IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-7 E-ISSN: 2347-2693

Efficient Simulation of SoC based on Design Checkpointing for Efficient Debugging

Nikhil K.M[1*], Veena M.B[2]

[1*]Dept. of ECE, BMS College of Engineering, Visveswaraya Technological University, Bengaluru, India
[2] Dept. of ECE, BMS College of Engineering, Visveswaraya Technological University, Bengaluru, India

[*]Corresponding Author: nikhil.km2@gmail.com, Tel.: +91 9141698810

Available online at: www.ijcseonline.org

Received: 27/May/2017, Revised: 13/Jun/2017, Accepted: 26/Jul/2017, Published: 30/Jul/2017

***Abstract*—** As the complexity of SoC design increases, functional verification of full chip takes more time and to verify the complete functionality of the design, many test cases need to be run. Here the simulation of DUT (Design-under-Test) is an important step to perform functional verification, which may require hours or days together for simulation. This can significantly impact on overall development time and time-to-market. The ability to restore regressions from a previously saved state can reduce considerably the development and debug cycle, thereby enhancing the productivity. This paper details a tool based checkpoint methodology for efficient simulation of Industrial SoC designs. This simulation checkpoint technique can greatly impact on reducing the development and debug cycle. Instead of saving the entire simulation state, only essential state of the simulation can be saved and the regressions are restored from that saved point in more dynamic way. It also describes how to avoid initial setup phase (RESET Phase) which is common for all test cases. These Checkpoint-restore techniques have been implemented to speed-up simulation of register-transfer level models and its effect on applying it to industrial SOC designs. In this Proposed work the experimental results on Industrial designs reveals that reset step which consumes approximately 17 hrs. can be restored in 43 minutes, saving almost 16 hrs. of simulation time for each of the testcases.

***Keywords*—** Checkpointing, Save/restore, simulation based functional verification.

I. INTRODUCTION

Companies designing and developing multiprocessor SoCs (MPSoCs) are progressively demanding verification team to upsurge the productivity by shortening development and debug cycles. Here the simulation speed is of excessive importance for complex MPSoCs platforms and hence, an augmented demand for simulation performance is anticipated. The accumulative complexity of System-on-Chip (SoC) designs, inspired by Moore's law on a single chip, faces novel challenges in the development and debug cycles. Under high time-to-market burden, to shorten product development cycle, system-level sculpting and simulation has turn out to be a essential part of the design method. Efficient system-level simulation of multi-core architectures is a challenging problem as it requires fast and at the same time truthful performance. Therefore, the core challenge is to accomplish system level simulations fast and accurately at the same phase. The Design checkpointing technique offers an opportunity to reduce the number of simulation runs by loading the state of the simulation to a file. Then this checkpoint image is used to restart the simulation in the equivalent state at a later point of time.

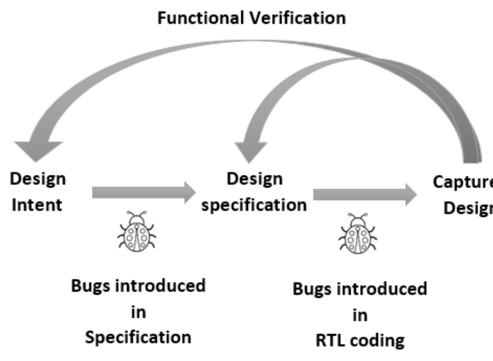


Figure 1. Typical Design Flow from specification to RTL description

Simulation-based functional verification is characterized by two inherently contradictory targets: the signal visibility and the simulation performance [3]. Fig. 1 illustrates a typical design methodology for SoCs. Starting with the intent of design, the SoC architect decides the specification for SoC using human language and it is transcribed into a Register Transfer Level (RTL) description using any Hardware Description Languages (HDL) such as Verilog, VHDL. This transformation process involves use of IPs from third parties or from older projects.

Here in order to ensure the correctness of the design, each step needs to be verified and any disparity in the between two consecutive steps is known as a design error or a bug. Implementation procedure is the important step in this process where bugs are introduced due to misinterpretation; such bugs can be eliminated with the assistance of more reliable vendor tools [1]. Design errors introduced to the specification and the captured design are identified as functional bugs which results in malfunctioning of the chip. Hence functional verification is implemented for identifying and fixing functional bugs so that the captured design meets the design intent [2].

The waveform generation time can be in days for handling the bigger designs. To manage it waveforms are generated for limited simulation time and limited hierarchies. This increases the time and effort in root causing the bug. In addition to that tools will crash occasionally and disk space will run out at critical times to augment to difficulty.

A simple issue can cost heavily and the time it takes to root cause bug can become unknown. Since the eventual goal of verification is to find bugs in the most efficient way and hence the amount of time spent on functional verification is key in overall cost equation.

A. The Cost of Verification

Verification of given design is an unavoidable evil which takes long time and costs more. Verification will not directly generate the revenue; the design being verified and taped out chip eventually makes money, not the verification. In order to market product and create revenues the design must be functionally correct without any bugs and must benefits the customer necessitates.

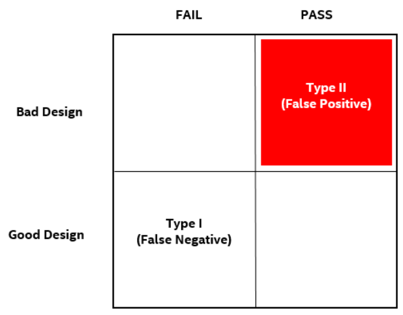


Figure 2. Type I & II mistakes in typical SOC Verification Process

Usually verification engineers encounter two types of errors in functional verification process: Type I and Type II mistakes shown in Fig. 2. Type I mistakes are called false negatives which are the easy to detect. When the misinterpretation is recognized, the implementation of the design is altered to mask those mistakes and the mistake no longer occurs. Type II mistakes are called false positives are the most critical ones. In such situation a corrupt design will be shipped accidentally, with all the potential penalties to the company.

B. Time-to-Market Trends

During the development of complex SoC designs for any applications such as military, industrial, and consumer grade electronic devices, it demands intense reductions in the time-to-market. In order to meet time to market trends, design cycles are shrinking for all type of applications. The complex SoC development technology and market challenges are greatly influenced by new verification methodologies and tools. A verification task consumes major percent of the total development work. Hence these verification deeds have to be done more efficiently in order to meet overall market challenges.

Rest of the paper is organized as follows, Section I contains the introduction of checkpointing and simulation, Section II contain the related work of Checkpointing, Section III contain the implementation details of checkpointing in simulation and Section V concludes research work.

II. RELATED WORK

Software based checkpointing is an extensively used method to create snapshots during a simulation [5]. Commercially available simulators have implemented Software based checkpointing and it has a consistent interface within the Verilog language via built-in methods (e.g., $save, $restart) [6]. The way of implementation of this type of checkpointing mechanism in each of the commercially available simulators from synopsis, cadence and Mentor graphics fluctuates in terms of disk space necessities and the stored information. If any simulation follows same characteristics for several hours of simulation then the saved image up to common phase can be shared across all tetscases which takes longer duration for the simulation. For System Level verification it takes more than a day for simulator for model bring up in RTL and then need to start various testing scenarios for DUT. This initialization phase can be saved to disk and simulation can be resumed directly from saved state. Typically, once a bug is exposed after hours or days of simulation, the simulation need to be repeated for hours or days to verify the bug fix which can be avoided by this checkpointing technique.

Simulation Checkpointing is a process by which simulator stores the state of the simulated system to disk and later it is restored from the saved checkpoint into the simulator as shown in fig. 3, which results in exact simulated state.

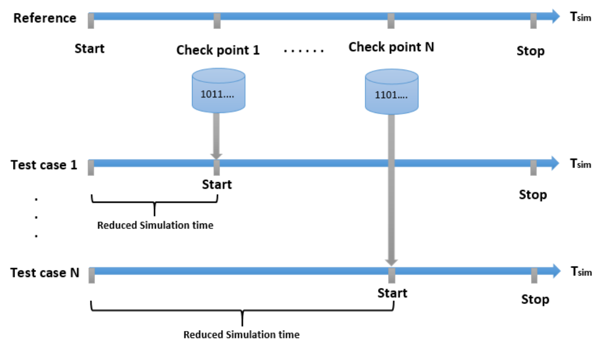


Figure 3. The checkpoint-restore mechanism for various testcases.

simulation at any point and resume it later. This checkpointing method is used to decrease the functional-verification effort of SoC Designs. A fresh simulation can be resumed from the saved state thus effectively speeding up the SoC verification. Here the state of simulation can be saved at any simulation time stamp and restored in order to skip hours or days of simulation and authenticate a bug fix. This allows speedy verification of bug fixes and enables longer simulations by replaying previously validated code. Furthermore, the file generated from the checkpointing technique requires a significant amount of disk space. Checkpoints are generated during the long runs can exceed available disk space for the projects.

The most significant usage of the Checkpointing - Restore feature in SoC simulations are:

- Time saving - If a simulation runs for longer duration and fails at final stage, in this case the user need not to re-run the same simulation in order to reach a failed state, but it can be re-loaded from previously saved state by using checkpointing technique.
- Simulation transfer - one can transfer a checkpoint image to anyone in order to make use in his/her simulation.

III. Methodology

In Checkpointing, we need to determine the correct place in our design which can be used as the end of the common setup phase. This varies depending on the style of the design. It could be based on a fixed simulation time, or a fixed location in the testbench, or it could be dynamic and programmed based on the activity in the design.

We have implemented checkpointing of our design in two ways.

1. Based on number of clock cycles.
2. Based on Events in Test environment.

In checkpointing based on number of clock cycles we can checkpoint our design based on user specified number of clock cycles at run time, which can be implemented using +plusargs support. This method provides the fault tolerance for regressions and useful when the user have no idea about where the program will pop out an error and stop simulation at any point of time.

Here we have include options of removing the stale checkpoints in order to address the disk space requirements, so that we can retain only user specified number of checkpoints before the failure.

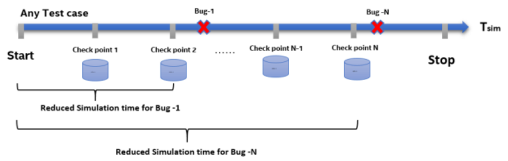


Figure 4. Periodic checkpointing based on specified number of clock cycles

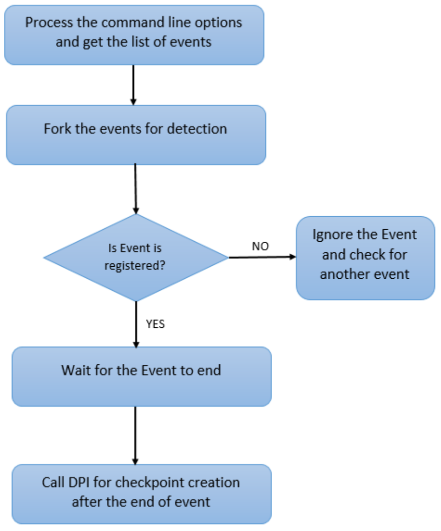
Fig. 4 illustrates periodic checkpoint generation; this shows how efficiently we can use cycle based checkpointing to fix a bug that has occurred in functional verification. One can specify the number of checkpoints that has to be retained prior to the current simulation time stamp, this will help in restoring our simulation from that point where the bug has occurred when the test fails. When the user encounters the Bug-1 at some point of time regression halts then the user can resume the regression from check point 2 after fixing the issue. This method will be very useful when the regressions take more time to complete. For example if a regression takes 48 hours to complete but if it is failed at 47th hour. Then this checkpoint mechanism can save lot of time in SoC development cycle. But this will add overhead for simulation time as many checkpoints are created and deleted in the corresponding regressions. And this will not add more overhead to disk space as old checkpoints will be deleted as 3 the new checkpoints are created. But cycle based simulation checkpoint will be slow when compared to usual runs and it mainly depends on the specified time interval in command line. Creation of checkpoint is invoked with the help of Verilog task whenever it has to be done in simulation environment. Fig. 5 shows the pseudocode for this method.

```
initial
begin
if ($value$plusargs ("SAVE_ENABLE=%d",
chk))
$display ("**** SAVE ENABLED ****");
if ($value$plusargs("SAVE_INTERVAL=%d",
save_interval))
$display ("Specified Interval for SAVE is
%d ns ", save_interval);
end

// checkpoint every specified time units
always
begin
// waits for the user specified interval
#save_interval
// save some old restarts
$system("mv -f save.chk save.chk.1");
// Call Verilog sysytem task for save the
state of simulation
$save ("save.chk");
$display ($time, "Finished SAVING....");
end
```

Figure 5. Pseudocode for cycle based checkpointing

In Event based checkpointing we can checkpoint our design before the start of User defined phases of UVM if that regression has path cleared the same. And we can play around different sequences for our test environment. This adds to productivity by avoiding the redundant runs for every test. The methodology works once the user has identified a set of common simulation steps for a set of regressions tests. And we can checkpoint our design based on several verilog events and various phases of UVM whose algorithmic flow is illustrated in figure 6.


Process the command line options and get the list of events
Fork the events for detection
Is Event is registered?
NO
Ignore the Event and check for another event
YES
Wait for the Event to end
Call DPI for checkpoint creation after the end of event

Fgure 6. Algorithm flow for Event based checkpointing

In this method a Perl script is used to process the command line options and it intimates about the check pointing information to the system verilog interface file. If the specified events have registered in factory database, then interface file will be aware about the occurrence of events and waits for the phase or events to finish. When the specified phase or event has been ended check pointing functions will be called through DPI support. During restore operations restart script will be called, which contains all the information to restart the simulation from saved point.

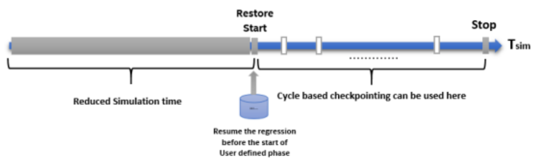

Restore Start
Stop
Tsim
....
Reduced Simulation time
Cycle based checkpointing can be used here
Resume the regression before the start of User defined phase

Figure 7. Efficient usage of Phase based and cycle based check pointing.

Checkpoints can be created even during restore runs as illustrated in fig. 7 and in the restore flow it is possible to enable the signal dump for all signals. During the restore flow, the sequence executed at a particular phase can be changed this will add more robustness to this methodology which is shown in figure 8.

```
my_test_init_seq init_seq;
my_base_test_seq test_seq;
initial
begin
uvm_report_info(get_full_name(),
$sformatf(Restored simulation with
test_seq = %s, seq_name));
if ($test$plusargs(seq=))
begin
($value$plusargs(seq=%s, seq_name))
test_seq = my_base_test_seq::type_id::
create(seq_name);
test_seq.start(seq_name);
end
end

if ($value$plusargs(TESTNAME=%s, name))
run_test(name);
else begin
uvm_report_info(get_full_name(), No test
name specified. Running default test
mytest);
run_test(mytest);
end
```

```
//compilation
vcs -sverilog uvm +incdir+../sv
file_name.sv -debug

//simulate to common state
./simv +UVM_TESTNAME=test_case1 -ucli -i
save.chk

//simulate different stimulus
./simv +seq=init_seq -ucli -i save.chk
```

Figure 8. Pseudo code for verifying the DUT for different stimulus using checkpointing advantage

Users can also checkpoint and restore the design state on different machines. The users should ensure that the machine where design state is being checkpointed and the machine where design state is restored should have similar OS specifications. Different OS systems may have different memory mappings for low level system libraries and can cause issues during state restore.

IV. Results and Discussion

The proposed methodology has been successfully implemented and tested on Industrial designs for various testcases. We have observed remarkable speedup in simulation. With the design scenarios where initial setup phase time has a higher percentage of overall time, the amount of throughput will be higher. It provides a valued impact to the verification engineer's productivity by shortening their typical development - debug cycle.

The RTL simulation of large SoC designs with a Verilog HDL simulator, the initial setup phase alone consumes 17 hrs. which can be restored in less than one hour using this checkpointing technique. Now one can play around with different stimulus to verify the complete functionality of DUT.

V. Conclusion

This method offers an effective method for the simulation of complex SoC designs based on checkpointing to speed-up the simulation process and to increase the debugging productivity with a small overhead for the first regression. The verification engineers can easily integrate this method into their existing verification environment with very less modification in the testbench code. The signal visibility can be achieved without sacrificing simulation speed.

Therefore, the SOC design verification which needs much lengthier simulation cycles can be overcome by this proposed approach, which results in a very good overall improvement in the verification productivity. In this proposed work the experimental results on industrial designs have shown more than 20x simulation speed-up when compared to a conservative simulation method.

In this method if the check pointed models are changed (e.g., signals added or removed) limits the reuse of saved state and sometimes and it is reused only in limited conditions. This significantly decreases its effectiveness while debugging. Also tool supported checkpoint mechanism suffers from lack of flexibility in using it.

References

[1] Kyuho Shim, Youngrae Cho, Namdo Kim, Hyuncheol Baik, Kyungkuk Kim, Dusung Kim, Jaebum Kim, Byeongun Min, Kyumyung Choi, Maciej Ciesielski, Seiyang Yang, "*A Fast Two-pass HDL Simulation with On-Demand Dump*", in Design Automation Conference, 2008 (ASPDAC 2008) Asia and South Pacific, 21-24 March 2008.

[2] K. Arya, R. Garg, A. Y. Polyakov, and G. Cooperman, "*Design and implementation for checkpointing of distributed resources using process level virtualization*", in IEEE Int. Conf. on Cluster Computing (Cluster16). IEEE Press, 2016.

[3] M. Monton, J. Engblom, C. Schrder, J. Carrabina, and M.Burton, "*Checkpoint and Restore for SystemC Models*" in Advances in Design Methods from Modeling Languages for Embedded Systems and SoCs. Springer, 2010.

[4] Bogdan- Andrei Tabacaru, Moomen Chaari, Wolfgang Ecker, Thomas Kruse, and Cristiano Novello "*Efficient Checkpointing-Based SafetyVerification Flow Using Compiled-Code Simulation*", Euromicro Conference on Digital System Design, 2016.

[5] B.A. Tabacaru, M. Chaari, W. Ecker, T. Kruse, and C. Novello, "*Comparison of Different Fault-Injection Methods into TLM Models*", Resiliency in Embedded Electronic Systems (REES), 1st International ESWEEK Workshop on, pp. 16, 2015.

Authors Profile

Mr.Nikhil K.M pursed B.E in Electronics and Communication from JNNCE, Shivamogga affiliated to VTU Belagavi. He is currently pursuing M.Tech in Electronics under Department of Electronics, BMSCE, Bangalore. His area interests include VLSI, Embedded systems, SoC deign and verifiction.

Dr. Veena M.B pursed B.E. & M.E., degree in Electronics & communication Engineering from University of Mysore, & university of Bangalore respectively, and Ph.D. degree in Electrical and Electronics engineering from V.T.U, Karnataka, India. Presently working as Associate professor in the Department of Electronics & communication Engineering, BMSCE, Bangalore. Her current research interests include VLSI, Wireless communication, Embedded systems, Signal & Image Processing. She is a Fellow member of Institution of Engineers (India) and is a Life Member of the Indian Society for Technical Education (ISTE).