2011

Persistent monitoring of digital ICs to verify hardware trust

Justin Richard Rilling
Iowa State University

Persistent monitoring of digital ICs to verify hardware trust

by

Justin Richard Rilling

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Joseph Zambreno, Major Professor
Phillip Jones
Thomas Daniels

Iowa State University

Ames, Iowa

2011

DEDICATION

To my wife, Jessica

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Joseph Zambreno, for his support and patience throughout my pursuit for a master's degree in computer engineering. He has challenged me to develop new skill sets, provided important feedback, and fostered an environment for growth. I owe a large part of my professional development to him.

Dr. Phillip Jones has been a positive influence and also deserves a hearty thank you for his assistance in my graduate studies. His "stop by anytime" approachability and hands-on tactics to education are deeply appreciated. I would also like to specifically extend gratitude to my committee members Dr. Joseph Zambreno, Dr. Phillip Jones, and Dr. Thomas Daniels, for their investments of time and energy to improve this work.

A special thanks to my wife Jessica for her support on the homefront in all the big and small ways that are hard to measure, but certainly ensured my success. Thank you to my family for providing me a strong foundation and for leading by example. I attribute my work ethic and dedication to excellence to you. And finally, thank you to my friends who provided comic relief as well as many subtle, and not so subtle, reminders to strive for a life/work balance.

ABSTRACT

The specialization of the semiconductor industry has resulted in a global Integrated Circuit (IC) supply chain that is susceptible to hardware Trojans – malicious circuitry that is embedded into the chip during the design cycle. This nefarious attack could compromise the mission-critical systems which implement these devices. While a trusted domestic IC supply chain exists with resources such as the Trusted Foundry Program, it's highly desirable to utilize the high yield, fast turn-around time, low cost, and leading-edge technology of the global IC supply chain. Research into the verification of hardware trust has made significant progress in recent years but is still far from a single, comprehensive solution. Most proposed solutions are one-time implementable methods that attempt to detect hardware Trojans during the verification stage of the IC development process. While this is a desirable solution, it's not realistic given the current limitations of hardware Trojan detection techniques. We propose a more comprehensive solution that involves the persistent verification of hardware trust in the field, in addition to several one-time methods implemented during IC verification. We define a persistent verification framework that involves the use of a few ICs from a secure process flow to persistently monitor and verify the operation of several untrusted ICs from the global supply chain. This allows the system integrator to realize the benefits of the global IC supply chain while maintaining the integrity of the system. We develop a *system monitor* which filters the IO of untrusted digital ICs for a set of patterns, which we refer to as *digital signal signatures*, to verify the operation of the devices.

CHAPTER 1. INTRODUCTION

Integrated Circuits (ICs) are the building blocks of modern electronics. They are used in everything from simple appliances to mission-critical devices. The integrity of ICs has come into question following recent events like the scandal in which 59,000 counterfeit ICs from China were sold to the US Navy [25]. These chips did not meet military specifications, making them more prone to failure in the field. In addition to the risk of low quality counterfeit chips, there has also been concern regarding the malicious modification of chips during the design or manufacturing process. This is known as a *hardware Trojan* attack. An adversary can insert a kill switch into a chip to disable it via a remote signal or add a backdoor to subvert the circuit's security mechanisms. This is a particularly nefarious attack due to the ease at which an adversary can infiltrate the global IC supply chain and the difficulty of detecting malicious modifications once they are inserted into the IC.

When ICs were first developed, the threat of a hardware Trojan attack was minimal. Semiconductor companies typically designed and fabricated their own ICs – a viable economic engagement for a single entity at that time and, in many cases, a necessary one since IC manufacturing practices were not standardized. Transistor counts on state-of-the-art ICs only numbered in the hundreds to thousands so it was feasible to detect malicious modifications via design observation and functional testing.

Since then, the semiconductor industry has experienced exponential growth due to the continual improvement of transistor technology. This improvement is reflected in the increasing number of transistors available on state-of-the-art chips, which has doubled every two years as predicted by Moore's Law (Fig. 1.1). As transistor technology improved, the semiconductor industry shifted to the highly specialized, global IC supply chain shown in Fig. 1.2. Financial

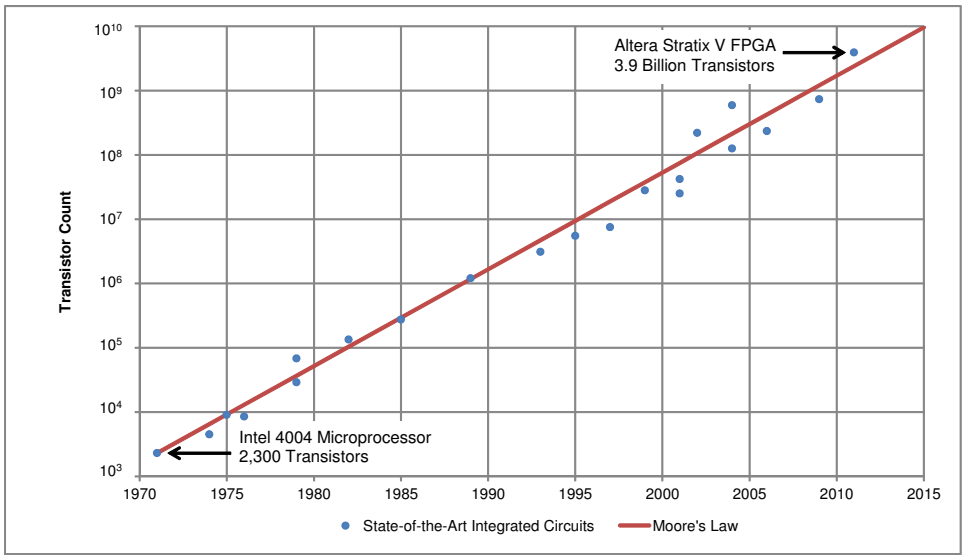


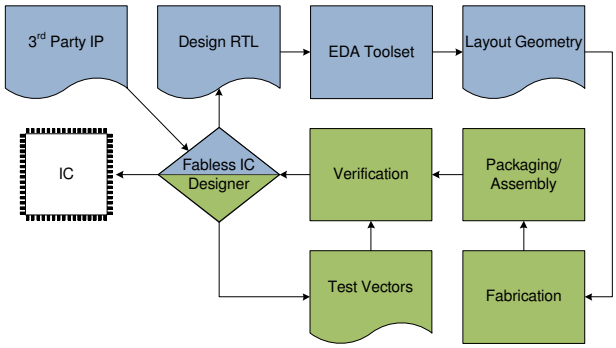
Figure 1.1: The number of transistors in latest-technology chips [8, 28, 47] has increased as predicted by Moore's Law which states that chip capacity will double every two years. This is due to continual improvements in transistor technology.

pressure and a need for greater design productivity has resulted in the common practice of outsourcing and offshoring IC fabrication, packaging/assembly and verification as well as some of the design effort [19, 27]. The global IC supply chain is more vulnerable to a hardware Trojan attack. An adversary can infiltrate the IC supply chain as a hardware designer, Electronic Design Automation (EDA) toolset developer, foundry worker, a Semiconductor Assembly and Test Services (SATS) company employee, etc. It's no longer feasible for a single entity to oversee the end-to-end development process of the typical IC. The majority of post-silicon services are located offshore which has raised concerns regarding national security. Also, since modern state-of-the-art ICs contain on the order of a billion transistors malicious modifications to these devices can easily go unnoticed.

In 2003, the U.S. Senate expressed their concerns regarding the globalization of the U.S. Semiconductor Industry, stating this migration posed a threat to national security and requesting action by the Department of Defense (DoD) and intelligence community [62]. Similar reports were published by the U.S. Defense Science Board task force [18], Semiconductor Equipment and Materials International (SEMI) [1] and IEEE Spectrum [4]. In 2004, the DoD and

Market	Leading Companies	HQ Location
Fabless Design	Qualcomm	United States
	Broadcom	United States
	AMD	United States
3rd-Party IP	ARM	United Kingdom
	MIPS Technologies	United States
	Synopsys	United States
EDA Toolsets	Cadence Design Systems	United States
	Synopsys	United States
	Mentor Graphics	United States
Fabrication	TSMC	Taiwan
	UMC	Taiwan
	SMIC	China
Packaging/Assembly	ASE	Taiwan
	SPIL	Taiwan
	STATS ChipPAC	Singapore

(a) Leading companies [7, 19, 39, 48]



(b) Process flow (derived from [16])

Figure 1.2: Global IC supply chain. Blue boxes represent pre-silicon stages of the supply chain while green boxes represent fabrication and post-silicon stages.

the National Security Agency (NSA) initiated the Trusted Foundry Program to establish a trusted domestic IC supply chain. To date, 49 accredited microelectronic service providers participate in the program offering a variety of services from design to fabrication to verification. Entities such as Sandia National Laboratories and Honeywell International provide design services, IBM Burlington provides fabrication services, and Rockwell Collins provides packaging/assembly/testing services [64]. While the domestic supply chain provides trustworthy ICs, it's desirable to utilize the high yield, fast turn-around time, low cost [52], and leading-edge technology [30] of the global supply chain.

The U.S. government has provided funding for research into the detection of hardware Trojans in ICs from the global supply chain [51, 53]. This research, along with other academic contributions, has primarily focused on two aspects of hardware Trojan detection.

- Activating hardware Trojans, which are inherently difficult to trigger
- Side-channel analysis methods to detect a hardware Trojan once it's, at least partially, activated

While progress has been made in both areas, each still faces significant challenges. Trojan activation methods must reduce an enormous set of possible input combinations, which is increasing with rising circuit complexity, to a subset that is practical to test and likely to trigger a Trojan if it exists on the chip. Side-channel analysis must distinguish changes in

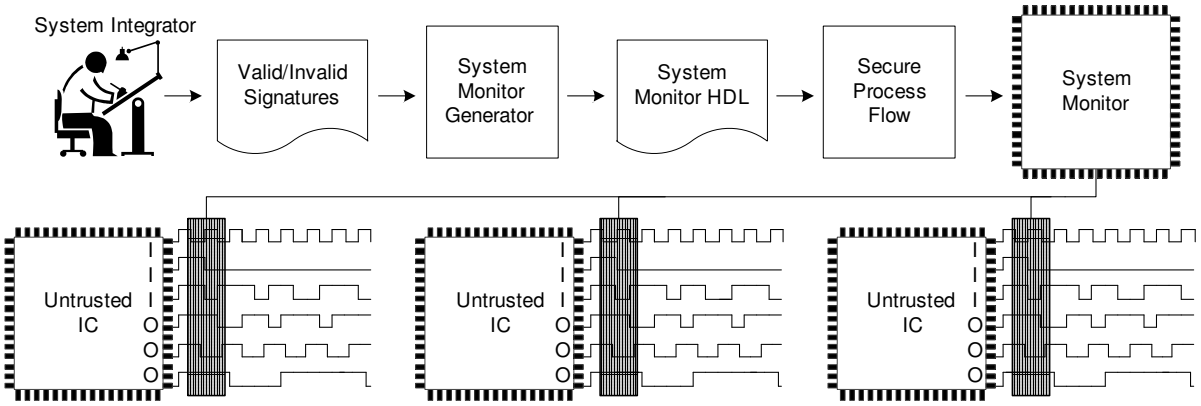


Figure 1.3: Overview of our persistent monitoring approach

power and delay characteristics due to process variation, which is increasing with decreasing transistor feature sizes, from the small signature of a hardware Trojan.

Most Trojan detection methods are designed for a one-time implementation during the verification stage of the IC development process. While it's certainly desirable to detect a hardware Trojan embedded in an IC prior to deployment in the field, it's not completely realistic given the current limitations of these methods. Moreover, the issues that plague hardware Trojan detection will increase as transistor technology continues to improve [9]. A more comprehensive framework is needed to address the hardware trust issue. We propose persistent verification of untrusted ICs in the field in addition to the one-time verification of ICs during the development cycle. A few trusted ICs, such as those from the domestic supply chain, can monitor and persistently verify many untrusted ICs, such as those from the global supply chain. This allows the system integrator to reap the benefits of ICs from the global supply chain while continually verifying that an undetected Trojan has not comprised the system.

Our approach to the persistent verification of untrusted ICs is shown in fig. 1.3. We develop a system monitor that filters the IO signals of untrusted digital ICs for a set of patterns, which we call *digital signal signatures*, to verify the chips are operating as expected. The process to develop a system monitor begins with the system integrator defining a set of valid and invalid signatures in a high-level language. A design automation tool, referred to as the system monitor

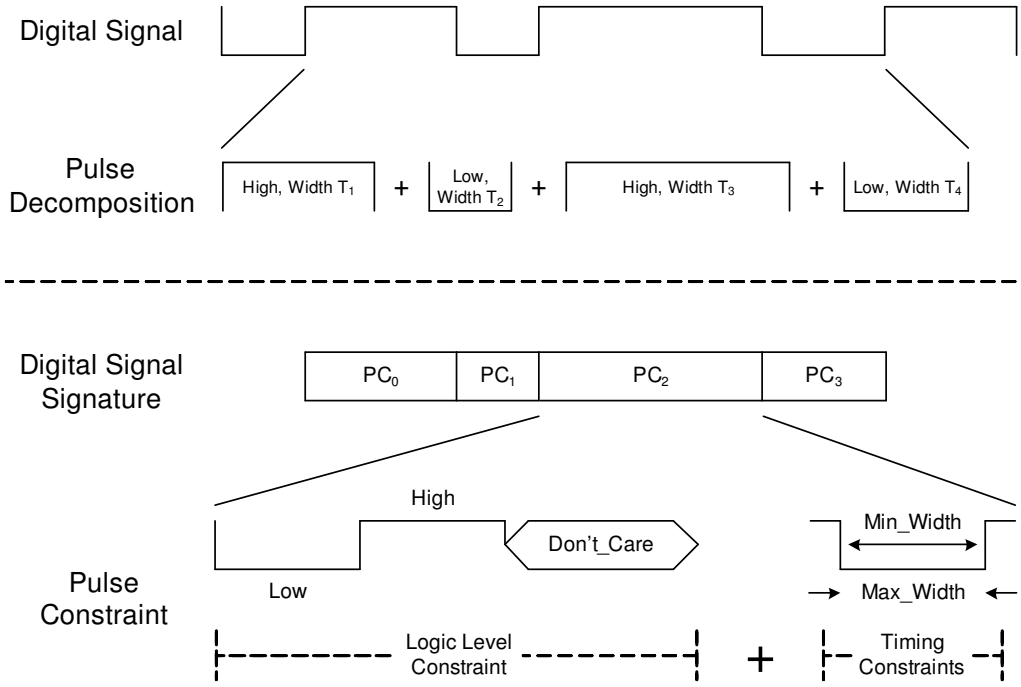


Figure 1.4: Digital signal pulse decomposition and signature definition

generator, then translates the high-level specification into a hardware descriptive language (HDL) that describes the monitor hardware. This HDL is then transformed into physical monitoring hardware via a trusted process flow using resources such as the Trusted Foundry Program.

The system monitor searches for a set of valid signatures as well as a set of invalid signatures based on the system integrator's unique knowledge of the system. The detection of a valid signature provides a certain level of confidence that the chip is functioning correctly. Conversely, the detection of an invalid signature proves malicious circuitry has been activated. While output signals are the primary monitoring target, input signals can also be monitored. Unexpected behavior discovered in input signals may indicate an adversary's attempt to trigger a hardware Trojan.

Figure 1.4 shows the decomposition of a digital signal into a series of consecutive pulses, also referred to as a pulse train. The pulse train is fully described by specifying the logic-level/width of each pulse and the pulse order. We call this description of a pulse train a *digital*

signal signature. Each pulse constraint is a combination of a logic level constraint and timing constraints. The logic level constraint defines the expected signal level of the pulse. It's an optional constraint and is either "low" or "high" when specified and "don't care" otherwise. The timing constraints define the expected width of the pulse. This width is expressed with minimum and maximum bounds. If an exact width is specified then the two bounds are set equal. This constraint is mandatory as a pulse must have a finite width to be meaningful.

In this work, we propose a persistent monitoring approach to verify hardware trust in digital ICs. We develop a system monitor architecture that filters the IO of untrusted ICs for a set of digital signal signatures. We define a high-level language to specify these signatures and build a prototype system monitor generator to translate signatures into HDL. We also evaluate the effectiveness of our monitoring system as well as the amount of overhead generated.

The rest of the paper is organized as follows: Chapter 2 describes work related to the verification of hardware trust. Chapter 3 focuses on the design, generation, and realization of the system monitor. Chapter 4 analyzes our approach in terms of security and overhead. Chapter 5 provides a summary of our work as well as a discussion of future work. We've also included an Appendix section describing our experiences in creating proof-of-concept hardware Trojans.

CHAPTER 2. RELATED WORK

2.1 Hardware Trojan Characterization

Hardware Trojans must be accurately characterized to enable the development of effective detection schemes. The classic definition of a hardware Trojan is a circuit that's composed of an activation mechanism and a malicious action. Wang, Tehiranipoor, and Plusquellic transformed this definition into a detailed taxonomy of hardware Trojans [70]. In their taxonomy (shown in Fig. 2.1) hardware Trojans are classified according to their physical, activation, and action characteristics. The following sections describe each category in detail.

2.1.1 Physical Characteristics

The four physical characteristics of a Trojan are distribution, structure, size and type. The distribution parameter specifies the Trojan's physical location on the chip. Typical locations for Trojans to reside on a chip include the processor, memory, input/output hardware, power supply, and clock grid [35]. A Trojan's structure refers to the physical change the Trojan imposes on the original circuit. Some Trojans, as described in [63], only modify manufacturing procedures and do not change the physical layout of the target hardware. Other Trojans, remove or add transistors which will result in a different layout and will change power and delay characteristics. In [32], the target hardware was optimized to offset the Trojan's size and impact on circuit structure. Trojan size refers to the number of transistors that are modified, added, or removed from the original circuit. Small Trojans are easier to activate but more difficult to detect [70]. A Trojan's type is either functional or parametric. Functional Trojans add or remove transistors from the device while parametric Trojans modify existing components. The addition of a kill switch is a functional Trojan while the thinning of a wire

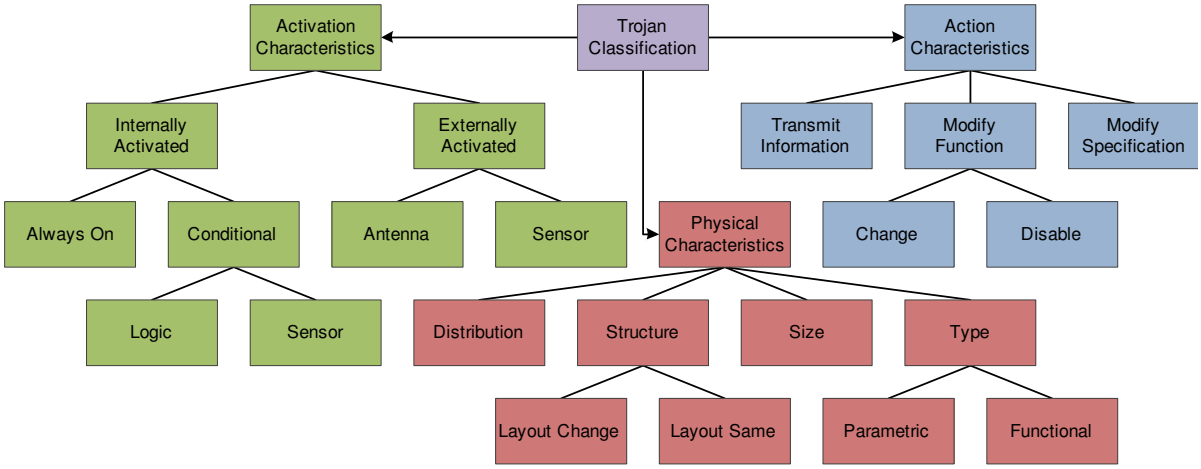


Figure 2.1: Trojan taxonomy (source: [67, 70])

to reduce reliability is a parametric Trojan.

2.1.2 Activation Characteristics

Trojans are either activated via mechanisms internal to the chip or through external mechanisms. Some internally activated Trojans are "always on" while others are triggered by a logic and/or sensor circuit. In the Activation Mechanism section of the Appendix, we describe a ring oscillator-based temperature sensor that triggered Trojans when a specific logic sequence was observed and the chip temperature was elevated a few degrees above room temperature. External triggers typically use an antenna or sensor mechanism to activate the Trojan. These are particularly desirable triggers for an adversary since they can be activated remotely.

2.1.3 Action Characteristics

Once activated, a hardware Trojan executes a malevolent action. A Trojan can transmit secret information from a chip to undermine the security of the system. For instance, security mechanisms in hardware architectures like AEGIS [66], XOM [41], and the TPM [23] all rely on the secrecy of hardware keys. These security mechanisms can be circumvented if the hardware keys are leaked. In [42], a class of lightweight Trojans were developed which leaked information via power side-channels. In [22], the Universal Serial Bus (USB) protocol was manipulated to

transmit secret data. Another type of Trojan action is the modification of the original circuit specification to reduce reliability. In [63], slight alterations were introduced to IC fabrication to reduce the lifetime of the chip by accelerating transistor wearout mechanisms like Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI)-this type of action is very difficult to detect and is especially treacherous in systems like space satellites where maintenance is difficult and costly. The third type of Trojan action is the functional modification of the circuit. An adversary can add/remove transistors to modify or disable the functionality of the circuit. In [37], processor hardware was compromised to create a login backdoor which gave the attacker privileged access to the machine.

2.2 Hardware Trust Verification

One of the largest single research efforts into hardware Trojan detection began in 2007. The Defense Advanced Research Projects Agency (DARPA) launched a three-year program called *TRUST for Integrated Circuits* to develop a Trojan detection method with an accuracy rate of at least 90% and a low false positive rate [51]. Test chips were created by the University of Southern California Information Sciences Institute. Lincoln Labs at the Massachusetts Institute of Technology was responsible for inserting malicious circuitry in the chips. Three groups: Xradia, Luna Innovations, and Raytheon, worked to develop efficient methods for locating the malicious insertions. The John Hopkins Applied Physics Lab then established methods to measure success [4]. The results from this study have not been published and therefore cannot be integrated into academic research at this time, but its existence does speak to the validity of Trojan research.

In 2010, the TRUST for Integrated Circuits program was succeeded by the Integrity and Reliability of Integrated Circuits (IRIS) program. The objective of this new program was to "develop the technology to derive the functionality of an IC to determine unambiguously if malicious modifications have been made to that IC, and to accurately determine the IC's useful lifespan from a physical perspective" [53].

Several Trojan detection schemes have been published recently in academic research. While

no scheme is a comprehensive solution to Trojan detection, each proposes an effective method of detection for a specific subset of hardware Trojans and/or a specific IC architecture. Trojan detection schemes can be divided into one-time and persistent methods. One-time methods are typically implemented during the verification stage of IC development. The IC is tested with high coverage input vectors that are likely to trigger a Trojan if it exists while monitoring the IC for uncharacteristic measurements. Persistent methods are implemented in the field. They continually monitor the IC for abnormal behavior that might indicate the activation of a hardware Trojan.

2.2.1 One-Time Methods

One-time hardware trust verification research is divided into two areas: the activation of hardware Trojans and the detection of, at least partially, activated Trojans via side-channel analysis. Trojan activation research faces the challenge of how to reduce the enormous size of possible inputs, which is increasing with rising circuit complexity, to a high coverage subset that is likely to trigger a Trojan if it exists on the chip. Side-channel analysis research must address the challenge of how to distinguish changes in power and delay characteristics due to process variation, which is increasing with the improvement in transistor technology, from that caused by a Trojan. The following section describes several one-time methods.

2.2.1.1 Trojan Activation

Given the size and complexity of modern ICs, exhaustive verification is not feasible. For instance, a circuit that has n-inputs and m-stages requires $2^{n} + m$ test vectors to conduct an exhaustive test. Attackers exploit this reality by designing Trojans that only activate under rare events that won't be tested during verification. Trojan activation research strives to improve this scenario by analyzing circuit structure to determine ideal Trojan locations and then developing methods to active these regions using either non-invasive test vector generation or Design-for-Test (DFT) strategies.

Controllability is the ease of which a primary input can manipulate the input of a specific

node and observability is the likelihood that the output of a specific node will have an effect on a primary output. Wolff et al. address Trojans that are triggered with nets that have low controllability and payloads with low observability [71]. Regions in a circuit that fit this profile are targeted as possible Trojan locations. A vector set is generated that is designed to activate these target regions and is augmented with the traditional set of Automatic Test Pattern Generation (ATPG) vectors used to verify the chip.

Banga and Hsiao developed a Trojan activation technique optimized for transient power analysis [10, 11, 12]. Their "sustained vector technique" consisted of vector sets that increased switching in a specific area of the chip while decreasing switching in the rest of the chip to magnify that area's power profile. If a Trojan was detected, then that region was further analyzed to isolate the Trojan's location. In another study, Banga and Hsiao introduced a DFT approach to Trojan activation known as VITAMIN (Voltage Inversion Technique to Ascertain Malicious Insertion in ICs) [13]. This technique switches the power and ground on certain gates to transform rare events into fairly common events. For instance, a large AND gate producing a high logic-level is a rare event but if the power connections are inverted, transforming the AND gate to a NAND gate, it becomes a common event. They combine VITAMIN with their *sustained vector technique* to achieve greater Trojan detection performance.

In [14], Banga and Hsiao concentrated on detecting Trojans embedded in third-party Intellectual Property (3PIPs). They define a four-step process that combines Trojan activation analysis and equivalence checking to produce isolated regions in the design that are probable to contain a Trojan. Zhang and Tehranipoor proposed a similar approach to 3PIP hardware Trojan detection in which *suspicious signals*, that might be associated with a hardware Trojan, were identified using coverage analysis [73]. The suspicious signal list was reduced using redundant circuit removal algorithms. They utilized sequential ATPG to activate the suspicious signals and compared the actual output with the expected output, searching for a mismatch that might indicate the activation of a Trojan. Their approach was validated using the trust benchmarks from Trust-Hub [2]

Salmani, Tehranipoor, and Plusquellic observed that Trojans are typically triggered on nets

that have a low switching probability [60, 61]. To accelerate the activation of these nets, manual switching circuitry was inserted in the chip. A switching probability threshold was defined and manual switching circuitry was added to any net under this specified threshold allowing the designer to trade decreased verification time for increased overhead.

Chakraborty et al. proposed a statistical approach to test pattern generation called MERO (Multiple Excitation of Rare Occurence) [20]. This method creates a minimal set of vectors that guarantee the excitation of nodes under a specified switching probability threshold a specified number of times. Analysis shows this approach can achieve comparable Trojan detection coverage as other methods with a significantly reduced set of test vectors decreasing activation time.

Jha and Jha used a randomization-based statistical approach to determine if a chip is equivalent to its design and free of hardware Trojans [31]. The algorithm outputs a confidence level that indicates the likelihood of a Trojan-free circuit. This confidence level improves the longer the algorithm runs, allowing the designer to choose between accuracy and verification run-time.

2.2.1.2 Side-Channel Analysis

Even after a Trojan is activated, Trojan detection is not a trivial task. A circuit abnormality must be observed to confirm the existence of a Trojan. Two leading methods of Trojan detection are logic-based testing and side-channel analysis [21]. Logic-based testing is the traditional method of IC verification and offers several advantages over side-channel analysis. The correct logic output of a circuit can be determined from the high-level circuit specification. This allows for the detection of Trojans which are inserted in the design stages as well as the manufacturing stages of IC development. Also, valid logic behavior is unambiguous so false positives are not an issue. However, the major disadvantage of logic-based testing is that it can only detect Trojans which affect the logic output of the circuit. The analysis of side-channels, like power and delay, is theoretically capable of detecting any Trojan that modifies the circuit layout, but there are some real-world practicality issues with this approach. The side-channel profile of a chip can't

be accurately determined from a high-level specification. It must be measured from a set of "golden" ICs which are destructively verified and used to authenticate the remaining chips. This only allows for the detection of Trojans which were inserted during the manufacturing stages of IC development. Another major issue is that manufacturing process variation caused by random dopant fluctuations, line edge and line width roughness, etc. is so significant at small transistor feature sizes, that it can mask the presence of a Trojan [50, 59]. Currently, hardware Trojans are only detectable if they're large enough to represent approximately .01% of overall chip [5, 49]. A significant research effort has focused on reducing the effects of process variation and improving hardware Trojan detection resolution. The following is a survey of these techniques.

Agrawal et al. first proposed the use of side-channels like power, temperature, and electromagnetic radiation to create a unique fingerprint of an IC [5]. Transient power measurements were taken from a set of *golden* chips and averaged together to establish a baseline. The *golden* chips were then destructively reverse-engineered and compared to their layout geometries to manually verify their integrity. Then the rest of the chips were measured and compared with the baseline measurement using statistical analysis to determine if the circuit was modified. Results showed Trojans that were 3-4 orders of magnitude smaller than the original circuit were distinguishable from 7.5% process variation.

Wang et al. proposed a DFT approach to side-channel analysis in which the IC was divided into four separate power grids, effectively reducing the Trojan to circuit size ratio [69]. Current measurements were taken from each power port over a period of time to get an accumulative charge versus time profile. If the accumulative charge exceeded a threshold, which included worst case process variation, then a Trojan was detected. Similarly, Rad et al. used nine power grids to further reduce the Trojan to circuit size ratio and analyze the transient current profile of the chip [56]. A calibration method was introduced and applied to each power port on every chip tested to mitigate process variation. In [55], Rad et al. test the sensitivity of this approach using a simulation that models TSMC's .18 μm process technology. Several Trojans, consisting of 2-input NAND gates, were embedded into the c499 ISCAS'85 benchmark. At a 10 dB signal-

to-noise ratio, a Trojan consisting of 4 NAND gates was detectable if input stimulus generated switching in the Trojan circuit. This was decreased to 7 gates when stimulus didn't generate Trojan activation. In [3], Rad et al. tested their approach on 45 custom test ICs created using a 65nm fabrication process. A Trojan was represented by a transistor that leaked a specified amount of current to the power grid and could be located at any one of the 4,000 test locations on the chip. The results revealed that their calibration and current analysis technique were very effective in mitigating the effects of environmental and process variation allowing for the detection of Trojans that leaked as little as 8 μA of current.

In [40], Li and Lach proposed the use of combinational logic delay paths to characterize a circuit. They introduced a DFT approach in which an efficient delay measurement circuit was added to several combinational logic paths on the chip. Since propagation delay is dependent on temperature; a ring oscillator-based temperature sensor was incorporated into the design to calibrate measurements. Rai and Lach extended this work, evaluating the performance of this approach using a simulation of modern process variation [57]. Their results proved this method was effective in distinguishing hardware Trojans from modern process variation but they also acknowledged this method was limited by the resolution of the clock generator (which drives the delay measurement circuit) skew-step and the number of feasible delay measurements.

In a similar work, Rajendran et al. utilized embedded ring oscillators in combinational logic to characterize delay paths on the chip [58]. Ring oscillators were created using existing circuit logic, a mux, and, possibly, an inverter depending on the type of gates in the logic path. An algorithm was defined that embedded ring oscillators through all gates on the chip in an efficient manner. This technique was applied to the ISCAS'85 benchmarks and results showed it induced moderate area cost while securing all gates on the chip and a test cost that scales linearly with the number gates. Jin and Makris used a similar delay-based Trojan detection method [33]. They gathered high-dimensional path delay information from the IC and then converted it to a lower-dimensional space to generate a unique fingerprint for comparison. A statistical method was implemented to reduce the effects of process variation.

Potkonjak et al. used both delay and static power side-channels along with gate-level

characterization techniques to detect hardware Trojans [54]. Linear programming and singular-value decomposition were used to detect inconsistent gate characteristics indicating the presence of a Trojan. Alkabani and Koushanfar implemented a very similar approach in [6] using multiple consistency checking.

In [34], Jin and Makris addressed a scenario in which hardware Trojans were embedded in a wireless cryptographic IC and attempted to leak the secret key to an adversary through the manipulation of transmission parameters. They observed that the attacker must impose a specific structure on the transmission parameters in order to leak the key. A statistical method was developed to recognize the specific key pattern in the transmission signal parameters. Test Trojans were created that leaked the cryptographic key via amplitude and frequency exploitations. Results confirmed this method's validity in detecting the leakage of a secret key through the manipulation of wireless transmission parameters.

Chakraborty et al. exploited the relationship between transient current and maximum operating frequency to reduce the effects of process variation [49]. This method was combined with their statistical approach to Trojan activation, MERO, to create a comprehensive Trojan detection scheme. Simulation showed a detection resolution of .04% with 20% process variation. These results were verified on a FPGA platform.

2.2.2 Persistent Methods

Given the limitations of one-time hardware trust verification methods, it's necessary to augment these methods with the persistent verification of hardware trust. These methods continually search for abnormal run-time behavior which might indicate the activation of a hardware Trojan. Similar to one-time methods, persistent methods address a specific architecture and/or class of Trojans.

The large transistor densities of modern ICs allows for the integration of an entire system on a single chip, a paradigm known as System-on-Chip (SoC). A typical SoC architecture contains several hardware modules (also referred to as hardware cores) connected by a central bus. In [36], Kim et al. addressed a scenario in which one of the hardware cores in a SoC contained

a hardware Trojan designed to snoop the system bus and degrade bus performance. They proposed modifications to typical SoC bus components like the address decoder, bus matrix, and arbiter to detect these attacks during run-time and mitigate their effect on the system. In a similar work, Huffmire et al. addressed a situation in which both secure and insecure data, commonly referred to as black and red data respectively, are present in a SoC and must be completely isolated from each other [26]. They designed a compiler that translated a high-level security policy into a reference monitor which was integrated into the SoC bus arbiter to maintain complete isolation of the two data types.

Bloom, Narahari, and Simha proposed a hardware/software approach to persistent hardware Trojan detection in a general computing system [17]. This scheme targets a hardware Trojan that implements a denial-of-service attack on the processor or a prediction, delay, or replay attack on main memory. An off-chip "hardware guard" was placed between the processor and main memory. The operating system was modified to be guard-aware, performing pseudo-random liveness checks and periodic memory protection checks with the hardware guard. Results showed this approach was effective in detecting the targeted set of hardware Trojans.

McIntyre et al. focused on persistent detection of hardware Trojans in a multi-core system [46]. They observed that it was very unlikely that two variant, but functionally equivalent, subtasks would activate the same Trojan in two different processing elements (PEs). Using this observation and the inherent redundancy of multi-core systems, they created a task scheduler that sent variant, but functionally equivalent, subtasks to different PEs. A mismatch in PE outputs indicated the activation of a Trojan. In this case, subtasks were further propagated to determine which PE produced the error and contained the hardware Trojan.

CHAPTER 3. APPROACH

For an IC to be trusted, either the supply chain from which it was produced must be trusted or the IC must be verifiably trustworthy [68]. While a trusted domestic IC supply chain exists with entities such as the Trusted Foundry Program [64], it doesn't have the large volume capabilities or leading-edge technology of the global supply chain. In this work, we propose using a few trusted ICs to persistently verify the trustworthiness of several untrusted digital ICs. As discussed in [29], the *trustworthiness* of an IC is "the degree to which the security behavior of the component is demonstrably compliant with its stated functionality." In our approach, we persistently monitor the IO of untrusted digital ICs to verify they are compliant with the original specification.

System integrators have a unique knowledge of their systems and can leverage this knowledge to place constraints on the IO signals of untrusted ICs to prove, with a certain level of confidence, they comply with their original specification. These constraints must be protocol-independent since a typical system will contain many ICs implementing several different protocols. To achieve this property, each digital signal is characterized as a series of consecutive pulses, also referred to as a pulse train. The pulse train is fully described by specifying the logic-level/width of each pulse and the pulse order. We call this description of a pulse train a *digital signal signature*. The following sections define a system monitor which searches for digital signal signatures in the IO of untrusted ICs to persistently verify their operation and alert the system if an abnormality is detected.

A digital signal signature taxonomy is shown in Fig. 3.1. A signature has a *type* that can be either valid or invalid. The discovery of valid signatures in a digital signal indicates the monitored device is functioning correctly to a certain degree of confidence. The discovery of

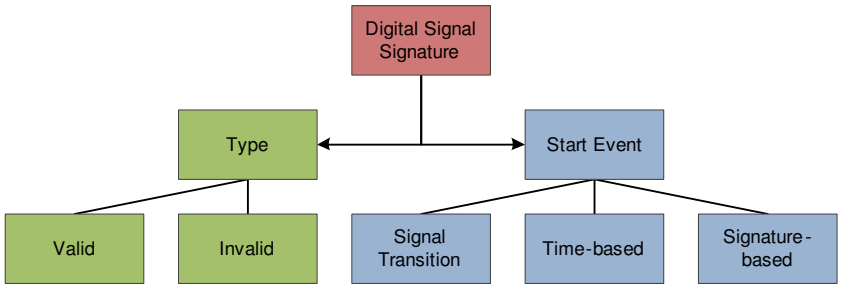


Figure 3.1: Digital signal signature taxonomy

invalid signatures in a digital signal proves the monitored device contains malicious behavior. A digital signal signature also has a *start event* which marks the beginning of the signature in the digital signal. The start event can be a signal transition, time interval, or even another signature. A signal transition start event simply enables the associated signature detector when a high-to-low and/or low-to-high signal transition is detected. For instance, the first high-to-low transition in a RS232 signal indicates the start of a packet. A timer-based start event enables the signature detector after a set number of clock cycles and is useful for periodic signatures. For some complex signals a simple transition or time-based start event may not be sufficient. In these cases, a signature can be specified as a start condition. The start event monitor continually searches for the signature start event and resets the signature detector once it's discovered. This will be discussed further in section 3.1.3.

3.1 System Monitor Design

Figure 3.2 shows a top-level diagram of the trusted system monitor. Several signals from untrusted ICs are simultaneously monitored to ensure the system is operating correctly. Each signal has an associated *Digital Signal Monitor (DSM)* that continually searches for a set of signatures within the signal. If a signal abnormality is detected, an alert signal along with the offending signal index is reported to the system.

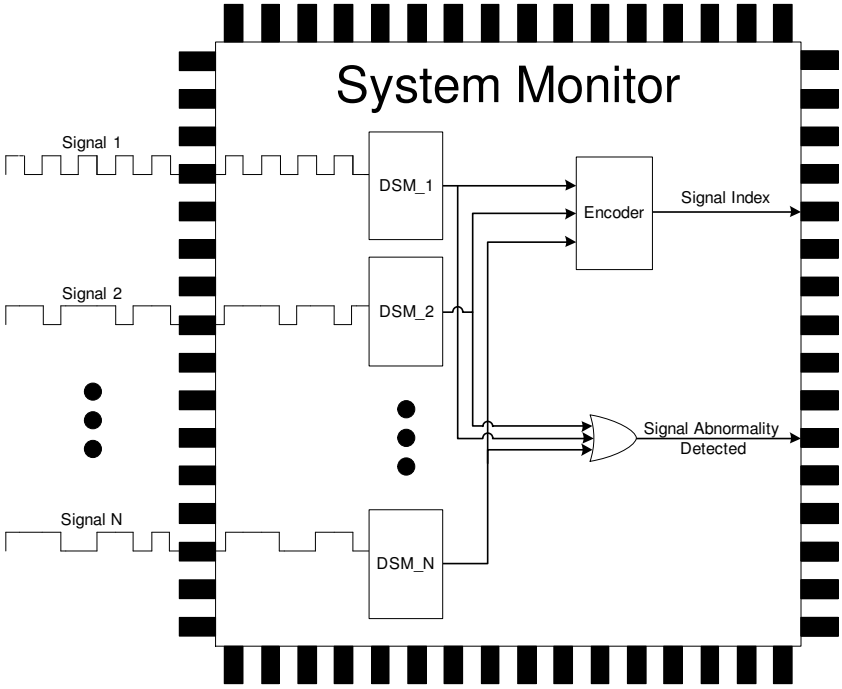


Figure 3.2: Top-level design of the trusted system monitor which is composed of several Digital Signal Monitors (DSMs) that each filter a digital signal of an untrusted IC for a set of signatures

3.1.1 Digital Signal Monitors (DSMs)

A digital signal monitor filters a signal for a set of signatures. Each signature has a corresponding hardware module called a *Signature Detector (SD)*. The SD searches for a specific digital signal signature. Each SD has an associated *Start Event Monitor (SEM)* which activates the SD by releasing the reset signal once the start event is discovered. All SDs that have the same start event share the same SEM.

An example digital signal monitor architecture is shown in Fig. 3.3. Blocks highlighted in blue, green, and red represent three typical interconnections of the DSM architecture. The blocks highlighted in blue show three valid signature detectors that have the same signal transition start event and are connected to the same start event monitor. Once the start event is discovered, the SEM activates the three SDs by releasing the reset signal. If one of the SDs detects a valid signature then all SDs will be reset once the next signal transition start event is detected. If all three SDs report a signature violation, then a signal abnormality is reported

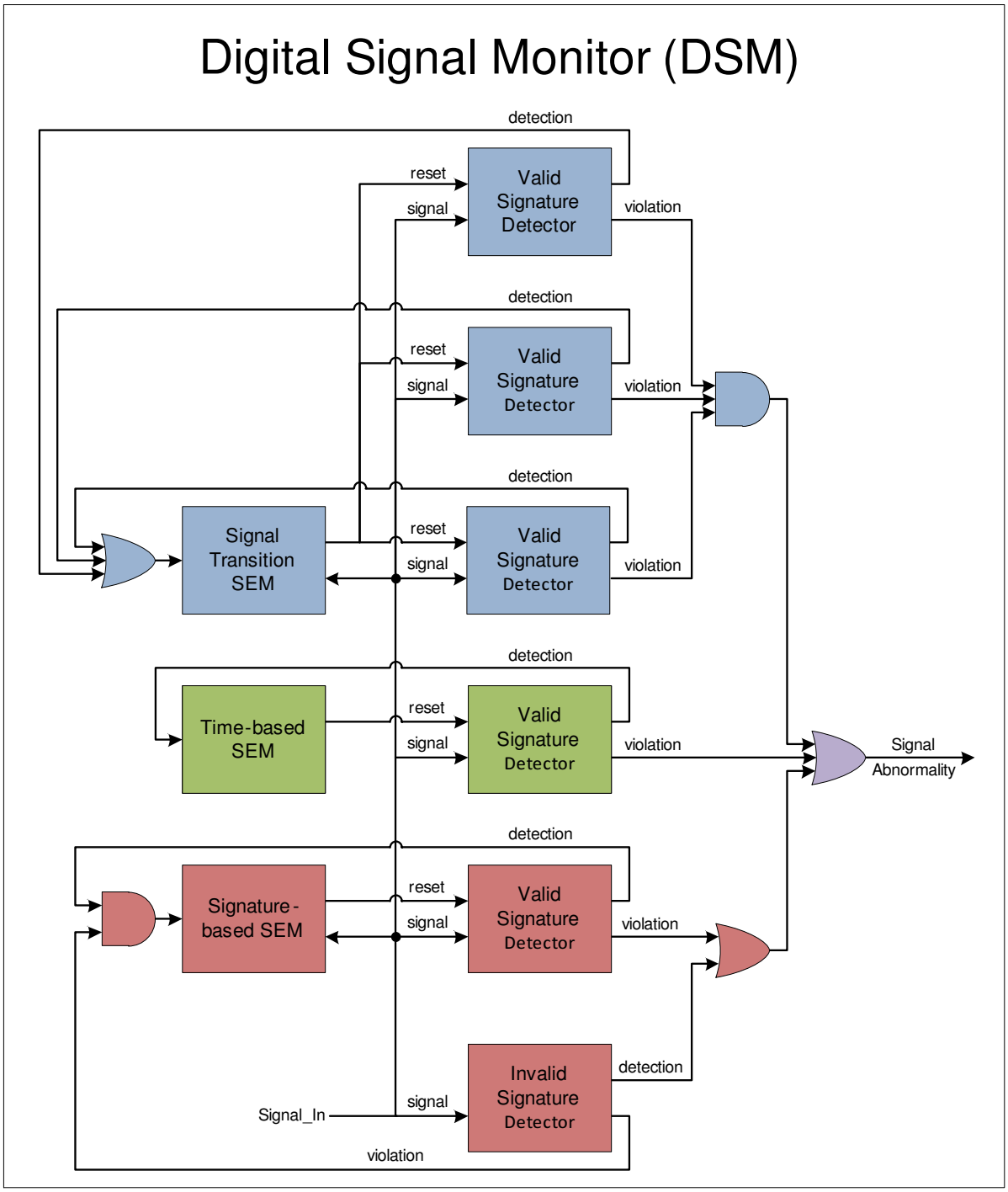


Figure 3.3: Example digital signal monitor architecture. Blocks highlighted in blue, green, and red represent three typical interconnections of the DSM architecture.

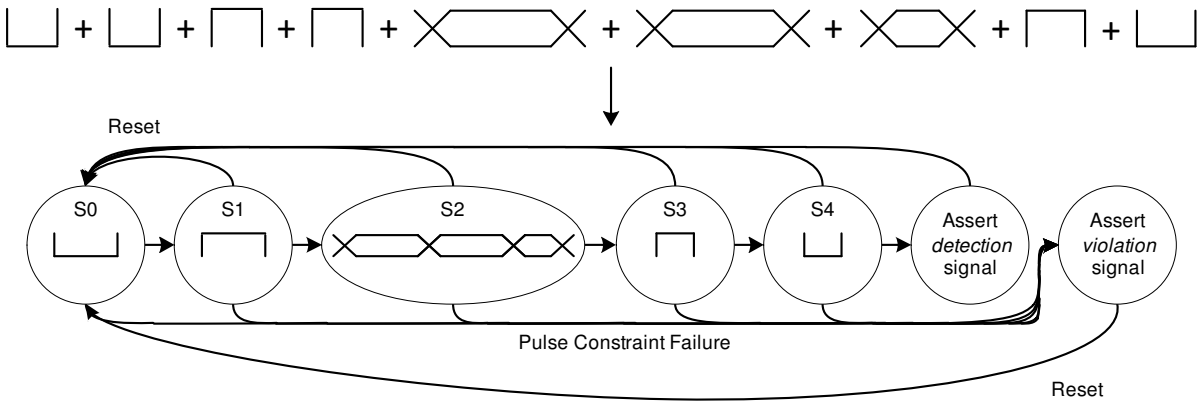
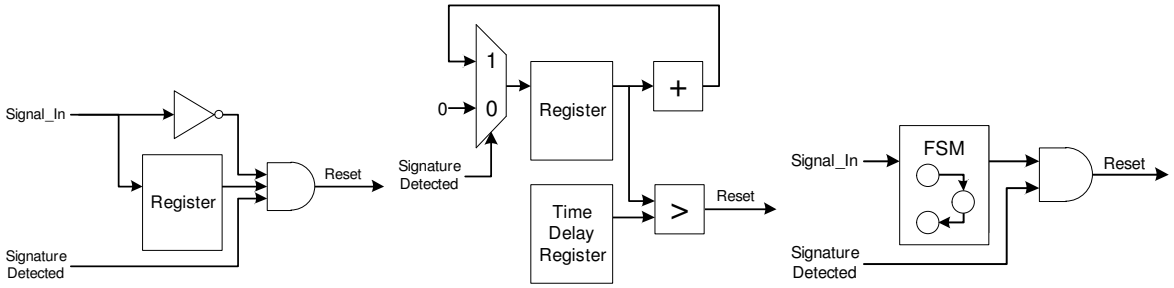


Figure 3.4: Example signature detector FSM

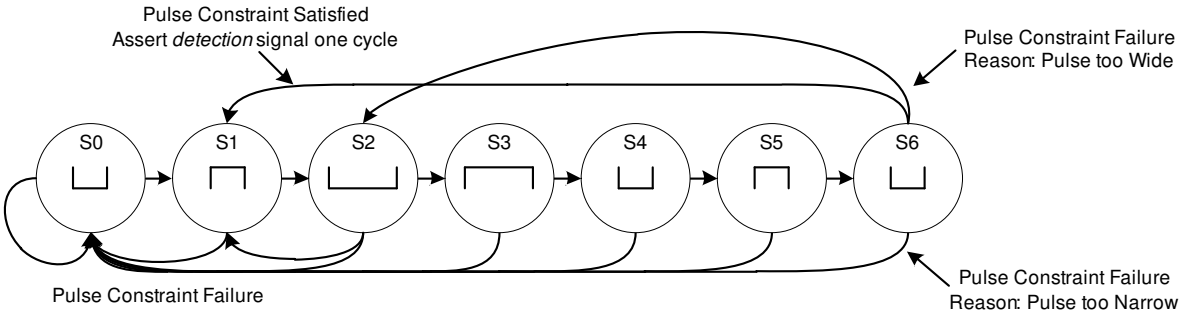
to the system monitor. The blocks highlighted in green show another start event monitor and associated signature detector. In this case, the SEM is time-based resetting the SD after a specific number of clock cycles. The blocks highlighted in red show a valid SD and an invalid SD connected to the same SEM. The detection of an invalid signature should assert the *signal abnormality* signal. To achieve this functionality, the *violation* signal of the valid signature block and the *detection* signal of the invalid signature block are OR'd together. If the valid signature is detected and the invalid signature detector reports a violation the SDs are reset when the next start event occurs.

3.1.2 Signature Detectors (SDs)

A unique *signature detector* is created for each signature to reduce overhead. The main component of the signature detector is a Finite State Machine (FSM). Figure 3.4 shows an example digital signal signature and the corresponding FSM. Each state of the FSM represents a pulse constraint or series of pulse constraints in which the logic-level constraint is the same. If the signal adheres to the pulse constraint(s) for a given state, the FSM transitions to the next state and finally asserts the *detection* signal. If any pulse constraint is violated, the FSM transitions to an error state in which the *violation* signal is asserted. The *reset* signal must be asserted to transition back to the initial state.



(a) High-to-low transition (b) Time-based (c) Signature-based



(d) Example signature-based FSM

Figure 3.5: Start event monitor design

3.1.3 Start Event Monitors (SEMs)

Each signature has a start event that marks the beginning of the signature within a signal. A start event is a change in signal state such as a high-to-low transition, a specified amount of time, or a combination of the two. A start event monitor effectively controls its associated signature detectors by resetting the SDs once a start event is discovered. It's designed to reset the SDs only if the *signature detected* signal is asserted. A low *signature detected* signal indicates the SDs are actively searching for a signature or a signal abnormality was discovered, neither of which warrants a reset. Note that the valid SDs are initialized to the *Assert detection signal* state and invalid SDs are initialized to the *Assert violation signal* state to address the startup case. Design details for the three types of SEMs are shown in Fig. 3.5.

The signal transition SEM registers the monitored signal and compares it to the unregistered signal to detect a change in signal state. A high-to-low signal transition SEM is shown in Fig. 3.5a. To create a low-to-high SEM, the inverter is simply moved to the registered signal.

A SEM can also be designed to detect both transitions by removing the inverter and XOR'ing the unregisterd and registered signals.

Figure 3.5b shows the design of a time-based SEM. This start event monitor resets its associated SDs after a certain number of clock cycles, independent of signal state. When the *signature detected* signal is asserted, an accumulator is enabled. The output of the accumulator is compared with a register value that corresponds to the time delay. Once the output of the accumulator is greater than the value of the time delay register, the SDs are reset.

When a start event cannot be uniquely identified by a signal transition or time delay, a signature-based start event monitor is used. Figure 3.5c shows the design of a signature-base SEM. A FSM continually searches the monitored signal for the start event signature. When the signature is discovered, the FSM outputs a one cycle pulse which will reset the associated SDs if the *signature detected* signal is high.

An example signature-based FSM is shown in Fig. 3.5d. States *S1, S4* and *S6* all contain equivalent pulse constraints and states *S1* and *S5* also share the same pulse constraint. If the pulse constraint of a given state is satisfied the FSM advances to the next state. If the pulse constraint in state S6 is satisfied, the FSM will output a one cycle pulse to the SEM indicating the start event was detected. The FSM will then transition to state S1 since states S0 and S6 have equivalent pulse constraints. If the pulse constraint of a given state is violated the FSM transitions to an earlier state rather than an error state. For instance, if the pulse constraint in state S6 is violated because the signal pulse is too narrow the FSM will transition back to S0. If the S6 pulse constraint is violated because the signal pulse is too wide the FSM will transition to S2 since states S4 and S5 are equivalent to states S0 and S1 respectively.

3.2 System Monitor Generation

While all digital signal monitors utilize a similar structure, each is specific to a set of signatures. In a real system this could result in hundreds of unique DSMs. Manually creating a system monitor containing hundreds of DSMs would be very tedious and error-prone. A system monitor generator tool is needed to automate the design process and ease the burden on the

system integrator. This tool should allow for the specification of digital signal signatures in a high level language and then translate this specification into HDL for the monitor design.

In this work we create a prototype System Monitor Generator (SMG) which is modeled after a similar design automation tool, Xilinx's CORE Generator. Xilinx maintains a set of hardware cores optimized for their FPGAs. Each hardware core is highly parameterized which allows the designer to customize the core for a specific application. CORE Generator aids the designer in modifying the appropriate parameters to achieve the desired functionality. The designer can customize a core using the CORE Generator GUI or a set of commands which the tool reads from a batch (.xco) file [72]. We developed a signature specification language, based on the CORE Generator batch file command syntax, to specify a set of digital signal signatures which implicitly specifies the customization parameters for a set of DSMs to create a system monitor. Our SMG tool translates a batch file containing a set of digital signal signatures into a HDL representation of the system monitor.

3.2.1 Signature Specification

A description of several common batch file commands is shown in Fig. 3.6a and an example batch file that specifies a customized distributed arithmetic FIR filter is shown in Fig. 3.6b. We leverage the *CSET* and *GENERATE* commands for our signature specification language. The CSET command is used to specify core properties. We use this command to set properties of a digital signal signature which will be described further in the following sections. The GENERATE command starts the customization process based on the parameters specified earlier in the file. We use this command to instruct the SMG to generate the HDL for the system monitor.

The CSET command typically equates a signal value to a single property. To reduce the space needed to specify several signatures, we condensed the typical CSET command syntax. We define two major properties, *start_event* and *constraint* (Fig. 3.7), to specify a set of digital signal signatures and create a customized system monitor. Each property takes multiple values and implicitly defines several other properties.

Command	Arguments	Function
CSET	<core_property=value>	Sets a core property value.
GENERATE	N/A	Elaborates the indicated command file.
SELECT	<core_name> <architecture> <vendor>	Selects the indicated core.
SET	<global_property=value> <project_property=value>	Sets a CORE Generator™ property value.
⋮	⋮	⋮

(a) Common batch file commands. Commands highlighted in blue were used in our signature specification language.

```
SET BusFormat = BusFormatAngleBracket
  ⋮
SET OutputProducts = ImpNetlist BmmFile
SELECT Distributed_Arithmetic_FIR_Filter Virtex Xilinx,_Inc. 7.0
CSET impulse_response = Symmetric
  ⋮
CSET coefficient_file = C:\coe_files\da_fir.coe
GENERATE
```

(b) Example batch file for a distributed arithmetic FIR filter

Figure 3.6: CORE Generator batch mode (source [72])

The *start_event* property defines a signature start event. The *signature type* and *signature #* values link the start event to the corresponding signature. If the signature doesn't exist it is implicitly declared. The *signal* value specifies the name of the signal being monitored. The *event type* denotes the type of start event using one of the three keywords: *transition*, *time-based*, or *signature-based*. The *event type value* indicates the specific start event value. For instance, the *event type value* for a *transition* start event is expressed using one of the three keywords: *falling_edge*, *rising_edge*, or *falling_and_rising_edge*.

The *constraint* property defines a pulse constraint for a signature. Similar to the *start_event* property, the *signature type* and *signature #* values indicate the corresponding signature and will implicitly declare a signature if it doesn't already exist. The *constraint #* is a value that indicates the relative ordering of the pulse constraints associated with a signature. The *signal*

CSET <**signature type**>_signature_<**signature #**>_start_event
= <**signal**> <**event type**> <**value**>

signature type: valid | invalid
signature #: positive integer representing the signature index
signal: name of the signal being monitored
event type: **transition** | **time-based** | **signature-based**
transition value: falling_edge | rising_edge | falling_and_rising_edge
time-based value: positive integer representing the time delay in clock cycles
signature-based value: positive integer representing the index of a defined signature

(a) *start_event* property definition

CSET <**signature type**>_signature_<**signature #**>_constraint_<**constraint #**>
= <**signal**> <**logic_level**> <<**width**> | <**min_width**> <**max_width**>>

signature type: valid | invalid
signature #: positive integer representing the signature index
constraint #: positive integer representing the constraint index
signal: name of the signal being monitored
logic_level: low | high | don't_care
width: positive integer representing the width of the pulse
min_width/max_width: positive integer representing the min & max widths of the pulse

(b) *constraint* property definition

Figure 3.7: Digital signal signature properties and values

value denotes the name of the signal being monitored. The *logic_level* value specifies the logic level constraint of a pulse and is expressed by one of three keywords: *low*, *high*, or *don't_care*. The *width*, *min_width*, and *max_width* values define the timing constraint of a pulse expressed in the number of clock cycles. If the exact timing constraint can't be determined, a minimum and maximum constraint must be specified.

3.2.2 SMG Prototype

The SMG is responsible for translating the signatures specification into HDL files. Figure 3.8 shows a high-level overview of our system monitor generator prototype. The SMG reads a *signature_spec.xco* file and translates this high level specification in several VHDL files. The top-

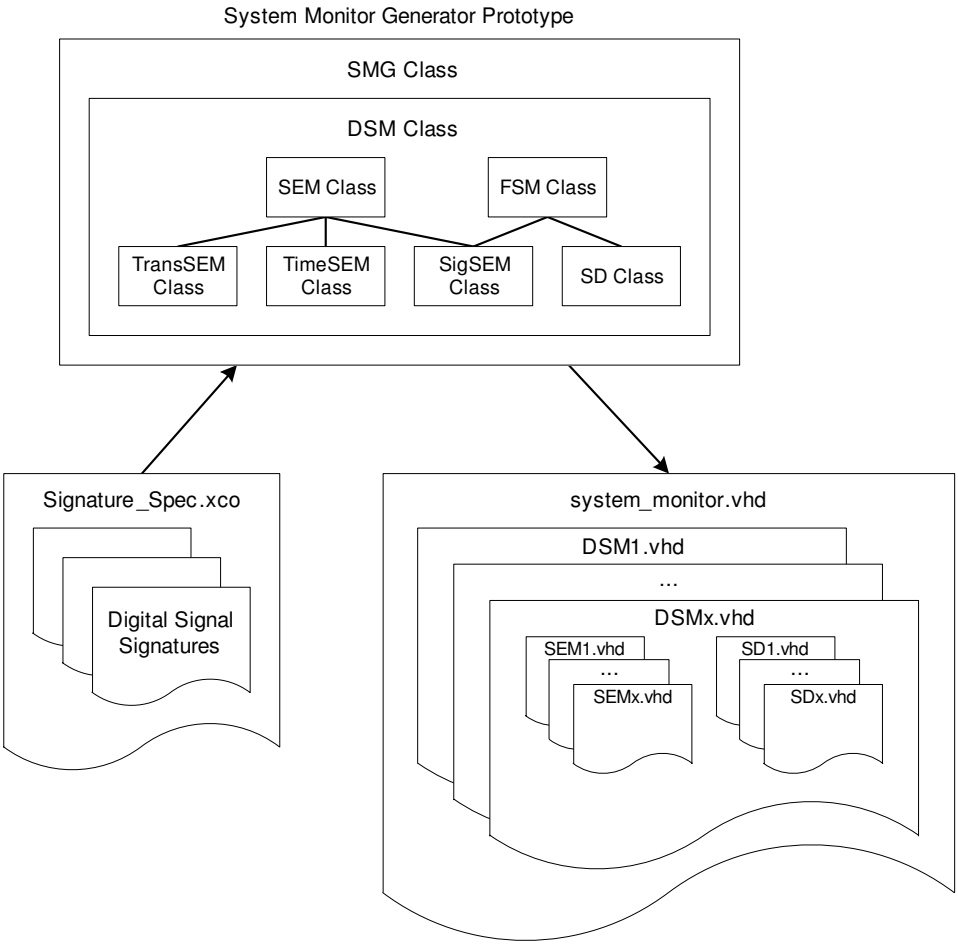


Figure 3.8: Overview of the system monitor generator prototype functionality

level system monitor file is *system_monitor.vhd*. This file instantiates and interconnects several digital signal monitors which are expressed in their respective *DSMx.vhd* files. Each DSMx.vhd file instantiates and interconnects several start event monitors and signature detectors expressed in their respective *SEMx.vhd* and *SDx.vhd* files. These files contain the descriptions necessary to develop customized hardware that monitors a set of digital signals for their corresponding signatures expressed in the signature_spec.xco file.

Our SMG prototype was developed using an object-oriented structure in C++. The program was divided into several classes, shown in Fig. 3.8, to manage code complexity and maintain organization. The *SMG class* is responsible for reading signature properties from the signatures specification file and writing the system_monitor.vhd file. When a signature

property is read it's sent to the appropriate *DSM object*. If an object doesn't exist for that signal, a new DSM object is instantiated. The *DSM class* interprets a set of received signature properties for a given signal and writes the corresponding DSMx.vhd file. Each property is transferred to the appropriate SEM or SD object. If an object doesn't exist for that signature a new object is instantiated. The *SD class* interprets a set of *constraint* properties for a given signature and writes the associated SDx.vhd file. Since the main component of a signature detector is a FSM and the signature-based SEM also implements a finite state machine, a FSM base class was created. This class provides the facilities to translate constraint properties into FSM states. The *SEM class* is a base class for the three types of start event monitor classes, *TransSEM, TimeSEM,* and *SigSEM*. It provides common functionality of a start event monitor to the derived SEM classes. Each derived SEM class interprets a start_event property and writes the appropriate SEMx.vhd file. Since the main component of the signature-based SEM is a FSM, the SigSEM class also inherits the FSM class.

3.3 System Monitor Realization

The physical hardware for a system monitor must have a high pin count to monitor signals from several ICs, capable of adequately sampling multiple digital signals simultaneously, and have the area resources necessary to support many digital signal monitors. The inherent parallelism and performance capabilities of a (semi-)custom hardware solution is ideal for this type of application. There are several hardware options to choose from including a fully customized hardware solution, standard cell Application Specific Integrated Circuits (ASICs), and Field-Programmable Gate Arrays (FPGAs). A fully customized hardware solution offers the best performance but the design-time and non-reoccurring engineering (NRE) cost makes this option not feasible unless the system is produced in extremely high volumes [38]. In most situations, it will be more feasible to implement the system monitor in an ASIC or FPGA. Choosing the best option will depend on the type of system.

3.3.1 Performance/Cost Considerations

An ASIC has a significantly higher Non-Recurring Engineering (NRE) cost and development time but is also capable of higher operating speeds, higher area density, and lower power consumption. An ASIC might be necessary for systems in which high frequency signals are monitored, the set of signature is very large, or low power is a primary concern. Also, if the system is produced in high volumes such that the higher NRE cost is offset by the lower cost per unit, it might be the more economically sound option. FPGAs have a higher cost per unit and lower performance but have a significantly faster development cycle. Also the reconfigurability of an FPGA could be a particularly useful feature in multifunction systems. A custom system monitor configuration can be designed for each function of the system. If the higher performance of an ASIC is not required, an FPGA will be the better choice as it will most likely be cheaper, is more flexible, and will allow for a faster time to market.

3.3.2 Secure Process Flow

Another consideration is the security of the ASIC and FPGA development processes. The system monitor must be developed using a secure process flow. Currently the domestic IC supply chain primarily supports ASIC development, offering several options for the implementation of an ASIC system monitor. Leading FPGA companies, like Xilinx and Altera, use the fabless business model and rely on offshore companies for fabrication. However, FPGAs are inherently less susceptible to a hardware Trojan attack since the hardware configuration is not known during the development cycle. Also, recently FPGA startup company Achronix announced it has partnered with Intel [45] and will be the first commercial FPGA company with a domestic end-to-end supply chain [44]. Their new high-end *Speedster22i* FPGA, built using Intel's 22nm 3-D Tri-Gate transistor technology, would be an ideal hardware device for a system monitor requiring this level of performance.

CHAPTER 4. EXPERIMENTAL ANALYSIS

In this section we conduct experiments that demonstrate the security and overhead of our persistent monitoring framework. We examine a situation in which the *TXD* signal of an RS232 core embedded in an untrusted IC is monitored to partially verify the functionality of the circuit. Ten hardware Trojan benchmarks are embedded into the RS232 core, each with a different payload. Four system monitors are created, each with a unique signature specification, to detect a Trojan activation. The effectiveness and overhead of each system monitor is evaluated.

This experimental setup is consistent with a situation in which a hardware Trojan is embedded into a RS232 core during the IC development cycle to modify the circuit's functionality or transmit information from the chip. The attacker could infiltrate any stage of the IC development cycle to embed the hardware Trojan – including the third-party company which developed the RS232 core, the EDA toolset company that translated the RS232 core into low level design files or the foundry where the IC was fabricated. We address forty different scenarios in which the system integrator, who is implementing the IC in his/her system, has one of four levels of knowledge regarding the functionality of the RS232 core which contains one of ten possible hardware Trojans.

4.1 Experimental Setup

Figure 4.1 shows the experimental setup consisting of a RS232 core transmitting ten packets which are sent in a burst. The packet structure includes one start bit, eight data bit in which the least significant bit is transmitted first, and one stop bit. The baud rate is 9600 which corresponds to a single pulse width of 104320ns.

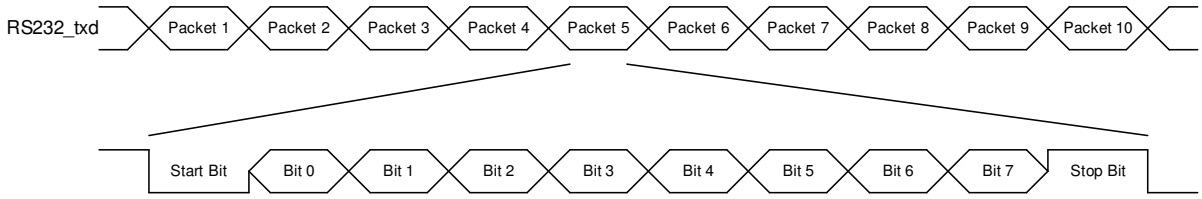
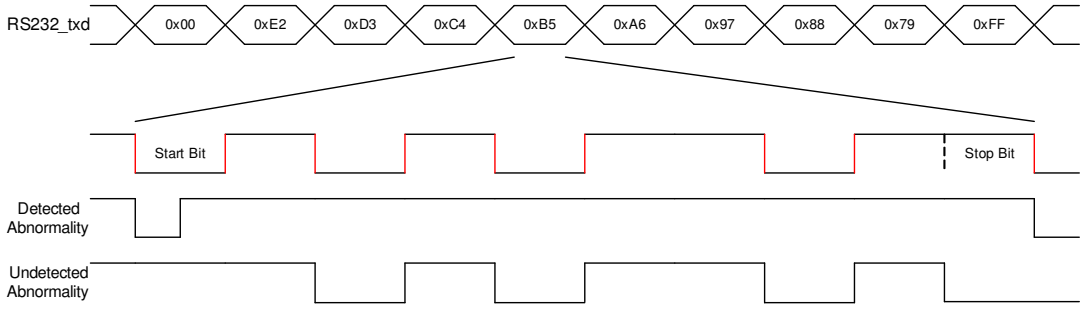


Figure 4.1: Experimental setup consisting of an RS232 core embedded in an untrusted IC that transmits ten packets

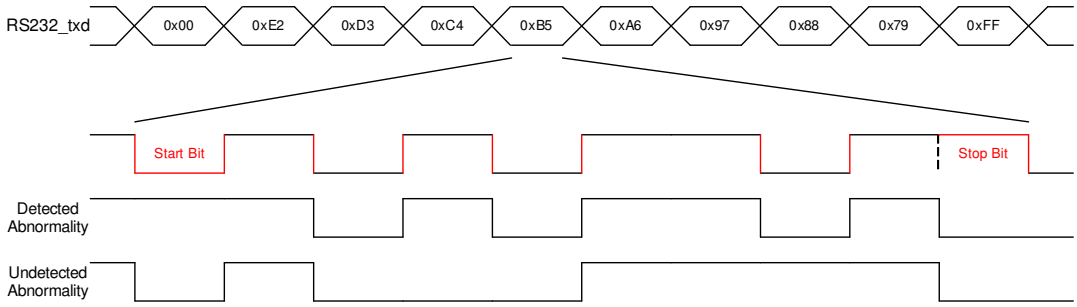
4.1.1 Signature Specifications

The system integrator must design signature specifications based on information known about the system operation prior to runtime, which we refer to as *static system knowledge*. The detail of this static system knowledge is dependent on the type of system and type of ICs which are monitored. We have developed a flexible persistent monitoring framework that allows for signature specifications of varying detail. Utilizing this functionality, we develop four signature specifications *SS1, SS2, SS3,* and *SS4* of varying resolution. SS1 has a minimal amount of detail consistent with a situation in which the system integrator has a limited static knowledge of the digital signal. Signature specifications SS2 and SS3 have a moderate amount of detail. SS4 contains the maximum amount of detail, consistent with a situation in which the system integrator has complete knowledge of the signal prior to runtime. Figure 4.2 shows a visual representation of each signature specification. Portions of the signal highlighted in red are constrained by the signature specification. A hardware Trojan payload that modifies this portion of the signal will be detected by the system monitor implementing this signature specification.

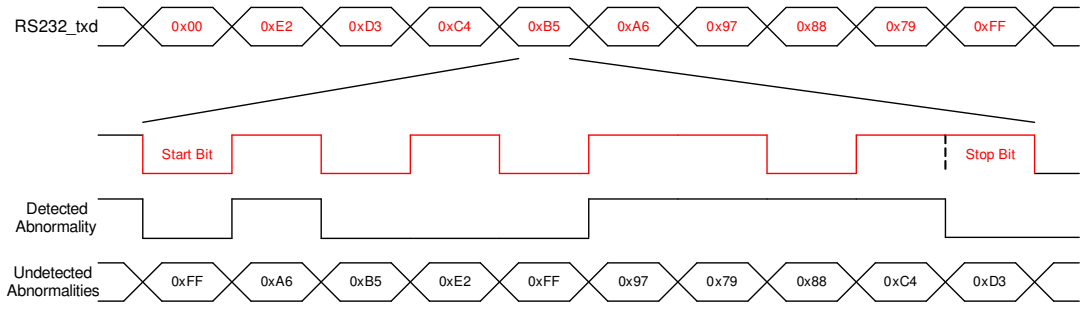
SS1, shown in Fig. 4.2a, contains the least amount of detail. It places a minimum and maximum width constraint on each pulse but doesn't require a specific logic level. The minimum pulse width constraint corresponds to the width of a signal pulse at a baud rate of 9600. The maximum pulse width constraint corresponds to nine consecutive single pulse widths that could occur either by a data value of 0x00 following the start bit or a data value of 0xFF followed by the stop bit. This will detect an abnormally small or large pulse width but will not maintain a strict packet structure. Note this does impose a loose constraint on time delay between packets.



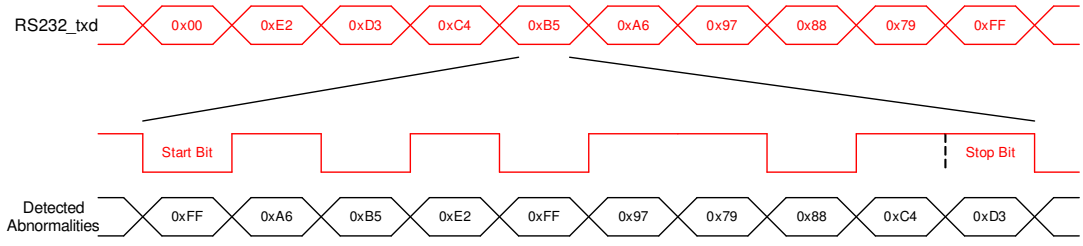
(a) SS1: A pulse must between one and nine standard pulse widths



(b) SS2: A packet contains 10 pulses. The start bit must be low and the stop bit must be high. The data pulses can be high or low but must be a standard pulse width.



(c) SS3: Each packet must be one of ten allowed packets.



(d) SS4: The exact packet order and inter-packet delay is also constrained.

Figure 4.2: RS232 signatures specifications. Portions of the signal highlighted in red are constrained by the specification.

SS2, shown in Fig. 4.2b, adds more detail to SS1 by including a signature that specifies the RS232 packet structure. This includes a single pulse start bit that is constrained to a low logic level, followed by eight data bits that can be either a high or low logic level but are constrained to a single pulse width, and then a single pulse stop bit that's constrained to a high logic level. In this specification, the signal must be adhere to the RS232 packet structure but the data content isn't constrained.

SS3, shown in Fig. 4.2c, assumes a situation in which the system integrator knows the valid data content of all transmitted packets but doesn't known that order in which they're transmitted. This signature specification adds more detail to SS2 by specifying a unique signature for each packet in which the logic level constraint for each data bit is defined.

SS4, shown in Fig. 4.2d, takes this one step further by defining the order in which the packets will be transmitted. This assumes the system integrator has complete knowledge of the signal prior to runtime. Any Trojan payload that modifies the TXD signal will be detected.

4.1.2 Benchmarks

Ten trust benchmarks were used to evaluate the effectiveness of the four RS232 specifications. Each benchmark consists of a RS232 core which contains an embedded hardware Trojan. Six of the benchmark Trojans modify the functionality of the RS232 core while the other four transmit secret information. The hardware Trojans are lightweight and would likely go unnoticed by the conventional one-time implementable verification methods.

4.1.2.1 Modify Functionality Benchmarks

Six of the trust benchmarks modify the functionality of the circuit. These benchmarks were obtained from Trust Hub, a website developed by leading hardware trust researchers to reinforce continuity in the hardware security and trust community [2]. Part of Trust Hub's mission is to establish a set of trust benchmarks. Currently, 22 benchmarks have been created each of which consists of a RS232 core with an embedded hardware Trojan. Each Trojan has a unique trigger and payload combination. The majority of the benchmarks include Trojans that

are difficult to activate requiring the assertion of an internal set of signals. These benchmarks are designed to test Trojan activation methods. Six of the benchmarks include Trojans that are easily activated by an input sequence. These benchmarks are designed for detection methods that focus on detecting the malicious payload rather than activating the Trojan. Since our method is persistent, it's irrelevant how the Trojan is activated. We are interested in the ability to detect a Trojan Payload. We use these six benchmarks to evaluate our signature specifications.

4.1.2.2 Transmit Information Benchmarks

The remaining four benchmarks, *RS232-EXTRA-BAND, RS232-HIGHER-BAND, RS232-VAR-DELAY,* and *RS232-VAR-WIDTH*, transmit secret information from the circuit. We created these benchmarks using our own experience in designing hardware Trojan prototypes (see Appendix A) and hardware Trojan design case studies [15, 32]. These benchmarks are triggered after a set amount of time and leak information in a covert manner that will not affect the functionality of the circuit. This type of hardware Trojan would be particularly difficult to detect during IC verification and would likely go unnoticed.

The RS232-EXTRA-BAND benchmark simply waits until the RS232 core is not transmitting a packet and then leaks the information using the extra bandwidth that's not utilized by the main system. The RS232-HIGHER-BAND benchmark embeds a higher bandwidth RS232 signal within a lower bandwidth RS232 signal as described in [15]. When the baud rate is 9600 and the Trojan trigger counter has expired, the RS232 core is switched to a baud rate of 115200 and leaks information in a manner in which the core appears to still be transmitting data at a baud rate of 9600. The RS232-VAR-DELAY benchmark uses inter-packet delay to leak information. Once the Trojan is triggered it waits for a scenario in which inter-packet delay is constant. It then starts leaking information by doubling the inter-packet delay when leaking a bit value of 1 and leaving the delay unmodified when leaking a bit value of 0. The RS232-VAR-WIDTH benchmark adjusts the width of a single pulse to leak information. The first three benchmarks are consistent with a situation in which the attacker has access or can gain

access to a system that's connected via RS232 to the system being monitored. This benchmark assumes the attacker has access to the physical RS232 wire. In this case, the Trojan payload can be more subtle. Once this hardware Trojan is activated, an extra cycle is added to any single pulse to leak a bit value of 1 and left unmodified to leak a bit value of 0.

4.2 Results

Each signature specification was transformed into the HDL for a system monitor using our prototype system monitor generator. This HDL was then used to evaluate the overhead induced by each signature specification as well as the effectiveness of each system monitor in detecting hardware Trojan activations in the trust benchmarks.

4.2.1 Overhead

As mentioned in Section 3.3, FPGAs are an ideal hardware platform for implementing a system monitor. We use three FPGAs, the Xilinx *XC3S100E, XC4VFX40,* and *XC5VFX70T*, to evaluate the overhead associated with each signature specification. These FPGAs represent three distinct performance levels. The XC3S100E is a low performance/economical FPGA, the XC4VFX40 is a moderate performance/moderately priced FPGA, and the XC5VFX70T is a high performance/high cost FPGA. Bitstreams were generated for each FPGA using the Xilinx ISE toolset. The results are shown in Table 4.1.

The area overhead induced by each signature specification, in terms of registers and LookUp Tables (LUTs), for the XC3S100E FPGA is shown in columns 2 and 3 of table 4.1. As expected, the more detailed signature specifications induce more area resources. Since the order of the RS232 packets are not preserved in SS3, the system monitor must search for each packet in parallel which means ten separate signature detectors are needed. This is why the area overhead is considerably higher for SS3. In SS4, the packet order is fixed which means only a single signature detector is needed to search for all ten packets. Even though SS4 contains significantly more detail that SS1 and SS2, it doesn't use that many more area resources. The absolute flip flop and LUT utilization is only shown for the XC3S100E FPGA but it's

Table 4.1: Overhead of the system monitor created for each signature specification

Signature Specification	**XC3S100E**			**XC4VFX40**	**XC5VFX70T**
	Regs	LUTs	Delay	Delay	Delay
SS1	20 (1%)	39 (2%)	5.966ns	3.444ns	2.714ns
SS2	57 (2%)	200 (10%)	7.090ns	3.683ns	2.869ns
SS3	237 (12%)	894 (46%)	6.226ns	3.478ns	2.685ns
SS4	87 (4%)	280 (14%)	7.689ns	4.037ns	3.139ns

approximately the same for the other two FPGAs. Note however that the other two FPGAs contain considerably more area resources, so the signatures specifications induce a much smaller relative utilization on these FPGAs.

Columns 4-6 of table 4.1 show the delay overhead induced by each signature specification. As expected, SS1 induces the smallest delay as it contains the least amount of detail and monitoring hardware. SS4 has the largest delay as it contains the most detail and most involved signature detector. An interesting observation is that SS2 induces more delay than SS3 even though SS3 is a more detailed signature specification. This is because extra logic is needed in the SS2 signature detector, to support the *don't_care* logic_level pulse constraints, which increases the circuit's critical path. Note that the system monitor delay is very technology dependent. A system monitor implemented on the XC5VFX70T FPGA is approximately twice as fast as a system monitor implemented on the XC3S100E FPGA. This should be considered when selecting a device for a system monitor in which high frequency signals are monitored. The system monitor must be clocked at least twice as fast of the maximum frequency of the digital signals that it's monitoring (the Nyquist rate) to adequately sample it.

4.2.2 Effectiveness

The four system monitors, which implemented signature specifications SS1-SS4, were connected to the TXD signal of each RS232 trust benchmark and simulated using ModelSim. The results are shown in table 4.2. As expected, the more detailed signature specifications detected a greater number of Trojan activations. An interesting observation is that the system monitor which implemented the least detailed signature specification, SS1, still detected the Trojan

Table 4.2: Results of trust benchmark simulations. An "X" indicates a detection of the hardware Trojan activation while a "–" indicates the contrary.

Benchmarks	**Trojan Payload**	**SS1**	**SS2**	**SS3**	**SS4**
RS232-TL08C0FPI0	Enables embedded inverters to increase circuit temperature	–	–	–	–
RS232-TR0CS02PI0	Changes one bit of the *TXD* signal	–	–	X	X
RS232-TR0ES12PI0	Changes four bits of the recieved data value	–	–	–	–
RS232-TR0FS02PI0	Disables the transmission part of the core	X	X	X	X
RS232-TR2AS0API0	Changes two bit of the *TXD* signal	X	X	X	X
RS232-TR30S0API0	Modifies the functionality of the transmission FSM	X	X	X	X
RS232-EXTRA_BAND	Transmits information using extra bandwidth	–	–	X	X
RS232-HIGHER_BAND	Transmits information via a higher bandwidth RS232 signal	X	X	X	X
RS232-VAR_DELAY	Transmits information by varying the delay between packets	–	–	–	X
RS232-VAR_WIDTH	Transmits information by varying the width of signal pulses	X	X	X	X

activation in half the benchmarks. This indicates that even simple constraints on a digital signal can provide a significant level of confidence that the circuit is performing correctly. Two benchmarks, *RS232-TL08C0FPI0* and *RS232-TR0ES12PI0*, were not detected by any system monitor. This is because the payload of these Trojan benchmarks didn't directly affect the TXD signal. However, in a real-world scenario it's possible these Trojans would still be detected. The RS232-TL08C0FPI0 benchmark activates inverters to increase the chip temperature to a point where performance is degraded and/or the circuit fails. All four system monitors would detect a circuit failure and may also detect performance degradation depending how it affects the RS232 core. The RS232-TR0ES12PI0 benchmark modifies the received value. This may produce an unexpected output that a system monitor detects.

CHAPTER 5. CONCLUSIONS

5.1 Summary of Results

In this work we proposed a more realistic, comprehensive solution to IC hardware trust verification in which persistent verification of ICs in the field is used in addition to one-time implementable methods. We develop a persistent verification framework in which a few ICs from a secure design flow are used to monitor several untrusted digital ICs. We design a system monitor that filters the IO of untrusted digital ICs for a set of signatures that indicate the IC adheres to its original specification and has not been compromised. A high level signature specification language is developed to easily specify these signatures. A design automation tool is created to translate signatures expressed in this high-level specification into HDL. We also discuss the types of physical hardware that would be suitable for the system monitor implementation.

We developed four trust benchmarks based on our experience in designing hardware Trojan prototypes (see Appendix A) and obtained six other benchmarks from Trust-hub to evaluate the effectiveness and overhead of our approach. Four system monitors were created with unique signature specifications that varied in detail. Experimental results showed even the system monitor with the least detailed signature specification detected a significant number of Trojan activations with reasonable overhead.

Our persistent monitoring approach compliments the one-time implementable methods which currently represent the majority of the hardware trust verification research. One-time methods attempt to activate and detect the presence of a hardware Trojan during the verification stage of the IC development cycle. However, Trojans are inherently difficult to activate and are also very hard to detect if the Trojan doesn't represent a significant portion of the

entire chip. Our approach is immune to the issue of Trojan activation since it's persistently enabled and the ability to detect a Trojan activation is independent of the Trojan to original circuit ratio.

5.2 Future Work

Our monitoring approach has made significant contributions to the persistent verification of digital ICs, but there's still considerable room for improvement in future work. One of the limitations to our approach is that hardware Trojans must have a payload that alters the output of an IC to be detected. While this includes most hardware Trojans, it will not detect hardware Trojans that covertly leak information from the chip. Future work could explore the persistent monitoring of side channels such as power, temperature, and electromagnetic analysis to add further Trojan activation detection capabilities. This could also allow for the monitoring of analog ICs.

Our persistent verification framework supports digital signal signatures defined for one signal with static constraints. This could be expanded to support more complex signatures that include dependencies between signals and dynamic constraints that describe the functionality of a circuit in greater detail.

The digital signal signatures must remain protocol-independent to address the many different protocols present in a large system. However, another layer of abstraction could be added to the system monitor generator that's protocol-depended to ease the burden on the system integrator. For instance an Ethernet class could be added to the SMG that translates a set of Ethernet parameters into digital signal signatures.

Currently, the system integrator is responsible for defining digital signal signatures. A tool could be developed to automatically define signatures from simulation data that describes the circuit's typical behavior. Such a tool could implement an algorithm to maximize signal coverage given the available monitoring resources. One of the key aspects of our approach is the unpredictable nature of the signature specifications. To maintain this property, randomness must be introduced to the signature selection algorithm to prevent a deterministic selection

process that an attacker could replicate.

The effectiveness of our approach could be reevaluated with a more realistic experimental setup and a more comprehensive set of trust benchmarks. Also, a more complete investigation of overhead is needed to determine how this approach scales and the feasibility of implementing this framework in a larger system.

The system monitor architecture could be modified to utilize the partial reconfiguration capabilities of a FPGA. In this case, more customized DSMs could be swapped in during specific system operations to offer high resolution monitoring. Also, this architecture may be useful when monitoring resources are limited. If all IO signal do not need to be monitored simultaneously, DSMs could be swapped in as needed.

APPENDIX PROOF OF CONCEPT

The 2010 Computer Security Awareness Week (CSAW) Embedded Systems Challenge hosted by the Polytechnic Institute of NYU presented student-led teams around the country with a hardware hacking challenge. Teams were given the RTL code for two different designs as well as a BASYS 2 evaluation platform with a Xilinx Spartan3E-100 Field Programmable Gate Array (FPGA). The two designs contained a ring oscillator-based hardening technique [58] to detect any design modification. The challenge was to surreptitiously embed malicious circuitry, also known as hardware Trojans [35], into the design. The following describes our experiences in circumventing the hardening technique and developing proof-of-concept hardware Trojans.

Competition Details

Our team concentrated the attack effort on the second design, codenamed "Beta", as we considered this to be the more vulnerable of the two designs. The Beta design consisted of an adder circuit with several embedded ring oscillators. A ring oscillator is a delay loop circuit, typically composed of wires and inverters, that oscillates at a particular frequency. This frequency is very sensitive to wire length, gate delay, and process variation [65, 24, 40]. In the Beta design, the embedded ring oscillators were inserted with the purpose of detecting any modifications to the hardware. Such modifications would change the wire lengths of the ring oscillator, resulting in a discrepancy in frequency values. The Beta design also included functionality that allowed the user to initiate a challenge for a particular ring oscillator value and receive the response via the 7-segment display on the BASYS 2 board. According to the challenge rules, all ring oscillator responses had to remain within 6.6% of their original value. This paper describes our successful circumvention of the Beta hardening scheme and describes

the design methodology we used to create a large and diverse set of hardware Trojans.

Attack Methodology

We implemented two different approaches to circumventing the ring oscillator-based protection mechanism of Beta:

- A *design lockdown* approach that fixed the location of the ring oscillators.
- A *ring oscillator emulation* approach that reproduced the functionality of the ring oscillators with a look-up table.

Design Lock Down

We first considered that changes made to the design would by default result in a modified physical placement and routing (P&R) of the ring oscillators. As noted previously, the queried ring oscillator value is by design very sensitive to wire length and other physical variations. By default, the Xilinx tools for P&R are completely automated to maintain high designer productivity but also offer options to give the designer more control. These options could be used to manipulate the location of the ring oscillators and fix the challenge responses while arbitrarily changing the rest of the design.

We used the Xilinx PlanAhead software (specifically the Floorplanner utility) to analyze the Beta design and to specify additional P&R constraints. With PlanAhead, we extracted the FPGA constraints file which we then applied to a compromised version of the Beta design. Using this method, the effect of the hardening scheme was reduced, but not completely removed. This allowed our team to implement a few small hardware Trojans that would have been detected without the use of this technique, but large Trojan designs still produced ring oscillator responses outside the acceptable variation. This was due to the fact that constraint violations were necessary to create valid placement and routing schemes for large designs which physically altered the ring oscillators.

Ring Oscillator Emulation

From a black box perspective, the Beta design takes in a sequence of two 8-bit values and outputs a corresponding 16-bit value. We reasoned that this functionality could be replaced by a module that would capture the input values and then fetch the corresponding output value from a lookup table, effectively emulating the ring oscillators. One issue with this approach was that inter-FPGA process variations would cause a slight difference between the correct response values on our FPGA and the correct response values on the FPGA used to evaluate our design. However, given the wide range of acceptable responses, the outputs from our chip would most likely fall within the contest's allowable variation [43]. Another issue was storing the entirety of the challenge-response pairs, for as the Beta design required a 30KB (for 15,360 16-bit challenge-responses) lookup table, there was only 9KB of dedicated memory available on the Spartan3E-100 FPGA.

Further testing led to two key observations regarding the ring oscillator frequency values. First, for a given input, the lower 5-6 bits of the ring oscillator response varied in a seemingly random manner. Therefore, it wasn't necessary to store these bits in the lookup table as they could be reproduced with a pseudo-random number generator. Second, not all challenge vectors produced oscillations and there appeared to be long strings of 0's and 1's in the list of ring oscillator responses. We hypothesized it would be possible to compress the ring oscillator frequencies to a size that would fit on the FPGA, but needed to further characterize the ring oscillators to confirm our theory.

This ring oscillator characterization required multiple testing and recording passes of the output values, in order to analyze average responses and the amount of variation. To manually conduct this testing procedure would have taken several weeks to complete and would have been very tedious and error-prone. Consequently, our team attempted two different approaches to obtain these ring oscillator values.

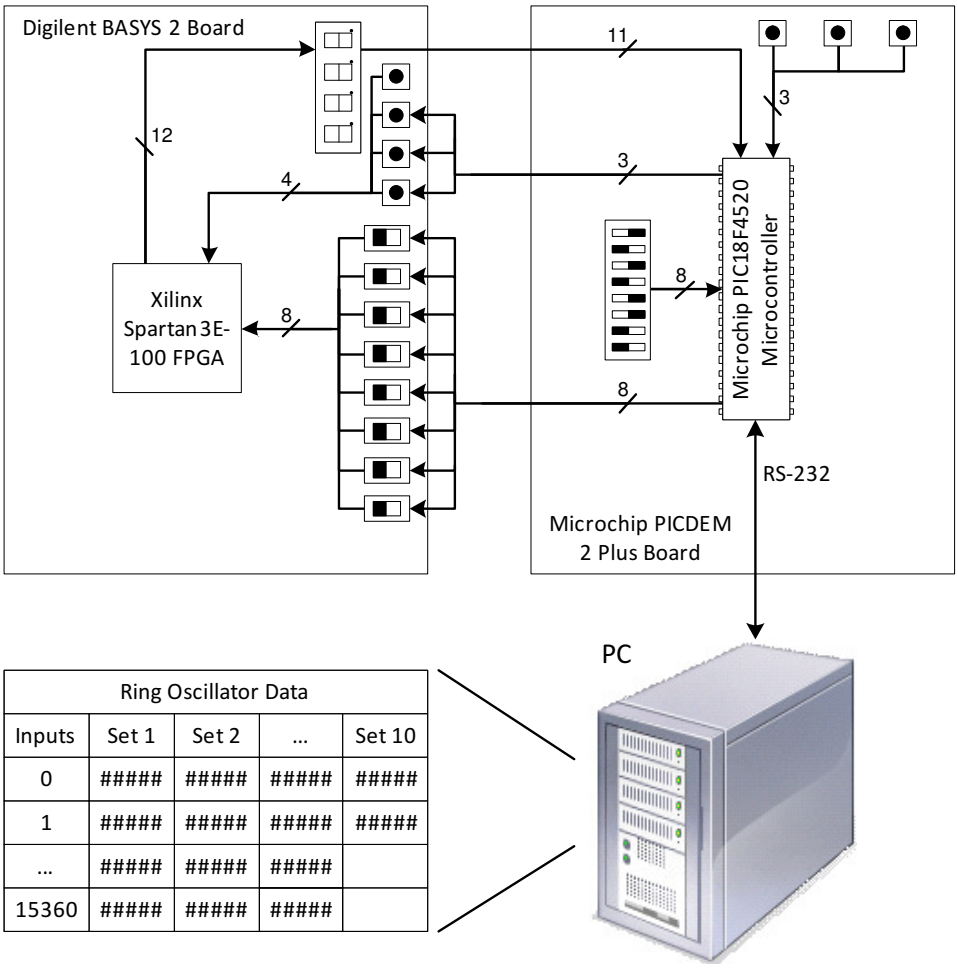


Figure A.1: Automated testing environment

Post-P&R Simulation

The Xilinx synthesis tools have the capability to generate timing models after each step of FPGA design generation. For example, after P&R, a Verilog or VHDL model of the design is created with components broken down into individual slices and LUTs, and timing information is added as delays for each individual component. The ModelSim HDL simulator was used to generate the timing results. Unfortunately, responses were off by a significant margin because the average-case timing information was not sufficiently accurate to model the ring oscillators. However, this simulation did correctly determine which challenge vectors produced oscillations and long strings of 0's and 1's. This confirmed that the dataset was compressible.

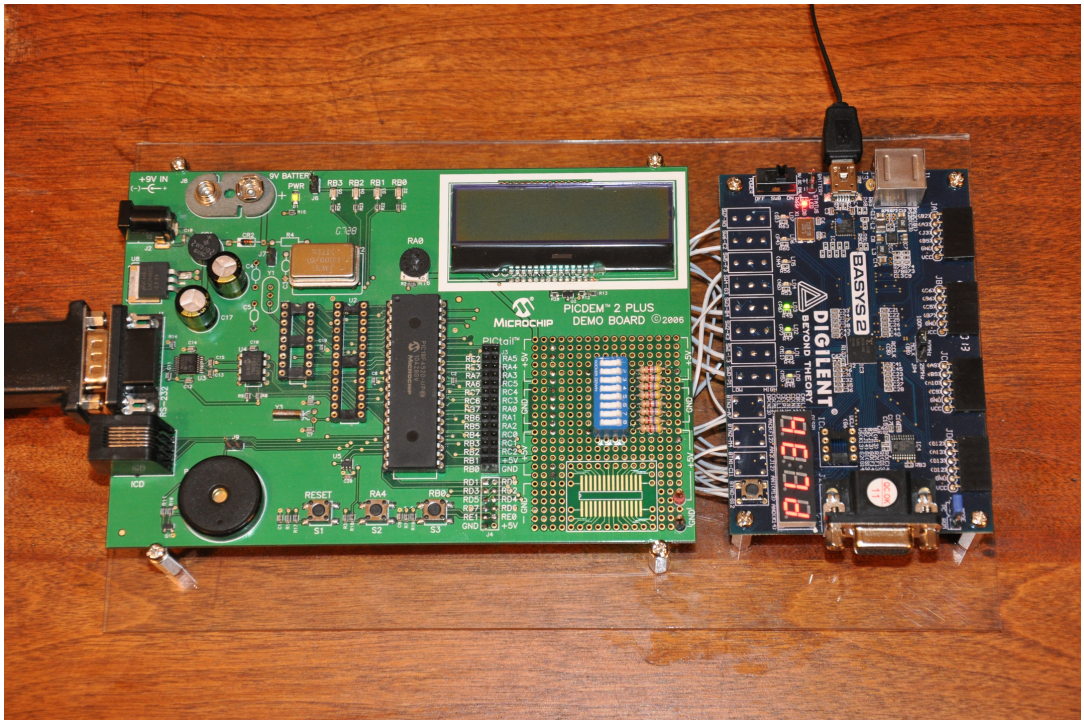


Figure A.2: Recording ring oscillator responses for all challenge vectors

Automated Testing Environment

We decided to take the direct approach of automating the testing procedure with additional hardware to obtain the ring oscillator values. Figure A.1 shows our automated testing environment. The input and output pins of the FPGA were connected to a microcontroller on a PICDEM 2 Plus evaluation board. A PC sent input vectors to the microcontroller, which set the appropriate inputs on the FPGA. The microcontroller then read the 7-segment display and sent the value back to the PC where it was analyzed and recorded. Manual testing was still possible via the switches and pushbuttons on the PICDEM 2 Plus board. A picture of the testing setup is shown in fig. A.2. The automated testing hardware reduced the time to test the Beta design from a period of several weeks to a few hours.

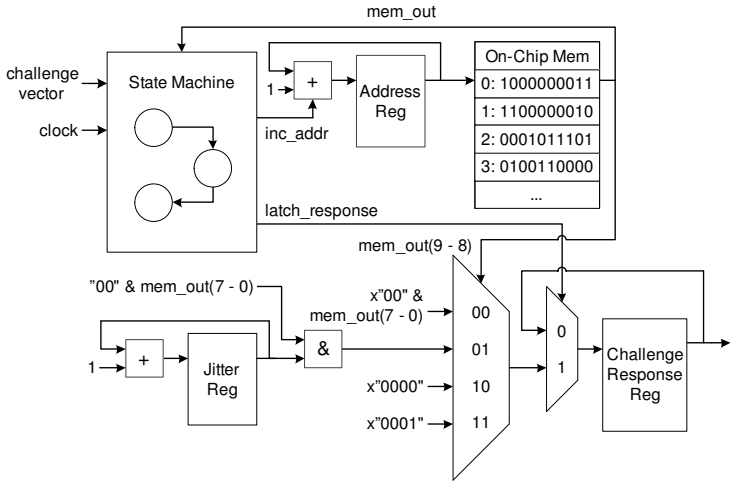
Using the automated testing hardware, we recorded each ring oscillator response 10 times for all 15,360 inputs. The 10 datasets were analyzed to determine the average ring oscillator response for each input and amount of variation present within the data. Three distinct groups

Ring Oscillator Frequency Compression Scheme	
Bit #	Compression Encoding Rules
9	If string of 0's or 1's, set to 1 If single RO frequency, set to 0
8	If Bit 9 == 1, set to RO_Freq(0) If Bit 9 == 0, set to RO_Freq(14)
7 - 0	If Bit 9 == 1, set to string length If Bit 8 == 1, set to RO_Freq(13 downto 6) If Bit 8 == 0, set to RO_Freq(7 downto 0)

Challenge Vector	Response
0x0000	0
0x0001	0
0x0002	0
0x0003	1
0x0004	1
0x0005	0x5D
0x0006	0x4C24
...	...

On-Chip Mem
0: 1000000011
1: 1100000010
2: 0001011101
3: 0100110000
...

(a) Compression scheme



(b) Architecture

Figure A.3: Ring oscillator emulator module

of data emerged: long strings of 0's and 1's with no variation, a few sporadic values greater than 1 but less than 2^8 which had no variation, and values greater than or equal to 2^{14} that typically had 5-6 bits of variation. Another interesting discovery was there were no values greater than $2^{15} - 1$ and, therefore, it wasn't necessary to store the most significant bit of the oscillator values in the lookup table as this was always zero.

We created the compression scheme shown in fig. A.3a to compress the ring oscillator dataset. Each table entry is 10 bits wide. The most significant bit determines if the table entry represents a string of values (0's or 1's) or a single frequency value. When a table entry represents a string, the next most significant bit specifies if the string is composed of 0's or 1's and the lower 8 bits specifies the length of the string. If the string length is greater than 255, then the string is divided into several table entries. When a table entry represents a single frequency value, the next most significant bit represents bit 14 of the frequency value. The lower eight bits represent either bits 13-6 or 7-0 of the frequency value, depending if the frequency is large (greater than or equal to 2^{14}) or small, respectively. This distinction is necessary because small frequency values have no variation whereas large frequency values always have some variation. Using this compression scheme, we reduced the size of the lookup table from

Beta Design	Flip-Flops	LUTs	BRAMs	Delay
Original	98 (5%)	152 (7%)	0 (0%)	3.409ns
w/ RO Emulator	128 (6%)	237 (12%)	4 (100%)	3.931ns

Table A.1: Ring oscillator emulator module overhead

30KB to less than 6KB.

A ring oscillator emulator module was created based on the compressed lookup table approach (see fig. A.3b). A state machine was implemented to latch the challenge vector, perform a sequential search of the compressed lookup table, and return the corresponding ring oscillator challenge response. To reproduce the slight variation present in the least six significant bits of values greater than or equal to 2^{14}, a counter incrementing every clock cycle was latched when the ring oscillator value was output. This appeared to be random variation to the user. Implementing this module in the Beta design allowed us to add any Trojan to the design without affecting the ring oscillator value displayed to the user.

Table A.1 shows the overhead introduced by the ring oscillator emulator module. The impact on area and delay was minimal. The remaining flip-flops and LUTs allowed for the implementation of elaborate Trojans several times the size of the original circuit. Since the original Beta design didn't utilize the on-chip block RAMs, we were able to use all 4 BRAMs to store the compressed ring oscillator frequency dataset.

Trojan Design

We designed a diverse set of Trojans based on the taxonomy shown in fig. 2.1 (derived from [35]). Each Trojan implemented a circumvention technique to mitigate the Beta hardening method as well as an effect and activation mechanism. While our Trojans specifically targeted the Beta design, we remained mindful of their real-world practicality.

Effects

We concentrated on two aspects of functional modification: the frequency of modification and the type of modification. These aspects are highly dependent on the target. A Trojan

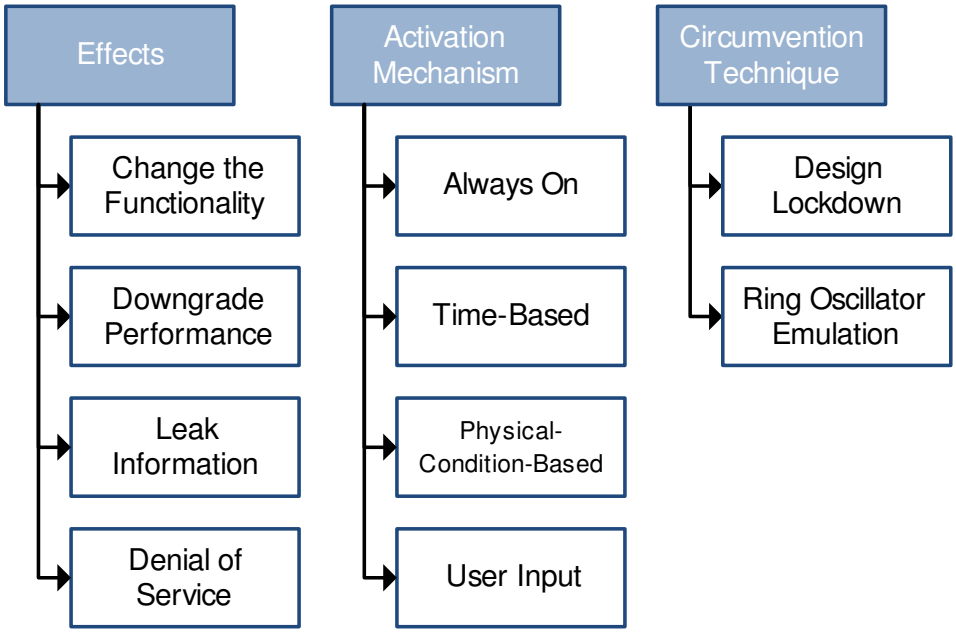


Figure A.4: Our Trojan Taxonomy (derived from [35])

that constantly modifies functionality will have a large initial impact but is more likely to be detected. A Trojan that produces malfunctions very sporadically will have less impact in the short term, but may have a larger impact in the long term since its presence could go unnoticed. Similarly, the type of malfunction should be large enough to have an impact but small enough to minimize detection.

Keeping these two aspects in mind, we implemented Trojans with both types of malfunction frequency while trying to keep the type of malfunction moderate. One Trojan replaced an AND gate with a XOR gate in the adder to create a sporadic malfunction dependent on the input value. Another persistently inverted a single bit of the output.

Since the Beta design didn't contain "secret" information, we focused on different methods of leaking information from the FPGA. The BASYS 2 board contained minimal hardware and peripherals. This offered both an advantage and a disadvantage in terms of leaking information. The disadvantage was that there were simply less channels to transmit information. The advantage was that all output peripherals (except the USB) were directly controlled by the FPGA, creating a larger Trojan design space to leak information. For instance, the FPGA

was connected directly to the VGA port. This allowed for the implementation of Trojans that exploited the VGA protocol, such as leaking information on the data lines during the horizontal and vertical syncs as described in [15].

One of our Trojans used the external oscillator IC socket on the BASYS 2 board as an antenna to leak information via an RF signal in the AM radio frequency range. The transmission could be heard as a beeping pattern on a standard AM radio. The radio had to be very close (a few inches) to the board as the antenna was not optimal, but extending the length of the antenna by placing a wire in the IC socket extended the reception distance to several feet.

Similar to functional modification, our performance degradation design methodology concentrated on the frequency and level of reduced performance. One way in which we degraded the performance of Beta was by implementing a Trojan that increased the timing delay of the addition operation. The Trojan held the inputs for a pre-set number of cycles before sending the data to the adder. Trojans on the opposite end of the spectrum produced a constant denial of service once activated.

Activation Mechanism

Activation mechanisms were chosen to correspond to the Trojan effect. For instance, the "Always On" activation mechanism would be inappropriate to pair with a "Denial of Service" effect as this Trojan would certainly be detected during chip verification given a real-world scenario. However, this activation mechanism may be appropriate for a Trojan that covertly leaks information. For many Trojans, we used a combination of triggers to complicate activation.

An example of an activation mechanism we implemented was one that used a combination of a specific user input with a physical-condition-based internal trigger. This trigger was designed to work with the ring oscillator emulator module circumvention technique. In this case, the ring oscillators embedded in the Beta design were no longer utilized and could be used to create a crude temperature sensor by exploiting the relationship between temperature and the frequency of ring oscillation [74]. Since an unexpected rise in temperature could activate the Trojan before it could affect its intended target, the activation mechanism was combined with

a specific user trigger. As temperature increased, the ring oscillator interconnection resistance increased, reducing the frequency of oscillations. The room temperature ring oscillator value was set as the basis for comparison. When the particular input was entered, the ring oscillator value was compared with the base value. If the difference was greater than a set threshold, then a Trojan was activated to produce a slight error in the result.

Results

Our ring oscillator emulation approach was successful in circumventing the Beta hardening mechanism and allowing the implementation of any Trojan the hardware resources would support. This was achieved by exploiting the relatively small challenge-response set associated with Beta's ring oscillator-based Trojan detection method. We acknowledge this approach was specific to the Beta hardening scheme and is limited by the number of challenge-response vectors in general. We also acknowledge this scheme was based on the assumption that the design would be verified at room temperature using standard operating voltage; the same environment in which we measured the ring oscillator frequency values. Our design was evaluated at the CSAW event in October 2010 and resulted in a 2^{nd} place finish in the embedded systems challenge.

BIBLIOGRAPHY

[1] Intellectual property (IP) challenges and concerns of the semiconductor equipment and materials industry. White paper, SEMI, 2008.

[2] Trust-hub. `http://trust-hub.org`, 2010. Accessed on 04/2011.

[3] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic. Detecting Trojans through leakage current analysis using multiple supply pad I_{DDQ}s. *IEEE Transactions on Information Forensics and Security*, 5(4):893–904, Dec. 2010.

[4] Sally Adee. The hunt for the kill switch. *IEEE Spectrum*, 45, May, 2008.

[5] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using IC fingerprinting. In *IEEE Symposium on Security and Privacy (SP)*, pages 296–310, May 2007.

[6] Y. Alkabani and F. Koushanfar. Consistency-based characterization for IC Trojan detection. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 123–127, Nov. 2009.

[7] Global Semiconductor Alliance. Semiconductor & fabless facts. `http://www.gsaglobal.org/resources/industrydata/facts.asp`, 2011. Accessed on 07/2011.

[8] Altera. Altera breaks semiconductor industry record for most transistors on an integrated circuit. `http://www.altera.com/corporate/news_room/releases/2011/products/nr-sv_milestone.html`, 2011. Accessed on 06/2011.

[9] Semiconductor Industry Association. International technology roadmap for semiconductors 2009 edition. `http://www.itrs.net/Links/2009ITRS/Home2009.htm`. Accessed on 06/2011.

[10] M. Banga, M. Chandrasekar, L. Fang, and M.S. Hsiao. Guided test generation for isolation and detection of embedded Trojans in ICs. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pages 363–366, 2008.

[11] M. Banga and M.S. Hsiao. A region based approach for the identification of hardware Trojans. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 40–47, June 2008.

[12] M. Banga and M.S. Hsiao. A novel sustained vector technique for the detection of hardware Trojans. In *International Conference on VLSI Design*, pages 327–332, 2009.

[13] M. Banga and M.S. Hsiao. VITAMIN: Voltage inversion technique to ascertain malicious insertions in ICs. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 104–107, July 2009.

[14] M. Banga and M.S. Hsiao. Trusted RTL: Trojan detection methodology in pre-silicon designs. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 56–59, June 2010.

[15] A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreno. A case study in hardware Trojan design and implementation. *International Journal of Information Security*, 10:1–14, 2011.

[16] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Design & Test of Computer*, 27(1):66–75, 2010.

[17] G. Bloom, R. Simha, and B. Narahari. OS support for detecting Trojan circuit attacks. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 100–103, July 2009.

[18] Defense Science Board. Task force on high performance microchip supply. http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA435563&Location=U2&doc=GetTRDoc.pdf, 2005. Accessed on 08/2011.

[19] Clair Brown and Greg Linden. *Chips and Change: How Crisis Reshapes the Semiconductor Industry*. The MIT Press, 2009.

[20] Rajat Subhra Chakraborty, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. MERO: A statistical approach for hardware Trojan detection. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 396–410, 2009.

[21] R.S. Chakraborty, S. Narasimhan, and S. Bhunia. Hardware Trojan: Threats and emerging solutions. In *IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 166–171, Nov. 2009.

[22] John Clark, Sylvain Leblanc, and Scott Knight. Hardware Trojan horse device based on unintended USB channels. *International Conference on Network and System Security (NSS)*, 0:1–8, 2009.

[23] Trusted Computing Group. Trusted platform module. http://www.trustedcomputinggroup.org/developers/trusted_platform_module/, 2011. Accessed on 07/2011.

[24] Jorge Guajardo, Sandeep Kumar, Geert-Jan Schrijen, and Pim Tuyls. FPGA intrinsic PUFs and their use for IP protection. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 63–80. 2007.

[25] Spencer S. Hsu. U.S. charges florida pair with selling counterfeit computer chips from china to the U.S. navy and military. http://www.washingtonpost.com/wp-dyn/content/article/2010/09/14/AR2010091406468.html, 2010. Accessed on 07/2011.

[26] Ted Huffmire, Brett Brotherton, Nick Callegari, Jonathan Valamehr, Jeff White, Ryan Kastner, and Tim Sherwood. Designing secure systems on reconfigurable hardware. *ACM Transactions on Design Automation of Electronic Systems*, 13:44:1–44:24, July 2008.

[27] Jeorge S. Hurtarte, Evert A. Wolsheimer, and Lisa M. Tafoya. *Understanding Fabless IC Technology*. Newnes, Newton, MA, USA, 2007.

[28] Intel. Moore's law timeline. `http://download.intel.com/pressroom/kits/events/moores_law_40th/MLTimeline.pdf`. Accessed on 06/2011.

[29] C.E. Irvine and K. Levitt. Trusted hardware: Can it be trustworthy? In *Design Automation Conference (DAC)*, pages 1–4, 2007.

[30] Len Jelinek. Consolidation thins the ranks of leading-edge semiconductor foundries. `http://www.isuppli.com/Semiconductor-Value-Chain/MarketWatch/Pages/Consolidation-Thins-the-Ranks-of-Leading-Edge-Semiconductor-Foundries.aspx`, 2011. Accessed on 06/2011.

[31] S. Jha and S.K. Jha. Randomization based probabilistic approach to detect Trojan circuits. In *IEEE High Assurance Systems Engineering Symposium (HASE)*, pages 117–124, 2008.

[32] Y. Jin, N. Kupp, and Y. Makris. Experiences in hardware Trojan design and implementation. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 50–57, 2009.

[33] Y. Jin and Y. Makris. Hardware Trojan detection using path delay fingerprint. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 51–57, 2008.

[34] Y. Jin and Y. Makris. Hardware Trojans in wireless cryptographic ICs. *IEEE Design & Test of Computers*, 27(1):26–35, 2010.

[35] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor. Trustworthy hardware: Identifying and classifying hardware Trojans. *Computer*, 43(10):39 –46, Oct. 2010.

[36] Lok-Won Kim, J.D. Villasenor, and C.K. Koc. A Trojan-resistant system-on-chip bus architecture. In *IEEE Military Communications Conference (MILCOM)*, pages 1–6, Oct. 2009.

[37] Samuel T. King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. Designing and implementing malicious hardware. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, pages 5:1–5:8, 2008.

[38] I. Kuon and J. Rose. *Quantifying and Exploring the Gap Between FPGAs and ASICs*. Springer, 2009.

[39] Mark LaPedus. Samsung lags in foudry rankings. `http://www.eetimes.com/electronics-news/4212366/Samsung-lags-in-foundry-rankings-`, 2011. Accessed on 07/2011.

[40] Jie Li and J. Lach. At-speed delay characterization for IC authentication and Trojan horse detection. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 8–14, 2008.

[41] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 168–177, 2000.

[42] Lang Lin, Markus Kasper, Tim Güneysu, Christof Paar, and Wayne Burleson. Trojan side-channels: Lightweight hardware Trojans through side-channel engineering. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 382–395, 2009.

[43] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. A large scale characterization of RO-PUF. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 94–99, June 2010.

[44] C. Maxfield. Achronix's next-gen FPGAs in Intel's 22nm process. http://www.eetimes.com/electronics-products/electronic-product-reviews/fpga-pld-products/4210286/Achronix-s-next-gen-FPGAs-in-Intel-s-22nm-process, 2010. Accessed on 08/2011.

[45] Dylan McGrath. Intel to fab FPGAs for startup Achronix. http://www.eetimes.com/electronics-news/4210263/Intel-to-fab-FPGAs-for-startup-Achronix?pageNumber=1, 2010. Accessed on 08/2011.

[46] D. Mclntyre, F. Wolff, C. Papachristou, S. Bhunia, and D. Weyer. Dynamic evaluation of hardware trust. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 108–111, July 2009.

[47] MIT. Timeline: March of the machines. http://www.technologyreview.com/computing/25151/, 2010. Accessed on 06/2011.

[48] Ron Molnar. IC assembly and test market poised for growth in 2010. *Chip Scale Review*, March-April 2010.

[49] S. Narasimhan, Dongdong Du, R.S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia. Multiple-parameter side-channel analysis: A non-invasive hardware Trojan detection approach. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 13–18, June 2010.

[50] S.R. Nassif. Process variability at the 65nm node and beyond. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8, Sept. 2008.

[51] Federal Business Opportunities. A TRUST for integrated circuits. https://www.fbo.gov/index?s=opportunity&mode=form&id=db4ea611cad3764814b6937fcab2180a&tab=core&_cview=1, 2006. Accessed on 07/2011.

[52] Federal Business Opportunities. Iarpa trusted integrated chips (tic) program. https://www.fbo.gov/index?s=opportunity&mode=form&tab=core&id=2822fed19cdcb74f464dae9718d5f3d8, 2011. Accessed on 08/2011.

[53] Federal Business Opportunities. Integrity and reliability of integrated circuits (IRIS). https://www.fbo.gov/index?s=opportunity&mode=form&id=0890f344196e5467640e56db7a33f597&tab=core&_cview=1, 2011. Accessed on 06/2011.

[54] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey. Hardware Trojan horse detection using gate-level characterization. In *Design Automation Conference (DAC)*, pages 688–693, July 2009.

[55] R. Rad, J. Plusquellic, and M. Tehranipoor. A sensitivity analysis of power signal methods for detecting hardware Trojans under real process and environmental conditions. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(12):1735–1744, Dec. 2010.

[56] R.M. Rad, Xiaoxiao Wang, M. Tehranipoor, and J. Plusquellic. Power supply signal calibration techniques for improving detection resolution to hardware Trojans. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 632–639, 2008.

[57] D. Rai and J. Lach. Performance of delay-based Trojan detection techniques under parameter variations. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 58–65, July 2009.

[58] J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri. Design and analysis of ring oscillator based design-for-trust technique. In *IEEE VLSI Test Symposium (VTS)*, pages 105–110, May 2011.

[59] S.K. Saha. Modeling process variability in scaled CMOS technology. *IEEE Design & Test of Computers*, 27(2):8–16, March-April 2010.

[60] H. Salmani, M. Tehranipoor, and J. Plusquellic. New design strategy for improving hardware Trojan detection and reducing Trojan activation time. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 66–73, 2009.

[61] H. Salmani, M. Tehranipoor, and J. Plusquellic. A layout-aware approach for improving localized switching to detect hardware Trojans in integrated circuits. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, Dec. 2010.

[62] Joseph Lieberman (U.S. Senator). Whitepaper on national security aspects of the global migration of the U.S. semiconductor industry. White paper, U.S. Senate, 2003.

[63] Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, and W. Clay. Process reliability based Trojans through NBTI and HCI effects. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 215–222, June 2010.

[64] Defined Business Solutions. Trusted foundry. `http://www.trustedfoundryprogram.org/`, 2011. Accessed on 07/2011.

[65] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the Design Automation Conference (DAC)*, pages 9–14, 2007.

[66] Gookwan Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the International Conference on Supercomputing (ICS)*, pages 160–171, 2003.

[67] M. Tehranipoor and F. Koushanfar. A survey of hardware Trojan taxonomy and detection. *IEEE Design & Test of Computers*, 27(1):10–25, 2010.

[68] M. Tehranipoor, H. Salmani, X. Zhang, X. Wang, R. Karri, J. Rajendran, and K. Rosenfeld. Trustworthy hardware: Trojan detection and design-for-trust challenges. *Computer*, 44:66–74, 2011.

[69] Xiaoxiao Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic. Hardware Trojan detection and isolation using current integration and localized current analysis. In *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS)*, pages 87–95, 2008.

[70] Xiaoxiao Wang, M. Tehranipoor, and J. Plusquellic. Detecting malicious inclusions in secure hardware: Challenges and solutions. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 15–19, 2008.

[71] F. Wolff, C. Papachristou, S. Bhunia, and R.S. Chakraborty. Towards Trojan-free trusted ICs: Problem analysis and detection scheme. In *Design, Automation and Test in Europe (DATE)*, pages 1362–1365, 2008.

[72] Xilinx. Core generator guide. `http://www.xilinx.com/itp/xilinx6/books/docs/cgn/cgn.pdf`. Accessed on 07/2011.

[73] X. Zhang and M. Tehranipoor. Case study: Detecting hardware Trojans in third-party digital IP cores. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 67–70, June 2011.

[74] Kenneth M. Zick and John P. Hayes. On-line sensing for healthier FPGA systems. In *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 239–248, 2010.