Configure, Confirm, Ship: Build Secure Processor-Based Systems with Faster Time-to-Market

Authors

Nicole Fern
Senior Hardware Security Engineer, Tortuga Logic

Ruud Derwig
Senior Staff Engineer, Synopsys Inc.

Abstract

Security is a first-order design requirement for processor-based systems. Processor designers implement security functionality directly into the hardware itself to protect the system at its most fundamental layer. System integrators that use processor IP such as Synopsys' DesignWare® ARC® processors must ensure that they configure and manage the protection and security features correctly, and that they do not introduce vulnerabilities. Evaluating the security of complex, highly combined hardware-software systems and ensuring these systems are free from vulnerabilities is hard. In this white paper, we show how Tortuga Logic's Radix-S security verification platform with Synopsys' ARC Processor IP offer a powerful solution for this complex problem. We demonstrate the combined hardware-software security verification by creating an example system comprised of the ARC processor IP and vulnerable software that configures the memory protection unit incorrectly. With the additional capabilities provided by Radix-S, we identify the flaw using pre-existing functional verification infrastructure. Furthermore, we show how system integrators can verify the security of protected debug logic with this technology.

Introduction

The Growing Importance of Security in Today's Processor-based Systems

Processor-based systems are proliferating rapidly, with over 1 trillion semiconductor devices sold in 2018 [1]. Processors are customized and used in new applications such as the Internet-of-Things (IoT), autonomous vehicles, artificial intelligence (AI), customized datacenter workloads, and more. We trust these systems to control and manage many aspects of our lives including the secure storage and transmission of financial data, protection of encryption keys, and safeguarding identifiable personal information and biometric data such as facial recognition characteristics and fingerprints. We also rely on these systems to keep us safe by governing home security and critical infrastructure, making the need for system security more important than ever. While software was once the main provider of security, system developers are realizing that true system security begins with hardware. Without hardware security, even the most secure software base will not be effective. Processor designers therefore employ hardware roots of trust, which are widely considered to be the most state-of-the-art tools for hardware security. A root of trust is a minimal amount of hardware (and its accompanying software) that performs security-critical functions to protect the entirety of the system from security exploits. Hardware roots of trust can perform device authentication, on-chip encryption, and bringing the system to a known and trusted state. Hardware roots of trust are used as the basis for security in all market verticals. Along with additional security features baked into a processor, hardware roots of trust are a necessary first step towards system security.

A Secure Development Lifecycle for Hardware

While security is important and implementing roots of trust is a common strategy, the verification of security features has not caught up in maturity. Hardware security verification often relies on manual efforts such as design architecture and code review or are applied to only one part of the design lifecycle. These efforts are a good start but do not encompass a comprehensive strategy to find most security vulnerabilities.

To effectively and methodically find security vulnerabilities, the hardware world can learn from the software world and implement a Secure Development Lifecycle (SDL). The SDL process was invented by Microsoft to systematically improve and test the security of software under development [2]. In the software world, an SDL provides a well-defined framework with best practices and best-in-class tools to create code that is significantly more secure than code developed without an SDL. Synopsys' Software Integrity products and services [3] for application security testing, composition analysis, and fuzz testing are well known and help maximize application security and quality without slowing down development for the software world. But what about the hardware world?

In the hardware world today, an SDL is not defined nor is it utilized by most hardware designers. A hardware SDL would specify a security threat model at its onset, then design, develop, and verify the hardware to protect against that threat model. An effective hardware SDL needs to consist of both 1) security verification at each stage of the design lifecycle, and 2) recommendations for types of tools needed to maximize the effectiveness of the process. Security must be checked at every stage of the design and with different tools, because different types of vulnerabilities can be introduced throughout the process, and some tools are better suited for detection at those times. A security vulnerability existing early on in the design lifecycle and caught at a much later stage than it was introduced can increase the cost of fixing that vulnerability by up to 30 times [4]. Also, state-of-the-art tools can automate the most complex analyses that would be error-prone and unreliable if performed manually. If a hardware SDL had been in place with both security verification for each stage and state-of-the-art tools, it would have greatly decreased the likelihood of recent security vulnerabilities like Meltdown and Spectre making it through to the final system.

Security is a System Requirement

Security verification has historically focused on software layers of the computing stack, from firmware and above, often not considering the underlying hardware implementations. As breaches can occur at any layer, security is not a property of a single layer in the computing stack but spans from hardware, to firmware, to the operating system, and to application software. Ensuring the security of the whole system requires the analysis of both the hardware and software. Looking at one alone is insufficient for providing the strongest security guarantees.

The Meltdown and Spectre attacks were a testament to the type of security vulnerabilities that can arise when the hardware and software are not analyzed together. The sophistication of these attacks spanned both microarchitecture processor features and the software executing on the processors. Validating functional correctness of hardware/software systems already requires enormous amounts of simulation, hardware emulation, and FPGA prototyping time. For modern processors, the interaction between hardware and software is complex and difficult to reason about or analyze with respect to security.

The most effective approaches to this daunting problem introduce technologies that identify security vulnerabilities using a high degree of automation while leveraging established functional verification environments. In doing so, end products can be engineered to have the highest levels of security with the least disruption to design schedules. Tortuga Logic's Radix-S product provides such a capability for ARC Processor IP, including verification of the ARC SecureShield™ Technology for creating trusted execution environments (TEEs) that do not leak secrets. System threat models can be specified and verified within existing functional verification environments that span both hardware and software. Radix-S provides a framework for ensuring that ARC Processor features are correctly configured and used, and that the processor is securely integrated into a larger system. For processors, it is not enough to verify the hardware alone. It is the software that ultimately determines the behavior of the processor and system. A significant part of the security-relevant configuration, such as configuring a memory protection unit (MPU) to prevent unprivileged read access to memory containing confidential information, is done by runtime software. Therefore, an effective security verification solution must cover both hardware and software.

DesignWare ARC Processor IP and SecureShield Overview

In today's products, software implements the bulk of customer facing functionality and consequently is the first target for attackers trying to circumvent product security—this is one of reasons why SDL processes started in the software world. However, system security cannot be guaranteed without a trusted hardware base. Without a TEE running the software that implements security functions such as secure boot, network authentication, and payment services, even the most secure software cannot be guaranteed. For example, the software could be altered by an attacker to bypass security checks. To further increase security and to reduce the

scope of software where the most stringent SDL practices have to be applied, a "divide and conquer" approach can be used. The full software stack is split into a normal part where less sensitive operations are executed, and a trusted part that encapsulates secrets and other sensitive data and the associated processing.

Figure 1 shows two ways of realizing such a split into a normal execution environment and a TEE. The left-side diagram shows a single processor TEE, where a single CPU executes both the trusted and the non-trusted applications. On the right, a traditional approach shows that each environment has its own, private CPU and other resources like memory or cryptographic accelerators. This approach is called a "hosted TEE" since the normal application processor acts as a host for the TEE CPU.

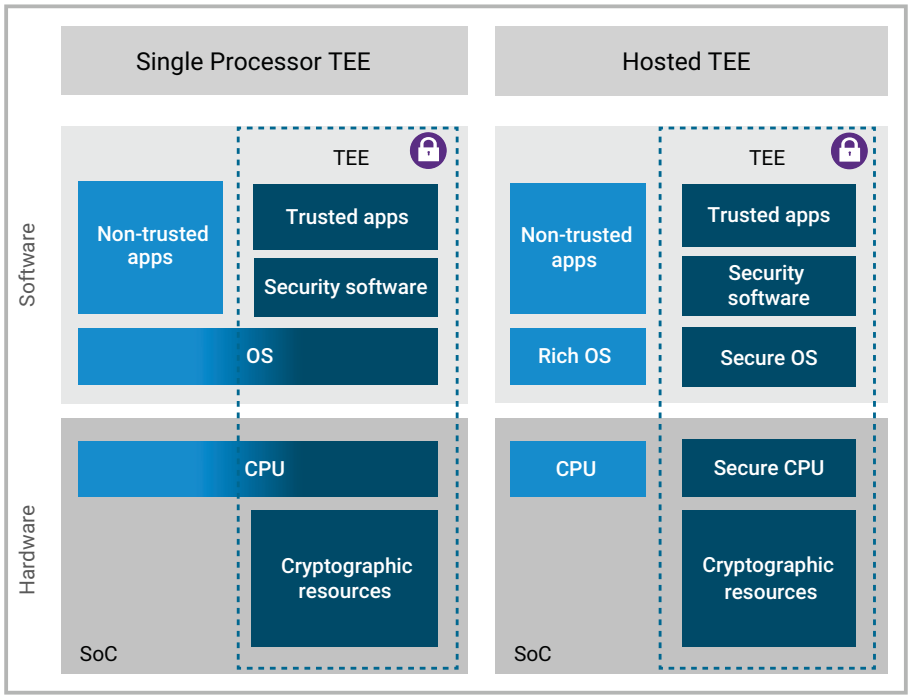

Figure 1: Trusted Execution Environments on a single (left) or dedicated (right) processor

Single-Processor TEE Approach

The benefit of the single-processor TEE approach is that fewer hardware resources are required since these are shared between normal and secure applications, resulting in lower system costs and a simpler design. However, the TEE secrets and processing should still be isolated from the normal, non-trusted processing. The Synopsys ARC processor IP can protect the TEE's secrets by means of SecureShield technology. SecureShield introduces a new privilege level for the processor: Secure Mode. Together with the usual kernel and user privilege levels, this gives in total four privilege levels for the ARC SecureShield processors: normal-user, normal-kernel, secure-user, and secure kernel.

Figure 2 depicts how SecureShield isolates secure and normal partitions on a single processor. Besides the additional secure privilege mode, an MPU provides the separation of memory and (memory mapped) peripherals into normal and secure partitions. Whereas trusted software running in secure mode can access both normal and secure resources, normal software can only access normal memory and peripherals: a privilege exception will occur when normal software attempts to access secure resources.

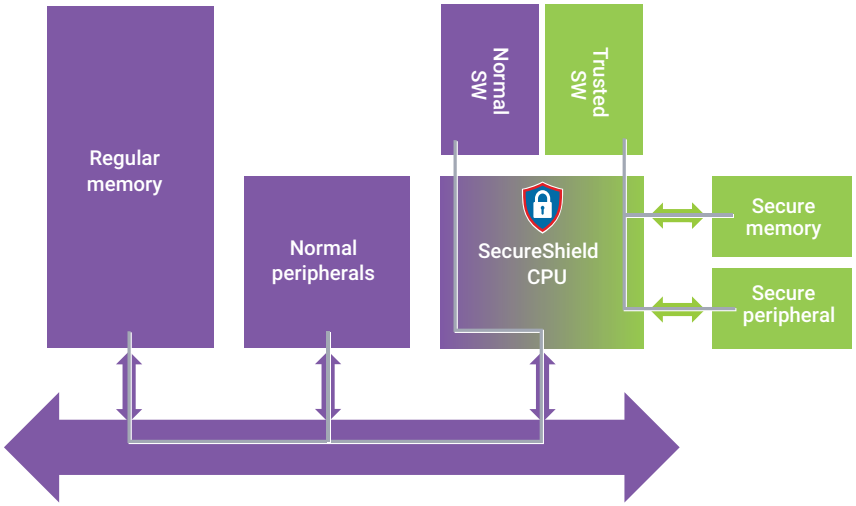

Figure 2: SecureShield isolating Normal and Secure partitions on a single processor

In addition to the secure privileged mode and MPU, Synopsys ARC processors offer other security features such as hardware stack boundary checking, side-channel protection against timing and power analysis physical attacks, and secure debug to only allow debug access from authenticated debuggers. Not all of these features are always required. Depending on the targeted applications, the ARC processor can be configured at design time to include only the required features to conserve power and area.

Hosted TEE Approach

The simplest way to separate trusted, secure software from normal, non-trusted software is to use multiple cores and allocate software to a core according to its security profile (right side of Figure 1). A general-purpose CPU runs—optionally on top of a small Real-Time Operating system (RTOS)—normal application software that is not specifically trusted and that could come from any source, including end-user programming. For security related software a second, secure CPU can be used running its own software stack fully isolated from the other processor. Since it does not share resources with the other processor, there is no risk of leaking secrets to or having unauthorized modifications made by the general-purpose CPU. This does require that the memory for each CPU is physically separated by tightly coupling it to the processor, using separate buses, or using other bus and address map protection mechanisms. An inherent drawback of this strict hardware isolation of normal and secure worlds is that communication between both worlds can become more complex. Either some shared resource like a small shared memory would be required, or a dedicated communication channel should connect both CPUs.

Tortuga Logic's Security Verification Overview & SDL

Validating system-level security vulnerabilities that span both hardware and software can be a challenging and resource-consuming task. Tortuga Logic's Radix-S finds vulnerabilities in both hardware and software to ensure that you build the most secure system possible.

Radix-S captures security threat models that cross both hardware and software. The tool analyzes the hardware design during execution of software for vulnerabilities, identifies the sources of the vulnerabilities, and provides guidance to resolve them. Radix-S integrates directly into the existing functional verification environments for ARC processors.

The SDL proposed in this paper follows a 4-step process:

1. Specification of security threat models: Tortuga Logic's Sentinel™ language allows designers to specify assets to be protected and the intended use of those assets in the form of security rules.

2. Generation of security model: The Radix-S tool imports the design RTL and the security rules and generates a security model for the design, which is fully synthesizable hardware IP.

3. Insertion of security model into functional verification environment: The security model executes in existing verification environments in parallel with the original RTL and the processor software to identify security rule violations.

4. Analysis of results: Analysis views in Radix-S provide the ability to analyze security rule violations and explore management of critical assets in the design.

Figure 3 shows the process of integrating Radix-S into existing design and verification environments for ARC processors. The inputs to Radix-S are the ARC design RTL and a set of security rules expressed in Tortuga Logic's Sentinel Language. From the ARC design RTL and Sentinel rules, Radix-S generates a security model (Verilog IP) that easily integrates into standard functional verification environments without disruption to existing workflows. The security model is used to check the Sentinel security rules during simulation and is only utilized during security verification analysis. The security model does not become part of the design implementation or impact area or timing overhead.

Most users' functional verification environments already include both the ARC RTL and software to be run on the ARC processor, meaning Radix-S can perform system-level security analysis of both the hardware and software without the development of custom test infrastructure. The Sentinel security rules, such as the three example rules detailed in the following section, are highly re-useable and can be used to find security vulnerabilities in multiple ARC RTL and software configurations that share the same threat model.

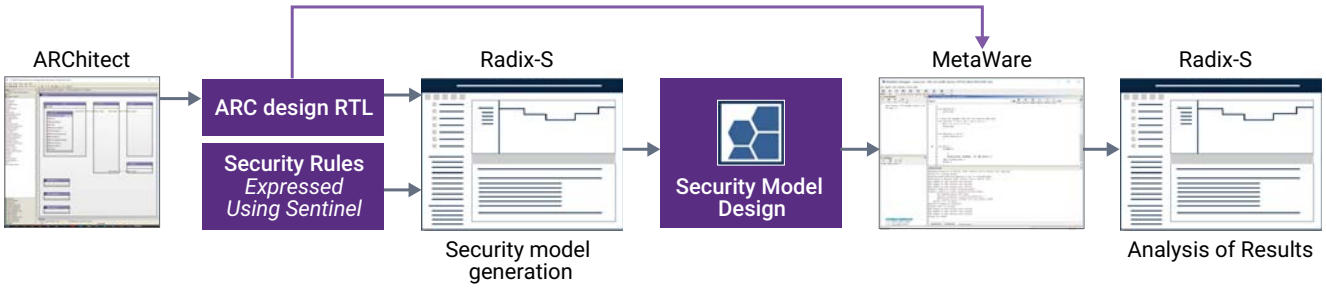

Figure 3: Radix-S Tool Flow

Case Study

Tortuga Logic and Synopsys have created and verified three security rules, based on threat models related to secure debug and configuration of the ARC MPU, that can be easily reused for other projects. Below, we describe how these security rules are specified and verified during hardware/software simulation of ARC processors.

Verifying Secure Debug Locked Mode

Debug mode provides increased controllability and observability for both hardware and software testing; however access to design internals poses a security risk. To ensure debug functionality can only be accessed by authorized users, ARC processors provide the option to configure secure debug features, including a locking/unlocking mechanism (Figure 4). Synopsys provides an example unlock module based on a simple challenge response protocol. Designers typically replace the example and define custom unlock logic to provide the security necessary to address specific threat models or to integrate the processor secure debug into a larger SoC secure debug design.

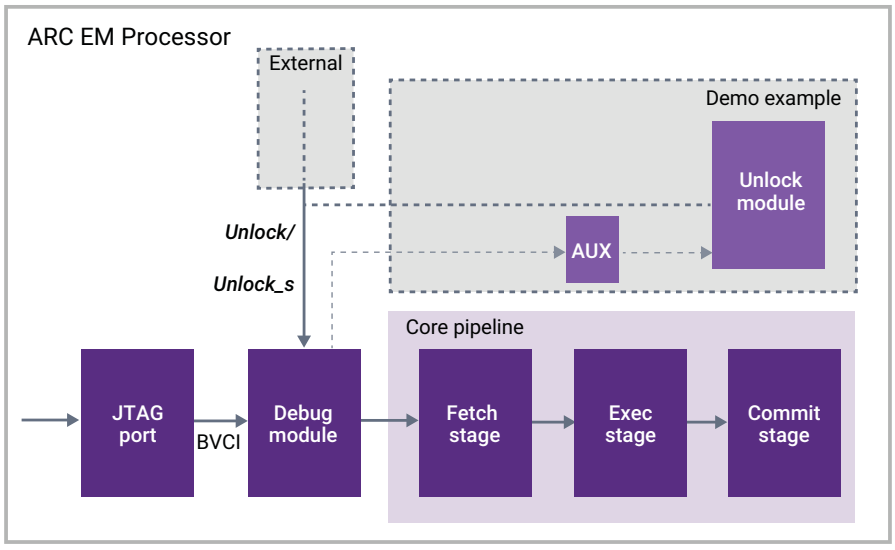

Figure 4: Secure Debug in ARC EM Processor

Designers should verify that the lock functionality implements the expected behavior and that no unexpected information flows occur when the debug access port is in locked mode. When in locked mode the debugger should not be able to access 1) processor registers, or 2) processor data memory.

Radix-S can verify that a specific ARC configuration satisfies this assumption when in locked mode by creating two security rules to track to the flow of the processor registers and data memory to the debug port. These rules are written using Tortuga Logic's Sentinel language. Sentinel provides a "no-flow" operator (=/=>) to create rules for tracking information flows between design signals. The "when" keyword specifies conditions which must be met before flow tracking is performed.

Security Rule #1: CPU register contents should never flow to the debug interface when the debug access port is in locked mode.
u_regfile_2r2w.$all_outputs when (!dbg_unlock) =/=> dbg.$all_outputs

u_regfile_2r2w is the module that implements the ARC EM register file. ***dbg_unlock*** is the unlock signal from Figure 4, and ***dbg*** is the debug module itself. The ***"$all_outputs"*** keyword is shorthand for describing the set of all output signals for a particular module instance in the design hierarchy. Rule #1 will fail if register file contents flow to debug module outputs when debug is in locked mode.

Security Rule #2: Data memory should never flow to the debug interface when the debug access port is in locked mode.
dccm_data_out when (!dbg_unlock) =/=> dbg.$all_outputs

Rule #2 will fail if data memory contents (***dccm_data_out***) can exit the debug access port while debug is in locked mode. Note that ARC processors feature single cycle access data closely coupled memories. Radix-S can verify both Rule #1 and #2 as part of the existing test flow in an ARC-based system to provide assurance that the debug port does not leak internal processor information when in locked mode. Designers can write additional Radix-S Sentinel rules to verify the custom unlock module itself.

Verifying Software Initialization of Secure Memory Regions

As was shown in the right side of Figure 1, ARC SecureShield technology can partition memory into secure and normal regions which should remain completely isolated. For example, cryptographic keys residing in secure memory used to encrypt confidential information should not be accessible by unprivileged programs. Unprivileged programs should only have access to normal memory, and the data in secure memory should never end up in normal memory. In this case study, the threat model covers malicious unprivileged programs running in normal mode with access to normal memory regions only—enforced by the ARC MPU. Malicious software will try to identify and exploit mistakes made during the configuration of the memory regions to extract sensitive information from programs executing in secure mode. Because Radix-S provides a concise mechanism for describing information flow properties, designers can verify specific memory protection configurations and easily adapt as design requirements evolve. Moreover, the rules do not check the MPU control registers directly but instead check the higher-level property that data from secure memory should not end up in normal memory. Therefore, not only a specific misconfiguration error would be checked, but any error that could lead to data leaking from secure to normal memory.

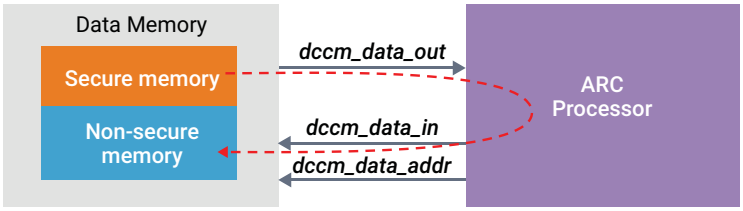

Figure 5: Potential Security Violation—Flows between isolated memory regions

The ARC's SecureShield MPU can be programmed to enforce memory regions with different access privileges. This way, "secure" programs can be isolated from "normal" programs. In this case study, two data memory regions are created:

1. Secure Data Region: 0x800000 to 0x801FFF
2. Normal Data Region: 0x802000 to 0x81FFFF

Radix-S verifies the isolation of these two memory regions by creating a Sentinel rule tracking the flow of information from the output of the data memory to the input of the data memory (in between the data will flow through the ARC processor as software is executing). Security Rule #3 will fail if information flows through the path shown as a red dashed line in Figure 5. One important aspect of Sentinel rules is that information flows are tracked over time through arbitrary combinational and sequential logic transformations. Data from secure memory can be pipelined or combined with other data inside the ARC processor and Rule #3 will still detect if any versions of the secure data are written into the normal data region.

Security Rule #3: Data from secure memory region should never flow to normal memory region
dccm_data_out when (dccm_data_addr >= 0x800000 && dccm_data_addr < 0x802000) =/=> dccm_data_in || (dccm_data_addr >= 0x800000 && dccm_data_addr < 0x802000)

Rule #3 states that the output data from the data memory module (***dccm_data_out***) is tracked when reading from the secure address range and the rule will fail if this information propagates to the input of the data memory module (***dccm_data_in***) unless the data is entering the memory during a write to the secure region. This extra check of the address range for secure information that arrives at ***dccm_data_in*** is necessary because secure software can read data from the secure region and then write data back to secure memory.

Rule #3 can detect software configuration errors made during the setup of the secure and normal memory regions. To test this assertion, Synopsys and Tortuga Logic tested Rule #3 in a design that configured the secure region to be 32 bytes shorter than expected. Because of this mistake, any sensitive privileged data written to the last 32 bytes of the secure region could be read by programs executing in normal mode.

The test program starts in secure mode and writes 'secret' data (words 0x7320794d and 0x65726365) to the misconfigured 32-byte region. The program then switches to normal mode, reads the data and stores it in normal memory. Figure 6 contains an excerpt from the log file produced during the simulation of the test program. The log chronicles all assembly instructions executed on the ARC processor in addition to any Radix-S rule failures that occur. Rule #3 fails when the first word of the privileged data is stored to the normal memory region.

```
112162855:   C     000035f4 1104040a|ld.ab    r10,[r1,4]      | w0: r1  <- 00801ff4 w1: r10 <- 7320794d ld [801ff0]
112162860:   C     000035f8 1104040b|ld.ab    r11,[r1,4]      | w0: r1  <- 00801ff8 w1: r11 <- 65726365 ld [801ff4]
112162865:   C     000035fc 2442104c|sub      r12,r12,1       | w0: r12 <- 00000000
-FAIL- Assertion assertion_TEST3 failed at time 112162868
- Occurred (1) time(s).
112162870:   C     00003600 1b040290|st.ab    r10,[r3,4]      | w0: r3  <- 00802aac st 7320794d -> [802aa8]
```

Figure 6: Log file produced during simulation showing rule failure when data from the secure memory region is stored in normal memory

The Radix-S Waveform View window (Figure 7) provides additional details about the rule failure. Radix-S enables easy visual inspection of information flows within the design by marking secure signals in red shading on top of the waveform. Figure 7 shows information propagation from the source signal in the top row (***dccm_data_out***) to several selected signals from the processor register file and the destination signal in the bottom row (***dccm_data_in***).

The propagation of source signal ***dccm_data_out*** to any signal in the design can be explored easily with Waveform View. Any design signal can be added to the waveform by the user and cycles during which dccm_data_out flows to the signal will be shaded red. In Figure 7, the specific data value read from the secure region (0x7320794d) propagates through the register file to the input of the data memory. Note that this example's ARC processor configuration implements additional ECC checking (the extra '0x38' byte prepended to the register file data) does not impact the Radix-S analysis.

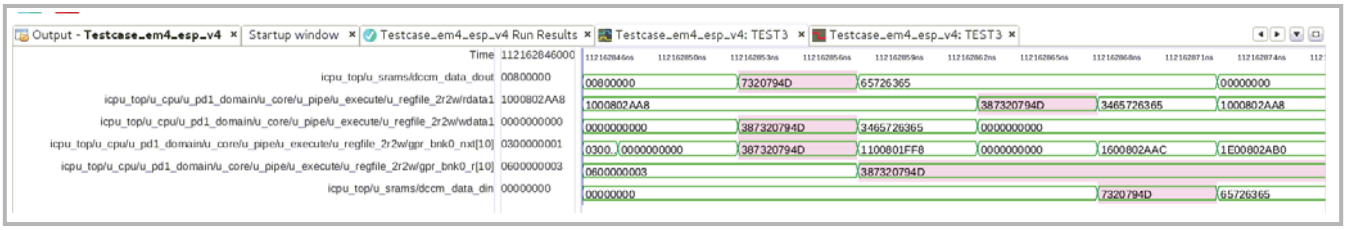

Figure 7: Radix-S Waveform View showing information flow from ***dccm_data_out*** to ***dccm_data_in*** leading to Rule #3 failure

Radix-S also provides a concrete path showing information flow from the rule source to destination signal. The information flow is presented graphically as a path through time and design hierarchy. The Path View for Rule #3 is shown in Figure 8, showing data read from secure memory (from ***dccm_data_out***) flowing through the ARC processor before it re-enters the data memory module during a write (through ***dccm_data_in***) to normal memory, causing the rule to fail. Path View provides valuable insight into the root cause of rule failures and makes the security analysis more efficient.

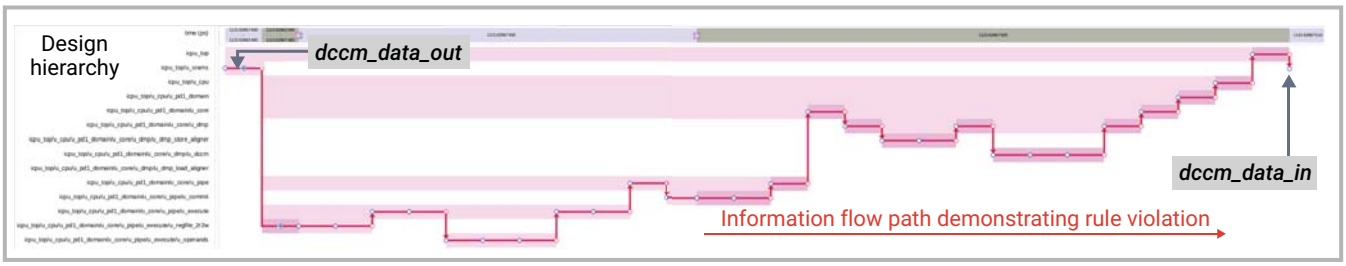

Figure 8: Radix-S Path View—a complete information flow path demonstrating rule failure

Conclusions

Security in modern systems is of utmost importance. Device manufacturers are including multiple security features and attack protections into both the hardware and software design. For example, the DesignWare ARC Processor IP includes many security functions in its SecureShield feature set. End-product system security, however, cannot be guaranteed by using a secure processor alone. The final product security results not only from using proven, secure hardware components. Configuration, integration and additional software have to be taken into account as well. When integrating a secure processor into a chip and developing the software for it, security vulnerabilities can still be introduced. Verifying the absence of these security vulnerabilities in systems comprised of hardware and software is a complex effort. Radix-S can verify that designers have integrated and programmed security features correctly, as the examples of the Secure Debug unlock module and Memory Protection Unit illustrate.

Radix-S is a powerful tool for ARC customers to identify and prevent any security vulnerabilities that are introduced by configuring and integrating the processor in their design, or by the platform and application software that is executed on it. Tortuga Logic's Radix-S software provides the capability to verify the security of any ARC Processor based design with the consideration of hardware integration and the software running on top of it. Improper programming of the ARC Processor can lead to security vulnerabilities, and Radix-S can detect these vulnerabilities during standard functional verification. Radix-S can be used as the basis for a true hardware SDL that enables designers to build secure, high-quality processor-based systems.

References

[1] https://www.semiconductors.org/more-than-1-trillion-semiconductors-sold-annually-for-the-first-time-ever-in-2018/

[2] https://www.microsoft.com/en-us/download/details.aspx?id=12379

[3] https://www.synopsys.com/software-integrity.html

[4] NIST, The Economic Impacts of Inadequate Infrastructure for Software Testing, May 2002. https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf