SystemC as Completing Pillar in Industrial OVM Based Verification Environments

Wolfgang Ecker
Infineon Technologies
Am Campeon 1-12
85579 Neubiberg
+49 89 234 45334
Wolfgang.Ecker@Infineon.com

Volkan Esen
Infineon Technologies
Am Campeon 1-12
85579 Neubiberg
+49 89 234 27814
Volkan.Esen@Infineon.com

Michael Velten
Infineon Technologies
Am Campeon 1-12
85579 Neubiberg
+49 89 234 65758
Michael.Velten@Infineon.com

Tudor Timisescu
Infineon Technologies
Am Campeon 1-12
85579 Neubiberg
+49 89 234 20144
Tudor.Timisescu@Infineon.com

ABSTRACT
This paper presents a novel TLM verification approach utilizing TLM+ as a reference model and providing a systematic path to RTL simulation as well. The approach is based on SystemC only but follows the established structure on an OVM testbench. Industrial relevant aspects as use of standards, early verification, and re-use of design items are established in this way.

Categories and Subject Descriptors
J.6 [**Computer-Aided Engineering**]: Computer-aided Design

General Terms
Standardization, Languages, Verification.

Keywords
SystemC, Testbenches, Verification

1. INTRODUCTION
The SystemC based virtual prototypes following the TLM abstraction and using the OSCI TLM 2.0 interface standards gain more and more momentum in today's industrial platform based design flows. They simulate 100x – 1000x faster than RTL models and thus enable early simulation and validation of the complete system including software. However making such a model is no unique process, neither in their time of making nor in their validation.

In a bottom-up approach, RTL models exist before TLM models. The task of verification is to guarantee that the TLM model represents all important features of the RTL model - but in an abstracted way. Simulation uses selected RTL testcases and makes an RTL co- and reference-simulation. The TLM model is connected to the RTL testbench via transactors acting as abstractors. The comparison requires a hand written module to consider the accuracy of the selected TLM modeling style.

In a top-down approach, the TLM model exists before the RTL model. An own testbench has to be built using all features of a state-of-the-art constrained random simulation methodology.

Unfortunately, there is no systematic and standardized way for such kind of testbench. Sometimes classic RTL testsbenches using HVLs [3,4] are linked via TLM interface to SystemC TLM models [5]. Mainly issues as reference model, stimulus generation, or coverage are not addressed. Further on, multi-language simulation has to be established, which relies on EDA vendor specific implementations, requires object to object translation, and often causes overhead to overcome some HVL feature limitations as in object oriented features.

In this paper we present an approach that relies on the classic OVM [1] approach but moves its abstraction one level higher. Further on, the complete framework is implemented in SystemC to avoid object translation and make full use of C++'s OOP features. Therefore, we use the SystemC OVM library from Uni Paderborn [6], a coverage library from Uni Paderborn and Infineon [7], an assertion library from Infineon [8], and the constraint random library CRAVE from University of Bremen [9].

The paper is structured as follows: First, we introduce the idea of moving an OVM framework one abstraction level higher by using TLM+ abstraction techniques [10]. Next, we discuss testcase abstraction as a consequence of the OVM framework abstraction and finally, we present an approach, how the TLM testbench can be reused for RTL verification in a top-down fashion.

2. Abstracting OVM
2.1 Classic OVM structure
In Figure 1, a typical OVM testbench structure for RTL verification is shown. The sequencer is fed with items via constrained randomization. These are abstract transactions. They get translated to signal level activity by the driver, which applies them to the pins of the RTL DUT.

Sequencer, driver, and monitor are combined into a so-called agent. Several agents are used in a testbench to apply stimuli to the RTL DUT and/or take responses hereof.

The monitor does the opposite operation as the driver, converting the activity on the RTL DUTs pins back to abstract transactions. These transactions collected by the monitor are then sent to a scoreboard.

The scoreboard contains a TLM reference model of the RTL DUT and a comparator comparing the responses of the DUT and the TLM reference. Therefore, the scoreboard is fed the exact same inputs and its outputs as the DUT's. It also takes care of collecting coverage to measure the overall progress. This structure allows for the best separation of concerns, vertical and horizontal reuse, and is used by all OVM verification IPs.

CODES/ISSS'12, October 7–12, 2012, Tampere, Finland.

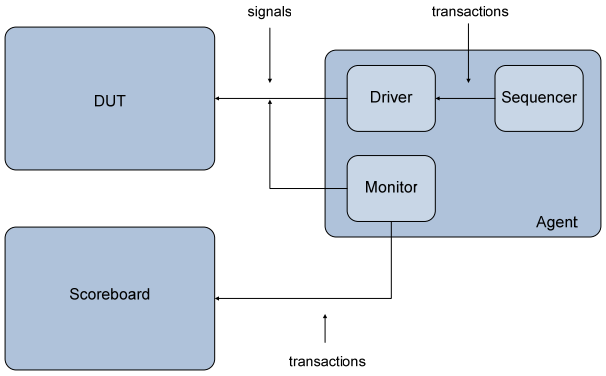

Figure 1 - Classic OVM testbench

2.2 OVM Abstraction

As mentioned before, to be able to build a TLM constrained random testbench, the testbench should operate at a higher level of abstraction than the DUT. This would mean moving from the structure shown in Figure 1 to something as in Figure 2.

In this setup, the sequencer is fed "transactions" at a higher level of abstraction than the DUT, which we call TLM+. What this higher level of abstraction exactly is will be discussed later in this paper. Exemplarily, collections of simple TLM transactions - that are functionally related and that are passed in one call (for example a set of configuration transactions) - might act as TLM+ abstraction.

The transactor has the same role as the driver in the previous picture, in that it translates from abstract stimulus (TLM+) to a format the DUT understands, in our case TLM. The observer does the same as the monitor, converting from TLM to the more abstract TLM+ and sending these to the scoreboard.

The scoreboard in the previous case contained a TLM reference model of the DUT. The challenge here is finding a more abstract description of the TLM model (which in itself is pretty abstract). A purely functionally oriented model that is register agnostic (as opposed to the TLM which is supposed to be register accurate) is an option for the TLM+ abstraction.

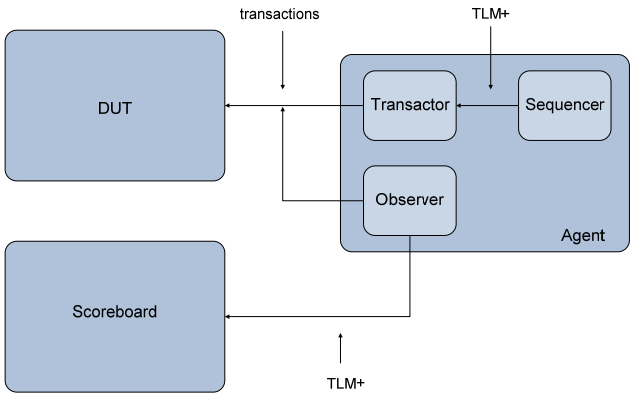

Figure 2 - Abstracted OVM testbench

3. Layered Approach to Stimulus Generation

3.1 TLM to TLM+

In the context of the current RTL verification methodology, the stimulus applied to the DUT is written at a higher level of abstraction than the DUT – instead of thinking at the signal protocol level, verification engineers can be much more productive by thinking in terms of transactions, called sequence items. For example, a write transaction to a certain address would abstract raising the write signal, putting the address on bus, putting the data on the bus and raising the ready signal. It is much easier for a human to think in terms of transactions and then have the driver convert to the signal protocol level. Such sequence item can be grouped together to create complex stimuli. The testbench also contains a reference model of the DUT, which works on the transaction level. This prevents the situation where the reference model ends up re-implementing the DUT.

For TLM verification, the two observations above apply as well. As a higher level of abstraction than TLM, TLM+ as defined in [10] is used. TLM+ achieves this higher level of abstraction in two dimensions:

- Interface abstraction: the underlying operation of the bus is not important anymore, data is no longer sent as a series of bus transactions limited by the bus width, but as a continuous stream; this saves on function calls and context switches, thus speeding up simulation

- Data abstraction: data is no longer processed throughout the chain, instead raw data is sent, along with the parameters with which it would have been processed; this saves on computational complexity, thus speeding up simulation

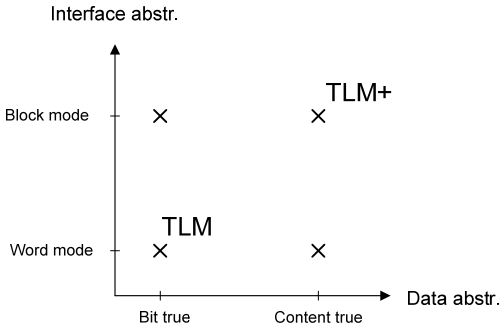

Figure 3 - TLM+ abstraction

The transactor (equivalent to the driver in RTL verification, but the name is different in order to avoid confusion) has the responsibility of converting from TLM+ stimulus to TLM transactions that the DUT can understand.

Let us analyze a simple example to better understand how such a setup would work. As a DUT let us take a simple serial to parallel peripheral, where the protocol of the serial ports is configurable through some control registers in the module. We will be concerned mostly with the parallel bus side.

If the DUT were an RTL model, as a sequence item for the bus side we would have an object that contains the command to be

executed (read or write), the address to which the access is to be executed and a data member. The data member for a write item would contain the data to be applied and for a read it would contain the value with which the DUT responded to the request. The driver would take these values and apply them to the appropriate signals according to the protocol.

For a TLM DUT, a simple sequence item has instead of a data member which is of the same size as the bus width, a stream of data we would like to pass to the model (or to read data in, for the case of a read transaction). The transactor would separate this long stream into individual transactions that the DUT would be able to interpret. Such a sequence item is more abstract than the TLM transactions the DUT operates with in just the interface abstraction dimension. We could do better, by applying data abstraction as well.

In the case of data abstraction, we actually have to interpret the data being sent. For our simple case of a serial to parallel peripheral, three simple operations can be performed on the bus: send via serial, receive from serial and configure for a certain serial protocol. Serial send and receive are just writes and reads, respectively to specific register addresses. The benefit in actually defining them as a send and a receive sequence item comes in the reference model; instead of having to examine the address field and the command to see if it was a write to the TX register or a read from the RX register, it would know directly what operation was performed and predict a certain output from the DUT. This will result in less code that focuses more on the functional aspects of the DUT. The third sequence item type, configuration, would go even further: instead of abstracting an access to just one register, it would abstract a sequence of accesses to the configuration registers. It would also contain the desired parameters of the serial protocol in an abstract form, a configuration object with fields for each parameter. The transactor would take this configuration object and encode it into the appropriate data values for each configuration register and create a TLM transaction for each.

Clearly, applying also data abstraction will result in a bigger productivity gain than just applying interface abstraction. The problem with data abstraction is that it requires DUT specific information. In the previous example, send and receive items require knowledge of the offsets for the TX and the RX register. The configuration item, aside from the offsets of the configuration registers, also requires information about the serial protocol and the modes that the DUT supports. The interface abstraction does, however, not require any DUT specific information, just protocol specific information (the way in which a data stream is split into multiple transaction may be different – some protocol may support pipelined accesses, some may support bursts, etc.).

As the effort spent on verification tasks does not directly contribute to the profit a chip will make, it is desired that the amount of code developed be minimized, hence reuse is very important. Mixing in DUT specific information in a piece of VIP will reduce its reuse drastically. As we have seen above, data abstraction does have this drawback, but interface abstraction does not.

The conversion from a data stream to a series of TLM transactions is non-trivial and is something that would be of interest for reuse. Data abstraction does however provide big leaps in productivity and is not to be ignored either. To leverage the advantages of both while mitigating the drawbacks, a layered approach is used.

3.2 Layering for RTL Verification

The foundation of any piece of VIP is the sequence item. For RTL verification, there is usually just one sequence item that can abstract all types of transactions the DUT can handle. Any other sequence item types would just add extra constraints to this item or facilitate error injection.

Sequences are built from sequence items, with the constituent items having some sort of temporal relation. A series of protocol specific sequences can be developed, for example, writing to an address and then reading the value from the same address back one clock cycle later, etc. A DUT specific layer can be built on top, which introduces some DUT specific information, such as writing a '1' to all registers in the design (requires knowledge of what offsets are actually present in the design from the whole address space of the protocol). Such sequences can be then used at the highest level of layering, virtual sequences, which coordinate stimulus between multiple verification components.

In Figure 4, such a layered stimulus hierarchy is presented. Light grey elements are part of the VIP package and can be reused across projects. White elements are project specific and cannot be reused (except for related projects, such as new derivatives, etc.).

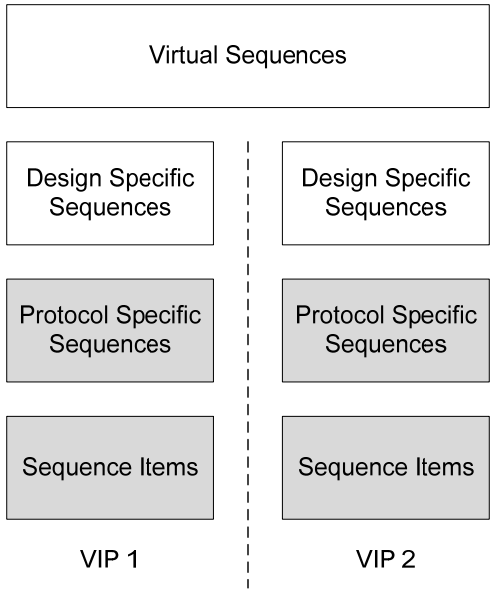

Figure 4 - Layered RTL stimulus

3.3 Layering for TLM Verification

We can have a nicely layered approach for TLM verification as well, in order to be able to achieve as much reuse as possible. The picture looks slightly different in this case (ignoring virtual sequences).

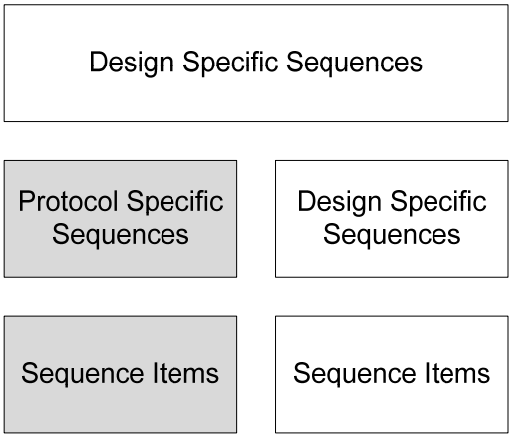

Figure 5 - Layered TLM stimulus

In Figure 5 sequence items in light gray only have interface abstraction. Protocol specific sequences are built up from these. Sequence items that employ also data abstraction are marked with white. Design specific sequences are built up from these as well (at the same conceptual level of hierarchy as the protocol specific sequences). All of the above are then be used to build up even higher sequences, which will also not be reusable.

4. Reuse from TLM to RTL Verification

Sequences from TLM verification have great potential of reuse for RTL verification. What is missing is the conversion from the TLM agent output (usually a TLM 2.0 generic payload) to the signal protocol level. This could be done in one adapter component. But assuming that the VIP for RTL verification already exists, a converter from the TLM agent's output to the RTL agent's driver sequence item is required. Such a conversion is easy to do, as both items sit at the same level of abstraction.

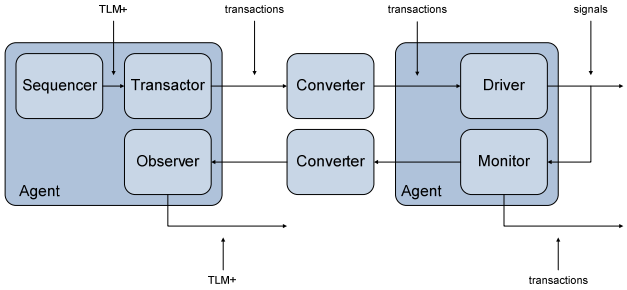

Figure 6 - Mixed TLM/RTL agent

In Figure 6, the converter between the transactor and the driver handles the conversion between the output format of the TLM agent (generic payload) and the sequence item of the RTL agent. To be able to provide monitoring, a converter doing the opposite must be placed between the monitor and the observer. This is only required if the reference model would be a TLM+ model. If using the previously verified TLM model as a reference model, this can be skipped.

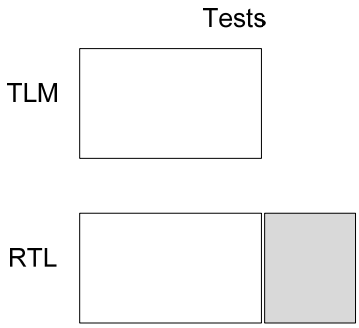

Figure 7 - Test hierarchy

The drawback with this approach is that the stimulus will all have to be written at the TLM+ level, because the VIP only has a TLM+ interface. The timing achievable with such sequences might be too coarse and RTL DUT's interesting stimulus scenarios might not be reachable. As RTL includes more detail than TLM, it will almost surely be the case that additional tests will have to be performed to validate the extra features. In Figure 7, such a test hierarchy is presented. The tests that compose the light gray suite are enough to verify the TLM model. The RTL model, providing additional levels of details in terms of bit and cycle accuracy, will require extra tests to verify these, pictured in the white test suite. Another drawback is that the source code for the RTL agent has to be modified, which is undesirable.

In Figure 8, a more advanced version of a mixed TLM/RTL agent is presented. The main difference here is that the RTL agent's sequencer is still present. The converter starts items on this sequencer, which will eventually get to the driver. In this case, the VIP can be stimulated using either the sequences developed for TLM verification (through the TLM agent sequencer) or with newly written sequences written at the TLM level of abstraction (through the RTL agent sequencer). This allows implementing the test hierarchy in Figure 7, while maximizing reuse.

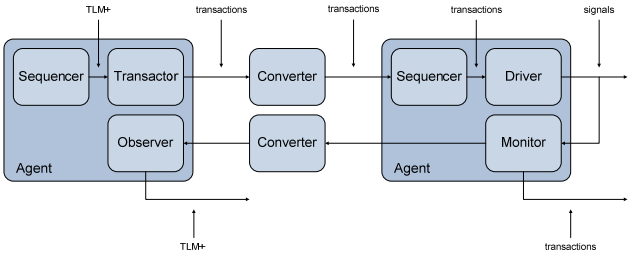

Figure 8 - Mixed TLM/RTL agent (advanced)

5. Summary and Outline

We presented a novel testbench structure supporting top down verification of a TLM DUT by using TLM+ abstraction in the testbench's agents and the scoreboards. Further on, this structure is extended to support reuse of testcases as well as addition of new testcases for an RTL DUT. It is important to mention, that this concept follows the OVM methodology and could also be implemented in SystemVerilog.

When available, we will move the testbench structure to a UVM [2] SystemC implementation. Finally, we plan a generator for basic testbench structures to simplify the use of the concept.

ACKNOWLEDGMENTS

This work has been partially funded as part of the funded Project SANITAS (01M3088) by the Federal Ministry of Education and Research.

6. REFERENCES

[1] Accellera Organization Inc. Open Verification Methodology (OVM). (2009,May). Available at: http://www.accellera.org/activities/ovm/

[2] Accellera Organization Inc. Universal Verification Methodology (UVM), May 2012. www.uvmworld.org

[3] IEEE Computer Society. Standard for the Functional Verification Language e. IEEE Std 1647-2011, 2011.

[4] IEEE Computer Society. IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language - IEEE Std 1800-2009, 2009.

[5] Glasser, M.; Bergeron, J.:TLM-2.0 in SystemVerilog. Proceedings of the DVCON2011. Available at: events.dvcon.org/2011/proceedings/.../04_2.pdf

[6] C. Kuznik and W. Müller. Functional Coverage-driven Verification with SystemC on Multiple Level of Abstraction. Proceedings of DVCON, 2011.

[7] M. F. S. Oliveira, C. Kuznik, W. Mueller, W. Ecker, and V. Esen. A SystemC Library for Advanced TLM Verification. In Proceeding of Design and Verication Conference (DVCON), Mar. 2012.

[8] W. Ecker, V. Esen, M. Hull, T. Steininger, M. Velten: Requirements and Concepts for Transaction Level Assertions. International Conference on Computer Design, 2006. ICCD 2006.

[9] F. Haedicke, H. M. Le, D. Große, and R. Drechsler. CRAVE: An advanced constrained random verification environment for SystemC. In International Symposium on System-on-Chip (SoC), 2012. Available at www.systemc-verification.org.

[10] W. Ecker, V. Even, R. Schwencker, and M. Velten, "Defining TLM+," in *Design & Verification Conference & Exhibition (DVCON)*, San Jose, USA, February 2010.

[11] Accellera Organization Inc. SystemC 2.3 (Includes Transaction-level Modeling). Available at: www.accellera.org/downloads/standards/systemc