DOT/FAA/AR-08/55

Air Traffic Organization
Operations Planning
Office of Aviation Research
and Development
Washington, DC 20591

Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure No. 43 Phase 3 Report

February 2009

Final Report

U.S. Department of Transportation
Federal Aviation Administration

Technical Report Documentation Page

1. Report No. DOT/FAA/AR-08/55	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle MICROPROCESSOR EVALUATIONS FOR SAFETY-CRITICAL, REAL-TIME APPLICATIONS: AUTHORITY FOR EXPENDITURE NO. 43 PHASE 3 REPORT		5. Report Date February 2009
		6. Performing Organization Code
7. Author(s) Rabi N. Mahapatra, Praveen Bhojwani, Jason Lee, and Yoonjin Kim		8. Performing Organization Report No. TAMU-CS-AVSI-72005
9. Performing Organization Name and Address Aerospace Vehicle Systems Institute Texas Engineering Experiment Station Texas A&M University Department of Computer Science College Station, TX 77843-3141		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DTFACT-03-Y-90018
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Air Traffic Organization Operations Planning Office of Aviation Research and Development Washington, DC 20591		13. Type of Report and Period Covered Final Report March – July 2007
		14. Sponsoring Agency Code AIR-120

15. Supplementary Notes
The Federal Aviation Administration Airport and Aircraft Safety R&D Division COTR was Charles Kilgore.

16. Abstract

This report discusses the findings concerning safety issues in using today's commercial off-the-shelf (COTS) microprocessors on aircraft. The report addresses the applicability of RTCA/DO-254 to microprocessors, documents potential safety concerns when using modern COTS microprocessors on aircraft, and proposes potential approaches for addressing these safety concerns.

The research was performed in multiple phases with participation from avionic system developers (BAE Systems, The Boeing Company, Lockheed Martin, and Smiths Aerospace) and Federal Aviation Administration organizations responsible for aircraft safety research and development. Phase 1 established the project scope and identified the research parameters, as well as reviewed the available literature and surveyed microprocessor users to identify the issues and potential solutions associated with the use of COTS microprocessors in regulated, safety-critical applications. Phase 2 developed the project objectives and found an approach to work toward the solution of these issues and the achievement of these objectives. Phase 3, documented in this report, evaluated the proposed approach and continued the development of processes, services, and prototype tool development. Phase 4, depending heavily on industry experience, will attempt to determine if new approaches can be developed to ensure system safety and provide more effective methods to accumulate safety evidence for certification while reducing the time and cost to develop and certify complex systems. These results will be documented in a Microprocessor Selection and Evaluation Handbook to facilitate application to real-time, safety-critical applications.

17. Key Words Microprocessor, System-on-a-chip, Qualification, Safety, Critical Systems, Avionics, Certification		18. Distribution Statement This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161.	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 43	22. Price

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application programming interface
AFE	Authority for Expenditure
BOMV	Buffer-Oriented Microarchitectural Validation
COTS	Commercial off-the-shelf
CPU	Central processing unit
FSM	Finite state machine
GCC	GNU compiler collection
GCS	Generator Constructor Set
GHS	Green Hills Software
GUI	Graphical user interface
IDE	Integrated Development Environment
ISA	Instruction set architecture
L1	Level 1
L2	Level 2
LRU	Least recently used
MAF	Microprocessor Approval Framework
MAI	Micro-Architectural Interface
MIL	Missed Instruction List
OS	Operating system
PMC	Project Management Committee
RAVEN	Random Architecture Verification Engine
RTL	Register transfer level
RTOS	Real-Time Operating System
RTPG	Random test program generator
SBC	Single-board computer
SoC	System-on-a-chip
SNU	Seoul National University

EXECUTIVE SUMMARY

The Microprocessor Evaluations Project researched methods to assess commercial off-the-shelf (COTS) microprocessors for safety-critical aerospace applications. Neither RTCA/DO-254 nor RTCA/DO-178B documents specify how microprocessors should be ensured. This project investigated assessment criteria and safety concerns for microprocessors and developed methods and procedures to (1) permit the safe, economical qualification of microprocessor applications with complex, nondeterministic architectures, (2) select microprocessors for safety-critical aerospace applications that can be proven to be safe, and (3) provide input to the Federal Aviation Administration for regulations and policy development regarding the design and test of COTS microprocessor components.

Authority for Expenditure (AFE) No. 43 Phase 3 used the Microprocessor Approval Framework to assess the COTS Freescale MPC7447 microprocessor and the Freescale MPC8540 system-on-a-chip, examine the microarchitectures of these microprocessors with Buffer-Oriented Microarchitectural Validation (BOMV), and explore the use of third-party tools, such as full-system simulators and microprocessor verification, to analyze the microprocessor features.

It was determined that BOMV was inadequate to resolve the safety issues of complex, nondeterministic microprocessors in critical applications due to two primary factors.

1. Microprocessor design data was found to be unavailable or inaccurate, making it impossible to create high-fidelity state models of microprocessor architectural features.

2. The growth in complexity, the short life expectancy, the rapid evolution, and the protected nature of microprocessors and systems-on-a-chip intellectual property precluded development and maintenance of high-fidelity state models.

The AFE No. 43 Project Management Committee decided that, while the Microprocessor Evaluation Framework and existing architectural models documented in this report were useful, the attempt to establish proof-of-safety by direct evaluation of the design of complex, multicore, nondeterministic microprocessors was road-blocked due to the tendency of the industry to protect market share and product information.

Phase 4, depending heavily on industry experience, will attempt to determine if new approaches can be developed to ensure system safety. Phase 4 will also provide more effective methods to accumulate safety evidence for the certification process while reducing the time and cost to develop and qualify complex systems.

1. INTRODUCTION.

The Aerospace Vehicle Systems Institute Authority for Expenditure (AFE) No. 43 Supplement 2 Phase 3 evaluated feature-modeling techniques proposed as a part of the Microprocessor Approval Framework (MAF) and investigated the feasibility of using third-party simulation tools for microprocessor and system-on-a-chip (SoC) safety analysis.

Current trends toward using commercial off-the-shelf (COTS) microprocessors present safety challenges, especially with growing design complexity, the vast array of supported features, and limited design documentation. A formal framework for the approval of COTS microprocessors in aerospace systems is essential. The proposed MAF that is applicable to COTS microprocessors was evaluated.

Phase 1 of this project

- reviewed the available literature and surveyed microprocessor users to identify the issues and potential solutions associated with the use of microprocessors in regulated, safety-critical applications.
- addressed the suitability of RTCA/DO-254.
- listed possible evaluation criteria and categories.
- discussed tentative classification of these criteria in safety-critical levels.
- describe the potential issues with obsolescence management.
- presented feature modeling as a potential approach to identify risks.
- presented test and validation aspects, and safety concerns, related to these aspects.
- outlined risk-related issues that arise in a SoC that integrates microprocessor internet protocols with peripheral components.

Phase 2 developed three major project objectives and found an approach to work toward the solution of the issues and achievement of these objectives. These objectives included formulating a process for assessing safety of emerging microprocessor features and SoCs, developing tools and guidelines to aid in the safety assessment process, and preparing a draft Microprocessor Selection and Evaluation Handbook. This draft Handbook is considered only an initial version of this Handbook that identifies preliminary evaluation criteria and safety concerns for microprocessors.

Phase 3, documented here, used the Buffer-Oriented Microarchitectural Validation (BOMV) technique on selected COTS microprocessors. It also investigates the feasibility of using third-party tools, such as Virtutech's Simics, Obsidian Software, Inc.'s Random Architecture

Verification Engine (RAVEN®), and IBM®'s Turandot, for the safety analysis of COTS microprocessors and SoCs.

The AFE No. 43 Project Management Committee (PMC) decided that, while the MAF and existing architectural models documented in this report were useful, the attempt to establish proof of safety by direct evaluation of the design of complex, multicore, nondeterministic microprocessors was roadblocked due to the tendency of the industry to protect market share and product information.

Phase 4, depending heavily on industry experience, will attempt to determine if new approaches can be developed to ensure system safety. Phase 4 will also provide more effective methods to accumulate safety evidence for the certification process while reducing the time and cost to develop and quantify complex systems.

2. THE BOMV EVALUATION.

In Phase 2, BOMV [1] was identified as a candidate technique to assess the safety of certain microarchitectural features of COTS microprocessors. A primary research goal for Phase 3 was to evaluate the effectiveness of this technique in assisting the aerospace industry in proving the safety of COTS microprocessors.

2.1 OVERVIEW.

The BOMV technique takes features described in architecture specifications, generates finite state machines (FSM) based on their behaviors, and fully verifies each FSM in the most efficient way possible. This technique focuses on the critical buffers of a microprocessor. A critical buffer is defined as any storage unit within a microprocessor that also contains state information. This state information then can be used to generate FSM to generate tests. For example, on-chip cache memories that support cache coherence protocols often provide state information for each cache block that signifies whether that block is being shared, modified, or is invalid. Refer to the Phase 2 Report [2] and to the original work [1] for additional information.

The BOMV technique is a sequence of the following procedures in sections 2.1.1 through 2.1.4.

2.1.1 Feature Identification.

Any feature that depends on a critical buffer may be modeled with this technique. Typical features that rely on critical buffers are branch prediction, register renaming, reservation stations, and the reorder buffer [1]. For example, the branch prediction feature of the IBM PowerPC 604® (the processor used to test this technique) relies on two critical buffers: the Branch Target Address Cache and the Branch History Table.

2.1.2 The FSM Construction.

For any feature, a set of control signals dictate the state of each entry in that feature's critical buffer. Any change in the control signals for an entry can be viewed as a state transition. Therefore, any entry in a critical buffer can be viewed as a finite state machine.

2.1.3 Test Generation.

To achieve 100% FSM transition coverage, transition tours or checking sequences may be used. Once a sequence of state transitions is selected, it is translated into instructions through the use of atomic sequences. An atomic sequence is a set of instructions that maps to a specific transition in the FSM. The test generator merely connects together atomic sequences that form a complete transition tour.

2.1.4 Simulation.

Once a sequence of instructions has been constructed, a simulator is used to track transition coverage. Since the test generator will always construct a complete transition tour, the simulator will always return 100% FSM transition coverage.

2.2 EVALUATION TECHNIQUE.

Although the BOMV technique is a technically feasible solution, there have been no major comparative studies that demonstrate its effectiveness.

The goal of this study is to determine if a statistically significant correlation can be established between the BOMV coverage metric and industry-standard coverage metrics. If this can be accomplished, BOMV can be used by the aerospace industry to assist in gaining regulatory approval for using COTS microprocessors in aircraft.

The research team selected the following industry-standard register transfer level (RTL) coverage metrics for comparison:

- Line coverage
- Conditional coverage
- Toggle coverage

Section 2.3 describes the application of this evaluation technique on the Sun™ Microsystems, Inc.'s OpenSPARC™ T1 processor.

2.3 CASE STUDY: THE SUN MICROSYSTEMS OpenSPARC T1 PROCESSOR.

In 2006, Sun Microsystems, Inc. released an open-source version of the UltraSPARC™ T1 processor to the general public. Full implementation details, such as the RTL description of the processor, simulation environments, and documentation, were included in this release. This study used version 1.4 of the OpenSPARC T1 processor.

For this study, the research team selected the Missed Instruction List (MIL) FSM and the Thread Selection FSM as the two critical buffers to evaluate the effectiveness of BOMV. Sections 2.3.1 and 2.3.2 describe the behaviors of these buffers and their corresponding FSM constructions, according to the OpenSPARC specification.

2.3.1 The MIL Behavior.

The MIL is a critical buffer within the SPARC core of the OpenSPARC T1 processor that tracks the instruction fetches that resulted in an instruction cache miss.

The OpenSPARC T1 specification describes the behavior of the MIL as follows:

> "The MIL cycles through the following states:
>
> 1. Make request.
> 2. Wait for an I-cache fill.
> 3. Fill the first 16 bytes of data. The MIL sends a speculative completion notification to the thread scheduler at the completion of filling the first 16 bytes.
> 4. Fill the second 16 bytes of data. The MIL sends a completion notification to the thread scheduler at the completion of filling the second 16 bytes.
> 5. Done." [3]

Through interpretation, an MIL FSM construction would appear as shown in figure 1.

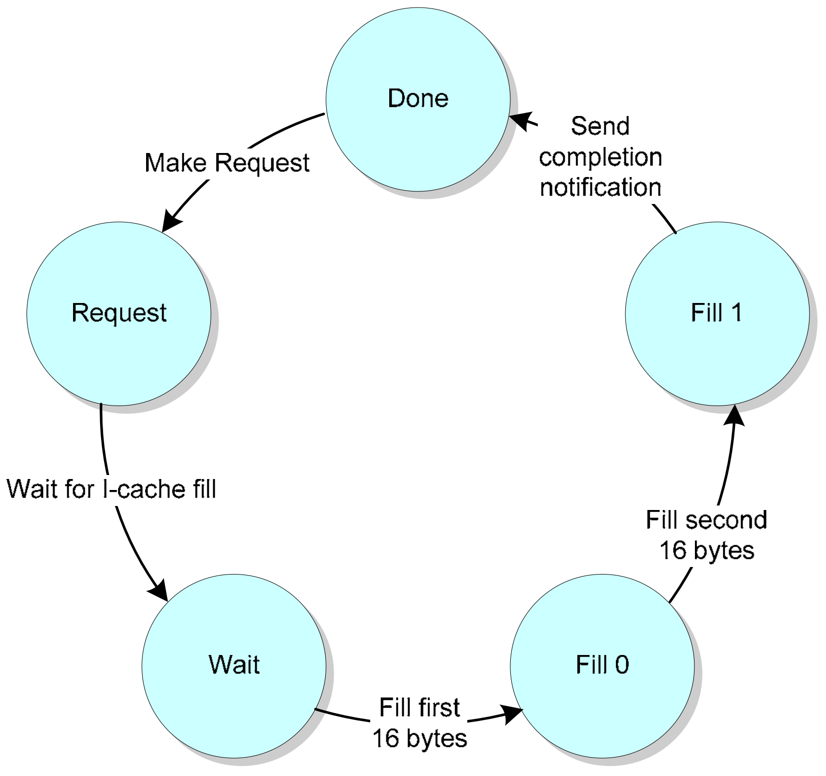


Figure 1. An MIL FSM Based on Specification

2.3.2 Thread FSM Behavior.

Each SPARC core of the OpenSPARC T1 processor can handle four threads. The operation of each thread is controlled by the Thread FSM. The Thread FSM speculatively or nonspeculatively executes, puts in queue, or halts threads so execution time of all threads is optimized.

The specification graphically shows the behavior of the Thread FSM. These graphics correspond to threads that are nonactive, active, or speculatively executing.

The nonactive thread state diagram shows three states: Idle, Active, and Halt. According to the specification, threads transition between these three states in the following ways:

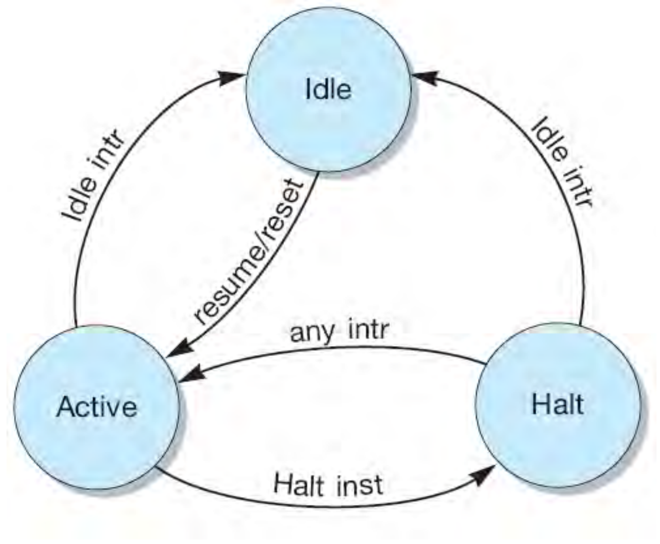
1. Resume or reset: Either the system has been initially reset from the Power-on state, or the thread is resumed through a software-initiated resume interrupt.

2. Idle intr: Software sends an idle interrupt.

3. Any intr: Once a thread is halted, any interrupt will make the associated thread active.

4. Halt inst: A special halt instruction is sent to the associated thread.

Figure 2 describes the behavior of the Thread Controller for a thread that traverses between active and nonactive states.



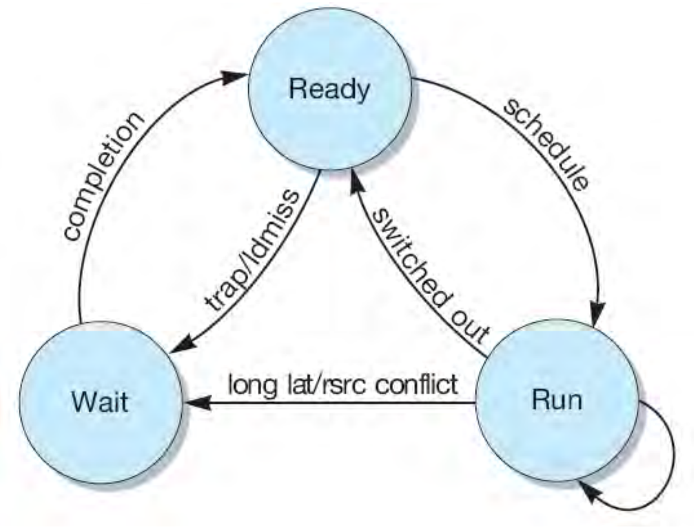
inst = instruction
intr = interrupt

Figure 2. Thread FSM: Basic Transition of Nonactive States [3]

The OpenSPARC specification describes the conditions necessary for an active thread to be placed in the wait state as follows:

> "An active thread could be placed in the wait state because of any of the following reasons:
>
> 1. Wait for an I-cache fill.
> 2. Wait due to store buffer full.
> 3. Wait due to long latency, or a resource conflict where all resource conflicts arise because of long latency.
> 4. Wait due to any combination of the preceding reasons" [3]

Figure 3 describes the behavior of the Thread Controller for an active thread. Threads may be processed or queued, depending on the thread scheduler, cache misses, or exceptions.



lat = latency
rsrc = resource

Figure 3. Thread FSM: Thread State Transition of Active Thread [3]

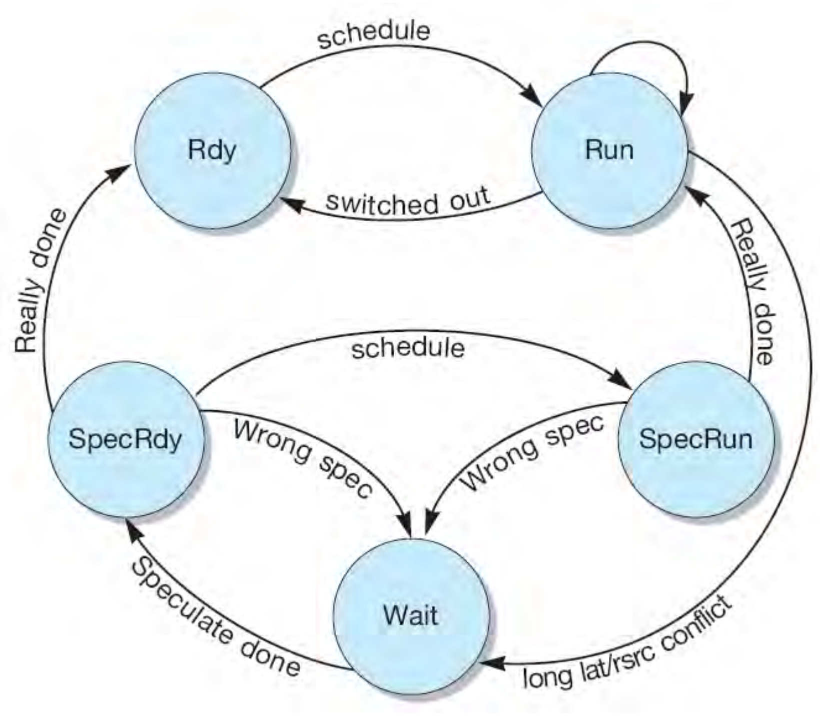
The OpenSPARC specification does not explicitly describe the transitions and states of speculatively running threads in detail. For a verification team attempting to create buffer-oriented models based only on specification details, such missing information poses serious problems. However, for the purpose of clarity, this report includes the following details regarding figure 4 based on inspection of the entire specification and the RTL of the OpenSPARC processor. Figure 4 describes the behavior of the Thread Controller for threads executing speculatively.

1. Threads may only enter speculative states by entering the Speculative Ready (SpecRdy) state from the Wait state.

2. A thread may transition from a speculative state to its normally running counterpart (SpecRdy to Rdy or SpecRun to Run) if an associated cache hit occurs, noted as Really done in the specification and figure 4.

3. Threads in a speculative state (SpecRdy or SpecRun) return to the Wait state if incorrect speculation occurs.

4. Threads in speculative states (SpecRdy or SpecRun) may transition as threads in normal states (Rdy or Run) through scheduling and switching activities.



spec = speculative
lat = latency
rsrc = resource

Figure 4. Thread FSM: State Transition for a Thread in Speculative States [3]

2.4 RESULTS.

For each selected critical buffer, assembly code-based tests were generated that would satisfy the requirement that all transitions in the FSMs would be traversed. RTL coverage metrics were then recorded as each test was executed to determine if a correlation exists between FSM transition coverage and RTL coverage.

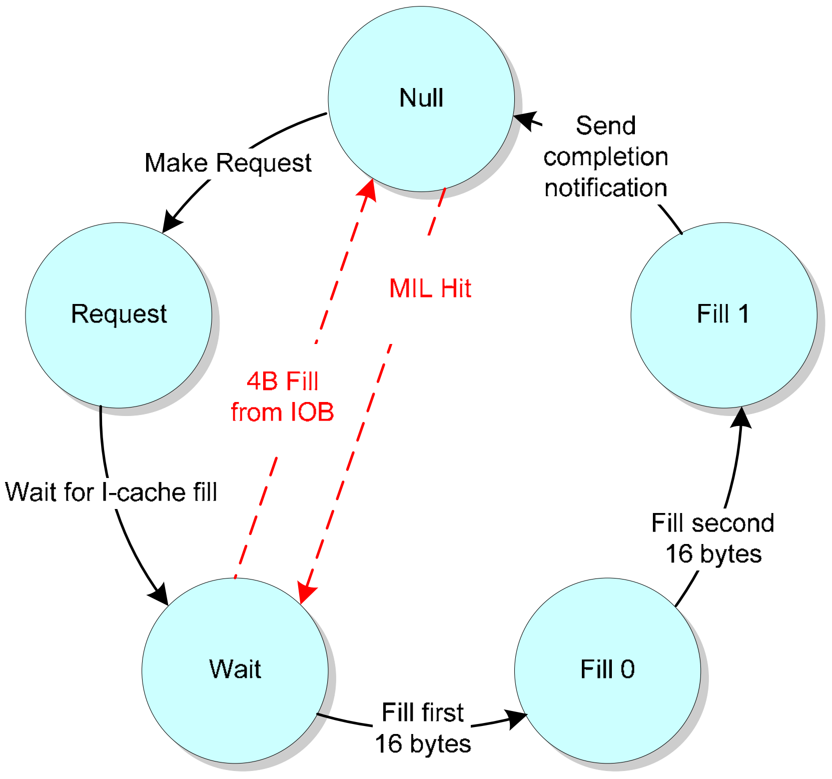
2.4.1 The MIL.

Although a complete transition tour of the MIL FSM generated through BOMV was constructed, the maximum RTL transition (toggle) coverage achieved was 66%, as shown in table 1.

Table 1. Coverage Range for MIL

Coverage	Line (%)	Conditional (%)	Toggle (%)
Minimum	76.00	40.28	37.14
Maximum	90.91	86.11	66.67

After reviewing the RTL implementation of the MIL, two undocumented transitions were found. Comparing figure 1 with figure 5, the difference between the specified behavior of the MIL with the actual implementation of the MIL is shown.



IOB = input output buffer
*Dashed transitions are undocumented in the specification.

Figure 5. An MIL FSM Based on RTL Implementation

2.4.2 Thread FSM.

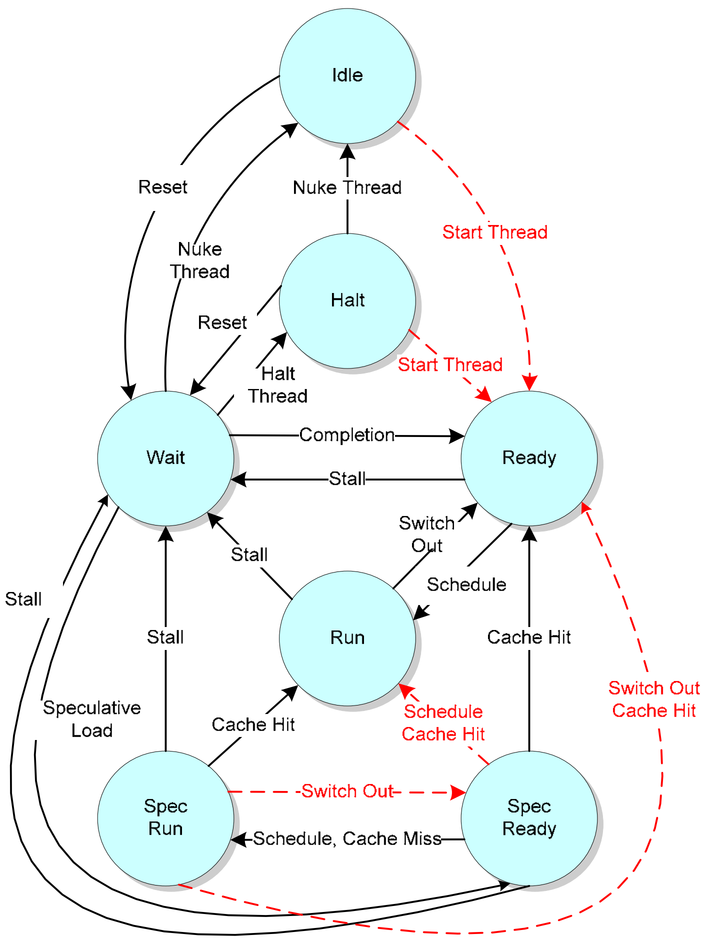
Table 2 shows the results of tests constructed from BOMV models of the Thread FSM. Similar to the MIL FSM, insufficient levels of coverage were achieved during a complete transition tour of the BOMV-based state machine. Maximum coverage for the various metrics ranged from 76.67% to 80.77%, suggesting that functionality not described in the specification exists for the Thread FSM.

Table 2. Coverage Range for Thread Controller

Coverage	Line (%)	Conditional (%)	Toggle (%)
Minimum	69.23	65.91	56.67
Maximum	80.77	77.27	76.67

Although the specification describes the Thread Controller with three separate FSMs for threads in Active, Nonactive, and Speculative states, the implementation of the Thread Controller combines these behaviors into a single FSM. This partitioning of the behavior into three FSMs creates ambiguity when transitioning between Active, Nonactive, and Speculative modes.

Figure 6 describes the full behavior of the Thread Controller as implemented in the RTL.



Note: Dashed transitions are undocumented in the specification.

Figure 6. Thread Controller FSM

After reviewing the RTL implementation of the Thread Controller, the following undocumented transitions were found.

1. Start Thread: Threads can be placed in the Ready state from either Idle or Halt states by a Start Thread condition.

2. Schedule Cache Hit: A thread can transition from the Speculative Ready (SpecReady) state to the Run state if a specific cache hit event occurs.

3. Switch Out Cache Hit: A thread can transition from the Speculative Running (SpecRun) state to the Ready state if a cache hit for a Waiting thread occurs.

4. Switch Out: A thread in the Speculatively Running (SpecRun) state may transition to the Speculatively Ready (SpecReady) state due to normal thread-switching activity.

Because these transitions are undocumented in the specification, detailed descriptions cannot be provided.

2.5 CONCLUSIONS.

These discrepancies can be attributed to one or more of the following conditions:

- Instrumentation error
- Test generation error
- Specification error

As shown in section 2.4, there are behavioral differences between the descriptions that the specification provides and the implementations of these buffers in the RTL. Because of this, establishing a significant correlation between the BOMV metric and RTL metrics does not signify that a meaningful relationship exists.

2.6 FUTURE DIRECTIONS.

Although no statistically significant correlation between BOMV coverage metrics and industry-standard coverage metrics could be established, these results are still useful in determining future research directions.

The evaluation of BOMV has demonstrated the effects of relying only on specification and other end-user documentation to observe the microarchitectural behaviors of a microprocessor. Short-term solutions may include the use of third-party tools that were created with the cooperation of manufacturers. In the future, direct participation of manufacturers during the safety assessment process would greatly improve the quality of the assessment.

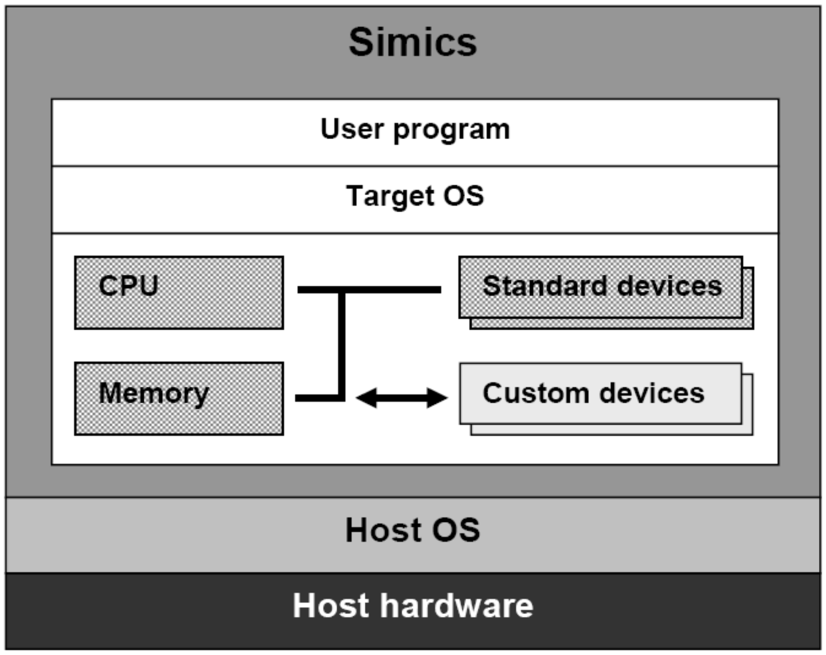
3. TOOLS EVALUATION.

The second activity of this research focused on the study and evaluation of commercial verification tools that may assist in the COTS microprocessor safety assessment process. Specifically, the AFE No. 43 Research Team acquired Simics and RAVEN for this activity.

3.1 THE SIMICS EVALUATION.

3.1.1 Introduction.

Simics is a full-system simulator that allows real workloads, such as operating systems (OS), network stacks, and complex applications, to run in simulation. In addition, Simics can model the binary interfaces to buses, interrupt controllers, disks, and video memory. An entire view of Simics is shown in figure 7. All target software, including OS and user programs can be run on the simulated hardware. There are several standard devices that are provided with Simics. These are devices, such as video cards, network interface cards, ports, and buses, which can be used easily in a simulation model. Custom-made devices also can be simulated in Simics by adding modules written using the Simics application programming interface (API).



CPU = Central processing unit

Figure 7. Simics Architecture

<u>3.1.2 Capability and Limitations of Simics</u>.

Simics provides more capabilities than general emulators or instruction set simulators, since it is an entire system platform running an OS. However, Simics also has some limitations with respect to timing accuracy and visibility to maintain an acceptable level of performance. Major capabilities and limitations of Simics can be considered from the following viewpoints:

- Coherency

  Simics can run the same unmodified binaries intended for the physical target platform, since the virtual system platform it provides is functionally identical to the real target environment. This allows for efficient software development and evaluation.

- Speed

  - Capability: Simics was designed with fast simulation as a primary consideration. Often, a Simics simulation will run as fast as the real hardware, or even faster.

  - Limitation: High-speed simulation in Simics is only possible without cycle accuracy. If a user requires cycle-accurate simulation allowing stalling instruction or data profiling, simulation speed is dramatically reduced.

- Accuracy

  - Capability: Simics is an event-driven simulator with a maximum time resolution of a clock cycle. During each cycle, events, such as device interrupts and internal state updates, are triggered and executed. In addition, the execution model can be extended by adding timing models to control the timing of memory operations, typically using the memory hierarchy interface.

  - Limitation: The default execution model of all processors supported by Simics is an in-order execution model. For out-of-order execution, the default processor models should be modified through the Simics Micro-Architectural Interface (MAI) [4]. Simics MAI transcends the default model by providing an infrastructure for parallelized and speculative execution, allowing a much more accurate timing simulation of complex processors. However, Simics MAI is only available for the UltraSPARC and x86/x86-64 processor models at this time.

- Variety

  - Capability: The core of Simics is the simulation model. Virtutech provides various models of individual devices as well as complete systems, as shown in table 3.

  - Limitation: The academic version of Simics only provides 10 processor models, whereas the commercial version of Simics can currently support 54 processor models.

- Visibility

  - Capability: Simics offers a greater level of visibility of system behavior compared to hardware evaluation platforms. With Simics, it is possible to log the activities of all components in the entire system, such as cache behavior, memory maps or read/write information, through a universal asynchronous receiver transmitter or Ethernet.

  - Limitation: Simics only supports predefined points of visibility, depending on the platform being simulated. It does not provide more detailed visibility offered by products, such as Seamless [5], showing RTL visibility of processor, system component, and buses.

- Extensibility

  A Simics module is an executable code loaded dynamically into the simulator, so it could virtually perform any operation. However, to be of any practical use, a module has to interact with Simics, other modules, or the user. Simics provides a set of functions (the Simics API) for the modules to work within the simulator framework, and a device modeling language has been designed to make writing new device models as simple as possible [4].

- Debug Facilities

Simics is designed to run not just the target system programs, but to gather a great deal of information during run time. Simics is specially designed to achieve good performance even with a high level of instrumentation and very large workloads. Simics provides a variety of statistics in its default configuration and allows various add-ons to be developed by advanced users and plugged into the simulator.

Table 3. Model Libraries for use With Simics

ISA	Simulated CPUs	AL	ISA	Simulated CPUs	AL
Digital Alpha	Alpha 21164	✓	IBM PowerPC/ POWER	Freescale PowerPC 8641D	
ARM®	Intel® StrongARM	✓		IBM POWER6	
	ARM926EJ-S		SPARC	LEON2	
	ARM966E-S			Sun SuperSPARC II	
IA-64	Intel Itanium®			Sun UltraSPARC II	✓
	Intel Itanium 2	✓		Sun UltraSPARC III / III Cu / IIIi	✓
MIPS®	MIPS-4Kc	✓		Sun UltraSPARC IV (+)	✓
	MIPS-5Kc			Sun UltraSPARC T1	
	PMC-Sierra E9000™		TI™	TI TMS320C64	
	QED RISCMarc™ RM7000™			TI MSP430	
IBM PowerPC/ POWER	AMCC®/IBM PowerPC® 403GCX		X86	Intel 80386DX	✓
	AMCC/IBM PowerPC 405GP	✓		Intel 80486SX	✓
	AMCC/IBM PowerPC 440 GP/GX	✓		Intel 80486DX	
	AMCC/IBM PowerPC 750 (FX/G)	✓		Intel 80486DX2	✓
	AMCC/IBM PowerPC 755			Intel Pentium®	✓
	AMCC/IBM PowerPC 970FX	✓		Intel Pentium/MMX	✓
	BAE Systems RAD750®			Intel Pentium Pro	✓
	Freescale PowerPC™ 7400			Intel Pentium II	✓
	Freescale PowerPC 7447			Intel Pentium III	✓
	Freescale PowerPC 7450			Intel Pentium 4	✓
	Freescale PowerPC 7457			Intel Pentium M	
	Freescale PowerPC 603e			Intel Core™	
	Freescale PowerPC 8260		AMD64/ EM64T	AMD Opteron™	✓
	Freescale PowerPC 8270			AMD Athlon 64™	
	Freescale PowerPC 8280			Intel Xeon®	
	Freescale PowerPC 8540			Intel Core 2	
	Freescale PowerPC 8548				
	Freescale PowerPC 8641				

AL = Academic license

3.1.3 Simics With Academic License.

The academic version of Simics provides the same capabilities and features of the commercial version. However, not all processor models are available in the academic distribution. For commercial users, the product may be customized with one or more proprietary elements, such as application-specific integrated circuits or field-programmable gate arrays. Table 3 lists all instruction set architectures (ISA) and central processing units (CPU) currently offered in Virtutech's model portfolio (the check notation (✓) indicates CPUs are supported by the academic license) [5].

3.1.4 Simulation Flow.

Simulation on Simics requires a model for the target device or system. It can be monitored in terminals on a host computer executing the model. To run an application on the target model, it must be compiled for that specific platform. A high-level language specification of the application (C or C++) is typically compiled with a cross compiler to generate a binary for the target model.

Cross compilers [6] are generally used in embedded system or heterogeneous system platforms. It is used for platforms where it is inconvenient or impossible to compile within the target environment, like microcontrollers, which operate on limited memory. This research used CrossTool [7], which is a cross compiler built on top of a GNU compiler collection (GCC). To execute the binary code on the target OS, Simics provides a way for the simulation system to access files that are located on the host system. SimicsFS [8] is a Linux® kernel file system module that communicates with the simulated device, namely a corresponding Simics module. Using SimicsFS, the binary code compiled on the host OS can be executed on the target OS.

Figure 8 shows entire simulation process on Simics.

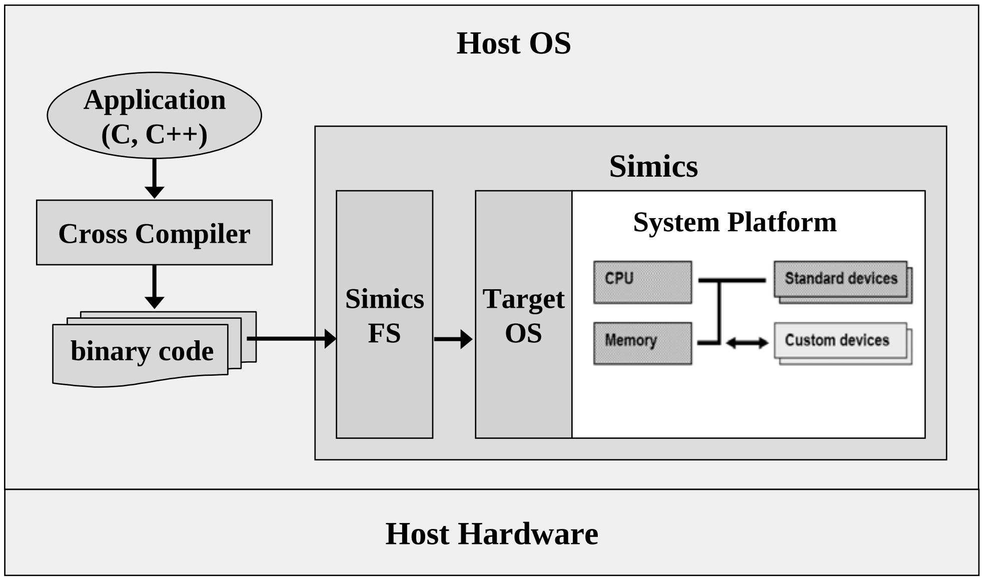


Figure 8. Simulation Process on Simics

3.1.5 Evaluation of Simics Simulation.

To show the capabilities of Simics, this research used the Ebony® IBM/AMCC PPC440GP [9] model running MontaVista® Linux 2.1. The Seoul National University (SNU) real-time benchmarks [10] were used to provide applications.

Ebony includes a PowerPC PPC440GP processor operating at 100 MHz, memory, flash memory, four peripheral component interconnect slots, two Ethernet interfaces, two serial ports, a real-time clock, and two I2C buses.

The research team has compiled the SNU benchmarks for this target platform. The SNU real-time suite is a collection of synthetic benchmarks. This suite is used for worst-case timing analysis. Using the CrossTool [8] cross compiler, the research team was able to generate executables for the IBM PowerPC platform. Using this simulation environment and benchmarks, the research team performed cache simulation and evaluation of benchmark execution times.

The SNU real-time benchmarks suite [10] was used for worst-case timing analysis. The benchmark programs are C sources that are collected from numerical calculation programs and DSP algorithms, as listed in table 4. The original programs are modified to be used in the experimental environment. The programs were collected from references 11-13.

Table 4. The SNU Real-Time Benchmarks List [10]

Program	Description
adpcm.c	CCITT G.722 ADPCM (Adaptive Differential Pulse Code Modulation) algorithm
bs.c	Binary search program
crc.c	CRC (Cyclic Redundancy Check) example program
fft1.c	FFT (Fast Fourier Transform) using Cooly-Turkey algorithm
fft1k.c	FFT (Fast Fourier Transform) for 1K array of complex numbers
fibcall.c	Fibonacci series function
fir.c	FIR filter with Gaussian number generation
insertsort.c	Insertion sort
jfdctint.c	JPEG slow-but-accurate integer implementation of the forward DCT
lms.c	LMS adaptive signal enhancement
ludcmp.c	LU decomposition algorithm
matmul.c	Matrix multiplication
minver.c	Matrix inversion algorithm
qsort-exam.c	Nonrecursive quick-sort algorithm
qurt.c	Root computation of quadratic equations
select.c	N-th largest number selection algorithm

3.1.6 Cache Simulation.

3.1.6.1 The g-Cache Model.

By default, Simics does not model any detailed cache systems. It uses a functionally accurate memory system that results in better performance of the simulation at the cost of timing accuracy. In this capacity, Simics acts as an instruction set simulator and not a cycle-accurate processor simulator, which means all transactions involving the level one (L1) and level two (L2) caches are ignored, and only misses to be fetched from the memory are considered. In Simics, all detailed cache transactions performed by the processor can be made visible by adding a cache model. Without a cache model, Simics does not model incoherence, and memory accesses are assumed to take zero simulation time to be performed.

Simulating a cache is implemented by adding a g-cache module, which is the standard cache model of Simics [6]. It handles one transaction at a time in a flat way: all needed operations (copy-back and fetch) are performed in order and at once. The cache returns the sum of the stall times reported for each operation.

The g-cache has the following properties:

- Configurable number of lines, line size, and associativity
- Physical or virtual index and tag
- Configurable write allocate or write back policy
- Random, true least recently used (LRU) or cyclic replacement policies
- Support for several processors connected to one cache
- Configurable penalties for read/write accesses
- Cache miss profiling

If the transaction is uncacheable, g-cache ignores it. If the transaction is a read hit, g-cache returns penalty_read cycles of penalty. If the transaction is a read miss, g-cache asks the replacement policy to provide a cache line to allocate. Then, the new cache line is emptied. The new data is fetched from the next level, incurring penalty_read_next cycles of penalty added to the penalty returned by the next level. The total penalty returned is the sum of penalty_read, plus the penalties associated with the copy-back (if any), plus the penalties associated with the line fetch.

Figure 9 shows the cache model implemented for this evaluation. Apart from L1 and L2 caches, it also contains the following modules:

- id-splitter: The id-splitter separates instruction and data accesses and sends them to separate L1 caches, namely, instruction and data cache.

- splitter: The splitter stops memory accesses that can cross a cache-line boundary. This will let uncacheable and correctly aligned accesses go through untouched, whereas others will be split in two accesses.

- trans-staller: The trans-staller is a simple device to simulate the memory latency. It stalls all accesses by a fixed amount of cycles. In addition, other parameter, such as line size, associativity, and read/write latencies, can be configured in a similar way as the research team did for transaction staller. In the same way, the research team can model the transaction splitter for instruction cache, the transaction splitter for data cache, the instruction-data splitter, and the L2 cache.

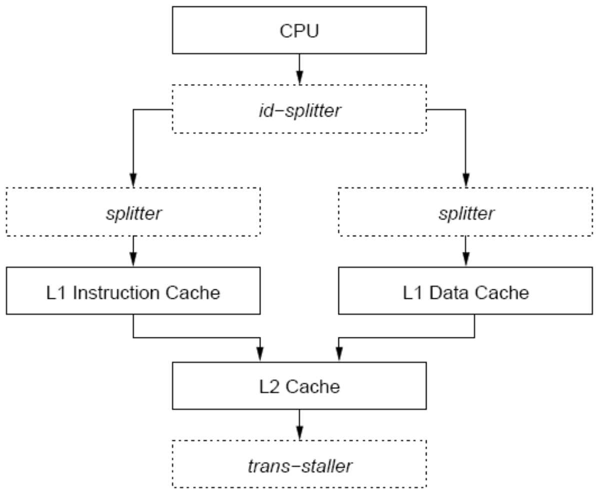


Figure 9. g-Cache Model [6]

Figure 10(a) shows the script to configure cache model on Simics.

```
# l2 cache: 512Kb Write-back

@l2c = pre_conf_object('l2c', 'g-cache')

@l2c.cpus = conf.cpu0

@l2c.config_line_number = 4096

@l2c.config_line_size = 128

@l2c.config_assoc = 8

@l2c.config_virtual_index = 0

@l2c.config_virtual_tag = 0

@l2c.config_write_back = 1

@l2c.config_write_allocate = 1

@l2c.config_replacement_policy = 'lru'

@l2c.penalty_read = 10

@l2c.penalty_write = 10

@l2c.penalty_read_next = 0

@l2c.penalty_write_next = 0

@l2c.timing_model = staller
```

(a) Script for Cache Configuration

```
# Cache statistics: l2c

Total number of transactions:     44087243

Device data reads (DMA):           0

Device data writes (DMA):           0

Uncacheable data reads:         5494

Uncacheable data writes:          340

Uncacheable instruction fetches:            0

Data read transactions:        30431

Data read misses:        4457

Data read hit ratio:        85.35%

Instruction fetch transactions:       320103

Instruction fetch misses:         2099

Instruction fetch hit ratio:        99.34%

Data write transactions:     43730875

Data write misses:         2664

Data write hit ratio:        99.99%

Copy back transactions:         3190

Lost Stall Cycles:     41393920
```

(b) Log File Generated From Cache Simulation

Figure 10. Cache Configuration and Generated Log File on Simics [6]

The g-cache model provides statistics that can be used to analyze cache behavior and to predict which configuration can be best-suited for a particular requirement. These are:

- Total number of transactions
- Device data read/write
- Uncacheable data read/write
- Instruction fetch
- Data read transactions
- Data read misses
- Data read hit ratio
- Instruction fetch transactions
- Instruction fetch misses
- Instruction fetch hit ratio
- Data write transactions
- Data write misses
- Data write hit ratio
- Copy-back transactions

The integrated cache results are created in a log file after simulation, as shown in figure 10(b).

3.1.6.2 Cache Configuration.

For the Ebony PPC440PG [9] target machine, the research team implemented L1 instruction cache with size 32 kb and data caches of size 16 kb along with 512 kb of L2 cache. Detailed cache configuration is as table 5.

Table 5. Cache Configuration

Configuration	L1 Instruction Cache	L1 Data Cache	L2 Cache
Line number	512	512	4096
Line size	64	64	128
Associativity	1, 2, 4, and 8	1, 2, 4, and 8	8
Write policy	-	Write through	Write back
Write miss policy	-	No-write allocate	Write allocate
Replacement policy	LRU	LRU	LRU
Read penalty cycles	3	3	10
Write penalty cycles	-	3	10

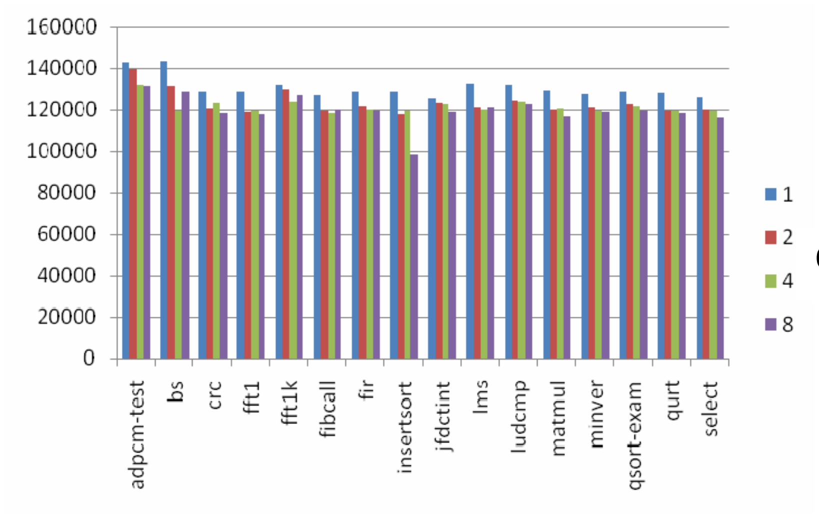
3.1.6.3 Result of Cache Simulation.

To obtain cache characteristics for the SNU real-time benchmarks, the research team is required to capture the cache behavior during the time the application executes on the target machine. A break point was first marked before the application was about to execute and another one after it completed execution. On the first breakpoint, the cache statistics were reset. This allowed Simics to capture only the cache statistics while the program or benchmark was in execution. The second breakpoint stored the cache data obtained for L1 instruction and data caches and L2 integrated cache into a log file. The data obtained reflected the statistics while the benchmark is in execution in the real workload environment simulated by Simics. Simulation was performed to show the L1 cache statistics under varying associativity (1, 2, 4, and 8) options for all SNU benchmarks.

3.1.6.4 Data Cache.

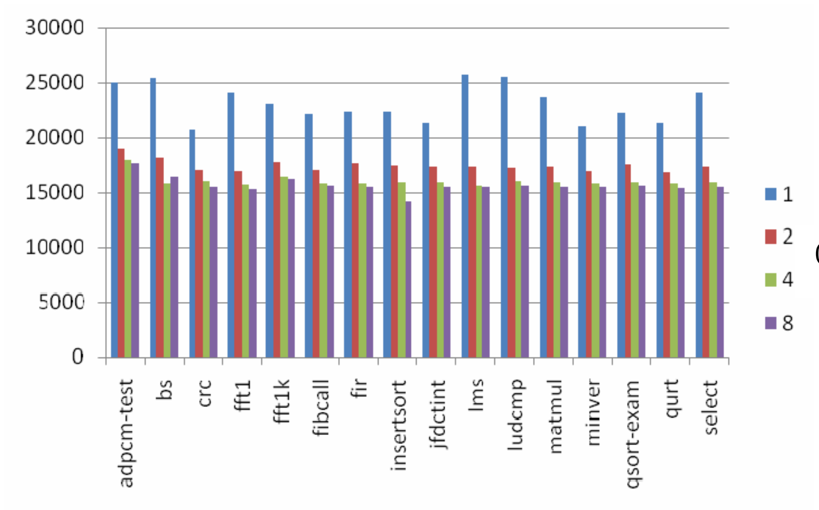
Figure 11(a) clarifies that by increasing the associativity, the number of write misses decreases. However, the maximum effect occurred when transitioning from a direct-mapped cache to a 2-way set associative. This is true for most benchmarks in SNU real-time suite. Figure 11(b) shows that the read misses decreased when the associativity increased. Moreover, a more pronounced effect happened for the transition from direct-mapped to 2-way set associative. The savings in the number of read misses for 4-way and 8-way configurations were somewhat significant for the insertsort benchmark only, while the other benchmarks have less difference in cache statistics.

Figure 12(a) shows that the cache write-hit ratio did not change much with changes in cache configuration. However, it is shown that benchmarks adpcm-test, fft1k, lms had a high write-hit ratio that stood out from the rest of the suite. As shown in the figure 12(b), the data read-hit ratio increased substantially when the associativity increased from direct-mapped to 2-way set. The change in read-hit ratio was not much in other transitions. For some benchmarks, the maximum read-hit ratio is attained for direct-mapped configuration after which there was negligible improvement in data read-hit ratio.



(shown left to right)

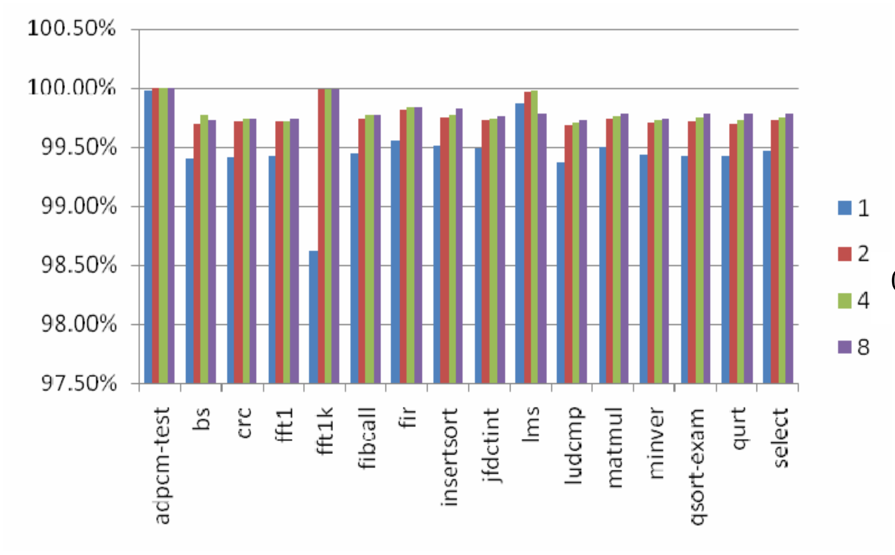
(a) Write Misses



(shown left to right)

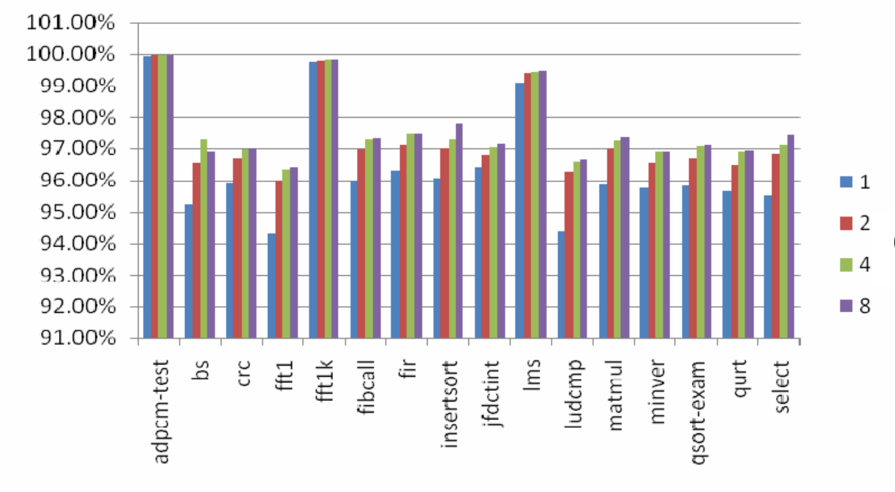
(b) Read Misses

Figure 11. Data Cache: Misses



(shown left to right)

(a) Write-Hit Ratio



(shown left to right)

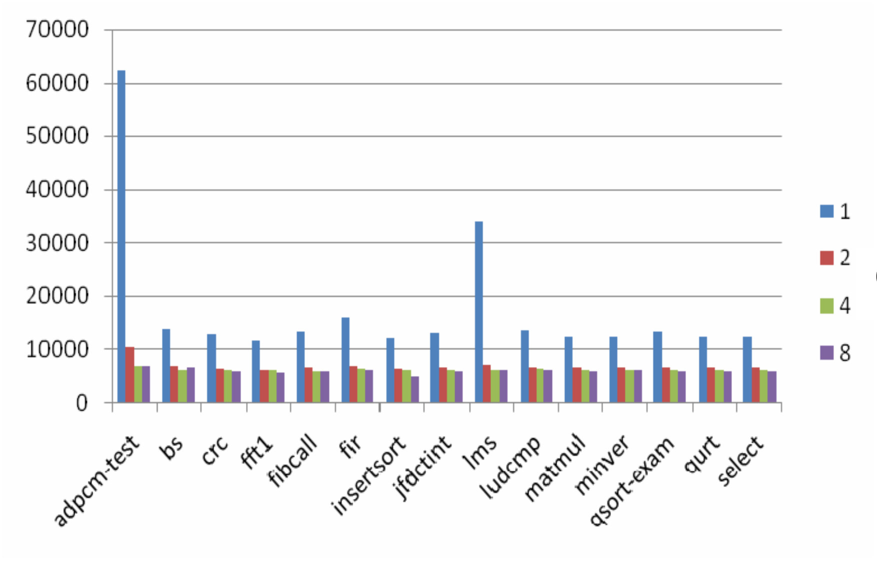
(b) Read-Hit Ratio

Figure 12. Data Cache: Hit Ratio

3.1.6.5 Instruction Cache.

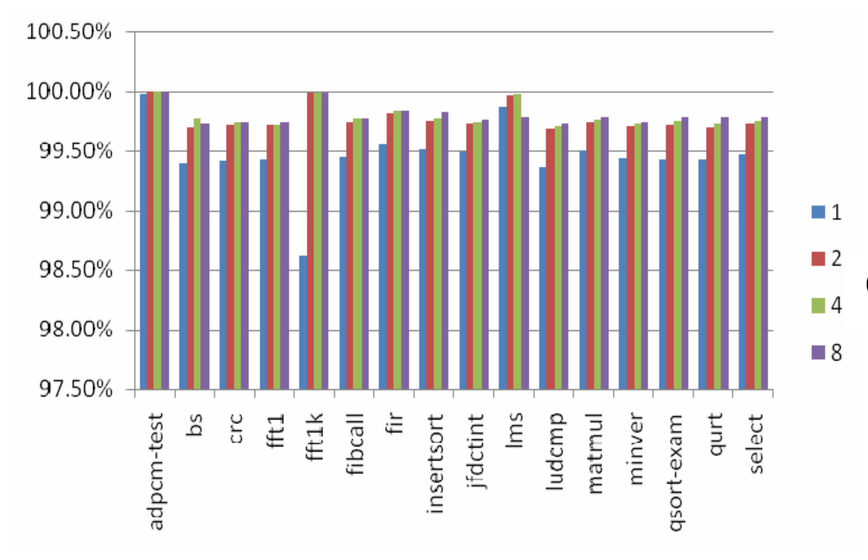
As shown in figure 13(a), the instruction fetch misses were reduced by approximately 50% when the associativity increased from direct-mapped to 2-way set. The move to higher associativity did not yield much improvement in the instruction fetch performance.

Figure 13(b) shows that there was not much difference in the instruction cache fetch-hit ratio for various associativity options.



(shown left to right)

(a) Fetch Misses



(shown left to right)

(b) Fetch-Hit Ratio

Figure 13. Instruction Cache Statistics

3.1.6.6 Simics Timing.

Simics is an event-driven simulator with a maximum time resolution of a clock cycle. In a single processor system, the length (in seconds) of a clock cycle is easily defined as 1 divided by the processor frequency set by the user. During each cycle, events are triggered and executed [6]. Events include device interrupts, internal state updates, and step executions. A step is the unit in which Simics divides the execution of a flow of instructions going through a processor; it is defined as the execution of an instruction, an instruction resulting in an exception, or an interrupt.

Steps are by far the most common events. Normally, one step is executed every cycle, so that step and cycle counts are the same. However, Simics timing can be extended by adding timing models to control the timing of memory operations, typically using the memory hierarchy interface. When timing models are introduced, steps are no longer equivalent to cycle counts. A step performing a memory operation (an instruction fetch or a data transaction) can stall for a number of cycles. Cycle events are performed during the stalling period as time goes forward. Step events are performed just before the step execution, as in the default model.

3.1.6.7 Simics Timing Configuration.

To evaluate execution time with cache penalty and memory latency, read and write penalty cycles for L1 and L2 cache and trans-staller in g-cache model (figure 9) should be configured. The read and write penalty cycles were already configured for cache simulation as in table 5. The trans-staller is a simple device that simulates the memory latency. It stalls all accesses by a fixed amount of cycles. The transaction-staller for memory is configured by the following declaration in the g-cache configuration script [6]:

```
@staller = pre_conf_object(‘staller’, ‘trans-staller’)
@staller.stall_time = 200
```

which indicates a stall time of 200 cycles.

3.1.6.8 Simulation and Result of Execution Time Evaluation.

To evaluate execution time for SNU real-time benchmarks, the routine ‘gettimeofday’ was used, which provides real execution time on Linux environment [14]. It offers microsecond resolution and can be used to estimate elapsed time as shown in the following C-code fragment in figure 14. All of the benchmarks have been modified by adding the C-code fragment.

```
#include <stdlib.h> /* definition of NULL */
#include <time.h> /* definition of timeval struct and protyping of gettimeofday */
double t1,t2,elapsed;
struct timeval tp;
int rtn;

main( )
{

rtn=gettimeofday(&tp, NULL);
t1=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;

 /* original benchmark code */

rtn=gettimeofday(&tp, NULL);
t2=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;
elapsed=t2-t1;
printf(“\n execution time : %f \n”, elapsed);
}
```

Figure 14. C-Code Fragment to Evaluate Execution Time

Table 6 shows execution times for all evaluated benchmarks under varying associativity of the L1 cache (instruction cache and data cache). The execution times are shown with microsecond resolution.

Table 6. Execution Time Evaluation

Benchmarks	Execution Time (second)			
	Associativity = 1	Associativity = 2	Associativity = 4	Associativity = 8
adpcm-test	2.461204	2.461313	2.461157	2.461542
bs	0.00002	0.00002	0.000021	0.000021
crc	0.000501	0.000501	0.000503	0.000502
fft1	0.001962	0.001962	0.001961	0.001995
fft1k	0.613021	0.61299	0.613156	0.613012
fibcall	0.000025	0.000025	0.000025	0.000025
fir	0.009675	0.009675	0.009677	0.009674
insertsort	0.000035	0.000036	0.000036	0.000035
jfdctint	0.000072	0.000073	0.000074	0.000072
lms	0.143179	0.142852	0.142854	0.142861
ludcmp	0.000983	0.000978	0.000982	0.000978
matmul	0.000068	0.000067	0.000066	0.000064
minver	0.000312	0.000313	0.000312	0.000315
qsort-exam	0.000131	0.000131	0.000131	0.00013
qurt	0.000382	0.000382	0.000382	0.000383
select	0.0001	0.000101	0.0001	0.000101

3.1.7 BAE Systems Simics Evaluation.

BAE Systems is evaluating the capabilities of Simics as a tool for software profiling support at the microarchitectural level and as a virtual software development environment for embedded software. Although related, each objective is being evaluated within its own study. Both studies are nearing completion. This section will describe the expectations from each study, a brief description of the task performed, and the conclusions.

3.1.7.1 Microarchitectural Profiling Support.

3.1.7.1.1 Objective.

The objective of this study was to evaluate Simics as a tool to gain insight into the operation of embedded flight-control software in a realistic system environment but at the microarchitectural level. This study was divided into two subtasks; one was to evaluate the modeling infrastructure, and the other was to evaluate the profiling support.

3.1.7.1.2 Assumptions and Expectations.

As a full-system simulator, the Simics model environment includes a model of the CPU, which is proprietary to Virtutech, and models of all the peripherals that comprise a single-board computer (SBC). Although Virtutech has a library of various standard peripheral models, Simics provides fairly extensive user support for modeling proprietary or new hardware, as necessary. It was expected that these features, combined with a robust network infrastructure support, would enable the target software to execute in a simulation environment that can span one or more SBC without needing any reconfiguration or adaptation. Thus, the simulation environment should be able to host systems at various granularities starting from a component consisting of a single SBC, a critical system consisting of multiple redundant SBCs, a complex system consisting of several directly interconnected cooperative systems, and even a system of systems interconnected through wide-area networks.

For the first subtask, the modeling aspect of the evaluation, BAE Systems wanted to verify the following:

- The intuitiveness of the model development toolset, which is an indication of the learning curve associated with the toolset.

- The expressibility and succinctness of the model description language. That is, ensure that the modeling environment can be used to describe desired properties, and that it can be described efficiently.

- The maintainability of the models developed.

In addition, as a system level simulation environment tool, BAE Systems expected Simics to provide easy access to system state information that is not easily visible in complex high-performance processors typically used in avionics systems today. Operating at the boundaries of a chip interface, traditional hardware-assisted monitoring approaches, such as in-circuit emulators, fail to provide access to even some basic state information, such as instruction and memory access traces, without significantly perturbing the target. As an example, branch-trace messages, a CPU hardware feature used to reconstruct an instruction stream, can slow down the performance of a processor enough so the system no longer meets its timeliness requirements. Similarly, other forms of instrumentation are often intrusive and may result in significantly altering the behavior of the target being monitored. This lack of visibility using traditional approaches is expected to become more severe with the next-generation avionics processors such as multicore computers and SoCs that have many of the traditional board-level logic components integrated directly in to the CPU.

For the second subtask, the profiling aspect of the evaluation, BAE Systems wanted to verify the following:

- The ease in extracting run-time profile data.

- The availability of interfaces to allow adding user-defined instrumentation.

- The ability to collect the data without impacting target behavior.

- The usability in an interactive development environment by ensuring that the time needed to execute the instrumented system is reasonable. Industry experience has demonstrated that cycle-accurate simulators are typically not practical for modeling realistic workloads, because of the tremendous amount of execution time required.

- The correctness or reasonable approximation of the collected data.

3.1.7.1.3 Tasks.

For the first subtask, BAE Systems modeled an internally developed PowerPC-based SBC used to host a modern digital flight-control system. The main modeled component was a BAE Systems proprietary bridge chip. The target model environment also needed to support the CsLEOS OS, the BAE Systems proprietary ARINC-653 compliant OS, and a proprietary board support package and bootstrap software.

For the second subtask, BAE Systems significantly enhanced the instrumentation in the provided g-cache model to report a variety of statistics at all granularities of the cache structure. BAE Systems also correlated trace data with cache behavior to gain insight into the operation of the embedded software.

3.1.7.1.4 Status and Conclusions.

At this time, the modeling subtask is complete and the profiling task is nearing completion. The Simics modeling environment was intuitive and expressive supporting a “C-like” modeling language along with Python language support. In addition, Simics’ proprietary scripting language, capable of executing within the command line interface, provided yet another language interface to the model execution. The extensive interfaces available allow a great deal of control of and insight into the model behavior. BAE Systems was able to instrument and profile virtually any model state parameter that the PMC was interested in to gain visibility into the microarchitectural behavior.

The main complaint about the Simics simulator is that it may be too flexible and expressible. The multiple language support and the numerous ways of accomplishing a task can lead to a level of confusion to a new user. Unfortunately, the current documentation does not do justice to the power and capabilities of Simics.

Also, Simics is not a cycle-accurate simulator and does not claim to provide microarchitecturally accurate timing behavior. Thus, high-fidelity performance evaluation using only the Simics environment is somewhat nonsensical. However, with the stall model, which is supported on a number of CPU models, Simics can be used to indicate and isolate interesting behavior that may be investigated further using different means. This ability to narrow the domain of search may be the key to making high-fidelity analysis feasible.

3.1.7.2 Virtual Software Development Evaluation.

3.1.7.2.1 Objective.

The objective of this evaluation was to use Virtutech's Simics model of a dual-core, processor-based SBC (ArgoNavis) for virtual software development prior to transitioning to actual hardware for final integration.

3.1.7.2.2 Tasks.

Four tasks were performed during the virtual software development evaluation. The first task was to develop bootstrap, drivers, test support functions, a board support package with Green Hills Software (GHS) INTEGRITY® Real-Time Operating System (RTOS), and performance-monitoring application software. The second task was to integrate these functions to the greatest extent possible in the simulated environment while assessing capabilities and limitations within the environment. The third task was to assess the level of change needed to transition the software to target hardware. The fourth task was to transition the integration of the software to target hardware, keeping track of problems found in hardware or software integration.

3.1.7.2.3 Status and Conclusions.

At this time, initial integration of the embedded software is nearly complete in the Simics environment. The Simics model provided very good support of most hardware features that were being used by the software that was developed. The environment provides a very powerful means of logging events occurring within the simulator to allow the user to debug problems with user software, as well as with models within the simulation. The completeness and fidelity of device ring functions (on-chip peripherals) needed some improvement, as well as the associated Simics documentation. Currently, the BAE Systems-developed embedded software can boot-up simulated cores and start-up the GHS INTEGRITY® RTOS that runs some basic user applications on each core. The Simics model has been configured to allow the GHS MULTI Integrated Development Environment (IDE) to be connected via simulated Ethernet interfaces to the RTSERV debug application that runs on each core, which allows for source line debug through the GHS MULTI IDE. The remaining task under this effort included making some minor changes to the BAE Systems developed software and then transitioning it to the actual dual core-based SBC hardware.

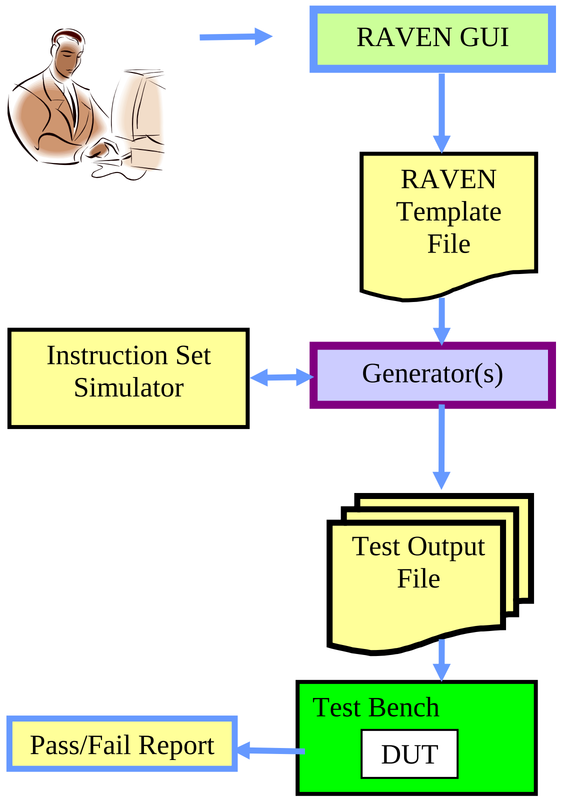
3.1.8 Lockheed Martin Simics Evaluation.

Lockheed Martin has also allocated resources to evaluate Simics capabilities. Like other participating members, Lockheed Martin has identified the need for additional ways to determine and understand the characteristics and functionality of complex microprocessors and SoC.

3.2 THE RAVEN EVALUATION.

3.2.1 What is RAVEN.

RAVEN is a random test program generator (RTPG) developed by Obsidian Software, Inc. (Austin, TX) [15]. This tool is intended for microprocessor verification by manufacturers. Models for microprocessor features are developed by Obsidian in collaboration with manufacturers. Figure 15 illustrates the basic test generation flow using RAVEN.



GUI = Graphical user interface
DUT = Device under test

Figure 15. The RAVEN-Based Verification Flow

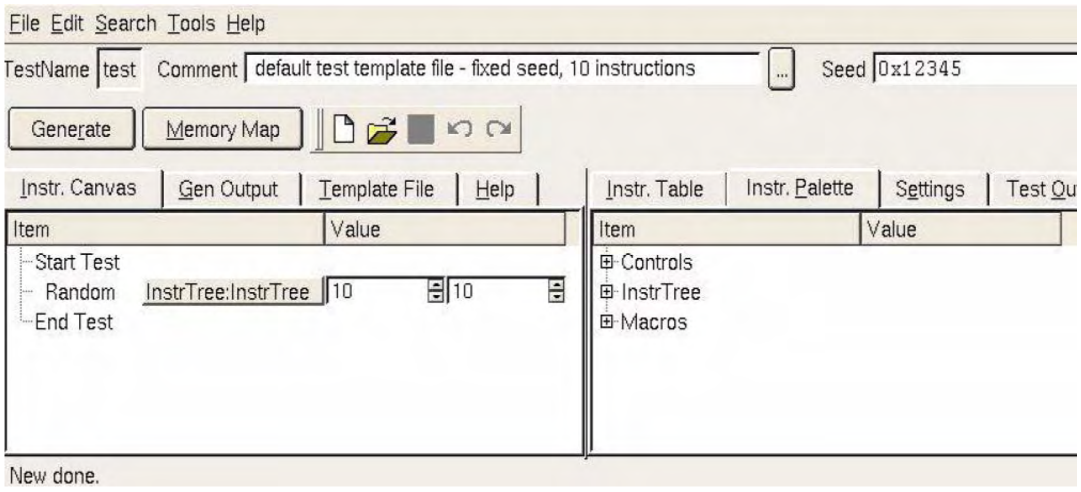
During Phase 2, the AFE No. 43 Research Team identified third-party tools that could possibly be used for microprocessor feature verification. Along with Genesys from IBM, RAVEN was also identified as a candidate for evaluation. The AFE No. 43 Research Team was provided with an academic version of RAVEN-ARM, a RTPG for the ARM processor family, for evaluation.

3.2.2 Using RAVEN.

User inputs into the RAVEN graphical user interface (GUI) are used to configure the RAVEN template file. This template file is input into the RAVEN generator. Based on the instructions selected in the template file, RAVEN generator interacts with the instruction set simulator and writes suitable memory and register images into the test output file. This test output file is then input into the test bench, which has the actual device under test. The test bench then compares the results produced by RAVEN with that obtained by actually running them on hardware (RTL) and generates a report file. The following section briefly describes how template files can be configured.

3.2.3 Configuring the RAVEN Template File.

Figure 16 shows a RAVEN screen shot of the instruction canvas being set to generate ten random instructions. The value has two fields, minimum and maximum. If the values were set to 5 and 10, respectively, then the number of instructions generated would lie between 5 and 10. In figure 16, both are set to 10 to generate 10 random instructions. The seed specified is given in the upper-right corner. Using different seeds will allow different tests to generate instructions each time the generate button is clicked. However, if it is expected to let RAVEN generate the seed, then a value of 0 can be entered in the seed field, and RAVEN automatically generates the seed for the user. If it is expected to emphasize on a particular instruction group during testing, this can be done using RAVEN by maximizing the InstrTree on the right side of the screen, as shown in figure 16.



TestName test Comment default test template file - fixed seed, 10 instructions Seed 0x12345

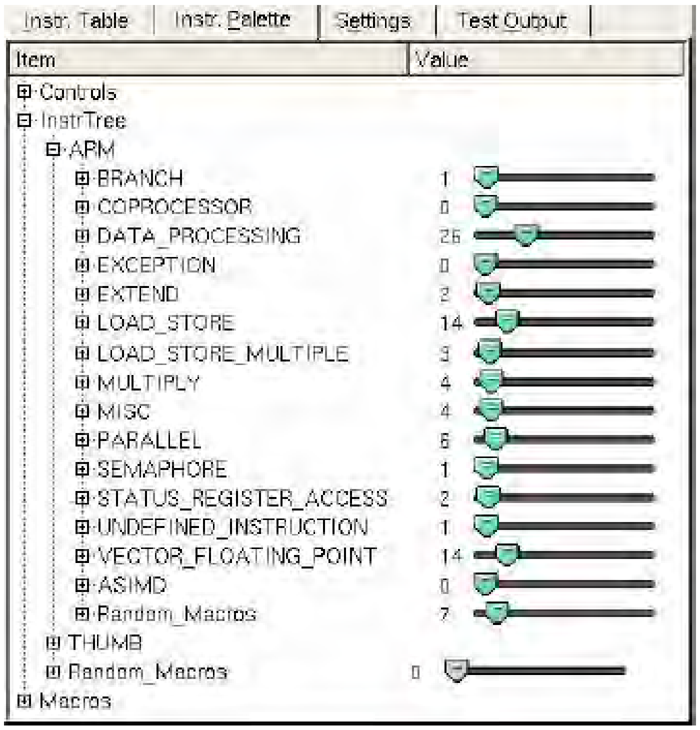
Item	Value		
Start Test			
Random	InstrTree:InstrTree	10	10
End Test			

Item	Value
⊞ Controls	
⊞ InstrTree	
⊞ Macros	

New done.

Figure 16. Setting RAVEN Seed

The instructions shown in figure 17 are for the ARM, Ltd. processor and will vary depending on the version of RAVEN. The values in the RAVEN screen shot in this figure indicate bias for random generation. For example, the data processing bias is set to 26, and the load store bias is set to 14. So if RAVEN picks ten instructions randomly, there is a higher probability of picking them from data processing instructions rather than load store instructions. However, if it is expected to test data processing instructions, set its bias to 100 and the biases of rest all instruction groups to 0. Such tests are called directed tests. Further, if a specific instruction is expected to be tested, further expand the instruction group and choose a high bias for that particular instruction. Once the test instructions are chosen, click on generate. This generates the template file with the necessary instructions based on biases. Once the generation is successful, the Test Output tab on the right pane gives the results of the generated test.



Item	Value
⊞ Controls	
⊟ InstrTree	
⊟ ARM	
⊞ BRANCH	1
⊞ COPROCESSOR	0
⊞ DATA_PROCESSING	26
⊞ EXCEPTION	0
⊞ EXTEND	2
⊞ LOAD_STORE	14
⊞ LOAD_STORE_MULTIPLE	3
⊞ MULTIPLY	4
⊞ MISC	4
⊞ PARALLEL	6
⊞ SEMAPHORE	1
⊞ STATUS_REGISTER_ACCESS	2
⊞ UNDEFINED_INSTRUCTION	1
⊞ VECTOR_FLOATING_POINT	14
⊞ ASIMD	0
⊞ Random_Macros	7
⊞ THUMB	
⊞ Random_Macros	0
⊞ Macros	

Figure 17. Setting Bias for Instructions to be Generated by RAVEN

RAVEN has several other features. For example, RAVEN uses randomly generated values for register initialization and reload. However, the user may want to initialize some or all registers with specific values or use specific reload values. This can be done by using the Settings/Registers function.

3.2.4 The RAVEN Generator Constructor Set.

To allow manufacturers to model features of microprocessors under design, Obsidian Inc. introduced the Generator Constructor Set (GCS) [16]. Using an architectural description language like syntax, features structure and operation can be specified and used to configure the RAVEN-GCS test program generator. The AFE No. 43 Research Team has not had an opportunity to work with RAVEN-GCS.

3.2.5 Capabilities of RAVEN.

RAVEN can be used for feature verification of microprocessors modeled for the RAVEN environment.

3.2.6 Limitations of RAVEN.

RAVEN uses models provided by Obsidian. These are developed in conjunction with manufacturers and are considered to be excellent reference models.

When using RAVEN-GCS, models will be developed using manufacturer-provided documentation, which limits the value of the models to its accuracy.

4. CONCLUSIONS.

Buffer-Oriented Microarchitectural Validation was selected as a candidate modeling technique for commercial off-the-shelf (COTS) microprocessors primarily because this technique does not require register transfer level implementation details. However, this study concludes that COTS microprocessor specifications and other publicly available documentation are inadequate for constructing these models due to ambiguous or insufficient information. The resulting modeling inaccuracies lead to incorrect microprocessor tests that do not assist in proving the safety of these components.

SIMICS, as a full-system simulator, allows for evaluation of application-execution performance. Verifying the abstracted models provided by Virtutech remains a challenge.

Using third-party microprocessor verification tools, such as Obsidian's Random Architecture Verification Engine, is effective when accurate models are used. Using manufacturer-provided documentation to model microprocessor features is once again limited by the accuracy of the data. Since this tool is intended for COTS microprocessor manufacturers, the availability of information is not a limitation. The value for user-side microprocessor verification using such tools is diminished by the unavailability of accurate models and complete data.

5. REFERENCES.

1. Utamaphethai, N., Blanton, R.D., and Shen, J.P., "A Buffer-Oriented Methodology for Microarchitecture Validation," *Journal of Electronic Testing,* Vol. 16, No. 1-2, pp. 49-65, February 2000.

2. Mahapatra, R.N., Bhojwani, P., and Lee, J., "Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure No. 43 Phase 2 Report," FAA report DOT/FAA/AR-08/14, June 2008.

3. Sun Microsystems, Inc., "OpenSPARC T1 Microarchitecture Specification," August 2006. http://opensparc-t1.sunsource.net/specs/OpenSPARCT1_Micro_Arch.pdf

4. Virtutech Corporation, "Simics Micro-Architectural Interface" (located in the /doc directory of the Simics software package), October 2006.

5. Mentor Graphics Corporation, "Seamless Hardware/Software Integration Environment," 2005. http://www.mentor.com/seamless

6. Brown, M.C., "Build a GCC-Based Cross Compiler for Linux," February 2005. http://www.ibm.comdeveloperworks/edu/l-dw-l-cross-i.html

7. Kegel, D., "CrossTool," December 2006. http://kegel.com/crosstool/

8. Virtutech Corporation, “Simics User Guide for Unix” (located in the /doc directory of the Simics software package), October 2006.

9. Virtutech Corporation, “Simics Ebony Target Guide” (located in the /doc directory of the Simics software package), October 2006.

10. SNU Computer Architecture and Network Laboratory, “SNU Real-Time Benchmarks,” http://archi.snu.ac.kr/realtime/benchmark/

11. Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes in C - The Second Edition*, Cambridge University Press, October 1992.

12. Embree, P.M., *C Algorithms for Real-Time DSP*, Prentice Hall, 1995.

13. Ahn, Hyun Soo, *Turbo C Programming for Engineering*, Kanam-Sa, 1992.

14. Texas A&M University Supercomputing Facility, “Timing and Performance,” July 2006. http://sc.tamu.edu/help/regatta/profiling.shtml#gettimeofday

15. Obsidian Software, “RAVEN Random Architecture Verification Engine,” http://www.obsidiansoft.com/images/pdf/datasheet.pdf

16. Obsidian Software, “The Generator Constructor Set: A New Approach to Random Test Generators,” http://www.obsidiansoft.com/images/pdf/RAVEN GCS Approach.pdf