequence extend base_sequence;
                        // New test sequence to override at the restoring simulation
    …
  endclass : test_sequence
  …
endpackage : test_pkg
```

```
<uvm_factory.tcl>
uvm_factory -override -by_type base_sequence test_sequence # Do uvm factory override
uvm_factory -print
```

```
<run script>
%> xrun … test_pkg.sv uvm_factory.tcl\
          -dbssnap check_point \
          -dbsname @save_snapshot_dir \
          -top test_pkg
```

Figure 5. Restoring Simulation and Tcl command for UVM factory override

Figure 5 shows the UVM codes for the restoring simulation. The UVM sequence named as "test_sequence" is located in the SystemVerilog package "test_pkg" and it is a test scenario executed in the restoring simulation. The sequence named as "base_sequence" in Figure 4 is overridden with a new "test_sequence". To make UVM factory overriding success, the sequence to override at restoring simulation should have the identical type with the sequence being overridden at saving simulation. Therefore, the "test_sequence" inherits the "base_sequence" as shown in Figure 4. And this UVM sequence to override is located inside of a SystemVerilog "test_pkg". As shown in Figure 5, the SystemVerilog package "test_pkg" file which includes the "test_sequence" is an input file for the restoring simulation. A simulator gets input data from the file. Then, the simulator compiles and elaborates the "test_pkg" before the restoring simulation starts. Also, there is another input Tcl command file named as "uvm_factory.tcl" for the restoring simulation. The Tcl command performs a UVM factory type overriding. With this testbench, the simulator can compile and elaborate a new "test_pkg" and perform a UVM factory overriding. Furthermore, we should let the simulator know which snapshot will be used for the restoring simulator.

The proposed SnR Simulation is a dynamic SnR Methodology with DPI-C and UVM Factory Override Mechanism. This dynamic SnR methodology is developed and implemented to complement the limitations of typical SnR methodologies in this paper. The advantages of the dynamic SnR methodology go beyond the limitations of typical SnR methodologies.

1) Verification engineers can set a save point at any point in the middle of test scenarios wherever the engineers want. Naturally, multiple save points can also be set. The multiple save points make each save file. And any point in the saved point can be selected as a start point at a restoring simulation by verification engineers.
2) A verification engineers can modify UVM sequences although previous, saving-required simulation is already finished. In other words, the verification engineers can add or modify test sequences at any time. Especially due to this reason, the new dynamic SnR simulation method supports to remove identical and repeated execution of common sequence while a verification engineers develop testbench. Furthermore, this method can help to reduce simulation TAT.

V. Experimental Result

Table I shows a result of the proposed SnR methodology. In this paper, two functional blocks, Block-A and Block-B, were used to confirm effects of proposed methodology. In test scenario of each block, SoC booting sequences and PHY initialization sequences are includes in common. And these two sequences are always re-run whenever

simulation is performed. Total number of test cases is around 30 per each block, and all test cases were performed twice depending on whether the proposed SnR methodology was used or not. For Block-A, the simulation TAT reduction effect was measured by specifying the point where the SoC booting sequence is completed as a saving point "check_point". For Block-B, the point where the PHY initialization sequence execution is completed was designated as a "check_point" and the simulation TAT reduction was measured. In the Table I, "Normal Simulation" column indicates the simulation time when the proposed methodology is not applied, and "Restore Simulation" column indicates the simulation time with the proposed SnR methodology. Eventually, simulation TAT of the Block-A was reduced by 88% and that of Block-B was reduced by 97%. To summarize, it was confirmed that the simulation time was reduced and the TAT decreased by more than 88% as the proposed SnR methodology was used. Moreover, the larger the amount of sequence in SnR is applied, the larger TAT reduction effect was confirmed.

TABLE I. 30 Test Cases Result of Save And Restore

	***Bock-A (Booting seq)			***Block-B (Booting seq + PHY init)		
	Normal Sim.	Restore Sim.	TAT Reduction	Normal Sim.	Restore Sim.	TAT Reduction
Simulation Time (min)	2507	303	88%	16161	537	97%

VI. CONCLUSION

In this paper, we proposed a new SnR methodology which can reduce simulation TAT drastically. In the study, we realized that all test scenarios have common sequences which must be executed during a simulation by analyzing a lot of test cases in actual mobile AP SoC project. And we are able to skip the common sequences in many test cases using UVM Factory override mechanism. We can confirm through various experiments the proposed SnR methodology can reduce a simulation TAT drastically.

As for future work, SnR methodology can be applied for power simulation and gate level simulation also. And it will be further more effective for them because their simulation time is extremely long and SnR methodology could be get greater effective for long simulation time or low speed simulations such as power estimation tests or gate-level simulations. Furthermore, SnR methodology also can be used for regression tests. And it is expected to reduce storage usage and simulator license usage.

REFERENCES

[1] Rohit K Jain and Shobana Sudhakar, Mentor Graphics "Want a Boost in your Regression Throughput? Simulate common setup phase only once." in DVCON US 2015

[2] Ed Powell, Ron Thurgood and Aneesh Samudrala. Hewlett Packard Enterprise "Using Save/Restore is Easy, Right? A User's Perspective on Deploying Save/Restore in a Mature Verification Methodology" in DVCON US 2019

[3] Accellera "Universal Verification Methodology (UVM) 1.1 User's Guide", May 18, 2011

[4] IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group, "IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language", 21 February 2013