Test Sequence Reuse from Block to System within IPCM

CDNLive! 2007 EMEA
Session 3.5

Iraklis Diamantidis
(iraklis@globetechsolutions.com)

Globetech Solutions
http://www.globetechsolutions.com

ABSTRACT

The Incisive Plan to Closure Methodology (IPCM), announced earlier this year, offers the blueprint for a complete verification process, from creating automated and executable plans to achieving system level closure. On the other hand, project windows continue to shrink and systems continue to grow, containing an ever increasing number of subsystems, clusters and blocks, most of which are often provided by 3rd party IP vendors. The effective integration of such elements into a coherent system has created a number of advanced verification applications that require increased efficiency in the way engineers can reuse verification environments at different levels of system abstraction. In order to enable such advanced verification applications, a scalable process is needed to seamlessly enable test sequence reuse across design layers, engineering teams and support chains.

This paper presents a mixed-language verification environment design process based on a "reverse-waterfall" model of stimulus generation. The solution enables sequence drivers at different layers to properly communicate and exchange test stimuli based on a concept of separation of protocol from data. A set of guidelines is proposed, complementary to IPCM, which provide a scalable solution for re-using sequences across design abstraction layers.

Introduction

As systems become ever more complex, encapsulating a plethora of subsystems, clusters and blocks, and 3rd party IP accounting for the majority of such components, it appears that the verification effort slowly shifts from verifying the components themselves, to verifying their integration into a larger unit.

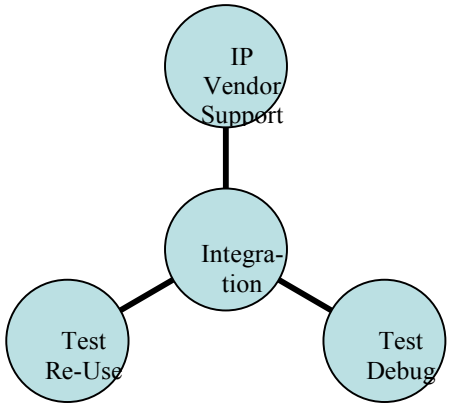
In the following sections, we shall examine the verification applications that facilitate an effective integration process, as well as the related problems that arise. Furthermore, we compare the typical system level data generation model against the proposed "reverse-waterfall" model and highlight its use in facilitating the integration process. Finally, we present real life experiences of the process using Globetech Solutions' own implementation as part of a recent customer testcase.

Verification Applications for Integration

As mentioned above, the increased integration challenges have created a number of advanced verification applications that require increased efficiency in the way engineers can reuse verification environments at different levels of system abstraction. These applications are discussed in the sections below.

Test Reuse

System integration testing requires specialized tests to be executed at different levels of abstraction. Such specialized tests often include interface intensive tests, targeted at exposing bugs at the block's interface specification. These tests are usually developed by the integrator, who has no intimate knowledge of the block(s) involved, thus making the process quite error-prone. Even worse, an invalid test that compromises a block's interface specification might be mistaken for an integration failure and needless effort will be spent investigating the entire system. Furthermore, the tests have to be re-written at every level of integration up to the system, and even across systems, with little or no reuse. With an expanding system complexity and emerging deep embedded core hierarchies, this proves to be a daunting, long and extremely error prone task.



Test Debug

System integration debug requires certain test replay at various system level abstractions, in order to better isolate bugs that arise from the integration testing mentioned above, and verify their fix. With any number of blocks/clusters/subsystems suspect and little or no expertise on them available during integration, it is becoming increasingly difficult to isolate a bug at a particular interface and verify the fix using additional tests tailored to the particular scenario.

A methodology is required, which will allow the system integrator to replay failing tests at various system level abstractions until the bug is isolated. Also, new tests need to be easily reproducible and executable at any abstraction in order to verify that the bug has been fixed.

3rd Party IP Vendor Support

Design IP vendors are finding it increasingly difficult to support their customers' integration efforts due to the wide range of design parameters and varying system topologies, with little or no information regarding the latter. This means that vendor provided tests, while extremely important for verifying the IP itself, have little use to the system integrator, since they can not easily scale to the system level. Such tests would need to be rewritten by the integrator, with little or no expertise on the IP itself.

With 3rd party IP accounting for the majority and some of the most critical components in a system, it is clear that such vendors need to be part of the overall integration process in order to enable test reuse across design layers, engineering teams and support chains.

Typical Stimuli Generation

The advanced verification applications mentioned above make it clear that a new way of generating stimuli in a complex and hierarchical system is required. When generating data for a typical cluster/subsystem/system, stimuli flow from top to bottom. A multi-channel sequence generator usually sits at the top and co-ordinates traffic to the sub systems, whereas downstream sequence generators are mostly passive and relay the transactions issued by the top-level generator (see Figure 1).

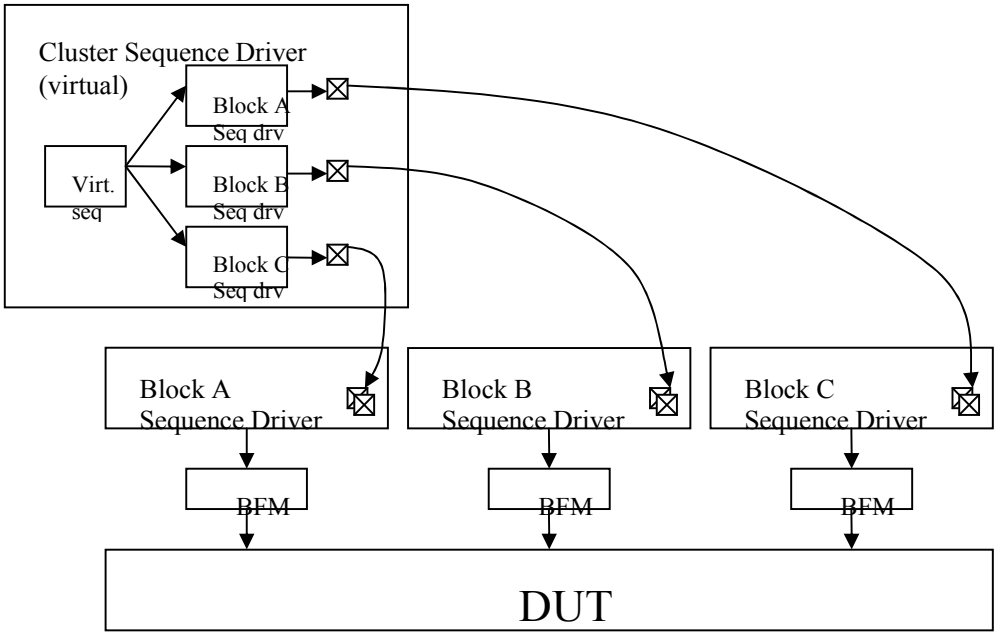


Figure 1: Typical Stimuli Generation Flow

The generated sequences from the top level multi-channel sequence generator are propagated down to the appropriate sub sequence drivers, where they are scheduled for delivery, using the appropriate Bus Functional Model (BFM), to the DUT. This process resembles a waterfall model, where data starts out from the top and flows downstream to the sub sequence drivers (Block A, B and C sequence drivers) and eventually to the DUT.

This flow offers great control and synchronization over the data flowing downstream to the blocks, but has some limitations during the integration of such a cluster into a larger system, where the cluster's native interface may no longer exist. In this case, the DUT still exists, but is embedded under a different interface, like a system-wide bus. This means that the DUT is only accessible through a new, system level interface and the cluster's test suite can no longer be executed (see Figure 2).

New, system level tests would have to be developed that use the new interface and target the cluster, incurring the problems mentioned in previous sections. The integrator would have to attempt to re-write any tests for the system level, a process that is inherently difficult since:

- No block/cluster level expertise may be available
- Protocol restrictions between the system and the cluster must be followed
- Other system components must be taken into account in order for the correct data to arrive at the cluster

A scalable and re-usable process is clearly needed in order to overcome these challenges. A specification for such a process would include:

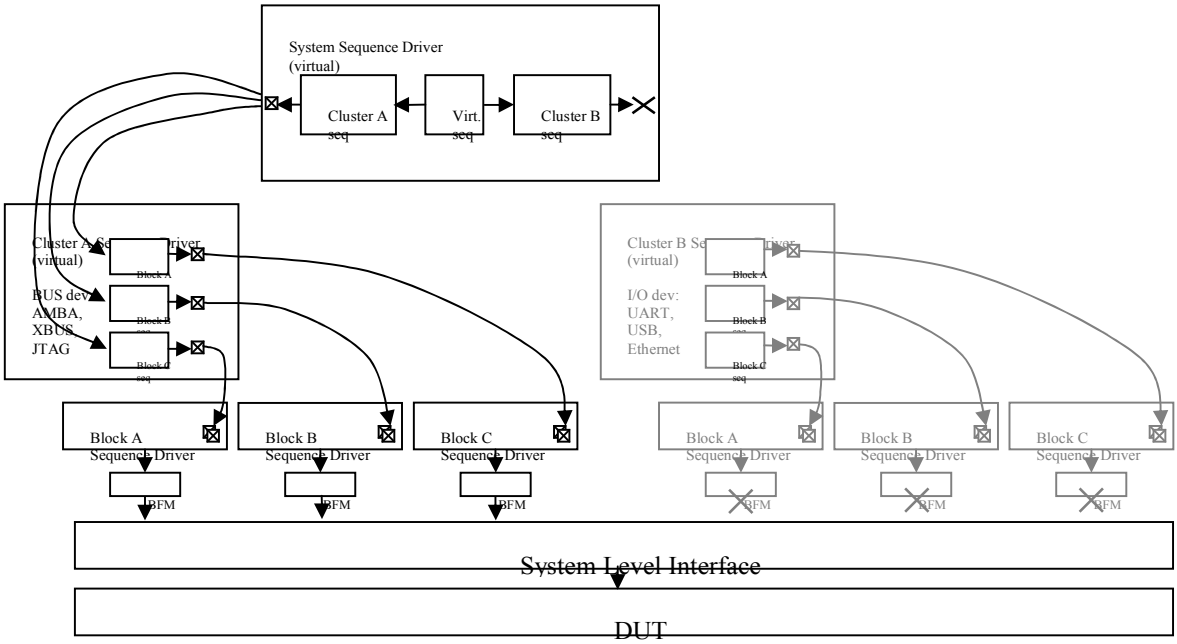


Figure 2: Cluster Integration in System

- **Separation of data and protocol**
  A sequence item already separates raw data from semantic information. However, protocol information is not necessarily separated from the raw data values. For example, consider a UART sequence item that encapsulates UART protocol specific information, such as start/stop bits and parity along with the actual payload to be transmitted.

- **Data communication to higher level entities**
  Once data is properly separated from protocol, it needs to be communicated to a higher level sequence driver. Since modern systems and verification environments may include components in different languages, e.g. SystemVerilog or SystemC, or abstraction layers (RTL/TLM), it is important that such communication is not limited to any specific language specific.

- **Data encapsulation within higher level entity**
  Once the data is received in the higher level entity, it must be encapsulated in the local format and protocol, in order to enable delivery by a local BFM. The verification environment must be intelligent enough in order to process the current system topology and shape the data to target the particular component that originated the data.

- **Delivery of data from higher level entity**
  Once the encapsulated data is ready, it can be fed to the local sequence driver like any other generated item. It can then be pushed to/pulled from the BFM and seamlessly delivered to the component that originated the data.

Reverse-Waterfall Stimuli Generation

Having considered the specification mentioned above, a new paradigm of generating sequence data can be developed, based on the “reverse-waterfall” model. Essentially, the test is now directed from bottom-to-top, instead of top-to-bottom in the typical stimuli generation model. The sequence at the block level executes like it would in

standalone mode, however, the sequence items are "trapped" and relayed to the parent (cluster) sequence driver.

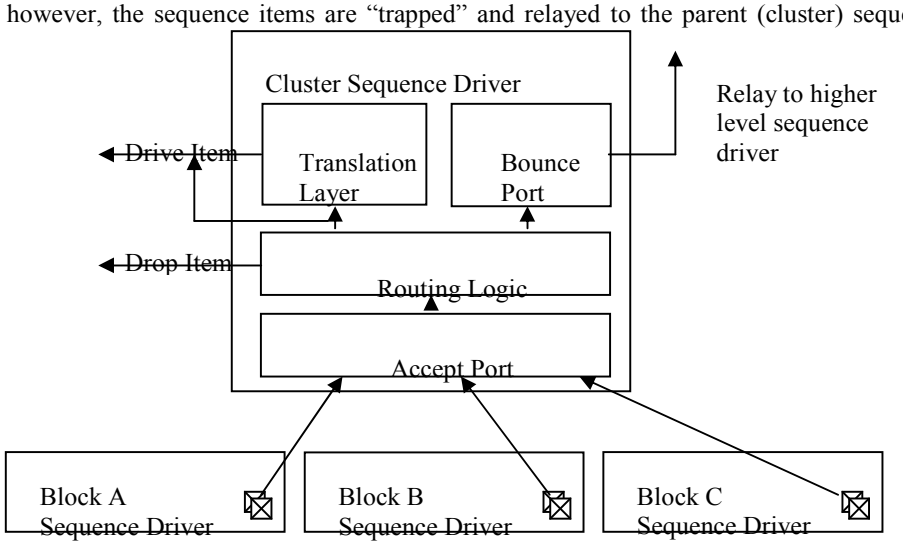


Figure 3: Reverse-Waterfall Stimuli Generation

From this point, the parent can decide whether to satisfy the sequence items locally, using a different sub-sequence driver that has an available and compatible interface to the DUT. If this is not possible, the sequence driver can attempt to bounce the item to a higher-level sequence driver until a suitable driver is found, or drop the item altogether.

The sequence driver is now extended with the following functionality:

- **Accept Port**
  - Used to accept sequences from its sub sequence drivers. The implementation of this port should be language independent to account for sequence drivers in other languages.
- **Routing Logic**
  - Used to inspect incoming sequence items and decide on the appropriate action to be taken by the sequence driver. Typical actions for the routing logic are:
    - Drop a sequence item
    - Drive a sequence item using an appropriate sub-sequence driver. Optionally translate the data from the item into a new protocol/format if interfaces are being crossed at this level
    - Combine compatible sequence items into a single transaction
    - Bounce a sequence item to a higher level sequence driver
- **Translation Layer**
  - Used to encapsulate the extracted data from incoming sequences into a new protocol or format, in order to be delivered via a new interface.
- **Bounce Port**
  - Used to bounce sequences to higher level sequence drivers, if they can not be driven locally. The implementation of this port should be language independent to account for sequence drivers in other languages.

This process scales very well in a system with deep embedded core hierarchies, as sequence items can bounce upstream through the sequence driver hierarchy until a suitable driver is found. If no such sequence driver is found and there are no further upstream drivers, the item can be dropped and the test optionally aborted. Another interesting application would be to check such items in coverage, thus producing interesting co-relations between varying system architectures. An example of a system-wide application of the "reverse-waterfall" model is shown in Figure 4.

Essentially, the test developed for Block B of Cluster A, is retargeted to use one of the available transport mechanisms in Cluster A. Since Block B is unaware of the system topology and available transport mechanisms in the system, it relays its test information further upstream until a suitable sequence driver is found. Once the sequence item is picked up by the system sequence driver, its data is extracted. The driver uses system topology information in order to select an appropriate transport mechanism through Cluster A and translates the data accordingly using the translation layer. The item is then relayed to the appropriate sequence and loaded in its respective sequence driver to be scheduled for delivery.

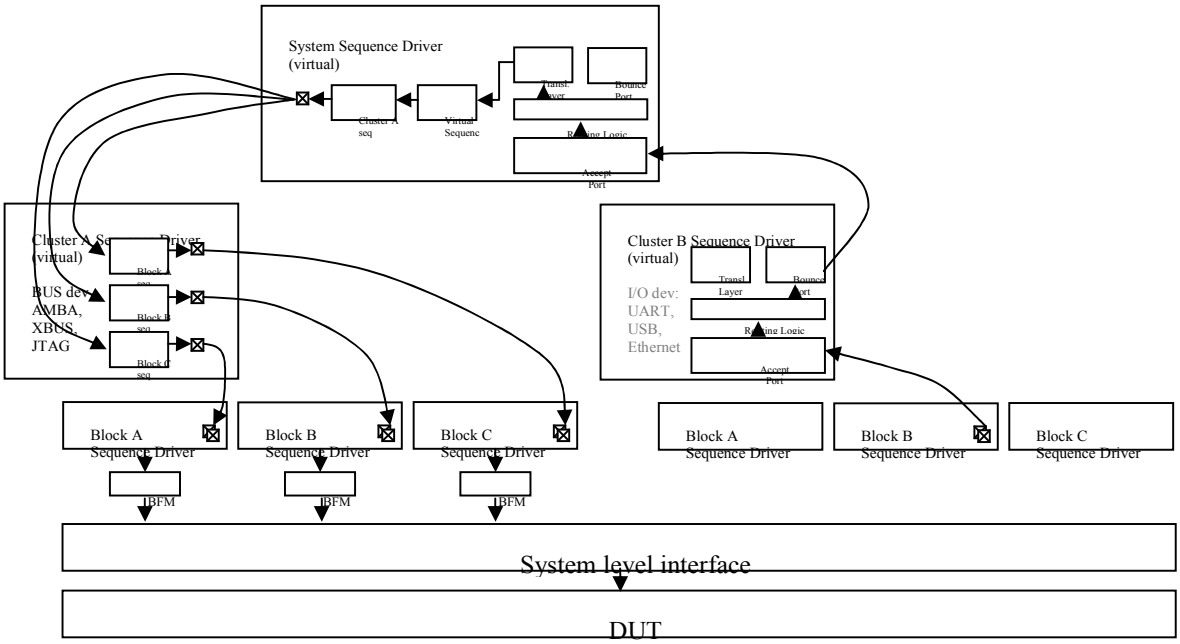


Figure 4: Test Reuse using Reverse Waterfall

This process adds several benefits to the advanced verification applications mentioned in previous sections. First, block level tests can be easily reused in higher levels of integration testing, improving reuse across the verification flow. Second, block level tests can be further randomized in order to isolate a bug discovered during integration and new tests can be developed to verify the fix. Finally, IP vendors can now deliver new tests that use specific design parameters and target specific IP behavior, while the integrator can reliably reuse such tests within different system topologies.

IPCM Compatibility

This "reverse-waterfall" process needs to be carefully implemented in order to be fully compatible with the IPCM framework. An implementation should use the predefined uRM constructs, like sequence drivers, sequences and BFMs in a non-obtrusive way, so both approaches (top-down and reverse-waterfall) can co-exist in the same verification environment. This is very important, as both approaches would be employed at different stages of the design and integration process.

An implementation of this process can take advantage of ***e***'s powerful Aspect Orientation Programming (AOP) capabilities in order to "hide" the functionality inside an aspect. This would allow for the required functionality to become active simply by changing the aspect of the sequence driver according to the verification scenario.

uRM's mixed language environment support provides the necessary constructs for effective communication of verification environments implemented in different languages (e.g. SystemC/SystemVerilog/e) and also at different abstraction layers (e.g. RTL/TLM). An implementation of the "reverse-waterfall" process would have to take advantage of this support to enable effective communication across verification environments in a system:

- **Intra-sequence driver communication**
  uRM's method ports can be used to implement language and abstraction independent communication
  e.g. bouncing a sequence item from one sequence driver to another
- **Intra-sequence driver synchronization**
  uRM's event ports can be used to synchronize transactions across a hierarchy of sequence drivers
  e.g. transmit the item_done event to a blocked downstream sequence driver in PULL_MODE
- **Intra-sequence driver data extraction and processing**
  uRM's support for common language constructs can be used to extract the data from the protocol and translate it for delivery from a new interface

Customer Evaluation

The "reverse-waterfall" process was implemented and used as part of a customer testcase of Globetech Solutions' Verification for Test (VFT) Compiler. VFT Compiler provides the automation required when verifying complex embedded test component topologies, allowing for tests developed for embedded blocks to be seamlessly reused at a variety of system configurations and levels of abstraction.

The first process of this customer testcase involved verifying the integration of 3rd party provided memory built-in self test (MBIST) controllers into an IEEE 1500 compliant wrapped core architecture. The IEEE 1500 is a scalable standard architecture for enabling test reuse and integration for embedded cores and associated circuitry. It uses serial and parallel test access mechanisms and a rich set of instructions suitable for testing cores, SoC interconnect and circuitry. The provider also made available the verification environment and test suite used to verify the IP itself, which would have to be used to verify the integration within the IEEE 1500 wrapper (see Figure 5).

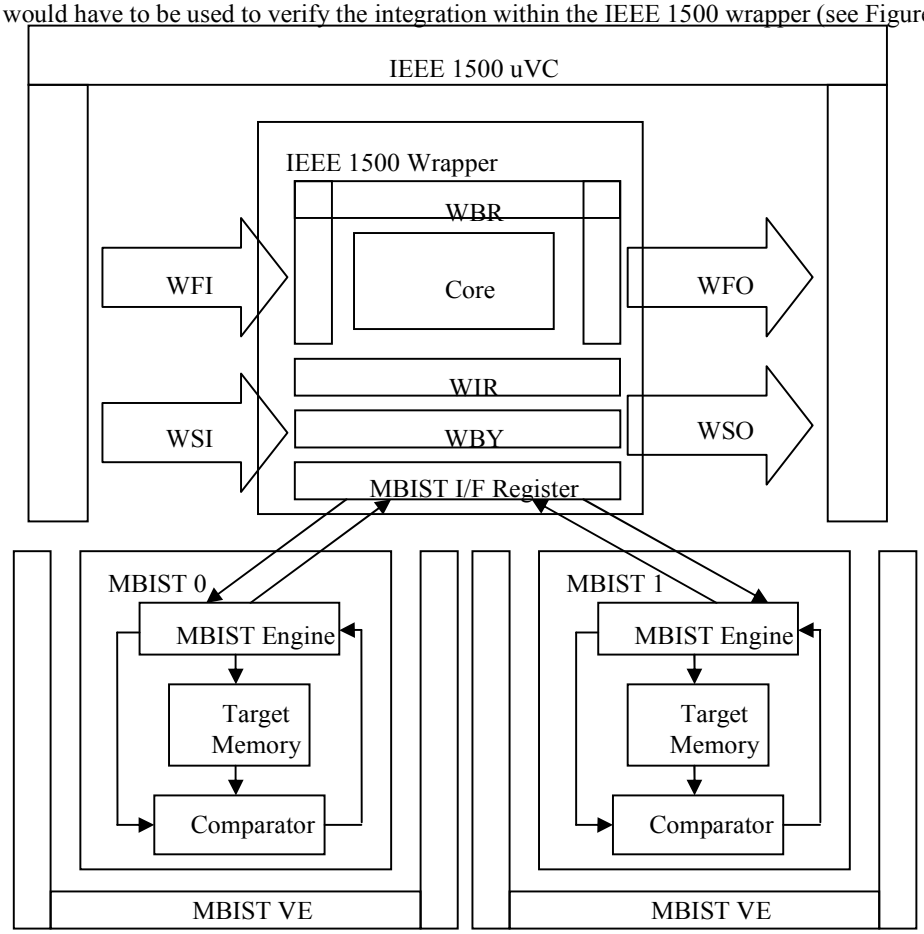


Figure 5: MBIST embedded in IEEE 1500 wrapper

VFT Compiler was used to generate a core-level verification environment which implemented "reverse-waterfall" as a new aspect of the existing environment. Using this new model, we were able to replay the 3rd party provided test suite by relaying sequences to an instance of an IEEE 1500 uVC, also instantiated by VFT Compiler, and translating them into IEEE 1500 transactions that target the embedded MBIST controller. Test stimuli were hence delivered to the MBIST DUT as if its native interface was still available (see Figure 6).

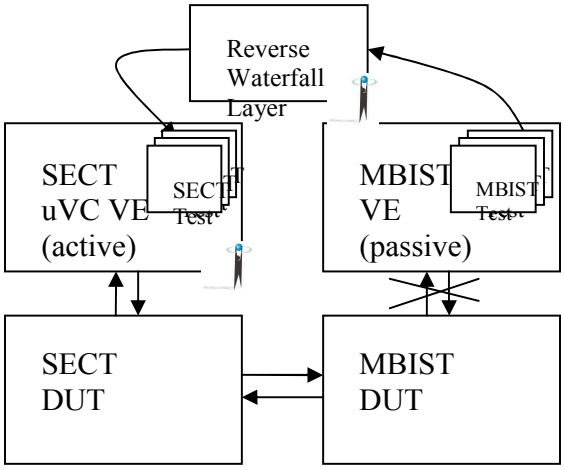


Figure 6: Test Retargeting

The second phase consisted of a broader system level scenario, involving several IEEE 1500 wrappers with embedded MBISTs, interconnected in a daisy chain (see Figure 7). This was a very interesting scenario for integration, as it was no longer sufficient only to translate the MBIST data into IEEE 1500 transactions, but also to account for system topology. Data would have to be properly offset, to account for the position of the IEEE 1500 wrapper inside the wrapper daisy chain in order to be properly delivered to the wrapper that contains the MBIST that requested it.

If, for example, the test is directed by the MBIST contained in the third wrapper in the daisy chain, then the data delivered by the system level generator must also take into account the two previous wrappers that precede the third one. Since IEEE 1500 is a serial scan based protocol, that would mean that the two previous wrappers should go into a bypass state, hence adding two extra bits in the data vector to be delivered.

More complex verification scenarios may also allowed for several MBISTs to direct the overall integration verification process (see Figure 7). In this case, the parent sequence driver chose to merge these transactions into a single transaction, as long as the merge did not violate the test flow or protocol rules. The full scan chain length was used and data from all MBISTs was concatenated to form a uniform data vector, which was eventually delivered to all the memory controllers in the system.

The full test suite developed by the IP vendor was successfully replayed, at all system levels and even with varying system topologies, with little or no need for manual intervention. At the end, the "reverse-waterfall" process was able to uncover several design bugs, related to violating the MBIST's and IEEE 1500 wrapper's interface specification, without any dedicated engineering effort.

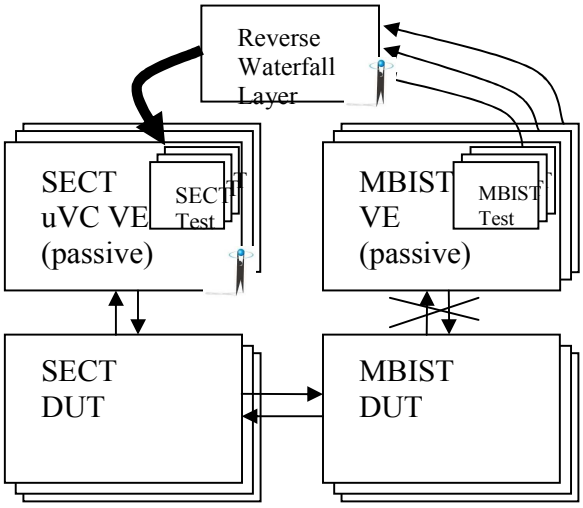


Figure 7: Multiple tests in single transaction

Conclusions/Future Work

The successful customer evaluation proved that the "reverse-waterfall" model can offer a great deal of automation and functionality during the integration of embedded cores in a complex test architecture. The main advantages of following a test sequence reuse approach to verifying block integration are productivity and predictability. Implementing environments per our proposed methodology allows for verification to begin very early in the integration process and with high levels of automation. Engineers can take advantage of this productivity gain to

also develop additional system-level scenarios using higher-level interfaces and flows for completeness. Schedule predictability is also improved using our approach. By describing the integration process as part of the core-level deliverable verification plan, IP vendors can thoroughly communicate aspects of their designs that need to be validated during integration. One of the resulting core benefits is that vendors' support overhead is hence significantly reduced.

Design for test topologies, such as the one presented in this paper, are structured and constrained architectures, making it easier to implement the "reverse-waterfall" model. However, since the process is scalable and uses predefined verification constructs, it could also be employed in other protocol based environments, and even custom designs. For the latter, it is expected that additional manual configuration will be required, thus decreasing the amount of automation possible. System topology information is critical for this type of application. System-level architectural description languages, a variety of which are currently being developed in the industry, can be used to drive and could greatly enhance the "reverse-waterfall" model and provide even more functionality and automation.

References

1) Incisive Plan-To-Closure Methodology, v 6.0, Cadence Design Systems, 2007
2) Verification for Test (VFT) Methodology v 1.0, Globetech Solutions, 2007
3) IEEE Computer Society, "IEEE Standard Testability Method for Embedded Core-based Integrated Circuits - IEEE Std. 1500-2005", New York: IEEE 2005.
4) I. Diamantidis, T. Oikonomou, and S. Diamantidis. "Towards an IEEE P1500 Verification Infrastructure: A Comprehensive Approach", presented at the 3rd IEEE International Workshop on Infrastructure IP (IIP), Santa Clara, CA, USA, May 4-5, 2005