A Novel Approach to Accelerate Latency of Assertion Simulation

Jack Yen, Synopsys Inc., Hsinchu, Taiwan (*jackyen@synopsys.com*)

Felix Tung, Synopsys Inc., Hsinchu, Taiwan (*yatung@synopsys.com*)

***Abstract*—RTL assertions are used in assertion-based verification (ABV) to capture design intent and provide portable monitors that check for correct behavior. Assertions improve observability coverage and error localization during simulation, reducing debug time significantly. However, ABV simulation can be slowed down by extra simulation events. In this paper, we present a novel approach to accelerate ABV simulation. We first create a light-weight ABV environment to minimize the number of simulation events. Then, we decouple the Testbench/DUT and the assertions. This allows us to record the Testbench/DUT simulation stimulus to waveforms and replay them to the assertions. Finally, we enable parallel waveform replay with a smaller number of assertions for each run. Our experimental results show that our approach can reduce the assertion simulation overhead by 57% on average.**

Keywords—Assertions; ABV; RTL Simulation; Verification and Validation

I. INTRODUCTION

ABV (assertion-based verification) [1] involves the use of assertions, which are formal statements that capture specific design properties or constraints. These assertions are written in separate modules or embedded within the design code. The purpose of ABV is to express the expected behavior or design intent and automatically check if the design adheres to these specifications. By incorporating assertions, designers can detect errors, bugs, or violations of design requirements early in the verification process. This helps to ensure the correctness and reliability of the design before it is implemented.

ABV can be executed through formal verification or simulation techniques. In ABV simulation, assertions are continuously monitored, and if violated, they promptly trigger alerts. The advantage of debugging and rectifying assertion failures lies in their proximity to the source of the issue, making the process more efficient compared to tracing back the root cause. However, it's important to consider that ABV simulation may introduce a drawback, as it can lead to increased simulation latency. This can pose a cost when running ABV simulations within nightly regressions, where time efficiency is crucial. Figure 1 illustrates a study of run time comparison with and without assertions in a regression suite. It shows the ABV simulation can take 68% extra run time on average.

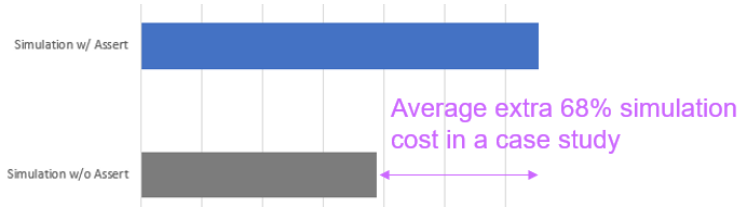

Figure 1. A study of run time comparison w/ and w/o assertions in a regression suite.

In this paper, we introduce an innovative method to enhance the speed of ABV simulation latency. Our approach involves decoupling the Testbench/DUT simulation from the assertion simulation. After completing the Testbench/DUT simulation, we capture the required signals' waveforms, which serve as input stimuli for the assertion simulation. Furthermore, by partitioning the assertions into smaller groups, we can parallelize the simulation of each group. This significantly reduces the simulation cost since each group is smaller in size compared to the entire set of assertions.

II. A NOVEL FLOW TO SPEED UP ABV SIMULATION LATENCY

A. An ABV Simulation Environment in Regression

Figure 2 depicts an ABV simulation regression environment comprising a Testbench, DUT, and Assertions. Within this environment, numerous tests are executed in the simulation, each involving the monitoring and evaluation of thousands of assertions. It is important to note that the presence of these assertions introduces a significant increase in simulation latency, adding to the overall runtime of the tests.

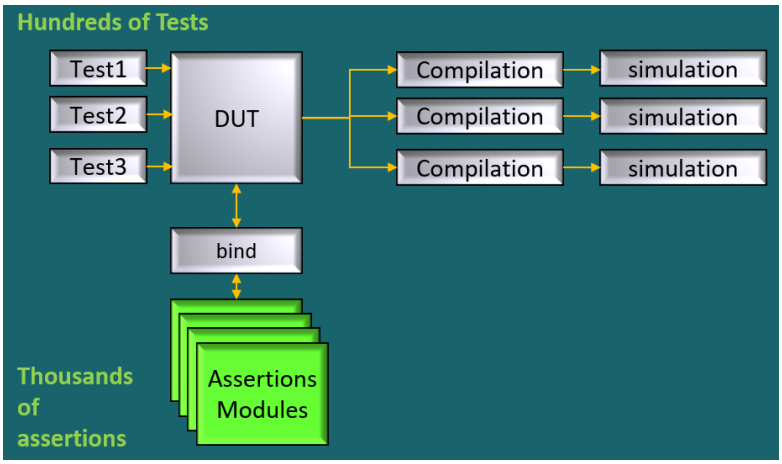

Figure 2. An ABV simulation regression environment

B. Craft a Light-weight ABV Simulation Environment

In order to achieve the decoupling of the Testbench/DUT and assertions, it is necessary to establish a lightweight ABV simulation environment. Within this environment, the statements in the Testbench and DUT become unnecessary, with only the design hierarchy and relevant signals retained in the DUT. These signals serve the purpose of driving the assertions.

To facilitate this process, a light-weight ABV generator has been developed. It parses the HDL design and generates a dummy Testbench (TB) and a skeleton DUT, as depicted in Figure 3. The approach followed by the light-weight ABV generator is illustrated in Figure 4, while Figure 5 provides an example of a module within the skeleton DUT.

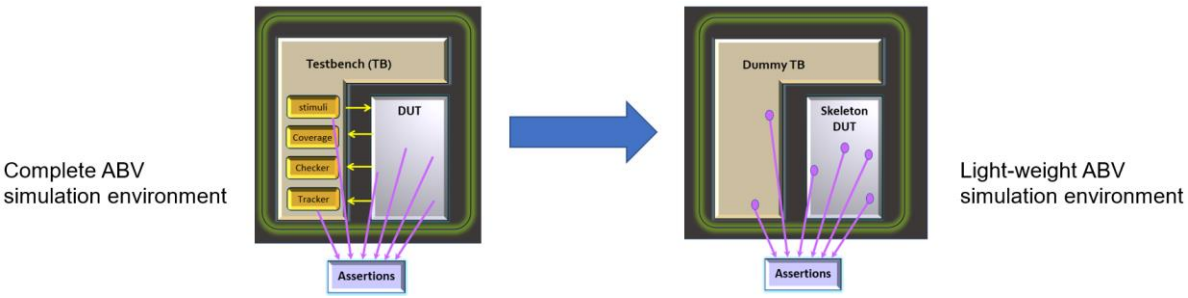

Figure 3. A light-weight ABV simulation environment.

```
def light_weight_ABV_generator ( HDL_Design ):
        for module in iterate_design_hierarchy ( HDL_Design ):
                skeleton_mod = generate_skeleton_module_with_signals_only( module )
                generate_sub_module_instantiation( skeleton_mod, module )
                write_out_skeleton_module (skeleton_mod )
```

Figure 4. A method to generate a light-weight ABV environment.

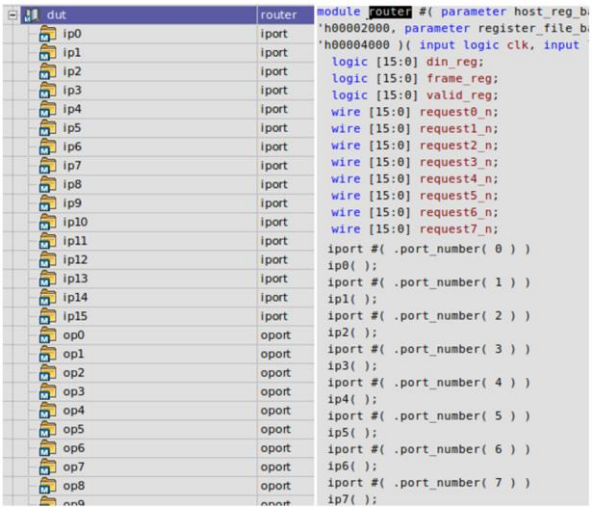

Figure 5. An example of a module in the skeleton DUT

C. Decouple Testbench/DUT and Assertion Simulation

With our lightweight ABV environment, we can modify the original setup by disabling assertions and conducting TB/DUT simulations while simultaneously capturing the required signals' waveforms (FSDB), as depicted in Figure 6. These "necessary" signals are derived from the signals declared in the TB/DUT and serve as drivers for the assertion simulations. Two sources contribute to these signals: the signals connected to the assertion modules' interfaces and the external reference signals utilized within the assertion modules. By traversing the statements of each assertion module, we can extract the relevant signals. The process of extracting these signals to drive the assertions is illustrated in Figure 7.

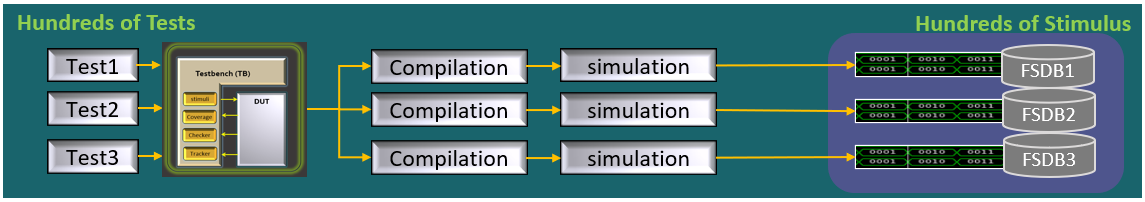

Figure 6. Running TB/DUT simulation and dumping signals waveform for assertions.

```
def assertion_driving_signal_extraction ( HDL_Design, Assertion ):
        assert_driving_signals = []
        for statement in iterate_assertion ( Assertion ):
                driving_signals = [signal for signal in statement if signal in HDL_Design]
                assert_driving_signals.append( driving_signals )
        return assert_driving_signals
```

Figure 7. A method to extract assertion driving signals in DUT

Once the waveforms have been obtained, the subsequent step involves simulating the light-weight ABV environment by replaying the captured signals' waveforms, as illustrated in Figure 8. This necessitates the use of a waveform replay engine. The methodology employed by the waveform replay engine [2] can be understood by referring to the algorithm outlined in Figure 9. Essentially, the engine iterates through the driving signals' waveforms in a time-based manner. As the simulation time progresses, if there are any value changes occurring at that particular time, the waveform replay engine updates the signal's value to simulate the ABV environment accordingly.

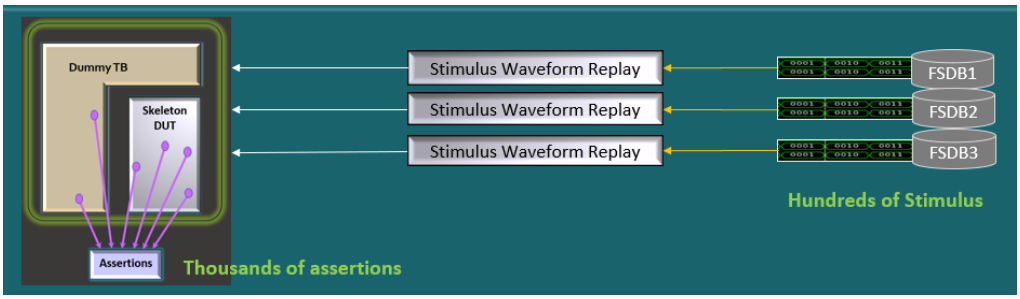

Figure 8. Simulate the light-weight ABV environment with waveform replay.

```
def waveform_replay ( Light_ABV, Waveform, Assert_Driving_Signals ):
        for signal, value, time in iterate_waveform_in_time_based ( Waveform, Assert_Driving_Signals )
                put_value_to_simulation ( Light_ABV, signal, value, time )
```

Figure 8. A method to waveform replay.

D. Parallel Assertion Simulations with Waveform Replay

The utilization of the light-weight ABV environment brings significant advantages, primarily due to our prior knowledge of all the assertion driving signals' waveforms. Consequently, we can divide the assertions into smaller groups. During each test, we can simultaneously execute the light-weight ABV simulation for these groups in parallel. By reducing the number of assertions in each group, we effectively reduce the simulation latency. Leveraging parallel runs further contributes to decreasing the overall elapsed time. The flow of this process is illustrated in Figure 9.

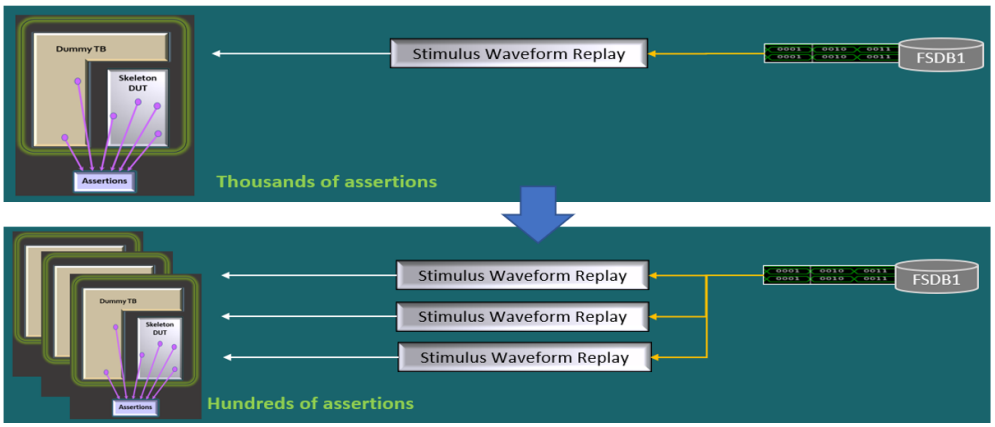

Figure 9. Parallel assertion simulation with waveform replay

III. A CASE STUDY

We incorporated an IP-level ABV simulation regression environment into our workflow. The regression suite consists of 127 tests, with each test utilizing 1000 assertions. The longest simulation test duration is approximately 2890 seconds, while the shortest one takes around 570 seconds.

Figure 10 provides a runtime comparison among three scenarios: (1) Simulation with assertions, (2) Simulation without assertions, and (3) Simulation with assertions using our approach. We consider (1) as the baseline. The experimental results demonstrate that our approach achieves a 24% reduction in simulation latency, with a maximum reduction of 34%.

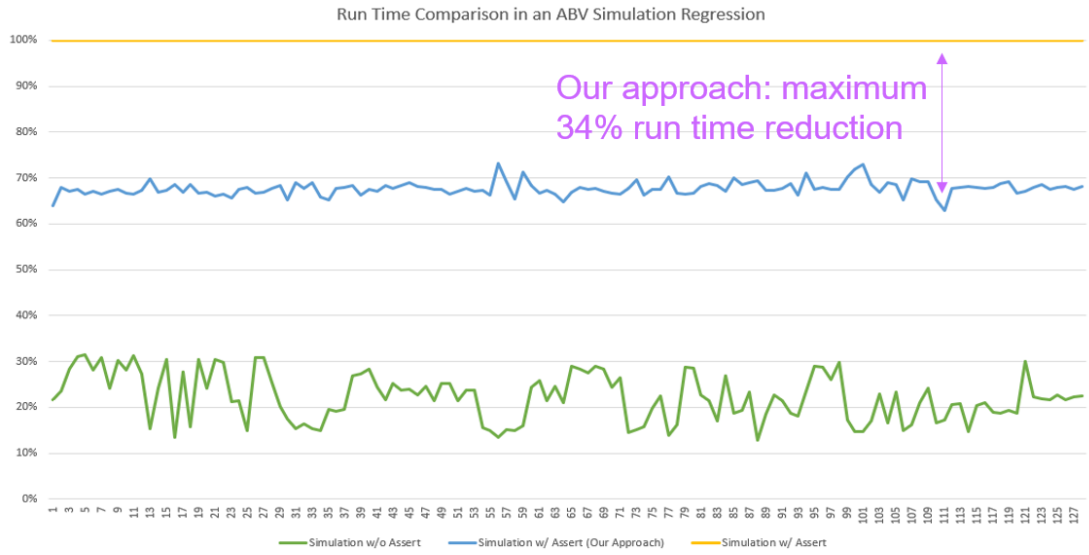

Figure 10. Run time comparison in an ABV simulation regression suite.

When considering only the simulation cost of assertions, Figure 11 presents a comparison between the original ABV environment and our approach of replaying waveforms in the light-weight ABV setup. The effectiveness of the runtime speedup is significant, as we achieved an average latency reduction of 57% and a maximum reduction of 75% in specific cases.

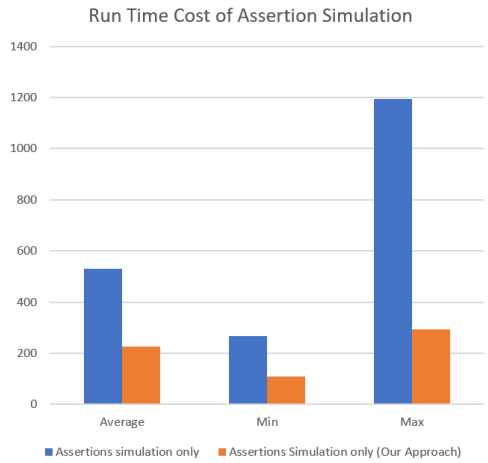

Figure 10. Run time cost comparison only considering assertions simulation.

IV. SUMMARY

We present a novel approach to improving ABV simulation latency by decoupling the Testbench/DUT and the assertion simulation. The utilization of a light-weight ABV environment, where the simulations are performed using recorded signals' waveforms, demonstrates significant reductions in simulation latency. The experimental results validate the effectiveness of the proposed approach. These findings highlight the potential of the proposed approach in enhancing verification efficiency and reducing runtime in the design process.

For future work, several avenues can be explored. Firstly, further optimization of the waveform replay engine and partitioning strategies can be investigated to achieve even greater reductions in simulation latency. Additionally, the proposed approach can be extended to handle more complex designs and larger sets of assertions. Furthermore, the impact of the approach on different verification metrics, such as coverage and observability, can be analyzed to assess its overall effectiveness. Lastly, exploring the integration of the proposed approach within existing verification methodologies and tools would be valuable for practical adoption and deployment.

REFERENCES

[1] H. Foster, “Applied Assertion-Based Verification: An Industry Perspective,” Foundations and Trends in Electronic Design Automatio, Volume 3, Issue 1, January 2009.

[2] J. Yen, “Enhancing Debug Efficiency up to 10X with Verdi Intelligent Debug Accelerator,” Synopsys Users Group (SNUG) Silicon Valley, March 2022.