

The Evolution of Processor Test Generation Technology

Eric Hennenhoefler
eric@obsidiansoft.com

Melanie Typaldos
melanie@obsidiansoft.com

Obsidian Software Inc.
www.obsidiansoft.com
August, 2008

Abstract

Random test generation (RTG) technology is the backbone of modern processor functional verification strategies. These programs create pseudo-random assembly level tests based on some level of user preferences. The resulting tests are used throughout the verification process from early RTL bring-up to the final steps of massive regression and sometimes even in post silicon hardware testing.

This paper provides an overview of the evolution of RTG across multiple generations of test generation technology including table-based generators, static test generators, dynamic test generators, knowledge based generators, and commercial grade knowledge based test generation systems. Also outlined are several usage models to help determine the right technology or combination of technologies for a given project based on the complexity of the verification challenge and life expectancy of the processor architecture.

Introduction

Random test generators are the backbone of modern functional verification methodologies for processors. Recent papers claim to have implemented random test generators in as little as a few weeks [1], whereas, experts in the field spend millions [2] and employ large research groups with the charter to maintain and enhance RTG technology. In reality, both approaches produce test generators but the level of robustness and the types of end users vary radically between these cases.

This paper provides an overview of various test

generation technologies available for creating pseudo random tests for the functional verification of processors. Each technology or method will be analyzed based on creation costs and the ability of the test generator to meet the needs of modern processor development teams. The paper also presents a methodology for determining the optimal technology for a new processor based on the complexity of the ISA and the implementation.

Finally an overview of the necessary technology and process needed to deploy Obsidian Software's commercial test generator, RAVEN®, will be reviewed.

Motivation

The goal of verification teams is to achieve bug free first silicon on schedule. The cost of failure is extremely high due to the high costs of mask sets and the loss of market window implied in a missed schedule.

A typical processor pre-silicon functional verification process involves running a large number of assembly level tests in RTL simulation. The more random tests that are run in RTL pre-silicon, the greater the chance the DV team has of finding all the bugs. The concept of massive regression combined with automated results checking and coverage results is the foundation of modern processor verification.

There are two primary problem spaces to which RTG can be applied. The first is enabling the massive regression system and the second is providing a way of building on existing knowledge to increase the productivity of the stimulus creation process.

Taxonomy of Test Generators

There is a vast landscape of test generators used in the industry today. These range from simple scripts and parameterized macros that can be created in a matter of weeks to full featured systems used by cutting edge processor verification teams. The following sections will provide an overview of the major types of generators. This outline will be presented in historical ordering. Table 1 provides a summary of existing RTG technologies.

Table 1: Overview of Verification Methods

Technology	Description
Directed ASM tests	DV engineers write tests by hand.
Table-based Generators	Simple tables of macros, instructions, and operands mixed with random parameters.
Static Generators	Tables are combined with complex procedural code.
Dynamic Generators	Uses the state of the ISA to create more robust tests.
Knowledge-based Generators	Dynamic test generators that allow testing knowledge capture for future projects. Multiple solvers are used to handle complex constraints and pipeline allocation. Typically GUI front end if usage extends beyond author.
Commercial Test Generators	Complex, comprehensive test generators available from 3rd parties

In many cases, a processor design team will select a simple test generator for the first project and gradually evolve it into a more advanced form. This evolution stems from several causes. The verification effort may be underestimated or minimized during the justification phase of a new product. During development of later revisions, the

verification phase estimate can be more realistic since it is based on knowledge gained in earlier revisions. Earlier designs tend to be simpler in structure with later designs adding more features and more complexity. Simple designs can often be verified using less sophisticated technology while verification of later designs may prove to be too complex for the simple approach. Products that go through several revisions and enhancements are likely to be those that have proven successful in the market and these tend to have better funding for both design and verification.

Directed ASM Tests

Description

Hand crafted assembly language tests.

Pros

- Easy to construct
- Simple to debug
- Requires DV engineers to learn the ISA

Cons

- Very time consuming
- Requires all DV engineers to learn the ISA and the software development environment
- Can be difficult to coordinate test creation to ensure coverage, especially of corner cases.
- Engineers' knowledge of the design can bias test creation.

Usage

- New ISA features
- Early bring up
- DV engineer training
- Targets known holes in test generation

Enables massive test generation	No
Enables knowledge capture	Very low
Coverage productivity gain	None

Table based Generators

Description

A simple test generator constructed from data tables. These generators have little to no logic in them; everything is in the tables or macros. Examples are UMA DGL macro generator and Specman CPU examples.

Pros

- Can be developed in about a month
- Can achieve high encoding coverage of very regular data path instructions
- Tables and macros are easy to understand without requiring a full knowledge of the ISA

Cons

- Quickly breaks down in complex ISA areas
- Macros have low randomness
- Large numbers of macros may be required to hit interesting corner cases.

Usage

Table based generators are a temporary solution until more robust generators are available

Enables massive test generation	No, instruction sequences are too regular.
Enables knowledge capture	Low, tables do not scale.
Coverage productivity gain	Very low.

Static Generators

Description

Static generators are similar to table based generators with the difference being that the majority of the instruction, operand, and data selection is now in complex procedural code

Pros

- Increased randomness over table based generators
- Better support for control flow instructions

Cons

- Lacks state information
- Insertion of helper instructions decreases randomness
- Macros are still required to reach difficult cases

Usage

Medium complexity projects along with directed tests

Enables massive test generation	Moderate, scales better on simple ISAs
Enables knowledge capture	Low, source code modifications required
Coverage productivity gain	Moderate

Dynamic Generators

Description

Test generator that uses the current state of the processor.

Pros

- Thorough test generation for complex ISAs
- Creates dense, interesting tests

Cons

- Slower test generation
- Significant engineer investment to create

Usage

- Medium to high complexity designs.
- Some directed tests still required.

Enables massive test generation	Moderate to high
Enables knowledge capture	Moderate
Coverage productivity gain	Moderate

Knowledge-based Generators

Description

Dynamic test generators combined with advanced constraint and pipeline solvers. These generators separate generic testing knowledge from ISA specific information. The ISA specific fraction of the test generator can be enhanced or swapped for use in future projects

Pros

Generic, reusable, constraint solvers

Scalable testing knowledge

Pipeline resource solvers

Cons

Development costs are high, comparable to a compiler development

Usage

Knowledge-based generators are capable of providing high-quality verification tests for all designs. Usage is often decreased if the tool lacks documentation, user interfaces, and an EDA support team.

Enables massive test generation	High
Enables knowledge capture	High
Coverage productivity gain	High

Commercial Test Generators

Description

Complex, comprehensive generators that abstract as much testing knowledge as possible into reusable cores. ISA specific information is added on top of a proven base

Pros

The reusable core of the generator has been used on multiple projects and is fully debugged

A full-featured test generator is available early since only the ISA specific layer needs to be added

The vendor provides a tool development team that is experienced with multiple architectures and

verification strategies as well as with the development of the tool itself

A full-featured, user friendly GUI usually provided

User manuals explaining the complete functionality of the tool are provided

Vendor supplies support personnel for learning and debugging

Improvements made to the core for other architectures or customers are included in updates

Cons

In a large company, the internal tools development group may resent the use of an outside vendor

Knowledge of the ISA must be made available to the tool developer

Acceptance and deployment to multiple groups may be an issue.

Usage

Commercial test generators are full featured generators that provide tests for designs of all levels. The verification phase of the design is expedited since the tool is available early in development

Enables massive test generation	High
Enables knowledge capture	High
Coverage productivity gain	High

Determining the Best Technology for a Project

The choice of the correct verification tool for a new design project will depend on many factors, among which are:

- The level of experience of the design and verification teams
- The complexity of the ISA / processor
- The project schedule
- The tools currently in use within the company
- The level of staffing for verification, design and tool development
- Funding available for verification

Several areas of microprocessor design are

increasing the complexity of the average design project. Microprocessors themselves have become increasingly complex, incorporating advanced memory management units, floating point, multimedia instructions, SIMD, VLIW and multi-tasking / multi-processing. In addition, processors are tending toward becoming microcontrollers or SoCs with the inclusion of DSP, DMA and other functions previously relegated to board components.

The more complex the design, the more complex and comprehensive tool is required for verification. Even on simpler designs, it may be advantageous to select a more thorough verification approach to avoid having to revamp or recreate a verification environment for potentially more complex follow-on projects.

Table 1 provides an overview of the functions provided by each type of generator, directed ASM, table-based, static, dynamic, knowledge-based and commercial. A comparison of the required testing parameters may be used along with this table may help determine the correct tool.

The RAVEN® Generator

For many projects, the complexity of the design and the need for fast verification preclude the development of an RTG from scratch. Design teams may attempt to improve an existing test generator, moving it from one of the simpler types to a more sophisticated knowledge-based generator. However, this is a major effort that could draw essential resources from the design team or require skills that are not available. This is especially true since the development of such an advanced tool requires experienced engineers who have knowledge of processor architecture, verification and high level software development. In addition, the end product of such an effort is likely to be difficult to use with little documentation.

At this point, a commercial RTG becomes attractive. The RAVEN® generator developed by Obsidian Software is often a better solution than that described above. RAVEN® is a full-featured

RTG composed of three main elements: 1) the core, 2) the ISA layer and 3) the GUI.

The RAVEN® core includes constraint solvers, a simulator interface, multi-processor support, and test output functions. The ISA layer includes information specific to the particular architecture under test. This layer is modified to support new ISAs. The GUI provides an easy-to-use interface to all of the configuration features of the generator. The generator itself can be run from the GUI or from the command line. The features provided by RAVEN® are outlined under the Commercial column in Table 2.

Table 2: Feature Comparison

	Table	Static	Dynamic	Knowledge	Commercial
Random Instructions	X	X	X	X	X
Random Data	X	X	X	X	X
Parameterized Macros X	X	X	X	X	X
Support for complex ISA		X	X	X	X
Control flow generation		X	X	X	X
Macros		X	X	X	X
Loops with low randomness		X	X	X	X
Link with ISS			X	X	X
State set up code not needed; enhances randomness			X	X	X
State aware generation			X	X	X
Integrated self-checking code			X	X	X
Loops with moderate randomness			X	X	X
Support for dynamic resource scheduling				X	X
Constraint Engine				X	X
Architectural pipeline solver				X	X
Complex Interrupt & exception support				X	X
Full Paging				X	X
Cache controls				X	X
Biases / API exposed to the user				X	X
Reusable generation core					X
Separate architecture layer for each ISA					X
Loops with full randomness					X
Multiprocessor / Multithreaded					X
Detailed instruction testing knowledge					X
User accessible settings to control test intent					X
User accessible parameters for instruction and operand selection					X
Iterators to support common testing methods; every instruction followed by every other instruction					X
GUI					X
Manual					X
Tutorials					X
Generator development system including regression					X
Commercial support					X

RAVEN® can generate significant tests for a new platform within a few weeks. Advanced features such as interrupt control and task switching may require more time to implement. Depending on the similarity of the new features to those already supported in RAVEN®. However, even while this development is proceeding, the full power of the core functions is available. In addition, the GUI is available and generally complete from initial deployment, although some user-configuration parameters may be added as architectural support becomes more complete – for example controls for the generation of task switches or exceptions may be modified to reflect specific architectures.

RAVEN® currently supports multiple architectures such as ARM, MIPS, PowerPC, and proprietary DSPs. Additionally, Obsidian can port RAVEN® to fit almost any proprietary architecture. Porting RAVEN® typically requires between 3-9 months depending on ISA complexity and similarities to existing architectures.

Once the initial version of RAVEN® has been deployed, verification engineers can immediately begin generating tests. The help provided in the tool and the manual are usually sufficient for engineers to understand how to use the tool. Obsidian also offers training classes and on-site support.

Conclusions

Random test generators have a long development history, most of it confined within the individual companies where the tool was to be used. As designs become more complex and design cycles shorter, verification has become a bottleneck in product development, typically taking 50-70% of the total product development time. Internal development of tools often saps resources from other areas of development. Commercial RTGs have recently become available and can be quickly ported to new architectures. These provide a basis of fully tested core functions upon which to build the architecture specific support.

References

- [1] Glassner, C. “Random Test Generation for a RISC Processor Core” *Verifyer*, June 2001
- [2] Aharon, A., D. Goodman, M. Levinger, Y. Lichtenstein, Y. Malka, C. Metzger, M. Molcho, G. Shurek “Test Program Generation for Functional Verifacaton of PowrPC Processors in IBM,” *Proceedings of 32nd ACM/IEE Design Automation Conference*, 1995

Contact Info

Obsidian Software Inc.
609 Castle Ridge Ct
Suite 210
Austin TX, 78746

(512)330.9818
dvinfo@obsidiansoft.com

