Case Study

Targeting Complex Power-mode Verification using Breker TrekSoC™ and Portable Stimulus at Broadcom, Inc.

Abstract

The following white paper discusses a verification project undertaken by engineers at Broadcom where a new portable stimulus methodology was employed to verify the power modes of a device, a process that had proven time-consuming and error-prone. This practical application of the Breker Portable Stimulus methodology significantly improved verification quality while dramatically improving the time required to create and execute the tests.

Breker Verification Systems, Inc.
www.brekersystems.com

Introduction

Broadcom was looking for a new verification solution to tackle the particularly complex task of verifying power modes in a new device. This operation leveraged complex synchronization mechanisms, and its verification required both embedded software tests working alongside hardware transactions and leveraged components from a UVM environment. Previous test cases had required many weeks to design, and a more effective solution was required.

The Chip Architecture

Information regarding the device itself has been removed for confidentiality reasons. It was a complex design, driven by an ARM M7 processor with a number of high-performance peripherals made up of IP blocks created both externally and internally to Broadcom. The IP blocks represented a chain of producer consumer units, with each requiring control of their power modes, see Figure 1. They had been thoroughly tested prior to integration on the SoC platform.

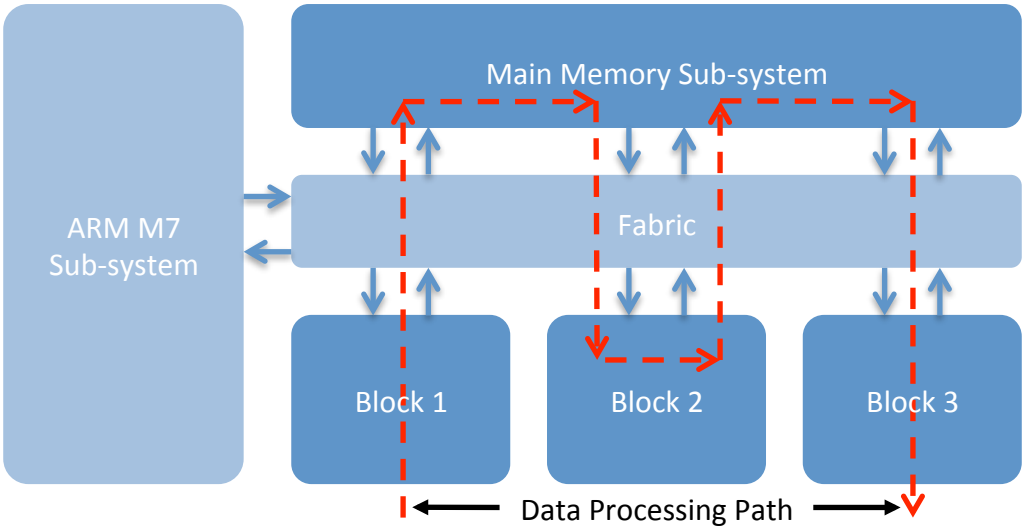

Figure 1: Abstract Design Representation of the Device

Data in the device had to be passed to the different peripherals in a specific order, from IP1 to IP2 to IP3 to exit from the chip. The data processing of IP1 had to be complete before IP2 received the data and so on. This required a degree of synchronization as data was written to and read from memory by the IP blocks across the device. This data synchronization had to be maintained during the power cycling of the device components.

Minimizing power consumption in this design was key and this device has a number of power domains that were cycled on demand, with the heavy involvement of the processor. A state machine was used to depict the start up and power down sequence of the power domains, see Figure 2, and it was this process that had to operate correctly under a variety of conditions and operational modes.

The Power Domain Verification Environment

The verification of this device was a complex process. Testing for correct power state operation given the myriad of potential operational conditions and timing characteristics required a well thought out verification plan detailing many complex test sequences. All possible transitions of the power states for all power domains had to be tested. Given that any power event could happen at any time, random testing was used to look for corner case conditions that could trip up power domain activation and reset. There were many potential scenarios that might not be completely obvious to the verification engineers, and it was important that these were exercised.

An existing test-bench for the design as a whole had been created which consisted of UVM sequences providing transactions to the peripheral I/O ports, coupled with C-tests running on the processor. An existing scoreboard for peripheral operation had been created, also using the UVM standard. Only limited coverage analysis was required to be performed, with the objective of the verification to run corner case scenarios of different types for bug hunting purposes. The entire design and test-bench had to be run in both the Synopsys VCS and Cadence Incisive simulation environments. This required careful coding to ensure no simulation mismatches would occur. The Synopsys low power analysis capability within VCS was also employed.

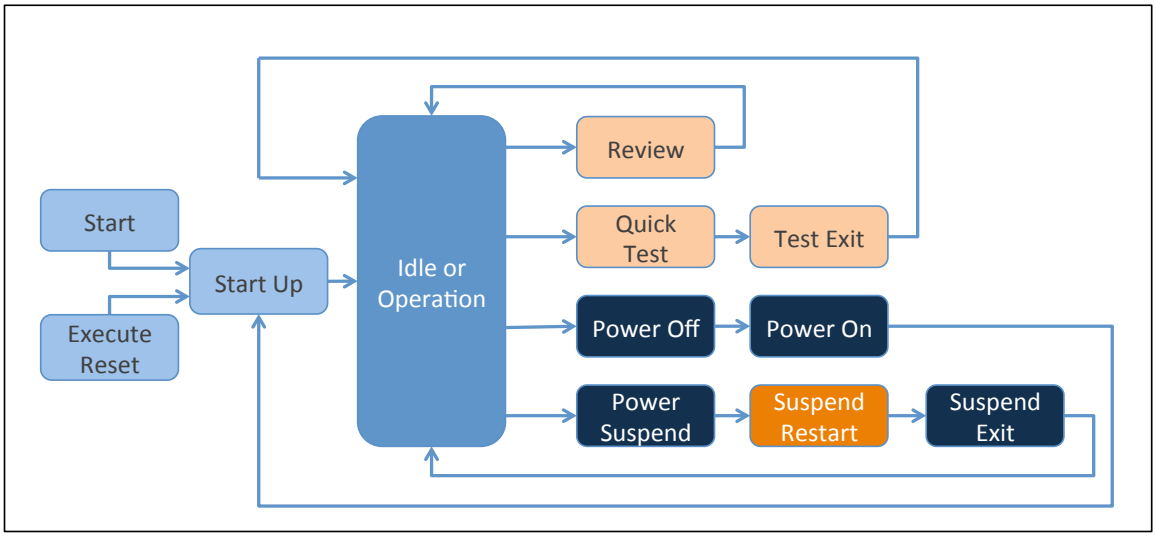

Figure 2: Power Cycle State Machine

Major problems with this environment included the synchronization of the UVM sequences with each other and with the processor C-tests. Randomization had to be conducted at a higher level than typical UVM constrained random test generation to drive an effective power state analysis. Prior to the use of Breker's TrekSoC, this had been coded in an ad hoc manner into the environment. When the test-bench found an issue, it was also very difficult to understand exactly what test component at what

time had tripped the problem, making debug slow and error prone. An easier and more rigorous method to perform this power event verification was required.

Layering Breker Portable Stimulus Test Cases into the Environment

The Breker TrekSoC tool was employed to produce a more effective test-bench and environment for the testing of the power modes in this device, see Figure 3. The approach was to create a graph-based scenario model of the entire test-bench and then include within this a subsection that specifically handled the power transition modes. As the scoreboard and UVM test sequences were already created, these were reused within the Breker environment, which allowed the engineer to focus her efforts on verifying the power transition modes themselves, operating with the existing functional tests.

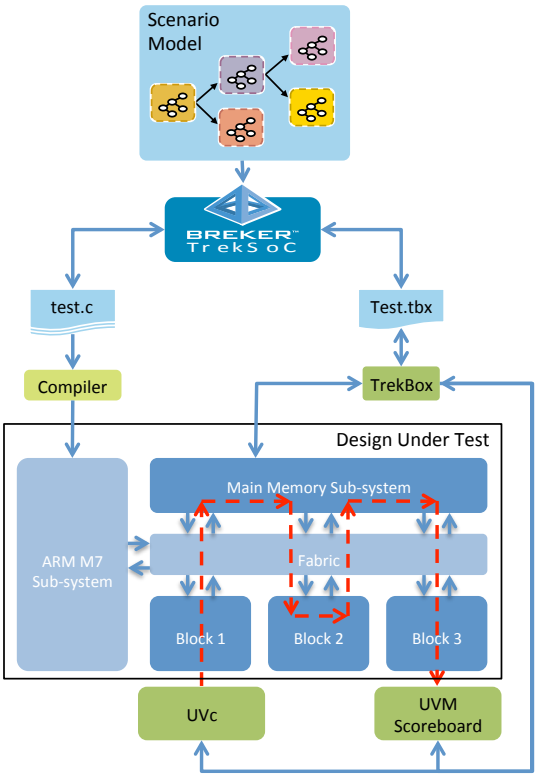

Figure 3: New Verification Environment

A complete graph-based scenario model for the power mode operation was written in C++ with just a few days effort. The test-bench was focused on the operational scenario under test at a verification specification abstraction level rather than the more traditional lower level required in a UVM test-bench. The Breker Design Analysis Environment provides a visual depiction of this scenario, as shown in Figure 4.

This abstract level of design made it easier to consider the states requiring verification without having to be concerned with individual test streams of transactions for the I/O ports or C-tests for the processor. This provided a far more complete, exhaustive and rigorous test model from which to derive the tests than would otherwise have been possible.

Furthermore, it was possible to configure the entire model for specific runs using high-level path constraints. These constraints could be applied to the graph at the system level in a top down fashion, allowing for detailed test configurations to be executed that focused on specific areas of the design or scenario operation, without changing the overall model. The application of high-level path constraints was far more efficient than deriving lower level constraints.

TrekSoC was then used to synthesize tests for the various IP ports within the device, along with the processor C-tests. Many thousands of tests were generated that

exercised a broad range of corner cases in the device. In particular a large amount of traffic between the IP blocks was generated coincident with power domains in the device being cycled. This wrung out the synchronization of events between the IP blocks, and multiple situations were uncovered where the power domain operation caused the test-bench to hang, indicating a problem.

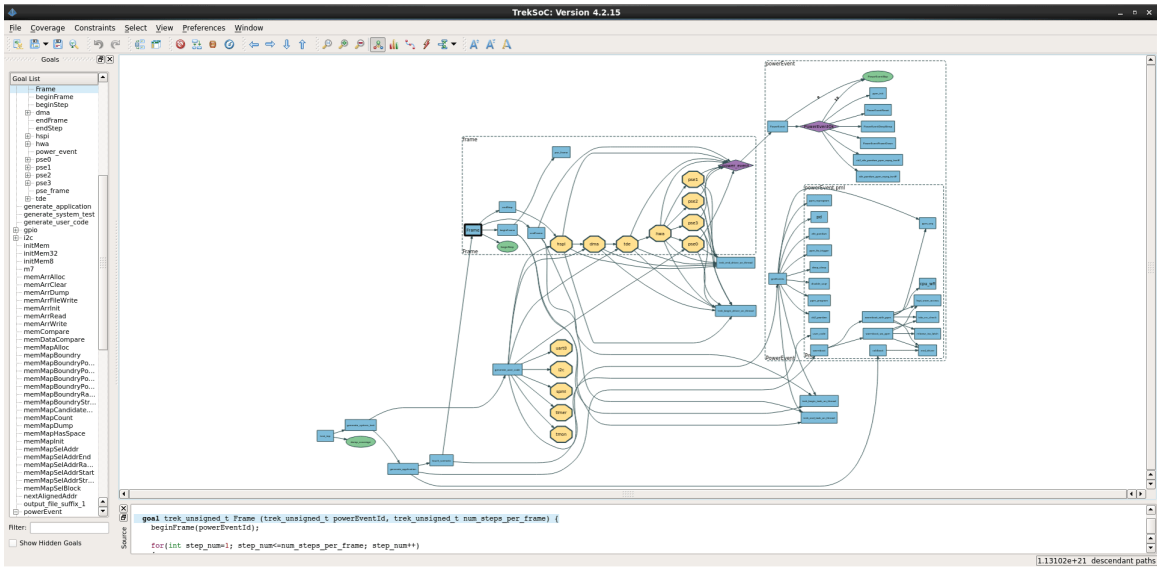

Figure 4: The Power and Functional Verification Scenario Model

The Basic Test Scenario

The test progression took the following form:

Firstly the device was reset and went through a start up process, followed by some basic system checks by the processor.

The device would then enter a warm start cycle, and during this phase memory content would be lost. The test process, which made use of C tests stored in memory, would have to allow for this, see below.

Once the warm start process is complete, the device functionality would commence, where complex chains of producer/consumer scenarios were cycled in a specific order to maintain data consistency and synchronization.

During this process at various intervals, a power domain reset would be executed, interrupting the producer / consumer processing in a random fashion. A warm start cycle would be executed and the device would then be expected to resume its consumer producer cycling operation as if nothing had happened.

As noted, the tests generated required a tight synchronization between the C test run on the M7 and the UVM test-bench. This is due to the fact that during some power mode transitions, the data in memory and, therefore, the progression of the C test

would be erased. To deal with this, landmarks were stored in the test-bench and retrieved in C when needed during the wake up process. Using the graph nomenclature, this was relatively easy to handle.

The Breker test map display, see Figure 5, was used to examine the various multi-threaded tests and their synchronization. The test map showed the various APIs in the system covered during simulation, thereby clearly revealing exactly the test being executed on what port when the problem occurred, and the last line of code executed. Using this display, it was easy to spot the point at which the testing stopped and uncovered an area. As the Breker tool is integrated with the Synopsys Verdi debug environment, Verdi was then invoked to examine the design itself and pinpoint the issue.

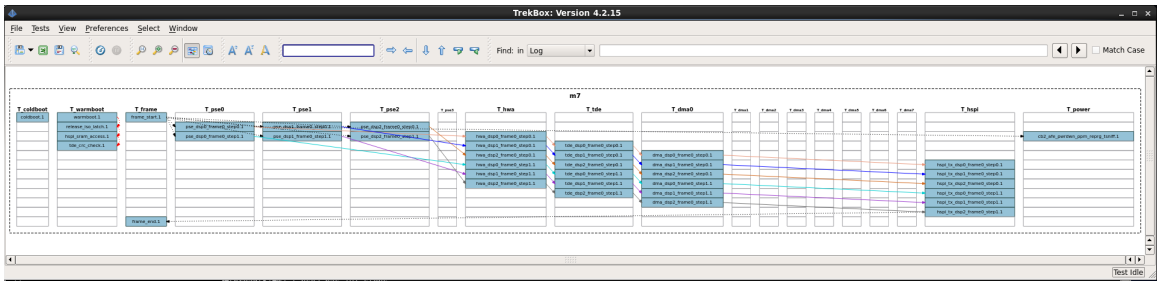

Figure 5: The TrekSoC Test Map Display Showing the Multi-threaded Tests and Synchronization Points

A typical issue might be some event not taking place when it should, for example an interrupt not occurring when expected. The TrekSoC display showed all the details of this, aligned with the tests right across the entire system, allowing Verdi to be brought up exactly on the right signal to figure out if the problem was in the test-bench or the design itself. This saved many hours of debug effort.

Resulting Verification Improvements

The various IP blocks in the system had already been thoroughly verified so, as expected, no individual block bugs were discovered. However a number of system level bugs were found mostly involved with the synchronization of the IP blocks, revealed during power cycling. These bugs were not discovered using the old test-bench and would have been hard to find without the extensive number of stress tests applied by the Breker methodology.

There were a number of advantages to using TrekSoC on this project:

1. It was much easier to code the graph than to manually code test-benches, as the test scenarios could be considered without the test clutter.
2. The final result was a more rigorous and complete verification exercise as the possibility that a key test case was missed was greatly reduced, due to the 1000s of combinations of test transactions that were generated, and the coverage check over the entire scenario graph.

3. In addition to the increased verification quality, a significant amount of time was also saved on this project, a 5X improvement in schedule.
4. The coverage solution contained within the system made it clear what had been tested and that important scenarios had been covered.
5. Debugging failing tests was dramatically simplified given the capability to visualize what is going on with each test and zero in on the root cause.

Breker made the entire process and learning curve very straightforward. Breker's application engineer was able to train the Broadcom engineer within a day or two and helped create the first scenario graph. From that point the Broadcom engineer created the full set of graphs within a week or two, quickly completing a project that could otherwise have required months of work.

Final Summary Numbers

The final key statistics for the original project and the improvement are shown in the table in Figure 6, and the graph in Figure 7.

Metric	Manual	Synthesis	Improvement
Test Authoring Time	2.5 months	2 weeks	5X
Unique, High-Impact Tests Generated	<500	>10,000	20X
Coverage Gap (100% - Coverage)	11%	3%	4X

Figure 6: Final Project Statistics

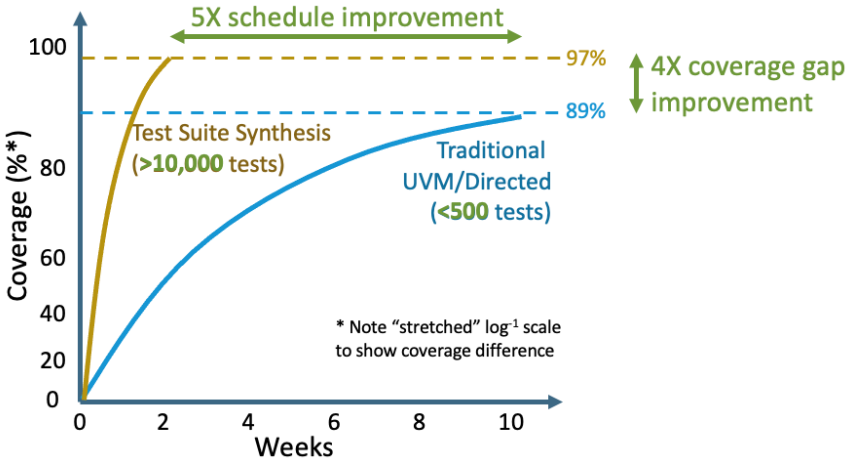

Figure 7: Graph of Comparative Results

These numbers are generally representative of the results achieved with this solution.