2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Simulated Emulation: Enabling Multiple Iterations in a Day During Early-Stage Emulation Bring-up

Shalini Maheshwari
Synopsys Inc
Hyderabad, India
shalinim@synopsys.com

Ashok Kumar Bhatt
Synopsys Inc
Noida, India
ashokb@synopsys.com

***Abstract*- SEM [Simulated Emulation] uses simulation to approximately replicate the behavior of Emulation. It enables users to sanitize the emulation model including the RTL, Transactors, compile configuration, and runtime testbench, targeting the first few million cycles for quick validation. The faster compile times of the simulation tool result in a much lower turnaround time, considerably improving productivity in the early design stages when the model changes frequently. It is used to debug fundamental design bring-up issues such as RTL connectivity, Power-on reset generation, clock generation, and basic modeling issues. Debug features of simulation may be used for flushing issues. Different modes of Transactors and Clocks are supported. This paper illustrates SEM as a methodology, applied in the context of Synopsys verification products VCS and Zebu. It describes two modes SEM can operate in, different use-cases the methodology has been leveraged in, and a peek into the performance gains.**

Keywords— Emulation, Verification, Shift-left

I. INTRODUCTION

The compile times in FPGA based emulation flow can be very long - especially the netlist generation, partitioning, and FPGA compile to generate bitstream consume considerable time even for small designs. In the initial stages of emulation bring-up, when the design or test environment configuration may change frequently, such modifications require repeated compilation with very few (or sometimes none) runtime cycles. Further, due to hardware architectural constraints, debugging on an emulation platform poses its own challenges and may require additional recompilation cycles. As a result, the user may be able to turn around only one iteration of changes in a day, and it can take up to several days or weeks until the model is stable enough to proceed with the actual emulation testing.

SEM addresses this concern by creating a methodology that enables users to run multiple iterations in a day during these early stages, leading to faster bring-up of designs in emulation. SEM accepts the same test environment as emulation, compiles the design for simulation, and aims to closely reproduce the emulation behavior in simulation. The objective is not to run the entire emulation test but only the first few million cycles to sanitize the new RTL drops and to verify the RTL connectivity, reset generation, clock connection and generation, etc., in order to quickly fix and validate the model.

The simulation compile time is significantly lower than emulation, drastically reducing the turnaround time for each iteration. SEM can take advantage of performance optimization techniques of the simulator for both compile and run time, as well as the debug capabilities of simulation. In simulation, it is possible to dump a large number of signals, possibly the entire design, without incurring a severe penalty. Further, hardware is expensive and the capability to sanitize the environment purely in software has an additional benefit of cost saving. However, the runtime performance of simulation is much slower compared to emulation, because of which this methodology is recommended only for shorter test cycles. Once the model is validated in SEM, the user can continue using the same setup for emulation.

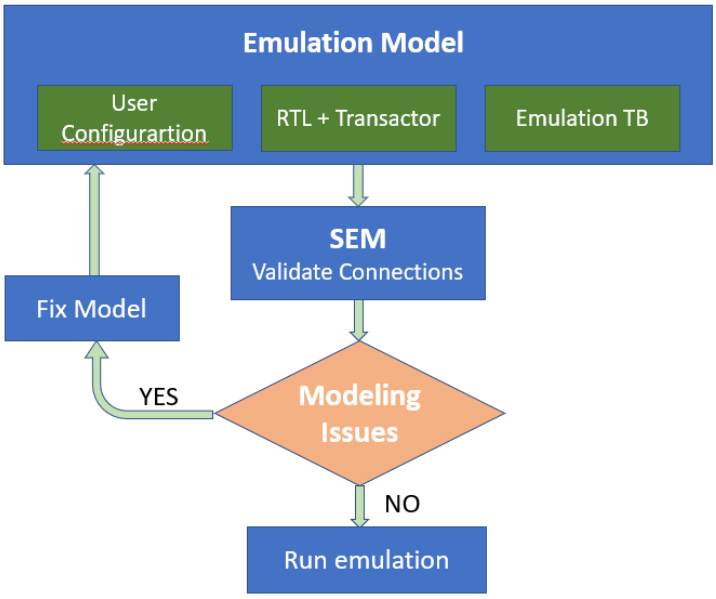

Fig. 1: SEM: Simulation-Emulation Continuum

	Flow	Compile Time (hrs)	Run Time (hrs)	Total Time (hrs)	TAT Gain
Case Study #1	Emulation	11:36	0:18	11:54	7.8X
	SEM	0:56	0:35	1:31	
Case study #2	Emulation	12:45	0:15	13:00	3.2 X
	SEM	1:30	2:40	4:10	
Case Study #3	Emulation	14	0:16	14:16	2.3 X
	SEM	1:15	4:50	6:05	

Fig. 2: TAT gain with SEM-Behavioral in one iteration

TAT gain depends on design size and activity, and can therefore be varied.

II. EMULATION CONCEPTS IN SEM

SEM enhances the simulation to approximate the functional behavior of emulation, and some of the key concepts that are addressed to achieve this are as follows:

- Uncontrolled Clock: The emulator generates the uncontrolled clock depending on the layout and timing characteristics of the design mapped on the hardware, and all user/controlled clocks are generated w.r.t. this. SEM can model an uncontrolled clock and generate the user clocks on its basis, to recreate the same functional ratios between the clocks.
- Controlled (user) clock: Emulator firmware allows a user to generate a clock with a specific relationship to other controlled clocks. SEM creates a simulation model for controlled clocks which can be instantiated and run in simulation.
- Timed Clock: Inherently, hardware does not have a notion of time, but the support of delay clocks in emulation offers a two-fold advantage. It eases migration from simulation to emulation and allows comparison of the functional behavior of simulation and emulation w.r.t. time. A firmware-based

mechanism allows the concept of #delays in emulation. SEM relies on the native simulation time but creates a simulation model for the firmware primitive.

- Xtor: Xtors facilitate the communication between hardware (design) and software (testbench), representing the interchange as a meaningful transaction rather than a transfer of data. In a DPI Xtor, the user encodes the hardware/software interchange in terms of DPI calls. The compiler translates DPIs into firmware primitives and transforms behavioral RTL into synthesizable code. SEM can leverage the compiler instrumentation to incorporate the same behavioral transformations as emulation.

III. SEM METHODOLOGY AND MODES

In an emulation test, the RTL design runs on one of the many hardware platforms, while the testbench runs on a Linux host connected to the hardware. SEM may be considered as a special case of the emulator, where the RTL runs in simulation and the simulation executable stands in for hardware.

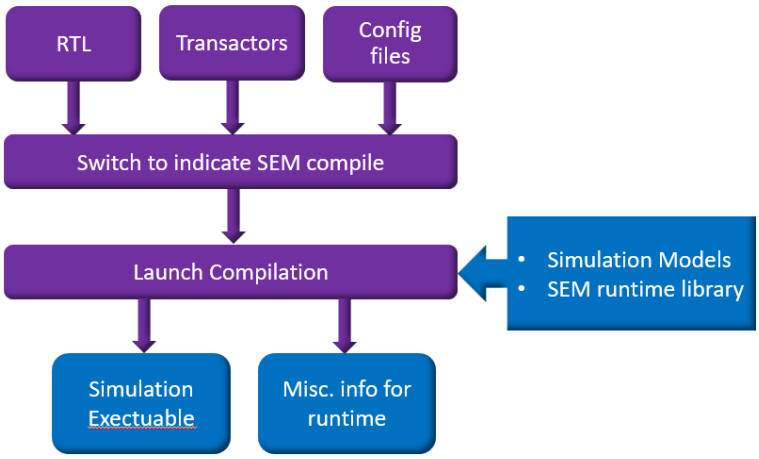

Fig 3: SEM compile flow: No use-model change for the user

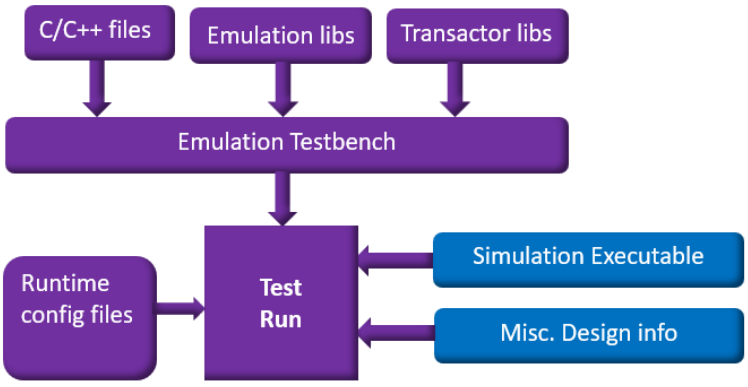

Fig 4: SEM testbench compile flow: No use model change for the user

SEM is available in two modes, and users may decide on the one most suitable for their use-case:

A.SEM-Cycle

This mode preserves cycle-accurate behavior, i.e. it matches the emulator functionality w.r.t. controlled clocks. It models the uncontrolled clock and can support either of the two clocking mechanisms. It uses the behavioral

compiler to compile DPI Xtors and also offers a way to simulate custom Xtors. In this mode, simulator and testbench run in separate processes synchronizing through IPC (Inter Process Communication) mechanisms. Two process architecture allows non-blocking run as the design can run concurrently with the testbench. However, matching the cycle-accurate behavior comes at the cost of performance.

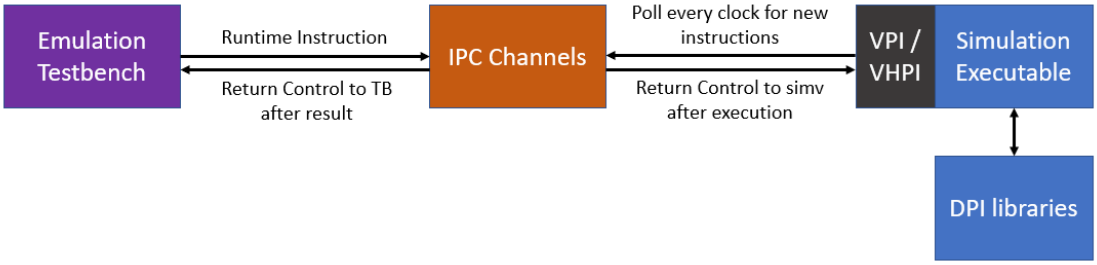

Fig 5: SEM-Cycle runtime architecture

B.SEM-Behavioral

This mode matches the functional behavior of the emulator but does not aim to conform to the clock cycles. It operates in Timed Clock mode and simulates DPI Xtors through native DPI support in the simulator. The simulation code is linked-in with the testbench, and the entire model runs in a single process. Avoiding the need to synchronize allows much faster performance in this mode. Since there is no need for cycle accuracy, it does not support controlled clocks. Due to single process architecture, it cannot support non-blocking run and can simulate the design in blocking run mode only.

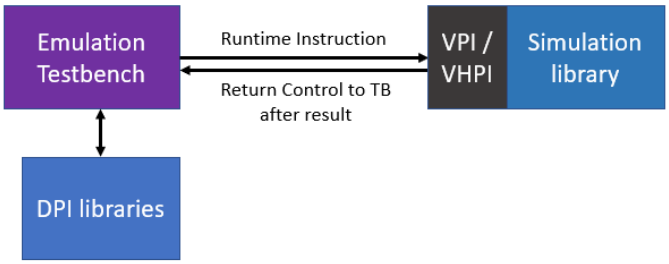

Fig 6: SEM-Behavioral runtime architecture

IV. USE-CASES

A. Transactor validation

Unit tests for validation of Transactors can be run very fast in SEM mode, allowing for small turnaround times during development, and quick validation of changes before a full emulation run. The user interface for simulation and emulation are different, so testbench changes are required between the two flows. SEM offers the advantage of allowing the use of the same testbench as emulation, rather than a different simulation testbench.

B. Gate-level simulation

We have an ongoing in-house project aiming to simulate the backend netlist using SEM, to reduce the burden on emulation resources while detecting issues in netlist synthesis. Issues that may arise due to incorrect translation of RTL to netlist may be detected in simulation, saving the time for FPGA mapping and more expensive emulation resources.

C. 2-state and 4-state Simulation

Emulation inherently operates in 2-state mode whereas simulation honors 4-states. With SEM, it is possible to validate the design in 2-state mode to match emulation or in 4-state mode to match simulation behavior.

V. CONCLUSION

The long compile times of FPGA-based emulation are a big overhead in the early stages of emulation setup when the model changes frequently. This paper describes SEM methodology, which incorporates the advantages of simulation in the emulation environment to reduce the iteration time for changes in the emulation model. Faster compile times and better debug capabilities of simulation allow users to reduce the emulation bring-up time for their designs, debug issues in the emulation model more easily and optimize the use of emulation hardware. Its great potential in several use-cases makes it an important tool in the shift-left paradigm in the simulation-emulation continuum. It does not propose to replace emulation, as the runtime performance of simulation is far slower, and it is difficult to completely match emulation behavior in simulation.

VI. FUTURE IDEAS TO EXPLORE

A. Regressions

SEM can be leveraged for regression validation in place of emulation. It can be particularly advantageous for unit testcases, quick validation, user regressions, etc. Full emulation runs may be scheduled on a reduced frequency, e.g., for sign-off validation or nightly runs, leading to more effective usage of emulation resources.

B. Backout Analysis

Determining the culprit for a regression in emulation tests using backout analysis can be time-consuming and expensive. The effort of analyzing breakages caused in netlist synthesis can be reduced significantly using SEM for backout analysis.

C. Avoiding Simulation Races and other simulation-emulation mismatch conditions

Emulation is inherently race-free, and therefore emulation-ready designs may use RTL modeling styles that could cause simulation races. Built-in checks to notify such conditions or the ability to bypass them can save debug time in SEM run. Other modeling styles that can cause mismatch in results can also be reported, such as multiple drivers, incomplete sensitivity lists, etc. though most of such checks fall in the purview of Lint tools.

D. Dynamic Software-Hardware Swap

Running the design on SEM further paves the way to dynamically swap the design between Simulation and Emulation. This usage will allow the user to run a few hundred cycles on simulation and then switch the design to emulation. This provides an easy way to allow initial blocks in the design. The capability to dynamically swap the design between simulation and emulation enables easier debugging by using almost all simulation runtime debug aids.

REFERENCES

[1] Emulation Systems | Synopsys Verification https://www.synopsys.com/verification/emulation.html
[2] High Performance Simulation | Synopsys Verification https://www.synopsys.com/verification/simulation.htm