Detoxify Your Schedule With A Low-Fat UVM Environment

A success story on using self-checking stimulus instead of a scoreboard to reduce development time.

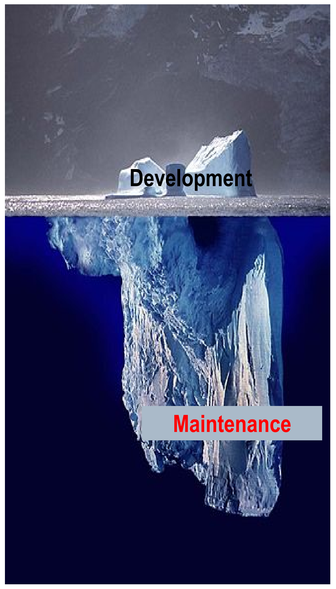
2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Executive Summary

•The decision to port a block-level environment to the system level is often made without considering the true cost and the real benefits
•Maintaining the transaction-level scoreboard is one of the biggest development costs when creating a portable block-level environment
•Significant time can be saved in maintaining verification infrastructure by using self-checking stimulus based on UVM sequences instead of a transaction-level scoreboard
•Concerns about non-portability of the block-level checkers to the system level can be addressed by focusing system-level verification goals on connectivity and integration of the verified parts
•Self-checking stimulus is implemented using UVM sequences to maximize reuse within the block
•Risks associated with using self-checking stimulus can usually be easily mitigated
•A DUT must meet certain criteria to benefit from self-checking stimulus, and not all DUT's are well-suited for this
•Considerable time was saved on our project by moving away from a transaction-level scoreboard to self-checking stimulus while still meeting our verification objectives

The True Cost of Portability

The true cost of making a unit environment portable to higher levels of integration is often underestimated.
Portability requires a transaction-level scoreboard. Maintaining a transaction-level scoreboard can consume considerable time in a project schedule.

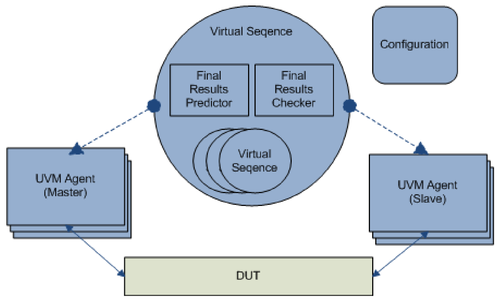


New and creative ways to arrive at the correct result may appear whenever the RTL is changed to fix bugs or fix timing, or when intensive random simulations are run to close coverage. These activities happen late in the project when schedule pressure is the highest, and cause the transaction-level scoreboard to break if it did not account for all possible ways to get the correct answer.

Let's Talk About Self-Checking Stimulus!

What is self-checking stimulus?

Self-checking stimulus brings together all aspects of the life of a DUT operation from transaction generation to checking of results under one encapsulation. This paper proposes self-checking stimulus using UVM sequences.



Why should I use self-checking stimulus?

Using self-checking stimulus instead of a transaction-level scoreboard reduces development time by avoiding the complexities of maintaining and porting the scoreboard.

How does self-checking stimulus avoid incurring the same maintenance costs?

A transaction-level scoreboard must predict all the possible ways to get to the correct results. Self-checking stimulus only checks the final result, and therefore avoids the churn from false fails due to multiple correct ways to arrive at the final result.

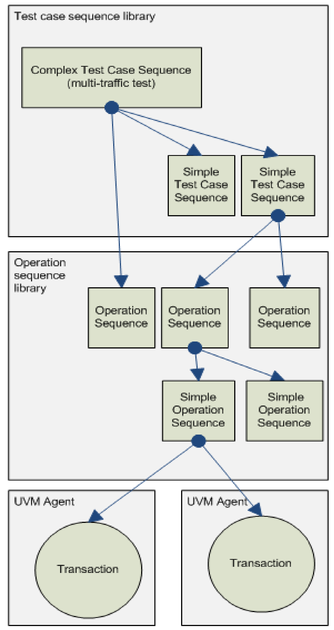
If a bench isn't portable, and uses self checking stimulus, as you propose, how do you deal with system issues that may not be exposed because there is no longer a reference model for that block at the system level?

It is not necessary to port a transaction-level scoreboard to verify the connectivity and integration of a verified block into a system. The bus protocol checkers/monitors used in block verification **are** portable to the system level when using self-checking stimulus, and already address one of the biggest risks of block integration: compatibility of the interfaces.

Will it take more effort to make each individual system level test self-checking?

No. At the system level, the checking needs to focus on connectivity and integration, so are not as detailed as the block. Furthermore, system level tests may be reused in post-silicon validation. In that case, the reusable tests would have to be self-checking regardless.

Stimulus Implementation



•Identify the basic operations and implement using UVM sequences.

•Build checking capabilities in the basic operation sequences and extend/reuse upwards.

•Test cases are UVM sequences that mix simple and/or complex operations.

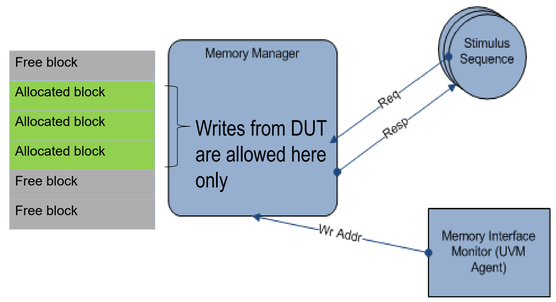
Mitigating Risks of Self-Checking Stimulus

Self-checking stimulus comes with inherent risks. Quite often though, these risks can be easily mitigated. As with any checking methodology, it is important to carefully consider the risks and risk mitigation plan in the context of your DUT and system before committing.

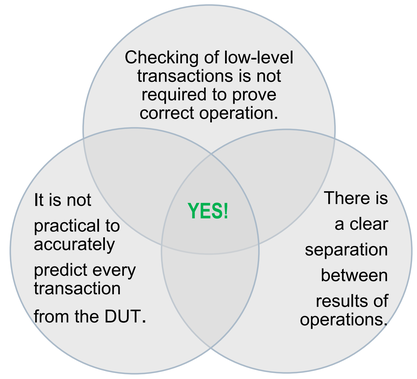
Example: Rogue transactions.

The Problem: Results are correct, but the DUT issued an extra 'write' transaction to an incorrect memory address , thereby corrupting the memory contents outside the range of addresses containing results. How can self-checking stimulus detect this?

The Solution: Stimulus should utilize a memory manager to maintain a list of free/allocated address blocks and couple this with a memory interface monitor to flag errors when the RTL accesses a free space.



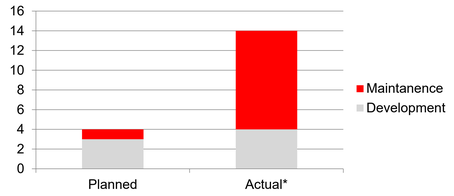
Is Self-Checking Stimulus Right For Me?



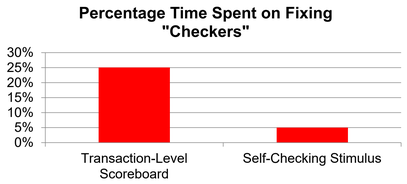
If these characteristics apply to your DUT, then consider the self-checking stimulus as your verification strategy!

Results

The chart below compares the planned and actual weeks for developing and maintaining the transaction-level scoreboard. *Note that the maintenance of the scoreboard was not fully complete when it was replaced with self-checking stimulus after 14 weeks of effort.



After moving to self-checking stimulus, the time spent to maintain the checking capability of the block environment was reduced by about 80%. This time includes debugging of failures due to the false firings of checkers.



Contact information

Nihar Shah
Hardware Advanced Development
Oracle Labs

5300 Riata Park Court
Austin, TX 78727

Email:
nihar.shah@oracle.com