Building Transaction-Based Acceleration Regression Environment using Plan-Driven Verification Approach

Shabtay Matalon, Leonard Drucker, Maya Bar, Michael Stellfox

Cadence Design Systems, Inc
San Jose, CA 95134
shabtay@cadence.com, leonard@cadence.com, maya@cadence.com, stellfox@cadence.com

Abstract – ***The paper presents flows and methodologies for using plan-driven verification testbenches with accelerated verification engines via transaction oriented interfaces. It describes how to reuse stimulus generation, coverage analysis, checking and error handling in reactive and regression environments while optimizing the testbench architecture and modeling style for high performance verification.***

I. Introduction

Transaction-Based Acceleration is a technique for taking an existing advanced simulation-based verification environment (usually incorporating a coverage-driven approach) and incorporating a hardware accelerator to provide the speed to perform system level verification and hardware/software co-verification. Traditionally, verification engineers performed the work of creating system level tests that execute on a hardware accelerator by writing directed tests (i.e. each test exercises only one scenario). Furthermore, these engineers typically created the entire regression environment using adhoc approaches requiring Make files, perl scripts, shell scripts, etc… Testbench automation tools like Incisive Specman Elite have been proven for many years to enable significant efficiency and quality improvements by enabling constrained-random, coverage-driven verification for block- and cluster-level verification. However, applying these same approaches to system-level verification has not been practical given the relatively slow speed of software simulators when simulating large system designs. With the advent of Transaction-Based Acceleration techniques, verification engineers can now take advantage of reusing a Specman simulation-based, coverage-driven verification environment created to verify block to cluster to system levels, and run at significantly higher speeds on a hardware accelerator. While this will significantly improve efficiency (since the verification environment can be reused) and can improve quality (since the generated tests exercise more corner case scenarios and capture coverage metrics), it does not guarantee that the system level verification coverage objectives will be met. These objectives could only be met by creating an efficient transaction-based acceleration and simulation regression environment defined using a Plan-Driven verification approach.

II. What Does Plan-Driven Verification Really Mean?

What is Plan-Driven verification? Why would you want to implement a Transaction-Based Acceleration regression environment using Plan-Driven verification approach?

Plan-Driven verification begins with the creation of a feature-based verification plan which captures "what" features need to be verified, independent of "how" the tests will be created. Once all of the features have been captured, they are linked to coverage metrics which observe when a specific feature is executed in the verification environment. Using a constrained-random verification environment enables automatically generating many permutations of test stimulus targeted at filling the coverage. As the coverage goals are achieved, this information is annotated back to the feature-based verification plan so that analysis can be performed to measure exactly what effort remains to verify all features defined in the plan.

The following example is taken from [1]:

Verification Plan Feature: The DMA engine in an MP3 SoC is responsible for moving song data from the flash memory to the SRAMs accessed by the DSP core.

Metrics (which can be observed with coverage): DMA configuration; Number of DMA transfers; DSP operation; Source/Destination addresses; etc…

Stimulus Scenarios: Implement the use case of writing song data from flash memory to SDRAM and have the DSP process the data. Leverage constrained-random verification environment to automatically create interesting stimulus scenarios:

- Configure the DMA engine to one of the 20 modes
- Select different types of song data

- Select different source and destination addresses
- Select different operation modes of the DSP
- Etc…

Constrained-random, coverage-driven verification is very thorough, because it tests the design in corner case scenarios which engineers might not typically think of (like performing a DMA operation with an illegal DMA configuration), while using coverage metrics to dynamically observe that all combinations have been executed. Plan-driven verification extends the coverage-driven approach with a thorough up front planning process and the capability to define and link executable coverage metrics to each feature of the plan to ensure that each feature is tested. Additional information that needs to be captured includes how each coverage metric will be measured (functional coverage, assertion checkers, etc…) which execution engine will be used (i.e. simulator, accelerator, formal analysis tool, etc.) and which operation modes or test flows will be applied. Plan-Driven verification provides a highly automated and predictable process for reaching verification closure as described in [2].

Traditional System-Level Methodology

A traditional System Verification approach would typically involve hand crafting a series of directed tests that implement a specific use-case. Reusing the above example, it might look something like: Use the default DMA engine configuration, move the song "Let it Be" from flash memory at address 0x0000 to SRAM at address 0xff00, start DSP operation in default mode (which might be a very important, frequently used, use-case that needs high performance to test in a reasonable time).

Plan-Driven System-Level Methodology

In a Plan-driven verification methodology, the engineering team creates a verification plan. The plan breaks down the design into features. The features have metrics associated with them, and the means by which each metric will be measured. This allows reports to be generated that detail which features have been exercised as well as which features have not been exercised. The same approach can be applied for block-level, cluster-level, and also system-level. The only differences would be which features are verified at each level, and which simulation engine to use at each level, where typically a hardware accelerator using transaction-based acceleration methodology is most applicable at the system-level. The benefits of this for Systems Verification engineers are many:

- Executable plans created by the verification team for block- and cluster-level verification can be re-used and extended for the system level verification
- The constrained-random, coverage-driven verification environment can be reused across software simulation and hardware acceleration
- Reports are easily generated showing the current status of the verification effort
- Re-running failed simulations is easy and can actually be programmed to occur
- Coverage metrics can be used to get a realistic picture of the verification status against its goals

III. Implementation Strategy

To achieve all possible benefits of a Transaction-Based acceleration regression environment using an executable verification plan, the environment needs to be architected to:

a. Create an executable verification plan that combines simulation and acceleration resources in a regression environment
b. Maximize performance of the verification environment when driving the hardware accelerator and the entire regression environment
c. Maximize reuse of the simulation-based verification environment

a. Bringing Acceleration into the verification plan

Executable verification plans, as described in the introduction section, can be extended to include features that will be verified on the hardware accelerator. An existing executable verification plan would contain features and associate coverage metrics with those features (to prove that the features were exercised). To enhance the plan for TBA acceleration, additional metrics enabled by TBA acceleration and derived from system level requirements can be added to those features. Additional metrics might be: system level assertions; additional coverage items; additional directed tests targeted at specific functionality; etc.

Additional attributes would be added to the verification plan that define which features will be verified on the acceleration regression engine and which of the test flows and acceleration operation modes will be used for different sections of the verification plan. These attributes include specifying that longer test scenarios should be run on the acceleration platform, and identifying test scenario attributes which determine how to extract the maximum performance by using buffering or pre-generation, and post analysis flows.

b. Maximizing overall performance

Maximizing overall performance of the regression environment entails maximizing the performance of the testbench (running on the host computer) by minimizing the time spent in the testbench, optimizing the utilization of the hardware accelerator, and reducing the number of times that both the host (called SW side) and the

accelerator (called HW side) need to synchronize passing data back and forth.

The architecture should incorporate the following principles:

1. The most active part of the testbench (BFM/monitors) is running on the accelerator at its peak speed.
2. The BFMs and monitors encapsulate only the interface protocol specific knowledge and thus can be reused from project to project
3. The BFMs and monitors are the only testbench components requiring clocks. When running on the accelerator, all clocks can be generated inside the HW side partition avoiding synchronization with the SW side on every clock edge.
4. BFMs and monitors can provide or gather "transaction data" over multiple clock cycles. During these periods the HW side can run w/o interruption.
5. Interaction between the HW side and SW side is fully asynchronous. It happens only when the HW side requests a new transaction or produces a new transaction.
6. Buffering mode can be used by non-reactive models and controlled by the user for each communication channel. It supports batching multiple transactions into the buffer reducing the number of HW/SW side synchronizations.
7. HW/SW communication does not take place during idle periods. The more idle periods that occur on certain interfaces, the more performance improves.
8. The testbench residing on the SW side is abstracted to higher level data items or user transaction-level API, and thus runs significantly faster with the BFM and the Monitor relegated to the HW side.
9. Significant performance can be obtained if the stimulus could be pre-generated and DUT response could be provided for post-processing checking.

c. Maximizing Simulation testbench reuse

The simulation verification environment (VE) development requires significant development effort, especially when taking into account reuse considerations. This development effort is most efficient by developing the environment with the e verification language and the Incisive Specman Elite tool in combination with the Incisive Plan-to-Closure Universal Reuse Methodology (URM). It is also possible to adopt configurable "off-the-shelf" Universal Verification Components (UVCs) for standard protocols like AMBA or PCI-Express, for example, rather than develop them from scratch. Specman is highly optimized for applying plan-driven verification methodology. Extending the Specman simulation VE to an accelerated environment provides the advantage of a single executable verification plan as well as reusing most of the constrained-random, coverage-driven simulation VE. The unique Aspect Oriented and extensibility features of the *e* language greatly simplify the effort to target and configure the same VE to be reused for both software simulation and hardware acceleration at user's control.

The following principles could be applied to maximize

1. Agent architecture

   Make sure that interface from the sequence driver to the BFM is invariant to HW or SW BFM. Same applies to the data collected from the BFM.

   Remove all clocks from the UVC working in acceleration mode.

2. Data items architecture

   Create simplified data structures based on the required level of verification. Some fields (mainly control fields) are not required in acceleration mode.

   Add configuration control for switching between data items types.

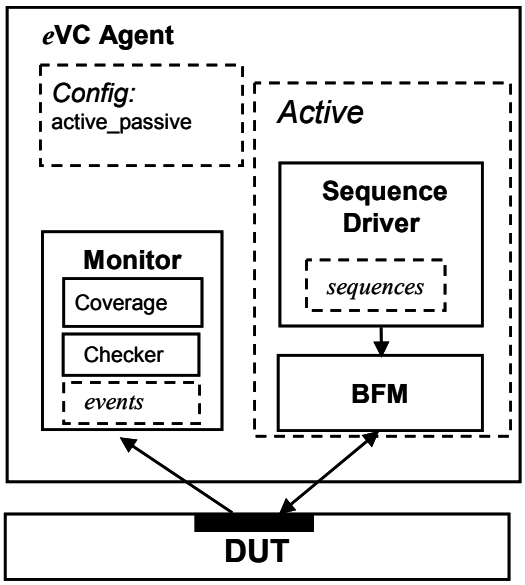
3. Test writing style

   Reduce time waits and context switching to the extent possible.

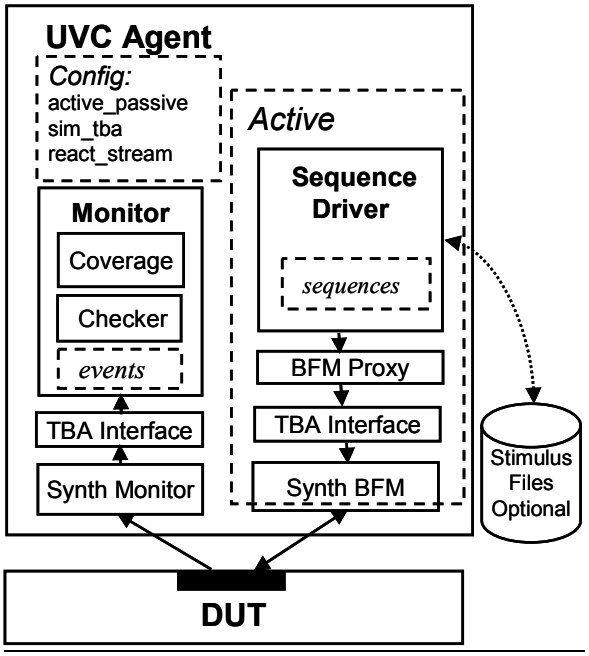
   Use reactive flows when the simplified data structures can provide sufficient performance.

   Use streaming flow in cases when the simplification of the data structures is limited or complicated to implement, In this case most of the data items are generated in parallel, by a number of concurrent Specman copies, with reduced reactivity level

The following figure presents a simplified view of simulation-based e Verification Component (eVC) architecture.



The following figure presents a simplified view of the modifications required to change an existing simulation-based eVC to a Universal Verification Component (UVC) which can also support TBA acceleration.



IV. Case Study

In this case study we will take a look at how our Transaction-Based Acceleration Regression methodology can address the issues outlined in the previous sections.

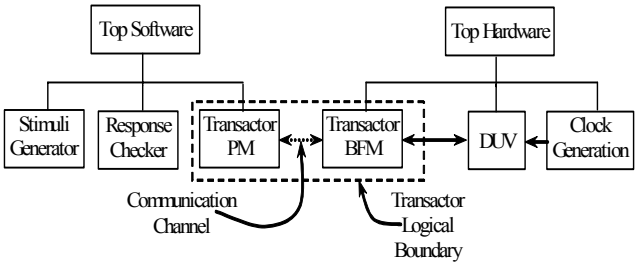
DES case study – We decided to use the Cadence AMBA High-performance Bus (AHB) e Verification Component (eVC) that was integrated with a TBA compliant transactor in conjunction with an open source decryption/encryption Data Encryption Standard (DES) blocks that served as the DUT. Multiple DES block instances were "daisy chained" to establish a DUT with configurable capacity. On the SW side, the eVC could produce (at user's testbench controls) transactions of variable length (transaction length is defined by the number of clocks used to assert the transaction). The eVC applied all transactions back-to-back to DUT meaning, that a new transaction was provided immediately when the pervious transaction was consumed.

The DES case study allowed us to run the tests we defined in our verification plan, each representing a typical test scenario combined with its corresponding acceleration data item. These tests allowed us to validate and refine our Transaction-Based Acceleration Regression methodology and measure the acceleration performance obtained by varying certain attributes in our verification environment.

a. Reuse considerations

SW/HW testbench partitioning -- Partitioning is done to maximize performance and reuse. It involves creating two verification sub-components which are separated from each other and communicate over uni-directional communication channels. The HW side partition is driven by clocks while the SW side is transaction-based and less clock dependent. The SW side could still scarcely use timed constructs such as waiting on time to allow the SW side and HW side to synchronize and exchange transactions.

The following figure presents a simplified view of SW/HW testbench partitioning.



Defining transactor architecture -- We used the Cadence Incisive Transaction Based Acceleration (TBA) Methodology which provides the basic architecture and guidelines as described in [3] and [4]. TBA presents a common transaction-based interface for simulation and acceleration that is fully congruent between the two modes – meaning that the environment will simulate exactly the same across both engines. It supports Verilog and VHDL on the HW side and SystemC/C++ or e testbenches on the SW side. It handles variable-length transactions of any practical size set by the transaction producer at run time.

It supports buffering, yet allows modeling of transactors such that buffering can be added at the end-user's control only when running in acceleration mode for increased performance. It supports automated accelerator/host synchronization and generates the synchronization signals by analyzing the transactors and operation mode settings. It supports timed testbenches and timed simulation control. It offers a configuration interface for controlling batching/reactive operation mode setting on a *global* or *per-channel* basis. And it enables a robust debug flow including support for transaction recording and analysis.

Communication and resource allocation -- TBA communication resources contain uni-directional buffered transaction channels that can move transaction data from the HW side to the SW side or vice-versa and non buffered state ports for status and configuration control. The transactor modeler can choose the number and types of communication channels used in each transactor.

To achieve maximum speed for a TBA environment on Cadence Palladium and Xtreme emulators, the transaction channels can be configured to support two different operation modes:

> ***Reactive mode*** - in which the synchronization between the HW and the SW is done on each transaction boundary.
>
> ***Batching mode*** - in which a batch of transactions can be driven into the buffers and the synchronization occurs when the channel buffer is full or empty.

Reactive mode insures that no latency is introduced by the buffered communication channel. When no reactivity is required, batching mode can be turned on allowing a batch of transactions to reside at the buffer at the expense of introducing latency. The latter will reduce the number of HW/SW side interactions for non-reactive testbenches resulting in increased performance.

TBA state ports can be used as status and configuration channels that allow the BFM to be configured at synchronization time or the testbench to obtain additional status data (for example when errors are encountered) for additional error logging and debugging.

Upon designing the transactor the developer assigns as many communication resources of each type as needed for transaction data transfer, configuration and status.

b. Performance considerations

Test Flows -- When driving an accelerated DUT, the performance requirements of the software side are very demanding. To achieve significant acceleration speedup (defined as 50X and above relative to simulation), the SW testbench execution time should not exceed 2% of the overall simulation time. The testbench performance should be analyzed (in simulation), in order to identify bottlenecks and to predict performance impact of data structure simplification.

To accomplish this goal, the following test flows were implemented:

> ***Reactive test flow*** – Used for tests in which sufficient performance can be provided via simplification of transaction-level data structures. Stimulus generation is simplified and configuration control is added reducing the testbench execution time on the host to achieve the performance goal. The constrained-random generated stimulus is simplified for system level by turning off some of the sophisticated corner cases that have been tested in simulation at the block and cluster levels.
>
> ***Streaming test flow*** – The performance is achieved using stimulus which is pre-generated concurrently by a number of Specman stand alone sessions. The pre-generated stimulus is retrieved by a single Specman session that drives the accelerator, and obtains the DUT response. The checking and coverage analysis of the results can be done in separate concurrent Specman sessions as well. In this case the level of reactivity is highly reduced.
>
> ***Hybrid test flow*** – Combines the above *Reactive* and *Streaming* test flows. Given that stimulus generation on some interfaces would require reactivity while on other interfaces it may not, and given that testbench execution speed on each interface may vary, pre-generated stimulus and reactive stimulus can be combined using the hybrid flow. The earlier could be restored while the latter can be combined on the fly. Similarly some level of results checking (such as simple error checking) could be done on the fly while more extensive results checking (such as coverage analysis) could use post-run analysis latter.

HW acceleration and SW simulation resource balancing -- The full verification plan leverages a common verification environment for both simulation-based and hardware accelerated verification, where sophisticated block level stimulus is generated and typically executed on a software simulator, while longer, more real-world stimulus scenarios are generated and executed on to the hardware accelerator. When using the *Streaming* test flow or the *Hybrid* test flow, the Specman simulator will be used for pre-generation of stimulus and post-run analysis of system level tests running on the accelerator.

Optimized resource balancing is obtained when the simulation resources and acceleration resources are balanced to complete the regression at the minimal possible time utilizing the available simulation licenses and the available acceleration resources. Of particular importance is avoiding idle time on the accelerator by assigning sufficient simulation licenses and workstation resources to constantly produce stimulus and check DUT responses for multiple tests while the accelerator is running on other tests.

This methodology could be further refined to randomly pre-generate individual stimulus sequences, and then during run-time on the accelerator, a single simulation license assembles the individual stimulus sequences into one long test. By optimizing the length of each of the stimulus sequences and by setting the accelerator operation into *Batching mode*, long pre-generated complex test regressions are executed very quickly on the accelerator.

The Incisive Enterprise Manager combined with dispatching tools such as LSF from Platform Computing allows optimizing HW acceleration and SW simulation resource balancing by spawning the tests to both targets types and managing the regression run process. Enterprise Manager is a complete verification management solution which not only includes regression management but also coverage and failure analysis and reporting capabilities.

c. Creating an Executable Plan

An executable verification plan is either created as part of the overall verification plan, including the earlier simulation-based effort, or a separate plan is created for the TBA verification effort. The plan is typically captured in a word processor such as Microsoft Word or the Adobe FrameMaker in accordance with the Incisive Plan to Closure Planning and Management Methodology. Once the plan is captured, it can be exported to an executable form by the Incisive Enterprise Manager tool where it is then used to manage the overall verification process.

While a verification plan can be structured in many different ways, our customer interactions have shown that there are some basic sections that are normally captured. These sections are:

Functional Requirements-- This section contains information related to the functional features that can be verified as a black box (i.e. using the specifications to determine which features need to be verified. Typically, no internal information which relates to the specific design implementation is used). This section is further decomposed into two sections:

> ***Functional Interfaces*** – This area defines verification information relating to external and internal interfaces that the device has, such as Ethernet, AMB bus, USB, etc.
>
> ***Functional Core*** – This area defines verification information relating to core features of the design, such as the example given in the opening section "The DMA engine is responsible for moving song data …". This is a core feature of the DMA block.

Design Requirements -- This section contains information related to design implementation details, such as pipelining, fifos, etc.

Verification Views -- This section contains information on different views of the verification content. Since these devices are very complex, there usually is different verification information that is being extracted to review in each view. For example, you might have milestones defined. In one milestone you define that the DMA block and Ethernet block should be 50% verified by the 4th week of the project. This becomes a verification view. Another view might be a register view, in which all the registers in the design are referenced by the verification view and checked.

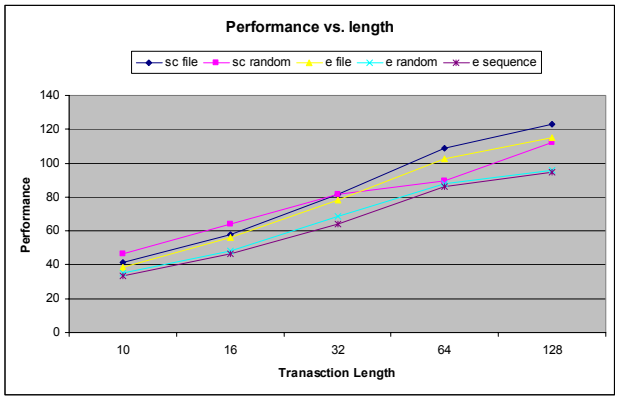
Verification Environment - This section contains information on the creation of the verification environment. This usually refers to the creation of: verification checkers, functional coverage models, verification components (which stimulate and respond to the design), etc.

To quickly create a verification plan, it is possible to start with a verification template, which comes standard with the Incisive Enterprise Manager. Fill in the Functional Requirements section with the functional information of the features that you will need to verify and then follow the Incisive Plan to Closure Planning and Management Methodology to create the corresponding coverage model and the rest of the process.

d. Study Performance Results

We initially used a testbench driving pre-generated stimulus from a file and then used relative simple stimulus generation and response checking conducted on the fly using the Reactive test flow. For these test cases, we saw performance running on the Cadence Palladium family of emulators and the Xtreme family of emulators as high as 120x over the Cadence Incisive Unified Simulator (IUS) simulation for multiple DES design blocks configured to create a 2M gates DUT.

The following chart illustrates the performance results obtained for these tests when either written in SystemC or Specman/e on the 2M gates DES DUT configuration. TBA operation mode was set to *Reactive mode* and buffer depth was assigned to 128, meaning a single transaction whose length is also 128 can fit exactly into the buffer.



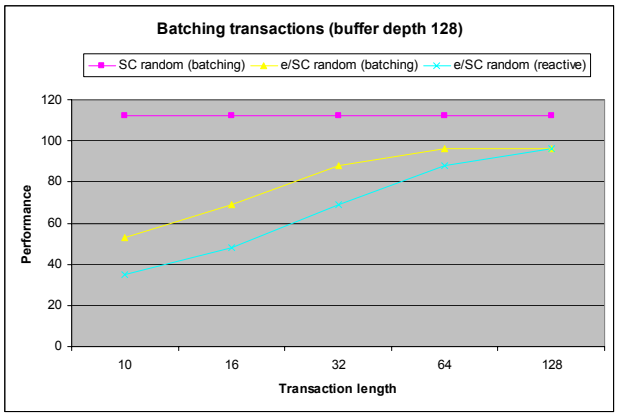
As can be observed in the chart, the performance obtained for SystemC and e was comparable for a similar type of

tests. As we slightly increased the testbench complexity from file-based to a simple random-based test to a simple sequence-based test, performance slightly decreased. However, the performance was mostly dominated by the length of the transactions applied via the TBA transaction interface as opposed to the type of the test.

As can be observed in the chart, the performance increased by 3x (from about 40x to about 120x) when the transaction length increased from 10 to 128 reflecting about 13x reduction in the number of HW/SW side synchronizations. This level of performance was obtained by applying back to back transactions (meaning no idle time between transactions) while TBA was running in *Reactive mode* (and w/o any buffering) on the Cadence Palladium and Xtreme emulators.

Later we decided to compare the impact of TBA *Batching* operating mode at various transaction lengths. First we ran only the SystemC random test case in *Batching mode* at various transaction lengths. Then we ran a mixed Specman/e and SystemC configuration. While the HW and SW side was configured to run in TBA *Batching mode,* Specman/e and SystemC maintained *reactive* communication semantics (meaning exchanging context on transaction boundary). Last we configured TBA to run in *Reactive mode* while still maintaining the same reactive communication between SystemC and e.

The following chart illustrates the results. *Batching mode* for the SystemC test case provided steady performance for all transactions lengths as the number of synchronizations was kept constant in this mode. Given that the testbench automatically throttled the messages into the TBA channel buffer (given its *Batching mode* setting), the number of HW/SW side synchronizations was only determined by the buffer depth set to 128.



For the mixed e/SystemC configuration running in *Batching mode*, performance was impacted by the reactive context switching between SystemC and e on transaction boundaries. However TBA *Batching mode* kept the HW/SW side synchronization constant providing better performance over the last test case running in TBA *Reactive mode* using purely reactive semantics across all interface boundaries.

One observation that can be made is that using *Batching mode* increased performance only when multiple transactions could fill TBA channel buffer. The performance for the mixed SC/e test case matched in *Batching* and *Reactive* modes for transaction length of 128 as only a single 128 length transaction resided in the buffer for both test cases.

The TBA methodology was also applied to a complex testbench driving the DES design via the AHB eVC. The eVC Agent was modified to conform to the UVC Agent architecture by making some modeling changes that simplified generation complexity and by introducing configurability that traded-off some of the UVC functionality in acceleration mode in lieu of performance. The sequence driver in the agent was enhanced to perform stimulus save and restore to support the *streaming flow* use model.

For some of the test cases that required reactivity, we used the *reactive flow* subject to configuration control setting that simplified the complexity of the test in acceleration mode. In this case we obtained performance in the range of 60X over simulation.

For the complex AHB test cases that did not require reactivity we used the *streaming flow* which provided run time performance equivalent to a file-base pre-generated stimulus.

Using the *streaming flow* with good load balancing of Specman on seven workstations and a single Cadence emulator, stimulus generation with segmentation each of the of test sequences into individual files (to allow concurrent run of all resources in the environment) and result checking, we saw overall improvement of 3X (using on 6 -7 Specman licenses) over what it would take to run the same test conducting generation on the fly using the *reactive flow*.

The TBA regression methodology promotes pre-compilation of the environment before the tests are applied. Unless changes were introduced to the design or to the UVCs, multiple tests can be applied w/o requiring recompilation of the DUT and or the UVCs.

V. Conclusions

Increased productivity and predictability

A common verification environment has been used for both software simulation and hardware acceleration. Significant performance improvement, in particular for system level verification, was obtained with a good, configurable testbench architecture and management of the TBA regression environment that maximized the testbench performance driving the hardware accelerator while maximizing simulation testbench reuse.

A single executable verification plan addressed both simulation and acceleration targets, using the results for coverage and closure analysis reports. The testbench generation tool, Specman, was proven to support plan driven methodology with adequate performance for driving hardware accelerated designs on the Palladium and Xtreme families of emulators.

References:

[1] Chip Design article “What's Your Verification Game "Plan?” by Hamilton Carter, http://www.chipdesignmag.com/display.php?articleId=587

[2] Cadence Incisive Plan-to-closure Methodology white paper “Reducing block, chip, and system design risk with a “plan-to-closure” verification approach”. http://www.cadence.com/whitepapers/reducing_block.pdf

[3] Cadence Incisive functional verification newsletter article “Leading-Edge Transaction-Based Acceleration Methodology” by Kevin Donovan of Cadence Design Systems. http://www.cadence.com/newsletters/new_pdf/incisive_mar05.pdf

[4] Cadence Incisive functional verification newsletter article “SystemC-Based Virtual SOC – An Integrated System-Level and Block Level Verification Approach, From Simulation to Co-Emulation” by Laurent Ducousso, Frank Ghennassia and Joseph Bulone of STMicroelectronics, and Neyaz Kahn and Ascension Vizinho-Coutry of Cadence Design Systems. http://www.cadence.com/newsletters/new_pdf/incisive_mar05.pdf