SANDIA REPORT

SAND2012-7940
Unlimited Release
Printed September 2012

Leveraging Formal Methods and Fuzzing to Verify Security and Reliability Properties of Large-Scale High-Consequence Systems

Joseph R. Ruthruff, Robert C. Armstrong, Benjamin G. Davis, Jackson R. Mayo, and Ratish J. Punnoose

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2012-7940
Unlimited Release
Printed September 2012

Leveraging Formal Methods and Fuzzing to Verify Security and Reliability Properties of Large-Scale High-Consequence Systems

Joseph R. Ruthruff
Quantitative Modeling and Analysis Dept.
Sandia National Laboratories
P.O. Box 969, MS 9155
Livermore, CA 94551-0969
jruthru@sandia.gov

Robert C. Armstrong
Scalable & Secure Systems Research Dept.
Sandia National Laboratories
P.O. Box 969, MS 9158
Livermore, CA 94551-0969
rob@sandia.gov

Benjamin G. Davis
Information Assurance Dept.
Sandia National Laboratories
P.O. Box 969, MS 9011
Livermore, CA 94551-0969
bgdavis@sandia.gov

Jackson R. Mayo
Scalable Modeling & Analysis Dept.
Sandia National Laboratories
P.O. Box 969, MS 919
Livermore, CA 94551-0969
jmayo@sandia.gov

Ratish J. Punnoose
Weapons Subsystems 2 Dept.
Sandia National Laboratories
P.O. Box 969, MS 9110
Livermore, CA 94551-0969
rjpunno@sandia.gov

Abstract

Formal methods describe a class of system analysis techniques that seek to prove specific properties about analyzed designs, or locate flaws compromising those properties. As an analysis capability, these techniques are the subject of increased interest from both internal and external customers of Sandia National Laboratories. Given this lab's other areas of expertise, Sandia is uniquely positioned to advance the state-of-the-art with respect to several research and application areas within formal methods. This research project was a one-year effort funded by Sandia's Cyber Security S&T Investment Area in its Laboratory Directed Research & Development program to investigate the opportunities for formal methods to impact Sandia's present mission areas, more fully understand the needs of the research community in the area of formal methods and where Sandia can contribute, and clarify from those potential research paths those that would best advance the mission-area interests of Sandia. The accomplishments from this project reinforce the utility of formal methods in Sandia, particularly in areas relevant to Cyber Security, and set the stage for continued Sandia investments to ensure this capability is utilized and advanced within this laboratory to serve the national interest.

Acknowledgments

The authors thank Jerry McNeish for his comments on a draft of this report.

This project was funded in the 2012 Fiscal Year by the Cyber Security S&T Investment Area in Sandia National Laboratory's Laboratory Directed Research & Development (LDRD) program. Any views expressed in this report do not necessarily reflect those of the Cyber Security Investment Area Team, the LDRD office at Sandia, or Sandia National Laboratories as a whole.

This page intentionally left blank.

Contents

List of Figures

Nomenclature

DOE	Department of Energy
HPC	High-Performance Computing
IAT	Investment Area Team
LDRD	Laboratory Directed Research & Development
NNSA	National Nuclear Security Administration
NW	Nuclear Weapons

This page intentionally left blank.

Chapter 1

Introduction

Formal methods describe a class of system analysis techniques that seek to prove specific properties about analyzed designs, or locate flaws compromising those properties. These techniques implement advanced logic-based algorithms on abstract mathematical system representations to rigorously verify critical properties — e.g., “no code injections” or “if X happens then Y always follows.” Formal methods offer an improved technical basis for reliability and security assessments of digital systems, especially in high-consequence system design domains.

Sandia National Laboratories is taking a growing interest in the area of formal methods in order to provide the highest levels of verification for high-consequence hardware and software systems in its digital design domains. In addition to the application of formal methods in key mission areas, Sandia seeks to advance the state-of-the-art with respect to the research literature in formal verification so this capability can best address Sandia’s unique problem spaces and analysis requirements. In particular, as Sandia has in other domains, this laboratory seeks opportunities where it can leverage its existing expertise in other research and development areas to advance a separate but related domain — in this case, formal methods.

The Cyber Security Investment Area in Sandia’s Laboratory-Directed Research & Development (LDRD) program has been a leader in recognizing the need for, and investing in, building a research program in the area of formal methods. Since its inception, the Investment Area Team (IAT) for Cyber Security has carefully considered the most fruitful research paths within formal methods. These considerations have weighed both (1) the needs of the research community and (2) Sandia’s individual areas of expertise and where this laboratory may be best positioned to contribute.

Fuzz testing (also known as fuzzing) is a technique for providing random input to applications in order to evaluate reliability [13]. For some techniques and applications, this random input is structured to a degree in order to generate more meaningful responses from the application under test. This methodology has been shown to be useful, when deployed effectively, in various domains for identifying both reliability and security flaws in systems [5]. However, the random input selected during fuzz testing, both its content and structure, can determine how effective the methodology will be in discovering flaws in systems. This LDRD project (Oracle/Project Number 158744), as originally proposed, sought to inform and direct automated fuzz testing in order to improve this latter capability’s effectiveness in identifying reliability and security flaws in high-consequence systems.

The direction to a fuzz testing technique in the LDRD would be provided through the use of formal methods. The proposed research direction was to inform the input spaces targeted by fuzz testing using the results of formal verification, such as by focusing on inputs and execution paths identified by formal verification that produced out-of-nominal outputs and program behaviors. The Cyber Security IAT wished to clarify the utility and risks in any research path involving formal methods before making a significant commitment in this area of research.

Toward this end, this LDRD project was a modest one-year effort in the 2012 Fiscal Year to:

1. Investigate the opportunities for formal methods to impact mission areas of concern to the National Nuclear Security Administration (NNSA).
2. More fully understand the needs of the research community in the area of formal methods.
3. Clarify the potential research paths that would best fit the mission-area interests of Sandia National Laboratories, the NNSA, and the Department of Energy (DOE).

This report presents the outcomes and results of this project toward delivering on each of the three aforementioned items. In Chapter 2, we provide more information and background on formal methods. This information is provided in an effort to help decision-makers at all levels of management — at Sandia and outside this Laboratory – understand the importance, potential, and caveats of these verification capabilities. Chapter 3 presents the example problems we studied to investigate the utility of formal methods techniques in security contexts. Chapter 4 outlines some of the scalability limitations that must be considered when using formal methods, and some research directions that might be considered to address these concerns. Finally, Chapter 5 summarizes the key accomplishments of this project.

Chapter 2

Formal Methods Overview

Problem Space

Software and hardware systems have suffered from design flaws since their inception, many of which have been very serious and resulted in significant security and reliability consequences — especially for high-consequence systems. News of many of these events have reached the public domain. For example, in the 1980s, software flaws in the Therac 25 radiation treatment machine resulted in at least six accidents where patients were exposed to X-rays at approximately 100 times the intended dose when a particular combination of operator keys were pressed [2]. In 1994, a flaw in the Intel P5 Pentium floating point unit was independently discovered and publicly disclosed by a professor at Lynchburg College in Virginia [11]. This design flaw caused certain floating-point devision operations to produce incorrect results — according to Intel due to missing entries in a digital divide operation lookup table [16]. Intel later announced a pre-tax charge of $475 million against their earnings in order to replace the flawed processors after a recall was issued [20]. And in 1996, an unmanned Ariane 5 rocket exploded 40 seconds into its maiden flight because an assignment of a 64-bit number to a 16-bit buffer overflowed. This overflow caused the rocket controller to change its flight path, whereafter it disintegrated and self-destructed [10].

How should system designers safeguard against such design flaws? System testing and simulation is a widespread and effective technique for finding software and hardware flaws, especially those that appear frequently during observable program executions. One primary benefit of this strategy is that individual test cases can often directly validate one or more system requirements, ensuring that an implemented system design is capable of "doing what it is supposed to do." Unfortunately, testing is not particularly effective at ensuring that a program "will not do what it is not supposed to do." This deficiency stems from several reasons:

1. *Cost-benefit regarding amount of effort.* The effort necessary to uncover flaws in a system's behavior space is, at its roots, related to the frequency with which that flaw manifests itself into undesirable and observable system behavior. For example, hypothetically, an operating system bug that crashes Windows 7 on average every 1,000,000 hours of use would require approximately 1,000,000 hours of testing to discover. Is this a cost-effective strategy for discovering such a bug? At the same time, the same bug would affect a significant number of users: 12,000 every day[1] given the 600 million Windows 7 licenses that Microsoft

[1]One in 50,000 users every 24-hour day.

Corporation as announced it has sold [6].

2. *Coverage of system behavior is limited.* By definition, each test covers only a single path of possible program behavior. This makes covering the entirety of a system's behavior impractical for non-trivial systems. (For example, a system with 300 state variables has a state space size of 2^{300} states — larger than the number of protons in the universe[2].)

3. *Testing does not guarantee correctness.* As Edsger Dijkstra, winner of the 1972 Turing Award,[3] famously stated, "Program testing can be used to show the presence of bugs, but never to show their absence!"

Formal Methods

Formal methods refers to a body of verification techniques for digital designs — both hardware and software — that work by building a mathematical model of an artifact and proving properties about it [8]. As a field, formal methods has been an area of active research interest for over 60 years [19]. Several subareas of proof methods have emerged since that time, each with their own strengths and weaknesses in different problem domains. For example:

1. *Type checkers* prove data integrity properties of arbitrary programs during compilation in order to help ensure safe system execution during runtime [8].

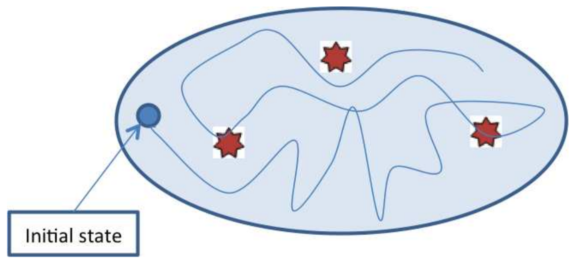
2. *Formal specification languages* provide mathematical descriptions of software or hardware on which an ultimate design implementation may be based. Unlike programming languages, specification languages describe what behavior a system should implement, but not how that behavior should be implemented in a system design. These languages can reveal inconsistencies, ambiguities, and incompleteness that might otherwise go undetected.

3. *Formal verification techniques* such as model checkers can exhaustively analyze a mathematical representation of a system's state space in order to provide proofs of correctness with respect to certain properties. These properties are most often categorized as *safety* properties that should never be true (e.g., "there should be no infinite loops in a software system") or *liveness* properties that should always be true (e.g., "an alarm sensor should always be active"). Such assertions can also be disproven with a verification tool identifying a counterexample to, or violation of, the correctness property.

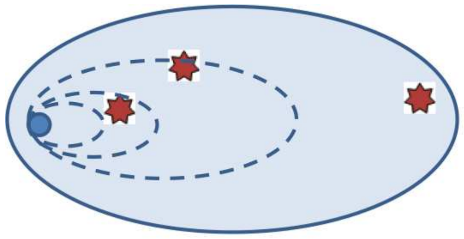
4. *Interactive theorem provers* support a user's search for a proof of correctness using tool-provided tactics. The tool automatically checks the user-generated proof to verify its correctness.

[2]The Eddington number: approximately 10^{80} [4]

[3]The Association for Computing Machinery's most prestigious technical award, and sometimes referred to as the "Nobel Prize" of Computing.



(a) Testing and simulation covering a deep but narrow path within a system's state space.



(b) Formal verification coverage expands outward, covering ever-larger portions of a state space.

Figure 2.1. Comparison of testing and simulation versus formal verification methodologies.

This LDRD project has focused primarily on the verification class of formal methods techniques. Traditional testing and simulation techniques are capable of deep explorations into a state space in search of bugs, albeit along the narrow path of a single program execution for each test (see Figure 2.1(a)). Thus, these techniques are input driven as the selected inputs for each test case drive the system execution path resulting from that input. In contrast, formal verification techniques exhaustively explore all areas of the reachable state space until there is nothing left to consider (see Figure 2.1(b)). Thus, these techniques are output driven as they seek to explore and analyze all possible outputs, and all areas of the possible state space, in order to reason about a program.

One advantage of formal verification tools is that they can be fully automated. These tools search for proofs of correctness with respect to certain properties, or counterexamples to asserted properties corresponding to design flaws, without the need for user interaction. Formal verification tools may be viewed as having a "lower cost of entry" compared to other classes of formal methods techniques such as theorem provers, in that once a model of a system exists and correctness properties of interest have been specified to the tool, the analysis tasks themselves are automatically performed. Finally, because of the automated nature of these techniques, they can be run on computing clusters, including high-performance computing (HPC) platforms, to support the verification of ever-larger systems. These advantages are all reasons why formal verification techniques are of interest to the problem spaces at Sandia, and synergistic with Sandia's existing solution capabilities (such as HPC).

However, the primary disadvantage of formal verification techniques stems from one of its strengths: its exhaustive verification of system state space in order to provide proofs of system behavior (or lack thereof). Formal verification techniques such as model checkers exhaustively explore state spaces to reason about behavioral properties of systems. Non-trivial systems, unfortunately, suffer from exponential explosions in the size of their behavioral state spaces, and despite research advances, the time and space complexity of formal verification techniques scales poorly with increasing problem size. Thus, this class of formal methods techniques commonly cannot verify realistic systems, such as microprocessors, software with cyber-security considerations, or complex problem scenarios such as those involving situational awareness.

Another key disadvantage of formal verification techniques apart from the scalability concerns is the information on which they operate. These techniques require a model of the system design under consideration, as well as a specification to the tool concerning the correctness property or assertion that is to be analyzed. In the case of the models, there are unfortunately very few existing capabilities for automatically building these models from the system design implementation. This means that these models are often designed separately from the design implementation of a system, meaning the models may be inaccurate or missing important information relevant to the analysis.

In the case of correctness properties, it is necessary for a formal verification tool to be instructed on what to look for to generate proofs of correctness for a system. For example, in the case of deadlocks for software systems, a tool must be properly instructed to look for a circular dependency between execution threads that are waiting on a resource before continuing execution. Security properties are often more difficult to express and check than a reliability example such as the foregoing case of deadlocks, because the conditions or vulnerabilities in a system that could lead to such security violations are often more exotic and difficult to express than traditional reliability properties. Also, the models must possess the semantic representativeness to provide the information needed for a formal analysis, which places a burden on both the language in which the model is specified and the designer creating that mathematical artifact.

Finally, it is worth drawing parallels between formal verification and other similar capabilities with which Sandia has experience and expertise. In particular, Sandia has been a pioneer in developing and implementing verification and validation methodologies to ensure the predictive capability and technical pedigree of conclusions drawn from continuum-physics models of engineered systems (e.g., [14]). Verification is a process to confirm that a continuum physics model is correctly computing results within the solution codes (i.e., it is implemented correctly, without errors), while validation is a process to confirm that modeling results are consistent with real world phenomena (i.e., the model is implementing the correct requirements).

When viewed from this perspective, the analysis performed by a formal methods tool is akin to the verification task, in that it ensures a system design is correct and error free — at least with respect to the correctness properties analyzed. Validation in this context, on the other hand, is what designers and analysts must do to ensure the mathematical model being analyzed (verified) is representative of the actual system design. The model that is actually analyzed by a formal methods technique is a mathematical model that may abstract many design details of the system. Formal equivalence checking is one methodology that can help ensure these system representations are equivalent to each other [9]. Figure 2.2 illustrates where such equivalence checking may fit into one hypothetical design and analysis workflow.

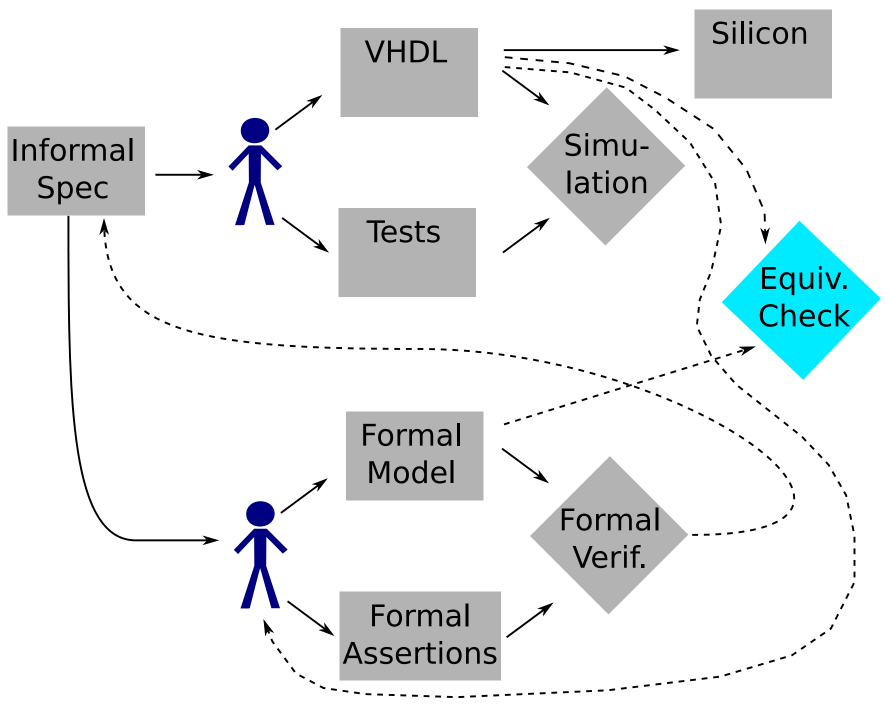


Figure 2.2. An example of equivalence checking (blue diamond) in a design and analysis workflow to validate the model created from a design specification for formal verification.

Chapter 3

Examples of Formal Verification

One key objective of this project as defined by the Cyber Security LDRD IAT was to survey and investigate the advantages and disadvantages of applying formal verification to security problems, and to provide the IA with a deeper understanding of formal methods and how they might map to cyber security problems. This chapter presents two example problems that were constructed and investigated to address this objective.

Rudimentary Security Problem: Bit-string Recognition

Design

The first example explored in this work involves a common security problem: bit-string recognition. At its roots, password recognition is a string recognition problem: verifying that a unique sequence of characters — and only that sequence — will provide access to a certain system resource. Since these letters, numbers, and symbols are ultimately represented in digital systems as sequences of binary bits, this is fundamentally a big-string recognition problem, and therefore models a rudimentary but fundamental cyber security problem.

It is also noteworthy that this problem also maps to recognition problems in analog domains — not just digital domains such as those involving password recognition. For example, in Sandia's Nuclear Weapons (NW) work, stronglinks are an important safety feature that serve as a sort of combination lock to prevent the usage of a weapon — in either normal or "abnormal" environments (e.g., scenarios where the weapon might be exposed to a fire) — unless deliberate action occurs [15].

Figure 3.1 illustrates how the bit-string problem can be fashioned after this analog domain. In the case of stronglinks, this domain is mechanical. When a stronglink receives a unique signal, it interrogates the signal to determine if it is correct, incrementally proceeding along the "tracking path" on a "pattern wheel" [15]. This tracking path can be thought of as a maze, with the unique signal providing the instructions to navigate the maze. As the signal is interrogated, if the next portion of the signal is correct, the stronglink mechanically proceeds along this tracking path. If at any point the signal is not correct, the component locks into a trapped state, rendering the weapon inoperable (top of Figure 3.1) [15]. The unique signal is typically composed of a series of electrical pulses (middle of Figure 3.1). The incremental up/down/neutral pulses of this signal can

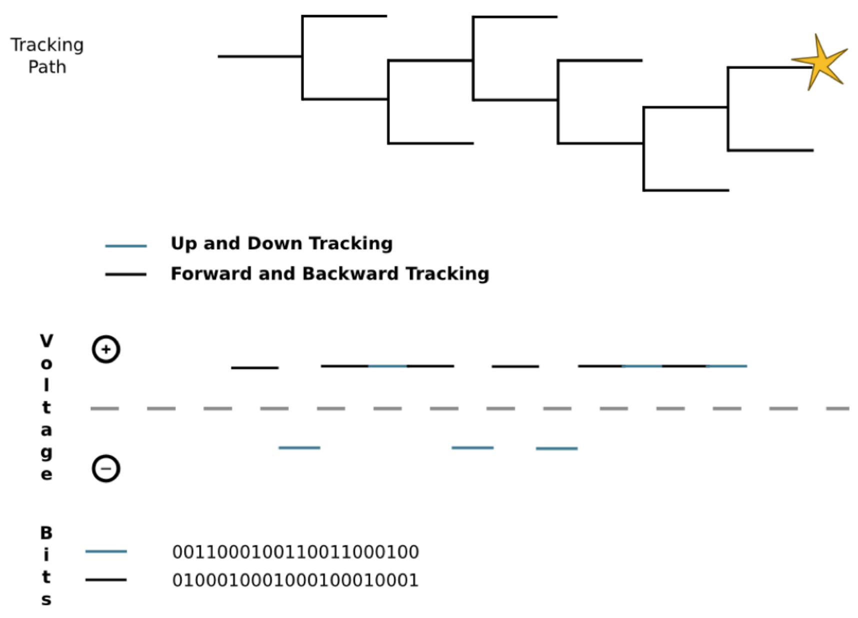


Figure 3.1. The bit-string recognition problem mapping to an analog domain in Sandia's Nuclear Weapons work.

be digitally represented by bits (bottom of Figure 3.1), allowing this "model" to be analyzed by a formal verification technique.

Consider a digital context where a circuit is cut to validate (i.e., recognize) input sequences such as for password recognition, or the model validation of a unique signal to be recognized by a stronglink. To analyze this example problem, an algorithm was developed to generate thousands of circuits "grown" using hybrid genetic programming. This algorithm was designed to generate circuits recognizing bit strings of up to 64 bits in length. It was also designed so that the generated circuits would retain realistic characteristics of human-engineered circuits. In particular, for relatively simple problems — i.e., those where a very short bit-string is to be recognized — exact solutions are obtainable, meaning the generated circuits will be perfect: they will properly accept exactly one bit-string, and reject all others. For relatively hard problems — i.e., those where very long bit-strings are to be recognized — the generated circuits would have rare design errors of less than 1%, meaning the circuit would improperly accept some bit-string sequences in addition to the correct sequence. Two reasons an automated approach was taken for generating these circuit models were to:

1. Create models that can automatically be formally analyzable — without human intervention.

2. Avoid potentially subjective translations by humans into a modeling language that would be suitable for formal verification

Results

To validate this model using formal verification, this project considered the use of model checking with the NuSMV open-source model checker [1]. NuSMV uses a modeling language that is similar to an HDL-level language. The circuits are represented as Boolean networks, thereby providing a representation approximately at the netlist level. The generated circuits are feed forward, thereby reminiscent of neural networks, and are therefore limited to low-complexity, rudimentary problem spaces (such as this one).

The results of using formal verification to validate these circuit models was convincing. For over 7,000 models, NuSMV had a perfect track record in:

- Proving that every error-free circuit was indeed error-free. The proof provided by NuSMV was an exhaustive exploration of the behavioral state space by the tool, identifying no cases where an invalid bit string was improperly recognized.
- Proving that every erroneous circuit with an error was indeed flawed. The proof provided by NuSMV of this flaw is a counter-example emitted by the tool showing the input improperly recognized by the circuit, and the behavioral path undertaken by the circuit during that erroneous recognition.

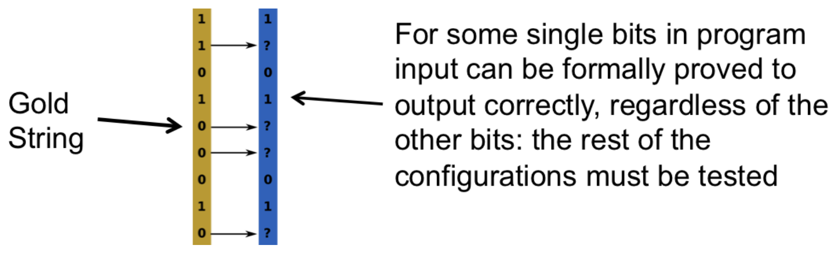
Furthermore, for each erroneous circuit, this investigation considered whether traditional testing and simulation would have uncovered the same flaw. This was performed by generating 1,000 random tests for each erroneous 32-bit circuit. In no case did any of these 1,000 tests identify the design flaws in the circuits. However, as stated above, formal verification via NuSMV identified the flaws in every case.

Exploring Synergy with Fuzz Testing

One direction explored for this example is possible synergy between the formal verification approach described earlier and fuzz testing. It was the assertion of this team that a fuzz-testing effort that is guided by insights gleaned from formal methods can provide utility to the fuzzer.

As stated earlier, and intuitively, a bit-string recognition program or circuit should recognize a single string, and reject all others. This string can be termed the “gold string.” Consider the case where a program or circuit designed to implement a bit-string recognition solution has design flaws, and other input strings other than the gold string are improperly accepted. One approach that could assist a designer’s effort to identify the specific flaw responsible for this erroneous behavior is to use formal methods to prove whether certain bits in an input string are responsible for the erroneous behavior in terms of the flawed design implementation.

Figure 3.2 illustrates this workflow. For any particular recognition program or circuit, certain single bits in an input string can be formally proven to be output correctly — that is, correctly set to that of the gold string — regardless of the rest of the bits. This effectively eliminates the possibility



Gold String
1
1
0
1
0
0
0
1
0
1
?
0
1
?
?
0
1
?
For some single bits in program input can be formally proved to output correctly, regardless of the other bits: the rest of the configurations must be tested

Figure 3.2. Example of using formal methods to guide an alternative methodology such as fuzz testing. In this case, formal methods could restrict a random fuzz search by eliminating bits that could be responsible for an observed system design flaw.

that the processing and recognition of those particular bits would be responsible for the erroneous acceptance of invalid bit strings, leaving a requirement that the rest of the configurations be tested. This ability to eliminate such bits means that the random search space explored by a fuzz-testing technique can be reduced to a more profitable space. For example, if it can be formally proven that 10 bits from a 64-bit gold string must be set for the string to be recognized, a fuzzing search can be improved 2^{10} or 1,000 times.

Rare High-consequence Events Example: Dual-zone Alarm System

Design

The second example explored in this project was also inspired by a similar problem observed in a real application in Sandia's mission space. This example concerns a dual-zone alarm sensor system. The design explored in this example is a simplistic representation of a system containing two alarm sensors to detect intrusions. When an intrusion is triggered in either zone, the respective sensor asynchronously updates an intrusion counter (*count*) in a non-volatile shared memory space. The system is designed to have controlled access to the memory to ensure that only one sensor at a time has write access to the counter.

In the actual application inspiring this example, a flaw found during the design phase of the dual-alarm system resulted in a race condition that can cause memory overwrites that would corrupt the intrusion count and result in an inaccurate number of intrusions being reported. An example of one of the many sequences of events that would cause this corruption is shown in Figure 3.3. In these cases, the counter would report a lower number of intrusions than actually took place.

From a verification standpoint, asynchronous, sequence-based events are very difficult to verify because the particular sequences that can cause problems can be rare, and the observable behavior fleeting in terms of the duration of time at which an error might be observed. Examples of this challenge are shown in Figure 3.4.

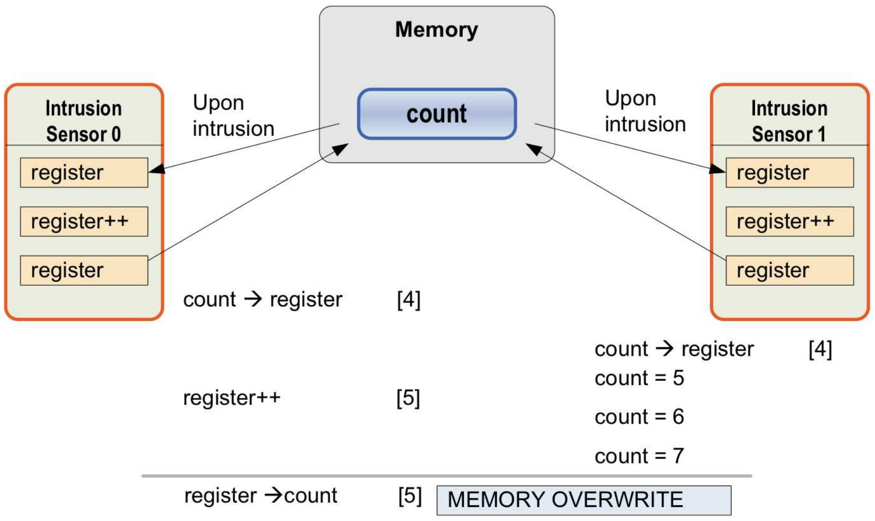
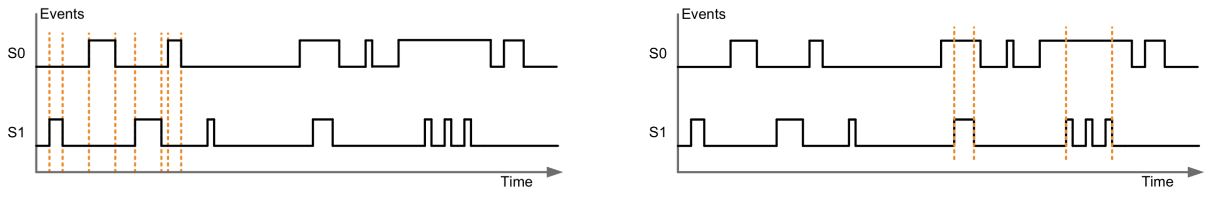


Figure 3.3. An illustration of how a race condition in the dual-alarm sensor system results in a memory overwrite that corrupts the intrusion count.



(a) Sequence-based events for two hypothetical alarm sensors *S*0 and *S*1.

(b) Interference where overlapping sequences may cause contention for the shared memory space storing the intrusion counter.

Figure 3.4. Sequences of events between two sensors in a hypothetical intrusion sensors, with possible asynchronous interference. In these examples, the upward ticks are intrusions detected by the sensors.

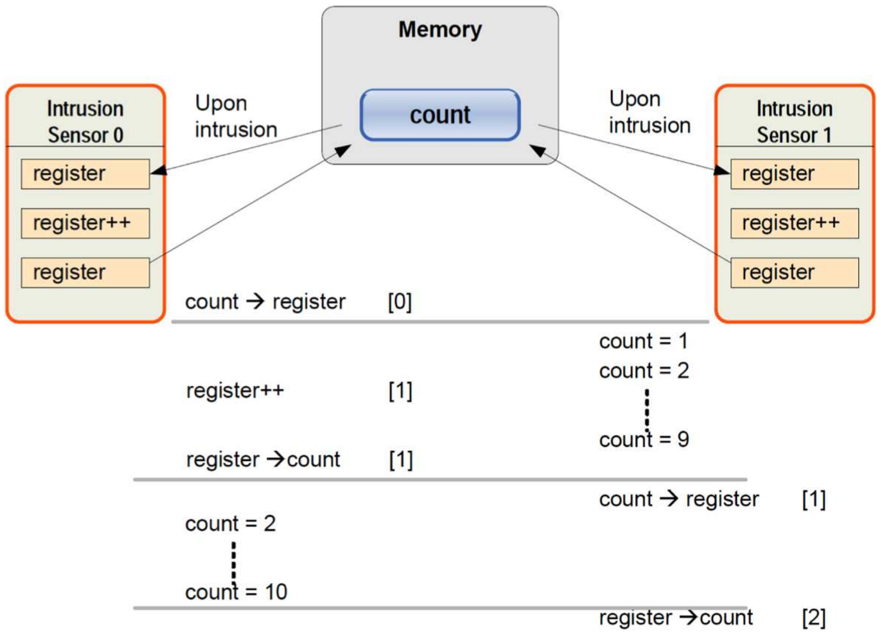


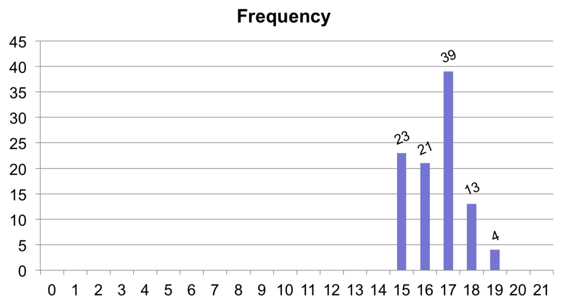
Figure 3.5. The lowest-probability scenario where the race condition in the dual-alarm sensor system results in an event where $count = 2$.

Traditional testing- and simulation-based approaches are not well-suited in verifying such systems because individual tests and simulations only scratch the possible behavior space of a system (recall Figure 2.1(a)), and are therefore unlikely to uncover rare events. It follows that they are especially unlikely to detected sequences of low-probability events. Formal verification techniques, on the other hand, are uniquely capable of identifying these events through the gradual but exhaustive verification of the entire behavioral state space (including all possible sequences of events), as illustrated in Figure 2.1(b).

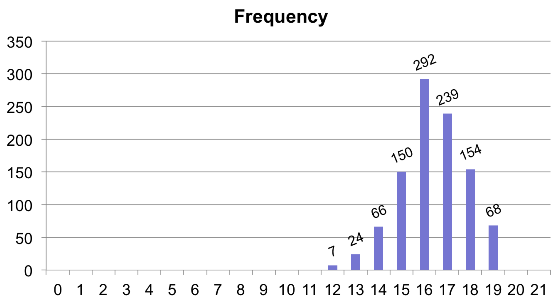
In this project, a comparison was performed regarding the separate abilities of testing and simulation versus formal verification to detect very low probability but potentially high consequence events associated with memory corruptions resulting in this race condition. Figure 3.5 illustrates the scenario corresponding to the "worst-case" sequence of events. In this scenario, 10 intrusions on each sensor, which should result in an intrusion $count = 20$, instead results in an intrusion $count = 2$.

Results

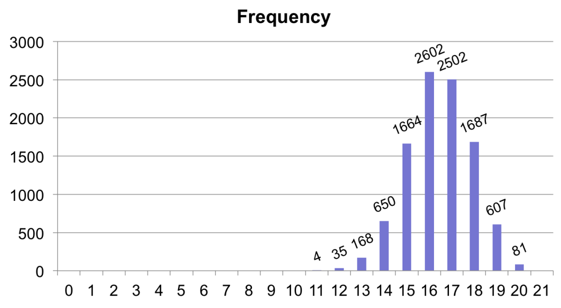
Formal verification was able to detect the presence of the race condition flaw in this system design, and the sequence of events leading to the "worst-case" memory space corruption of $count = 2$, within a few minutes of execution on a desktop workstation using a single processor. In particular, a Promela model [7] was built based on a version of this system's requirements and analyzed using



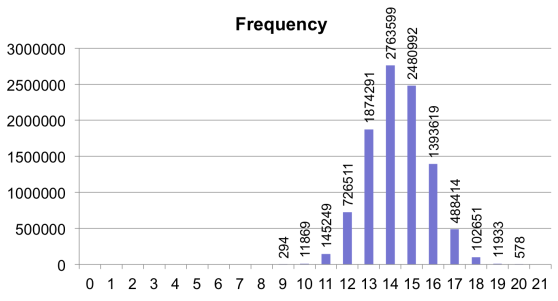
(a) 100 random simulations.



(b) 1,000 random simulations.



(c) 10,000 random simulations.



(d) 10,000,000 random simulations.

Figure 3.6. Four separate studies attempting to use random testing and simulation to identify the lowest-probability event (*count* $= 2$ on the horizontal axis) in the dual-alarm system.

the SPIN model checker [7] on that single processor.

In contrast, as Figure 3.6 illustrates, testing and simulation, using up to 10,000,000 randomly generated tests, was not able to identify this same "worst-case" sequence of events. In fact, as Figure 3.6(d) shows, the 10,000,000 simulations did not even come close to this worst-case scenario, finding sequences of events leading only to *count* being as low as 9. These 10,000,000 simulations took approximately one hour to run on the same workstation.

Chapter 4

Scalability Concerns Regarding Formal Verification

As mentioned earlier, within the class of formal methods techniques, formal verification algorithms rigorously verify the absence of specified flaws in hardware and software systems. As such, despite their use of abstract mathematical representations of a system to verify critical system properties, these techniques inherently suffer from scalability limitations that restrict their ability to reason about ever-larger and more complicated systems. The fundamental reason for this is an explosion in the size of a system's potential behavioral state space that must be analyzed and reasoned about. These state spaces most often grow exponentially for non-trivial digital designs. As a simple example of this type of growth behavior, a microprocessor with 1,000,000 transistors has $2^{1,000,000}$ possible states. By increasing the exponent, it is easy to see how adding transistors exponentially increases the theoretical state space size.

Which formal verification techniques should be focused on to deliver meaningful analyses for digital systems despite these challenges? Perhaps the most widespread formal verification technique is model checking, which explicitly and exhaustively explores state spaces to reason about behavior properties of finite-state systems. Because non-trivial systems suffer from an exponential state-space explosion, rigorous simplifications such as symmetry and partial-order reductions are used by the algorithms to collapse state spaces to manageable sizes. Application of increased computational capabilities has also made it feasible to use model checking to analyze increasingly large problems.

However, the time and space complexity of model checking still scales poorly with increasing problem size. This commonly puts realistic systems, especially those with complex design architectures or complicated correctness properties to verify — e.g., systems with important cybersecurity considerations such as scenarios involving situational awareness — out of the reach of model checkers. In fact, such scalability considerations are the primary reason model checking has not been more widely adopted for the verification of both software and hardware systems.

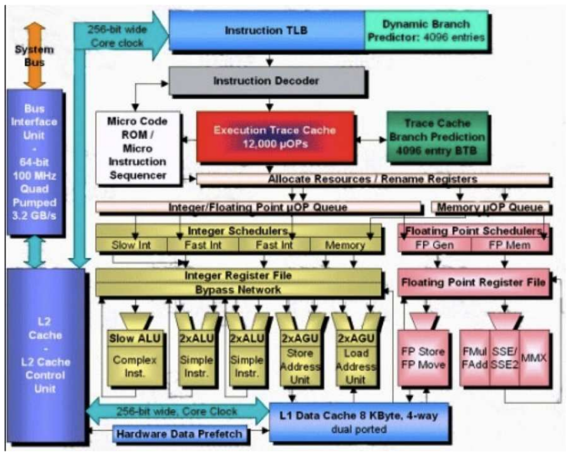
For this reason, the research community has sought to improve the scalability of model checking techniques since their inception. Ideally, model checking techniques would be capable of distributed state space exploration using parallel operations on computing clusters with very large numbers of processors. (Of course, other meaningful improvements can be imagined, but the ability to run on thousands of processors as a high-performance computing code, cannot be dismissed.) However, in the formal methods research literature, parallel and distributed model checking is a generally unsolved problem, whose difficulty derives from contention for the shared representation of the state space. Although parallel and distributed programming frameworks have been proposed for some model checking techniques (e.g., [17, 12]), to date there remains no technique capable of scaling to large computing clusters with thousands of processors, and the research community has not seen a meaningful improvement in this area in years.

If capabilities for fully parallel and distributed formal verification existed, they might be viewed as a top-down approach to verification in that the analysis would take place at the full system design level and explore the resulting state spaces at the entry points to system execution. An alternative strategy for addressing scalability is what might be viewed as a bottom-up approach, where verification takes place for individual components or subsystems. The results from these separate analyses would then be composed into increasingly large assessments until the entire system can be reasoned about in a mathematically sound fashion given the individual verifications performed at lower levels of the design. This is a research need called out by the White House National Science and Technology Council as follows: "The research challenges of this theme include . . . (m)athematically sound techniques to support combination of models and composition of results from separate components" [18].

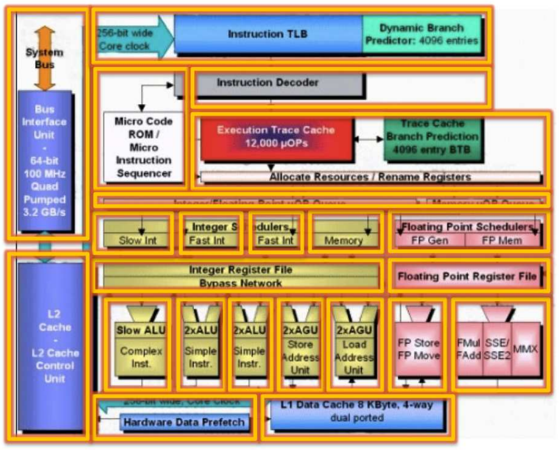
Figure 4.1 illustrates how this approach may work in practice when developed by supporting research. Intel uses formal methods as their principal verification strategy for their microprocessors, including the i7 Core Processor. While the authors of this report do not have detailed knowledge about how formal verification is specifically integrated into Intel's design and analysis workflows, it is safe to surmise that Intel cannot analyze the entire microprocessor at once, as it is known that microprocessors are generally far too large to analyze with formal verification due to their large state spaces — particularly for general-purpose commodity processors. Therefore, Intel and similar entities must verify individual components and subsystems, thereby presenting the opportunity stated above to compose those results into an analysis of the system. In short, composing results would be highly synergistic with the current workflows already utilized by designers and analysts, who already have to verify components or subsystems individually due to scalability restrictions, but would like to reason about the full system if that were possible.

Other than these approaches, the current state-of-the-art includes some model checking techniques that can operate in parallel on a small scale. For example, SPIN is one model checker that has been extended to support multi-core processor architectures [7]. The "portfolio" method is an alternative strategy that attempts many different configuration options for formal method techniques — not just model checkers — in an embarrassingly parallel attempt to determine the optimal configuration for any particular problem [3].

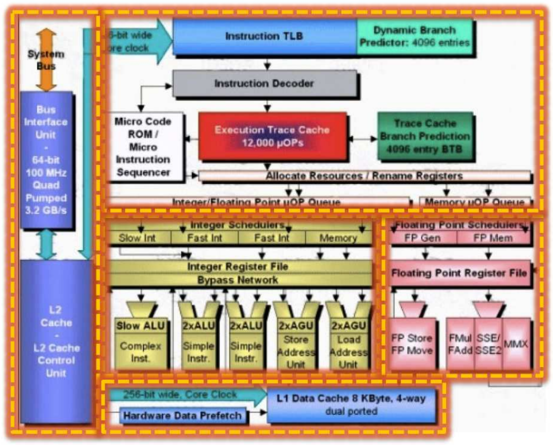
These methods, while important advances in the research literature, fall short of a fully parallel and distributed approach that scales to thousands of processors, which would allow model checking techniques to be brought to bear on increasingly complex problems, such as those pertaining to cyber-security. They also do not attempt to address the challenge of composing formal analyses from small portions of a system into a rigorous analysis for the system as a whole. However, a project seeking to address this latter challenge has been funded by the Cyber Security Investment Area in FY13 within Sandia's LDRD program.


System Bus
Bus Interface Unit - 64-bit 100 MHz Quad Pumped 3.2 GB/s
256-bit wide Core clock
Instruction TLB
Dynamic Branch Predictor: 4096 entries
Instruction Decoder
Micro Code ROM / Micro Instruction Sequencer
Execution Trace Cache 12,000 µOPs
Trace Cache Branch Prediction 4096 entry BTB
Allocate Resources / Rename Registers
Integer/Floating Point µOP Queue
Memory µOP Queue
Integer Schedulers
Slow Int
Fast Int
Fast Int
Memory
Floating Point Schedulers
FP Gen
FP Mem
Integer Register File Bypass Network
Floating Point Register File
L2 Cache - L2 Cache Control Unit
Slow ALU Complex Inst.
2xALU Simple Instr.
2xALU Simple Instr.
2xAGU Store Address Unit
2xAGU Load Address Unit
FP Store FP Move
FMul FAdd
SSE/ SSE2
MMX
256-bit wide, Core Clock
L1 Data Cache 8 KByte, 4-way dual ported
Hardware Data Prefetch

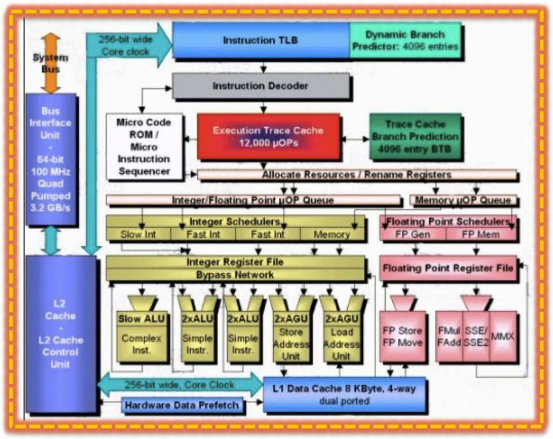
(a) High-level layout of the Intel i7 Core Processor.



(b) Hypothetical individual verification analyses of microprocessor components (orange boxes).



(c) Verification of subsystems by composition from the individual analyses (dashed orange boxes).



(d) A full-system verification by composition from individual analyses.

Figure 4.1. Composing formal analyses from one hypothetical verification workflow for Intel's i7 Core processor. The orange boxes denote verified areas of the design — in the latter stages via composition from previous results (dashed orange boxes).

Chapter 5

Conclusions

The purpose and goals of this one-year LDRD project were to survey and investigate the advantages and disadvantages of applying formal methods to problem space of relevance to the Cyber Security Investment Area. It was also desired to provide the Cyber Security IAT with a deeper understanding of formal methods more generally, particularly with respect to the scalability of the techniques and their applicability to security problems.

Cyber Security at Sandia has broad reach across the laboratory, impacting work in Homeland Security, Energy Infrastructure, Nuclear Weapons, and work for others. As such, this project team examined two hypothetical security problems that were designed to provide broad insights into the applicability of formal verification techniques into these problem spaces. The selected problems were inspired by real problem instances observed by team members in the aforementioned mission areas. The results provided by these example problems are quantitative, technically sound, and convincing — supporting growing acknowledgment of the applicability of formal methods both internally within Sandia and externally, including Sandia customers.

This project also explored possible synergy between formal methods and fuzz testing to clarify future research paths in this direction. Finally, this project identified opportunities for advancing the scalability of formal verification techniques. Many of these research opportunities are synergistic with Sandia's strengths and targeted to the needs of both the larger research community and the national security interests of the country.

Several efforts within this laboratory have momentum as a result of this project. First, a Sandia team with expertise in formal verification is well-positioned to deliver state-of-the-art research results in composing formal analyses in a new three-year LDRD project that is starting in FY13. As discussed in Section 4, this research direction has synergy with the design and analysis workflows already being utilized to employ formal methods given the existing scalability concerns of those techniques. Second, this team and others at Sandia are prepared to build additional research programs in other formal methods areas, especially concerning the scalability of the techniques. In fact, this team in particular has several research ideas that will be developed and proposed in the months to come.

Third, we have broadened our connections with industry and academic institutions working on research and development areas in formal methods. This has occurred through separate outreach efforts as well as connections made at research conferences. We are actively considering collaborating with some of these parties in our formal methods research projects beginning in FY13.

Fourth, members of our team have used the knowledge and expertise they have acquired through their involvement in this project to provide positive and immediate impact to other projects in Sandia's mission areas. In particular, some of this project's team members were approached to separately contribute to the analysis and verification of the in-progress design on an NW subsystem. These researchers have developed and conducted novel analyses that have resulted in key improvements to that NW subsystem. These design improvements will increase the reliability of the NW subsystem under both normal and abnormal conditions, and illustrate the clear impact of capabilities fostered by LDRD to improve design and verification processes in Sandia's NW mission area. The capabilities developed to conduct this analysis are rooted in both new research directions that drive the state-of-the-art with respect to verification of high-consequence systems, and effective methodologies and frameworks to seamlessly integrate new verification practices into NW design efforts. Although this LDRD project did not support the development and application of these capabilities to this NW subsystem, the support of the Cyber Security LDRD IA should be credited with investing in developing the knowledge base and expertise in formal methods that allowed these capabilities to be developed. Without these capabilities supported by LDRD and the close collaborative partnership between those researchers with NW designers, identifying the design opportunities for improvement in this NW subsystem would not have been possible.

Finally, this project has positioned Sandia to continue identifying and implementing formal methods for security properties and complex cyber security scenarios, given the unique and challenging problem spaces in Sandia's mission areas. There is active interest at present in the ability of formal methods to improve the security and reliability of engineered digital systems — hardware and software — and Sandia is no exception to this trend. Given this lab's unique areas of expertise, Sandia is well positioned to advance the state of the art in formal methods in the years to come.

References

[1] NuSMV: A new symbolic model checker. http://nusmv.fbk.eu (Last Accessed: 13 Sept. 2012).

[2] S. Baase. *A Gift of Fire*. Pearson Prentice Hall, 2008.

[3] M.B. Dwyer, S. Elbaum, S. Person, and R. Purandare. Parallel randomized state-space search. In *Proceedings of the* 29th *International Conference on Software Engineering*, pages 3–12, Minneapolis, MN, U.S.A., May 2007.

[4] A. Eddington. *The Philosophy of Physical Science*. Cambridge University Press, London, 1939.

[5] J.E. Forrester and B.P. Miller. An empirical study of the robustness of Windows NT applications using random testing. In *Proceedings of the* 4th *USENIX Windows Systems Symposium*, pages 59–68, Seattle, WA, U.S.A., 2000.

[6] S. Guggenheimer. Computex 2012 keynote address. Microsoft Corporation, June 2012. http://www.microsoft.com/en-us/news/Speeches/2012/06-06Computex2012.aspx (Last Accessed: 13 Sept. 2012).

[7] G.J. Holzmann and D. Bonacki. The design of a multi-core extension of the SPIN model checker. *IEEE Transactions on Software Engineering*, 33(10):659–674, October 2007.

[8] J. Hurd. Formal methods overview. In *Industrial Formal Methods Course*. Galois, Inc., 2012.

[9] A. Koelbl, Y. Lu, and A. Mathur. Formal equivalence checking between system-level models and RTL. In *Proceedings of the 2005 IEEE/ACM International Conference on Computer-aided Design*, pages 965–971, San Jose, CA, U.S.A., November 2005.

[10] J.L. Lions. ARIANE 5: Flight 501 failure. Report by the Inquiry Board, July 1996. http://www.di.unito.it/ damiani/ariane5rep.html (Last Accessed: 13 Sept. 2012).

[11] J. Markoff. Flaw undermines accuracy of Pentium chips. In *The New York Times*. November 24 1994.

[12] I. Melatti, R. Palmer, G. Sawaya, Y. Yang, R.M. Kirby, and G. Gopalakrishnan. Parallel and distributed model checking in Eddy. *International Journal on Software Tools for Technology Transfer*, 11(1):13–25, January 2009.

[13] B.P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12), December 1990.

[14] W.L. Oberkampf, T.G. Trucano, and C. Hirsch. Verification, validation, and predictive capability in computational engineering and physics. Technical Report SAND2003-3769, Sandia National Laboratories, Albuquerque, NM, U.S.A., February 2003.

[15] D.W. Plummer and W.H. Greenwood. A primer on unique signal stronglinks. Technical Report SAND93-0951, Sandia National Laboratories, Albuquerque, NM, U.S.A., August 1993.

[16] FDIV Replacement Program. Statistical analysis of floating point flaw: Intel white paper. Technical Report CS-013005, Intel Corporation, July 2004. http://www.intel.com/support/processors/pentium/sb/cs-013005.htm (Last Accessed: 13 Sept. 2012).

[17] Robby, M.B. Dwyer, and J. Hatcliff. Bogor: An extensible and highly-modular software model checking framework. In *Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 267–276, Helsinki, Finlind, September 2003.

[18] National Science and Technology Council. *Trustworthy Cyberspace: Strategic Plan for the Federal Cybersecurity Research and Development Program*. Executive Office of the Presiden, Washington, D.C., U.S.A., December 2011.

[19] A.M Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69. University Mathematical Laboratory, Cambridge, England, June 1949.

[20] N. Wingfield. Chip firms no longer ignore even the least offensive bugs. In *The Wall Street Journal Interactive Edition*. http://online.wsj.com/article/SB879552760162274000.html (Last Accessed: 13 Sept. 2012).

DISTRIBUTION:

1	MS	1327	William Hart, 1464
1	MS	0621	Dallas Wiener, 5632
1	MS	9110	Ratish Punnoose, 8229
1	MS	9110	Paul Yoon, 8229
1	MS	9158	James Costa, 8950
1	MS	9152	Robert Clay, 8953
1	MS	9159	Jackson Mayo, 8953
1	MS	9159	Jerry McNeish, 8954
1	MS	9155	Joseph Ruthruff, 8954
1	MS	9151	Robert Hutchinson, 8960
1	MS	9158	Robert Armstrong, 8961
1	MS	9158	Keith Vanderveen, 8961
1	MS	9011	Benjamin Davis, 8965
1	MS	9011	Navid Jam, 8965
1	MS	0899	Technical Library, 8944 (electronic copy)
1	MS	0359	D. Chavez, LDRD Office, 1911

Sandia National Laboratories