# TPAD: Hardware Trojan Prevention and Detection for Trusted Integrated Circuits

Tony F. Wu, Karthik Ganesan, *Student Member, IEEE*, Yunqing Alexander Hu, *Student Member, IEEE*, H.-S. Philip Wong, *Fellow, IEEE*, Simon Wong, *Fellow, IEEE*, and Subhasish Mitra, *Fellow, IEEE*

***Abstract*—There are increasing concerns about possible malicious modifications of integrated circuits (ICs) used in critical applications. Such attacks are often referred to as hardware Trojans. While many techniques focus on hardware Trojan detection during IC testing, it is still possible for attacks to go undetected. Using a combination of new design techniques and new memory technologies, we present a new approach that detects a wide variety of hardware Trojans during IC testing and also during system operation in the field. Our approach can also prevent a wide variety of attacks during synthesis, place-and-route, and fabrication of ICs. It can be applied to any digital system, and can be tuned for both traditional and split-manufacturing methods. We demonstrate its applicability for both application-specified integrated circuits and field-programmable gate arrays. Using fabricated test chips with Trojan emulation capabilities and also using simulations, we demonstrate: 1) the area and power costs of our approach can range between 7.4%–165% and 7%–60%, respectively, depending on the design and the attacks targeted; 2) the speed impact can be minimal (close to 0%); 3) our approach can detect 99.998% of Trojans (emulated using test chips) that do not require detailed knowledge of the design being attacked; 4) our approach can prevent 99.98% of specific attacks (simulated) that utilize detailed knowledge of the design being attacked (e.g., through reverse engineering); and 5) our approach never produces any false positives, i.e., it does not report attacks when the IC operates correctly.**

***Index Terms*—3-D integration, concurrent error detection, hardware security, hardware Trojan, randomized codes, reliable computing, resistive RAM (RRAM), split manufacturing.**

## I. Introduction

THERE is growing concern about the trustworthiness of integrated circuits (ICs). If an untrusted party fabricates an IC, there is potential for an adversary to insert a malicious hardware Trojan [1], an unauthorized modification of the IC resulting in incorrect functionality, and/or sensitive data being exposed [2]. These include (but are not limited to) [3]:

1) *Modification of Functional Behavior Through Logic Changes:* For example, extra logic gates may be inserted to force an incorrect logic value on a wire at some (arbitrary) point in time [Fig. 1(a)] [2].

Manuscript received January 10, 2015; revised May 9, 2015; accepted July 21, 2015. Date of publication August 28, 2015; date of current version March 17, 2016. This work was supported by the IARPA Trusted Integrated Chips Program. This paper was recommended by Associate Editor S. Bhunia.

T. F. Wu, K. Ganesan, Y. A. Hu, H.-S. P. Wong, and S. Wong are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: tonyfwu@stanford.edu).

S. Mitra is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA, and also with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2474373

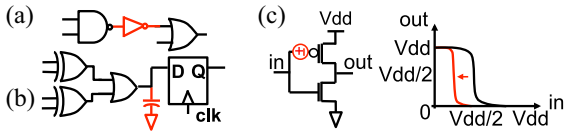


Fig. 1. Examples of hardware Trojans. (a) Modification of functional behavior through logic changes. (b) Electrical changes such as changing timing characteristics. (c) Reliability changes such as accelerating aging with NBTI.

2) *Electrical Modification:* For example, extra capacitive loading may be placed on a circuit path to alter timing characteristics of the IC [Fig. 1(b)] [2].
3) *Reliability Degradation:* For example, aging of transistors [e.g., negative bias temperature instability (NBTI)] may be accelerated, degrading reliability [Fig. 1(c)] [4].

A hardware Trojan is detected when we report malicious modifications to an IC. A hardware Trojan is prevented when we stop it from being inserted. Part of the challenge in detecting or preventing hardware Trojans arises from the fact that the methods for inserting Trojan attacks are numerous (e.g., [2], [5] for a comprehensive list).

Trojan insertion methods include (but are not limited to):

1) Modification of register transfer level (RTL) description or layout by malicious computer-aided design (CAD) tools without the designer's knowledge [6].
2) IC modification anytime between tapeout and fabrication, since it is possible to reverse engineer (complete or partial) netlists from layouts [7], [8]. Such modifications may be applied to all or a sample of fabricated ICs. IC modification can span a wide range from modification of circuits to doping profiles of a few transistors [9].
3) Leakage of information, e.g., through fluctuations in IC power consumption [10].

Existing techniques for hardware Trojan detection exhibit one or more of the following limitations (see Section VII).

1) Some techniques rely on IC testing for Trojan detection, which alone limits the scope of Trojans. For example, Trojans activated by a "time bomb" [2], [11] or accelerated circuit aging [4] may not be detected.
2) Nondestructive visual inspections can be circumvented by carefully hiding Trojans [9]. For example, an adversary can change the doping profile of transistors. Without IC delayering, detection becomes difficult due to limited imaging depth, especially for 3-D ICs [46].
3) Destructive IC testing is effective only when the percentage of ICs attacked is high (e.g., many chips on a wafer must be attacked), as shown in Section VII-A.
4) IC fingerprinting, such as circuit path delays, leakage power, heat, or switching activity, relies on statistical models to distinguish between malicious activities versus variability during manufacturing or

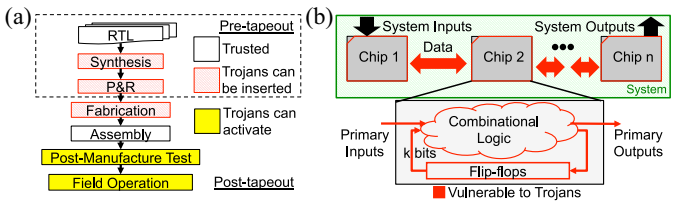


Fig. 2. (a) Assumptions on when Trojans can be inserted or activated. (b) Assumptions on vulnerable system and chip components.

normal operation. This makes sufficiently small Trojans undetectable [12]. Moreover, such techniques are prone to reporting false positives [80].

5) Error-detecting codes [14] can be compromised if the code construction is known.

This paper presents a Trojan Prevention and Detection (TPAD) architecture for digital systems. TPAD enables test time and runtime detection of Trojans inserted during fabrication. TPAD also prevents Trojan insertion during logic synthesis and layout design.

In this paper, we assume that logic synthesis tools, physical design tools, as well as IC fabrication are untrusted, meaning CAD tools, and/or foundries can insert Trojans [Fig. 2(a)]. However, we do require the RTL or system specification as well as system assembly to be trusted, meaning that an adversary cannot insert a Trojan during these stages.

The following are some of the important TPAD features.

1) TPAD detects hardware Trojans that produce incorrect logic value(s) that propagate to sequential elements (flip-flops and latches), on-chip memories, or I/O pins.
2) TPAD has a 0% false positive rate, i.e., it does not report Trojan detection when the IC operates correctly.
3) TPAD can be used for Trojan detection during post-manufacture IC testing and also during field operation (concurrently during system operation or through periodic testing). As a result, Trojans triggered by rare events such as time bomb attacks can be detected. If used in conjunction with system-level recovery techniques (such as those used for fault-tolerant computing), it may be possible to recover from many attacks [89, Appendix B]. Since TPAD improves the observability of signals inside an IC, test time Trojan detection is expected to be enhanced compared to traditional functional testing.
4) The hardware overhead associated with TPAD is flexible, and can be adjusted based on targeted attacks and implemented functionality.
5) TPAD can be used in conjunction with (or without) split manufacturing [15]. TPAD can be combined with emerging nonvolatile memories (e.g., resistive RAM or RRAM) and their monolithic 3-D integration [16] to reduce hardware overheads.
6) TPAD can be combined with a secure CAD tool flow (Section VI) to prevent insertion of Trojans during logic synthesis and physical design.
7) Currently, TPAD does not target Trojans that leak information without modifying logic values transmitted through the I/Os of the attacked IC (e.g., using a radio or through fluctuations in power [10]).

Our overall approach was briefly outlined in [85]. In this paper, we present an in-depth description of TPAD, experimental results from test chips, as well as various implementation tradeoffs with respect to area, power, and delay. The TPAD architecture is introduced in Sections II and III. Section IV presents an implementation of the TPAD architecture that can be used for any arbitrary digital system. For general processor designs such as OpenRISC [35], the key results of TPAD are:

1) Area overheads range from 114% to 165% with 34% to 61% power overheads.
2) Near-0% speed impact can be achieved (through additional pipelining in the TPAD architecture).
3) The corresponding detection rate for uninformed attacks (attacks that do not require detailed knowledge of the design before inserting a Trojan) ranges from 87.5% to 99.6%.
4) TPAD prevents sophisticated attacks requiring some detailed knowledge of the implemented design (launched by CAD tools or foundries) with a rate ranging from 96.2% to 99.98% (Section III).

In [89], we also demonstrate TPAD for fabricated FPGA test chips. Section V presents TPAD implementations for specialized (accelerator) designs, such as a Lempel–Ziv (LZ77) data compressor test chip [38] and a fast Fourier transform (FFT) engine [37]. For such accelerators, the key results are as follows.

1) TPAD area overheads can be 7.4%.
2) TPAD detects 99.998% of uninformed attacks.

We further show that for LZ77 data compression, performance metrics such as compression ratio (CR) can be traded off for area overhead with no impact on TPAD.

In Section VI, we discuss how TPAD can be used to prevent logic synthesis and physical design tools from inserting Trojans. Section VII compares existing approaches with TPAD. Section VIII concludes this paper.

Reference [89, Appendix A] presents a detailed discussion of various attack types. Reference [89, Appendix B] discusses possible approaches to recover from Trojans.

## II. System Architecture Overview

TPAD is derived from the concept of concurrent error detection (CED) [17] for fault-tolerant computing. The "classical" CED approach is illustrated in Fig. 3(a): given a function, an output characteristic predictor (OCP) predicts an output characteristic (e.g., parity of output bits) for each input to that function. Another function (checker) calculates the actual output characteristic and checks it against the predicted output characteristic. The checker reports an error whenever the two do not match. However, the problem of detecting attacks by hardware Trojans is different than CED for fault-tolerant computing for the following reasons.

1) CED for fault-tolerant computing generally targets single faults (that may occur at random depending on the fault model). For example, CED techniques that predict output parity (or the count of 0s or 1s in an output word) can be compromised if an attacker inserts a Trojan which flips output bits of the function such that the number of 1s in a given output word is preserved.
2) An attacker must not be able to derive the OCP; otherwise, the attacker could modify both the function outputs and the OCP outputs such that the checker does not detect errors.
3) CED techniques for fault-tolerant computing generally assume that only one of the units (function, OCP, and checkers) is faulty (at any given time). For hardware Trojans, one cannot make such assumptions.

One option is to implement a chip entirely on a reconfigurable platform such as an FPGA (with CED for the mapped

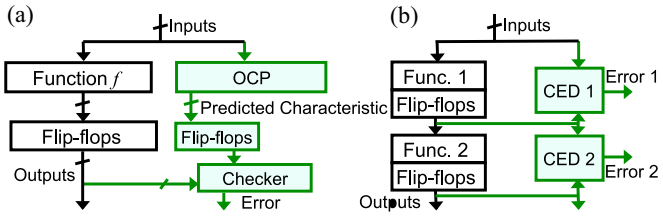


Fig. 3. (a) General block diagram of a CED scheme. (b) Separate CED techniques for separate functions of a chip.

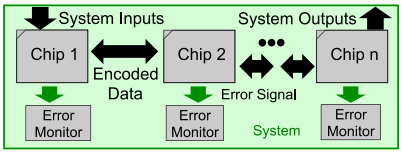


Fig. 4. Trusted system. Each chip is implemented with TPAD. Data communication between chips is encoded. Error monitors check encoded error signals and determine if an attack has occurred.

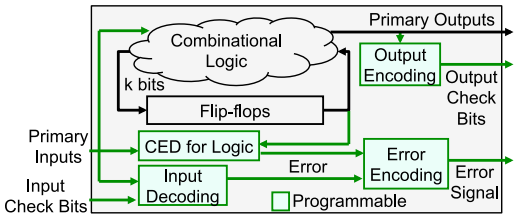


Fig. 5. TPAD architecture for each chip, including output encoding, input decoding, CED for logic, and error encoding.

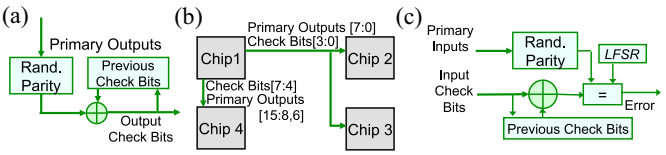


Fig. 6. (a) Output encoding. (b) Same encodings for the same set of primary outputs; separate encodings for different sets of outputs. (c) Input decoding.

logic). This can potentially mitigate vulnerabilities to sophisticated attacks that rely on reverse engineering. However, CMOS FPGAs can incur significant area and performance overheads compared to application-specific integrated circuits (17–27× area and 18–26× performance in [18]). While monolithic 3-D integration has been shown to somewhat reduce these costs (5–8× area overheads and 10–15× performance overheads in [19]), these overheads may still be too high for many applications. Furthermore, an increasing number of accelerators (e.g., signal processing blocks) [20] are being embedded in FPGA chips. These accelerators, along with I/Os, are vulnerable to Trojans. Thus, FPGAs that are manufactured by untrusted foundries can still be vulnerable.

TPAD overcomes the limitations of classical CED techniques. At the same time, it avoids the high overheads of FPGAs through selective hardware programmability (Section III) in Trojan checking circuitry.

Fig. 2(b) shows a digital system with circuit components vulnerable to Trojan attacks highlighted in red. The system operates in a trusted environment; however, all chips are vulnerable to Trojans. While wires (or channels) between chips may not vulnerable to attacks (since the system may be assembled in a trusted environment), any chip with a Trojan may use them to send incorrect data.

A block diagram of the system architecture with TPAD is shown in Fig. 4. As mentioned in Section I, we assume that the system is assembled in a trusted environment. Thus, any Trojan attack within the system will originate from at least one chip. Each chip in the system encodes its outputs and receives encoded inputs. Specifically, chip 1 outputs data and corresponding check bits, so chip 2 can use them to verify the data (Sections II-A and II-B). Encoded error signals sent from each chip convey the state of all checkers within the chip (Section II-E). The error monitors (Section II-F) then interpret these signals, and determine if an attack has occurred.

Each chip implemented using TPAD includes four modules (Fig. 5): 1) output encoding; 2) input encoding; 3) CED-based Trojan detection; and 4) error encoding (Sections II-A and II-E).

### A. Output Encoding

TPAD encodes primary outputs [Fig. 6(a)] using randomized parity encoding (Section IV-A). A separate encoding is used for each subsequent chip in the system that receives a different set of the primary outputs; the same encoding is used for chips receiving the same set of primary outputs [Fig. 6(b)]. The primary outputs and their check bits can be transmitted to destination chips serially (same or different pins) or in parallel.

A randomized parity code word (explained in Section IV-A) is calculated for the primary outputs during each clock cycle. The check bits of this code word are then XORed with the previous output check bits (stored in flip-flops) to form the output check bits (e.g., $110101_2 \oplus$ previous output check bits $010101_2$ produces output check bits $100000_2$). The output check bits are then stored and used in the next clock cycle. Thus, the output encoding at a particular time is a function of the history of the primary outputs in the preceding clock cycles (the starting check bits are initialized at chip startup to be uniformly random).

In order for a chip to check its primary inputs, it must use the same randomized parity encoding scheme as the sender's output encoding. To ensure this property, first-in, first-outs (FIFOs) or handshaking protocols may be required.

An attacker can attempt to derive the randomized parity scheme by adding hardware to the chip that stores randomized parity outputs and solves linear equations. However, even an attack that requires fewer additional gates can usually be detected using nondestructive post-manufacture inspections; thus, a complex attack such as this is expected to be detected [89, Appendix A].

### B. Input Decoding

The primary inputs [Fig. 6(c)] of each chip are encoded according to the method described in Section II-A. The decoding process checks for attacks at the outputs of the sender as well as attacks at its own inputs. Suppose that, primary inputs (e.g., $FA_{16}$) and input check bits (e.g., $1_{16}$) are received. The input check bits are then XORed with the previous cycle's input check bits (e.g., $B_{16}$) to calculate the expected randomized parity bits (e.g., $1_{16} \oplus B_{16} = A_{16}$). The actual randomized parity bits are calculated from the primary inputs (e.g., $A_{16}$). Since both sets of randomized parity bits (expected and actual) are equal, no attack has occurred in this case. However, when a pin attack [89, Appendix A] occurs, the expected parity bits will not match the actual bits; thus, the pin attack will be detected.

### C. Logic CED

On-chip logic is protected using CED (Fig. 5) as introduced at the beginning of Section II. When the combinational logic is separated into independent blocks, different CED schemes

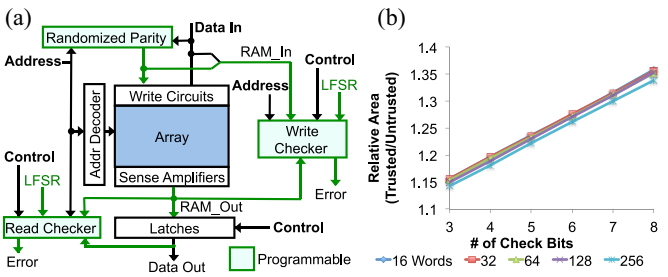


Fig. 7. Memory architecture. (a) Block diagram of data encoding before writes and checking after reads/writes. Blocks with programmability indicated in green (Section III). (b) Relative area cost of implementing trusted RAM compared to a nontrusted RAM versus number of check bits.

may be used for each of the blocks as shown in Fig. 3(b). A general CED technique for any digital system is discussed in Section IV, while application-specific techniques are discussed in Section V. These techniques overcome the limitations of classical CED in detecting Trojan attacks.

### D. Memory CED

Trojans inserted in an on-chip RAM (e.g., read/write logic, address decoder logic, and memory cells) can alter the data, the location in which the data is stored, or the operation of the RAM (read versus write). To detect such attacks, the RAM is protected using a randomized parity code (Section IV-A). In TPAD, during a write operation, both the address (e.g., $BE_{16}$) and data bits (e.g., $124_{16}$) are used to calculate check bits (e.g., $6_{16}$) [Fig. 7(a)] to ensure that correct data are written to the correct location. These check bits are stored along with the data. During read operation, the address (e.g., $BE_{16}$) and data (e.g., $124_{16}$) are used to calculate the expected check bits (e.g., $6_{16}$). These are compared with the check bits read out from the memory (e.g., $6_{16}$) and an attack is detected if they do not match. For example, if the same data and check bits (as the above example) were retrieved during a read operation for a different address (e.g., $BF_{16}$), the expected check bits would not match the retrieved check bits (e.g., $2_{16} \neq 6_{16}$). To hide the randomized parity code construction from adversaries, both the encoder and checkers are protected with switchbox (SB) programmability (Section III).

For detecting attacks related to a write operation, the RAM must operate in write-through mode, a feature in many RAMs [23]. This means that during a write operation, the data that are being written are sensed and appear at the output of the RAM. Thus, immediately following a write operation, the input and output of the RAM block can be checked for attacks. Latches are used to ensure data out only changes during a read operation [Fig. 7(a)]. Reference [93, Sec. II-D, Table I] shows the specific checking step that detects a Trojan for each operation (read, write, or idle). Fig. 7(b) shows the relative area of this Trojan detection scheme for various numbers of check bits as well as different RAM sizes. A word size of 16 bits (data in and data out) was used for each case. The GlobalFoundries 65 nm SRAM compiler was used to obtain RAM areas. Checking circuits were synthesized and placed-and-routed using the same technology.

### E. Error Signal Encoding

Error signals from various CED checkers are inherently vulnerable. For example, if a single bit is used to indicate that a Trojan attack has been detected, an adversary can simply attack that bit to indicate no attack.

TABLE I
REVERSE ENGINEERING SUCCESS PROBABILITIES

| Type of Attack | Configurations/Output bit | | |
|---|---|---|---|
| | $2^{64}$ | $2^{72}$ | $2^{80}$ |
| Wire-length Tracing* | $5.4\times10^{-20}$ | $2.1\times10^{-22}$ | $8.3\times10^{-25}$ |
| DS & FO Matching* | $5.5\times10^{-20}$ | $2.1\times10^{-22}$ | $8.4\times10^{-25}$ |
| Subcircuit Matching* | 0.0 | 0.0 | 0.0 |
| CP: $\theta = 0.1$ | $1.18\times10^{-3}$ | $5.08\times10^{-4}$ | $2.18\times10^{-4}$ |
| CP: $\theta = 0.05$ | 0.0375 | 0.0249 | 0.0165 |

* Simulated using OpenRISC w/ Randomized Parity codes (Section IV).

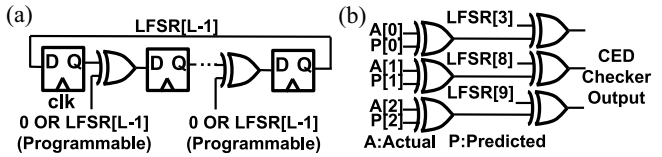


Fig. 8. (a) LFSR of length $L$ with programmable feedback polynomial. (b) Sample CED checker design with $r = 3$ bits.

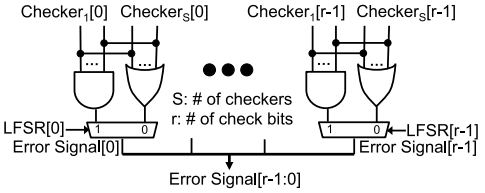


Fig. 9. Error signal encoding. For each of the $r$ bits, if LFSR value is 0, the OR of checker output bits is chosen. If it is 1, the AND is chosen.

Totally self-checking checkers [17] are also inadequate, because an adversary can insert a Trojan into the checker that can make the checker output appear to be valid.

A uniform random sequence might be considered to prevent an adversary from guessing the meaning of the error signal since it has maximum entropy [24]. However, a deterministic error signal is needed so that it can be interpreted by the error monitor (Section II-F); thus, a linear feedback shift register (LFSR) [25] is natural for this purpose. The polynomial and seed are made programmable to prevent an adversary from compromising the LFSR during design and fabrication. XOR gates are inserted in a shift register as shown in Fig. 8(a) to ensure that any primitive polynomial of degree $L$ can be realized. Programmability can be realized with RRAM SBs for low area cost (Section III).

Because of the large number of primitive polynomials for a sufficiently large degree (e.g., for $L = 64$ it is on the order of $10^{17}$ [86]), the probability that an adversary correctly guesses which polynomial will be used during runtime is negligible. If the adversary is able to observe outputs of the LFSR during runtime, the evolution of the error signal may be deduced [26]. However, even an attack that requires fewer additional gates can usually be detected using nondestructive postmanufacture inspections; thus, a complex attack such as this is expected to be detected [89, Appendix A].

A subset of $r$ bits in the LFSR of a given design (e.g., $35_{16}$), determined during design time, is used to encode the error signals for the CED checker. This subset can be chosen arbitrarily since the characteristic polynomial and seed of the LFSR are programmable. The checker must operate such that when no attack is detected, the error signal will be equal to the $r$ LFSR bits (e.g., $35_{16}$), as shown in Fig. 8(b). If the error signal takes any other value (e.g., $36_{16}$), an attack is detected. The subset of $r$ bits can vary for different designs.

All checkers within the same clock domain use the same LFSR. In any clock cycle when there is no attack

detected, all of these checkers output the exact same signal. Thus, signals can be combined to reduce the number of output pins needed for the chip. To do so, each corresponding bit (e.g., bit 1 of checker 1, bit 1 of checker 2, etc.) is combined as shown in Fig. 9. When the LFSR bit is 1, the AND of the checker outputs is chosen.

### *F. Error Monitors*

Error signals must be interpreted by a trusted party to ultimately decide whether an attack has occurred. Separate LFSRs are used within each error monitor to generate expected error signals. These LFSRs are configured the same (characteristic polynomial, seed, and clock) as the LFSRs that generate the corresponding error signals. The error signal received should match the expected error signal; otherwise, an attack has occurred. To realize this error monitor, an older technology node from a trusted foundry may be used.

## III. Switchbox Programmability

As discussed in Section II, the output encoding, input decoding, logic CED, and error encoding blocks in Fig. 5 must be protected against modifications that stop Trojans from being detected. TPAD builds on ideas from privacy techniques in social networks [27] in order to prevent circuit reverse engineering and sophisticated attacks. We represent logic gates in a netlist as vertices in a graph and wires as directed edges between vertices. The netlists corresponding to the output encoding, input decoding, logic CED, and error encoding blocks in Fig. 5 are modified to prevent adversaries from gaining detailed knowledge of their functionalities. The technique used has two important features.

1) Each modification to a netlist will require adding an extra logic element (Definition 1). The area and power cost of each such element is made small via the use of an emerging memory technology.
2) The total number of modifications (overall cost) to a given netlist is flexible, and can be tuned based on the level of security desired. We show (Sections III-A and V) that one can ensure exceptionally high security against a very wide variety of attacks while maintaining low total cost.

While the adversary can still bypass the checkers by storing a previous state of the system with additional hardware, these attacks can be easily detected using nondestructive post-manufacturing inspections [89, Appendix A]. TPAD uses random switching [27] to hide functionality of checkers.

*Definition 1:* Let $e_1 = (x, z)$ and $e_2 = (y, w)$ be two disjoint wires (i.e., $x \neq y \neq z \neq w$). We say $e_1$ and $e_2$ are switchable if they can be reconfigured as $\widetilde{e}_1 = (x, w)$ and $\widetilde{e}_2 = (y, z)$.

This can also be generalized to larger sets of wires. Let

$$S = \{(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)\}$$

be any arbitrary set of $k$ disjoint wires. We say the wires in $S$ are switchable if for any permutation $\pi : [k] \rightarrow [k]$, $S$ can be reconfigured as $\widetilde{S} = \{(a_1, b_{\pi(1)}), (a_2, b_{\pi(2)}), \ldots, (a_k, b_{\pi(k)})\}$. We refer to any circuitry that makes wires switchable as an SB. The total number of possible states of all the SBs in a circuit is referred to as the number of configurations of the circuit. An example of modifying a circuit using an SB is illustrated in Fig. 10. Different configurations of the SB yield different functions. Thus, when certain wires in a design are made switchable, attackers must know the intended SB configuration to deduce the intended functionality.

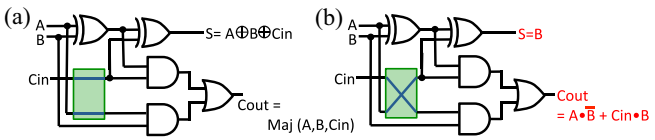


Fig. 10. Example of modifying a full adder with a two-input SB. There are two possible configurations. (a) Parallel mode. (b) Crossed mode.

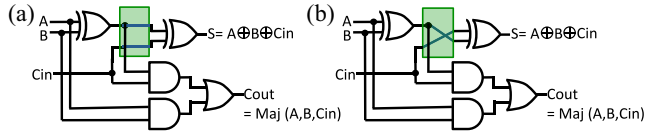


Fig. 11. Degenerate SB insertion in a full adder with a two-input SB Both configurations of the SB, (a) and (b), yield the same functionality.

However, simply inserting an SB into a design does not ensure that an incorrect configuration will lead to incorrect functionality. For example, a degenerate case of SB insertion is shown in Fig. 11. Regardless of the configuration of the SB, the circuit will behave as a full adder. For this reason, our technique for inserting SBs involves formal equivalence checking to verify that each SB that is inserted can lead to different functionality (Step IV of Algorithm 1).

However, degenerate cases can still occur when a larger number of SBs are inserted into a design. For instance, if $x$ two-input SBs are inserted into a design, there are $2^x$ total configurations of the SBs. If $x$ is small, it is feasible to run equivalence checking to verify that only one of the $2^x$ configurations yields the intended functionality of the design. However, the number of equivalence checks needed to verify this property grows exponentially in $x$, making the problem of guaranteeing no degenerate cases intractable for large $x$.

To simulate the percentage of degenerate cases for a large number of SBs, 64 SBs were inserted into the OCP of an OpenRISC [35] instruction fetch module. $10^6$ incorrect SB configurations were applied and equivalence checking was performed to compare the resulting functionality with the intended design. No degenerate cases were found. We note, however, that only a small fraction (0.05%) of the $2^{64}$ different configurations were explored in these simulations. Therefore, a larger experiment would need to be performed in order to verify that degenerate cases are highly unlikely for a given design. The percentage of degenerate cases also depends on symmetry properties of the function. SB insertion would need to account for such properties in order to provide guarantees. Further discussion is provided in [89].

In hardware, a low area-cost technique for switching sets of wires is enabled through the use of RRAM technology. Advances in RRAM technology have shown the feasibility of monolithically integrating RRAM with CMOS [28]. Its cell size has been shown to be smaller than other nonvolatile RAM technologies at $4F^2$ ($F$ is the feature size of the technology node) [29]. RRAM provides a nonvolatile configuration memory for SBs so chips do not have to be reconfigured when the power is cycled, making chip boot-up more secure since there is no need for potentially insecure programming sequences to be sent to the chip. Moreover, when the design rarely requires reprogramming, long write times are negligible.

Fig. 12 illustrates the design of an RRAM SB as described and fabricated in [28]. Each cell consists of two RRAM elements arranged as a resistive divider, a programming transistor, and a pass transistor. The two RRAM elements in each cell are always programmed to be in different states, thus pulling

**Algorithm 1** Random Switchbox (SB) Insertion

**Inputs:** Netlist $G = \{V, E\}$, Number of configurations per output bit: $2^t$.

**Outputs:** New netlist $\tilde{G} = \{\tilde{V}, \tilde{E}\}$ with $\geq 2^t$ configurations per output bit.

Initialize: $\tilde{G} = G$.

Step I: Choose $v \in V$ uniformly at random and group all the vertices within radius 1 of $v$, $N_1(v)$.

Step II: Search $\tilde{G}$ for $u \in V$ s.t. in-degree and number of outputs of $N_1(u)$ match $N_1(v)$, and $N_1(u) \cap N_1(v) = \emptyset$. If not found, retry Step I.

Step III: Let $N_1(v)$, $N_1(u)$ each have $a$ incoming wires $\{(x_1, z_1), \ldots, (x_a, z_a)\}$ and $\{(y_1, w_1), \ldots, (y_a, w_a)\}$, and $b$ outputs $\{s_1, \ldots, s_b\}$ and $\{t_1, \ldots, t_b\}$ respectively.
**For** $1 \leq i \leq a$:
Insert two-input SB in $\tilde{G}$ with inputs $(x_i, y_i)$, outputs $(z_i, w_i)$.
**For** $1 \leq j \leq b$:
Insert two-input SB in $\tilde{G}$ with inputs $(s_j, t_j)$, first output for all outgoing edges of $s_j$, second output for all outgoing edges of $t_j$.

Step IV: **For** each SB inserted in Step III:
Consider G with only this SB added.
Compare functionality with this SB crossed vs. this SB parallel.
If equivalent: Remove this SB.
Else: With probability 1/2, adjust input and output wires of SB so that crossed SB will yield same functionality as $G$.

Step V: Let $G_i$ denote the subgraph of $\tilde{G}$ corresponding to the logic cone which ends at the $i$-th output bit of $G$. If $\forall i$, there are at least $t$ SBs in $G_i$, output $\tilde{G}$. Otherwise repeat steps I-IV.

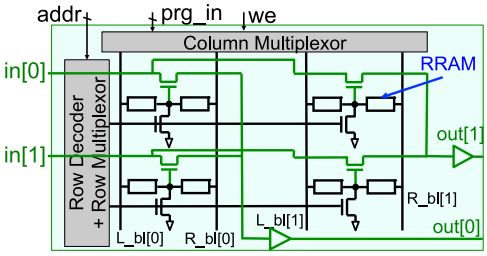


Fig. 12. Schematic of RRAM-based SB [28]. Address bus (addr), program data in (prg_in), and write enable (we) are used to program the SB to a particular configuration.

the pass transistor gate either high or low, allowing or denying a given input to propagate to an output. The address bus (addr), program data in (prg_in), and write enable (we) are only used when configuring the SB, making the programming interface much like an SRAM interface. The SBs can be implemented with as little as 0.04% area overhead compared to the rest of a chip (Section V-A) since the RRAM cells can be placed in the metal interconnects above the active layer of transistors on a chip as shown in [85]. Alternatively, standard technology such as flip-flops or lookup tables can be used to configure SBs, though nonvolatility and area would be sacrificed.

Algorithm 1 gives a method for randomly inserting SBs into any netlist, given the target number of configurations as an input. Fig. 13 illustrates the outcomes of the steps of Algorithm 1, when performed on the circuit as shown in Fig. 13(a), with four as the number of configurations per output bit. The effectiveness of this algorithm in preventing sophisticated attacks is evaluated in Section III-A.

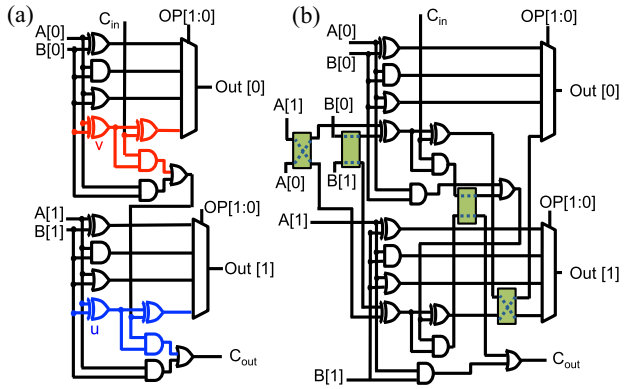


Fig. 13. Steps of Algorithm 1 applied to a 2-bit arithmetic logic unit targeting four configurations per output bit. (a) Step I: choose $v$ and its neighborhood (highlighted in red) and Step II: choose $u$ and its neighborhood (highlighted in blue). (b) Step III: place SBs (shown in green), Step IV: test that SBs are non-degenerate (satisfied) and choose configurations (shown in dotted lines), and Step V: test for at least two SBs within logic cone of each output bit (satisfied).

While it is possible to insert SBs into the original functions instead of the checking circuitry, this leaves the entire end-to-end CED scheme visible and it can be modified to never detect errors in the original functions. Thus, a trusted end-to-end CED scheme is imperative for detecting Trojans. When split manufacturing is used, SB programmability is not needed. The area savings with split manufacturing are discussed in Sections IV-B–V-A.

### A. Resilience to Sophisticated Attacks

In this section, we provide results showing the effectiveness of SB programmability in preventing attacks that require some detailed knowledge of the design. First, we state an assumption that is used to set the number of configurations input to Algorithm 1 when it is applied in Sections IV-B–V.

*Assumption 1:* If an adversary learns the definition of any characteristic function computed by any attack detection circuit (e.g., logic CED, output encoding, and input decoding), a Trojan can be inserted into the design that cannot be detected.

For example, assume that an OCP with two check bits is designed for a function with four outputs. Assume that the first check bit is used to predict the parity of output bits 1 and 2, while the second check bit is used to predict the parity of output bits 3 and 4. If an adversary figures out the first characteristic function and flips the first two bits at the output of the main function, this attack would not be detected.

Assumption 1 is used in Step V of Algorithm 1. The Trojan detection circuitry is decomposed into separate logic cones for each output bit. Then, programmability is selectively inserted into each logic cone in order to hide the functionality.

The following list states reverse engineering attacks and how they are prevented by SB programmability. Each attack was simulated on an OpenRISC CPU protected with randomized parity codes (Section IV). The probability that the attack succeeded is shown in Table I. An attacker can try all possible SB configurations or examine the physical aspects (i.e., gates and/or wires) of the circuit or subcircuits. A more general attack model [configuration prediction (CP) attack] assumes the attacker can determine the correct configuration for each SB with a given probability. Even if this probability is high, the overall probability of reverse engineering the circuit can still be made negligible by inserting a large number of SBs.

*1) Brute Force Attacks:* The attacker isolates a single output bit in the netlist, and searches through all possible SB

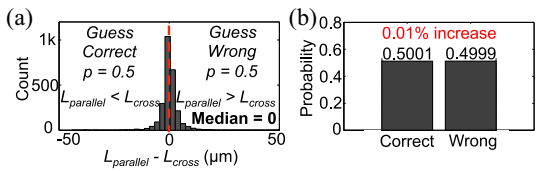


Fig. 14. (a) Wire-length tracing results. The attack succeeds 50% of the time. (b) DS and FO matching attack results. Using this attack resulted in only a 0.01% advantage in success rate over random guessing.

configurations along the logic cone leading to it. For example, the adversary may configure the SBs in all possible ways, create a list of all possible functionalities, and then guess the intended configuration. We prevent these attacks by inserting a large number of SBs within each logic cone leading to an output bit in all of the Trojan detection circuits. For example, in Section IV-B, we insert between 64 and 80 SBs along every logic cone in the Trojan detection blocks of a CPU. The number of SB configurations of each cone is too many to exhaustively test in order to reverse engineer the functionality.

*2) Configuration Prediction Attacks:* In a $\langle\theta\rangle$ CP attack, the adversary can correctly guess the intended configuration of any SB with probability $1-\theta$. Since the intended configuration of each SB is chosen independent of the others, the minimum bias strategy to guess the overall configuration is to guess each SB configuration independently [88]. Hence, if there are $x$ SBs within a logic cone, a $\langle\theta\rangle$ CP attack assumes an adversary can guess the configuration of that logic cone correctly with probability $(1-\theta)^x$. We prevent such attacks by inserting a large number of SBs within each logic cone leading to an output bit in all of the Trojan detection circuits, so these attacks succeed with negligible probability (i.e., $x$ is made large so that $(1-\theta)^x$ will be small; Table I).

*3) Wire-Length Tracing:* If a physical design tool was given the intended configurations of the SBs, wire lengths for the intended SB configurations could be optimized to minimize delay, while wire lengths for incorrect configurations could be ignored. Then, an adversary could compare the wire lengths of both configurations for each SB and find the one resulting in shorter wires. We prevent this attack by not providing the tool with the intended configurations (Section VI). Thus, no single SB configuration is optimized at the expense of others during physical design [Fig. 14(a) and Table I]. To demonstrate this, SBs in the parallel configuration were inserted in various OpenRISC OCP designs (Section IV) with the intended configuration undisclosed to the physical design tool. The difference in the maximum wire lengths for parallel ($L_{\text{parallel}}$) and crossed ($L_{\text{cross}}$) configurations was determined for each SB. If $L_{\text{parallel}} < L_{\text{cross}}$, the adversary would guess parallel; otherwise, the adversary would guess crossed. The median value (out of 2500 trials) of $L_{\text{parallel}} - L_{\text{cross}}$ was 0, indicating that an adversary would guess correctly with probability 0.5 (no advantage over random guessing).

*4) Drive Strength and Fanout Matching:* If SBs were inserted post-physical design, an adversary would be able to match (strong or weak) driving cells at the inputs of SBs to (large or small) fanouts (FOs) at the outputs. For example, let SB inputs be driven by gates with drive strength (DS) 1× and 4×, and SB outputs have FOs 1 pF and 4 pF. An adversary can guess that the 4× cell was meant to drive the 4 pF load to minimize delay, thus revealing the SB configuration. We prevent this attack by inserting SBs before physical design and using SBs with equal capacitances at the input ports. Since physical design will size both driving cells to drive the SB, the DS of both inputs will be similar, regardless of FO. This attack was implemented 4000 times on various OpenRISC OCP designs. Random guesses for the SB configuration were used to break ties. As shown in Fig. 14(b), this attack provided only a 0.01% increase in success rate when compared to random guessing.

*5) Subcircuit Matching:* An adversary can search for isomorphic subcircuits shared by a function and its OCP. Since the OCP is a composition of characteristic functions with the main function, isomorphic subcircuits may be used to compute the same logic values in both blocks. An attack that flips the outputs of both subcircuits would not be detected by the CED tests, since check bits would still match. SB insertion breaks up logic cones in the OCP, leading to far fewer isomorphic subcircuits. Equal modification to the remaining isomorphic subcircuits is detected with high probability, indicating that even though they are constructed with the same logic gates, they are not used to compute the same logic values (Table I). This attack was implemented on the OpenRISC CPU (Section IV) 40 000 times and was unsuccessful every time.

TABLE II
NOTATION

| | |
|---|---|
| $\{0,1\}^n$ | Binary strings of length $n$ |
| $x_i, i \in \mathbb{Z}$ | $i$th element of vector $x$. |
| $x^T$ | Transpose of vector $x$. |
| $\oplus$ | Modulo-2 sum (XOR). |
| $\mathbb{F}_2$ | Binary Field: {0,1} with $\oplus$, mod-2 multiplication |
| $M^{n\times k}(\mathbb{F})$ | Set of $n\times k$ matrices over field $\mathbb{F}$. |

## IV. Randomized Coding for Logic CED and Secure I/Os

### A. Randomized Parity Codes

We assume that the reader has familiarity with the basics of coding theory [22] and its use in fault-tolerant computing [17].

We provide only the minimum definitions here. The notation used is defined in Table II. All the presented codes are binary, but they can be extended to larger finite fields.

*Definition 2:* A $k$-dimensional parity function $g : \{0,1\}^k \to \{0,1\}$ is a parity of a subset of bits of a $k$-dimensional binary vector. Explicitly, $\forall x \in \{0,1\}^k$

$$g(x) = \bigoplus_{i=1}^{k} a_i x_i,\ a_i \in \{0,1\}.$$

*Definition 3:* An $(n,k)$ parity (linear) encoding $g : \{0,1\}^k \to \{0,1\}^n$ is a mapping where each coordinate function is a $k$-dimensional parity function. We call $n$ is the blocklength of the code, and $k$ is the number of information bits.

*Definition 4:* A parity-check matrix $H \in M^{r\times n}(\mathbb{F}_2)$ for an $(n,k)$ linear encoding $g$ is a binary matrix for which $Hx^T = 0$ iff. $\exists y \in \{0,1\}^k$ such that $g(y) = x$. $C \subseteq \{0,1\}^n$ with $C = \{x \in \{0,1\}^n | Hx^T = 0\}$ is called the code. The elements of $C$ are called the code words. A linear code is systematic if its parity check matrix has form $[A|I_r]$ where $I_r$ is the $r \times r$ identity matrix and $r = n - k$ is the number of check bits.

Fig. 15 illustrates how linear systematic codes are used in CED [30]. The information bits of a code word are computed by a logic function. The OCP predicts the corresponding check bits, and the checker computes the actual check bits and compares them with the prediction. We now introduce a randomized linear code construction.

*Construction 1 (Randomized Parity Codes):* Let $R^{n\times k}$ denote the ensemble of all $(n,k)$ linear systematic codes with all

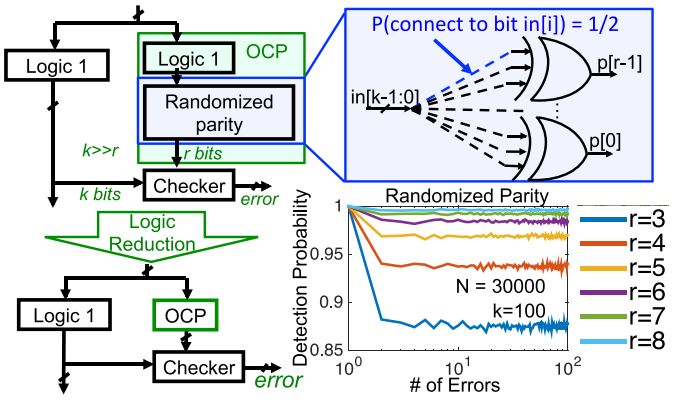


Fig. 15. General method to create an OCP. Detection probability for various number of check bits for $k = 100$ is shown for randomized parity codes.

**Algorithm 2** Randomized Parity Construction

**Inputs:** Number of information bits $k$. Number of check bits $r$.
**Output:** Parity-check matrix for an $(r+k, k)$ randomized parity code.
Step I: Choose a systematic parity-check matrix $H \in M^{r\times(r+k)}(\mathbb{F}_2)$ uniformly at random.
Step II: If $H$ satisfies constraints of Construction 1: output $H$. Else: repeat Steps I-II.

rows and all columns in the parity check matrix nonzero. An $(n, k)$ randomized parity code is a code that has been sampled uniformly at random from $R^{n\times k}$.

Randomization of the code choice, in conjunction with SB programmability (Section III), prevents an adversary from knowing the parity check matrix. The nonzero constraint on each row of the parity check matrix is to avoid the zero function. The nonzero constraint on the columns is to ensure that each input bit is included in at least one parity function. Algorithm 2 constructs a randomized parity code.

For any given error pattern, a code chosen uniformly at random from the ensemble of all $(n, k)$ binary linear systematic codes will detect the errors with probability $1 - 0.5^r$. However, since Construction 1 introduces dependencies between columns in the parity check matrix, the standard proof of the error detection probability will not hold [33]. Fig. 15, therefore, shows the detection probability for randomized parity codes, for $k = 100$, simulated using Monte Carlo (30 000 trials). The detection probability is 1 for single-bit errors and approaches $1 - 0.5^r$ as the number of errors increases [34]. If a nonlinear code is used instead of a parity code, the detection probability for a large number of errors can be higher at the cost of additional area [31].

### B. OpenRISC CPU Demonstration

To evaluate TPAD, we constructed an OpenRISC 1200 CPU core [35] using randomized parity code checking (Section IV-A) for logic CED, input decoding, and output encoding. Designs were synthesized and place-and-routed with the Nangate 45 nm CMOS standard cell library [36]. The number of check bits for each pipeline stage ranged from 3 to 8 bits, and SB programmability (Section III) was included for all checking circuitry and I/Os. The same number of check bits was used for all pipeline stages within the processor. The relative size and power of the overall design for three check bits (which corresponds to a detection rate of 87.5%) is given in Fig. 16. This shows an area overhead of 114% and

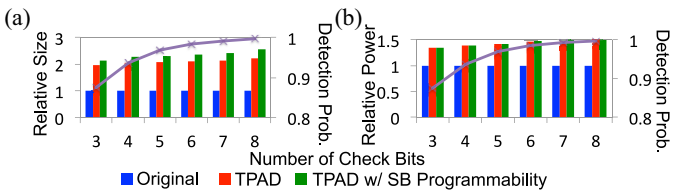


Fig. 16. (a) Area and (b) power overheads and detection probability for secure OpenRISC 1200 processor design using randomized parity codes with $2^{64}$ SB configurations per check bit for all OCPs and checkers.

TABLE III
AREA, POWER, AND DELAY OVERHEAD TRADEOFFS FOR OPENRISC

| $r$ | Overheads for # of SB configurations / OCP bit | | | | | | | | | Logic Attack Detection Probability |
|---|---|---|---|---|---|---|---|---|---|---|
| | $2^{64}$ | | | $2^{72}$ | | | $2^{80}$ | | | |
| | Area | Power | Delay | Area | Power | Delay | Area | Power | Delay | |
| 3 | 114 | 34 | 0 | 122 | 34 | 0 | 121 | 34 | 0 | 0.875 |
| 4 | 127 | 39 | 0 | 130 | 39 | 0 | 131 | 39 | 0 | 0.937 |
| 5 | 131 | 43 | 0 | 135 | 43 | 0 | 139 | 43 | 0 | 0.968 |
| 6 | 137 | 47 | 0 | 142 | 47 | 0 | 145 | 47 | 0 | 0.984 |
| 7 | 142 | 53 | 0 | 146 | 53 | 0 | 150 | 53 | 0 | 0.992 |
| 8 | 157 | 60 | 0 | 158 | 60 | 0 | 165 | 61 | 0 | 0.996 |

power overhead of 34%. The number of SB configurations used for each bit of both the OCP and characteristic function in Algorithm 1 was $2^{64}$. This corresponds to a delay increase of 20% (the critical path is now in the OCP) if the same number of pipeline stages is used. However, this delay overhead can be reduced to close to 0 if an extra pipeline stage is added in one of the OCPs and its corresponding checker. The area and power overheads of this change were found to be negligible. Additional results for $2^{72}$ and $2^{80}$ SB configurations for each OCP bit are given in Table III. To simulate the detection probability, a uniformly random number of bit flips were introduced to the system at uniformly random times.

Most of the area overhead of randomized parity coding is due to the inefficient logic reduction of the OCP. This results in a significant portion of the OCP resembling the original function. While this seems to imply that the chip is vulnerable to sophisticated attacks (i.e., an adversary can make the same change to the original function and OCP), SB programmability prevents such attacks by breaking up logic cones in the OCP (Section III). Furthermore, >99% of the original boundary nets (i.e., $in[k-1:0]$ in Fig. 15) have been reduced, rendering finding full logic cones from the original circuit difficult (see [89, Sec. IV-B] for details). Although SB programmability creates additional overhead, the marginal area (17–33%) and power (0.4%) costs are low.

## V. OVERHEAD REDUCTION VIA SPECIALIZED OCPS

While randomized parity for logic CED offers high Trojan detection probability for arbitrary designs, it does not necessarily offer the best area overheads (similar to application-specific tradeoffs for fault-tolerant computing [17]). Application-specific methods can possibly decrease the area overhead. However, since sophisticated Trojans are able to escape many checks (as discussed in Section II), application-specific CED techniques for fault-tolerant computing cannot be used unmodified for TPAD. For example, sum of squares (a common CED technique for FFT circuits [32]) cannot detect any attack that permutes (in any arbitrary way) the FFT outputs.

### A. LZ Data Compression Engine

For certain invertible functions, the inverse is simpler to implement than the forward function. In such cases, TPAD can

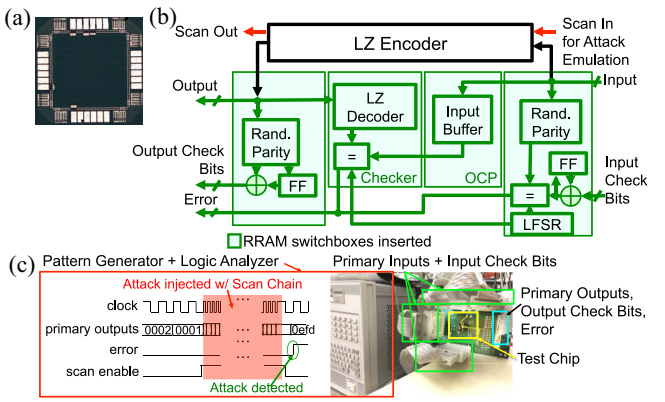


Fig. 17. LZ Encoder design using TPAD. (a) Die photo of LZ encoder using TPAD fabricated in 65 nm technology. (b) Block diagram of an LZ encoder using TPAD. The OCP is an input buffer while the checker is made from the inverse function and an equality checker. RRAM SBs (64) were inserted using Algorithm 1 along the cones ending at each of the 24 output bits. (c) Test setup for Trojan emulation with sample output result.

use the inverse to detect Trojans efficiently. We demonstrate this by implementing an LZ data compressor chip. The chip uses the sliding window version of the LZ data compression algorithm [38] (i.e., LZ77). The CED scheme used to protect the encoder is described in [21]. A LZ77 code word is represented as a fixed length binary tuple $(C_p, C_l, C_n)$, where $C_p$ is the pointer to the location in the dictionary, $C_l$ is the match length, and $C_n$ is the next character.

To detect attacks, the LZ code word is decompressed by a dictionary lookup and compared with a copy of the input stored in a buffer. If there is any mismatch, an error has occurred. Because LZ77 is a lossless compression algorithm, changing a code word to any other word will result in errors being detected. This inverse function method alone, however, is not enough to prevent an adversary from inserting Trojans. For instance, an adversary can make the same change to the output of the inverse function and the output of the input buffer to make the compared words match. Thus, SB programmability realized with RRAM (Section III) is needed for the CED. Moreover, output encoding and input decoding for the I/Os of the chip are needed to protect against pin attacks.

Since the dictionary keeps a record of past inputs, this CED method will detect not only any logic attack, but also decoupling attacks [89, Appendix A] in the compressor since the pointers to each dictionary entry change after every cycle.

A block diagram and die photo of our split-manufactured test chip, implemented in 65 nm technology, with secure I/O encoding (Sections II-A and II-B) is shown in Fig. 17. To emulate logic attacks, a number of random flip-flops inside the chip were flipped at random times using a scan chain. To emulate electrical attacks, flip-flop values were reverted to their previous values (setup time violations) or changed to their next values (hold time violations). To emulate reliability attacks, internal wires were forced high or low at random times. Pin attacks [89, Appendix A] were emulated by flipping a uniformly random number of bits at the primary inputs at a uniformly random time. The test setup used and a sample result are shown in Fig. 17(c). The error signal was monitored for detected attacks and we observed a 99.998% detection rate for logic attacks (Table IV) and 99.6% for pin attacks.

The 0.002% of undetected logic attacks consisted of those that degraded the CR (uncompressed size/compressed size) but did not alter the decompressed word [21]. For example, when the LZ compressor is stuck at an "unmatched" state, meaning

TABLE IV
COVERAGE AND OVERHEAD FOR SPECIALIZED OCP EXAMPLES

| Area/Power Overhead (%) | | Attack Coverage (%) | | | | |
|---|---|---|---|---|---|---|
| Circuit: | Secure Arch | Random Error | Pin | Logic | Electrical | Reliability |
| LZ77 | 27/9 | 99.998 | 99.6 | 99.998 | 100 | 100 |
| FFT 128 | 7.4/7 | 100 | 99.6 | 99 | 100 | 100 |

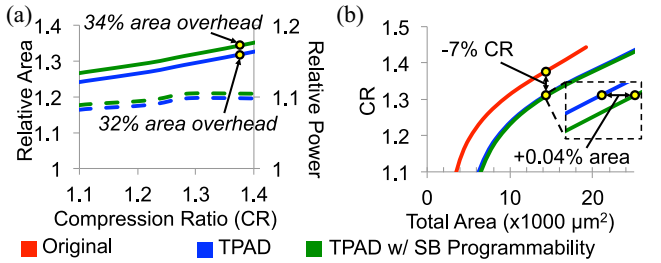


Fig. 18. CR and area tradeoff for LZ encoder design. (a) Relative area and power compared to an LZ encoder design without TPAD for fixed CR. (b) CR for fixed area. Results shown are after physical design.

no input character ever matches anything in the dictionary, no compression is achieved ($C_l = 0$). Thus, the LZ decompressor will output the original string.

The tradeoff between the CR and the area overhead required to secure the design was further analyzed. Configurations of the LZ77 compressor for dictionary sizes ranging from 16 to 256 words, maximum matched length ranging from 4 to 256 words, and input word width ranging from 8 to 32 bits were synthesized and place-and-routed using the Nangate 45 nm standard cell library [36]. The tradeoff between the CR and the area is shown in Fig. 18 (details in [89]).

For example, suppose a design requires a CR of 1.38. The area overhead of implementing this design with TPAD using SB programmability would be 34% with a 10% power overhead [shown in Fig. 18(a)]. If SB programmability is not needed, the area overhead would decrease to 32%. Sacrificing a small amount of CR can reduce the area cost further. If 0% area overhead is needed, one can incur a 7% decrease in CR from 1.38 to secure the design [Fig. 18(b)]. The area overhead of SB programmability is 0.04%.

### B. Fast Fourier Transform Engine

To further demonstrate the impact that OCP selection can have on the area overhead of TPAD, we implement a pipelined FFT engine. The FFT engine is implemented using the Cooley–Tukey algorithm [37] with half-precision floating-point arithmetic.

While Parseval's theorem is a popular CED method for FFT circuits [32], an adversary can change the output of the FFT such that the sum of squares is still preserved. For example, an adversary can permute the outputs in arbitrary ways and still preserve the sum of squares. Thus, Parseval's theorem cannot be used for Trojan detection. Instead, we make use of the Plancherel theorem for the discrete Fourier transform, explained in [93, Sec. V-B]. Distinguishing between attacks and finite-precision errors is also detailed in [89].

A block diagram of the architecture is shown in Fig. 19(a). The OCP computes the inner product of the input and a vector $y^*$ while the checker computes the inner product of the output and another vector $Y^*$ and compares it to the output of the OCP. Here $y$ and $Y$ form a precomputed Fourier transform pair (details on how to choose this pair are discussed in [89]). The design was synthesized and place-and-routed in Nangate

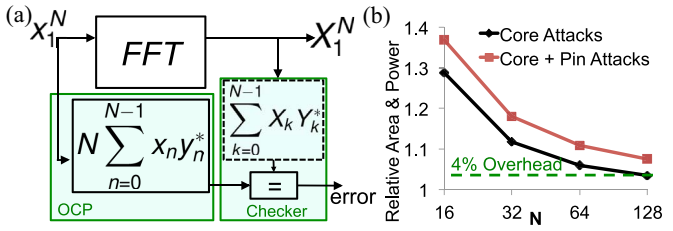


Fig. 19. (a) Block diagram of the specialized OCP function implementing the Plancherel theorem. (b) Relative area/power compared to those of the FFT.

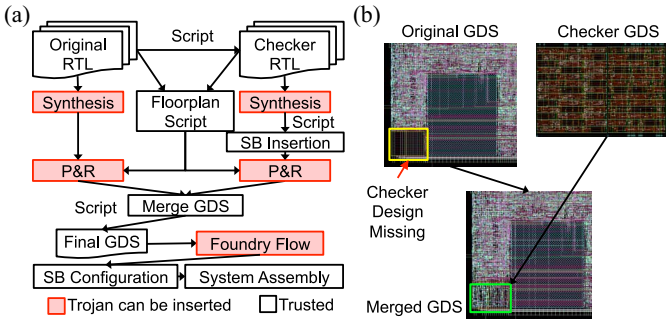


Fig. 20. (a) Split design flow for security against compromised CAD tools. The potentially compromised tools are highlighted in red. Synthesis tools and physical design tools are potentially compromised. (b) Original GDSII and checker GDSII are merged into a single GDSII file.

45 nm technology. Fig. 19(b) shows the overhead compared to the number of points in the FFT. For larger FFT designs, the overhead decreases. The CED circuits take 4% area overhead for a 128-point FFT. I/O encoding to protect against pin attacks incurs another 3.4% area overhead, yielding a total of 7.4% area overhead. The power overheads were found to be the same as the area overheads.

Logic attack coverage was found to be 99% (Table IV), where the 1% corresponded to Trojans that altered both the OCP and FFT outputs such that the Plancherel theorem was still satisfied for the $y$ and $Y$ vectors chosen. In these simulations, random white noise was used for $y$ and its corresponding FFT was used as $Y$. The vectors used had small norm to prevent overflow and none of the values in $y$ or $Y$ were zero. All electrical and reliability attacks were caught while pin attack coverage remained at 99.6% with eight check bits.

## VI. Prevention of CAD Tool Attacks

While SB programmability protects against reverse engineering during fabrication, the design phase is still vulnerable to attacks by CAD tools [6]. Since CAD tools have complete information of the entire chip design, it is possible for them to strategically insert Trojans such that the Trojan detection circuitry cannot detect them (e.g., a Trojan is inserted in both the main function and in the OCP such that predicted check bits and actual check bits are still equal). Verification-based methods for detecting Trojans can be used [13], [87] but are prone to false negatives (i.e., the Trojan may not be detected). It is likely that all fabricated chips will contain such Trojans; thus, these Trojans can be detected by destructively testing a few chips after fabrication (Section VII-A). However, if CAD tool providers have close relationships with foundries, it may be possible to fabricate a limited amount of chips containing Trojans along with the Trojan-free chips (Section VII-A).

A split-design flow with TPAD can ensure Trojan detection (after fabrication). The split design flow [shown in Fig. 20(a)] is used to decouple checker information from the original design information. As mentioned in Section II, we assume the original RTL is Trojan free; thus, it is used by a trusted script to generate the RTL for the Trojan detection circuitry (i.e., input decoding, output encoding, logic CED, and error encoding in Fig. 5). The original RTL and Trojan detection RTL are synthesized separately to prevent any CAD tool from receiving complete design information at any time. There must be no retiming during synthesis of each part to preserve the timing for the final netlist. A malicious synthesis or physical design tool would have no knowledge of the other part of the chip when inserting a Trojan. Fig. 20(b) illustrates the outcomes of this split design flow. A trusted floor planning script is used to partition the die into separate blocks (one for original design and one for Trojan detection circuitry). The original and Trojan detection portions are place-and-routed separately, each keeping its counterpart black-boxed. After physical design, the two layouts are merged with a simple, trusted, script and sent for fabrication. The split design flow is required when malicious CAD tools are assumed in the threat model. Requirements are summarized in Table V.

TABLE VI
Comparison of Split Design Flow Versus Nonsplit Flow

| Circuit | Performance Loss | Area Difference | LVS Pass |
|---|---|---|---|
| LZ77 | No | 0% | Yes |
| OpenRISC | No | 0% | Yes |
| FFT64 | No | 0% | Yes |

Designs were synthesized and placed-and-routed with our split design flow and compared with those without the split design flow. Specifically, circuits were compared for functionality (LVS), critical path delay, and total area. Our comparison results (Table VI) show no decrease in performance.

## VII. Survey of Trojan Detection Methods

### A. Destructive Versus Nondestructive Methods

Methods for detecting hardware Trojans can be divided into destructive and nondestructive methods. Destructive methods include postmanufacture tests and/or inspections that render the chip to be nonoperational (or change its lifetime reliability characteristics). Destructive methods are ideal for identifying Trojans that affect all (or a large number of) chips in a batch, since testing a few chips would be enough. For example, scanning electron microscopy (SEM) or focused ion beam (FIB) [45] could be used to destructively delayer chips and verify transistors and interconnects [89]. Destructive stress tests may be used to detect reliability attacks [41], [89].

However, this approach becomes difficult when Trojans are only inserted into a selected few chips. For example, assume that if a chip containing a Trojan is destructively tested or inspected, the Trojan will be found and that particular chip will be destroyed. If only 0.05% of chips have Trojans out of a batch of 10 000 chips, 58% of the chips must be destroyed to achieve 99% Trojan detection probability. Thus, nondestructive methods are needed when a relatively small number of chips are attacked. This is illustrated in Fig. 21, and the analysis is detailed in [93, Sec. VII-A].

### B. TPAD Versus Other Techniques

The following list categorizes various methods for TPAD. A comparison is given in Table VII.

TABLE V
SUMMARY OF HARDWARE REQUIREMENTS UNDER VARIOUS MANUFACTURING AND ATTACK MODELS

| Feature: | Manufacturing Technology | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | No Split-manufacturing | | | | | | Split-manufacturing after M1 | | | | | |
| | CAD Attacks Only | | Foundry Attacks Only | | | Both | CAD Attacks Only | | Foundry Attacks Only | | | Both |
| | Logic/ Pin/ Electrical | Reverse Engineer | Logic/ Pin/ Electrical | Reverse Engineer | Reliability | All | Logic/ Pin/ Electrical | Reverse Engineer | Logic/ Pin/ Electrical | Reverse Engineer | Reliability | All |
| TPAD Arch. | R | N | R | R | R | R | R | N | R | N | R | R |
| Programmability | R | R | R | R | N | R | N | N | N | N | N | N |
| Split Design | R | R | N | N | N | R | R | R | N | N | N | R |

R – This feature is **required** to protect against the above set of attacks when the above manufacturing technology is used.
N – This feature is **not required** to protect against the above set of attacks when the above manufacturing technology is used.

TABLE VII
COMPARISON OF DIFFERENT TROJAN DETECTION AND PREVENTION SCHEMES

| Approach | Refs | Prevention or Detection | Destructive | Split-manufacture Required? | Attacks Covered | Area Overhead | Concurrent | False Positives |
|---|---|---|---|---|---|---|---|---|
| TPAD | This paper | Both | No | No | 1-6 | 7.4 - 165% based on design | Yes | No |
| Split after M1 | [43]-[44] | Prevention | No | Yes | 1 | 0 % | No | N/A |
| Test-only BEOL | [45] | Both | Yes | Yes | 1-5 (~) | Dies destroyed | No | No |
| 3D chip stacking | [46][94] | Both | No | Yes | 1, 3-6 | TSVs, Control Chip | Yes | No |
| EMR Imaging | [48],[50] | Detection | No | No | 2-4 (^+) | 0 % | No | No |
| SEM or FIB | [49] | Detection | Yes | No | 2-5 (~) | Dies destroyed | No | No |
| XOR or LUT Insertion | [51][52] | Prevention | No | No | 1 | Gates + Pins | No | N/A |
| Silencing | [53] | Prevention | No | No | 1-3($) | Encryption, Input Reorder | Yes | N/A |
| HARPOON | [54][55] | Prevention | No | No | 1 | 5 – 20 % | No | N/A |
| FANCI, FIGHT | [56][57] | Prevention | No | No | 1 ($) | 0 % | No | Yes |
| Control Monitoring | [58][59] | Detection | No | Yes (#) | 2-6(&) | Monitoring Circuits | Yes | No |
| FPGA w/ Checking | [60]-[63] | Both | No | No | 1, 3-6 | FPGA | Yes | No |
| On-chip sensing of delays | [64]-[68] | Detection | No | Yes (#) | 2-6(&*) | Second clock, latches, sensors. | Can be | Yes |
| BISA | [69][70] | Detection | No | No | 2-4(&*) | ATPG logic and pins | No | No |
| TeSR | [71][72] | Detection | No | Yes (#) | 2-4(&*) | Retry+Sense+Compare logic | No | Yes |
| Dynamic power | [73][74] | Detection | No | No | 2-4 (*^) | Dummy FFs | No | Yes |
| GLC | [77]-[79] | Detection | No | No | 2-4 (*^) | 0 % | No | Yes |
| Multimodal Fingerprinting | [80]-[83] | Detection | No | Can be (#) | 2-4 (*^) | Can be 0% | Can be | Yes |
| Random Test Patterns | [85]-[86] | Detection | No | No | 2-4 (^) | 0 % | No | No |
| ODETTE/ VITAMIN | [87]-[88] | Both | No | Yes (#) | 1-4 (^) | $Q$ and $\overline{Q}$ pins | No | No |

Attack Coverage Meanings: 1-Prevents Reverse-Engineering, 2-Detects Pin Attacks, 3-Detects Logic Attacks, 4-Detects Electrical Attacks, 5-Detects Reliability Attacks, 6-Detects Circuit Errors. Conditions Meanings: *-Determined by variability/confidence interval, &-Only along monitored paths, ^- Only those found by test vectors, +-Only top/bottom layers and uncongested middle, ~-Remaining chips insecure, $-RTL Level Only, #-Security monitors built by trusted foundry

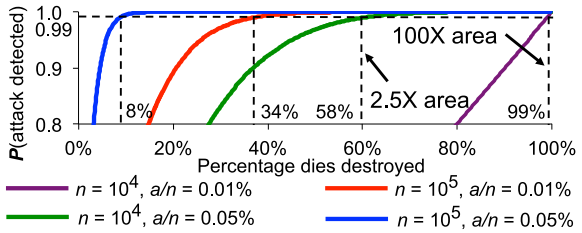


Fig. 21. Probability of detecting a Trojan in a batch of chips with destructive testing of individually sampled chips. Curves are labeled with $n$: total number of chips in batch and $a$: number of chips with Trojans.

*1) Split Manufacturing:* A split at the first metal layer (or very low metal layers) can prevent an untrusted foundry from reverse engineering a design since most interconnects are missing [39], [40]. However, split manufacturing does not provide Trojan detection by itself. Destructive stress tests on selected back-end-of-line stacks can detect reliability attacks [41]. Trusted monitor chips stacked on top of an untrusted chip using through-silicon vias (TSVs) may be used to actively detect attacks [42] or selected wires may be lifted (using TSVs) to a secure layer to obfuscate a design [90]. Large TSV pitch can lead to area inefficiencies [43].

*2) Imaging:* Trojans can be detected by electromagnetic radiation imaging to search for physical anomalies [44], [46]. This approach does not prevent an adversary from reverse engineering the design. SEM or FIB can also be used, but require delayering the chip, thus destroying it [45].

*3) Logic Encryption:* These nondestructive techniques prevent, but do not detect, Trojans by placing additional logic gates and making them partially controllable [47], [48]. Reference [49] is a concurrent method to prevent Trojans by using external modules to control operation. However, delay penalties are introduced and additional pins are required.

*4) FSM Analysis:* The state transition diagram of a circuit is modified so that a specific sequence of transitions must take place before the circuit operates [50], [51]. Since an untrusted foundry could have access to the entire design, the state transitions can possibly be inferred. RTL-level Trojans may be detected by identifying signals that rarely change [52], [53]. This approach can result in false positives, and it cannot detect Trojans inserted by the foundry.

*5) Active Monitoring:* These nondestructive techniques allow for Trojan detection by monitoring (some of) a chip's functional behavior during runtime [54]–[59] (e.g., control signals in a processor). Reference [54] is limited to RTL-level

Trojans. Reference [55] requires a snapshot of all I/O signals of a chip to be obtained periodically and analyzed for attacks. These techniques only monitor a subset of signals (and the properties to be checked may not be exhaustive), leaving the systems vulnerable. References [56]–[59] require reconfigurable logic (e.g., FPGAs), thus increasing area overheads.

*6) Delay Fingerprinting:* Delay measurements are taken along certain paths and compared to those from pretapeout timing analysis [60]–[62]. If a Trojan is inserted, the delay may not match the model. References [63] and [64] presented strategies for inserting test points to make various paths observable, these methods are prone to false positives since Trojans must be distinguished from variability and noise [80].

*7) Switching Power Analysis:* These techniques detect Trojans by measuring dynamic current waveforms and comparing with a model for an intended design [67]–[72]. If there is a significant difference between the two current waveforms, a Trojan is assumed. References [71] and [72] are concurrent solutions that perform classification in real time. These methods are prone to false positives since Trojans must be distinguished from variability and noise [80].

*8) Gate-Level Characterization:* Measurements are taken, leakage power [73], [74] or delays of individual gates [75] are estimated, and Trojans are identified. Gate-level characterization (GLC) shares the limitations of other fingerprinting techniques (e.g., [80]).

*9) Multimode Fingerprinting:* Several fingerprinting methods are combined and GLC can be used to compare the results with a model [76]–[79]. While stronger, these methods share similar limitations as GLC.

*10) Activation by Test Vectors:* Random test patterns are used to attempt to activate Trojans during post-manufacture testing [81]–[84]. Banga and Hsiao [83], [84] introduced programmability to accelerate Trojan activation during post-manufacture testing and to provide logic encryption. However, these methods are nonconcurrent; thus, they are not able to detect Trojans that activate after post-manufacture testing (e.g., time bombs and reliability attacks).

## VIII. Conclusion

TPAD protects digital systems from hardware Trojan attacks through a combination of special CED techniques and selective programmability. As a result, it can prevent and detect hardware Trojans with high probability and no false positives. We demonstrated a variety of detection techniques, from general randomized parity coding (that can be expensive for general designs) to specialized detection techniques (tailored for special functions) that significantly reduce TPAD area overheads. Hardware test chip results demonstrate the effectiveness and practicality of TPAD.

While many aspects of TPAD have been extensively explored in this paper, several opportunities for future work remain. These include as follows.

1) SB insertion algorithms that avoid degenerate cases and/or ensure that multiple checking circuits can be created (Section III).
2) Software checking techniques that can complement TPAD for systems supporting both hardware and software programmability.
3) Techniques for recovering from hardware Trojan attacks [89, Appendix B].

## References


[1] S. T. King *et al.*, "Designing and implementing malicious hardware," in *Proc. LEET*, San Francisco, CA, USA, 2008, pp. 1–8.

[2] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware Trojans," *Computer*, vol. 43, no. 10, pp. 39–46, Oct. 2010.

[3] O. Sinanoglu *et al.*, "Reconciling the IC test and security dichotomy," in *Proc. 18th IEEE Eur. Test Symp.*, Avignon, France, May 2013, pp. 1–6.

[4] Y. Shiyanovskii *et al.*, "Process reliability based Trojans through NBTI and HCI effects," in *Proc. NASA/ESA AHS*, Anaheim, CA, USA, Jun. 2010, pp. 215–222.

[5] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan./Feb. 2010.

[6] J. A. Roy, F. Koushanfar, and I. L. Markov, "Circuit CAD tools as a security threat," in *Proc. HOST*, Anaheim, CA, USA, Jun. 2008, pp. 65–66.

[7] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating UCI: Building stealthy and malicious hardware," in *Proc. IEEE Symp. Security Privacy (SP)*, Oakland, CA, USA, 2011, pp. 64–77.

[8] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," in *Proc. CHES,* Lausanne, Switzerland, 2009, pp. 363–381.

[9] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware Trojans: Extended version," *J. Cryptogr. Eng.*, vol. 4, no. 1, pp. 19–31, Apr. 2014.

[10] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware trojans through side-channel engineering," in *Proc. CHES*, Lausanne, Switzerland, 2009, pp. 382–395.

[11] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Proc HOST*, Anaheim, CA, USA, Jun. 2008, pp. 15–19.

[12] S. Wei, K. Li, F. Koushanfar, and M. Potkonjak, "Hardware Trojan horse benchmark via optimal creation and placement of malicious circuitry," in *Proc. DAC*, San Francisco, CA, USA, Jun. 2012, pp. 90–95.

[13] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "VeriTrust: Verification for hardware trust," in *Proc. DAC*, Austin, TX, USA, 2013, pp. 1–8.

[14] V. Tomashevich, Y. Neumeier, R. Kumar, O. Keren, and I. Polian, "Protecting cryptographic hardware against malicious attacks by nonlinear robust codes," in *Proc. DFTS*, Amsterdam, The Netherlands, Oct. 2014, pp. 40–45.

[15] IARPA. (2011). *TIC Program*. [Online]. Available: http://tinyurl.com/marcypp

[16] P. Batude *et al.*, "Advances, challenge and opportunities in 3D CMOS sequential integration," in *Proc. IEDM*, Washington, DC, USA, Dec. 2011, pp. 151–154.

[17] D. K. Pradhan, *Fault-Tolerant Computer System Design*. Upper Saddle River, NJ, USA: Prentice Hall, 1996.

[18] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture," in *Proc. FPGA*, Monterey, CA, USA, Feb. 2011, pp. 5–14.

[19] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 216–229, Feb. 2007.

[20] Altera. (Jul. 5, 2014). *The Industry's First Floating-Point FPGA*. [Online]. Available: http://www.altera.com/literature/po/bg-floating-point-fpga.pdf

[21] W.-J. Huang, N. Saxena, and E. J. McCluskey, "A reliable LZ data compressor on reconfigurable coprocessors," in *Proc. FCCM*, Napa Valley, CA, USA, 2000, pp. 249–258.

[22] S. Lin and D. J. Costello, *Error Control Coding*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[23] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability*. Boston, MA, USA: Morgan Kaufmann, 2006, p. 519.

[24] A. Shamir, "On the generation of cryptographically strong pseudorandom sequences," *ACM Trans. Comput. Syst.*, vol. 1, no. 1, pp. 38–44, Feb. 1983.

[25] N. R. Saxena and E. J. McCluskey, "Primitive polynomial generation algorithms implementation and performance analysis," Dept. Electr. Eng., Stanford CRC, Stanford, CA, USA, Tech. Rep. 04-03, Apr. 2004.

[26] T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only," *IEEE Trans. Comput.*, vol. C-34, no. 1, pp. 81–85, Jan. 1985.

[27] X. Ying and X. Wu, "Graph generation with prescribed feature constraints," in *Proc. SDM*, vol. 9. Sparks, NV, USA, 2009, pp. 966–977.

[28] Y. Y. Liauw, Z. Zhang, W. Kim, A. E. Gamal, and S. S. Wong, "Nonvolatile 3D-FPGA with monolithically stacked RRAM-based configuration memory," in *Proc. ISSCC*, San Francisco, CA, USA, Feb. 2012, pp. 406–408.

[29] C.-H. Wang *et al.*, "Three-dimensional $4F^2$ ReRAM with vertical BJT driver by CMOS logic compatible process," *IEEE Trans. Electron Devices*, vol. 58, no. 8, pp. 2466–2472, Aug. 2011.

[30] N. A. Touba and E. J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error-detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 7, pp. 783–789, Jul. 1997.

[31] M. G. Karpovsky and A. Taubin, "A new class of nonlinear systematic error detecting codes," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1818–1820, Aug. 2004.

[32] J.-Y. Jou and J. A. Abraham, "Fault-tolerant FFT networks," *IEEE Trans. Comput.*, vol. 37, no. 5, pp. 548–561, May 1988.

[33] J. K. Wolf, A. Michelson, and A. H. Levesque, "On the probability of undetected error for linear block codes," *IEEE Trans. Commun.*, vol. 30, no. 2, pp. 317–325, Feb. 1982.

[34] K. M. Cheung, "The weight distribution and randomness of linear codes," TDA Progress, Jet Popul. Lab., Pasadena, CA, USA, Tech. Rep. 42-97, 1989.

[35] OpenCores. (2012). *OR1200 OpenRISC Processor*. [Online]. Available: http://opencores.org/or1k/OR1200_OpenRISC_Processor

[36] Nangate. (2011). *Nangate 45nm Open Cell Library*. [Online]. Available: http://www.nangate.com/?page_id=22

[37] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comp.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.

[38] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[39] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ICs using split fabrication," in *Proc. HOST*, Arlington, VA, USA, May 2014, pp. 1–6.

[40] K. Vaidyanathan *et al.*, "Efficient and secure intellectual property (IP) design with split fabrication," in *Proc. HOST*, Arlington, VA, USA, May 2014, pp. 13–18.

[41] K. Vaidyanathan, B. P. Das, and L. Pileggi, "Detecting reliability attacks during split fabrication using test-only BEOL stack," in *Proc. DAC*, San Francisco, CA, USA, Jun. 2014, pp. 1–6.

[42] J. Valamehr *et al.*, "A 3-D split-manufacturing approach to trustworthy system development," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 4, pp. 611–615, Apr. 2013.

[43] C. L. Yu *et al.*, "TSV process optimization for reduced device impact on 28nm CMOS," in *Proc. IEEE Symp. VLSI Technol.*, Honolulu, HI, USA, Jun. 2011, pp. 138–139.

[44] P. Song *et al.*, "MARVEL—Malicious alteration recognition and verification by emission of light," in *Proc. HOST*, San Diego, CA, USA, Jun. 2011, pp. 117–121.

[45] S. Takeshi *et al.*, "Reversing stealthy dopant-level circuits," in *Cryptographic Hardware and Embedded Systems*. Berlin, Germany: Springer, 2014, pp. 112–126.

[46] M. Bajura *et al.*, "Imaging integrated circuits with X-ray microscopy," in *Proc. 36th GOMACTech Conf.*, Orlando, FL, USA, Mar. 2011, pp. 1–4.

[47] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, Oct. 2010.

[48] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 66–75, Jan./Feb. 2010.

[49] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *Proc. IEEE Symp. Security Privacy (SP)*, Oakland, CA, USA, 2011, pp. 49–63.

[50] R. S. Chakraborty and S. Bhunia, "HARPOON: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[51] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan attacks using key-based design obfuscation," *J. Electron. Test.*, vol. 27, no. 6, pp. 767–785, Dec. 2011.

[52] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using Boolean functional analysis," in *Proc. ACM CCS*, Berlin, Germany, 2013, pp. 697–708.

[53] D. Sullivan, J. Biggers, G. Zhu, S. Zhang, and Y. Jin, "FIGHT-metric: Functional identification of gate-level hardware trustworthiness," in *Proc. DAC*, San Francisco, CA, USA, 2014, pp. 1–4.

[54] A. Waksman and S. Sethumadhavan, "Tamper evident microprocessors," in *Proc. IEEE Symp. Security Privacy (SP)*, Oakland, CA, USA, 2010, pp. 173–188.

[55] J. Backer, D. Hely, and R. Karri, "Reusing the IEEE 1500 design for test infrastructure for security monitoring of systems-on-chip," in *Proc. DFT*, Amsterdam, The Netherlands, 2014, pp. 52–56.

[56] M. Abramovici and P. Bradley, "Integrated circuit security: New threats and solutions," in *Proc. CSIIRW*, Knoxville, TN, USA, 2009, pp. 1–3.

[57] S. Dutt and L. Li, "Trust-based design and check of FPGA circuits using two-level randomized ECC structures," *J. ACM Trans. Reconfig. Technol. Syst.*, vol. 2, no. 1, Mar. 2009, Art. ID 6.

[58] S. Trimberger, "Trusted design in FPGAs," in *Proc. DAC*, San Diego, CA, USA, 2007, pp. 5–8.

[59] S. Mal-Sarkar, A. Krishna, A. Ghosh, and S. Bhunia, "Hardware Trojan attacks in FPGA devices: Threat analysis and effective countermeasures," in *Proc. GLSVLSI*, Houston, TX, USA, 2014, pp. 287–292.

[60] J. Li and J. Lach, "At-speed delay characterization for IC authentication and Trojan horse detection," in *Proc. HOST*, Anaheim, CA, USA, 2008, pp. 8–14.

[61] X. Wang, Y. Zheng, A. Basak, and S. Bhunia, "IIPS: Infrastructure IP for secure SoC design," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2226–2238, Aug. 2015.

[62] M. Li, A. Davoodi, and M. Tehranipoor, "A sensor-assisted self-authentication framework for hardware Trojan detection," in *Proc. DATE*, Dresden, Germany, 2012, pp. 1331–1336.

[63] S. Wei, K. Li, F. Koushanfar, and M. Potkonjak, "Provably complete hardware Trojan detection using test point insertion," in *Proc. ICCAD*, San Jose, CA, USA, 2012, pp. 569–576.

[64] S. Wei and M. Potkonjak, "Malicious circuitry detection using fast timing characterization via test points," in *Proc. HOST*, Austin, TX, USA, 2013, pp. 113–118.

[65] K. Xiao and M. Tehranipoor, "BISA: Built-in self-authentication for preventing hardware Trojan insertion," in *Proc. HOST*, Austin, TX, USA, 2013, pp. 45–50.

[66] K. Xiao, D. Forte, and M. Tehranipoor, "A novel built-in self-authentication technique to prevent inserting hardware Trojans," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 12, pp. 1778–1791, Dec. 2014.

[67] S. Narasimhan, X. Wang, D. Du, R. S. Chakraborty, and S. Bhunia, "TeSR: A robust temporal self-referencing approach for hardware Trojan detection," in *Proc. HOST*, San Diego, CA, USA, 2011, pp. 71–74.

[68] S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, and S. Bhunia, "Improving IC security against Trojan attacks through integration of security monitors," *IEEE Des. Test. Comput.*, vol. 29, no. 5, pp. 37–46, Oct. 2012.

[69] M. Banga and M. S. Hsiao, "A novel sustained vector technique for the detection of hardware Trojans," in *Proc. VLSI Design*, New Delhi, India, 2009, pp. 327–332.

[70] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware Trojan detection and reducing Trojan activation time," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 1, pp. 112–125, Jan. 2012.

[71] Y. Jin, D. Maliuk, and Y. Makris, "Post-deployment trust evaluation in wireless cryptographic ICs," in *Proc. DATE*, Dresden, Germany, 2012, pp. 965–970.

[72] Y. Jin and D. Sullivan, "Real-time trust evaluation in integrated circuits," in *Proc. DATE*, Dresden, Germany, 2014, pp. 1–6.

[73] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey, "Hardware Trojan horse detection using gate-level characterization," in *Proc. DAC*, San Francisco, CA, USA, 2009, pp. 688–693.

[74] S. Wei and M. Potkonjak, "Scalable hardware Trojan diagnosis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* vol. 20, no. 6, pp. 1049–1057, Jun. 2012.

[75] S. Wei, A. Nahapetian, M. Nelson, F. Koushanfar, and M. Potkonjak, "Gate characterization using singular value decomposition: Foundations and applications," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 765–773, Apr. 2012.

[76] F. Koushanfar and A. Mirhoseini, "A unified framework for multi-modal submodular integrated circuits Trojan detection," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 1, pp. 162–174, Mar. 2011.

[77] A. N. Nowroz, K. Hu, F. Koushanfar, and S. Reda, "Novel techniques for high-sensitivity hardware Trojan detection using thermal and power maps," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 12, pp. 1792–1805, Dec. 2014.

[78] K. Hu, A. N. Nowroz, S. Reda, and F. Koushanfar, "High-sensitivity hardware Trojan detection using multimodal characterization," in *Proc. DATE*, Grenoble, France, 2013, pp. 1271–1276.

[79] Y. Liu, K. Huang, and Y. Makris, "Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting," in *Proc. DAC*, San Francisco, CA, USA, 2014, pp. 1–6.

[80] C. Lamech, R. M. Rad, M. Tehranipoor, and J. Plusquellic, "An experimental analysis of power and delay signal-to-noise requirements for detecting Trojans and methods for achieving the required detection sensitivities," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 3, pp. 1170–1179, Sep. 2011.

[81] S. Jha and S. K. Jha, "Randomization based probabilistic approach to detect Trojan circuits," in *Proc. HASE*, Nanjing, China, 2008, pp. 117–124.

[82] D. Y. Deng, A. H. Chan, and G. E. Suh, "Hardware authentication leveraging performance limits in detailed simulations and emulations," in *Proc. DAC*, San Francisco, CA, USA, 2009, pp. 682–687.

[83] M. Banga and M. S. Hsiao, "ODETTE: A non-scan design-for-test methodology for Trojan detection in ICs," in *Proc. HOST*, San Diego, CA, USA, 2011, pp. 18–23.

[84] M. Banga and M. S. Hsiao, "VITAMIN: Voltage inversion technique to ascertain malicious insertions in ICs," in *Proc. HOST*, San Francisco, CA, USA, 2009, pp. 104–107.

[85] S. Mitra, H.-S. P. Wong, and S. Wong, "The Trojan-proof chip," *IEEE Spectr.*, vol. 52, no. 2, pp. 46–51, Feb. 2015.

[86] OEIS. (Jul. 5, 2014). *Number of Primitive Polynomials of Degree N Over GF(2)*. [Online]. Available: https://oeis.org/A011260

[87] B. Cakir and S. Malik, "Hardware Trojan detection for gate-level ICs using signal correlation based clustering," in *Proc. DATE*, Grenoble, France, 2015, pp. 471–476.

[88] J. L. Massey, "Guessing and entropy," in *Proc. ISIT*, Trondheim, Norway, 1994, p. 204.
[89] T. F. Wu *et al. TPAD: Hardware Trojan Prevention and Detection for Trusted Integrated Circuits*. [Online]. Available: http://www.arxiv.org
[90] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara, "Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation," in *Proc. USENIX Security*, Washington, DC, USA, 2013, pp. 495–510.

**Tony F. Wu** received the B.S. degree in electrical engineering from the California Institute of Technology, Pasadena, CA, USA, in 2011, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2013, where he is currently pursuing the Ph.D. degree.

His current research interests include design and fabrication of monolithic 3-D integrated systems using emerging technologies.

**Karthik Ganesan** (S'09) received the B.S. degree in electrical engineering and computer science and the B.A. degree in statistics from the University of California at Berkeley, Berkeley, CA, USA, in 2013, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2015, where he is currently pursuing the Ph.D. degree.

His current research interests include coding theory, applied probability, and their uses in reliable computing and low-power systems design.

**Yunqing Alexander Hu** (S'12) received the B.S. degree in electrical engineering from the California Institute of Technology, Pasadena, CA, USA, in 2012, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2014, where he is currently pursuing the Ph.D. degree.

His current research interests include design and fabrication of secure 3-D integrated systems using emerging technologies.

**H.-S. Philip Wong** (F'01) received the B.Sc. (Hons.) degree from the University of Hong Kong, Hong Kong, in 1982, the M.S. degree from Stony Brook University, Stony Brook, NY, USA, in 1983, and the Ph.D. degree from Lehigh University, Bethlehem, PA, USA, in 1988.

He is the Willard R. and Inez Kerr Bell Professor with the School of Engineering, Stanford University, CA, USA. In 2004, he joined Stanford University, Stanford, CA, USA, as a Professor of Electrical Engineering. From 1988 to 2004, he was with the International Business Machines (IBM) Thomas J. Watson Research Center, Yorktown Heights, NY, USA, where he held various positions from a Research Staff Member to the Manager, and the Senior Manager. While he was the Senior Manager, he had the responsibility of shaping and executing IBM's strategy on nanoscale science and technology as well as exploratory silicon devices and semiconductor technology. His academic appointments include the Chair of Excellence of the French Nanosciences Foundation, Grenoble, France, the Guest Professor with Peking University, Beijing, China, a Honorary Professor with the Institute of Microelectronics of the Chinese Academy of Sciences, Beijing, a Visiting Chair Professor of Nanoelectronics with the Hong Kong Polytechnic University, Hong Kong, and the Honorary Doctorate degree from the Institut Polytechnique de Grenoble, Grenoble. He is the Founding Faculty Co-Director of the Stanford SystemX Alliance, Stanford, an industrial affiliate program focused on building systems. His current research interests include carbon electronics, 2-D layered materials, wireless implantable biosensors, directed self-assembly, nanoelectromechanical relays, device modeling, brain-inspired computing, and nonvolatile memory devices such as phase change memory and metal oxide resistance change memory.

Prof. Wong served as the Editor-in-Chief of the IEEE TRANSACTIONS ON NANOTECHNOLOGY from 2005 to 2006, the Subcommittee Chair of the International Solid State Circuits Conference from 2003 to 2004, the General Chair of the International Electron Devices Meeting in 2007. He has been a member of the Executive Committee of the Symposia of Very-Large-Scale Integration Technology and Circuits since 2007. He served as an elected member of the Electron Devices Society AdCom from 2001 to 2006.

**Simon Wong** (F'99) received the bachelor's degrees in electrical engineering and mechanical engineering from the University of Minnesota, Minneapolis, MN, USA, in 1975 and 1976, respectively, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 1978 and 1983, respectively.

His industrial experience includes semiconductor memory design at National Semiconductor, Santa Clara, CA, USA, from 1978 to 1980, and semiconductor technology development at Hewlett Packard Laboratories, Palo Alto, CA, USA, from 1980 to 1985. He was an Assistant Professor with Cornell University, Ithaca, NY, USA, from 1985 to 1988. Since 1988, he has been with Stanford University, Stanford, CA, USA, where he is currently a Professor of Electrical Engineering. He is on the board of Pericom Semiconductor, Milpitas, CA, USA. His current research interests include understanding and overcoming the factors that limit performance in devices, interconnections, on-chip components, and packages.

**Subhasish Mitra** (F'13) was a Principal Engineer with Intel, Santa Clara, CA, USA. He is the Chambers Faculty Scholar of Engineering with the Robust Systems Group, Department of Electrical Engineering and the Department of Computer Science, Stanford University, Stanford, CA, USA. His X-Compact technique for test compression has been key to cost-effective manufacturing and high-quality testing of a vast majority of electronic systems, including numerous Intel products. X-Compact and its derivatives have been implemented in widely used commercial electronic design automation tools. His work on carbon nanotube imperfection-immune digital very large scale integration (VLSI), jointly with his students and collaborators, resulted in the demonstration of the first carbon nanotube computer, and it was featured on the cover of NATURE. The U.S. National Science Foundation presented this paper as a Research Highlight to the U.S. Congress, and it also was highlighted as "an important, scientific breakthrough" by the *BBC, Economist, EE Times, IEEE Spectrum, MIT Technology Review, National Public Radio, New York Times, Scientific American, Time, Wall Street Journal, Washington Post*, and numerous other organizations worldwide. He and his students published several award-winning papers at major venues such as the IEEE/ACM Design Automation Conference, the IEEE International Solid-State Circuits Conference, the IEEE International Test Conference, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN (CAD), the IEEE VLSI Test Symposium, Intel Design and Test Technology Conference, and the Symposium on VLSI Technology. His current research interests include robust systems, VLSI design, CAD, validation and test, emerging nanotechnologies, and emerging neuroscience applications.

Prof. Mitra was a recipient of the Presidential Early Career Award for Scientists and Engineers from the White House, the highest U.S. honor for early-career outstanding scientists and engineers, the ACM Special Interest Group on Design Automation/IEEE Council on Electronic Design Automation A. Richard Newton Technical Impact Award in electronic design automation, "a test of time honor" for an outstanding technical contribution, the Semiconductor Research Corporation's Technical Excellence Award, and the Intel Achievement Award, and the Intel's highest corporate honor. At Stanford, he has been honored several times by graduating seniors "for being important to them during their time at Stanford." He has served on numerous conference committees and journal editorial boards and also served as an Invited Member of Defense Advanced Research Projects Agency's Information Science and Technology Board. He is a fellow of ACM.