

Intelligent Interleaving of Scenarios: A Novel Approach to System Level Test Generation

Shady Copty, Itai Jaeger, Yoav Katz, Michael Vinov

IBM Research Laboratory in Haifa
Haifa University Campus, Mount Carmel
Haifa 31905, Israel
{shady,jaeger,katz,vinov}@il.ibm.com

ABSTRACT

Parallelism in system architecture and design imposes a verification challenge as the exponential growth in the number of execution combinations becomes unwieldy. We report on a method for system-level test case generation. This method relies on dynamic interleaving of scenarios from the core level or sub-system level. We discuss the relevance of this method for the system level. We also describe a tool that implements this method and show how it was used in IBM for system verification of the Xbox 360 chip and Power Management in the Cell processor, as well as verification of the pSeries eServers. We claim that this method shortened the system level verification cycle and allowed reuse in and across projects, which led to exposure of system-level bugs in a relatively short time.

Categories and Subject Descriptors

B.6.3 [Logic Design]: Design Aids—*Verification*

General Terms

Verification, Measurement, Experimentation

Keywords

Functional verification, System verification, Test generation

1. INTRODUCTION

Demands for higher performance with limited power consumption is squeezing computer designs unwillingly to architectures that utilize parallelism. This is evident in desktop computing, as single processor desktops are becoming a thing of the past [3, 12]. We observe similar trends in servers, as clusters are relying on intricate parallel protocols [11, 4], and certainly in the realm of super computing [9], as the abundance of processors is reaching the tens of thousands.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2007, June 4–8, 2007, San Diego, California, USA.

Copyright 2007 ACM 978-1-59593-627-1/07/0006 ...\$5.00.

Although its hard to argue with demands for performance, one stands puzzled at the complexity these architectures bring to the efforts of design and verification. This stems mainly from the exponential growth of execution combinations.

Verification of hardware designs is usually done hierarchally, from block to unit, core, chip and all the way to system-level. The major challenge posed by the multiplicity of processors and complication of IO protocols is at the system level. This is because higher levels of parallelism have been introduced at the system level.

This paper presents an approach for off-line generation of test-cases for system-level verification of parallel computer systems. We describe the method of dynamically interleaving test scenarios for various parts of a system, to verify the system as a whole. Test scenarios usually concern a certain aspect of the system; interleaving two such scenarios would typically exercise both aspects. We show how dynamic interleaving by an intelligent test generator can also exercise the interaction between those aspects. This method allows easy reuse of scenarios from lower levels of verification for the system-level, and across different projects that share certain components.

The paper continues as follows: Section 2 discusses the relevance of this approach to the verification of multiprocessor computer systems in light of their unique characteristics. Section 3 describes a tool that supports this approach. Section 4 describes the use of this method for verification of IBM pSeries eServers and the Cell processor system as well as verification of the Xbox 360 chip. We conclude with a summary section.

2. IMPORTANCE OF SCENARIOS INTERLEAVING FOR SYSTEM-LEVEL VERIFICATION

The objective of system-level verification is to verify the integration of previously verified components [2]. The verification plan must identify the major risk areas of integration and devise scenarios to trigger errors in these areas. To capture the essence of the system these scenarios typically include interactions between cores.

Random interleaving of scenarios is one method of creating interesting system-level scenarios that cause the cores to interact. This is not possible in all systems; DVD systems, for example, define a pipeline workflow with strict ordering of data transfers between cores. Such systems typically employ a single controller core and a specific software model. Verification of these systems requires creating a system-level scenario that takes into account the

system level workflow [5, 14]. Some random variance is allowed in the scenarios, but random interactions between the cores are not within the scope of the legal system behavior.

Other systems, such as general purpose computer systems, are more decoupled, and allow a large variance in the interactions between the cores. The cores interact asynchronously. There is no single core that controls the behavior of all other cores. Such systems typically support a large number of system and sub-system configurations. It is important to verify the conjunction of functionalities of the different cores, since errors are usually triggered from a specific interaction between the cores. Some errors can only be exposed if the cores interact locally in time and space, that is, if the interaction involves utilizing the same resources at the same time.

As an example, consider an SMP server, as depicted in Figure 1. The system is composed of several processor cores connected to memory via a central processor local bus (PLB). IO devices connect to a peripheral bus (PHB) and the PHB is connected to the PLB by an IO bridge. In this system, there are various interactions that can take place in parallel and asynchronously. For example, one processor can access memory, while a second processor reads a memory mapped register of an IO device and a second IO device performs a Direct Memory Access (DMA) to memory. Executing all these interactions simultaneously is interesting, since they collide on a shared resource (PLB bus). Making the target DMA address the same as the target of the memory access of the processor will increase the quality of the scenario, because it will exercise additional cache and memory logic.

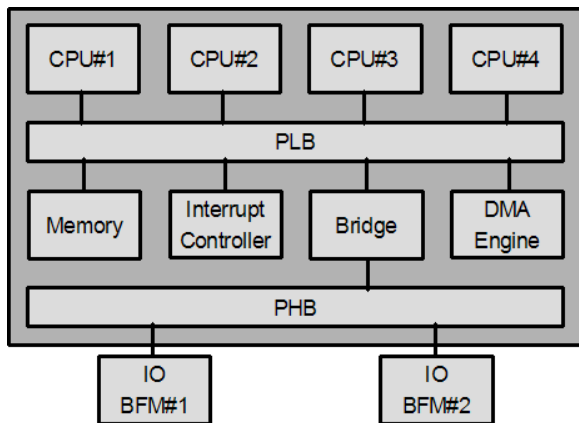


Figure 1: SMP system

The transaction-based verification (TBV) approach is a common methodology that supports specification of system-level scenarios [6, 7, 15]. The test specification is expressed as a series of abstract system-level transactions. Examples of such transactions are CPU-to-memory data transfers, IO initiated interrupts, and DMA transfers. Each transaction has a set of parameters that can be specified, for example, the source and target address of a DMA transaction. The transactions are converted to the actual stimuli by transactor code, which maps the abstract transaction into the low level signals or resource initialization. Most verification environments support the TBV approach. The verification engineer can force resource contention within a scenario by restricting the parameters of the different transactions in the scenario.

Consider the scenarios in Figure 2. The scenario on the left attempts to trigger interesting memory subsystem events by forcing one processor to repeatedly write to a certain address, while a sec-

ond processor repeatedly reads from the same address. The scenario on the right stresses the IO interconnect by causing repeated memory mapped IO, DMAs, and interrupts between a processor and an IO device.

Memory subsystem scenario	IO subsystem scenario
P ← Choose processor	I ← Choose IO adapter
Repeat 100	P ← Choose processor
P write to address X	Repeat 50
P2 ← Choose processor	P performs MMIO to I
Repeat 100	I performs DMA
P2 reads from address X	I interrupts P

Figure 2: Two system level scenarios

Each scenario has value in itself, but a scenario that generates interleaved stimuli for both scenarios is very interesting in the system, especially if it collides on resources. Collision can occur when common addresses are selected in the scenarios. This happens for example, when the DMA address of the IO scenario is selected as equal to the address accessed by the processors in the memory subsystem scenario. Resource contention can also occur if we choose the same processor to participate in both scenarios.

Intelligent interleaving of scenarios is not a simple task. In on-line testbench generation environments that support multi-threading [10, 17] it is possible to run each scenario in a separate thread, and thus create simultaneous stimuli. If the verification environment does not support multi-threading, then the remaining alternative is to run the two scenarios sequentially. However, neither solution will generate resource contentions between the scenarios in the different threads, unless global information is maintained and communicated between the transactors (for example, a set of shared addresses). This requires all transactor code to be aware and use these global mechanisms.

In offline test generation solutions, there are two approaches for interleaving. The first approach is to interleave the output, that is, generate each scenario separately and combine the output tests. The integration is a costly process that is highly sensitive to changes in the simulation environment and test format. In addition, there is no ability to create intentional resource collisions between the two scenarios. A second approach is to interleave the input, creating a combined scenario by interleaving the input test specifications. It is extremely difficult to statically preprocess the input test templates and combine them into a legal interleaved test template. A simpler solution is to concatenate the input test specifications. However, the different scenarios would be generated sequentially and not interleaved in one another. This has a negative side effect, in that for a specific core, all the transactions it participates in from the first scenario will be generated (and often executed) before all its transactions from the second scenario.

The following section describes a method for dynamic interleaving of the scenarios while using a common generation state between scenarios to create resource contentions.

3. SCENARIO INTERLEAVING SUPPORT IN X-GEN

3.1 An Overview of X-Gen

X-Gen is a test generation technology for computer systems and SoCs [8]. Since 2002, X-Gen has been the primary stimuli generator for functional verification of most of IBM's high-end systems,

including all the pSeries eServers and the Cell processor-based systems [18, 16].

X-Gen adheres to the model-based paradigm introduced in [1]. The generator is partitioned into two separate components: a generic engine, which is capable of generating test-cases for a variety of computer systems; and a knowledge base, which includes an abstract system model and the expert testing knowledge.

The high level architecture of the X-Gen tool is shown in Figure 3. The abstract system model describes the system’s components and its transactions. Testing knowledge is the verification expertise which biases test generation towards bug-prone areas; it does not require explicit input from the user. Examples of testing knowledge building blocks provided by X-Gen include: (1) a collision mechanism that biases the test-case towards the reuse of certain system resources (e.g., many accesses to the same cache-line); (2) a placement mechanism that biases memory accesses towards events such as cross-page (an access that begins in one page and ends in another) or segment boundary.

The topology of the system under test is described in the configuration file, while the verification scenario is defined in a test template file. The scenario definition is done using a proprietary special-purpose programming language developed for this task. The language allows users to define scenarios ranging from fully deterministic to totally random.

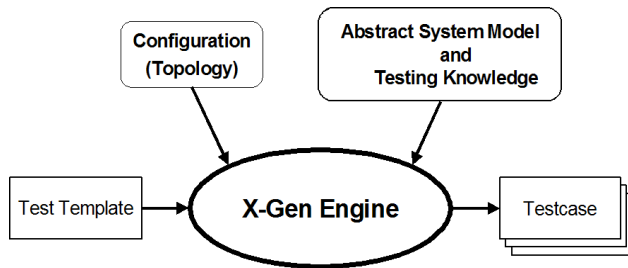


Figure 3: X-Gen: High-level Description

3.2 Interleaving Interface in X-Gen

X-Gen supports the dynamic interleaving of test templates. It generates test cases that include transactions from all supplied templates. Users have control over how test templates are interleaved, ranging from simple template concatenation to real transaction-level interleaving.

The tool accepts a list of test templates to be interleaved. For example, Figure 4 describes a generation of a test case from two templates: Template_A and Template_B. Template_A requires creating a sequence of 100 CPU-to-memory accesses, while Template_B requires creating a sequence of 50 IO initiated DMA accesses. The resulting test case includes transactions from both templates.

Users can specify a relative weight for each of the input templates. X-Gen generates transactions according to the assigned weights ratio. To comply with the requested weights, some of the templates may be generated multiple times. To keep the semantics of the scenarios specified in the templates intact, each template would be generated an integer number of times, that is, the generator would never generate a partial test template. The first test template is defined as a baseline and is generated exactly once.

In some cases, test templates contain sequences that cannot tolerate the intervention of foreign transactions during their course. In these cases, the corresponding part of the test template should be marked as unbreakable. The test case generator will not generate any transaction from another template until the generation of

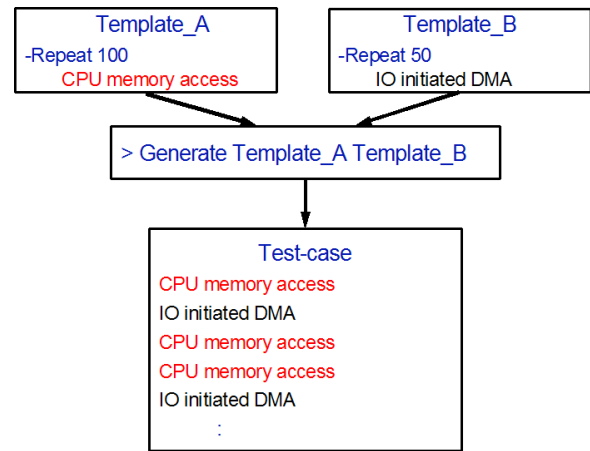


Figure 4: Interleaving output

an unbreakable section completes. An example of an unbreakable section might be a series of accesses to consecutive addresses in memory; the verification value of this scenario would decrease if a transaction from another template accesses an unrelated address during the series.

The tool provides more refined control over the interleaving patterns. Users can constrain specific areas in the scenario where the amount of foreign transactions injected into the scenario through interleaving is limited to a certain range. This is in addition to the weights mentioned above, which control the global progress pace for each scenario. In section 4, we provide an example where this control is needed to interleave power management scenarios with other scenarios. Additionally, users may define sections of a certain test template to be generated at the very beginning or very end of the combined test case, in case the template is being repeated, by marking them as Prologue or Epilogue accordingly.

It is important to note that the testcase generator holds global knowledge on the state of the design and generation process resources. This knowledge is leveraged to create events that involve more than one test template at a time. A powerful example of an inter test template event is generating address collision between CPU accesses to memory coming from one template and DMAs that also target the same memory coming from another template.

3.3 Algorithm

Upon activation of the generator, it starts generating transactions according to one of the test-templates. After every transaction, an arbiter decides whether to remain with the current test template or leap to another test template. This decision is affected by the templates weights and any unbreakable marking. The generation of each transaction is affected by the testing knowledge, which uses the shared internal state of the generator. The shared data is updated after the generation of each transaction. All the generated transactions are redirected into a single test case. The result is a test case that complies with each of the input test templates. A pseudo-code of this algorithm is shown in Figure 5.

4. REAL LIFE EXPERIENCE IN IBM

4.1 Power Management Verification of the Cell

The interleaving capability of X-Gen was used in the verification of the power management features of the Cell Broadband Engine (Cell BE) [16]. Cell BE (see Figure 6) is a heterogeneous chip mul-

```

S ← GetAllTemplates()
baseline_template ← GetBaseline(S)
While !Completed(baseline_template)
  if IsBreakable(current_template) or !Empty(S) then
    current_template ← WeightedChoice(S)
  GenerateSingleTransaction(current_template)
  if Completed(baseline_template) and
    Completed(current_template) then
    S ← S - current_template
  UpdateSharedTestingKnowledgeData()
  WriteTransactionToTestCase()

```

Figure 5: Interleaving algorithm

tiprocessor that consists of an IBM 64-bit Power Architecture core, augmented with eight specialized co-processors based on a novel single-instruction multiple-data (SIMD) architecture called Synergistic Processor Unit (SPU), which is for data-intensive processing. The system is integrated by a coherent on-chip bus. The SPU accelerators operate from a local storage that contains instructions and data for a single SPU. Memory access is performed via a DMA-based interface using copy-in/copy-out semantics to and from the local storage. The chip also contains an IO bridge to external devices. The architecture allows concurrent transactions between the different cores: DMA transfers between the local stores and memory, memory mapped access by the PowerPC processor, IO devices, DMA, and interrupts.

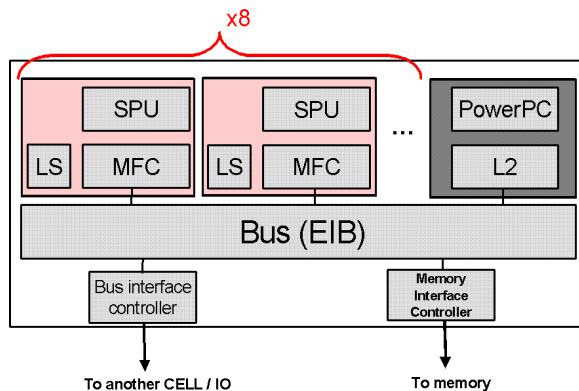


Figure 6: Cell BE Architecture

The architecture also supports various states of power management for the processor cores. These states are invoked by code sequences running on the processors. One power management scenario is for the PowerPC to halt an SPE and restart it later in simulation. Another scenario is to slow the chip clock frequencies and then resume to full speed.

It is desirable that these power management sequences be mixed with other scenarios. To achieve this, a short test template was created that selects a single SPU, shuts it down, and then restarts it (see Figure 7). Because interleaving this test template in other scenarios can cause the test case simulation time to increase considerably, the test template specified a maximum number of interactions to interleave between the shut down and the restart sequences. Also, an interleaving ratio between the test templates was selected to limit the number of invocations of the sequence. During the generation of the power save scenario, X-Gen applies its built-in testing knowledge to select an SPE that participates in the interleaved scenario.

```

S ← Choose an SPE
PowerPC MMIOs to shut down S
(min transactions = 5 , max transaction = 10)
PowerPC MMIOs to restart S

```

Figure 7: Cell Power Saving Scenario

4.2 Combining IO and MP Testing for the Xbox Chip Verification

Another application of interleaving support was employed in the chip-level verification of the Xbox processor chip. The Xbox chip architecture consists of three symmetric dual-threaded PowerPC cores. The processors are connected via a cross bar and the chip interfaces the graphic processing unit and IO using a front side bus.

In this case, a large set of multi-processor and multi-threaded PowerPC oriented test templates, originally written for PowerPC based servers verification, were ported to the Xbox verification environment. Using X-Gen's interleaving support, these templates were then mixed with newly written templates that address the dedicated IO-subsystem of the Xbox chip. As a result, test-cases that combined highly-complex MP/MT scenarios with the unique IO activity of the chip were generated. These test-cases demonstrated additional testing knowledge, such as resource contention between the PowerPC cores and the IO devices. In summary, the simulation environment was capable of generating very complex test cases combining IO and MP activity within a record time of several days and with minimal additional development effort. As a result of this effort several chip level bugs were exposed.

4.3 Interleaving as Part of System-level Verification Methodology of eServer

Perhaps the most extensive application of the interleaving feature was demonstrated in the deployment of a new system-level verification methodology and maintenance practice for the i/pSeries eServers verification. The IBM i/pSeries systems support a wide variety of configurations ranging from entry level single core machines to enterprise level 64-core machines [13]. The IO capabilities of the systems also vary and include both standard PCI interface and high bandwidth low latency interconnect such as InfiniBand adapters.

The system verification organization is faced with the constant challenge of creating and implementing a verification plan for the large number of system configurations. In addition, the verification team is often required to verify several generations of the systems concurrently.

A common drawback in previous attempts to deal with these challenges was that changes to an existing core (or function) or the addition of a new one immediately called for an update to the lion's share of existing templates.

As part of the new methodology, a separate set of scenarios were created for each function (PCI, Infiniband, Multi processing). These scenarios are then interleaved, eliminating the need to manually modify test scenarios in order to check the conjunction of functionalities. The method has already been successfully deployed in two large-scale projects.

5. SUMMARY

In this paper we discussed the importance of interleaving scenarios for the verification of hardware designs. This is especially relevant for system-level verification of multi-processor computer systems. We described a method and an algorithm for creating highly coordinated system level test-cases, which was implemented in the

X-Gen system-level stimuli generator. The tool was used as the primary system-level test generation technology for verification of all IBM pSeries computer systems and the Cell processor system. We also described the application of this method for power management verification in the Cell processor, and in the chip verification of the Xbox 360 chip, where this method allowed the exposure of several chip-level bugs in a relatively short time.

6. REFERENCES

- [1] A. Aharon, Y. Lichtenstein, and Y. Malka. Model-based test generator for processor design verification. In *Innovative Applications of Artificial Intelligence (IAAI)*, 1994.
- [2] K. Albin. Nuts and bolts of core and SoC verification. In *38th Design Automation Conference (DAC '01)*, 2001.
- [3] AMD. Amd multi-core. <http://mutlicore.amd.com/>.
- [4] I. T. Association. <http://www.infinibandta.org/>.
- [5] G. Berry, A. Bouali, J. Dormoy, and L. Blanc. Top-level validation of system-on-chip in estereel studio. In *Proceedings of the IEEE International High Level Design Validation and Test Workshop*, 2001.
- [6] S. D. Besyakov V. Constrained random test environment for SoC verification using vera. SNUG Boston, 2002.
- [7] Brahme et. al. The transaction-based verification methodology. Cadence Berkeley Labs Technical Report CDNL-TR-2000-0825, August 2000, 2000.
- [8] R. Emek, I. Jaeger, Y. Naveh, G. Bergman, G. Aloni, Y. Katz, M. Farkash, I. Dozoretz, and A. Goldin. X-Gen: A random test-case generator for systems and SoCs. In *Seventh IEEE International High-Level Design Validation and Test Workshop, HLDVT-02*, pages 145–150, 2002.
- [9] Gara et. al. . Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2):195, 2005.
- [10] F. Haque, J. Michelson, and K. Khan. *The Art of Verification with Vera*. Verification Central, 2001.
- [11] IBM. IBM system Cluster 1600. <http://www-03.ibm.com/systems/clusters/hardware/1600.html>.
- [12] Intel. Intel multi-core. <http://www.intel.com/multi-core/>.
- [13] International Business Machines. pSeries systems. <http://www-03.ibm.com/systems/p/>.
- [14] A. Nahir, A. Ziv, R. Emek, N. Ronen, and T. Keidar. Scheduling-based test-case generation for verification of multimedia SoCs. In *43th Design Automation Conference (DAC '06)*, 2006.
- [15] P. Rashinkar, P. Paterson, and L. S. L. *System-on-a-chip Verification Methodology and Techniques*.
- [16] K. Shimizu, S. Gupta, T. Koyama, T. Omizo, J. Abdulhafiz, L. McConville, and T. Swanson. Verification of the cell broadband engine processor. In *43th Design Automation Conference (DAC '06)*, 2006.
- [17] Verisity Design. Spec-based verification. <http://www.verisity.com/resources/whitepaper/specbased.html>.
- [18] D. W. Victor, J. M. Ludden, R. D. Peterson, B. S. Nelson, W. K. Sharp, J. K. Hsu, B.-L. Chu, M. L. Behm, R. M. Gott, A. D. Romonosky, and S. R. Farago. Functional verification of the POWER5 microprocessor and POWER5 multiprocessor systems. *IBM Journal of Research and Development*, 49(4):195, 2005.