An Automatic Visual System Performance Stress Test for TLM Designs

George F. Frazier
Cadence Design Systems, Inc.
georgef@cadence.com

Neeti Bhatnagar
Cadence Design Systems, Inc.
neeti@cadence.com

Woody Larue
Cadence Design Systems, Inc.
larue@cadence.com

Vincent Motel
Cadence Design Systems, Inc.
vmotel@cadence.com

ABSTRACT
Performance analysis is an important aspect of TLM 2.0-based system design. While case-specific performance analysis can be hand coded into any model, it is possible to compute useful performance analysis metrics in a generic fashion for TLM 2.0 models.

This work shows how the TLM 2.0 framework can be leveraged to create an automatic visual system performance stress test for SOC designs. The approach is generic yet powerful – all that is required is a model of the system which follows the TLM 2.0 standard and the ability to increase the initiated traffic on the system or selected portions of the system in a steadily increasing fashion as time advances. Using the stress test framework requires no code changes on the part of the system designer.

We demonstrate use of the framework in the design of a high performance interconnect for a high speed memory sub-system. For the stress test, the framework measures throughput of the interconnect. The "stress" point we observe involves increasing traffic from the initiators to the memories. At capacity, the amount of data overwhelms the system.

Categories and Subject Descriptors
D.3.3 **[System-Level Design]:** Experience using ESL and/or TLM for system-level design and verification.

General Terms
Algorithms, Performance, Design, Standardization, Verification.

Keywords
TLM, ESL, System-Level Design, Performance analysis, Virtual Platforms, SystemC, C++.

1. INTRODUCTION
Performance analysis is an important aspect of TLM 2.0-based system design. While case-specific performance analysis can be hand coded into any model, it is possible to compute useful performance analysis metrics in a generic fashion for TLM 2.0 models[1]. This is possible because TLM 2.0 specifies a standard bus model for memory-mapped architectures that can be instrumented to compute performance metrics.

We begin by describing a tool for automatic TLM 2.0-based performance analysis and looking at the semantics of the response status of the generic payload.

Then we show how the TLM 2.0 framework can be leveraged to create an automatic visual system performance stress test for SOC designs. The user selects a set of paths in the design and the framework monitors the values of the generic payload objects as they flow through the transport functions in the selection. From this, a moving window of system throughput is computed. In addition, the response status values computed by the transport functions are monitored to track failure percentages. In many well-constructed TLM 2.0 designs, throughput and TLM response status are causally linked. For such systems, once the load approaches maximum throughput (capacity) the number of failed transactions will increase until throughput plunges as the system fails to function correctly.

Finally, we demonstrate use of the framework, and construction of an automatic stress test, in the design of a high performance interconnect for a high speed memory sub-system. The stress test provides an abstract way to visualize when the system reaches capacity. The framework automatically creates a line chart with time on the X axis and throughput and TLM response status values on the Y axis. At first, throughput increases and all responses have the TLM_OK_RESPONSE value. Shortly before capacity is reached, throughput drops off and the graphs for response status values other than TLM_OK_RESPONSE begin to spike, indicating the system has started to fail.

2. TLM 2.0-BASED PERFORMANCE ANALYSIS
Typically, case-specific performance metrics are computed as a part of simulated models of system components. Cache models, for example, often compute and report ratios of hits and misses[2]. Bus models often contain monitors used to sample latencies and report statistics on the usage of buffers[3]. While case-specific performance analysis can be built into any simulated model, because TLM is a standard targeted for memory-mapped bus models[4], it is possible to create tools that compute useful performance analysis metrics in a generic fashion without requiring any source code changes on the part of the designer. Such an approach allows automatic collection of quantitative data from a TLM simulation.

In our approach, we developed a tool having native TLM 2.0 knowledge to monitor the traffic between all TLM initiators and targets during the simulation. It computes two basic types of system performance metrics: untimed and timed. Untimed metrics can be useful for functional models lacking a high degree of timing accuracy that still provide enough detail to answer important questions about design trade-offs.

From simple counting metrics, more complex statistics can be computed. The same basic counting infrastructure is the basis for timed performance metrics. Timed statistics add one more key element: latencies. Timed metrics are valuable only to the degree that a sufficient level of timing accuracy is available in a design, as described in section 3. With TLM modeling, there is a tradeoff between timing accuracy, simulation speed, and model development effort. Functional models based on the loosely-timed coding style generally are not accurate enough for timing metrics to yield much information. However for TLM designs using the approximately-timed coding style that approach cycle accuracy, timing accuracy is sufficient to make the gathered performance metrics very informative. The high performance interconnect described in section 4 is such a model.

For such models, average and peak throughput (the number of bytes transferred as a function of time) at each initiator or target interface is still the fundamental metric, but the minimum, maximum, and mean latency values can be of critical importance too.
Those metrics form the basis of the stress test in section 5.

3. MODELING OPTIONS FOR APROXIMATELY TIMED SYSTEMS

3.1 GENERAL ASPECTS OF APROXIMATELY TIMED MODELING

The TLM 2.0 standard is specifically aimed at modeling memory-mapped buses, where multiple initiators initiate transactions in parallel, with or without synchronization. The transactions flow through an interconnect structure that routes them to multiple targets that make the appropriate processing and send the responses back to the initiators.

To correctly model the performance of such systems, the time to initiate, transmit and process the transactions must be taken into account. Most physical implementations would use clocks to time the transfers, but transaction level modeling, which is based on function calls to represent transaction, precludes the use of clocks. Instead, the timing of the elements is modeled by delays between simulation events, usually representing a count of clock cycles. This approach results in faster simulation speed and simplifies the overall modeling effort.

Modeling accumulative delays is straightforward in blocking models based on the `b_transport` function of TLM 2.0, but it is generally not sufficient to represent the numerous transactions that flow in parallel into a modern pipe-lined design that is usually modeled with a bidirectional sequence of `nb_transport_fw` and `nb_transport_bw` non-blocking function calls.

Delays are distributed throughout various parts of the system:

- Initiators send transactions at a specific throughput, which defines the delay between the start of two consecutive requests.
- The interconnect structure can present transmission delays and generally introduces delays where it must serialize multiple concurrent transactions on a single physical link (e.g. in arbiters).
- The targets add latency during the processing of the transactions.

By adding those delays, models take into account the physical limitation of the elements in terms of bandwidth (because the physical clock frequencies are not infinite). They should also take into account the limited number of concurrent transactions (because the physical buffer sizes are not infinite).

3.2 LOAD ADAPTION

We should note however that in a well constructed interconnect, it is possible to adapt initiators whose peak throughput is higher than the maximum bandwidth of the targets, with some conditions:

- The peak throughput should be limited in time, so that average throughput is within the capacity of the system.
- Some sort of FIFO elements must be used to buffer the extra traffic during activity peaks and transmit the transactions at a rate acceptable by the targets.

Similarly, if some buffering is available before the arbiter, and if the partial bandwidth allocated to the initiator is sufficient for its average throughput, an initiator can send traffic to an arbiter in a non-blocking fashion without waiting for the arbiter to select its transactions. As a result, the traffic load adaptation and the absorption of the peaks strongly depend on FIFOs. With FIFO models of infinite size, systems would seem to run well for any value of throughput from the initiators, but the number of transactions held in the FIFOs might never decrease and could increase infinitely.

It is a better modeling practice to represent FIFOS using a finite size that supports:

- A sufficient total number of transactions
- A sufficient level of write data on the forward paths
- A sufficient level of read data on the backward paths

The latter two parameters are needed if the TLM 2.0 transactions are not limited in size, otherwise the FIFOs could hold transactions transporting an arbitrarily large number of bytes, which does not correspond to physical reality.

Finally, we need to consider the behavior of a FIFO when it cannot hold a new transaction because it has filled up or lacks capacity to store the data of the transaction.

Two main options are possible:

- Stall the initiator until the FIFO fill level has decreased and the transaction can be accepted.
- Return with an error response to the initiator.

The choice can depend on the actual system being implemented. Not all initiators may support variable traffic rates (they cannot be stalled). From a modeling point of view, returning errors enables a simpler initiator design and can help pinpoint performance weaknesses of a system under a specific load.

3.3 RESPONSE STATUS OF THE GENERIC PAYLOAD

The TLM 2.0 generic payload includes some of the attributes found in typical memory-mapped bus protocols such as command, address, data, byte enables, single word transfers, burst transfers, streaming, and response status[4].

Response status is implemented as an enum with values:
TLM_OK_RESPONSE, TLM_INCOMPLETE_RESPONSE,
TLM_GENERIC_ERROR_RESPONSE,
TLM_ADDRESS_ERROR_RESPONSE,
TLM_COMMAND_ERROR_RESPONSE,
TLM_BURST_ERROR_RESPONSE,
TLM_BYTE_ENABLE_ERROR_RESPONSE.

The Generic Payload class includes two helper functions – `is_response_ok` and `is_response_error` – to determine

the error status of a generic payload object. `is_response_ok` is true if and only if the response status if TLM_OK_REPONSE and `is_reponse_error` returns true if and only if the response status is not equal to TLM_OK_REPONSE. Thus in a proper TLM 2.0 design, any response other than TLM_OK_RESPONSE indicates an error condition.

The TLM 2.0 standard specifies a precisely defined semantics for TLM_ADDRESS_ERROR_RESPONSE, TLM_COMMAND_ERROR_RESPONSE, TLM_BURST_ERROR_RESPONSE, and TLM_BYTE_ENABLE_ERROR_RESPONSE. Models must adhere to the definitions, in order to ensure interoperability of the models and debuggability of the system.

However the TLM_GENERIC_ERROR_RESPONSE can be used with more flexibility to indicate other types of errors not defined by the standard. The errors related to performance modeling, such as errors from FIFOs, can be represented using TLM_GENERIC_ERROR_RESPONSE.

Section 4 describes a high performance interconnect modeled at the AT level that uses this strategy for indicating failed transactions when buffers are filled up to their maximum capacity.

4. A HIGH PERFORMANCE INTERCONNECT FOR A HIGH SPEED MEMORY SUB-SYSTEM

As a case study, we examine the design of a high performance interconnect with the following architectural requirements:

- High performance:
  - Good utilization of the total memory bandwidth.
  - Keep latency reasonably low.
- Many parallel initiators, fewer parallel memory interfaces.
- Initiators:
  - Usually not all active at the same time.
  - Very different traffic profiles.
  - Different priorities.
- Target memories:
  - Large capacity and high speed SDRAM, such as the upper speed bins of DDR3 SDRAM[5].
  - Limited number of independent interfaces (at most 2 or 4).
  - Able to receive transactions from any of the initiators (fully shared).
  - Complex timing:
    - Depends on address locality (penalty for row change).
    - Depends on command sequence.

Figure 1 shows the specification of an interconnect structure for 64 initiators and 4 SDRAM interfaces running independently and concurrently.

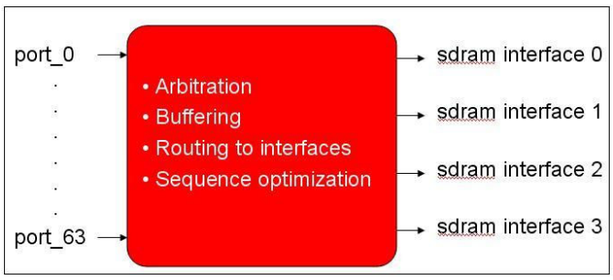

Figure 1. Generic Interconnect

Several architectural options are possible to design this interconnect, depending on the tradeoffs of cost and performance. The two extreme are:

- A full crossbar that enables all paths from any initiator to the memories in parallel and only arbitrates transactions in front of the memories. This is the solution with the highest silicon area.
- A single arbiter to serialize the transactions from all initiators and a single router to select the target memory interface. The area is minimal, but the total throughput is very limited.

We consider an intermediate solution: a semi-cross-bar made of three main layers of transaction arbitration and routing. It enables several parallel transaction paths but is not as large as a full cross bar.

Figure 2 shows the forward path. Figure 3 shows the backward path.

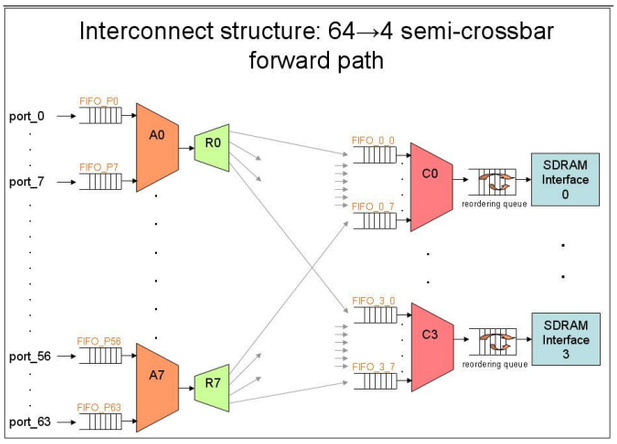

Figure 2. Forward Path.

Arbiters A0 – A7 and C0 – C7 arbitrate between incoming transactions for use of a shared resource. They implement a priority policy and perform the routing on the backward path. Routers R0 – R7 simply route transactions according to their addresses. From a performance perspective, routing is transparent. They must arbitrate on the backward path. The FIFOs between the ports and arbiters support simple buffering of transactions waiting for arbitration. They have a limited size (for transactions and write data) and return errors when they are full. There is no buffering on backward path. The reordering queues provide a size limited buffering capability similar to the FIFOs. They implement an algorithm to re-order transaction sequence so that SDRAM usage is more optimal. They do not need to make any buffering on backward path.

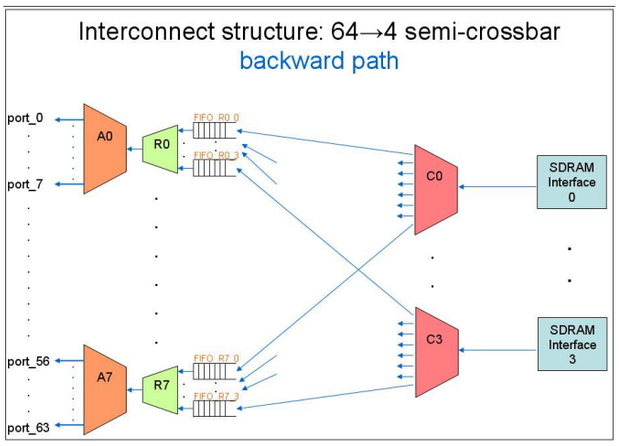

Figure 3. Backward Path.

The interconnect has two types of arbiters. The simple arbiter (Figure 4) selects one request, sends it, and waits for the response before selecting a new request. A simple arbiter supports only one outstanding transaction per port (required for non-reentrant targets). It serializes full (request + response) transactions. Transmission time may be modeled in the target.

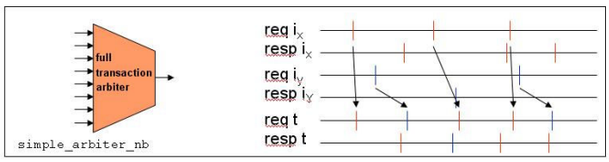

Figure 4. Simple Arbiter.

The simple req arbiter (Figure 5) selects one request, sends it, and selects a new request without waiting for the response. It supports any number of outstanding transactions and must tag the requests to route the responses. It serializes requests (responses are already serialized by target). Simple req arbiters model the time needed to transmit a request, typically constant + proportional to write data size.

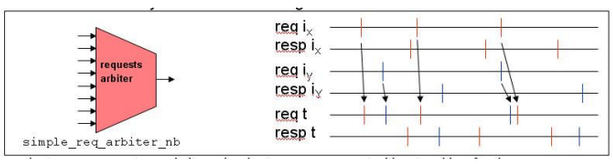

Figure 5. Simple Req Arbiter.

5. THE STRESS TEST.

To demonstrate the automatic visual stress test, we took a design using AT level TLM 2.0 modeling of the system described in section 4. Again, the model buffers elements up to the capacity of the buffers. After that, the `nb_transport_x` calls set the TLM_GENERIC_ERROR_RESPONSE response status for reads and writes that cannot be successfully completed. Given the fact that the interconnect has a fixed upper capacity, driving input beyond that capacity will result in failed transactions.

The Response Status Chart and Throughput Chart shown in Figures 6 and 7 respectively constitute a visualization of the system stress test for the design of section 4. In both charts, time is displayed on the X axis. The activity ratio of the design is defined as zero when all initiators are "off" and producing no traffic (reads and writes through the interconnect to a memory), and one when all initiators are "on" and producing traffic at maximum capacity. The charts show the results from a simulation where the activity ratio was slowly increased from zero to one over a span of 100 us (of simulated time).

5.1 RESPONSE STATUS

The Response Status Chart (Figure 6) shows the response status as the activity ratio is increased. The green line shows the number of successful transactions (TLM_OK_REPONSE) processed in the 100 us sampling period. The orange line shows the number of transactions which failed (TLM_GENERIC_ERROR_RESPONSE) because the system was overloaded and queues were full. Notice that at low activity ratios (system load) almost all transactions were successful, but after the activity ratio exceeds 0.2,

5.2 THROUGHPUT

This chart shows the offered load and throughput as the activity ratio is increased. Notice that the offered load increases linearly with activity ratio (as you would expect). The throughput of transactions through the system closely tracks the offered load (because at low loads nearly all the transactions are successfully processed). Once, the activity ratio approaches 0.2, the throughput levels off and declines slightly with increasing activity ratio. This is expected because once the system is completely saturated, nothing more can be pushed through the system regardless of how many transactions are generated by the initiators.

5.2 GENERIC

To reiterate, because of the standardization of the generic payload under TLM 2.0, this test is constructed in a generic fashion – there is no need for the user to modify their code as long as they use a tool for constructing such performance analysis that analyzes the values of the generic payload. The two charts constitute an automatic visual stress test for a TLM 2.0 design.

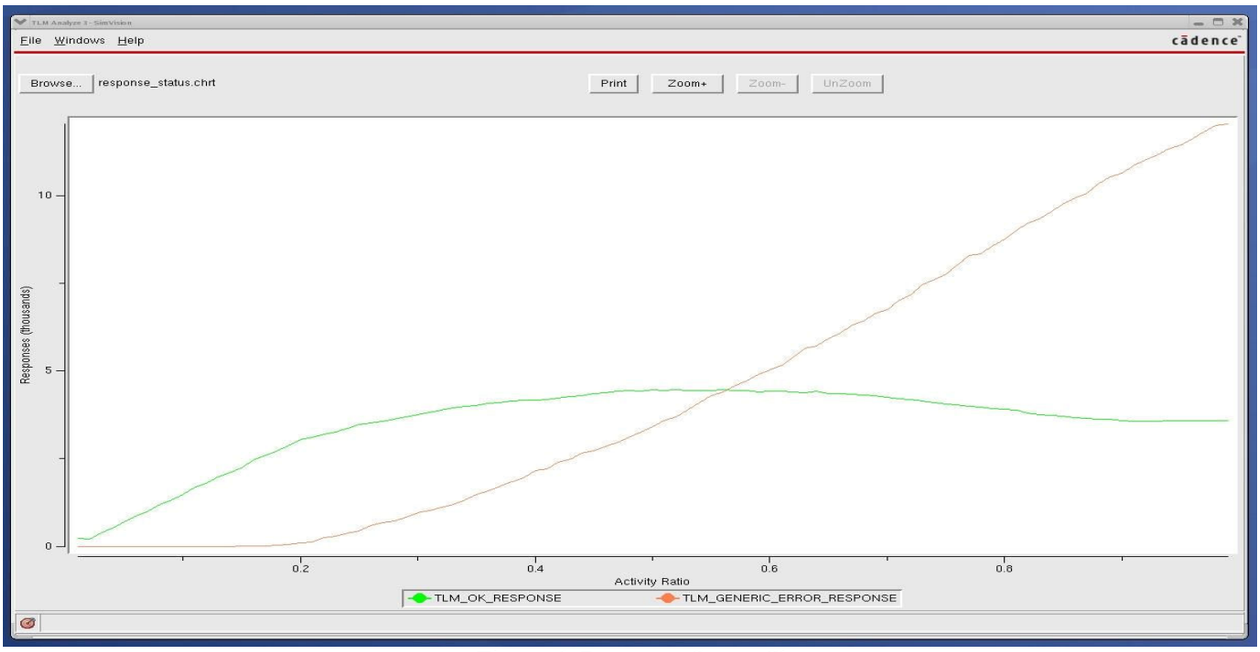

Figure 6. Response Status Chart.

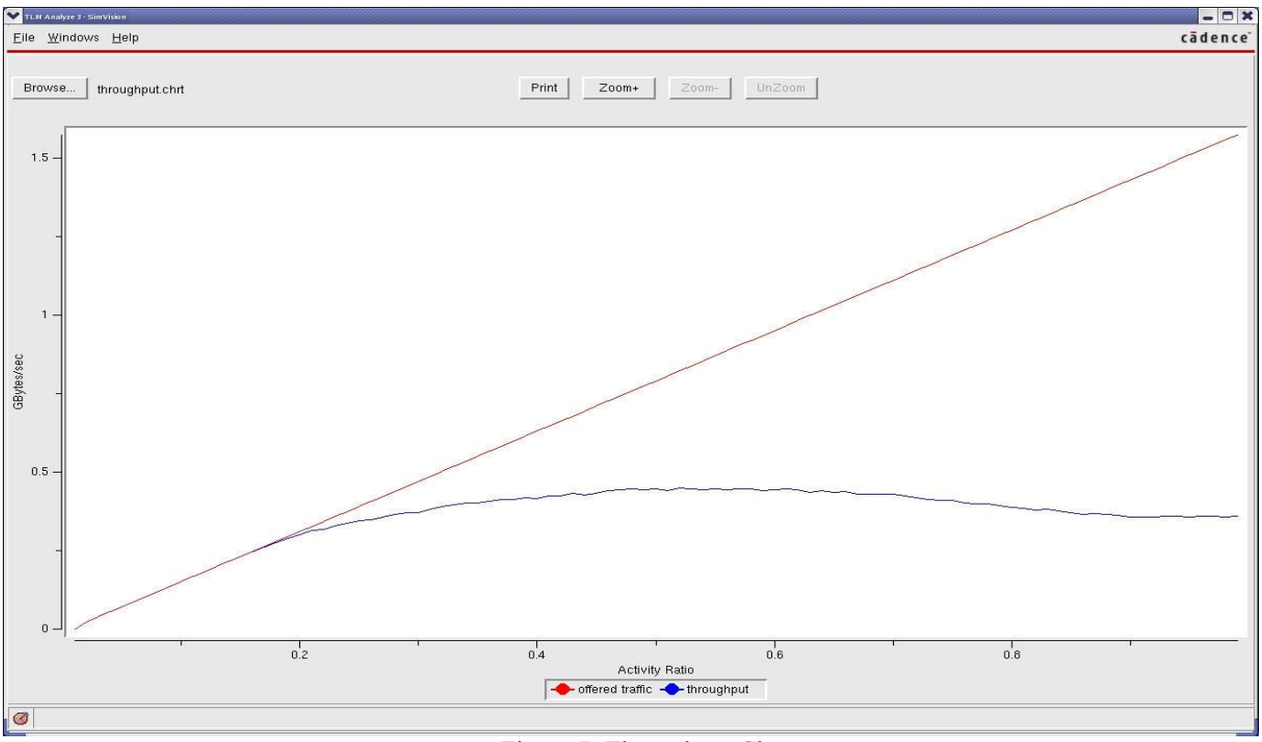

Figure 7. Throughput Chart.

6. CONCLUSION.

This work has demonstrated the use of a new SystemC technology that automatically generates system performance statistics based on the TLM 2.0 standard. It shows how the tool can be used to construct an automatic visual stress test for well constructed TLM 2.0 designs modeled at the AT level of abstraction. The existence of the generic payload – that makes interoperability between models of disparate blocks of IP possible – enables the creation of performance analysis metrics and tools without modifying individual designs. For results to be meaningful in real usage, a sufficient level of timing must be modeled in the system. The model must also be well constructed – when buffering elements are out of capacity, the response status has to be set accordingly. These sorts of automatic analyses, made possible by the TLM 2.0 standard, bring the promise of better debugging and analysis of SystemC-based SOCs.

7. REFERENCES

[1] Frazier, G., Motel, V., Bhatnagar, N., Larue, W. "Automatic Quantitative Analysis of Simulations of TLM 2.0 Loosely Timed Models." Proceedings of DesignCon. Feb. 2010.

[2] Goodman, J. "Using cache memory to reduce processor-memory traffic." Proceedings of the 10th annual international symposium on Computer architecture. Pages 124-131. Stockholm Sweden. 1983.

[3] Allan, G. "The Love/Hate relationship with DDR SDRAM Controllers," Design and Reuse. http://www.design-reuse.com/articles/13805/the-love-hate-relationship-with-ddr-sdram-controllers.html.

[4] Open SystemC Initiative. "TLM-2.0 Standard." http://www.systemc.org/downloads/standards/tlm20. http://www.cadence.com/rl/Resources/white_papers/tlm-wp.pdf.

[5] Jedec Global Standard Committee. http://www.jedec.org/standards-documents.