

Constrained Random Test Environment for SoC Verification using VERA

Victor Besyakov, David Shleifman

Tundra Semiconductor Corporation

victor_besyakov@tundra.com, david_shleifman@tundra.com

ABSTRACT

Tundra Semiconductor is creating the next generation of system interconnect designs. Our typical design supports 10 buses, with at least four of the buses having different protocols. All of the buses map many different transactions from a one bus to another bus.

In system verification, every block of the design must undergo detailed block level verification. Because of the complexity of the designs and the amount of resource and time associated with verification efforts it is almost impossible to complete 100% functional coverage. However, if the functional verification is targeted to three to five typical applications where the device is going to be used, then the desired coverage is reduced which also reduces verification time.

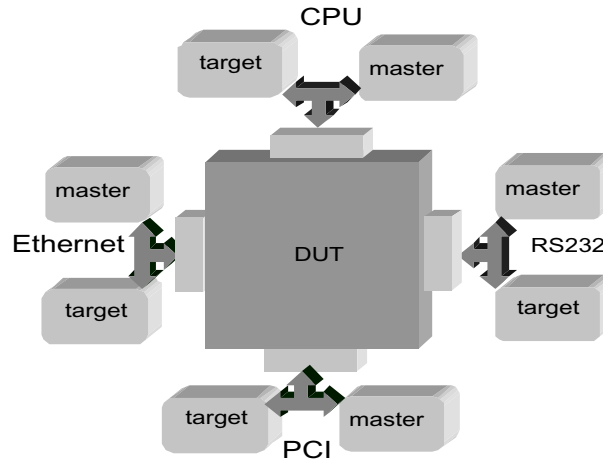
In this document, we describe the criteria to effectively exercise multiprotocol traffic. Based on these criteria we demonstrate how to assert hierarchal constraints in order to reproduce “real world” traffic. Finally, we describe System Scenario Generator (SSG) that implements these hierarchal constraints. The SSG is scalable to support any number of buses of the same type, is expandable to accommodate buses with new protocols, and is adjustable to incrementally shift from a directed test to whatever degree of randomness is required.

1	Introduction	3
1.1	Article Structure	4
2	A Workable Verification approach	4
2.1	Existing Traffic Models	5
3	Hierarchy of Constraints	5
3.1	Budgets and Common Metrics of Data Traffic	7
3.2	System Layer Constraints	9
3.3	Session Layer Constraints	10
3.4	Bus layer constraints	11
3.5	Transaction Layer Constraints	13
3.6	Task layer budget	13
4	Implementation Details	14
4.1	Stack Oriented Structure	14
4.2	SSG Design	15
5	Application results	16
5.1	Testbench structure	16
5.2	Results and Conclusions	17
6	Acknowledgments	17
7	References	18

1.0 Introduction

This article deals with the requirements of verifying a hypothetical system that is represented in Figure 1.

Figure 1. Hypothetic reference system



The system consists of the Design Under Test (DUT) and four end-points. Each end-point can send/receive transactions to/from each end-point through the DUT. All end-points are based on different protocols. The CPU end-point has explicit transaction attributes. The PCI end-point has an implicit transaction size and, as a result, a large variety of the transaction completions. The Ethernet end-point represents a packet oriented protocol. The RS232 end-point deals only with one-size transactions.

The design of the interconnection DUT device for this system must take into account many possible protocol constraints (for example, specific bus ordering rules, possible deadlock situations, heterogeneous nature of a protocol itself, cache coherency, etc.). In this environment, even the implementation of a simple reset strategy is a challenging task (for example, in the “worst case” there are 24 possible reset scenarios).

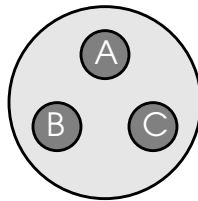
The verification of the DUT along is the challenging task. In order to verify the protocol conversions, we must send a large number of the test vectors. The bus ordering rules force the verification to stress the DUT by using a different series of sequential test vectors. The cache coherency generates additional transactions which propagate in a particular direction. The deadlock situations influences all of the above mentioned parameters.

In a traditional verification approach, the verification of an interconnection system requires the generation of test vectors, which are designed to eventually cover the whole space of the transaction combinations. This traditional method of verification is represented by the large circle in Figure 2. However, the approach is proving to be inefficient and unrealistic with the highly complex devices of today, coupled with increased time-to-market pressures. For example, a

design which includes four end-points, and 16 different transaction types requires at least $(4 \cdot 16)! = 10^{89}$ transactions.

Alternatively, assume that we know the class of applications and their patterns of transaction traffic where the device will be used (see small circles “A”, “B”, and “C” in Figure 2.). We can limit the number of transactions to resemble application-specific combinations. Using this method our verification goal is to achieve 100% coverage inside the areas “A”, “B”, and “C”, and 0% coverage outside these areas. This enables us to efficiently and thoroughly test a device in a shorter time frame.

Figure 2. The space of test vectors



1.1 Article Structure

This article will address the following areas:

- How to make this verification method workable (see “A Workable Verification approach” on page 4).
- We select 5 criteria to describe multiprotocol traffic: data-to-transfer, elapse time, throughput, utilization, and efficiency. We show how to combine these measures in an elementary constraint unit called budget (see “A Workable Verification approach” on page 4).
- We demonstrate how to assert hierarchal constraints in order to reproduce real-world traffic (see “Hierarchy of Constraints” on page 5).
- Our implementation of this approach is the System Scenario Generator (**SSG**) (see “Implementation Details” on page 14).
- We describe how to set up random verification environment using the SSG. We also present our results based on a practical use of the SSG (see “Application results” on page 16).
- We summarize advantages and disadvantages of the SSG (see “Results and Conclusions” on page 17).

2.0 A Workable Verification approach

SoC design dictates an IP-centric verification strategy. In fact, SoC design can be seen as a mesh of individual IP blocks. Its verification can be broken into two phases: block-level and system-level [16]. The focus of a block-level phase is to verify thoroughly every individual IP block using its own test suite. The focus of a system-level phase is twofold: first, to confirm that SoC provides the functionality required in its target application, and second, to demonstrate that the individual IP blocks interact properly with each other.

Directed tests work according to the WYTWYVO principle: What-You-Thought-of-is-What-You-Verify-Only. Random Test, on the other hand, is designed to catch something you did not think of. Because of this we definitely need Random Tests. However, they have to be constrained to resemble application specific patterns of transaction traffic. To satisfy both requirements, we introduce System Scenario Generator (SSG). The SSG is capable to generate random transaction traffic in accordance with user-defined constraints.

We review the existing traffic models prior to approach the SSG architecture (section 2.1). Then, in section 3.0 we discuss user-defined constraints. Finally, in section 4.0, we present the SSG.

2.1 Existing Traffic Models

Internet researchers have faced the same problems when they started to simulate Internet. The most complete problem classification can be found in [1, 2]. Several articles describe network metrics used for modeling and performance analysis [3, 4, 5, 6]. Many of the Internet traffic generators solved multiprotocol interaction problems [7, 8, 9, 10, 11].

The Internet researches are concentrated around two different models: a series analyzing model and a structural model [14, 15]. The series analyzing model is widely used for network engineering. The structural model considers self-similarity nature of network traffic [18]. “The unique feature of structural traffic models is that they are capable of explicitly accounting for the hierarchical nature of today’s network architecture” [14].

3.0 Hierarchy of Constraints

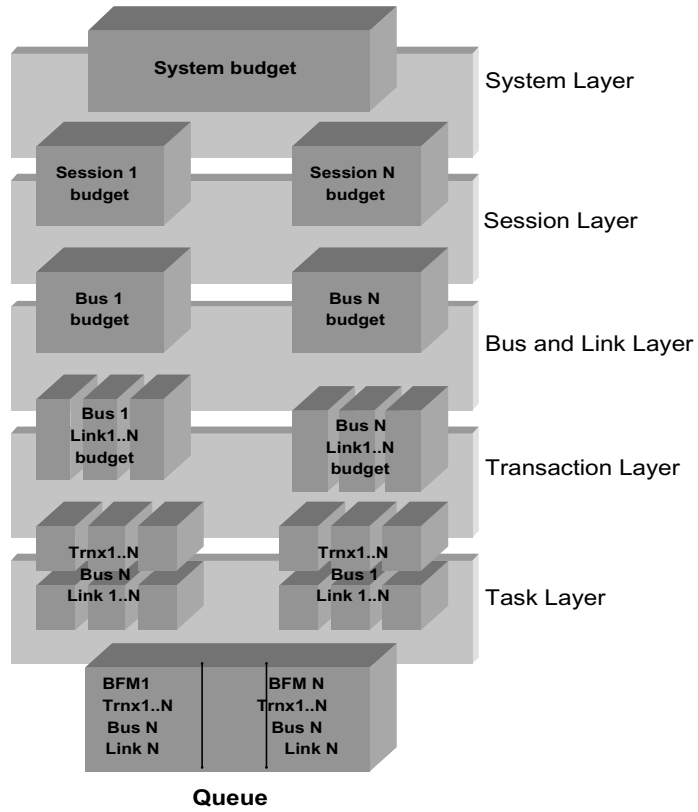
In the interconnection system, traffic from the different sources can be characterized by the following common parameters and measurements: transaction type, transaction size, and transaction route. In the real system, all those parameters are linked together and they reflect the synthetic nature of an interconnection system.

We consider that self-similarity [18] behaves independently on different parts of the system (see “Existing Traffic Models” on page 5). Moreover, traffic parameters may have a different distribution and self-similarity in each system part. With respect to the random generation, it means the VERA built-in unified random generator must be constrained to reproduce the desirable traffic patterns. This consideration, combined with a structural model approach, is the basis for the multi-layered hierarchy of constraints.

The SSG operates with elementary system constraints called *budget*. The budget itself can be considered in different ways. It can be seen as interprocess communication element (described in section 4.2). It can also be seen from a functional point of view. The budget declares two system resources: a data and an elapsed time. We allocate system resources moving budget through the

multi-layered hierarchy of constraints (see Figure 3.).

Figure 3. Budget Breakdown



Each hierarchical layer obtains budget from the upper layer, breaks it according to the constraints, and derives budget for the lower layer. The rules how to break the budget should be defined by the users for each particular layer (see sections 3.2- 3.6).

Section 3.1 defines the budget metrics in terms of the common network criteria.

3.1 Budgets and Common Metrics of Data Traffic

There is a large variety of network traffic characteristics used by the network community, such as locality, burstness, reachability, or capacity. We choose the metrics which can reflect data packets density and data packets size of different protocols. Table 1 on page 7 represents the metrics set used by SSG.

Table 1: Common metrics of a data traffic

#	Name	Units	Function/equation	Comment/Ref
1	Data-To-Transfer (dt)	Byte		Amount of data that should be transferred.
2	Time (elt)	ns		The time during which data should be transferred.
3	Utilization (utl)	%	$utl = \text{busy_cycles} / (\text{busy_cycles} + \text{idle_cycles})$	Utilization is the measure for the relation between used (busy) and unused (idle) bus time in the data traffic.
4	Efficiency (eff)	%	$eff = \text{transferred_bytes} / \text{max_transferred_bytes}$	The efficiency is calculated as the number of <i>actually</i> transferred bytes divided by the number of bytes that <i>theoretically</i> could be transferred within the <i>used</i> clock cycles.
5	Throughput (thr)	KB/s	$thr = \text{transferred_bytes} / \text{elapsed_time}$	The throughput is the amount of data transferred per elapsed time.

For internal representation, the SSG uses only two parameters “dt” and “elt”. However, it is important to give the user an opportunity to specify an input in terms of bus performance measures, such as “utl”, “eff”, and “thr”. The rest of this section is devoted to finding the relation between these terms. Any bus can be characterized by an operational frequency “f” and the width of the data “bus_width” (in bytes).

The maximum possible throughput “max_thr” is:

$$\text{max_thr} = f * \text{bus_width}$$

For any given time “elt”, the amount of data that was actually transferred is:

$$\text{transferred_bytes} = \text{elt} * \text{thr}$$

The amount of data that could, theoretically, be transferred is the number of busy cycles multiplied by “bus_width”:

$$\text{max_transferred_bytes} = (\text{elt} * \text{utl} * f) * \text{bus_width}$$

Therefore, the efficiency can be calculated as

$$\begin{aligned} \text{eff} &= \text{transferred_bytes} / \text{max_transferred_bytes} = \\ &= (\text{elt} * \text{thr}) / (\text{elt} * \text{utl} * f * \text{bus_width}) = \\ &= (\text{thr} / \text{utl}) / \text{max_thr} \end{aligned}$$

There are two ways to provide the throughput to the Generator:

$$\text{thr} \text{ OR } (\text{eff} \text{ AND } \text{utl}) \quad [3.1.1]$$

From the other side, the throughput can be derived in terms of “dtf” and “elt”:

$$\text{thr} = (\text{dtf} / \text{elt})$$

There are three ways to provide “dtf” and “elt” to the Generator:

$$(\text{dtf} \text{ AND } \text{elt}) \text{ OR } (\text{dtf} \text{ AND } \text{thr}) \text{ OR } (\text{elt} \text{ AND } \text{thr}) \quad [3.1.2]$$

The above rules, namely [3.1.1] and [3.1.2], give all five possible combination of parameters required as an input to the Generator (see Table 2:).

Table 2: Possible combinations of traffic metrics

Combination #	Subset of independent parameters		
	1	Data-To-Transfer	Time
2	Data-To-Transfer	Throughput	-
3	Time	Throughput	-
4	Data-To-Transfer	Efficiency	Utilization
5	Time	Efficiency	Utilization

3.2 System Layer Constraints

The system layer is responsible for the common system environment and controls session flow. The session distribution constraints are placed into the session budget table (see Table 3:). It defines the sessions sequence flow and budget allocated to each session (dtt, elt).

Table 3: Test Scenario Breakdown

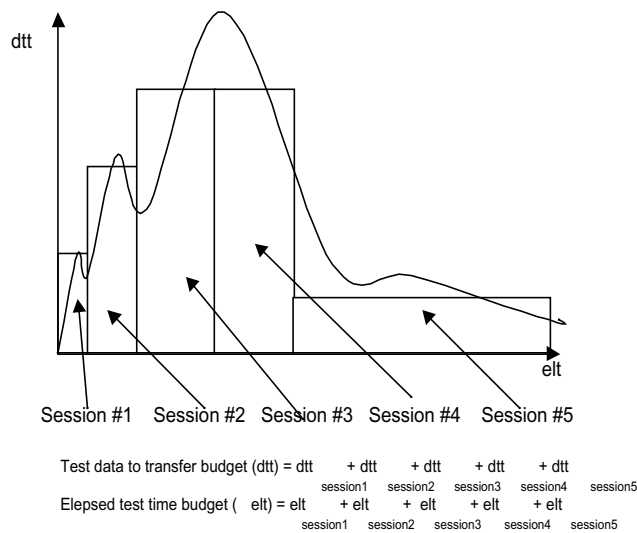
Session Number	Time Budget[%]	Data to transfer budget[%]
1	5	3
2	5	50
3	10	25
4	10	30
5	30	7

System Budget defines the following common test parameters:

- simulation time
- number of bytes should be transferred during the test
- desirable throughput for each testing interface

In terms of the SSG, test is broken down in several sessions. Figure 4. shows an example of the possible test scenario breakdown.

Figure 4. Test Scenario Breakdown



System layer simulates the real application traffic distribution during the system operation time.

3.3 Session Layer Constraints

Session layer is responsible for the session implementation. It obtains the budget per session from the system level as an input and derives the bus level budget as an output. The bus distribution constraints are placed into the bus budget table (see Table 4:).

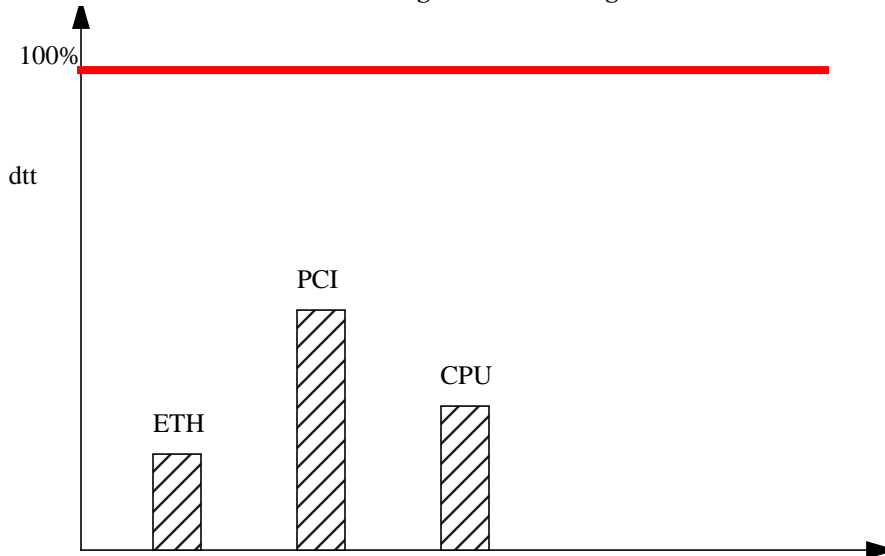
Another part of the session layer activity is to inform system layer when the session budget is expired. The end of the session is function “OR” of two events: budget time expired and budget data to transfer expired. All other layers have same event mechanism.

Table 4: Bus Budget Breakdown

Bus name	Data-To-Transfer %
CPU	30
PCI	50
Ethernet	20

The data in Table 4: is reflected in the histogram in Figure 5.

Figure 5. Bus Budget Breakdown



3.4 Bus layer constraints

The bus layer is responsible for the bus activity. It obtains the budget per bus from the session layer as an input, and derives the transaction level budget as an output. The transaction distribution constraints are placed into the transaction budget tables (see Table 5:, Table 6:, Table 7:):

Table 5: CPU bus Transaction breakdown

Link name	Data-To-Transfer %
CPU_to_PCI_IO	20
CPU_to_PCI_MEM	80

Table 6: RS232 bus Transaction breakdown

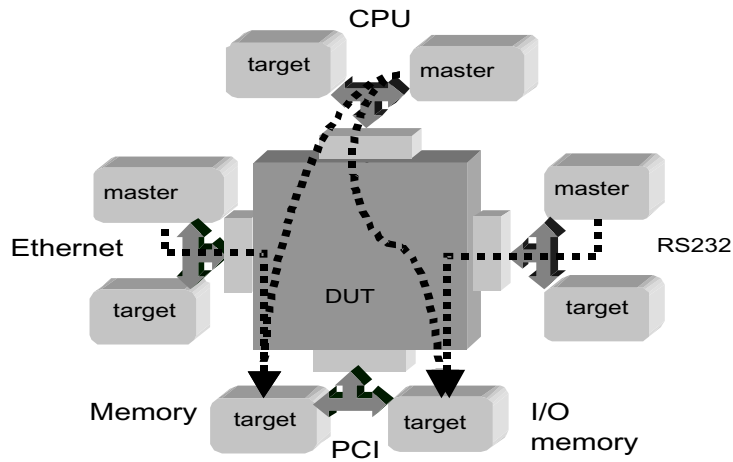
Link name	Data-To-Transfer %
RS232_to_PCI	100

Table 7: Ethernet Transaction breakdown

Link name	Data-To-Transfer %
Ethern_to_PCI	100

Here, the *link name* is the functionally independent data path between one master end-point and one distinguished target end-point. In fact, all possible links for our system are already defined by system memory mapping. Figure 6. shows s how the bus could be divided into link directions.

Figure 6. Routing direction.



In Figure 6. there are two target end-points connected to the PCI bus. The first target end-point is connected to memory space. The second target end-point is connected to I/O space. Therefore, the following list shows possible data paths:

1. From CPU to PCI Memory space
2. From CPU to PCI I/O space
3. From RS232 to PCI I/O space
4. From Ethernet to PCI Memory space

For more information see Table 5:, Table 6:, and Table 7:

3.5 Transaction Layer Constraints

The transaction layer controls the transactions sequential order, and data size. The transaction layer obtains budget per link from the bus level as an input, and derives the task layer budget as an output. Actually, only this layer deals with random variables. Table 8: defines distribution of the transaction size. Table 9: defines transactions types itself.

Table 8: Transaction Size breakdown

Transaction Size (Byte)	Usage (%)
8	20
16	10
32	40
64	30

Table 9: Transaction Type Breakdown

Transaction index	Transaction name	Usage (%)	Link	Size
1	Read	10	CPU_to_PCI_IO	8
2	Write	5	CPU_to_PCI_IO	8
3	Read	10	CPU_to_PCI_MEM	16 32
4	Write	5	CPU_to_PCI_MEM	16 32
5	Read burst	35	CPU_to_PCI_MEM	64
6	Write burst	35	CPU_to_PCI_MEM	64

3.6 Task layer budget

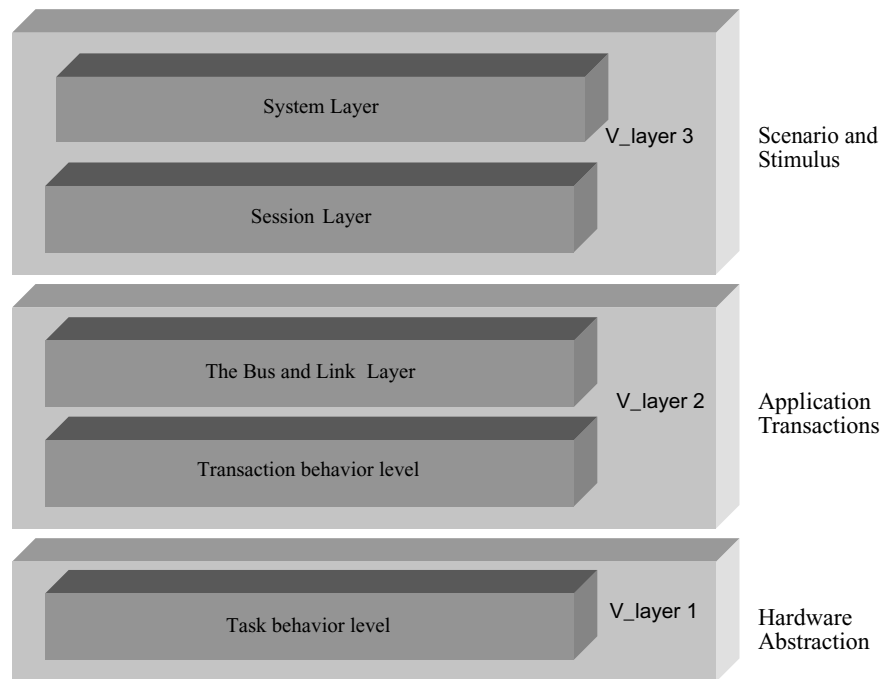
The task level distributes transactions uniformly between bus masters. It obtains the transaction queue as an input, and arbitrates transactions per master.

4.0 Implementation Details

4.1 Stack Oriented Structure

The concept of the verification stack is referred in [12, 13]. However, real verification applications demand more detailed stack structure, in order to clarify stack component dependency and layers interaction. The Figure 7. shows SSG stack organization vs. layered solution proposed by Mohhamed Hawana and Rindert Schutten [13].

Figure 7. The SSG Stack Structure vs. Layered Solution [12, 13].

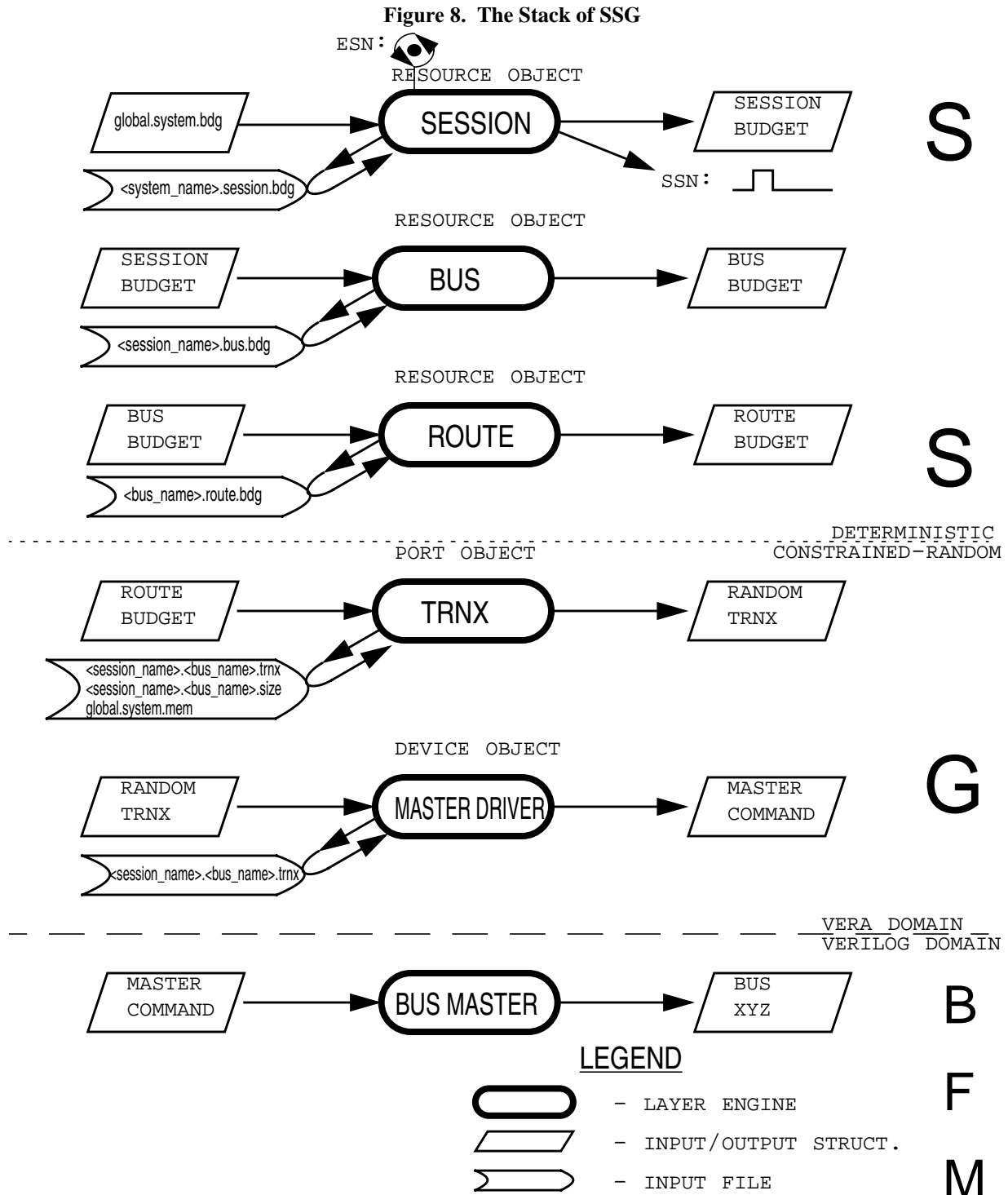


Why use a layered structure? There are several considerations in the favour of a layered structure:

1. Input constraints are organized hierarchically (see section 3.0). Therefore, they can be easily decoupled into the groups corresponding to an established structure of layers.
2. According to an analysis [13] it saves about 25% of an effort during development of verification environment.
3. It is good for a system with concurrent processes with an interprocess communication mechanism [17].

4.2 SSG Design

The hierarchy of constraints has been partitioned (refer to item 1 in the list above) into six layers: Session, Bus, Route, Transaction, Master Driver, and Bus Master. These six layers form the Stack of SSG. It is shown schematically in Figure 8. Comparing Figure 8. with Figure 7. reveals one additional “Bus Master” layer underneath of the SSG Stack. It is associated with a BFM activity.



In Figure 7. each layer has an engine (shown in bold). The layer engine is responsible for processing the input queue and submitting the result to an output queue (refer to the Figure 8.). For instance, the bus layer engine has an input queue and output queue (shown by a straight arrow). It retrieves the session budget structure (shown by a parallelogram) from an input queue. The result of the processing is submitted to an output queue as the bus budget structure. The route layer engine retrieves the bus budget structure from an input queue. The result of the processing is submitted to an output queue as route budget structure. This process continues through the system.

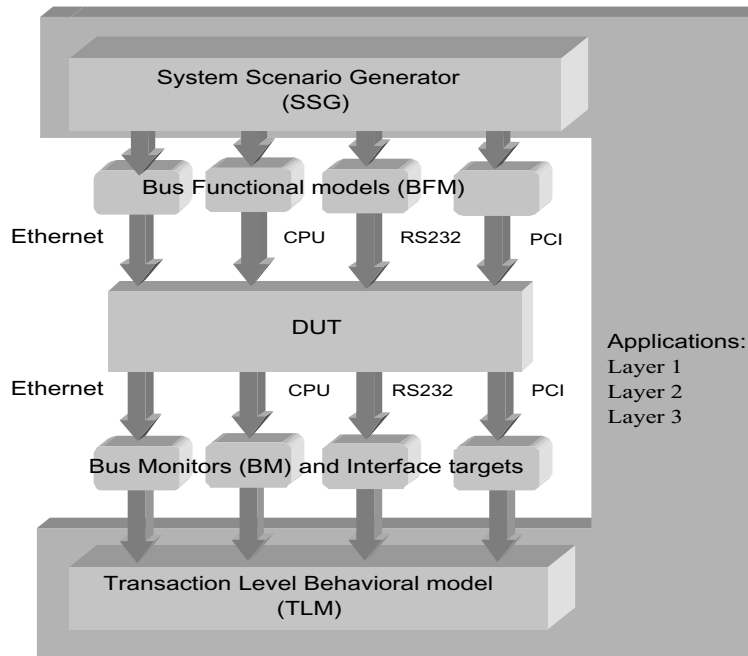
During the processing phase, the layer engine identifies the type and name of an incoming data structure. It uses them to construct the name of an input file. The layer engine reads the constraint parameters from an input file (see Figure 8.).

5.0 Application results

5.1 Testbench structure

Figure 9. shoes the random simulation environment for the hypothetical DUT with four end-points (refer to Figure 1.).

Figure 9. Simulation environment



1. System Scenario Generator
2. Bus Functional Models
3. Transaction Level behavioral Model.
4. Bus Monitors.

The SSG sends transactions towards the end-points. The bus monitors collect the information and delivers it to the VERA domain. Transaction level behavioral models use this information to predict an expected transaction and to compare it with the actual transaction.

5.2 Results and Conclusions

We have used this simulation environment with two systems: one system with three end-points and one system with four end-points. SSG succeeded in sending any type of the traffic distribution (by type, size, or by link). SSG demonstrated a large yield as compared to directed tests. We were able to pass 1,000,000 transactions over 200 hours.

We found the following benefits to this verification system:

1. The SSG is scalable to support any number of buses of the same type.
2. It can be easily adopted for new product development. In fact, an introduction of a new protocol requires about two weeks (including debug phase) to make SSG compatible with an acquired bus functional model. The SSG is expandable to accommodate buses with new protocols.
3. The SSG is adjustable to incrementally shift from a directed test to any required degree of randomness. We plan on applying some directed tests using SSG in the future.

However, there are the following limitations:

1. System configuration is difficult according to the system memory map; it requires a large effort.
2. The management of the input tables becomes impossible when there are more than 10 to 15 end-points. However, this limitation could be overcome by elimination of manual table writing.

In conclusion, the SSG system saved a large amount of time in verification and makes it possible to succeed in delivering well-tested products to market in a timely manner. Also, because of this system we are now much more confident in the verification of our designs.

6.0 Acknowledgments

We are grateful to all members of the Tundra System Verification Group for their support and positive attitude. Special thanks to Mr. Plouffe for his numerous discussions, suggestions, and remarks. Special thanks to Ms. Rhyno for the proofreading.

We are thankful to the management of Tundra Semiconductor for allowing us to make this publication.

We are deeply grateful to our families for putting up with late night writing sessions, and long absences while we were writing this article.

7.0 References

- [1] Vern Paxson and Sally Floyd. 1997. Why We Don't Know How To Simulate The Internet Network Research Group, Lawrence Berkeley National Laboratory University of California, Berkeley 94720, U.S.A. LBNL-41196
- [2] Sally Floyd and Vern Paxson (2001) "Difficulties in simulating the Internet". AT&T Center for Internet Research at ICSI (ACIRI) Berkeley, CA 94704 USA
- [3] Kimberly C. Claffy 1994. Internet traffic characterization. Ph.D. dissertation. UNIVERSITY OF CALIFORNIA, SAN DIEGO.
- [4] Marc F. Pucci 2002. Combining Multilayer, Topological and Configuration Information in Performance Analysis. Passive and Active Measurement Workshop.
- [5] Vern Paxson. 1996. Towards a Framework for Defining Internet Performance Metrics. Network Research Group, Lawrence National Laboratory.
- [6] Kun-chan Lan, John Heidemann . 2002. Multi-scale validation of Structural Models of Audio Traffic. Information Sciences Institute, University of Southern California.
- [7] The list of Traffic Generators. <http://www.caip.rutgers.edu/~arni/linux/tg1.html>
- [8] Netspec. Experimental testing of the network performance. <http://www.ittc.ku.edu/netspec/>
- [9] Packet Shell. The Protocol Testing Tool. <http://playground.sun.com/psh/>
- [10] Rude/Crude. Real-time UDP Data Emitter and Collector. <http://rude.sourceforge.net/>
- [11] MGEN. Multicast/unicast UDP/IP traffic emitter. <http://manimac.itd.nrl.navy.mil/MGEN/>
- [12] "A Layered Verification Approach for AMBA-Based SoC Using OpenVera" in verification avenue, issue 4. Synopsys 2002.
- [13] "Testbench Design, a Systematic approach" in verification avenue, issue 3. Synopsys 2002.
- [14] Walter Willinger and Vern Paxson. Discussion of "Heavy Tail Modeling and Teletraffic Data" by S.R. Reznik. AT&T Labs ñ Research and Lawrence Berkeley National Laboratory.
- [15] Kun-Chan Lan. Rapid generation of structural model from network measurement. 2002. Ph.D. dissertation proposal. Department of Computer science USC.
- [16] For a more detailed description, see section 3.1 in "Pseudo Random Test development for SoC Verification Using VERA" by Denis Parcheminal, Giuseppe Bonanno. SNUG Europe 2001.
- [17] See section 13.2.5 in "Software Development with C++" by Kjell Nielsen. Academic Press

1995, Cambridge, MA.

[18] W. Willinger, V. Paxson and M. S. Taqqu. “Self-similarity and heavy tails: Structural modeling of network traffic” in *A Practical Guide to Heavy Tails: Statistical Techniques for Analyzing Heavy Tailed Distributions* (eds. R. Adler, R. Feldman and M. S. Taqqu), Birkhauser Verlag, Boston 1998.