Reality and Challenges in Using Save/Restore in SoC Verification Environment

Satish Naidu
Amol Bhinge

Freescale Semiconductor, Inc.
Austin, USA

www.freescale.com

ABSTRACT

One of the major challenges in keeping up with increasing SoC design complexity is the time spent in running longer simulations and the lengthier debugging turnaround times. One approach to mitigate this challenge and improve the overall productivity is to use a save/restore mechanism; by restoring a simulation run from a saved state which could be the reset or bring-up phase. However, the challenge is how well this concept can scale up realistically in a SoC verification environment. Several factors can affect the overall value from using Save/Restore. These include: testbench methodology, design configuration requirements, nature of the stimuli, and tools' support. In this paper, we present our experience in using save/restore for both RTL and gate simulations, and how the aforementioned factors influenced architecting and building the complete verification environment. Our results showed saving about 25-30% of simulation times for gate regressions, and around 10-20% productivity for RTL regressions.

Table of Contents

1 Introduction ... 3
2 Factors influencing using Save/Restore feature ... 4
2.1 TESTBENCH METHODOLOGY ... 4
2.2 DESIGN CONFIGURATION REQUIREMENTS ... 6
2.3 STIMULI ... 6
2.4 SIMULATION TOOL ... 6
3 Building verification environment with Save/Restore ... 7
3.1 ARCHITECTING THE TESTBENCH ... 7
3.2 COMPILATION AND SIMULATION FLOW ... 8
4 Debugging flow with Save/Restore ... 9
5 Results ... 10
6 Conclusions and Recommendations ... 11
7 References ... 11

Table of Figures

Figure 1 - Legacy Verification Methodology ... 5
Figure 2 - Save/Restore in UVM ... 8

Table of Tables

Table 5.1 - Simulation Statistics for a RTL simulation ... 10
Table 5.2 - Simulation Statistics for RTL regression ... 10
Table 5.3 - Simulation Statistics for a Gate Level Simulation ... 11

1 Introduction

A main goal of doing SoC verification is to build an efficient environment to enable finding as many bugs in the design under test as early as possible without compromising the quality of the testbench. One aspect of the increasing complexities in the design is the increasing time spent in reset/initialization phase of the design, which has a direct impact on running every simulation. In our previous SoC verification environment, our simulation regression suite comprised of multiple regression subsets where each subset included all the tests that target a particular block in the design. Obviously, it was desired to reduce the time taken to finish the regression as fast as possible to enable more verification cycles. On careful investigation of a particular test in of one of the regression subsets, we found that the reset and initialization phase of the design was taking up to 60-70% of the total simulation time, and we found that most of the other tests in the same regression subset did the same thing. Considering that there were hundreds of regressions running, there was clearly a huge amount of simulation time wasted due to repeating the reset/initialization in every test. This also resulted in poor LSF resource utilization. If we could have reduced the redundant simulation cycles, we could have completed the regressions faster and maximized the LSF resource utilization. Faster simulation results would mean faster regression completions which would have allowed us to keep up with the faster design releases and faster turnaround times.

An approach to mitigate this challenge and improve the overall productivity is to use a save/restore mechanism. The concept of restoring a simulation run from a saved state, which could be the reset or initilization phase, to avoid repeating the same simulation cycles is well understood. However, the challenge is whether this concept can scale up realistically in a SoC verification environment. Several factors contribute to or limit the extent of the value from using a Save/Restore mechanism in a SoC verification environment.

First and foremost factor is the choice of testbench methodology and how phasing and sequencing are defined in it. Traditional verification methodologies with phasing and sequencing closely coupled have challenges in implementing a Save/Restore scheme. However, a methodology like UVM which decouples phasing and sequencing would be a better choice. We will discuss about the merits and demerits of it later in this paper. Second important factor is the design configuration requirements and how major configurations setting that need to happen early during a simulation run may limit the value. A design which expects major configurations to be done during the reset sequence will not be able to fully benefit from this feature. Also, sequences which use directed tests for configuration has a better chance in using this feature compared to a fully random configuration/initilization sequence. A third factor is the nature of the stimuli used. For example, compile-independent stimuli like C and TCL has a better chance to benefit from using this feature vs. other types of stimuli. A fourth factor is the simulator and level of support in it for this feature and the verification tasks‘ requirements.

In this paper, we present our experience in using the Save/Restore feature for both RTL and gate simulations in our latest SoC verification environment. We show how the aforementioned factors played in and influenced some of the decisions we made while architecting the testbench and building the verification environment. We discuss the overall flow with using Save/Restore feature including compile, simulation and debug. We show how using Save/Restore feature in

RTL and gate level simulations improved simulation productivity by reusing the snapshots created by one simulation despite a little initial performance overhead which can be accommodated to increase the overall productivity of the regressions. We also show how using Save/Restore feature along with VCS Unified Command Line Interface (UCLI) forces helped in enabling faster debugging without the need to recompile the simulation model and making forward progress to foresee the next simulation issue in lesser time. In the end, we conclude with some recommendations and tool enhancement requests.

2 Factors influencing using Save/Restore feature

2.1 Testbench Methodology

A verification methodology which has phasing and sequencing coupled together will not be able to utilize this feature extensively. For example a methodology in which the phasing is common for all components in the testbench (Components, Drivers, Monitors, Sequences, Stimulus, etc.) has limitations in using the Save/Restore feature. When all these components which are of same type execute in parallel in each phase, it is difficult to implement the Save/Restore mechanism in one single place.

In the methodology described in Figure 1, which our previous SoC verification environment was based on, there can be only one startup stimulus which executes first before the user stimuli is executed, it might seem to be a better place to put the Save/Restore feature in it. But if we take a closer look at the transaction Sequencing, the user stimuli is restricted to use only the tasks following execute task and will not be able to use restart and configure tasks. A lot of our stimuli were using configure tasks to perform user specific configurations and hence implementing Save/Restore feature on this legacy environment was a little difficult and hence had a restriction on the number of users who could utilize this feature.

On the other hand when we considered using UVM as our methodology for the latest SoC verification environment, we saw that the sequences were no longer part of the components and were extended out of uvm_object which doesn’t have the concept of phasing. Phasing was all contained to uvm_component classes and hence sequences which will be coded could be used as plug and play as needed.

So, there are two different options to implement the Save/Restore feature in UVM. We can implement it inside tests, which extend from uvm_test components or it can be part of the sequences itself. We took the approach of using both test and sequences to implement this feature.

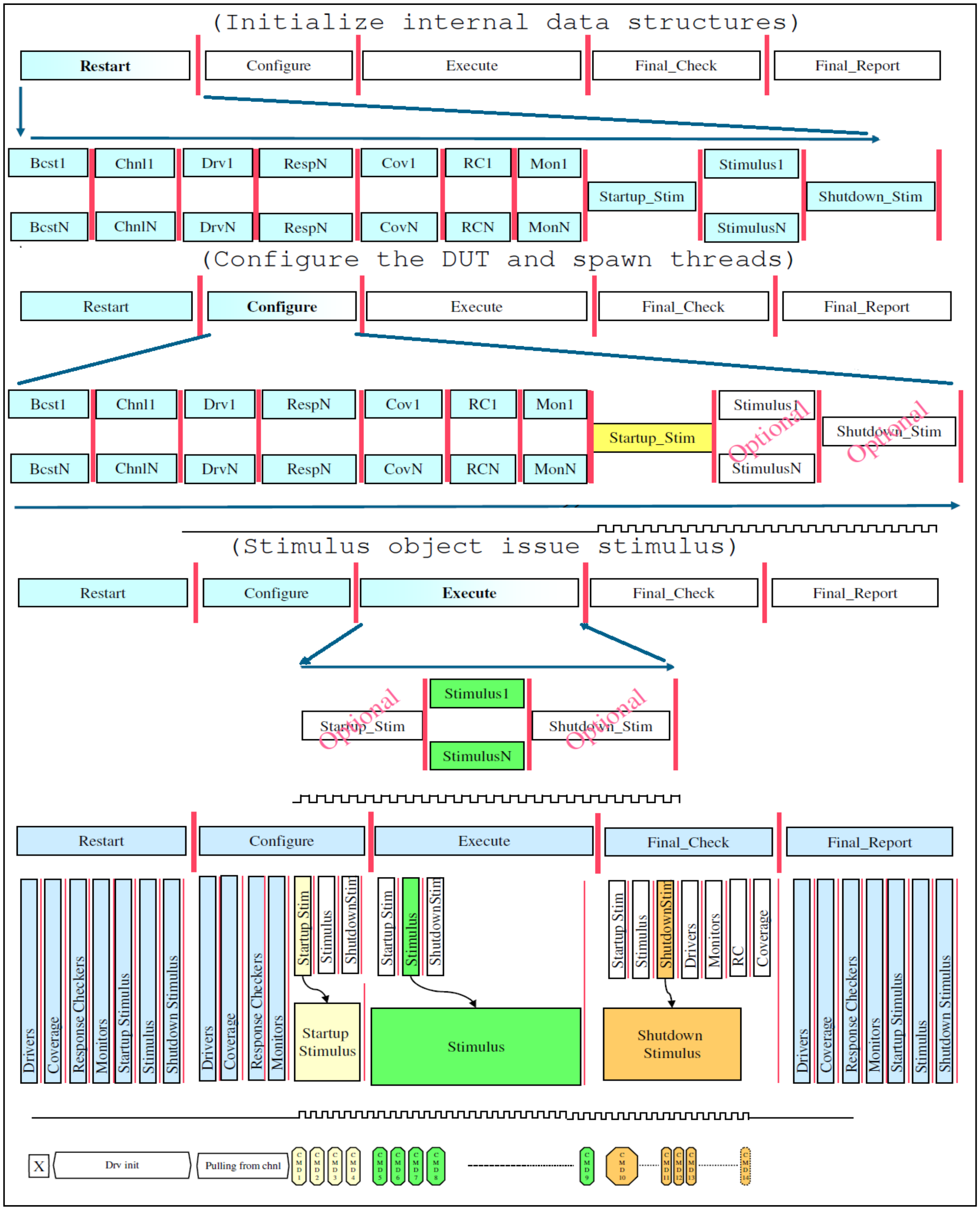

Figure 1. Legacy Verification Methodology

2.2 Design Configuration Requirements

There are few design requirements which need to be considered before implementing the Save/Restore feature. IP designs are smaller and have more ports to be toggled to hit functional scenarios, whereas SoC has fewer I/Os which funnel in the data/control signals to the specific IP block. So, Save/Restore feature might not be a good solution at IP level design verification. A typical SoC has a reset, clock and few configuration pins which determine the behaviour of the design during or after the reset sequence is executed. SoC designs which have more such configuration pins to be configured at time 0 will not be able to fully utilize this feature. For example, consider a design which has a pin which determines the personality of the design (number of processors, I/Os, etc.). If we capture the default behaviour of the design in a snapshot simulation, then we can’t use this snapshot to restore and verify the functionally on a different personality. Designs which expect users to load instructions/data early enough during the reset/configure sequence in the entire simulation will not be able fully utilize this feature as well. All these requirements need to be kept in mind before implementing Save/Restore features in the testbench.

2.3 Stimuli

For constrained random based testbench stimuli, the value of using Save/Restore feature can vary depending on the design configuration requirement, as mentioned in the previous section. However, for subsets of regressions that target certain IP block, the design configuration will most likely be the same, therefore running UVM based stimuli (sequences) will benefit from Save/Restore feature.

However, , methodologies which use C based tests/stimuli are the best ones to fully utilize Save/Restore feature, especially from debug perspective, C based tests which run on embedded processors are compiled, independently of the testbench and design compilation, at simulation time and preloaded into the memories at some point in simulation which are executed on the embedded processors. We can preload different instructions on the embedded processors using the same initial saved simulation snapshot, since they don’t require re-compilation of the whole testbench model.

Similarly for TCL based tests/stimuli. TCL based tests can be loaded into the simulation at a specific restore point of simulation time which started using the same initial saved simulation snapshot.

2.4 Simulation tool

Support of the Save/Restore mechanism in the simulation tool has to be at the level to achieve the objectives of the verification tasks, the debug use model, and the flow for running regression.

3 Building verification environment with Save/Restore

3.1 Architecting the Testbench

Architecting the testbench to fully utilize the Save/Restore feature is the biggest challenge. We often tend to bring-up testbench to get things up and running by taking shortcuts which need to be re-looked at a later point in time to add support for Save/Restore features. In one of the earlier projects, we implemented some shortcuts to configure the design using `$deposit` and forces and it happened to be part of an initial block. The configuration values were captured using run-time `$plusargs`, and most of the tests were using these run-time arguments to configure the design and verify various functionalities. This coding style restricted us from using Save/Restore feature since most of the tests tried to configure the design differently and everything happened at time 0 since it was inside an initial block. After analysing the design, it so happened that the design doesn't need these configurations to be available at time 0 and could delay this loading of configuration to a later time, which could open up the window to implement the Save/Restore feature and the initial block was later changed to an always block based on an event.

Some of the design configurations were also included in the StartupStimulus of the legacy methodology described in Figure 1. The memory initialization and instruction preloading for the embedded processors were done as part of the StartupStimulus as soon as the reset was asserted. When we moved to the UVM, we captured memory initialization and preloading as separate sequences and we executed these sequences just before they were supposed to be executed. This gave us more control on inserting save snapshots before executing these sequences. Our previous SoC was taking 60-65% of simulation time in finishing reset and configure sequences which could be saved now by creating a snapshot after configure sequence.

Referring to Figure 2, we architected the UVM testbench in such a way that we have only one `uvm_test` component which runs multiple virtual sequences in different UVM phases based on the runtime argument which specifies the virtual sequence name to be run in a particular phase. By doing so, we could implement the `$save` statements at the beginning of every phase. We also captured a few snapshots inside the reset sequence. We have some legacy tcl based tests which run using internal tools. These tests don't need recompilation of the design/testbench once the model is built.

Most SoCs have fuse reading process during the Power on Reset (POR) sequence. One way to speed up the POR sequence is to bypass fuse reading. But this might hide potential bugs in the design. With Save/Restore feature creating a simulation snapshot after the completion of fuse read would shorten the POR time for subsequent simulations, hence avoiding design shortcuts in simulations.

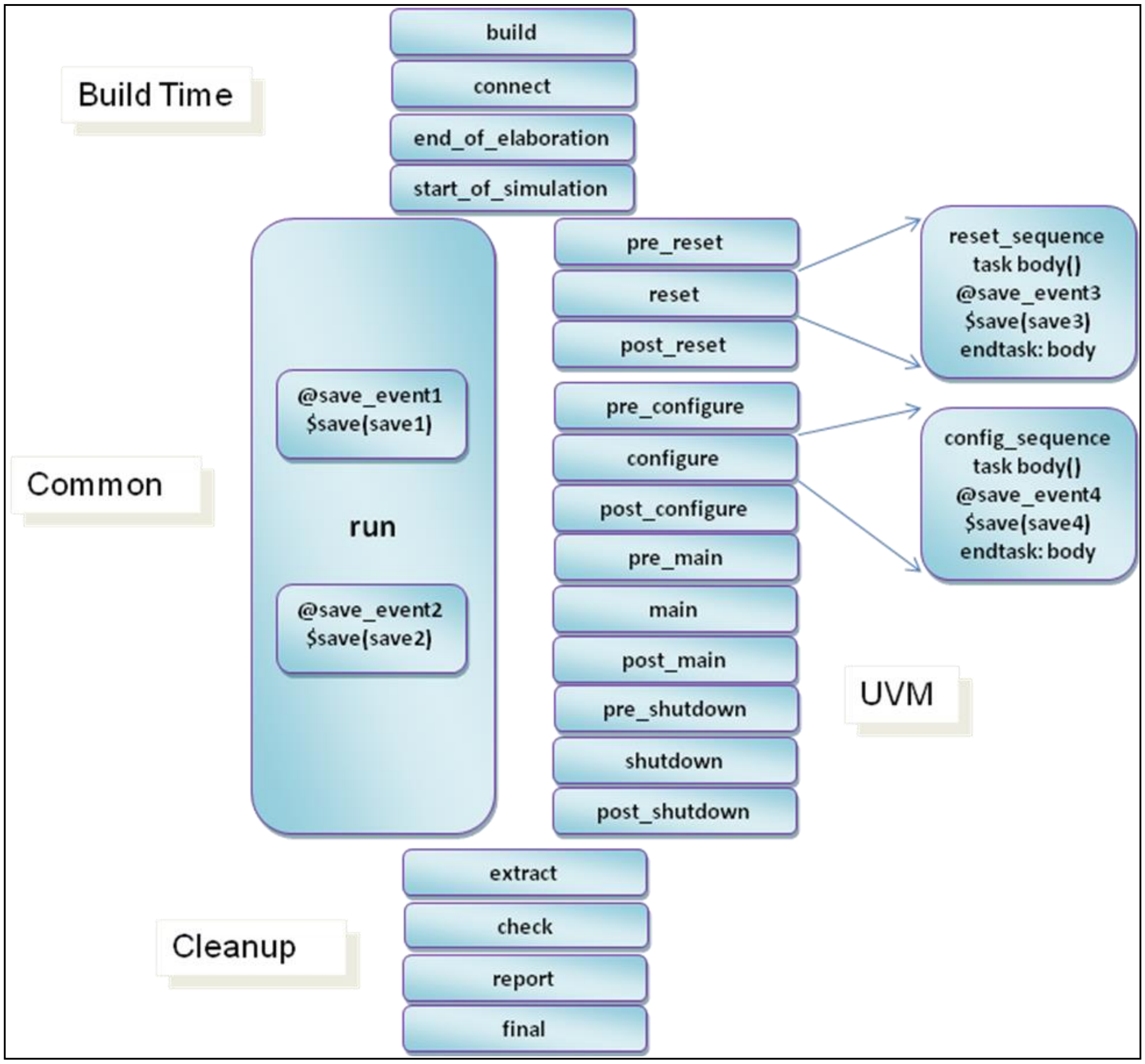

Figure 2. Save/Restore in UVM

3.2 Compilation and simulation flow

Our verification team uses Synopsys® VCS as the simulator. VCS and associated simulator ecosystem supports multiple use-models. Users are enabled to customize as per their methodologies and desired usage models.

The two common flows that VCS supported for Save/Restore feature are:

- UCLI [2] based flow where VCS has a `save` command and a `restore` command. These commands can be incorporated in user's TCL script that can control the simulation.

- IEEE standard system tasks based flow using `$save` and `$restart` system tasks. These system tasks can be added to the System Verilog testbench.

Both flows support re-seeding in an advanced constrained based environment. VCS User Guide[3] has more details on the flows and the seeding and re-seeding.

In our UVM based testbench we chose to use the system task `$save` for creating the simulation snapshot(s). We surrounded the call(s) to `$save` by conditions controlled by runtime `$plusagrs` to give the user the flexibility to enable the Save feature or not. The code looks something like:

```
if ($test$plusargs(“save”)) begin
   $save(“save.chk”);
end
```

Since we used the system task based flow, there was no additional compile time options needed to enabled Save/Restore feature

For running simulation from a saved simulation snapshot; i.e. restoring, we used the option `–r <saved_sanpshot_name>`. The simulation commands looked like,

Running initial simulation

```
% simv  +save –l sim_save_sim.log
```

Running subsequent simulation

```
% simv +UVM_VERBOSITY=UVM_LOW +soc_main_vseq=scn1_vseq –l
scn1_vseq.log -r soc_configure_vseq
```

4 Debugging flow with Save/Restore

Save/Restore feature improves debugging time by a huge factor. Long running tests are very hard to debug and reproduce quickly. With the help of Save/Restore feature and VCS UCLI features, we were able to create snapshots at multiple simulation points and restore using the closest snapshot and proceed further my using UCLI forces to make quick forward progress to overcome minor testbench/logic issues.

For C based tests which are running on the embedded processors, we were able to create a simulation snapshot at a point where the processor/core ready signal is asserted in the design. We then were able to restore from this snapshot, while making updates to the C code. This saved a lot of turnaround times in terms of recompilation and rerunning till the snapshot point.

Some of the challenges faced with using Save/Restore feature was in dumping waveforms with this feature enabled. If the save simulation was launched in a non-UCLI mode then the restore simulation will not support UCLI mode and vice-versa. Similarly, if we use UCLI to dump waveforms then we can’t generate waves for the restore simulation alone which is a useful feature for debugging. These simulator issues should be kept in mind while architecting the testbench.

5 Results

There was an initial one time simulation time overhead to generate the simulation snapshots, which utilized one LSF resource. This overhead was affordable looking at the total performance benefits.

After adding support to run regressions using Save/Restore feature, we saved 30% of simulation time and also increased LSF performance utilization as well. Table 5.1 captures the simulation statistics for a single RTL simulation. Table 5.2 shows the total time saved in running regression using the Save/Restore feature. We can also see from table 5.3, we saved 59.8% of simulation time on a single test on Gate level simulation using Save/Restore feature.

RTL	Elapsed Time	CPU Time	Vir. Memeory
Normal Simulation	4779 sec	4537.3 sec	12236.3 MB
Normal simulation with +save option	4999 sec	4621.3 sec	12245.9 MB
One time performance overhead with +save	4.6%	1.8%	
Simulation with restore from snapshot2 (sys_ready)	3710 sec	3520 sec	12240 MB
Time saved in %	22.3 %	22%	

Table 5.1 Simulation Statistics for a RTL simulation

	Total Time
Normal Regression	30 hr.
Regression with restore from sys_ready	21 hr.
Time saved in %	30%

Table 5.2 Simulation Statistics for RTL regression

Gate Level Simulation (GLS)	Elapsed Time	CPU Time	Vir. Memeory
Normal Simulation	4558 sec	4775.8 sec	104609.5 MB
Normal simulation with +save option	9336 sec	7188.9 sec	104615.6 MB
One time performance overhead with +save	104%	50.5%	
Simulation with restore from snapshot2 (sys_ready)	477 sec	1919.2 sec	104615.3 MB
Time saved in %	89.5%	59.8%	

Table 5.3 Simulation Statistics for a Gate Level Simulation

6 Conclusions and Recommendations

Considering the increasing complexities in the SoC designs and increasing simulation times, Save/Restore feature has significantly improved the turnaround time and debugging efforts. Implementing this feature in our RTL environment helped in porting it to the Gate environment and it proved to be a good investment for returns.

Here are some of the recommendations for new users who are interested in implementing the Save/Restore feature:

1. Understand the design prior to implementing any testbench support.
2. If your simulations are running longer and has a same reset sequence pattern for most of the tests, you should consider implementing Save/Restore feature.

We still have a recommendation for the simulator tool which would increase the overall debugging time. Currently if we recompile the model after the snapshot was generated, the snapshots are all stale and need to be regenerated. But, if only a particular sequence was updated after the snapshot was generated and the model was recompiled, we should have an option to allow the snapshot to use the new recompiled model to run the updated sequence. Of course the user should make sure that the updated sequence will not have any impact on the simulation prior to the snapshot generated.

7 References

[1] VCS Manual $VCS_HOME/doc/UserGuide/pdf/uvm_users_guide_1.1.pdf

[2] VCS Manual $VCS_HOME/doc/UserGuide/pdf/ucli_ug.pdf

[3] VCS Manual $VCS_HOME/doc/UserGuide/pdf/vcsmx_ug.pdf