Received 24 February 2025, accepted 20 March 2025, date of publication 26 March 2025, date of current version 8 April 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3555166

SURVEY

A Survey on the Design, Detection, and Prevention of Pre-Silicon Hardware Trojans

JONATHAN CRUZ AND JASON HAMLET, (Senior Member, IEEE)
Sandia National Laboratories, Albuquerque, NM 87185, USA

Corresponding author: Jonathan Cruz (jwcruz@sandia.gov)

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under Contract DE-NA0003525.

ABSTRACT The complexity of the semiconductor design lifecycle and globalized manufacturing process creates concern over the threat of deliberate malicious alterations, or hardware Trojans, being inserted into microelectronic designs. This has resulted in a significant corpus of hardware Trojan research including Trojan design and benchmarking efforts and development of corresponding metrics and detection and prevention techniques, over the last two decades. In this survey, we first highlight efforts in Trojan design and benchmarking, followed by a cataloging of seminal and recent works in Trojan detection and prevention and their accompanied metrics. Given the volume of literature in this field, this survey considers only pre-silicon techniques. We make this distinction between pre- and post-silicon to properly scope and provide appropriate context into the capabilities of existing hardware Trojan literature. Each major section (design, prevention, and detection) is accompanied by insights, and common pitfalls, which we highlight can be addressed by future research.

INDEX TERMS Hardware Trojans, hardware Trojan benchmarking, hardware Trojan detection, hardware Trojan prevention.

I. INTRODUCTION

Trusted and assured design of integrated circuits (ICs) and hardware designs targeting field programmable gate arrays (FPGAs) has emerged as a significant concern to the microelectronics community, particularly given the globalized microelectronics supply chain and the widespread adoption of third party intellectual property (3PIP). In addition to concerns such as counterfeiting and overproduction, the threat of hardware Trojans has received considerable interest in recent decades, particularly following publication of a report describing what many speculate to have been a hardware Trojan placed within a radar system [1]. Hardware Trojans are stealthy, malicious alterations to a design, that aim to violate its integrity, confidentiality, or availability. The complexity and interdependence of the IP lifecycle fosters several threat scenarios in which Trojans can be inserted. From malicious insiders, malicious IP vendors to malicious CAD tools, Trojans can be inserted at nearly every point in the lifecycle. Therefore, it is important to develop solutions that can validate designs against Trojans and offer downstream protections.

The threat of unanticipated malicious alterations has received considerable attention, resulting in the study and development of several Trojan countermeasures which we broadly characterize as detection and prevention techniques. Trojan detection tackles the problem of identifying a potential Trojan that has been inserted by an untrusted or malicious adversary with access to intermediate design files during the IP lifecycle or from any infield manipulations. Many methodologies, including targeted test generation, formal methods, reverse engineering, and machine learning have been leveraged in the detection of hardware Trojans. Trojan prevention techniques discourage or altogether prevent the insertion of stealthy Trojans in a design. These approaches rely on design-level techniques to reduce the attractive Trojan space in a design or tolerate errant behavior during runtime. Figure 1 illustrates a breakdown of pre-silicon-level

The associate editor coordinating the review of this manuscript and approving it for publication was Somchart Fugkeaw.

Trojan countermeasures. Researchers have also furthered the attack space, devising new Trojan architectures, insertion mechanisms, and Trojan behaviors that further warrant more effective countermeasures.

To this end, this document surveys notable advances in the field of hardware Trojans, including hardware Trojan design and insertion, Trojan detection, and prevention techniques. While we introduce the entire Trojan threat space, we focus on attacks and solutions that are applicable pre-silicon. Unlike previous surveys in this area, we draw connections between attacks and defenses by including a discussion on a collection of novel Trojan architectures and connect them to existing countermeasures. We also provide insight into recent trends, potential pitfalls, and overall recommendations to address the threat of pre-silicon hardware Trojans.

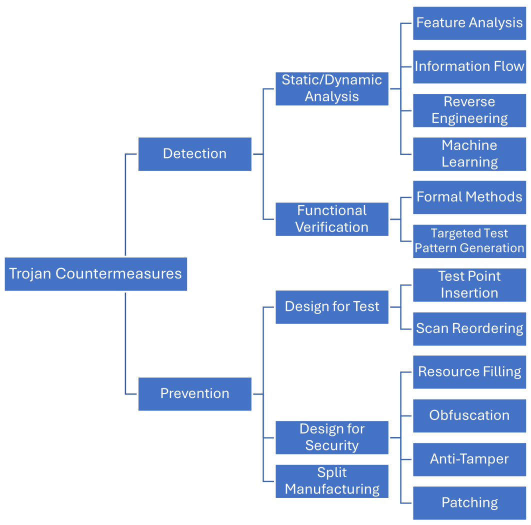

FIGURE 1. Pre-silicon hardware Trojan countermeasure taxonomy.

The rest of this survey is organized as follows: in Section II we describe hardware Trojans as consisting of an optional *trigger* (Section II-A) and a *payload* (Section II-B), each of which can be either combinational or sequential. Section II-C describes the threat model for Trojan insertion at various points in the hardware lifecycle, as well as selection of nodes in a circuit to target for constructing triggers or impacting with payloads. Next, Section II-D describes a variety of hardware Trojans that were designed either to target specific aspects of a platform or architecture or to bypass a particular Trojan detection mechanism. Section III describes a variety of tools for automatically inserting Trojans in netlist and RTL designs. Section IV presents a wealth of literature describing pre-silicon Trojan detection techniques. Approaches for preventing Trojan insertion are described in Section V, and the survey concludes in Section VI.

II. HARDWARE TROJAN ARCHITECTURES

Hardware Trojans can be introduced throughout the design process by malicious insiders, third party intellectual property (3PIP) vendors, and individuals at manufacturing and fabrication facilities [5], [6].

As shown in Figure 2 hardware Trojans consist of two parts: an optional *trigger* used to activate the Trojan and a *payload* that implements the Trojan functionality. When the triggering condition is met, the payload is activated. Fundamentally, hardware Trojans have two main objectives: to remain undetected until activated and to carry out some desired effect. Undetectability or stealthiness is governed by features that can be observed during standard verification and validation, which tend to fall under functional or parametric anomalies. A Trojan's functional stealthiness is often achieved through a trigger sub-circuit, which controls when to activate a payload sub-circuit that carries out the intended malicious effect. There are many ways to implement these Trojan sub-circuits. While traditional architectures exist, Trojan implementations are often left up to the creativity of the attacker so long as the two objectives are met.

A. TRIGGER ARCHITECTURES

As previously mentioned, the hardware Trojan trigger serves as the mechanism with which an attacker can exert fine-grained control over when the payload is deployed. A trigger often takes the form of a comparator, observing existing signals in the design such that when some rare or undefined signal pattern occurs, the Trojan becomes active. A trigger may also monitor side-channel phenomena, such as signal switching [7]. Such Trojans are out of scope for this survey. Nevertheless, the trigger can still contribute to parametric side-channel stealthiness directly by tapping low switching signals or through size (e.g., number of gates and/or number of signals tapped). The trigger circuit can also indirectly affect side-channels by controlling a parametric payload. We can broadly categorize trigger architectures by the components used to construct them: combinational and sequential logic.

1) COMBINATIONAL

Combinational trigger architectures, as the name suggests, consist purely of combinational logic. Therefore, the corresponding payload is only active while the trigger condition is maintained. These triggers may monitor internal signals, primary inputs to a circuit, or both. An attacker will typically tap hard-to-activate nets in order to achieve a target activation probability while maintaining a low side-channel footprint [8], [9]. A prototypical example is a Trojan targeting a circuit containing a cryptographic module when a specific plaintext is provided as input [10]. As the input space of potential plaintexts is often 128-bits or more such a Trojan is highly unlikely to be activated during simulation or testing, which is an important characteristic of Trojan triggers.

Some researchers have suggested integrating combinational triggers into processor architectures. For instance, [11] suggests monitoring the instruction decoder and the operand buses that connect the register file with the multiplier in a

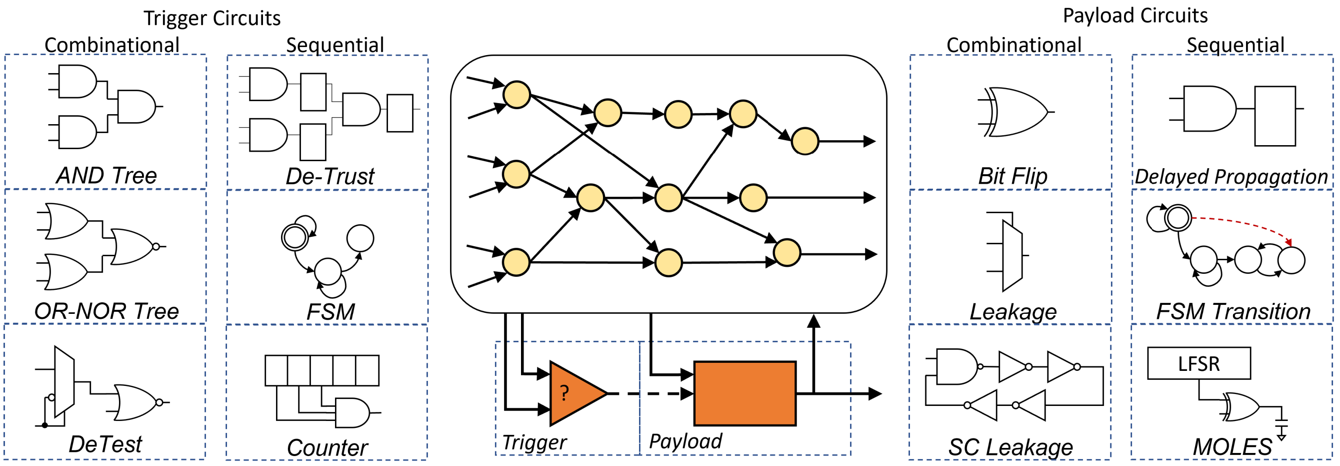

FIGURE 2. Example hardware Trojan architectures including DeTrust [2], DeTest [3], and MOLES [4] designs.

RISC processor core to activate a Trojan when particular multiply operations associated with cryptographic operations are detected.

The researchers in [12] introduce a Trojan that is triggered when the circuit reaches a defined temperature. This Trojan can be activated remotely by sending a large number of network requests to the target device, artificially raising its temperature [12].

2) SEQUENTIAL

Sequential triggers are Trojan logic that have a sense of state or incorporate such signals from the host design. Because of their design, sequential triggers can (generally) use fewer trigger inputs or tap signals of higher activation probabilities to achieve a final activation probability equivalent to that of their combinational counterparts [13]. Sequential triggers can take several forms. Counter based or timebomb triggers consist of a typically large counter that increments or decrements either on every clock cycle or when some other condition is met. When the counter reaches a specified value, the Trojan is activated. Finite state machine (FSM) triggers consist of a sequence of comparisons. The FSM transitions from one state to another when some set of signal values is observed. This type of trigger is essentially a sequence of combinational triggers [14], [15].

3) ALWAYS ON

Hardware Trojans that do not use a trigger circuit are referred to as *always on* Trojans. In such cases, the attacker forgoes functional stealthiness of the trigger and supplements it at the payload sub-circuit. We further discuss always-on payloads in Section II-B.

4) ACTIVATION PROBABILITY

One of the cornerstone features that is a measure of Trojan stealthiness is activation or signal probability. Activation probability refers to the likelihood that a wire or net will be activated to a specific value after applying input stimulus. Activation probability of a Trojan trigger, refers to the activation of all trigger nets to their rare values simultaneously. For combinational triggers, the activation probability of a set of triggers $R, r \in R$ is defined as $\prod_{i=0}^{|R|} p(r_i)$, with the assumption that all trigger inputs are independent and $p(x)$ as the function for extracting the signal probability of wire x. Signal probability can be calculated via simulation or by assuming some initial probability at the inputs and propagating them through the design. Sequential Trojan activation probability with a set of R triggers and m states can be calculated as $m \times \prod_{i=0}^{|R|} p(r_i)$.

Notably, the more nets one taps for a Trojan trigger, the lower the activation probability but the higher the footprint in terms of routing penalty, switching activity, and gate area [16]. Therefore an attacker is likely to prioritize selecting hard-to-activate nets to achieve a target activation probability with low overhead [9].

B. PAYLOAD ARCHITECTURES

The payload subcircuit is the manifestation of the malicious effect which, in broad terms, can be made to cause change in functionality, denial of service, leakage of sensitive or critical information, and degradation. Examples include the following: modifying microprocessor instructions, disabling a receiver, and leaking a cryptographic key, respectively [10]. Similar to the trigger, the payload architecture can be made of combinational or sequential gates. Though the trigger mostly contributes to functional stealthiness, the payload can be designed in such a way to have a delayed or muted effect adding to functional stealthiness [4], [8], [17].

1) COMBINATIONAL

A combinational payload is any payload design that includes only combinational elements. Due to the nature of their design, combinational payload circuits typically have effects that are immediately observed. For greater stealth, attackers crafting such payload structures ideally target hard-to-observe nets to mask their effect [8].

A variety of combinational payload structures have been proposed [5], [6], [10]. In [15] and [18], the authors suggest an inversion or bit-flip payload. This is implemented with an XOR gate that has one input from the trigger and one input from the targeted node. When the Trojan is triggered, the value of the targeted node is inverted. A second common payload replaces one signal in the design with another to cause functional failure or leak a design asset [19]. This is implemented with a simple 2:1 multiplexer (MUX) with the trigger used as the select input and the two signals as the other inputs. In [20], the authors create a stuck-at-1 fault with an OR gate. One input of the gate is held at logic '1' and the other input is the targeted signal. When the Trojan is enabled, the targeted signal is stuck-at-1. Additional payloads can be found in the extensive literature on this topic, which was summarized in a recent survey [6].

2) SEQUENTIAL

A sequential payload is any subcircuit that includes sequential elements in its design. These payloads can increase the time it takes to observe the Trojan effect, and allow for more complex functions, thereby increasing the difficulty of detection. Unlike combinational payloads, sequential payloads do not necessarily need to select hard-to-observe payload nets as the subcircuit architecture can contribute in this manner, such as in [4]. However, sequential payloads often come at the cost of larger area and power footprints.

From existing literature, sequential payloads have typically been employed to leak data off-chip. One example is the MOLES payload described in [4]. In this approach, a linear feedback shift register (LFSR) configured to generate a maximal length sequence is used as a pseudo-random number generator (PRNG). Select bits of this LFSR are XORed with the data to be leaked. The outputs of the XOR gates are passed through a capacitive load. This results in a small amount of power consumption on $0 \rightarrow 1$ transitions, resulting in a data-dependent power draw that an attacker with physical access to the chip can measure. A similar technique creates a temperature side-channel [21]. This is done by loading a data register into a shift-rotate register. The least-significant bit (LSB) of this register is the enable input to a ring oscillator. The output of the ring oscillator is connected to several output pins. When the ring oscillator is enabled the parasitic capacitances of these pins are rapidly charged and discharged, causing the temperature of the chip to increase and decrease in a data specific fashion.

The authors of [22] suggest some additional sequential payloads. The first of these targets encryption circuits by modifying their finite state machines to reduce the number of rounds during the encryption operation. This has the effect of weakening the cryptosystem. A second sequential payload targets cellular automata pseudo-random number generators (CA-PRNG) by replacing the output of the CA-PRNG with a selected value. Additional sequential payloads can be found in a recent Trojan design survey [6].

3) ALWAYS ON

As previously mentioned, always on Trojans consist entirely of the payload circuitry. Because the Trojan is always active, it is not practical to implement an always on Trojan that alters the functionality in any way as this will more easily show up during testing. Therefore, always on Trojans typically exist as side-channel leakage, such as the MOLES Trojan [4] or ring oscillator leakage Trojans. What these types of Trojans gain in functional stealthiness (no observable functional perturbations), they generally lack in footprint, with either large areas or high switching activities.

4) OBSERVABILITY

Observability is the measure of effort required to propagate a wire value to an observable point in the design, such as primary output, or scan-chain flip flop. Calculating observability depends on the distance away from and the functionality between the observable point and the net. For selecting payload nets, attackers will prioritize hard-to-observe nets such that, even under activation, the Trojan effect will not be immediately observed, increasing functional stealthiness [9], [23]. For example, bounded model checking is a formal verification approach for proving specifications or properties of a design under test within a specific number of cycles. Even if a Trojan is activated, the specified number of cycles may be insufficient for propagating the malicious effect, and the Trojan will remain undetected [24].

C. TROJAN IMPLEMENTATION AND PLACEMENT

Given the nature of the semiconductor supply chain, there are several points in the design life cycle at which an attacker can implement a hardware Trojan [5], [6]. In this section, we first describe the semiconductor IP design flow, followed by an introduction to the Trojan threat models. Then, we describe the different levels of abstractions for Trojan insertion and the pros and cons of each. Additionally, we further discuss Trojan placement that can be performed at the architectural level (e.g., Trojans in the control path versus Trojans in the data path).

1) SEMICONDUCTOR IP DESIGN FLOW

Figure 3 illustrates a sample design flow with example verification steps for both application specific integrated circuits (ASICs) and FPGAs. A design house may start with a high level design specification (spec) outlining expected behavior and functionality of their product. Design engineers will then implement this specification through hardware description languages (HDL) either directly using a behavioral register transfer level (RTL) description in a language such as VHDL or Verilog or with high-level

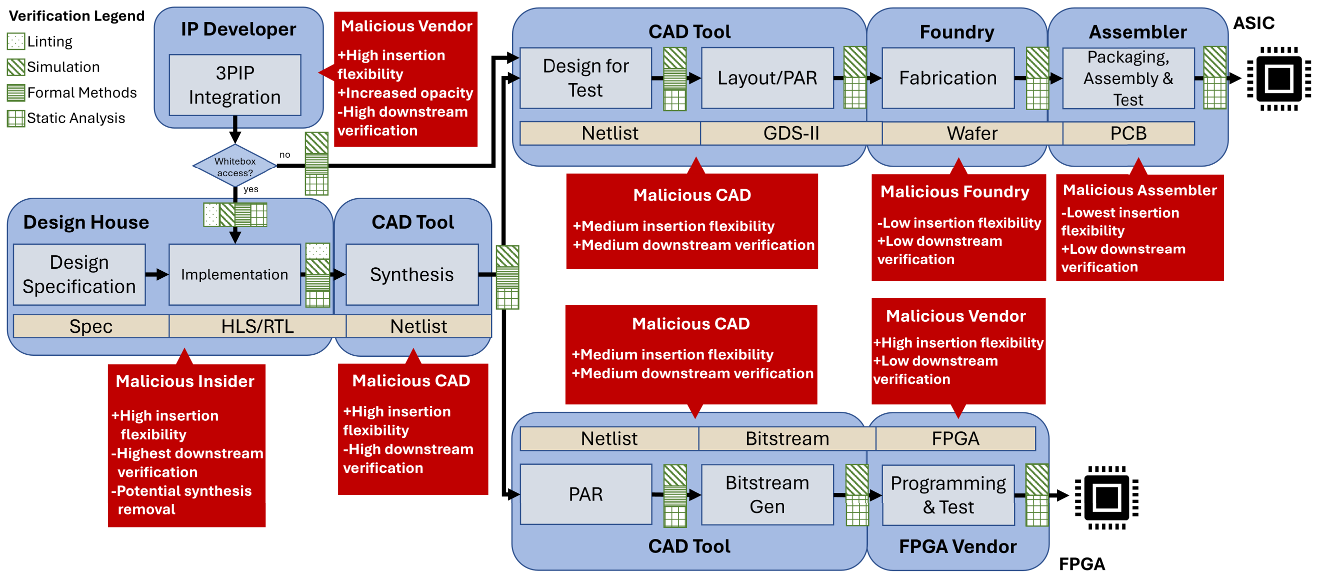
Verification Legend
Linting
Simulation
Formal Methods
Static Analysis
IP Developer
3PIP Integration
Malicious Vendor
+High insertion flexibility
+Increased opacity
-High downstream verification
Whitebox access?
no
yes
Design House
Design Specification
Implementation
Spec
HLS/RTL
CAD Tool
Synthesis
Netlist
Malicious Insider
+High insertion flexibility
-Highest downstream verification
-Potential synthesis removal
Malicious CAD
+High insertion flexibility
-High downstream verification
CAD Tool
Design for Test
Layout/PAR
Netlist
GDS-II
Foundry
Fabrication
Wafer
Assembler
Packaging, Assembly & Test
PCB
ASIC
Malicious CAD
+Medium insertion flexibility
+Medium downstream verification
Malicious Foundry
-Low insertion flexibility
+Low downstream verification
Malicious Assembler
-Lowest insertion flexibility
+Low downstream verification
Malicious CAD
+Medium insertion flexibility
+Medium downstream verification
Malicious Vendor
+High insertion flexibility
+Low downstream verification
Netlist
Bitstream
FPGA
PAR
Bitstream Gen
Programming & Test
CAD Tool
FPGA Vendor
FPGA

FIGURE 3. Sample semiconductor IP design flow with example verification steps.

languages, such as C or SystemC, in combination with a high-level synthesis (HLS) tool. The design files are provided to computer aided design (CAD) tools for electronic design, also referred to as electronic design automation (EDA). The CAD tools generate downstream design files used to fabricate or program the final product. Throughout the design process, verification is performed to ensure correctness is maintained as shown by the verification boxes between the major design stages of Figure 3.

For the ASIC flow, the CAD tool synthesizes the RTL into a netlist representation, a design format also at the HDL-level, but described through primitive logic gates available to the ASIC. Optionally, the CAD tool can augment the design with design for test (DfT) infrastructure to aid in downstream test and debug. Finally, the CAD tool generates the layout in GDS-II format, which is provided to a foundry for fabrication. An assembler packages the wafer produced from the foundry and potentially places them on a printed circuit board (PCB). All formats described up until the generation of the wafer are referred to as pre-silicon.

For designs targeting FPGAs, CAD tools also synthesize the RTL to a netlist, but using primitive logic available to the target FPGA. This logic then goes through place-and-route (PAR), a process that maps the primitives expressed in the netlist to locations available in the physical FPGA and connects them together. Finally, the CAD tool generates an output format for programming the FPGA to behave as designed.

2) TROJAN THREAT MODELS

The complexity of the semiconductor IP design flow gives ample opportunity for an adversary to inject a hardware Trojan. We describe the different threats illustrated in Figure 3 that can exist from the perspective of a trusted design house.

a: MALICIOUS INSIDER

A malicious or rogue insider is an adversary that is an employee within an organization. Under this threat model, the adversary has access to soft IP, in-house verification flow, and other intimate design information. While malicious insiders have the most flexibility for insertion, they may encounter the most hurdles as they potentially have to bypass the entire verification flow to be successful [5], [6].

b: MALICIOUS IP DEVELOPER

Malicious IP developers can insert hardware Trojans within the provided IP. These developers will, generally, not have access to the design house other than potential interfaces; therefore, the scope of the Trojan attack is limited to their IP [5], [6]. Third-party designers will often encrypt their IP to protect the confidentiality of their assets preventing whitebox access, which can further hinder design analysis and aid in Trojan stealthiness.

c: MALICIOUS CAD TOOL

Given the reliance on third-party CAD, malicious CAD tools have the potential to insert Trojans during the design and verification flows [25], [26]. The CAD tools are also provided with intimate design information, often including design constraints and the design itself, with which to conduct a fine-grained attack. Similar to malicious insiders and IP developers, the CAD tool is at the stage where several levels of verification are performed, which increases the chances of detection. The authors in [26] outline a broad level of attacks available from CAD tools.

d: MALICIOUS FOUNDRY

The foundry is provided GDS-II representation of the design in order to fabricate the chip onto silicon wafers. With this information, the foundry can modify the design to insert a Trojan [5], [6]. GDS-II is a rather firm representation as it is right before fabrication. This limits the flexibility available to the adversary in the form of space available or constraints imposed; however, post-silicon verification does not have the same level of transparency available at pre-silicon, making it harder for inserted Trojans to be detected [27].

e: MALICIOUS ASSEMBLER/TESTING

Assemblers, after receiving wafers from the foundry, package and assemble them onto a PCB and run tests. A malicious assembler has access to the physical hardware and test patterns provided by the design house to check for correct function [5], [6]. Assemblers can also use the testing information to help guide the Trojan insertion process [28].

f: MALICIOUS FPGA VENDOR

The malicious FPGA vendor offers the FPGA hardware and is therefore similar to both malicious foundry and assembler in access to material [5], [29]. An adversary at this stage can insert a Trojan into the hardware itself, independent of the design programmed to the device [29].

3) DESIGN-LEVEL PLACEMENT

IP exists in different representations or abstractions as it proceeds through the design flow. Trojans can be placed during any of these phases. However, when the insertion occurs and at what abstraction affects not only the ease of Trojan insertion but also detection. In sections II-C3a-II-C3e, we discuss published Trojans at different design abstractions and the types of verification that each may encounter.

a: HIGH LEVEL SYNTHESIS

High-level synthesis (HLS) refers to the process of using higher-level languages (such as C/C++) than those typically used to design hardware. Trojan insertion at HLS may manifest as a malicious CAD tool that inserts Trojan logic in the intermediate representation that is then translated into RTL [30]. From an attacker perspective, it is beneficial to insert a Trojan as early as possible in the design flow to mitigate detection via formal equivalence checking between downstream representations since if the golden is compromised, formal equivalence checking will not be able to detect such Trojans. Furthermore, RTL code generated from HLS is not as human-readable as designer generated RTL code, which can obfuscate the Trojan logic.

b: REGISTER-TRANSFER LEVEL

Trojans inserted at Register-Transfer Level (RTL) are generally more flexible than those at other levels of abstraction as the Trojan design can be described behaviorally with a few lines of code. However, with added flexibility and ease of insertion comes ease of detection. RTL Trojans are subject to detection via human inspection and code or branch coverage metrics. This can be mitigated to some extent by writing Trojan logic in an obfuscated manner or with IP encryption [31]. Encrypted IP makes it much more difficult for a defender to analyze the IP for any malicious implants. Nevertheless, Trojans inserted at the beginning stages of the IP lifecycle are exposed to the highest amount of downstream verification.

An important aspect of RTL Trojans is the interaction with the synthesis tool. With pre-synthesis insertion, the synthesis tool can apply optimizations such that the Trojan logic and original design become tightly coupled, such as through resource sharing, and become harder to identify and remove [25], [32]. Though with some Trojan designs, such as those that do not trigger functionally or have no observable effect [17]), the synthesis tool may remove the Trojan logic. Attackers also have less control over the structure of the synthesized Trojan.

c: GATE-LEVEL

Gate-level Trojan insertion generally occurs after synthesis or when a netlist representation of the design is available. Trojan insertion at gate-level is still at a semi-flexible state of the design phase where design passes can be performed to meet design constraints. An attacker will generally insert the Trojan as a few extra logic gates that activate under hard-to-activate conditions. However, an attacker must be conscious of design constraints such as ensuring low switching activity for the Trojan logic and not introducing critical path delays [5], [6].

Due to the nature of their construction, gate-level Trojans enjoy inherent stealthiness due to the phenomena known as the "sea of gates." Identifying gate-level Trojans in larger designs is extremely difficult as these Trojans are made up of the same elements as the underlying circuit. Therefore, if 3PIP is received as a gate-level netlist, some level of reverse engineering [33], [34] to understand the netlist functionality is typically required. We also note that gate-level Trojans have seen the highest diversity of implementation across all abstractions. This can be attributed to the number of relevant Trojan functional and structural features, such as signal probability and distance from outputs, available for calculation at the gate-level (Table 6) and to general research attention (Table 1).

d: LAYOUT-LEVEL

Trojans can also be inserted at the layout-level, either in GDS-II files or directly in masks. The primary threat actor for this type of Trojan is an untrusted foundry [5], [6]. Trojans inserted at this level are less flexible than other types of Trojans due to the lower level of abstraction, need for physical space and available routing channels, difficulty of understanding the design, and the need to ensure that the Trojan does not prevent the circuit from operating correctly or result in yield failures. Trojans at this stage are discussed in terms of how they are inserted, rather than their individual components or effect. Engineering change orders (ECO) have been discussed as a practical way to implement a Trojan as a malicious foundry [35].

e: COMPONENT-LEVEL

Component-level Trojans are those implanted on a manufactured printed circuit board (PCB). In this case, malicious components such as capacitors [7], transistors [36], or ICs [37] are added to the board. Trojan footprint is of utmost importance at this stage, as the physical area and routing are limited. It is common to select few components of small size to add to the PCB [36]. Component-level Trojans can be further masked via package mismarking or marking removal [38]. PCB-level Trojan are out of scope for additional consideration in this survey, though examples are available [36], [39].

4) ARCHITECTURAL-LEVEL PLACEMENT

Trojans can inhabit different portions of the design. Here we briefly introduce Trojans in the controller, datapath, bus, and memory and discuss published examples of them.

a: CONTROL VS DATAPATH

At a high level, designs are generally implemented with a datapath and an FSM controller, so we distinguish between Trojans inserted in the control or datapath portions of a design. Control path Trojans generally tap signals from the FSM or alter the FSM to introduce unwanted transitions [40], [41]. Datapath Trojans are generally the prototypical denial-of-service or leakage-based Trojans that alter the datapath such as by flipping a bit or leaking an asset [10], [15].

b: COMMUNICATION BUS

Communications busses are attractive targets for Trojan insertion due to their role in transmitting data among IP. A hardware Trojan technique that utilizes incompletely specified behavior of buses to permit covert communication between peripherals on the bus is introduced in [42]. Bus protocols typically have control signals to indicate when valid bus transactions are underway, with control and data signals often unspecified during idle cycles. This unspecified behavior during idle cycles is used to implement a covert Trojan communication channel between modules connected to the bus. The covert channel can be one-way or bidirectional, and between any two modules on the bus. The specifics on the implementation are bus protocol dependent, but in all cases include re-using the existing data and control signals to define and communicate Trojan transactions during bus idle cycles [42].

c: MEMORY

Existing literature discusses how Trojans can also be placed in the physical memory of a device, such as in the static random access memory (SRAM) [43]. Such Trojans can induce faulty behavior in the SRAM are including resistive shorts, bridges, and opens in the SRAM array. The authors of [43] propose a method for an untrusted foundry to insert Trojans in SRAM. The SRAM is typically tested using *march tests*, which induce certain patterns in the SRAM array. To avoid detection by such tests, the trigger condition for this Trojan is a condition that is impossible for standard march tests to produce. Trojans within CPU cache memories have also been investigated, including to leak data [44] or disrupt cache coherence [45].

D. ADVANCED TROJAN ARCHITECTURES

Beyond different abstractions or placement, Trojan designs have evolved in parallel with Trojan countermeasures. Below, we describe Trojan architectures that were developed to thwart standard verification, specific Trojan countermeasures, or to take advantage of some design or platform artifact.

1) TARGETING STANDARD VERIFICATION

Many Trojans are inserted such that they remain undetected during standard verification. More advanced Trojans have been developed which aim to bypass more robust verification efforts or that use verification heuristics in their favor to remain undetected [3], [17], [41], [46], [47], [48].

a: SATISFIABILITY AND OBSERVABILITY DON'T CARE TROJANS

The authors of [17] present Trojans that make use of internal satisfiability don't care (SDC) and observability don't care (ODC) conditions. SDCs occur when an internal node has input conditions that cannot functionally occur. ODCs occur when an internal node has input conditions under which the output of a node does not influence any primary output, which means that it does not matter if the value of the node is logical '0' or '1'. A fault event such as voltage or clock glitching is required to activate these Trojans, so during normal operation these Trojans are invisible. However, they are susceptible to removal during synthesis as this type of logic is seen as unnecessary by optimizers [17].

b: 'X' DON'T CARE TROJANS

Don't care Trojans refer to malicious logic that leverages the use of explicit or implicit don't care ('X') values in behavioral logic [46]. Explicit don't cares refer to the assignment of 'X' to parts or all of a variable. Attackers can hide Trojan logic in this unspecified behavior. Implicit don't cares result from under-specified behavior, such as a default statement in a case statement. In an FSM, this can result in don't care states and transitions that an attacker can use to mount an attack [41], [47].

The Toggle MUX is a hardware Trojan trigger design that leverages X-optimistic behavior of simulators to insert malicious logic [48]. Toggle MUX constructs a multiplexer with an if-else statement with the select line driven by 'X'.

The Verilog languages standard dictates the else branch to be executed in the event an 'X' is in the conditional. Therefore, if such logic is present in an X-optimistic simulator, the attacker can make the Trojan logic inactive during simulation by forcing constant 0 in the else branch. To activate the Trojan in-field, a toggle circuit is added to the MUX select line to prevent any guess on the actual resolved value of 'X' and a rectifier circuit is added to the MUX output to force the trigger on.

c: DeTest

The authors in [3] note that most Trojan architectures lend themselves to high controllability (hard-to-activate) and high observability (hard-to-observe), which are features defenders often use for detection purposes. Therefore, the goal of the DeTest Trojan is to decrease the controllability of the trigger and the observability of the payload while still maintaining resilience to logic testing. For trigger controllability, the authors suggest adding a MUX, with the select signal the same as the same payload victim signal. This allows the Trojan to trigger, but the payload effect is suppressed. The authors suggest using a test-enable pin as the select signal if it exists, but this also requires an extra MUX at the output to ensure the payload effect is suppressed.

To reduce observability, a similar modification is suggested. The payload circuit output signal is MUX'd with a benign signal and routed back to the benign signal so that when the payload circuit output value matches the benign signal, its value is selected to propagate. Now the payload circuit output signal, which originally is hard to observe, has cases where it can propagate to an output with low effort but with no measureable effect.

2) TARGETING TROJAN COUNTERMEASURES

The development of more effective Trojan detection approaches was quickly followed by investigation into Trojan design approaches specifically developed to evade detection by these techniques [2], [4], [31], [49], [50], [51]. As is typical in the security field, the introduction of these more sophisticated Trojans has been met with more advanced countermeasures. This trend is likely to continue, although we caution that research tends to focus on detecting a specific type of Trojan or evading a specific countermeasure. Approaches that do not consider the broader ecosystem of Trojans and countermeasures may have shortcomings at detecting or evading other techniques.

a: DeTrust

DeTrust [2] is a collection of trigger design techniques developed to evade detection by unused circuit identification [52], FANCI [53], and VeriTrust [54], which are Trojan detection techniques described in Sections IV-A1a and IV-A1b.

First, the hardware Trojan is implemented using a specific coding style suggested in [31]. These techniques consist of breaking up comparisons and counters into sequences of sub-comparisons and sub-counters, and by integrating the trigger and payload logic into the circuit using logical expressions rather than "if-else" or "case" statements.

FANCI identifies suspicious logic by identifying combinational logic that has weakly affecting inputs, meaning that the output is rarely changed in response to a change in some of the input values. To avoid detection by FANCI, the DeTrust approach ensures that the control values for all Trojan-related signals are comparable to those for signals implementing the benign functionality of the design. This is accomplished by spreading the trigger condition across multiple sequential levels. For instance, a 64-bit trigger might be split into 16 separate 4-bit triggers, and then these values combined four at a time across several sequential layers. The control values for each of these combinations are much larger than in the straightforward implementation, preventing them from being detected by FANCI.

VeriTrust identifies Trojan triggers by finding redundant input terms to the Boolean equation for a wire. To avoid detection by VeriTrust, DeTrust ensures that Trojan payloads are driven only by non-redundant inputs when the Trojan is not triggered.

Blinding HT [49] is a recent Trojan design technique developed to avoid detection by FANCI by ensuring that the control values for the trigger signals never become too small, and to avoid detection by VeriTrust by ensuring that the trigger signals are not redundant in the logic cone of the targeted payload wire. In Blinding-HT the trigger is a large set of signals within the design that have small switching probabilities. These signals are used to initialize a shift register. A small number of consecutive shift register bits are combined with a Boolean function and used as input to a MUX. When the trigger value for these bits is obtained, the MUX increments a counter and shifts the register. Two circuits of this form are used to generate two trigger signals. These trigger signals are then used in tandem to target a wire within the design. The method of impacting this wire is chosen to ensure that the trigger signals are not redundant to the functionality of the targeted wire.

b: De-UCI

Several techniques have been suggested for avoiding detection by unused circuit identification (UCI), which is discussed in Section IV-A1b. Briefly, the UCI technique finds signal pairs (s, t) where s and t are separated by a logic cloud and for which s carries the same value as t for every test case. This situation indicates that logic between s and t can be replaced with a wire without causing any of the tests to fail. In this scenario the logic between s and t is flagged as potentially malicious.

Several methods for avoiding detection by this approach have been suggested. The researchers in [50] suggest ensuring that, for all pairs (s, t) with malicious logic in the intervening logic cloud, there is at least one input that causes $s \neq t$ and that does not trigger the Trojan. This ensures that UCI will not flag the Trojan as malicious.

The authors of [31] suggest code writing styles to evade detection. In particular, breaking up large comparisons into a set of smaller comparisons allows for the smaller comparisons to be more easily activated. However, the larger Trojan comparison still remains difficult to activate. The second approach replaces if-then constructs with equivalent implementations in logic gates.

c: MOLES

The Malicious Off-Chip Leakage Enabled by Side-Channels (MOLES) circuit [4] is an always-on side-channel leakage Trojan designed to leak an asset over a period of time through code division multiple access (CDMA). MOLES includes a linear feedback shift register (LFSR) as a pseudo random number generator (PRNG) to generate numbers with which to XOR the asset to be leaked over a targeted side channel. The side-channel traces can then be demodulated with knowledge of the LFSR sequence to recover the key.

d: k-XOR-LFSR

The k-XOR-LFSR Trojan is a counter-based design with the counter implemented as a $k - bit$ LFSR [51]. A carefully selected subset of j bits of this LFSR are then XORed with j signals from the target design. The bits of the LFSR included in the XOR are selected so that the output of the XOR tree is constant for $u - 1$ clock cycles, and then inverted on the u^{th} clock cycle, where $u \approx k$. To increase the number of cycles prior to Trojan activation without making the LFSR too large, the LFSR itself can be driven by a slower clock than the target design. This is easily implemented using frequency dividers.

3) TARGETING PLATFORMS

Oftentimes, the Trojan structure is described in a generic format that users plug-and-play into different designs. Trojans specific to a design may not be as flexible as generic Trojans, but they are potentially more difficult to detect. Trojans that target a platform are specialized architectures that leverage a platform specific artifact or attribute to achieve higher obscurity and functional stealthiness than would otherwise be available from a generic template [25], [32], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64], [65], [66], [67].

a: HARDWARE OBFUSCATION TROJANS

Tampering Attack on Any Locked (TAAL) circuit is a targeted Trojan attack for logic locking based IP protection solutions [55]. The authors assume the attacker can identify the key-inputs of a locked netlist. With this information, TAAL-based attacks leak the key either directly or through logic. TAAL based Trojans follow generic Trojan architectures.

Trojans based on Logic Locking (TroLL) use functionality that exists for logic-locking approaches to also carry out a malicious effect [56]. These Trojans are implemented by modifying and reusing the modification and restoration units of a subset of targeted locking artifacts in a deployed logic locking solution.

b: FPGA TROJANS

A Trojan technique that targets FPGA designs is presented in [25]. The Trojans are inserted at design time by monitoring the synthesis process for an IP core for which a Trojan has been pre-designed. When the target core is identified, a malicious look-up table (LUT) is added to the design. At this point, the payload signals are not propagated into the design, so the Trojan is inactive and appears not to impact the functionality of the design. Later, during the bitstream-generation process, the synthesis tool identifies the malicious LUTs and allows the payload to propagate. This is an always-on Trojan.

Another FPGA-specific Trojan is discussed in [57] and expanded upon in [32]. Here, Trojan logic is inserted in the unused space in an under-utilized resource, termed FPGA dark silicon. The attacker routes the Trojan trigger nets to the unused space and modifies the underlying functionality of the truth table to implement the Trojan logic.

c: MICROPROCESSOR TROJANS

There have been several Trojan architectures targeted at microprocessor designs. While the Trojan architecture is not novel, the payload effects are customized to the microprocessor platform and include attacks such as leakage via memory write duplication [58], and privilege escalation [59]. HarTBleed [60] is a notable CPU-targeted Trojan architecture that aims to disrupt or leak sensitive memory contents. The Trojan is triggered in a capacitor style, monitoring cache access and data. Several processor-based Trojans were developed in [61]. Processor Trojans in an SoC format are available on Trust-Hub [62].

d: NEURAL NETWORK TROJAN

A hardware Trojan architecture for neural network implementations is proposed in [68] and [69]. The authors adopt traditional Trojan architectures and apply them to specific neural network components. For example, a combinational trigger can tap into a set of hard-to-achieve neural states and the payload can modify the activation function of one or more neurons. The authors suggest it is more feasible to insert a Trojan toward the output layers due to the inherent robustness of neural networks against random noise. However, the trigger condition must be sufficiently stealthy so as to remain undetected during normal operation.

e: TrojanZero

TrojanZero is a design and insertion technique to achieve zero area and footprint Trojans in ASICs [63]. Gates with activity close to 0 or 1 are replaced with constants to potentially gain area/power slack for Trojan insertion. If there is no effect on the provided test set, the removal is kept. The Trojan designer can then iteratively insert the Trojan into the design until the original area and power parameters are met.

4) APPROXIMATE COMPUTING TROJANS

Researchers have proposed inserting hardware Trojans in approximate computing applications to mask the observability [64]. With approximate computing, there exists an acceptable level of error in the output in exchange for reduced area and power. An example Trojan inserted in approximate computing architectures is Malicious Approximate Accelerator Synthesis (MAAS), which inserts Trojans during the approximate circuit synthesis such that the malicious effect is masked within the accepted error tolerance [65].

a: HARDWARE TRIGGER SOFTWARE PAYLOAD TROJANS

The concept of altering software with hardware Trojans is discussed in [66]. The Soft-HaT Trojan is fabrication-level attack that adds a programming circuit and efuse to create a multi-stage trigger. The traditional trigger, which the authors argue can use non-rare nets, is only connected once the efuse has been programmed, thereby avoiding accidental activation during functional testing. Several software payloads are discussed, such as bypassing address restrictions in the memory management unit and disrupting the program counter. While Soft-HaT is a foundry-level attack and out of scope for this survey, the merit of software payloads as a paradigm is worth mentioning.

The Jinn Trojan was designed to circumvent security protections available at the software level [67]. Fundamentally, both the trigger and payload reside in hardware, yet the effect of the payload surfaces as a bypass of security critical software checks such as bounds checking for array access and branch misprediction. To correctly time the effect of the payload, the Trojan trigger is selected to monitor memory addresses or data flow to identify bounds check instructions.

5) PERSPECTIVE AND OUTLOOK

Fundamentally, Trojans are designed to subvert common conventions, assumptions, or verification flows to remain stealthy. From the above architectures, we see these intentions being applied across different levels of abstraction, and under more vigorous defenses and verification efforts than basic Trojans to achieve this stealthiness goal. We note an interesting trend of hiding Trojans in under-utilized components such as partially used LUTs or logic locking structures. Other Trojans, such as those with software payloads, present a new and interesting way to look at possible payloads and open the door for more robust defenses and analysis flows. Table 1 summarizes the advanced architectures discussed in this survey.

III. AUTOMATED TROJAN INSERTION

Designing hardware Trojans can be a difficult, time-consuming process. Due to this, in the early days of Trojan research, Trojan benchmarking was limited in scope with only a few Trojan architectures and host designs available. Automated Trojan insertion was introduced to alleviate these problems and allow for scalable benchmark creation and Trojan attack exploration. In general, automated Trojan insertion works by identifying potential trigger or payload locations of a design-under-test and then inserting user-directed Trojans. In this way, defenders can evaluate the Trojan attack space for their design, without relying on the presence of publicly available Trojanized benchmarks or evaluations. We will now discuss several relevant automated Trojan insertion frameworks.

A. TAINT

The Tool for Automated Insertion of Trojans (TAINT) is a semi-automated tool for Trojan insertion in RTL and FPGA netlists [19]. The payloads inserted by TAINT consist of a MUX that replaces one signal in the design with another when some triggering condition is met. TAINT can sometimes implement these Trojans without adding any additional hardware to the design by modifying the look-up tables already present in the target circuit. TAINT is semi-automated in that it presents the user with a front-end tool and guides the user through the Trojan insertion process. The trigger condition can be selected directly from the netlist, or a Boolean equation for the trigger can be provided to the tool. Combinational comparator and sequential counter based triggers are supported. TAINT provides a database of RTL Trojans drawn from Trust-Hub [97] and the Embedded Security Challenge [98], and users can add additional Trojans to this database. The tool will suggest locations in the design to place these Trojans, or the user can choose a location. Multiple Trojans can be inserted into the design. Finally, the tool outputs the Trojanized design either as VHDL, Xilinx Design Language (XDL), or a Native Circuit Description (NCD) netlist.

B. TRIT

Cruz et al. proposed a Trojan Insertion Tool (TRIT) to automatically insert Trojans into gate-level netlists [15]. The TRIT flow first converts the netlist into a hyper-graph representation and then performs logic simulation to estimate the signal probability for each wire. TRIT then identifies a set of candidate wires by selecting nets with a signal probability less than some user-defined threshold. With this information, TRIT automatically constructs a set of valid Trojan trigger conditions and potential payloads based on user configurations, such as the desired number of trigger nets. Valid trigger conditions refers to a set of nets from the candidate net population that can simultaneously achieve their rare values. Valid payloads are nets that will not cause a combinational loop when connected to a validated trigger set and that will have an observable effect. Finally, TRIT constructs the Trojan body, connects the trigger and payload nets, and inserts the Trojan into the netlist. TRIT has the ability to construct several different types of Trojan effects, including functional denial of service, leakage, and always

TABLE 1. Comparison of advanced trojan architectures.

Trojan	Implementation	Trigger	Payload	Strategy	Detection Method
DeTrust [2], [49]	ASIC/FPGA	Multi-stage	–	Break up trigger logic via multiple sequential levels	[51], [70]–[75]
De-UCI [50]	ASIC/FPGA	Custom comb.	Functional change	Defeat UCI pair flagging	–
DeTest [3]	ASIC/FPGA	MUX based	Functional change & Leakage (MUX based)	Lower testability of Trojan with cover signals	[3]*
MOLES [4]	ASIC/FPGA	Always-on LFSR	Leakage	Leak asset through CDMA side channel	[24], [33], [70]–[72], [76]–[81] [74], [75], [82]–[92]
k-XOR-LFSR [51]	ASIC/FPGA	Always-on LFSR	Functional change & leakage	LFSR-based counter	[51], [73]
SDC / ODC [17]	ASIC/FPGA	Generic comb.	–	Trigger conditions invalid, Payload effect not observable	[93]
RTL Don't Care [46], [47]	RTL	Generic comb.	Leakage	Utilize unspecified behavior in dangerous don't care to leak info	[41], [46], [47], [94]
Toggle MUX [48]	FPGA	MUX based	–	Use X as trigger to bypass functional validation	[48]*
Malicious LUT [25]	FPGA	Unary LUTs	–	Trojan activated during bitstream programming	[25]*
Dark Silicon [32]	FPGA	Generic comb.	Functional change & Leakage	Implement Trojan logic in partially utilized LUTs	[81]
HarTBleed [60]	μP	Multi-stage	DoS, Functional change, Leakage	Capacitor style monitoring cache data & access	[60]*
Neural Network [68], [69]	ASIC/FPGA	Generic comb.& seq.	DoS, Functional change	Modify activation operation	[68]*
A2 [7]	ASIC	Multi-stage	–	Timing based trigger via capacitor	[95], [96]
TrojanZero [63]	ASIC	–	–	Modify design to create parametric room for insertion	–
TAAL [55]	ASIC/FPGA	Generic comb.& seq.	Leakage	Identify key inputs and leak	–
TroLL [56]	ASIC/FPGA	Generic comb.	Functional change	Re-use locking hardware	[56]*
Soft-HaT [66]	ASIC	Multi-stage.	DoS, Functional change, Leakage	Efuse & programming circuit; SW payloads	–
Jinn [67]	ASIC/FPGA	Multi-stage seq.	Functional change	Trigger monitors memory & Bypass SW security measures	[67]*
MAAS [65]	ASIC/FPGA	Generic comb.	Functional change	Payload introduces error within tolerance	–

* refers to discussion of mitigations

on Trojans, and architectures. TRIT supports combinational and sequential Trojans, and a large set of TRIT-generated benchmarks are available online [97].

An important conclusion from their work highlighted the importance of the presence of Trojan and benchmark diversity for evaluating hardware Trojan detection schemes. The authors in [99] propose a similar flow to TRIT, but use transition probability estimation instead of signal probability.

C. TRIT-DS

TRIT-DS is an automated Trojan insertion tool for FPGA netlists [32]. TRIT-DS expands upon TRIT to include FPGA-specific paradigms and inserts Trojans into under-utilized resources or FPGA dark silicon (DS). TRIT-DS offers two variants of DS Trojans: 1) monolithic and 2) distributed dark silicon. Monolithic DS Trojans modify a single LUT to include Trojan logic, whereas distributed DS Trojans split the Trojan logic across multiple under-utilized LUTs. Because DS Trojans are inserted into already used resources, they enjoy reduced area and power overheads as compared to traditional ASIC Trojans. Moreover, the TRIT-DS framework offers timing-aware insertion to minimize any impact on delay.

The authors showed that DS Trojans can evade Trojan detection techniques that are successful in detecting ASIC implementations of the Trojan. Specifically, the machine learning technique [81] is shown to be around 40% less accurate in detecting DS-style Trojans under a zero-day attack model.

D. MIMIC

Machine Intelligence based Malicious Implant Creation (MIMIC) is an automated Trojan insertion tool that utilizes machine learning (ML) to identify suitable places to insert Trojan templates [100]. The approach proposes three separate ML models for Trojan insertion: a trigger model, a payload model, and a Trojan model. The trigger and payload model are used to identify trigger and payload nets, respectively. They are trained on *net features* extracted from a given set of trigger and payload nets from example Trojans. The set of trigger and payload nets are validated to ensure that the triggers can simultaneously reach their rare values and that the corresponding payload does not form a combinational loop. MIMIC then constructs a virtual Trojan by binding the trigger and payload nets to the provided template. Finally, the Trojan model is used to identify the most fit virtual Trojan to insert into the design.

E. HATE

The HArdware Trojan Emulation (HATE) [101] platform includes a hardware Trojan generator for targeting microprocessors. The hardware Trojans are *trigger-only* models that include combinational and sequential variants that are triggered by microprocessors values such as register values, instructions and operands, read and write values, and sequences of these. The hardware Trojan generator produces triggers with varying numbers of states, operand values, and number and value of registers.

F. BLACK-HAT HLS

The idea of a malicious HLS CAD flow for inserting Trojans is discussed in [30]. The authors discuss three types of Trojans: degradation, battery exhaustion, and downgrade attacks. Degradation attack is performed during scheduling and can add delays. Battery exhaustion and downgrade attacks both occur after the binding phase, with battery exhaustion altering the datapath to drain the battery and downgrade attacks hampering the security properties. The Trojans are controlled via a counter circuit or input sequence monitoring for sufficiently low activation probability.

G. ATTRITION

Attacking Static Hardware Trojan Detection Techniques Using Reinforcement Learning (ATTRITION) is a tool that focuses primarily on identifying trigger conditions that are unlikely to be detected during testing and validation [102]. To do this, ATTRITION forms triggers from rare nets in the design one net at a time. This is done to ensure that the resulting set of rare nets is compatible, meaning that there is an input pattern that results in all of the nets simultaneously obtaining their rare values, and also to avoid selection of rare nets that are dependent on each other. If the selected nets are dependent, then *any* test pattern that results in one of the nets obtaining its rare value will cause the dependent net to reach its rare value as well. This process increases the likelihood of detection.

H. ADVERSARIAL TROJAN EXAMPLES

Adversarial examples are targeted data points crafted to fool machine learning models. The authors in [103] present a framework for generating adversarial Trojan examples at the gate-level. A metric, Trojan-net concealment degree (TCD), is used to define the probability that a machine learning model correctly classifies Trojan nets. To generate adversarial examples, each Trojan net/gate has a list of corresponding transformations that can be applied. The transformation that causes the smallest TCD is kept. In this way, the authors can generate several Trojan variants that will perform poorly on models trained on traditional Trojan examples. However, it is worth noting that the model under evaluation is needed to generate these adversarial examples.

I. TROJAN PLAYGROUND AND TROJAN FORGE

Trojan Playground is a Trojan insertion framework that utilizes reinforcement learning (RL) [104]. Trojan Playground uses controllability and observability to short list a set of nets as candidate Trojan nets. During each training iteration, the reinforcement learning agent randomly inserts an AND-triggered, XOR-payload Trojan and then takes one of several actions for each trigger input. The RL agent is rewarded if the current set of nets selected are part of the candidate nets and functionally valid as checked with PODEM [105]. The reward function gives more weight to higher fan-in Trojans.

TrojanForge [106] is an extension of Trojan Playground that generates Trojans in a generative adversarial network (GAN)-like loop. An RL agent is used as the generator, and a test pattern based solution is used as the detector. The RL agent is rewarded positively for crafting Trojans that evade the detector, neutrally for successful, but detected Trojans, and negatively for invalid Trojans. The set of rare nets that can be used in the Trojan trigger are the result of a series of prunings that involve functional dependency and feature analysis. A payload is then selected such that no combinational loops are formed, similar to other methodologies.

J. BioHT

BioHT [107] proposes an automated framework that introduces Trojans in a tapeout ready design via ECO. Candidate trigger net selection is guided by transition probability and spatial clustering to ensure resistance to activation and low footprint. For payload selection, BioHT performs information flow analysis and calculates a similarity score to help identify high value signals in a design. A library of template Trojan structures are available for insertion. Once the Trojan nets have been identified, and the Trojan template selected, the Trojan is inserted by expressing it as an ECO. Finally, the Trojan triggers are validated using formal verification.

K. PERSPECTIVE AND OUTLOOK

Automated Trojan insertion is a useful tool for exploring the potential Trojan attack space. Producing large, diverse datasets is important for both training and evaluation of machine learning based detection approaches [81], [103].

Steady improvements have been made over the years in the design and insertion efforts, increasing abstraction scope, and improving automation, tune-ability, and Trojan stealthiness. More recent approaches incorporate machine learning, which can enhance Trojan diversity, leading to more robust datasets. Moving forward, it would be beneficial for automated benchmark generation frameworks to include targeted architectures, such as those described in Section II-D, and Trojans tailored to the host design. A summary of the automated tools appears in Table 2.

1) POTENTIAL PITFALLS

As automated Trojan benchmarking gains more traction for large dataset creation and threat modeling, we discuss several drawbacks and points for improvement of existing works.

a: RANDOM TROJANS

A common criticism of automated Trojan insertion is that "random" hardware Trojans are not representative of an intelligent attacker [102], [108]. While this point is valid, automated Trojan insertion still offers the benefit of exploring a wider breadth of the Trojan space than would be explored with a more tailored approach. To this end, there is a need to include more Trojans that are representative of the Trojan corpus beyond generic combinational and sequential Trojans.

b: BEYOND STATIC PROBABILITY

From the surveyed works on automated Trojan insertion, one of the only criteria reported as a measure of stealthiness is static probability. While important, there are other features that can contribute to functional stealthiness. To more accurately capture Trojan impact, feature based values should be selected to quantify both functional and parametric stealthiness.

IV. PRE-SILICON TROJAN DETECTION

Pre-silicon Trojan detection is a Trojan countermeasure that aims to identify any existing Trojans in an untrusted design, or any potential Trojans that may be introduced further down the supply chain. These techniques attempt to detect Trojans in hardware design files before the manufacturing process. Broadly speaking, pre-silicon detection techniques can be grouped as static and dynamic analysis and functional verification approaches. We further categorize these into feature analysis, information flow tracking, reverse engineering, machine learning, targeted test pattern generation, and formal methods. Designs under test in this phase are generally in the

TABLE 2. Comparison of automated Trojan insertion frameworks.

Method	Abstraction	Trigger	Payload	Automation	Insertion Strategy
TAINT [19]	FPGA	User-provided	User-provided	✗	Manually modify LUT values or reroute primitives
HAL [73]	ASIC/FPGA	User-provided	User-provided	✗	Manual modifications
TRIT [15]	ASIC	Signal probability	DoS, Functional Change, Leakage	✓	Random
HATE [101]	μP	Architectural state	–	✓	Random tap of architectural state for triggers
Black-hat HLS [30]	HLS	Template	DoS, Functional Change	✓	Insertion during scheduling or after binding
Improved [99]	ASIC	Signal probability	DoS, Functional Change, Leakage	✓	Random
TRIT-DS [32]	FPGA	Signal probability, critical path	DoS, Functional Change, Leakage	✓	Random & Insertion into underused primitives
MIMIC [100]	ASIC	Functional & Structural features	DoS, Functional Change, Leakage	✓	ML models trained on multi-features to ID triggers/payloads
Adversarial [103]	ASIC	Trojan-net concealment degree	DoS, Leakage	✓	Transform existing Trojan templates
ATTRITION [102]	ASIC	Signal probability	DoS, Functional Change	✓	Reinforcement learning to generate trigger sets
Trojan Playground [104]	ASIC	SCOAP	DoS, Functional Change	✓	Reinforcement learning to generate trigger sets
BioHT [107]	Layout	Transition probability, Spatial awareness	DoS, Functional Change, Leakage	✓	Engineering Change Order insertion
TrojanForge [106]	ASIC	Switching Probability, SCOAP	DoS, Functional Change	✓	GAN loop with reinforcement learning insertion

form of RTL, or gate-level netlists. Common Trojan threat models at this stage include 1) malicious third party IP (3PIP) vendors, 2) rogue or disgruntled insiders, and 3) malicious EDA software.

A. STATIC AND DYNAMIC ANALYSIS

Static analysis approaches investigate a static representation of the design to determine the presence of malicious logic. Such analysis can include detecting anomalous Boolean functions, netlist level features, or leakage paths. Dynamic analysis examines the design behavior in a dynamic state, often through simulation. Several of these approaches have been proposed that analyze circuits to attempt to find logic that is unused, mostly unused, or that does not impact the results of verification tests. Such logic is suspicious, as it apparently does not have an impact on the functionality of the circuit.

1) FEATURE ANALYSIS

Feature analysis approaches calculate metrics in a design to be used as discriminators for distinguishing between Trojan and benign nets. Once these features are extracted techniques such as thresholding or template matching are used to bin design artifacts such as nets, or lines of code as suspicious or benign.

a: BOOLEAN CONTROL

FANCI is a static Boolean function analysis technique that identifies nearly unused circuits in netlists [53]. This technique is based on the observation that Trojan trigger inputs usually have only a weak impact on the output of a circuit. To identify such weakly affecting input signals, FANCI uses the *control value* (CV) to estimate the impact that an input signal has on an output signal driven by it. The control value is calculated by forming a truth table or an approximate truth table for an output and then, for each input, finding the number of patterns for which changing only that input results in the output changing. This value is then divided by the size of the truth table, giving $CV = count/size_{tt}$. Each fan-in logic cone for each primary output and flip-flop input undergoes these calculations. Oftentimes, the control values of the logic cone are described by a summary metric, such as the median or mean, which is then compared to a threshold. If the value is below the threshold, then the primary output or flip-flip input associated with this vector is determined to be malicious.

Waksman also suggests a metric termed *triviality*, which is similar to signal probability of the truth table, but described as the weighted average of the control values. If the resulting value is 0 or 1, then the circuit is a contradiction or tautology, respectively. This metric is suggested for use on large modules where instead of generating the complete truth table some number of random input vectors are selected and the truth table is found for only these values.

ANGEL was developed as a static analysis approach that decreases the false positive rate of FANCI and that can detect Trojans implemented using the DeTrust approach [73]. It is based on Boolean function analysis and graph neighborhood analysis, and as with FANCI, it focuses on detecting logic cones with weakly affecting inputs. As with FANCI, ANGEL considers control values. However, rather than finding the control values for the fan-in logic cone of only the gate under consideration, ANGEL additionally includes the control values for the fan-in cones of the predecessor gates up to some depth d of predecessors. Sequential elements are ignored in this process but the predecessor gates of the input ports of the sequential elements are included in the graph cut. This is done in an attempt to detect trigger logic implemented across several sequential layers, which is an implementation technique suggested in DeTrust [2].

b: UNUSED LOGIC

Some feature analysis techniques use dynamic simulation to check if there is logic that is unactivated, redundant, or unreachable. Such logic is then deemed suspicious. Hicks et al. introduced a Trojan detection technique termed *unused circuit identification* (UCI) [52]. This approach identifies signal pairs (s, t) where s and t are separated by a combinational logic cloud and for which s carries the same value as t for every test case. This situation indicates that logic between s and t can be replaced with a wire without causing any of the tests to fail. In this scenario the logic between s and t is flagged as potentially malicious. If exhaustive testing of the circuit is possible, then this logic can safely be removed from the design. However, such testing is usually not possible.

The VeriTrust approach consists of a *tracer* and a *checker* [54]. This technique assumes that a set of verification tests is available, and that the hardware Trojan is not activated during these tests. The tracer is responsible for identifying any products or sums in the Boolean equation for a signal that are not activated during the verification tests. The checker then determines if there are any redundant inputs to the logic cone, and removes them if they exist. This has the effect of removing the Trojan trigger from the logic cone. Three types of checker are proposed:

- Remove unactivated products or sums from the expression
- Set unactivated products or sums as don't cares, and resynthesize the expression
- Identify non-redundant inputs rather than redundant inputs

The first checker is the simplest, but it is not guaranteed to eliminate all trigger inputs. This is because it is not guaranteed that the trigger inputs will always be grouped exclusively in unactivated terms. Some of the trigger inputs may also appear in activated terms, and which occurs depends on the initial equation. The second checker removes more trigger inputs than the first, but also cannot be guaranteed because logic synthesis does guarantee optimality during logic minimization. The third checker can be implemented by determining whether changing the input results in a change in the output for all input patterns. This is an expensive operation, so the author's propose an equivalent approach that involves comparing products and sums, and prove that it is equivalent to the exhaustive approach. After identifying non-redundant inputs, the redundant inputs can be removed.

Functional Identification of Gate-level Hardware Trustworthiness (FIGHT) is a metric-driven static analysis method for identifying suspect nets in a design [109]. The FIGHT framework first calculates the Boolean control of all nets and then creates a Boolean control distribution for the entire circuit. Outliers from this distribution are considered targets for use as Trojan triggers, and the circuit is assigned a security of 0 meaning it is vulnerable to Trojan insertion.

Fern et al. [46] use mutation testing coupled with manual inspection to analyze undetected faults, which are those that do not have an observable effect on the test suite. The authors posit undetected faults could be due to unspecified functionality and therefore warrant detailed inspection. A set of rules are used, such as high impact attacker-observable signals, to reduce the amount of manual review required.

The hardware Trojan catcher (HaTCh) algorithm begins by applying a set of test vectors to the circuit to identify those wires that are not activated by any of the test vectors [110]. Additional logic is then added to the design to track these unactivated signals during circuit operation and, if any of them transition, then an output flag is raised to indicate that a Trojan has been activated. Any Trojan that falls under the formal definition outlined by the authors will be detected.

Bomberman is a dynamic analysis technique introduced to detect timebomb Trojans [111]. The authors define a timebomb Trojan as a set of sequential elements that 1) do not repeat any values without resetting and 2) do not enumerate all possible values without causing some observable effect. To identify these Trojans, Bomberman first constructs a data flow graph from a suspect RTL design and identifies signals at the outputs of sequential elements through graph traversal. Initially, all sequential elements are marked as suspect. Then Bomberman arbitrates on all sequential elements based on the results of dynamic simulation – if the behavior violates any one of the properties, the sequential elements are marked as benign. Finally, a list of suspicious sequential elements are returned for further analysis.

c: CONTROLLABILITY AND OBSERVABILITY

Sandia Controllability and Observability Analysis Program (SCOAP) [112] values have proven to be a useful metric for Trojan detection. SCOAP is used to short-list nets for further analysis. For example, the logic signature based approach presented in [113] begins by segmenting the circuit into smaller sub-circuits. Then, within each segment a subset of nodes, such as those with low controllability or observability, are selected. Next, all possible input vectors are applied to the segment, and the values of the target nodes are collected. For each input, a vector consisting of the value of each of the target nodes is created and added as a row in a matrix. Next, the most similar rows in this matrix are identified. Ideally, the nodes with high similarity are not dependent on each other. A logical function describing the relationship between the selected nodes is created, and a circuit implementing this function is added to the circuit to serve as a run-time sensor. If Trojan logic activates and changes the values of any of the nodes in a way that deviates from the relationship observed during simulation, then the sensor will detect this.

d: CONTROL AND DATA FLOW GRAPH

The control and data flow graph (CDFG) is an important data structure available at higher levels of abstraction. It describes the flow and behavior of hardware as it executes through its finite state machine and datapath. Several works perform analysis on the CDFG to detect any suspicious behavior. For example, FASTrust [71] uses several structural features of the CDFG, such as presence of large loops, large fan-in, large loop in-degrees, and small out-degrees to label code as potentially malicious. ML-FASTrust extends FASTrust by including combinational logic analysis, which includes signal probability and signal toggle rate to reduce false positive rates [72].

CFG structural Trojan template matching is used in [77]. That work describes parameterized structural templates for a variety of Trojan triggers. A subgraph matching algorithm is employed to identify any subgraphs from the CFG that match any of the defined templates. A data-flow centric structural mapping has also been used for Trojan detection [114]. A "golden" library of known Trojan-free and Trojan-inserted designs is constructed from open source benchmarks, and used during the comparison with the design under test. Table 3 compares the detection capabilities of these feature-based detection approaches. In the table, Comb., Seq., and Degrad. refer to combinational triggers, sequential triggers, and degradation payloads, respectively.

2) INFORMATION FLOW TRACKING

Information flow tracking is a specialized subset of static and dynamic analysis. Analyzing information flow is fundamental to maintaining confidentiality of semiconductor designs [115]. To this end, several methods and language extensions have been developed for hardware information flow tracking (IFT) [78], [116], [117], [118]. IFT has also been used to detect potential Trojans that leak sensitive information. In [118], the authors showcase gate-level information flow tracking (GLIFT) for Trojan detection. With a GLIFT augmented representation of the netlist under test, sensitive assets are marked as critical and all other inputs as non-critical. Then a property can be written to check if the outputs remain tagged as non-critical. If the property passes, then no leakage via Trojans is detected. If the property fails, then a counter-example is provided to show the leakage path.

RTLIFT is an RTL-level information flow tracking technique [78]. RTLIFT generates instrumented, synthesizeable RTL code for analyzing information flow. The IFT is implemented via Verilog libraries which translate common operations to their IFT equivalents. They offer conservative and precise variants of the libraries depending on user needs. To track explicit information flow, the data is first labeled with security tags. Then from the data flow graph, each operation is replaced with its equivalent IFT operation. Implicit flow refers to potential information flow as the result of conditionals. RTLIFT tracks implicit flow using the control flow graph and correspondingly augments the code. Finally, security properties can be defined to validate information flow policies and detect Trojan introduced leakage paths.

Nahiyan et al. argue that GLIFT-based approaches cannot distinguish between Trojan-introduced leakage paths and suffer from scalability concerns [79]. Therefore, they propose an ATPG-based technique for information flow analysis by modeling asset leakage as stuck at fault testing. For a given asset, their technique identifies all observable points in the fanout and applies a bit mask so that each observable point can be addressed individually. Then for each observable point, if stuck-at fault testing returns any patterns, then a potential path is identified. The algorithm then continues through the fanout cone. An observable point with a valid path from an asset is considered potentially malicious and flagged for further inspection.

High level information flow tracking (HLIFT) [47] is an RTL approach for identifying leakage Trojans. HLIFT operates on the statement-level control and data flow graph representation of the design. From a Trojan-inserted design, HLIFT gathers a database of CDFG structures that can be considered malicious. The authors discuss three features: 1) the presence of control paths with data to output only, 2) presence of control paths with data in input and output and 3) control loops with data as input and output. Then HLIFT is run with the previous CDFG structures on a suspect IP to identify potential leakage Trojans.

EXhaustive IntEgRiTy Analysis (EXERT) is an exhaustive test pattern generation technique that utilizes IFT, formal analysis, and ATPG [119]. Given a netlist and an internal point of interest, EXERT iteratively performs fan-in analysis from a point of interest until either flip flops or primary inputs are reached. Upon reaching a flip flop its type, either datapath or FSM, determines the method of generating intermediate test patterns. For datapath flops, the combinational block is extracted and provided to a SAT solver. For FSM flops, the remaining FSM flops are extracted to construct a state transition representation, and then the patterns are extracted by traversing the FSM. These intermediate patterns are compressed by converting to an and-inverter graph, and the fan-in analysis continues until primary inputs are reached. The authors further enhanced the work (EXERTv2 [120]) to support analysis on systems with multiple, concurrent FSMs.

Formal methods have also been used for IFT. For example, properties can be specified to check whether an asset can flow to an observable point [24]. An alternative to the model checking approach of GLIFT, the authors in [121] and [122] present ways to model IFT through the Coq theorem prover. A method for converting a tracking augmented netlist to a formal syntax for theorem proving is presented in [122]. The Z3 solver is used in [123]. GLIFT augmented designs along with assertions are passed to the Z3 solver to reason about the presence of leakage paths induced by Trojans. IF-Tracker is a formal-based IFT framework at the SoC-level [124]. The abstract syntax tree for all IPs in the SoC are extracted and merged. In particular, the dataflow graph is used for taint tracking. Formal tools are then used to validate IFT properties for specified signals. We summarize the different IFT based approaches in Table 4.

3) FUNCTIONAL REVERSE ENGINEERING

Functional Reverse Engineering (RE) seeks to understand third party IP behavior received in a form, such as a netlist, where the behavior is not obvious. Through reverse engineering, designers can come to better understand the datapath and FSM of the design under verification. In and of themselves, RE techniques do not identify potential malicious structures without additional domain knowledge or analysis; however, RE is a vital tool in the analysis for Trojans. Subramanyan et al. first proposed using RE to identify Trojans [125]. They are able to infer the identified additional logic is malicious due to knowledge of the underlying circuit.

REFSM is an approach for reverse engineering the control logic of finite state machines from flattened netlists [33]. By treating the sequential triggering conditions of sequential Trojans as FSMs, this approach can aid designers in identifying sequential Trojan logic. The authors show that REFSM can be used to identify and isolate sequential Trojan state logic.

TABLE 3. Summary of feature-based pre-silicon Trojan detection techniques.

Approach	Method	Tested Abstraction	Trigger Comb.	Seq.	Always On	Payload† DoS	Functional Change	Leakage	Degrad.	Advanced Trojan Architectures†
UCI [52]	Unused Logic	RTL	✓	✓		✓	✓	✓		
FANCI [53]	Boolean Control	ASIC	✓	✓		✓	✓	✓	✓	
Veritrust [54]	Unused Logic	RTL/ASIC	✓	✓		✓	✓	✓	✓	
FIGHT [109]	Boolean Control	ASIC	✓	✓		✓	✓			
FASTrust [71]	CFG Analysis	RTL/ASIC	✓	✓		✓	✓	✓	✓	DeTrust [2], MOLES [4]
Fern et al. [46]	Unused Logic	RTL	✓					✓		Don't Care [46]
HaTCh [110]	Unused Logic	ASIC	✓	✓		✓	✓	✓	✓	DeTrust [2], k-XOR-LFSR [51]
CFG matching [77]	CFG Analysis	RTL	✓	✓		✓	✓	✓	✓	MOLES [4]
ML-FASTrust [72]	CFG Analysis	RTL/ASIC	✓	✓		✓	✓	✓	✓	DeTrust [2], MOLES [4]
ANGEL [73]	Boolean Control	ASIC	✓	✓	✓	✓	✓	✓		DeTrust [2], k-XOR-LFSR [51]
Bomberman [111]	Unused Logic	RTL		✓		–	–	–		

"–" refers to techniques that do not expressly target an aspect of the Trojan sub-circuit
† refers to detectability of tested Trojans

TABLE 4. Summary of information flow tracking pre-silicon Trojan detection techniques.

Approach	Method	Tested Abstraction	Trigger Comb.	Seq.	Always On	Payload† DoS	Functional Change	Leakage	Degrad.	Advanced Trojan Architectures†
Jin et al. [121]	Formal	RTL	–	–	–			✓		
BMC [24]	Formal	RTL	✓	✓	✓			✓		MOLES [4]
GLIFT [118]	Formal	ASIC	✓	✓	✓			✓		MOLES [4]
RTLIFT [78]	Formal	RTL	✓	✓	✓			✓		MOLES [4]
ATPG-IFT [79]	ATPG	ASIC	✓	✓	✓			✓		MOLES [4]
Qin et al. [122]	Formal	ASIC		✓				✓		
HLIFT [47]	CDFG	RTL	✓	✓	✓			✓		DeTrust [2], Don't Care [47]
Zhang et al. [123]	Formal	ASIC	✓		✓			✓		
IF-Tracker [124]	Formal	RTL	✓	✓		✓		✓		
EXERT [119]	ATPG	ASIC	✓	✓		✓		✓		
EXERTv2 [120]	ATPG	ASIC	✓	✓		✓		✓		

"–" refers to techniques that do not expressly target an aspect of the Trojan sub-circuit
† refers to detectability of tested Trojans

RELIC-FUN is a functional reverse engineering approach that uses k-feasible cut enumeration to identify classes of functionality in a netlist [126]. From a target signal, RELIC-FUN uses breadth-first search to enumerate the first k-cut slice. Then, for each class of functionality, both the target slice and the class are checked for functional equivalence by comparing truth tables while taking input ordering into account. RELIC-FUN then returns the type and number of each functional class, some of which may be identified as malicious upon further inspection and analysis.

HAL is an open source tool for netlist reverse engineering [73]. It takes a gate-level netlist as input and converts it to an internal graph data structure. HAL employs software libraries to operate on the graph and interpret subgraphs as binary decision diagrams. HAL operates modularly and includes plugins which offer different functionality such as simulation, visual inspection, and register grouping [34]. DANA is a HAL plugin for understanding register groupings. DANA first starts with an initial grouping, with all flops in their own group. Then, through a series of analyses or passes, DANA iteratively refines the grouping based on features such as having the same predecessors or successors, number of predecessors, and same control signals.

The approach proposed in [127] assumes the existence of a golden design netlist and access to fabricated chips that may have had Trojans inserted by the untrusted foundry. A netlist for the manufactured circuit is extracted using reverse engineering techniques. Then, the controllability and observability metrics for both the golden design netlist and the reverse engineered netlist are compared. If a mismatch occurs, then the path of gates from the net in question to the primary inputs are extracted for each netlist and compared. Any discrepancy in gates of the reverse engineered netlist from the golden are considered maliciously added by the foundry. Table 5 compares the different reverse engineering techniques that have been suggested for Trojan detection.

4) MACHINE LEARNING

Machine learning based static analysis is an increasingly popular solution for detecting hardware Trojans [128]. Unsupervised, supervised, and graph learning approaches have been investigated.

a: COMMON FEATURES

Previously, we described activation probability and observability as two features describing Trojan trigger and payload functional stealthiness. With the advent of ML based techniques, several other features have been used to describe Trojan behavior to help distinguish benign from malicious behavior. Features are grouped by whether they are calculated by netlist function or structure. Functional features can be calculated dynamically through simulation or statically via

TABLE 5. Summary of reverse engineering pre-silicon Trojan detection techniques.

Approach	Method	Tested Abstraction	Trigger Comb.	Seq.	Always On	Payload† DoS	Functional Change	Leakage	Degrad.	Advanced Trojan Architectures†
Subramanyan et al. [125]	Reverse Engineering	RTL		✓		✓	✓			
REFSM [33]	Reverse Engineering	RTL/ASIC	✓	✓	✓		✓	✓	✓	MOLES [4]
HAL/DANA [34], [73]	Reverse Engineering	FPGA/ASIC	✓					✓		
S. Rajendran et al. [127]	Reverse Engineering	ASIC	✓	✓			✓			

"–" refers to techniques that do not expressly target an aspect of the Trojan sub-circuit
† refers to detectability of tested Trojans

some input assumptions and propagation. Common functional features, beyond signal probability, include switching activity (signal activity), and sequential and combinational controllability. Structural features try to capture not only the structure of the Trojan but also its location in the netlist. We caution the inclusion of large numbers of structural features as attackers can easily achieve high variability in structure through construction. Some ML techniques also include neighborhood features, which are the feature values of the surrounding nets. Table 6 summarizes the list of features and the relevant works that use them.

b: UNSUPERVISED LEARNING

Unsupervised machine learning techniques are a subset of algorithms that group data without explicit training or having learned from a labeled dataset. One of the first unsupervised methods for Trojan detection was introduced in [141]. Nets are clustered by ordering points to identify cluster structure (OPTICS) based on pair-wise signal value correlation [142]. This was followed by the controllability and observability for hardware Trojan detection (COTD) technique that uses unsupervised clustering analysis of the controllability and observability of individual gates to differentiate Trojan gates from benign gates [8]. Controllability is a measure of the difficulty of setting a signal to a particular value, while observability is a measure of the difficulty of propagating a signal to an observation point, such as a primary output of the circuit. The COTD approach finds the combinational 0-controllability, combinational 1-controllability, and combinational observability for each signal in the design [112]. Then, the signals are clustered using *k-means* cluster analysis with $k = 3$ to produce a list of Trojan signals and a list of genuine signals. Salmani further improves COTD for Trojans in [134] by adding multiple rounds of clustering and suspicious signal filtering via *gradual n justification*. The authors in [131] argue that controllability and observability alone are not sufficient for more advanced Trojans. They augment the clustering by introducing difference amplified controllability and observability and then additionally cluster on dynamic transition probability.

The Sequential/Combinational Controllability and Observability features for hardware Trojan Detection (SC-COTD) approach expands on COTD by using both sequential and combinational testability measures as input to an ensemble of $k - means$ clustering models [132]. The results from the various clustering models are filtered and then combined with a majority voting procedure to produce the final classification. Four models are used for the clustering, with the input to two of the models being a tuple consisting of the combinational controllability and combinational observability, and the input to the other two being the sequential controllability and sequential observability. Each of these is clustered by $k - means$ models with $k = 2$ and $k = 3$. The filtering is based on the size of the clusters and a heuristic defining the maximum proportion of the circuit that the Trojan can occupy. PL-HTD [137] uses local outlier factor (LOF) to classify wires in a netlist as benign or suspect. A total of 55 structural and functional features are filtered using Xboost. Then, PCA is applied to reduce the dimensionality of the data. After feature extraction and normalization, outliers returned from LOF are considered suspect.

c: SUPERVISED LEARNING

The first pre-silicon supervised learning work to gain prominence trained a support vector machine (SVM) model on gate-level Trust-Hub Trojans with five features: fan-in, distance from flip-flop inputs and outputs, and distance to and from primary inputs and primary outputs [138]. The authors then studied the performance by changing to a random forest model and analyzing 51 structural netlist features. These features eventually narrowed to the most beneficial 11, resulting in an improvement in detection accuracy [139]. The work was extended to use neural networks [140].

Verification of IP Trust (VIPR) is a supervised ML technique for Trojan detection originally targeted for FPGA netlists [81]. They organize their models into an ensemble voter to increase overall robustness. To train their models, the authors extract sets of functional and structural features for every net in a labeled Trojanized benchmark. VIPR is the first machine learning approach to include an automated Trojan benchmark generator for training to avoid dataset bias of the limited Trust-Hub Trojans. The ensemble of trained models is applied to a suspect design and reports a set of potential Trojan nets. This work identified a need for separate sets of models trained for combinational and sequential Trojans due to the observed feature distributions for these populations of Trojan architectures. An ASIC version of VIPR is presented in [129]. The authors forgo an ensemble and use a single SVM. They also apply a set of post-processing algorithms

TABLE 6. List of common features used in machine learning Trojan detection.

Feature	Type	Abstraction		Works
		RTL	Gate	
Signal Probability	Functional	✓	✓	[81], [129], [130]
SCOAP	Functional	✓	✓	[8], [81], [129], [131]–[134]
Switching Activity	Functional	✓	✓	[81], [91], [129]
Gate Type	Functional		✓	[92], [129], [130], [135], [136]
C(D)FG Node Type	Functional	✓		[82], [88]
Distance to/from constants	Structural	✓	✓	[103], [137]
Distance to/from PI/PO	Structural	✓	✓	[81], [91], [103], [129], [130], [136]–[140]
Distance to/from FF	Structural	✓	✓	[91], [103], [129], [138]–[140]
Distance to/from MUX	Structural		✓	[91], [103], [137], [139], [140]
Distance to/from loops	Structural		✓	[103], [137]
Fan-in/Fan-out	Structural	✓	✓	[81], [91], [92], [103], [129], [130], [136]–[140]

to remove potential false positives by identifying structurally isolated suspect nets. Post-processing after machine learning classification is similarly discussed in [133]. This approach uses SCOAP values of nets in a gate-level netlist as the only features. Interestingly, the work in [133] also shows high accuracy in identifying true negatives, which generally is not shown in machine learning Trojan detection works.

In addition to training a generalized model, the authors in [129] propose design-specific training and evaluation via *pseudo self referencing* (PSR). PSR removes the requirements of golden design and Trojan benchmark availability. With the assumption that the Trojan logic is an insignificant fraction of the entire design, the authors first label all nets in a design as benign. They then introduce Trojans into the design via an automated Trojan insertion tool to learn Trojan features and train a design-specific ML model. Once trained, the ML model is reapplied on the original netlist with labels removed in an attempt to correctly identify any Trojan nets.

R-HTDetector is a neural network based machine learning model for Trojan detection in gate-level netlists [103]. The authors train a neural network with three middle layers on Trojan inserted benchmarks, but also incorporate adversarial training to handle Trojan data that appears outside of the training dataset. More discussion of adversarial example generation appears in Section III-H.

Shapely ensemble boosting (SEB) is a supervised and explainable learning framework for Trojan detection [91]. The authors use the same features as [139], but include switching activity and dynamic power. Since SEB includes explainable learning, Pan et al. are able to determine which features contribute to the Trojan detection accuracy. Of an initial 55 features, 8 were identified as meaningful, with the highest importance placed on switching activity, dynamic power and total number of nets.

At the RTL level, the authors in [76] use a probabilistic neural network (PNN) to reduce false positives returned by matching RTL CFGs against the CFGs of known hardware Trojan triggers [77]. They train the PNN on the Trojan trigger CFGs looking at features such as number of 2-4 node graphs, number of nodes and edges, and product of edges weighed by signal probability. An artificial immune system (AIS)-based algorithm is proposed in [82]. The AIS is trained on features extracted from control and data flow graphs of Trust-Hub RTL designs. The authors describe structural features mined from the CDFG including connectivity, control flow conditions, and control flow frequency. Yang et al. note the insufficient accuracy of AIS and propose a supervised approach for RTL analysis using random forests [74]. The RTL is divided into nodes with features extracted from each node including number of operands, proportion of different behavioral and logical operations, presence of control flow statements, control flow activity, and connectivity to I/O ports.

d: GRAPH LEARNING

A recent trend in machine learning based static detection approaches utilizes graph neural networks (GNNs). The use of GNNs in Trojan detection is natural due to the inherent graph structures available in RTL CDFGs and gate-level netlists. Moreover, GNNs remove the need for feature engineering of structural features [130]. The graph neural network for hardware Trojan detection (GNN4TJ) technique operates at the register transfer level [87]. After flattening the design to a single Verilog file an abstract syntax tree (AST) is generated for the design, and then a dataflow graph is extracted for each signal in the design. These DFGs are combined to create a DFG for the entire circuit. In this graph the vertices are the signals, constants, and operations present in the design, and there is an edge e_{ij} between vertices v_i and v_j if v_i depends on v_j or if v_j is an operator applied to v_i. This graph is then processed by a GNN to produce an *embedding* of the graph. This embedding is then processed by a multi-layer perceptron, which is a type of neural network. The output of this is a label indicating whether there is a Trojan in the design. An opensource generalized GNN framework for Trojan detection and other hardware security concerns is available in [143].

Yasaei et al. use a graph convolutional network (GCN) for detecting Trojans in RTL designs [88]. The GCN is trained on data augmented DFGs extracted from the RTL. The data augmentation process tags nodes as constants, signals, or one of many different logical, conditional, or bit manipulation operations. GATE-Net is another GCN based

Trojan detection solution that operates on netlists [135]. Gate-Net employs contrastive learning when training their GCN, which uses differences in the training data rather than explicit data labeling. The netlist is represented as a collection of subgraphs, with each subgraph generated from the fan-in of every gate. Feature-wise, Gate-Net is augmented only with gate type information, removing any need for feature engineering.

TrojanSAINT is a GNN based approach that operates at the gate-level, and so can be applied pre- or post-silicon [136]. The netlist is first parsed and translated into a graph in which gates are vertices and wires are edges. A feature vector for each gate is also produced. This feature vector consists of: 1) Gate type, using one-hot encoding, 2) Input and output degree of the gate, 3) Shortest distance to a primary input, and 4) Shortest distance to a primary output. Next, the graph sampling approach GraphSAINT [144] is used to construct subgraphs. These subgraphs are then processed using a GNN that was originally developed for netlist reverse engineering [145].

The researchers in [90] also approach the Trojan detection problem as a GNN classification problem. This approach uses both structural and behavioral features. The structural features are extracted from the netlist using a graph learning technique, with features including gate types and counts, as well the connectivity between gates. The behavioral features are extracted under both normal and abnormal operating conditions. These features are collected by first performing a timing simulation of the design under normal operating conditions. Then, the timing simulation is repeated, but with a clock frequency that exceeds the maximum operating frequency of the circuit, which induces some errors in the circuit functionality. Then outputs of the two simulations are compared to extract bit flip data at the outputs of the circuit. These structural and behavioral features are extracted from both clean and Trojanized implementations of a circuit. As the Trojanized variants have a different internal structure than the clean circuits, the bit flips observed during the overclocking simulations differ between clean and Trojanized variants. A GNN is trained on both normal and Trojanized circuits, and then used to classify previously unseen circuits as either normal or Trojanized.

NHTD-GL [130] uses GNNs to localize the Trojan trigger and payload circuits via node-wise detection. An undirected representation of the netlist is used in order to capture neighboring nodes with the graph encoder. Every node is augmented with a feature vector consisting of signal probability, gate type, distance from PIs and POs, and fanins and fan-outs. The GNN is trained in batches with benign batches merged with sampled Trojan nodes to combat data imbalance resulting from the Trojan circuit size being significantly smaller than the original design size.

GNN4HT [92] is a two stage graph learning approach that represents the netlist as an undirected graph. It uses ''global'' features to get a sense of position and neighborhood, including betweenness and eignvector centrality. The first stage in GNN4HT performs node-wise classification to identify potential Trojan nodes. The second stage then classifies these localized Trojan nodes into different Trojan functionality bins such as leakage and denial of service. Similar to previous approaches, the authors address data imbalance by creating structurally diverse, but functionally equivalent Trojan variants. Table 7 compares the results for the different GNN-based techniques, while Table 8 summarizes the machine learning techniques and elaborates on the Trojans used for evaluation of the various approaches.

B. FUNCTIONAL VERIFICATION

Functional verification is an integral part of the semiconductor design flow. Standard verification techniques such as logic testing and formal methods are used to verify that the design behaves according to the functional specification. Researchers have adapted these techniques for aiding in Trojan detection. Here we highlight techniques that utilize these approaches for detecting potential deviations from expected behavior.

1) LOGIC TESTING

A number of techniques propose to use logic testing for Trojan detection. The logic testing based approaches aim to create a set of targeted test vectors that activate hardware Trojans during verification. Logic testing is applicable to both pre- and post-silicon Trojan detection. Here, we provide an overview of the relevant approaches.

a: STATISTICAL TEST GENERATION

Statistical test generation techniques approach the problem of hardware Trojan detection through the properties of *N-detect* testing, which tries to activate a set of potential Trojan triggers N times through a set of carefully crafted test vectors. By increasing the amount of time a set of trigger nets are activated, *N-detect* testing increases the likelihood of activating and observing the hardware Trojan. These approaches generally sample from the potential Trojan space to evaluate their efficacy as enumerating all possible Trojans is infeasible.

Multiple Excitation of Rare Occurrences (MERO) is a test pattern generation technique targeted towards both activating and propagating the effects of triggered hardware Trojans [146]. MERO first constructs a graph representation of the input netlist and performs simulation to identify a set of suspect nets. From these suspect nets, MERO samples a population of validated $m-trigger$ Trojan nets with a random payload net such that no combinational loops are formed. From an initial set of vectors, MERO iteratively perturbs the vectors with the goal of maximizing the number of rare nodes satisfied for a given pattern. Finally, MERO calculates trigger and Trojan coverage values from the sampled population of Trojans.

Multiple Excitation of Rare Switching (MERS) is a statistical test generation technique for Trojan detection [16].

TABLE 7. Comparison of learning result metrics for graph learning Trojan detection approaches.

Approach	Abstraction	Detection-level	Accuracy	Precision	Recall	TNR	F1 Score	GNN Comparisons
GNN4TJ [87]	RTL	Circuit-wise	–	0.923	0.966	–	0.940	–
HW2VEC [143]	RTL	Circuit-wise	–	0.8811	0.8928	–	0.88563	[87]
GATE-Net [135]*	ASIC	Node-wise	–	0.975	0.96	–	0.97	–
GCN [88]	RTL	Node-wise	0.996	0.989	0.884	–	0.931	–
TrojanSAINT [136]	ASIC	Node-wise	–	–	0.98	0.96	–	–
Trojan Vaccine [90]	ASIC	Circuit-wise	0.9315	0.9335	0.9138	–	0.9128	[87]
NHTD-GL [130]	ASIC	Node-wise	0.998	0.978	0.890	–	0.921	[87], [135]
GNN4HT [92]	ASIC	Node-wise	–	–	0.9428	0.9722	–	[87], [130], [136], [143]

TNR = True Negative Rate
* averaged between combinational and sequential performance
"–" refers to unreported metrics

TABLE 8. Summary of machine learning pre-silicon Trojan detection techniques.

Approach	Method	Tested Abstraction	Trigger Comb.	Seq.	Always On	Payload† DoS	Functional Change	Leakage	Degrad.	Advanced Trojan Architectures†
OPTICS [141]	Unsupervised	ASIC	✓	✓		✓	✓	✓	✓	
COTD [8]	Unsupervised	ASIC	✓	✓		✓	✓	✓	✓	DeTrust [2], k-XOR-LFSR [51]
SVM [138]	Supervised	ASIC	✓	✓		✓	✓	✓	✓	
Random Forest [139]	Supervised	ASIC	✓	✓		✓	✓	✓	✓	
Neural Network [140]	Supervised	ASIC	✓	✓		✓	✓	✓	✓	
PNN [76]	Supervised	RTL	✓	✓		✓	✓	✓	✓	MOLES [4]
VIPR-FPGA [81]	Supervised	FPGA	✓	✓		✓	✓	✓	✓	Dark Silicon [32], MOLES [4]
Artificial Immune System [82]	Supervised	RTL	✓	✓	✓	✓	✓	✓	✓	MOLES [4]
Huang et al. [131]	Unsupervised	ASIC	✓	✓		–	–	–	–	
PL-HTD [137]	Unsupervised	ASIC	✓	✓		✓	✓	✓		
GNN4TJ [87]	Graph Learning	RTL	✓	✓		✓	✓	✓	✓	MOLES [4]
SC-COTD [132]	Unsupervised	ASIC	✓	✓		✓	✓	✓		
GATE-Net [135]	Graph Learning	ASIC	✓	✓		–	–	–	–	
GCN [88]	Graph Learning	RTL	✓	✓	✓			✓	✓	MOLES [4]
Lo et al. [133]	Supervised	ASIC	✓	✓		✓	✓	✓	✓	
R-HTDetector [103]	Supervised	ASIC	✓	✓		✓	✓	✓	✓	
Improved COTD [134]	Unsupervised	ASIC		✓			✓			
TrojanSAINT [136]	Graph Learning	ASIC	✓	✓		✓	✓	✓	✓	
Trojan Vaccine [90]	Graph Learning	ASIC	✓	✓	✓	✓	✓	✓	✓	MOLES [4]
NHTD-GL [130]	Graph Learning	ASIC	✓	✓	✓	✓	✓	✓	✓	
VIPR-ASIC [129]	Supervised	ASIC	✓	✓		✓	✓			
SHEB [91]	Supervised	ASIC		✓	✓			✓		MOLES [4]
GNN4HT [92]	Graph Learning	ASIC	✓	✓	✓	✓	✓	✓	✓	MOLES [4]

"–" refers to techniques that do not expressly target an aspect of the Trojan sub-circuit
† refers to detectability of tested Trojans

Fundamentally, MERS works similarly to MERO in that it 1) identifies a set of rare nets with low signal probabilities, 2) applies an initial test vector set, and 3) iteratively modifies the vector set to increase Trojan activity. With MERS, increased Trojan activity refers to the switching activity of rare nets, or toggling from non-rare to rare values. The intuition behind MERS comes from the observation that high fan-in Trojan triggers, while potentially difficult to detect with logic testing, will likely have a measureable effect on power consumption via switching. Therefore, MERS operates in vector pairs to achieve net toggling. For each vector pair, MERS perturbs one vector in the pair and measures the impact on switching activity, with the goal of maximizing the number of rare nets that are toggled. Finally, the resulting test patterns are optimized via reordering to maximize the switching of the rare nets while minimizing the switching of the remaining circuit. This final optimization is proposed to increase sensitivity to post-silicon side-channel analysis.

Similar to increasing sensitivity to power side-channels, a test pattern generation technique for increasing sensitivity to delay testing is introduced in [147]. Delay-based sensitivity increases only if the Trojan trigger and the Trojan path are active. This approach attempts to generate a test vector ordering such that subsequent pairs have maximal Hamming distance and feature distance, defined as the count and position of activated rare nodes. The initial vector set consists of vectors that activate the largest number of suspect nets in the design.

The previous techniques use signal probability thresholding to distinguish suspicious nets. Sebt el al. argue that hardware Trojan susceptibility metrics based on differences in SCOAP values for estimating switching activity are more accurate [148]. They apply custom directed test generation algorithms termed ExciteHT to activate and propagate combinations of suspect nets.

In [18] a *type-n* Trojan is defined as one that has n trigger inputs. The payloads considered in this work are XOR gates with the trigger as one input and the target wire as the other. As such, when the Trojan is active a signal in the design is inverted. A conditional stuck-at fault pattern of order n, denoted as $CSP - n$, is a test pattern that detects a stuck-at fault while setting specified values on n other signals. Type-n Trojans are detectable by $CSP - n$, though due to the complexity in generating such patterns this work considers test generation for $CPS - 1$ and whether such patterns generalize to detecting higher-order Trojans. The largest Trojans considered in this work are Type-4. Trojans with such small triggers may not be practical, and it is not clear if the approach scales well to detect Trojans with larger triggers.

The trigger activation by repeated maximal clique sampling (TARMAC) approach uses a satisfiability modulo theories (SMTs) solver to construct a test corresponding to each maximal clique in a circuit in an attempt to activate Trojan triggers [149]. In a graph a *clique* is a set of vertices having the property that there is an edge between every pair of vertices. A *maximal clique* is a clique that cannot have any additional vertices added to it. The TARMAC approach begins with a list of rare signals. It then constructs a satisfiability graph consisting of the set of rare signals that can simultaneously obtain their rare values in response to an input pattern. Valid Trojan trigger conditions that consist only of rare signals form a clique within this satisfiability graph. As an input vector that satisfies all of the signals within a clique also satisfies any subset of that clique, it is sufficient to find the maximal satisfiable cliques and input vectors that satisfy them.

The authors of [150] propose AdaTrust, which is a testing based Trojan detection scheme built off of their previous works [151], [152]. This approach consists of three phases. In the first phase, ATPG techniques are used to generate high-coverage test patterns that activate at least some of the Trojan logic. The power side-channel is measured for each of these test patterns, and then the patterns are ranked using the relative power difference (RPD). The test patterns with the largest RPD are then used as the starting point for the second phase, in which the test patterns are iteratively modified in an attempt to activate more Trojan logic while also preventing activation of as much of the benign circuitry as possible. In the final phase, several test patterns are superimposed to further magnify the Trojan signal.

b: ATPG AND FORMAL METHODS

Some techniques identify suspect nodes and use ATPG, formal methods, or a combination of the two for scalable test pattern generation. The authors of [153] use a joint directed testing and formal methods based approach for Trojan detection in partial scan designs using ATPG and model checking. The overall flow uses ATPG and model checking separately and jointly to generate test patterns to activate suspect nets. The main contribution lies in combining ATPG and model checking to more efficiently explore the state space in partial scan designs. The authors note that ATPG struggles with non-scan designs and model checking struggles with large state spaces. Because these struggles are complementary, the approach uses ATPG on the scan-chain portion and model checking on the non-scan portion in order to achieve better scalability than either approach individually.

DCAPE generates directed tests of RTL designs through assertions for activating targeted data paths [75]. These datapaths, or data channel security targets, are determined from low activation registers, which are calculated by the Levenberg-Marquardt algorithm. By propagating register activation probabilities, the approach detects DeTrust Trojans.

c: SYMBOLIC EXECUTION

Symbolic execution has been used for Trojan detection in RTL and HLS. RTL symbolic execution test generation is introduced in [154]. Path conditions from the symbolic engine are given to an SMT solver to generate test vectors. To remove the need for a golden model, metamorphic testing of the vector set is used to validate the design behavior. If (input, output) pairs violate a metamorphic relation, a method of defining some expected property of the design behavior, the violation is deemed a Trojan. SymbA is a similar RTL symbolic execution approach [86]. However, while [154] operates on CFGs extracted from RTL, SymbA uses the CFG extracted from the C level.

d: CONCOLIC TESTING

Concolic testing is the combination of concrete testing and symbolic execution for targeted test generation. A directed test-generation method for activating Trojans in RTL is presented in [80]. It attempts to activate Trojans by finding tests that exercise rare branches and assignments in the RTL. Initially, some assignment statements are replaced with equivalent *if-else* statements. This reduces the problem of finding rare assignments to the problem of finding rare branches, simplifying the implementation. Then, the design is instrumented by adding a *$display* statement for each functional statement in the design. Next, the design is simulated using a set of random inputs, and the number of times each branch is executed is counted. Any branch that is not activated during these simulations is considered rare and targeted for test generation. At this point the concolic testing begins. A target is selected for coverage and a concrete simulation is performed. If this simulation does not activate the target, then symbolic execution takes place. Concrete simulation and symbolic execution then occur iteratively until a test is found.

Another concolic testing approach for RTL designs is presented in [85]. This approach specifically attempts to generate tests that target hard to activate branches and scenarios that can be converted to equivalent branch statements. The process begins by instrumenting the design. For this, a CFG is generated for the design, each basic block is assigned a unique identifier, and *$display* statements are appended to the end of each basic block to print that basic block's identifier. This provides a trace of which basic blocks are executed during a simulation. Next, a contribution-aware edge realignment is performed on the CFG. This associates targets with the basic blocks that have direct contributions to it, and greatly reduces the search space for finding paths that lead to activation of the target. Next, a distance measure is used to determine which paths are "closest" to the target. Finally, the concolic testing occurs using a greedy path exploration scheme. To reduce the number of targets a pruning approach that eliminates all of the dominator target nodes of a target is employed. This is followed by a clustering approach to help find the most profitable initial path.

e: GENETIC ALGORITHMS

Genetic algorithms have been used to improve the coverage for logic testing based Trojan detection approaches [155], [156]. In [155], suspect nets are first selected selected via thresholding on signal probability. As with any genetic algorithm, the authors define a fitness function for their problem. The test vector fitness is a function of the number of unique triggered potential Trojan triggers, favoring new combinations over previously triggered ones. Test vectors are initially randomly generated with 'children' vectors, the result of swapping a certain section of bits between two 'parent' vectors and occasional random perturbations. Any Trojan triggers sampled that remain untriggered after several generations of the genetic algorithm are generated from a SAT tool. Through genetic algorithm, a higher coverage of potential Trojans is achieved with nearly the same number of test patterns as previous statistical methods.

TRIAGE [156], another genetic algorithm for test pattern generation, is similar to [155] in that it uses controllability, observability, and transition probability to determine if a net belongs to a population of rare nodes. Test vectors go through the same crossover and mutations as described in [155]. However, the number and frequency of rare nodes activated by a resulting test vector are used in the fitness function of TRIAGE instead of in generating trigger combinations from sets of rare nodes. The genetic algorithm terminates after a certain number of generations has passed or a target number of rare nodes have been activated using N-detect principles.

f: REINFORCEMENT LEARNING

Reinforcement learning is a recent trend in test pattern generation. Test Generation using Reinforcement Learning (TGRL) uses both rareness and testability to identify nodes to target for test generation under the assumption that trigger nodes are likely to have low controllability and payload nodes are likely to have low observability [112], [157]. The testability metrics and rare signal values are used as input to a reinforcement learning model that performs ATPG. The reinforcement learning model is rewarded for generating test vectors that activate rare signals and propagate them to observation points.

AdaTest is a reinforcement learning based logic testing technique [158]. This technique first applies circuit profiling, which calculates the transition probability via logic simulation and SCOAP metrics for each wire. AdaTest determines rare nets from the transition probability. The RL model attempts to generate a test vector set with a reward function that 1) prioritizes activating the largest number of rare nets, 2) prioritizes rare nets with high SCOAP values, and 3) achieves DAG-level diversity, where the DAG structure is a netlist representation with test set applied. Diversity is measured as the Hamming distance between two DAGs with different sets of test patterns. At runtime, AdaTest starts with an initial set of vectors and then iteratively generates new patterns and selects those with high reward values until the termination conditions are met.

Trojan Playground also proposes a reinforcement learning detection scheme that generates test vectors, applies them to the circuit, and then checks for deviations from a golden model [104]. The agent attempts to toggle rare nodes in the circuit under the assumption that Trojan triggers and payloads are usually dormant. Similarly, in DETERRENT [159] a learning agent tries to maximize the number of simultaneously satisfiable rare nets. The process is run until a certain threshold of large, distinct sets of rare nets are identified, and the corresponding test patterns are then generated. Table 9 summarizes the different logic testing techniques and their reported detection capabilities.

2) FORMAL METHODS

Formal methods can be used to reason about the security of the design through property or equivalence checking to identify potentially malicious perturbations. Techniques such as theorem proving, symbolic execution, and property checking have been used for Trojan detection.

a: THEOREM PROVING

Theorem proving refers to the process of constructing and formally proving different theorems relating to a system. Proof-assistants, such as Coq, have been used in proof-carrying hardware frameworks [160] that allow users to formally prove equivalence between the design specification and a formal representation of the design. Such techniques can be used to show the absence of Trojan. In [161] Coq was extended as formal-HDL to support RTL design descriptions and security properties for these RTL designs. Through functional and storage security properties, the authors describe how Trojans that can cause malfunction in the datapath or leak information through memory access would be thwarted in an 8051 microprocessor. Unfortunately, translating an RTL design to formal-HDL is a manual process. Efforts have been made to automate this process with VeriCoq [162]. VeriCoq-IFT [163] is another extension that adds support for information flow policies.

b: SYMBOLIC EXECUTION

Symbolic execution-based Trojan detection takes advantage of automated constraint generation to explore the state space of a design. The authors of [41] propose a symbolic approach for detecting potential Trojans that uses don't care transitions

TABLE 9. Summary of logic testing pre-silicon Trojan detection techniques.

Approach	Method	Tested	Trigger			Payload†				Advanced
		Abstraction	Comb.	Seq.	Always On	DoS	Functional Change	Leakage	Degrad.	Trojan Architectures†
MERO [146]	Statistical Testing	ASIC	✓	✓			✓			
MERS [16]	Side Channel Sensitivity	ASIC	✓	✓			✓			
ATPG [153]	ATPG& Formal	ASIC	✓	✓			✓			
Path Delay Testing [147]	Side Channel Sensitivity	ASIC	✓				✓			
CSP [18]	Fault Testing	ASIC	✓				✓			
Lixiang et al. [154]	Symbolic Execution	RTL	✓	✓		✓	✓			
ExciteHT [148]	Directed Testing	ASIC	✓	✓		✓	✓	✓		
SymbA [86]	Symbolic Algebra	RTL	✓	✓				✓	✓	MOLES [4]
Ahmed et al. [80]	Concolic Testing	RTL	✓	✓		✓	✓	✓	✓	MOLES [4]
Saha et al. [155]	Genetic Algorithm	ASIC	✓	✓			✓			
TRIAGE [156]	Genetic Algorithm	ASIC	✓			–	–	–		
AdaTest [158]	Reinforcement Learning	ASIC	✓				✓			
TARMAC [149]	Logic Testing	ASIC	✓	✓			✓	✓	✓	
AdaTrust [150]	Side Channel Sensitivity	ASIC	✓				✓		✓	
TGRL [157]	Reinforcement Learning	ASIC	✓			–	–	–		
Trojan Playground [104]	Reinforcement Learning	ASIC	✓				✓			
DETERRENT [159]	Reinforcement Learning	ASIC	✓			–	–	–		
DCAPE [75]	Directed Testing	RTL	✓	✓		✓	✓	✓		DeTrust [2]

"–" refers to techniques that do not expressly target an aspect of the Trojan sub-circuit
† refers to detectability of tested Trojans

in an FSM. A don't care transition is an implicitly defined transition from one state to another through the introduction of don't cares or a transition from an unreachable state to a reachable state, such as through a default statement. The approach, which is applicable at RTL and gate-level, uses the KLEE symbolic execution engine. The design under test is first converted to a C++ representation, which is then compiled to LLVM intermediate representation so that KLEE may analyze the FSM. KLEE is then used to identify FSM reachability through forward traversal of the state space and then potential don't care transitions via backward traversal. If primary outputs or internal register values change when transitioning from an unreachable state to a reachable state or as the result of a don't care transition, the approach assumes these states or transitions are suspicious and performs subsequent exploration to identify potential hardware Trojans.

c: PROPERTY CHECKING

Properties can be written to express different functional or security requirements that are likely to fail in the presence of a Trojan [70]. The authors of [164] introduce a property checking approach that operates on designs written in C or SystemC. Verilog to C translation tools, such as [165] can be used to make the approach applicable to designs written at the register transfer level. It is assumed that there is a simulation testbench that does not activate the Trojan. This approach is only applicable to Trojans with triggers implemented with an *if-elsif-else* clause, *switch-case* statement, or a *for* or *while* loop. This technique begins by expanding ternary operators into their corresponding if-else conditional statements. Next, a software profiler is run to identify any lines of code that are not activated by the simulation. Any such lines are treated as potential hardware Trojans. Next, for any block of code that was not executed during the simulation, if the first line of that block is an *if-elsif-else* clause, *switch-case* statement, or a *for* or *while* loop, then an assertion is generated for that line. The assertion states that the condition required by that line never occurs. These assertions are inserted into the C or SystemC description, and then the property checker attempts to generate counterexamples that exercise this code. If any counterexamples are found then these are added to the testbench and the design is re-simulated. Then, the user examines the output to determine if it is reasonable or if it appears that a Trojan is present.

ForASec is a model checking approach that addresses state-space explosion by partitioning large circuits into smaller subcircuits [166]. In this approach, the circuit is first translated into the symbolic model verification (SMV) model checking language using side-channel models of individual logic gates. Next, the design specification is translated into functional and non-functional linear temporal logic (LTL) properties. The SMV model of the circuit is then verified against the LTL properties. Next, a set of hardware Trojan benchmarks are also converted into SMV models, and these are inserted into the SMV model of the circuit under verification. This model is then verified using the LTL properties, and any suspicious gates are identified.

Antón et al. use interval property checking (IPC) to prove the absence of sequential Trojans in RTL designs [89]. IPC allows state machines to start in arbitrary states, permitting comparisons of two instances of the same design, one while the potential Trojan is active and one while it is dormant. Then, the authors can prove that 1) no perturbations occur from the primary input to the first level logic and 2) all subsequent logic levels between the two instances are equivalent when the same inputs are applied to the two designs.

As noted in Section II-D1b, Trojans can take advantage of unspecified functionality. The approach in [94] models such Trojan behavior as $C \wedge (f_{x \rightarrow x_0} \oplus f_{x \rightarrow x_1})$, where x is a signal in design f and C is some condition. This property can be provided to an SMT solver or to an equivalency checker. In the case of equivalency checking, two instances of the design with x under two different conditions are compared. Unfortunately, the enumeration of the (condition, signal) pairs is a manual effort.

Determining or even crafting properties to formally verify circuits can be a difficult task. Automatic security assertion extraction (ASAX) tackles the problem of identifying properties to assert for a design under test [83]. RTL invariants and signal properties gleaned from specifications or behavioral simulation can be used to auto-generate properties regarding correct signal behavior.

d: EQUIVALENCY CHECKING

Using equivalency checking between two variants of the design was suggested in [160] and [167]. Generally, a miter circuit is formed between two functionally equivalent designs with the same inputs and outputs. Then, a formal engine is applied to check for mismatched outputs under any input conditions. After examining the counter-example, designers can perform further analysis to determine if the source of the discrepancy is a Trojan.

The approach proposed in [168] extracts a set of polynomials from a netlist and a second set of polynomials from the circuit specification, which is assumed to be Trojan-free. These two sets are then checked for equivalence, with any gates in the implementation not required for satisfying the specification considered suspicious. Formulating the Trojan detection problem as an equivalence checking problem is useful, but suffers from the requirement of having a golden model.

Zero-suppressed binary decision diagrams (ZDD) have been shown to be an efficient representation of Boolean functions for equivalency-checking based Trojan detection in Galois-field arithmetic designs [169]. The ZDD representation of each output bit and gate is computed, followed by computation of the corresponding polynomial degree of the gate ZDDs. Suspicious gates, which are those with a high polynomial degree, are compared to the specification with an equivalency checker to detect Trojans. Table 10 presents a summary of formal method based detection approaches and elaborates on their experimental setups.

C. METRICS

Throughout the development of different Trojan detection techniques, researchers have proposed metrics to measure the success of their approaches and the vulnerability of circuits to Trojan insertion. We now describe several of these metrics.

1) STANDARD VERIFICATION METRICS

A survey of coverage metrics, which can be used to ensure that all or most of the logic in a circuit is tested, is presented in [170]. These metrics check to see if the various expressions, branches, and paths within the RTL description of the design are exercised by the test suite. Other metrics are based on circuit structure and measure whether each binary node is toggled at least once, that each register is initialized and read from, and that each register to register path is exercised. Such metrics may also examine counters and whether they obtain their maximum and minimum values. Metrics based on state transition graphs ensure coverage or reachability of the various states, paths, and transitions in finite state machines. As their name implies, functionality based metrics measure how much of the functionality of the circuit is exercised by the test suite.

In most cases, standard verification practices are insufficient in detecting the presence of Trojans. For example, a test may have 100% code coverage, yet still contain unexecuted code that an attacker can use for Trojan logic [52]. To this end, researchers have developed metrics targeted towards detecting hardware Trojans.

2) TRIGGER AND TROJAN COVERAGE

First introduced in [146], trigger and Trojan coverage are coverage-based metrics for assessing a test pattern's ability to activate and propagate Trojans. For a given sampled Trojan population T, trigger coverage ($Trig_{cov}$) and Trojan coverage ($Troj_{cov}$) are defined according to Equation 1.

$$Trig_{cov} = \frac{T_{activated}}{T} \quad Troj_{cov} = \frac{T_{propagated}}{T} \tag{1}$$

A noteworthy relationship between these two metrics is that $Troj_{cov} \leq Trig_{cov}$ because for any Trojan effect to be propagated it must first be activated. The metrics can also be abstracted beyond test patterns and be applied to any Trojan detection technique that considers a sample of potential Trojans during evaluation. A potential pitfall of these metrics lies in the assumed Trojan population. A defender should clearly state what type and how many Trojans are under consideration when calculating trigger and Trojan coverage. Table 11 compares different logic testing techniques and their reported testing information and coverage metrics.

3) CONFIDENCE VALUE

A confidence value (ConfVal) for comparing Trojan detection approaches is proposed in [104]. This metric trades off the undesirability of false positives (FP) and false negatives (FN) with parameter α according to

$$ConfVal = \frac{1 - FP}{\frac{1}{\alpha} + FN} \tag{2}$$

D. PERSPECTIVE AND OUTLOOK

As it is not possible to prove the absence of Trojan functionality in a design, it is not possible to have complete assurance that a design is free of hardware Trojans. The history of the literature shows that, as Trojan detection techniques become more powerful, new Trojan implementation styles are developed that avoid detection. For instance, the introduction of FANCI, VeriTrust, and UCI [52], [53], [54] led to the development of DeTrust Trojans [2] for evading FANCI and VeriTrust, Trojans undetectable by UCI [50],

TABLE 10. **Summary of formal methods pre-silicon Trojan detection techniques.**

Approach	Method	Tested Abstraction	Trigger Comb.	Seq.	Always On	Payload† DoS	Functional Change	Leakage	Degrad.	Advanced Trojan Architectures†
Proof Carrying [161]*	Theorem Proving	RTL	✓	✓		✓	✓	✓		
Rajendran et al. [70]	Property Checking	RTL	✓	✓		✓	✓	✓		MOLES [4]
Veeranna et al. [164]	Property Checking	RTL	✓	✓		✓	✓	✓		
Fern et al. [94]	Property Checking	RTL	–	–	–			✓		Don't Care [46]
Farahmandi et al. [168]	Equivalency Checking	ASIC	✓	✓		✓	✓	✓		
ForASec [166]	Model Checking	ASIC	✓	✓			✓	✓	✓	
Interval Property Checking [89]	Property Checking	RTL	✓	✓	✓	✓	✓	✓	✓	MOLES [4]
Dai et al. [41]	Symbolic Execution	RTL/ASIC		✓		✓				Don't Care [46]
ASAX [83]	Property Checking	RTL	✓	✓	✓			✓		MOLES [4], DeTrust [2]
Ito et al. [169]	Equivalency Checking	ASIC	✓					✓		

"–" refers to techniques that do not expressly target an aspect of the Trojan sub-circuit
† refers to detectability of tested Trojans
* refers to discussion instead of experimentation

TABLE 11. **Comparison of coverage values for logic testing Trojan detection approaches.**

Approach	Rare Threshold	Trigger Size	Trojans Sampled	Test Length	Trigger Coverage	Trojan Coverage	Comparisons
MERO [146]	0.2	4	100000	14534*	93.50	82.78	–
Saha et al. [155]	0.1	4	100000 (C) 10000 (S)	19425	79.44	66.43	[146]
TRIAGE [156]	0.1	–	100000	22857	81.00	–	[146], [155]
TARMAC [149]	0.1, 0.005	8	1000	19697	93.10	–	[146]
TGRL [157]	0.1	–	1000	21955	96.12	–	[146], [149]
DETERRENT [159]	0.1	4	100	3638.25	95.75	–	[149], [157]
AdaTest [158]	0.1 (C), 0.0005 (S)	3	50	945.06	97.83	98.25	[146], [156]

(C) = combinational; (S) = sequential
"–" unreported metrics
* reported for 2 trigger, and threshold of 0.2

and suggested coding styles for avoiding all three detection techniques [31]. In response to this, the ANGEL technique was developed to detect DeTrust Trojans [73]. While Trojans designed to evade ANGEL have not yet been published, it is likely that such Trojans will be designed. With attackers having the advantage of knowing the detection techniques, and defenders facing the challenge of detecting *any* possible Trojan, we generally expect Trojan development to outpace detection capabilities, and for this reason those requiring high levels of hardware assurance should adopt a *defense in depth* technique that involves combining several distinct Trojan detection approaches at various points in the design cycle. For example, such an approach might include RTL based detection approaches, netlist based techniques, and pre- and post-silicon testing techniques. Combining these with Trojan prevention approaches, such as those described in Section V, may lead to even higher levels of assurance.

1) POTENTIAL PITFALLS

As summarized in Section IV-A4, in recent years the community has shown a high interest in logic testing and machine learning based Trojan detection. While published results often appear impressive, we want to highlight potential pitfalls. In particular, we discuss assumptions on thresholding, potential problems with training data for machine learning, and general experimental testing of Trojan designs.

a: UNREALISTIC ASSUMPTIONS

Oftentimes logic testing techniques are bound by strong attacker assumptions to limit the scope of the problem. For example, researchers often first identify a population of suspect nets from which the attacker can choose. However, attackers are free to choose both non-suspect and suspect nodes to construct their Trojans [15]. Additionally, for approaches that use thresholding to determine rare net populations, the experimental results use relatively high thresholds (0.1) and low fan-in (2)-(4), which result in non-rare Trojan triggers [146]. To this end we suggest, in addition to setting a net level threshold, to also select a target Trojan activation probability to ensure the Trojans under evaluation by pre-silicon detection methods meet some minimum functional stealthiness requirements.

b: MACHINE LEARNING AND TRAINING DATA

In practice, ML-based techniques should be applied with caution, as such general purpose models are highly dependent on the selected features, the availability of training data, and the host designs of both the training data and the circuits that the trained models are applied to. Additionally, general purpose models may struggle with covering the diverse collection of published Trojan architectures presented in Sections II-A - II-D.

c: COVERING THE KNOWN KNOWNS

Due to the wide corpus of Trojan literature, it is difficult to keep track of all the different hardware Trojans as they appear. We observe that a majority of the research targets generic combinational and sequential Trojans without much attention to specific Trojan architectures such as DeTest, DeTrust, and DeUCI. While it may be argued that some techniques target fundamental Trojan representations and so include all extensions and implementations, we encourage researchers to begin including either testing of the known Trojans outlined in this work or a discussion on how the detection techniques can be extended to other Trojan types.

V. PRE-SILICON TROJAN PREVENTION

Trojan prevention is the other major category of Trojan countermeasure techniques. At a high-level, Trojan prevention seeks to make it difficult or otherwise undesirable for Trojans to be inserted in a design through introduction of design processes or modifications. Pre-silicon hardware Trojan prevention takes advantage of the flexibility offered at these stages to incorporate the design philosophy or testing infrastructure necessary to create hostile-to-Trojan environments that decrease the likelihood of Trojan insertion.

A. DESIGN FOR TEST

Design for Test (DfT) is an important semiconductor verification design practice for increasing testability and improving test coverage. The scan chain is an integral DfT architecture that aids in testing the design post fabrication. DfT for Trojan prevention includes design techniques that utilize or modify existing DfT infrastructure to increase testability of potential Trojans.

1) TEST POINT INSERTION

Researchers have attempted to adapt the scan chain to help in Trojan detection through insertion of additional scan flip-flops in key locations. A design for test Trojan prevention technique is presented in [171]. There, the authors propose identifying signals in a design for which there is a large discrepancy in the probability P_1 that the signal is a logical '1' and the probability P_0 that it is a logical '0'. Scan flip flops are inserted prior to these signals to allow controlling them to their infrequent values. This procedure eliminates rare nets in the design, complicating hardware Trojan trigger design and increasing the likelihood that Trojans are triggered during test. In [172] the authors also insert test points to increase transition probability, but instead insert MUXs with a directly connected FF for immediate control. The test points are also inserted with fan-out analysis in order to minimize the number of test points.

A DfT strategy for increasing the complexity of Trojan insertion is presented in [173]. In this work the authors assume an attacker goal of learning sensitive information, such as a cryptographic key. Given this assumption, the technique begins by identifying sensitive assets within the design, and then defines a metric to determine how closely related to the sensitive signals other signals are. Based on the selection of sensitive signals, and signals closely related to them, probe cells are then inserted into the design. These cells monitor internal signals and raise an alert if abnormal behavior is detected on them. Test vectors are generated at the time of probe insertion, and then these are applied to the manufactured device. Deviation of measured probe values from the expected values indicates that the design has been changed in some way, perhaps by an activated Trojan. If this occurs, then reverse engineering or some other form of analysis can be performed to identify the source of the deviation.

ODETTE is another DfT strategy for detecting Trojans [174]. First, the state space of the design is expanded by using both the Q and $\overline{Q}$ outputs of the non-scan flip flops present in the design. These flip flop outputs are MUXed so that either of them can be selected. This improves the controllability of difficult to control nodes, increasing circuit transition activity levels. These flip flops are then partitioned into a number of sets, with each set having a minimal number of correlated flip flops. This helps to ensure that more combinations of flip flop states can be achieved within the design. Together, these techniques enable greater activity of circuit logic, aiding in Trojan detection. Additionally, the authors suggest that the approach can also be used to obfuscate the underlying circuit as an attacker will not know whether the Q or $\overline{Q}$ outputs of the flip-flops will be used in the circuit. Consequently, it should be more difficult for the attacker to determine the functionality of the circuit and so designing Trojans that target it will also be more difficult.

Maximizing transition probability of hard to control nets can increase the likelihood of complete or partial Trojan activation and subsequent detection. Propagation of maximal transition probability (PMTP) is a DfT insertion [175] that facilitates other Trojan detection schemes such as logic testing and side channel analysis. PMTP inserts logic gates as test points to resolve conflicts for achieving maximal transition probability.

Wei et al. insert test points to increase the number of observable points for timing based analysis [176]. The test points are also inserted such that they break reconvergent paths, facilitating analysis. This technique is further improved with partitioning [177].

2) SCAN REORDERING

Scan reordering is another power-aware DfT technique that has been adapted for Trojan prevention [178]. Salmani and Tehranipoor divide the layout into a number of regions, with separate scan chains in each region. By reordering, designers can localize switching activity to a region of the design and increase the sensitivity of an inserted Trojan to power analysis based detection. Ritesh et al. discuss partitioning the scan chain into different segments to provide greater control over circuit activity in different regions of the design [179]. Table 12 provides a comparison of DfT based solutions for Trojan prevention.

B. DESIGN FOR SECURITY

Design for Security (DfS) solutions are architectural changes or design paradigms introduced to protect or maintain some aspect of the design's security. As opposed to DfT, DfS goes beyond improving testability metrics to address the potential for Trojan insertion. Broadly speaking, DfS tries remove the available space for Trojans through resource

TABLE 12. Summary of design for test pre-silicon Trojan prevention techniques.

Approach	Method	Tested Abstraction	Trigger Comb.	Seq.	Always On	Payload† DoS	Functional Change	Leakage	Degrad.	Advanced Trojan Architectures†
Salmani et al. [171]	Test Point Insertion	ASIC	✓				✓			
ODETTE [174]	MUX Non Scan Flip-Flops	ASIC	✓				✓			
Salmani et al. [178]	Scan Reordering	ASIC	✓	✓		–	–	–	–	
Zhou et al. [172]	MUX Test Point Insertion	ASIC	✓	✓			✓			
Ritesh et al. [179]	Scan Partitioning	ASIC		✓			✓			
Wei et al. [177]	Test Point Insertion	ASIC	✓				✓			
PMTP [175]	Conflict-based Test Point Insertion	ASIC	✓				✓			

"–" refers to techniques that do not expressly target an aspect of the Trojan sub-circuit
† refers to tested Trojans

filling, anti-tamper mechanisms, design obfuscation, or by increasing tolerance to abnormal behavior.

1) RESOURCE FILLING

Resource filling or starvation-based approaches limit resources available for Trojan insertion. This can include occupying physical space and routing, and removing internal nets with low controllability. Resource filling can occur at different levels of abstraction, though the majority of the work addresses the layout-level, where space becomes a valuable resource.

a: LAYOUT FILLING

Ba et al. describe a technique that fills the empty spaces in a layout with additional combinational logic [180]. This prevents post-layout Trojan insertion by denying attackers the space needed to implement and route the Trojan functionality. These functions are then tested post-fabrication to ensure that they have not been modified. Starting with a placed, but not routed, design this approach first identifies the empty spaces, and then fits the maximum number of flip flops into them. Next, the remaining space is filled with logic cells implementing one or more combinational functions. A single function is inserted if no flip flops fit in the space; otherwise, the flip flops are used as shift registers to shift data into and out of the functions. Finally, the circuit is routed and the process is repeated until the maximum routable occupation ratio is achieved.

Built in Self Authentication (BISA) is a Trojan prevention technique that fills unused space with testing logic independent of the original design [181]. The goal is to design BISA such that the test engineer can achieve complete functional coverage of the added logic. An LFSR is used to generate the test vectors and a multiple input signature register (MISR) is used for the output. A test engineer can then apply the tests and observe the output to detect any potential alterations.

The authors of [182] propose a metric for evaluating layouts for their vulnerability to Trojan insertion at the foundry and a related algorithm for reducing the vulnerability of layouts with respect to this metric. The layout is divided into small bins, and then the Trojan vulnerability factor for each bin is calculated as the product of the available routing resources within the bin and the ratio of the unused space within the bin to the area of the smallest cell in the library. An incremental placement algorithm sorts these bins by their vulnerability and then shifts the unused space in the layout to congested regions to reduce the overall vulnerability factor of the layout.

TroMUX [183] uses MUX-based locking to fill any unused layout spaces. Locking of security critical flip flops is prioritized, followed by locking as many low controllability nets as possible. TroMUX is scoped to prevent additive-based Trojans by filling the available space and also preventing design analysis through obfuscation.

GDSII-Guard is an engineering change order (ECO) based placement and routing technique for reducing a layout's vulnerability to Trojan insertion at the foundry [184]. This approach measures Trojan vulnerability as a weighted sum of the ratios of available space for placing new cells and available regions for adding routes between the optimized layout and the original layout. The ECO operators include shifting cells to reduce contiguous regions of unused space, local adjustments to placement density, and scaling of routing widths.

b: LOGIC FILLING

Several approaches have been proposed to fill empty or under-utilized primitives in FPGAs to prevent Trojan insertion. The authors in [185] introduce a method of filling unused space, both empty and under-utilized, in an FPGA with dummy logic. In [57], the authors use logic locking to obfuscate the design for Trojan prevention. They insert key bits into the unused space of under-utilized primitives, such as LUTs, and modify the function to lock the original functionality. Similar to TroMUX, this approach may provide added resilience against Trojan insertion by locking the original design.

Filling Logic and Testing Spatially (FLATS) is another logic filling approach for Trojan prevention [186]. FLATS first ensures that after synthesis a pre-defined number of LUTS with mapped logic have at least one input and output reserved and then tied together to create an oscillator. Supplementary logic is also added for controlling the added oscillators via an LFSR. The LFSR seed is kept secret to prevent an attacker from identifying the selected oscillators. Once the design is placed-and-routed, FLATS undergoes a registration phase, which categorizes a subset of the LUTs as beacons. The corresponding activation inputs and physical locations of these LUTs are noted. Another subset of LUTs are registered as authenticators, and their activating sequences are then selected and recorded, along with their distance from the beacon LUTs. For tamper detection, detector LUTs are selected, and their sequence registered after simulating design functionality. An IR camera can then be used in-field to authenticate and detect potential tampering by identifying any discrepancies among the distances between beacons and detectors due to hardware Trojan insertion. Moreover, because FLATS fills unused LUTs it complicates Trojan insertion.

c: ROUTE FILLING

The authors of [187] argue that BISA and similar techniques fail to completely fill usable area. They propose denying attackers the resources required for routing Trojans by filling unused portions of the layout with polysilicon. Any changes to the polysilicon by an attacker can be detected via a delay measurement readout circuit. The authors show they are able to achieve 100% final occupation of the die, completely removing the attack surface. Table 13 compares the different filling approaches discussed.

2) ANTI-TAMPER

Anti-tamper Trojan prevention includes design techniques that attempt dissuade attackers from tampering with the design. While anti-tamper's scope is broader than the problem of hardware Trojans, we focus on the anti-tamper techniques specifically introduced to address Trojan insertion.

a: TPAD: TROJAN PREVENTION AND DETECTION

The TPAD Trojan prevention and detection approach is based on concurrent error detection (CED) [190]. This technique is applicable to systems consisting of multiple chips in which each chip has TPAD protections. The outputs of each chip are encoded using a random parity scheme. Both the output bits and the parity value are then routed to the inputs of another chip. This chip can then verify the parity bits to ensure that the outputs have not been modified since the encoding occurred. CED is applied to both the logic and any memory in the design [191]. Error signals from the CED are encoded and transmitted to a separate error monitor.

b: TARGETED TAMPER EVIDENT ROUTING

The targeted tamper evident routing (T-TER) technique proposed in [96] aims to prevent Trojan insertion at the foundry. This approach shields security critical wires with guard wires, preventing attackers from routing Trojan signals to or adjacent to these wires. The guard wires surround all surfaces of the security critical wires, preventing attackers from integrating Trojan circuitry with the protected wires.

c: SECURE SYNTHESIS

Secure synthesis approaches attempt to synthesize or incorporate standard cells that makes the design more hostile to Trojan insertion. The authors of [192] propose synthesizing designs to a subset of the cells from the gate library that have low Trojan miss rates. Trojan miss rate, as defined by the authors, measures the sensitivity to detecting modifications via area differences. In this technique cells are iteratively replaced with functionally equivalent cells with different drive strengths and resynthesized, keeping the largest observed gradient calculation based on Trojan miss rate, and timing, power, and area statistics.

3) OBFUSCATION

Hardware obfuscation, logic locking, and logic redaction are maturing techniques primarily proposed to protect design confidentiality. Many works in this area suggest Trojan resilience is implicit by nature of the defense, and some work vaguely argues Trojan prevention [193]. We highlight techniques that specifically target hardware Trojans or include in-depth discussions on the topic of Trojan prevention besides IP protection.

a: LOGIC LOCKING

Logic locking is the process of locking the functionality of a design through the insertion of locking gates controlled by a key value. Only after providing the correct key values for all locking gates does the circuit function as intended.

In [194], the authors introduce AND/OR locking gates at nets with low controllability, with the goal of removing all hard to control nets in the design. The type of locking gate inserted depends on whether the net was low 1 or low 0 controllability. This prevents the attacker from selecting candidate nets to craft a stealthy trigger within the locked circuit. This idea is improved in [195], where the authors remove hard to control nets, introduce more diverse locking gates and create false hard to control nets. When an incorrect key is provided, the locking gates appear as attractive nets for crafting Trojans triggers. However, when the correct key is applied, the net retains its original behavior, making the resulting Trojan much easier to detect than intended by the attacker. Reusing logic masking to facilitate path-delay-based Trojan detection (ESCALATION) uses XOR/XNOR/MUX locking gates to create shorter paths in the design [196]. The authors argue short paths are easier to verify against perturbations outside of process variation. Therefore, increasing the number of short paths reduces the attack surface area for Trojan insertion and makes it easier to identify delay signatures to compare against the fabricated design.

b: LOGIC REDACTION

Researchers have proposed removing access to critical logic altogether by mapping to programming logic to thwart attempts at understanding designs for tailored Trojan insertion. This process is referred to as logic redaction. Security with ambiguous netlists (SWAN) [197] obfuscates critical logic by implementing it in a programmable fabric. Fixed function logic blocks with programmable crossbars are used

TABLE 13. Summary of resource filling pre-silicon Trojan prevention techniques.

Approach	Method	Tested on Trojans	Evaluation Criteria	Results
BISA [181]	Layout Fill	✗*	Test Coverage	100% BISA test coverage
Ba et al. [180]	Layout Fill	✗	Layout utilization	92% final occupancy rate
Hossein et al. [182]	Layout Fill	✗	Maximum Vulnerability Factor (MVF)	48.33% reduction in MVF
Khaleghi et al. [185]	Logic Fill	✗	Logic and routing utilization	100% prog. logic usage & 7.77% routing usage
Karam et al. [57]	Logic Fill	✗	Logic utilization	Max LUT usage increased 235%
Polysilicon [187]	Route Fill	✓	Layout utilization & Readout Accuracy	100% poly layer filled & Detection of removal attack
GDSII-Guard [184]	Layout Fill	✗	Layout utilization	98.7% site utilization & 98.9% track utilization
TroMux [183]	Layout Fill Locking	✗	Layout utilization and attack resilience	96.7% utilization; protect against SCOPE [188] and MuxLink [189]

* discussion instead of implementation

to reduce overhead. The flexibility of programmable logic allows for many functionally equivalent designs, increasing the obfuscation. Additionally, SWAN allows for duplicating logic and creates canary logic driven by LFSRs, similar in function to BISA, in the unused space. Successful Trojan insertion is complicated by the number of valid configurations that can exist.

c: STATE SPACE OBFUSCATION

State space obfuscation transforms the original state space by adding a locking FSM. Authorized users must correctly traverse the locking FSM to achieve normal FSM functionality. State space obfuscation has also been proposed as a Trojan prevention technique [198]. HARPOON ([199]) and similar methods introduce a locking FSM by adding extra flip-flops and state transitions, such that a user must supply the correct unlocking sequence to open the design [200]. The locking FSM also controls modification cells throughout the datapath that corrupt its functionality. For Trojan prevention, the authors argue that 1) the introduced obfuscated state space is more attractive to insert Trojans and 2) these added states are unreachable in normal operation. Any Trojans inserted in this space would be disabled during authorized usage. Table 14 compares the obfuscation approaches and how they are evaluated.

4) PATCHING

We refer to techniques that detect and address structural and functional anomalies or tolerate abnormal functional behavior as *patching*. The broad class of patching-based prevention techniques is a super set of design-level techniques capable of supporting in-field 'patches' or updates. We discuss three types of Trojan prevention countermeasures: modification, redundancy, and patching.

a: TARGETED MODIFICATION

The authors of [84] propose denying an attacker the ability to trigger a Trojan or achieve the effect of the Trojan's payload by making modifications to netlists. This technique assumes that there exists a suite of test vectors that are adequate for verifying the correct functionality of the circuit and asserts that any circuits that correctly implement the behavior defined by this test set, other than those that trivially provide the expected output for a given input vector, are equally correct. Under this assumption, the netlist can be modified arbitrarily so long as the validation set is satisfied. Consequently, Trojan detection techniques can be applied to the netlist, and any suspected Trojan gates can be targeted for modification. These modifications may aim to prevent a Trojan from being triggered, such as by setting a node that rarely transitions to the value that it normally holds, or by making random modifications to the logic cone near the suspect gate. After the modification is made, the test suite is applied to the circuit to determine if the changes have impacted the desired behavior of the circuit. If the test suite is satisfied, then the modification has had no functional effect on the circuit. This outcome represents a variant of the circuit that is as correct as the original circuit, but that has modified the unintentional, and perhaps Trojan, behavior of the design.

b: HARDWARE REDUNDANCY AND TOLERANCE

Solutions for fault tolerance have been leveraged to combat hardware Trojans. A triple modular redundant (TMR) solution for Trojan tolerance in FPGAs is proposed in [29]. A comparator circuit is initially used to monitor the output of two identical instances of the design. Any discrepancies in the outputs are flagged and subsequently enable a third instance of the design. Majority voting of the output provides tolerance to any potential Trojan activation. In [201] an optimization algorithm is used to explore dual modular redundancy (DMR) architectures through high level synthesis. The algorithm ensures that similar operations in the DMR operations are assigned to hardware resources from different vendors. The authors in [29] note that an attacker aware of this solution can duplicate the Trojan in multiple instances. As a result, they suggest using structurally or architecturally diverse, but functionally equivalent instances. In this way, it becomes more difficult for an attacker to disrupt both instances in the same manner.

Bobda et al. propose running 3PIP cores in sandboxes and providing them with isolated resources to prevent them from accessing other portions of the design [202]. A run-time property checker is used to detect deviations from expected behavior of the 3PIP that may indicate the presence of a Trojan. This work is extended in [203] with a high-level synthesis approach for implementing the sandboxes.

Several approaches employ modified high level synthesis flows to create Trojan resilient output designs. A high-level synthesis flow that minimizes unused circuit area and the number of nets carrying security critical information is

TABLE 14. Summary of obfuscation pre-silicon Trojan prevention techniques.

Approach	Method	Tested on Trojans	Evaluation Criteria	Results
HARPOON [198]	State Space Obfuscation	✓	Reduction in successful Trojan insertion and Trojan effects	53.50% benign Trojans, 75.90% false nodes 12.22% increase in Trojan coverage
AND/OR Locking [194]	Logic Locking	✗	Reduction of circuit-wide transition probability	Up to 36% reduction in rare "0" and 36% reduction in rare "1" signals
Zhang et al. [195]	Logic Locking	✗	Reduction of circuit-wide transition probability	78% reduction in rare nets and introduction of fake rare
ESCALATION [196]	Logic Locking	✓	Reduction of maximum shortest paths & increase detection probability	Increase delay based detection probability by 34%
SWAN [197]	Logic Redaction	✓	Resilience to random and analytical bit flipping	From 2% random and <1% analytical attack success

TABLE 15. Summary of patching pre-silicon Trojan prevention techniques.

Approach	Method	Tested Abstraction	Trigger Comb.	Seq.	Always On	Payload† DoS	Functional Change	Leakage	Degrad.	Advanced Trojan Architectures†
Bobda et al. [202]	Sandboxing	FPGA	✓	✓	✓	✓				
Hamlet et al. [84]	Targeted Modification	ASIC	✓	✓	✓			✓	✓	MOLES [4]
Bobda et al. [203]	Sandboxing	FPGA	✓	✓	✓	✓				
DFFC [204]	Path Delay	ASIC	✓						✓	
EnSAFe [205]	Security Monitor	FPGA	✓	✓		✓		✓		

† refers to tested Trojans

proposed in [206]. This technique involves annotating the high-level circuit description to identify those functions and variables that are security critical and those that are not. During synthesis the security critical portions of the circuit are "dispersed" so that a Trojan must tap into multiple locations within the circuit to obtain the critical information. For instance, the key for a block cipher may be dispersed so that the full key does not exist on a single bus, but rather is spread across several distinct buses.

TL-HLS is a high-level synthesis approach for inserting dual modular redundancy into CDFGs to allow for test and run-time Trojan detection [207]. This technique specifically addresses Trojans inserted in 3PIP, and requires that functionally equivalent implementations of the 3PIP modules are procured from distinct vendors. In [208], a related HLS technique is proposed to protect against Trojans that are distributed across several 3PIP cores. This approach requires that interdependent 3PIP are acquired from different vendors to reduce the likelihood of collusion amongst the 3PIP to implement a Trojan. This technique also uses DMR for test and run-time Trojan detection.

c: HARDWARE PATCHING

Hardware patching is an architectural solution that supports in-field updates or patches to be applied to fix any bugs discovered after hardware deployment [209], [210]. Trojan-specific patching has been presented in [204] and [205]. A hardware patch solution for Trojan induced timing errors is presented in [204]. The patching is realized through dynamic flip flop conversion (DFFC) which is a structure that can detect potential timing violations and lets the late data through. The patch is applied to paths that are long enough and have at least one suspicious SCOAP value, with false paths being removed via test pattern generation. EnSAFe achieves patchability through eFPGAs [205]. Security status monitors are placed on system or inter-IP communications buses. These collect and send data to a central monitoring engine. The central engine contains remotely upgradeable security policies. Upon detecting a violation, the central engine takes action to correct or mitigate the issue. Experimental demonstrations show detection and prevention of leakage and denial of service Trojans. A summary comparison of patching approaches that evaluate against Trojan insertion are presented in Table 15.

C. SPLIT MANUFACTURING

Split manufacturing is a technique in which some of the routing is "lifted" from the lower, front end of line (FEOL) metal layers to the higher, back end of line (BEOL) metal layers [211], [212], [213], [214]. As the BEOL can typically use less advanced technology nodes, this permits the FEOL processing to be performed by modern, untrusted commercial foundries and the BEOL to be completed by a less advanced but trusted foundry [215]. Split manufacturing withholds some design information from the commercial foundry, complicating reverse engineering and Trojan insertion. Trojan prevention techniques build on traditional split manufacturing approaches by inserting both dummy wires and dummy logic simultaneously with the process of identifying which wires to lift from the FEOL to the BEOL metal layers. We acknowledge the vast literature on split manufacturing [216], but only discuss works with significant discussion or experimentation on the problem of Trojans.

Imeson et al. [211] discuss split manufacturing for 3D ICs where wire lifting is performed between IC stack layers. More interestingly, the authors introduce the *k-security* metric, which measures the likelihood associated with a targeted modification attack. Wire lifting in their approach is done such that a targeted *k-secure* metric is achieved. The authors of [213] propose combining split manufacturing with camouflaging the vertical interconnects in 3D integrated ICs. A variety of methods for partitioning layouts into two groups of gates are considered, along with discussion of the resulting tradeoffs between power, performance, area and security, with the security metric relating to the number of vertical interconnections in the resulting 3D IC.

It has been shown that proximity and simulated annealing attacks can undermine the security of split manufacturing [214]. Yang et al. [214] suggest incorporating global wire length information to swap gate pairs that are not in candidate locations such that there is a minimal effect on the wire length. More recently limited programmable gates (LPGs), which are lightweight, less-flexible LUTs, were proposed to aid in achieving k-security [217]. During synthesis, gates will be mapped to these LPGs, which are used to increase the isomorphism available in the design and reduce the need to hide wires.

D. METRICS

The Trojan prevention literature has introduced several metrics for measuring aspects of a design that are vulnerable to Trojan insertion.

1) MEASURE OF VULNERABILITY TO TROJAN INSERTION

Vulnerability to Trojan insertion is the most common form of Trojan prevention measure. Many of these measures calculating vulnerability by noting that the opposite of prevention is the allowance of insertion. The specific calculations measuring vulnerability depend on the level of abstraction, but the reoccurring theme is an attempt to measure which resources are available to an attacker for crafting a stealthy Trojan.

a: TROJAN VULNERABILITY AT RTL

The authors of [218] present metrics for vulnerability to Trojan insertion at RTL. At each line of RTL, a statement weight (S_W) and hardness (S_H) are calculated according to Equation 3 where U is the current upper limit, L the current lower limit, U_O the original upper limit, and L_O, the original lower limit

$$S_W = \left(\frac{U - L + 1}{U_O - L_O + 1}\right) \quad S_H = \frac{1}{S_W} \tag{3}$$

These upper and lower limits refer to the range of data a variable can take and are calculated as the code is traversed. A line of code with high statement hardness, which corresponds to low statement weight, is considered difficult to execute and hence vulnerable to selection for Trojan insertion.

For observability analysis a data flow graph is first constructed with the edge weights (w) between two vertices calculated as the sum of the S_W of each assign statement with the source node on the left hand side and the destination node as the right hand side of the assign statement. Observability ($O(e_0, e_n)$) from one edge to another is then calculated according to Equation 4 for a set of ordered nodes $E = \{e_0, \ldots, e_n\}$ in the data flow graph with e_0 the source node, e_n the destination node, and PO the set of primary outputs according to

$$O(e_0, e_n) = \begin{cases} e_n \notin PO & 0 \\ e_n \in PO & \prod_{i=0}^{n-1} w(e_i, e_{i+1}) \end{cases} \tag{4}$$

b: TROJAN VULNERABILITY AT GATE-LEVEL

Salmani et al. express Trojan vulnerability at gate-level as the presence of nets that are likely to contribute to a hardware Trojan [219]. These nets are those that have suspiciously low switching, are not on the critical path, and that cannot propagate faults. The set of vulnerable nets (V) is summarized by Equation 5

$$V = \{e \in E \,|\, Power(e) < \theta_p \wedge Delay(e) > \theta_t \wedge e \notin T\} \tag{5}$$

where E is the set of all wires in a design, θ_p is the threshold for suspicious switching activity, θ_t is the critical path delay, and T is set of wires with testable faults. *Power* and *Delay* compute the switching activity and delay of the wire, respectively. More vulnerable designs have a larger number of vulnerable nets.

c: TROJAN VULNERABILITY AT LAYOUT-LEVEL

Trojan insertion at the layout-level requires physical space to insert Trojan gates and routing, and nets appealing for Trojan construction. A layout-level vulnerability metric that considers 1) the available space and routing, 2) number of non-critical routing paths, and 3) number of nets with low switching activity is proposed in [220]. Equation 6 measures the vulnerability of a region (r) as the available white space (WS) multiplied by the unused routing (UR)

$$V(r) = WS(r) \times UR(r) \tag{6}$$

Equation 7 measures the vulnerability to delay and power sensitive Trojans by multiplying the available physical space vulnerability ($V(r)$) by the number of nets that have low switching and that are not on the critical path ($N_{NC\&LP}(r)$). Note that Equation 7 can be modified to separately consider Trojans that have low switching, or that have no delay impacts.

$$V_{dp}(r) = V(r) \times N_{NC\&LP}(r) \tag{7}$$

From equations 6 and 7, we might conclude that the larger the presence of available, attractive resources for Trojan insertion, the more vulnerable the design is to Trojan insertion.

From the split manufacturing perspective, the most commonly used metric is *k-security* [211]. A design is said to be *k-secure* if all gates are indistinguishable from at least k other candidate gates.

d: IC ATTACK SURFACE

IC Attack Surface (ICAS) [221] is an evaluation framework for assessing vulnerability to or feasibility of Trojan attacks at the layout-level. It computes three metrics: Trigger Space, Net Blockage and Route Distance. Trigger space is a measure of the available contiguous placement sites, with higher weight associated with more connected sites. For net N blockage is a measure of the congestion within some distance from N on the current layer and the layers immediately above and below N. Finally, route distance computes the minimum distance from an available space through unblocked sites.

e: FSM TROJAN VULNERABILITY

A metric for vulnerability to FSM Trojans is presented in [40]. Equation 8 computes the number of dangerous don't care states (DDCS) as a fraction of the total number of FSM transitions. DDCS are defined as don't care states with access to a critical state. Therefore, this metric considers Trojans that can potentially modify the FSM and FSM transitions

$$V_{FSM} = \frac{|DDCS|}{Total \quad Transitions} \tag{8}$$

f: TROJAN VULNERABILITY FACTOR

Trojan Vulnerability Factor (TVF) is a measure of a circuit's vulnerability to Trojan insertion [13]. TVF looks at vulnerability from the perspective of maximal cliques in a satisfiability graph, which is a data structure that abstractly represents the potential Trojan population for a given design. A maximal clique of size n, in this case, is equivalent to an n-trigger Trojan subpopulation. All derivative Trojans ($\sum_{i=2}^{n} \binom{n}{i}$) that can be crafted by the rare nets in the maximal clique can be activated by the same test vector that activates the maximal clique. Therefore, TVF is defined as

$$TVF = \sum_{\forall c \in TMC} |c_i| \tag{9}$$

where c is a maximal clique, and TMC is the set of maximal cliques of a target size. The target size of a clique can be selected to represent Trojans of a particular activation probability or stealthiness. In this way, TVF suggests that design vulnerability to Trojan insertion is subject to the number of and size of the maximal cliques.

A derivative metric proposed in [13] is maximal clique participation (MCP). This is a net-level metric defined by Equation 10, where MC is the set of all maximal cliques and e is the current net under calculation. A net that has higher participation in maximal cliques has a higher probability of participating in an attacker crafted Trojan trigger

$$MCP = \frac{\sum_{\forall c \in MC} |e \cap c_i|}{|MC|} \tag{10}$$

2) MEASURING TROJAN RESILIENCE OF LOCKING APPROACHES

To measure the effectiveness of logic locking as a defense against Trojan insertion Cruz et al. propose two metrics, CTR and FTR [222]. Coarse-grained Trojan Resilience (CTR) is a measure of the shift in population between suspect and benign nets when comparing the locked (l) and unlocked (u) modes of a logic locked design. In this case, suspect (Γ) and benign (N) are states of a net that refer to the likelihood of a net contributing to Trojan construction or the original design, respectively. Moreover, CTR can be applied to signal probability, controllability, observability, and other net features

$$CTR = \left(1 - \frac{\Gamma_l \cup \Gamma_u}{\Gamma_l}, 1 - \frac{N_l \cup N_u}{N_l}\right) \tag{11}$$

CTR, as shown in Equation 11, is expressed as the value pair (CTR_Γ, CTR_N), where CTR_Γ refers to the ratio of nets that remained suspect in locked versus unlocked modes. CTR_N is a similar ratio, but refers to the change in normal net population between locked and unlocked modes. In an ideal case, logic locking is expected to confuse the attacker and achieve enough feature level transformations such that 50% of the population of nets that are suspect in the locked design were originally benign.

Fine-grained Trojan Resilience (FTR) is a supplementary metric to CTR and provides the average magnitude of change where $\mathcal{E}_l$ is the relevant feature value of the corresponding wire in the locked netlist, regardless of whether it is benign or suspect

$$FTR = \left(\frac{\sum_{i=1}^{x} |\Gamma_{u_i} - \mathcal{E}_{l_i}|}{x}, \frac{\sum_{i=1}^{y} |N_{u_i} - \mathcal{E}_{l_i}|}{y}\right) \tag{12}$$

3) TRIGGER AVOIDANCE RATE

To evaluate software-based prevention approaches, the trigger avoidance rate (TAR) metric is proposed [223], [224]. From an assumed population of Trojans (T) that could be triggered by a sequence of opcodes from an original software code, TAR is defined as

$$1 - \frac{T_{activ}}{T} \tag{13}$$

where T_{activ} is the set of Trojan triggers still present or activated via simulation in the software. A slight modification of trigger avoidance rate is proposed in [224]. Here, TAR is a measure between two functionally equivalent software programs ($v1$, $v2$)

$$TAR = \frac{T_{v1} - T_{v2}}{T_{v1}} \tag{14}$$

Note, the Trojans under evaluation are those triggered by a sequence of opcodes and operands. From Equations 13 and 14, the defender's goal is to achieve 100% trigger avoidance rate. Similar to trigger and Trojan coverage, it is important to clearly outline the assumed Trojan population when using this metric.

E. PERSPECTIVE AND OUTLOOK

As it may be impossible to prevent attackers from inserting a Trojan during design or given physical access to a chip, defenders should instead focus on discouraging Trojan insertion by

- making it difficult to ascertain the underlying functionality of the design

- making Trojans easily detectable
- making Trojan insertion prohibitively costly
- tolerating errant behavior

We note, it is difficult to prevent an attacker with physical access to a chip from making modifications to the circuitry within the IC. While several approaches that attempt to prevent such modifications are described in Section V, it should not be assumed that these are sufficient to prevent attacks by well-resourced adversaries. Consequently, these techniques should be augmented by detection based approaches, such as those introduced in Section IV to identify attack attempts as they occur and to permit an appropriate response, such as zeroizing memory.

1) POTENTIAL PITFALLS

The surveyed prevention countermeasures do not completely remove the possibility of a Trojan. Rather, they play a game of probabilities. Many of these techniques introduce design changes that attempt to skew the probability in the defender's favor. However, there are often overlooked points or conservative assumptions made by these countermeasures.

a: REMOVING RARE NODES

As many Trojan triggering methods utilize rare nodes in the design to construct the trigger, one technique for preventing Trojan insertion is to attempt to deny the attacker opportunity to construct a trigger by eliminating rare nodes [171]. Unfortunately, there are two fundamental problems with this approach. First, assuming it is possible to completely remove all rare nets we have still not prevented Trojan triggers, as the attacker has options such as adding logic to the design to construct the trigger or forming a counter-based trigger using the non-rare nodes. Second, it is difficult to define what constitutes rareness of a node, as any value chosen for this is inherently arbitrary. For instance, if 0.1 is selected as a threshold for rareness, it seems reasonable that 0.11 should also be considered rare [13], which eventually leads to the conclusion that 0.5 is the only non-rare value. As such, in practice it is infeasible to eliminate rare nodes from a design, and so an attacker will always have opportunity to identify and utilize rare nets [9], [13].

b: INTRODUCED NEGATIVE STATE SPACE

State space transformation and obfuscation, and other Trojan prevention techniques based on modifying or introducing new sequential behavior, introduce *negative state space* in the form of unspecified functionality in the expanded state space. This unspecified functionality may be used by an attacker to insert state-transition based Trojans [225]. By definition, these solutions introduce an intractably large negative state space. This makes it infeasible to validate the circuit functionality when the design is in the negative space, and so it cannot be guaranteed that Trojans have not been inserted into this space.

c: PROTECTING PREVENTION TECHNIQUES

Several Trojan prevention techniques make architectural or design-level modifications to a circuit. Amongst these are approaches that incorporate design for security and test techniques, as well as various logic-locking approaches and many state-space obfuscation techniques. While some of these approaches can prevent or tolerate most observable harm, they can also introduce identifiable artifacts and structures that can then be bypassed or removed by an attacker, hindering the effectiveness of the Trojan prevention solution [55]. Defenders should be mindful of this possibility when applying such solutions and should include in their threat models adversaries that have significant reverse engineering capability. We specifically caution readers regarding logic locking and obfuscation, as this field has seen rapid defense-to-attack lifecycles of defenses claimed to be provably secure.

VI. CONCLUSION

In recent decades, hardware Trojans have received considerable interest from the research community, which has investigated the design, detection, and prevention of hardware Trojans. Much of the effort in Trojan design has assumed that an attacker has access to inject the Trojan at design time, typically either in RTL or netlists. Such insertions may be most easily accomplished through malicious 3PIP, though other mechanisms, such as intentional subversion by an insider, are possible. Many detection approaches that operate on RTL or netlists have subsequently been developed. However, Trojans can be inserted at any stage of the design or even manufacturing process, and so detection techniques specifically targeting Trojans introduced into GDS-II files by an untrusted foundry have also been explored. These techniques are often based on side-channel analysis, though in principle the netlist based detection techniques can also be used to detect this class of Trojans if the considerable effort required to extract a netlist from silicon is undertaken. A variety of Trojan prevention techniques have also been suggested. Many of these techniques attempt to deny the attacker opportunity to insert a Trojan, either by eliminating access to attractive trigger wires by increasing their controllability, filling the space in the layout that could be used for placing Trojan gates, or adding monitors to detect Trojan activity on or near sensitive wires. Other approaches, such as split manufacturing, attempt to prevent the attacker from gaining enough knowledge about the functionality of the circuit to insert a Trojan.

Hardware Trojan design and defense is no exception to the cat-and-mouse game of security. Defenders have the significant challenge of detecting any possible Trojan, and so it is likely that Trojan design techniques that evade existing detection approaches will be developed. While more advanced and specialized detection approaches may be introduced to counter these, it might be expected that Trojan development will tend to outpace detection capabilities.

TABLE 16. Glossary of common acronyms and terms.

Acronym/Term	Description
3PIP	Third Party IP
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BEOL	Back End of Line
BISA	Built In Self Authentication
CAD	Computer Aided Design
CED	Concurrent Error Detection
CFG	Control Flow Graph
CDFG	Control and Dataflow Graph
Comb.	Combinational
CPU	Central Processing Unit
CV	Control Value
DAG	Directed Acyclic Graph
Degrad.	Degradation
DFG	Dataflow Graph
DfS	Design for Security
DfT	Design for Test
DMR	Double Modular Redundancy
DoS	Denial of Service
DS	Dark Silicon
ECO	Engineering Change Order
EDA	Electronic Design Automation
FEOL	Front End of Line
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GAN	Generative Adversarial Network
GDS-II	Graphic Design System
GNN	Graph Neural Network
Golden Design	Reference design format assumed to be correct
HDL	Hardware Description Language
HLS	High level Synthesis
IC	Integrated Circuit
IFT	Information Flow Tracking
IP	Intellectual Property
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look up Table
ML	Machine Learning
MOLES	Malicious Off-chip Leakage Enabled by Side-channels [4]
MUX	Multiplexer
Netlist	HDL format consisting of logic primitives connected by wires
ODC	Observability Don't Care
PAR	Place and Route
PCB	Printed Circuit Board
PRNG	Pseudo-Random Number Generator
RE	Reverse Engineering
RL	Reinforcement Learning
RTL	Register Transfer Level
SAT	Satisfiability
SCOAP	Sandia Controllability and Observability Analysis Program [112]
SDC	Satisfiability Don't Care
Seq.	Sequential
SMT	Satisfiability Modulo Theories
SMV	Symbolic Model Verification
SoC	System on Chip
Spec	Design Specification
SRAM	Static Random Access Memory
SVM	Support Vector Machine
TMR	Triple Modular Redundancy
μP	Microprocessor

Moreover, moving forward, defenders should pay mind to the scope of the problem. As demonstrated by the breadth of this survey and the evolving nature of the field, practitioners are cautioned against developing a false sense of security from newly proposed defenses. Due to this, we suggest the following:

- Using clear language and terminology when describing what Trojan types are considered by a threat model, or addressed by a countermeasure.
- Development of a more representative Trojan benchmark database or benchmarking framework that includes state-of-the-art Trojan designs to support evaluation of defensive techniques.
- The development of layered defensive strategies that combine several complementary detection and prevention approaches, both pre- and post-silicon, for ensuring Trojan coverage and high levels of design assurance.

APPENDIX
GLOSSARY

See Table 16.

ACKNOWLEDGMENT

This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan.

REFERENCES

[1] S. Adee, "The hunt for the kill switch," *IEEE Spectr.*, vol. 45, no. 5, pp. 34–39, May 2008.

[2] J. Zhang, F. Yuan, and Q. Xu, "DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 153–166.

[3] N. Zhang, Z. Lv, Y. Zhang, H. Li, Y. Zhang, and W. Huang, "Novel design of hardware trojan: A generic approach for defeating testability based detection," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 2020, pp. 162–173.

[4] L. Lin, W. Burleson, and C. Paar, "MOLES: Malicious off-chip leakage enabled by side-channels," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design-Dig. Tech. Papers*, Nov. 2009, pp. 117–122.

[5] S. Bhunia and M. Tehranipoor, *The Hardware Trojan War*. Cham, Switzerland: Springer, 2018.

[6] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill, "Ten years of hardware trojans: A survey from the attacker's perspective," *IET Comput. Digit. Techn.*, vol. 14, no. 6, pp. 231–246, Nov. 2020.

[7] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 18–37.

[8] H. Salmani, "COTD: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 2, pp. 338–350, Feb. 2017.

[9] S. Dupuis, M.-L. Flottes, G. Di Natale, and B. Rouzeyre, "Protection against hardware trojans with logic testing: Proposed solutions and challenges ahead," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 35, no. 2, pp. 73–90, Apr. 2018.

[10] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *J. Hardw. Syst. Secur.*, vol. 1, no. 1, pp. 85–102, Mar. 2017.

[11] J.-F. Gallais, J. Großschädl, N. Hanley, M. Kasper, M. Medwed, F. Regazzoni, J.-M. Schmidt, S. Tillich, and M. Wójcik, "Hardware trojans for inducing or amplifying side-channel leakage of cryptographic software," in *Proc. 2nd Int. Conf. Trusted Syst.*, Beijing, China. Cham, Switzerland: Springer, Dec. 2011, pp. 253–270.

[12] P. Dash, C. Perkins, and R. Gerdes, "Remote activation of hardware trojans via a covert temperature channel," in *Proc. 11th EAI Int. Conf. Secur. Privacy Commun. Networks*, Dallas, TX, USA. Cham, Switzerland: Springer, Oct. 2015, pp. 294–310.

[13] J. Cruz, P. Slpsk, P. Gaikwad, and S. Bhunia, "TVF: A metric for quantifying vulnerability against hardware trojan attacks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 7, pp. 969–979, Jul. 2023.

[14] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[15] J. Cruz, Y. Huang, P. Mishra, and S. Bhunia, "An automated configurable trojan insertion framework for dynamic trust benchmarks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1598–1603.

[16] Y. Huang, S. Bhunia, and P. Mishra, "MERS: Statistical test generation for side-channel analysis based trojan detection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 130–141.

[17] W. Hu, L. Zhang, A. Ardeshiricham, J. Blackstone, B. Hou, Y. Tai, and R. Kastner, "Why you should care about don't cares: Exploiting internal don't care conditions for hardware trojans," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 707–713.

[18] Z. Zhou, U. Guin, and V. D. Agrawal, "Modeling and test generation for combinational hardware trojans," in *Proc. IEEE 36th VLSI Test Symp. (VTS)*, Apr. 2018, pp. 1–6.

[19] V. Jyothi, P. Krishnamurthy, F. Khorrami, and R. Karri, "TAINT: Tool for automated insertion of trojans," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 545–548.

[20] D. Stefan, C. Mitchell, and C. G. Almenar, "Trojan attacks for compromising cryptographic security in fpga encryption systems," Tech. Rep. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=058241cbd64481f1a4ca8dfcfbee0160c318bfb1

[21] A. Baumgarten, M. Clausman, B. Lindemann, M. Steffen, B. Trotter, and J. Zambreno, "Embedded systems challenge," Tech. Rep. [Online]. Available: https://scholar.google.com/scholar_lookup?&title=Embedded%20Systems%20Challenge%20%28Iowa%20State%20University%29&publication_year=2008&author=Baumgarten%2CA

[22] N. G. Tsoutsos, C. Konstantinou, and M. Maniatakos, "Advanced techniques for designing stealthy hardware trojans," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–4.

[23] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards trojan-free trusted ICs: Problem analysis and detection scheme," in *Proc. Design, Autom. Test Eur.*, Mar. 2008, pp. 1362–1365.

[24] J. Rajendran, A. M. Dhandayuthapany, V. Vedula, and R. Karri, "Formal security verification of third party intellectual property cores for information leakage," in *Proc. 29th Int. Conf. VLSI Design 15th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2016, pp. 547–552.

[25] C. Krieg, C. Wolf, and A. Jantsch, "Malicious LUT: A stealthy FPGA trojan injected and triggered by the design flow," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2016, pp. 1–8.

[26] K. Basu, S. M. Saeed, C. Pilato, M. Ashraf, M. T. Nabeel, K. Chakrabarty, and R. Karri, "CAD-base: An attack vector into the electronics supply chain," *ACM Trans. Design Autom. Electron. Syst.*, vol. 24, no. 4, pp. 1–30, Jul. 2019.

[27] K. Basu and P. Mishra, "Efficient trace signal selection for post silicon validation and debug," in *Proc. 24th Int. Conf. VLSI Design*, Jan. 2011, pp. 352–357.

[28] M. Xue, R. Bian, W. Liu, and J. Wang, "Defeating untrustworthy testing parties: A novel hybrid clustering ensemble based golden models-free hardware trojan detection method," *IEEE Access*, vol. 7, pp. 5124–5140, 2019.

[29] S. Mal-Sarkar, R. Karam, S. Narasimhan, A. Ghosh, A. Krishna, and S. Bhunia, "Design and validation for FPGA trust under hardware trojan attacks," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 3, pp. 186–198, Jul. 2016.

[30] C. Pilato, K. Basu, F. Regazzoni, and R. Karri, "Black-hat high-level synthesis: Myth or reality?" *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 4, pp. 913–926, Apr. 2018.

[31] J. Zhang and Q. Xu, "On hardware trojan design and implementation at register-transfer level," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 107–112.

[32] J. Cruz, C. Posada, N. V. R. Masna, P. Chakraborty, P. Gaikwad, and S. Bhunia, "A framework for automated exploration of trojan attack space in FPGA netlists," *IEEE Trans. Comput.*, vol. 72, no. 10, pp. 2740–2751, Oct. 2023.

[33] T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 655–660.

[34] N. Albartus, M. Hoffmann, S. Temme, L. Azriel, and C. Paar, "DANA universal dataflow analysis for gate-level netlist reverse engineering," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 309–336, Aug. 2020.

[35] T. Perez, M. Imran, P. Vaz, and S. Pagliarini, "Side-channel trojan insertion—A practical foundry-side attack via ECO," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[36] A. Bhattacharyay, S. Yang, J. Cruz, P. Chakraborty, S. Bhunia, and T. Hoque, "An automated framework for board-level trojan benchmarking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 2, pp. 397–410, Feb. 2023.

[37] D. Mehta, H. Lu, O. P. Paradis, M. A. M. S., M. T. Rahman, Y. Iskander, P. Chawla, D. L. Woodard, M. Tehranipoor, and N. Asadizanjani, "The big hack explained: Detection and prevention of PCB supply chain implants," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 16, no. 4, pp. 1–25, Oct. 2020.

[38] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[39] H. Zhu, X. Guo, Y. Jin, and X. Zhang, "PCBench: Benchmarking of board-level hardware attacks and trojans," in *Proc. 26th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2021, pp. 396–401.

[40] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[41] R. Dai and T. Yavuz, "A symbolic approach to detecting hardware trojans triggered by don't care transitions," *ACM Trans. Design Autom. Electron. Syst.*, vol. 28, no. 2, pp. 1–31, Mar. 2023.

[42] N. Fern, I. San, Ç. K. Koç, and K. T. Cheng, "Hiding hardware trojan communication channels in partially specified SoC bus functionality," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1435–1444, Sep. 2017.

[43] X. Wang, T. Hoque, A. Basak, R. Karam, W. Hu, M. Qin, D. Mu, and S. Bhunia, "Hardware trojan attack in embedded memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 1, pp. 1–28, Jan. 2021.

[44] M. N. I. Khan, A. De, and S. Ghosh, "Cache-out: Leaking cache memory using hardware trojan," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 6, pp. 1461–1470, Jun. 2020.

[45] G. A. Chacon, C. Williams, J. Knechtel, O. Sinanoglu, and P. V. Gratz, "Hardware trojan threats to cache coherence in modern 2.5D chiplet systems," *IEEE Comput. Archit. Lett.*, vol. 21, no. 2, pp. 133–136, Jul. 2022.

[46] N. Fern, S. Kulkarni, and K.-T.-T. Cheng, "Hardware trojans hidden in RTL don't cares—Automated insertion and prevention methodologies," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2015, pp. 1–8.

[47] C. Wang, Y. Cai, and Q. Zhou, "HLIFT: A high-level information flow tracking method for detecting hardware trojans," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 727–732.

[48] C. Krieg, C. Wolf, A. Jantsch, and T. Zseby, "Toggle MUX: How X-optimism can lead to malicious hardware," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[49] Y. Zhang, M. Ge, X. Chen, J. Yao, and Z. Mao, "Blinding HT: Hiding hardware trojan signals traced across multiple sequential levels," *IET Circuits, Devices Syst.*, vol. 16, no. 1, pp. 105–115, Jan. 2022.

[50] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating UCI: Building stealthy and malicious hardware," in *Proc. IEEE Symp. Secur. Privacy*, May 2011, pp. 64–77.

[51] S. K. Haider, C. Jin, and M. van Dijk, "Advancing the state-of-the-art in hardware trojans design," in *Proc. 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, 2017, pp. 823–826.

[52] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 159–172.

[53] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using Boolean functional analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 697–708.

[54] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "VeriTrust: Verification for hardware trust," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, May 2013, pp. 1–8.

[55] A. Jain, Z. Zhou, and U. Guin, "TAAL: Tampering attack on any key-based logic locked circuits," *ACM Trans. Design Autom. Electron. Syst.*, vol. 26, no. 4, pp. 1–22, Jul. 2021.
[56] Y. Liu, A. Jayasena, P. Mishra, and A. Srivastava, "Logic locking based trojans: A friend turns foe," 2023, *arXiv:2309.15067*.
[57] R. Karam, T. Hoque, S. Ray, M. Tehranipoor, and S. Bhunia, "Robust bitstream protection in FPGA-based systems through low-overhead obfuscation," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–8.
[58] T. Reece, D. B. Limbrick, X. Wang, B. T. Kiddie, and W. H. Robinson, "Stealth assessment of hardware trojans in a microcontroller," in *Proc. IEEE 30th Int. Conf. Comput. Design (ICCD)*, Sep. 2012, pp. 139–142.
[59] N. G. Tsoutsos and M. Maniatakos, "Fabrication attacks: Zero-overhead malicious modifications enabling modern microprocessor privilege escalation," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 1, pp. 81–93, Mar. 2014.
[60] A. De, M. Nasim Imtiaz Khan, K. Nagarajan, and S. Ghosh, "HarTBleed: Using hardware trojans for data leakage exploits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 968–979, Apr. 2020.
[61] A. Damljanovic, A. Ruospo, E. Sanchez, and G. Squillero, "A benchmark suite of RT-level hardware trojans for pipelined microprocessor cores," in *Proc. 24th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2021, pp. 51–56.
[62] S. Tarek, H. A. Shaikh, S. R. Rajendran, and F. Farahmandi, "Benchmarking of SoC-level hardware vulnerabilities: A complete walkthrough," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jun. 2023, pp. 1–6.
[63] I. H. Abbassi, F. Khalid, S. Rehman, A. M. Kamboh, A. Jantsch, S. Garg, and M. Shafique, "TrojanZero: Switching activity-aware design of undetectable hardware trojans with zero power and area footprint," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 914–919.
[64] F. Regazzoni, C. Alippi, and I. Polian, "Security: The dark side of approximate computing?" in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Jun. 2018, pp. 1–6.
[65] Q. A. Ahmed, M. Awais, and M. Platzner, "MAAS: Hiding trojans in approximate circuits," in *Proc. 24th Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2023, pp. 1–6.
[66] M. M. Alam, A. Nahiyan, M. Sadi, D. Forte, and M. Tehranipoor, "Soft-HaT: Software-based silicon reprogramming for hardware trojan implementation," *ACM Trans. Design Autom. Electron. Syst.*, vol. 25, no. 4, pp. 1–22, Jul. 2020.
[67] K. Dharsee and J. Criswell, "Jinn: Hijacking safe programs with trojans," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 6965–6982.
[68] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," 2018, *arXiv:1806.05768*.
[69] J. Clements and Y. Lao, "Hardware trojan design on neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.
[70] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.
[71] S. Yao, X. Chen, J. Zhang, Q. Liu, J. Wang, Q. Xu, Y. Wang, and H. Yang, "FASTrust: Feature analysis for third-party IP trust verification," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2015, pp. 1–10.
[72] X. Chen, Q. Liu, S. Yao, J. Wang, Q. Xu, Y. Wang, Y. Liu, and H. Yang, "Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 7, pp. 1370–1383, Jul. 2018.
[73] M. Fyrbiak, S. Wallat, P. Swierczynski, M. Hoffmann, S. Hoppach, M. Wilhelm, T. Weidlich, R. Tessier, and C. Paar, "HAL—The missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 3, pp. 498–510, May 2019.
[74] J. Yang, Y. Zhang, Y. Hua, J. Yao, Z. Mao, and X. Chen, "Hardware trojans detection through RTL features extraction and machine learning," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2021, pp. 1–4.
[75] H. Wang, Q. Zhou, and Y. Cai, "Static probability analysis guided RTL hardware trojan test generation," in *Proc. 28th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2023, pp. 510–515.
[76] F. Demrozi, R. Zucchelli, and G. Pravadelli, "Exploiting sub-graph isomorphism and probabilistic neural networks for the detection of hardware trojans at RTL," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, Oct. 2017, pp. 67–73.
[77] L. Piccolboni, A. Menon, and G. Pravadelli, "Efficient control-flow subgraph matching for detecting hardware trojans in RTL models," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–19, Oct. 2017.
[78] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1691–1696.
[79] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, and M. Tehranipoor, "Hardware trojan detection through information flow security verification," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2017, pp. 1–10.
[80] A. Ahmed, F. Farahmandi, Y. Iskander, and P. Mishra, "Scalable hardware trojan activation by interleaving concrete simulation and symbolic execution," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–10.
[81] T. Hoque, J. Cruz, P. Chakraborty, and S. Bhunia, "Hardware IP trust validation: Learn (The untrustworthy), and verify," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–10.
[82] F. Zareen and R. Karam, "Detecting RTL trojans using artificial immune systems and high level behavior classification," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2018, pp. 68–73.
[83] C. Wang, Y. Cai, Q. Zhou, and H. Wang, "ASAX: Automatic security assertion extraction for detecting hardware trojans," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 84–89.
[84] J. R. Hamlet, J. R. Mayo, and V. G. Kammler, "Targeted modification of hardware trojans," *J. Hardw. Syst. Secur.*, vol. 3, no. 2, pp. 189–197, Jun. 2019.
[85] Y. Lyu and P. Mishra, "Scalable concolic testing of RTL models," *IEEE Trans. Comput.*, vol. 70, no. 7, pp. 979–991, Jul. 2021.
[86] A. Vafaei, N. Hooten, M. Tehranipoor, and F. Farahmandi, "SymbA: Symbolic execution at C-level for hardware trojan activation," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2021, pp. 223–232.
[87] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque, "GNN4TJ: Graph neural networks for hardware trojan detection at register transfer level," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1504–1509.
[88] R. Yasaei, S. Faezi, and M. A. Al Faruque, "Golden reference-free hardware trojan localization using graph convolutional network," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 10, pp. 1401–1411, Oct. 2022.
[89] A. L. D. Antón, J. Müller, L. Deutschmann, M. R. Fadiheh, D. Stoffel, and W. Kunz, "A golden-free formal method for trojan detection in non-interfering accelerators," 2023, *arXiv:2312.06515*.
[90] R. Hassan, X. Meng, K. Basu, and S. M. P. Dinakarrao, "Circuit topology-aware vaccination-based hardware trojan detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 9, pp. 2852–2862, Sep. 2023.
[91] Z. Pan and P. Mishra, "Hardware trojan detection using Shapley ensemble boosting," in *Proc. 28th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2023, pp. 1–8.
[92] L. Chen, C. Dong, Q. Wu, X. Liu, X. Guo, Z. Chen, H. Zhang, and Y. Yang, "GNN4HT: A two-stage GNN-based approach for hardware trojan multifunctional classification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 1, pp. 172–185, Jan. 2025.
[93] L. Wu, X. Li, J. Zhu, J. Zheng, and W. Hu, "Identifying specious LUTs for satisfiability don't care trojan detection," in *Proc. IEEE 34th Int. Syst. Chip Conf. (SOCC)*, Sep. 2021, pp. 170–175.
[94] N. Fern, I. San, and K. T. Cheng, "Detecting hardware trojans in unspecified functionality through solving satisfiability problems," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 598–504.
[95] Y. Hou, H. He, K. Shamsi, Y. Jin, D. Wu, and H. Wu, "R2D2: Runtime reassurance and detection of A2 trojan," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Apr. 2018, pp. 195–200.
[96] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "T-TER: Defeating A2 trojans with targeted tamper-evident routing," 2019, *arXiv:1906.08842*.
[97] *Chip-level Trojan Benchmarks*. Accessed: Aug. 17, 2023. [Online]. Available: https://www.trust-hub.org/#/benchmarks/chip-level-trojan
[98] *Embedded Security Challenge*. Accessed: Aug. 17, 2023. [Online]. Available: https://www.csaw.io/esc
[99] S. Yu, W. Liu, and M. O'Neill, "An improved automatic hardware trojan generation platform," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 302–307.

[100] J. Cruz, P. Gaikwad, A. Nair, P. Chakraborty, and S. Bhunia, "A machine learning based automatic hardware trojan attack space exploration and benchmarking framework," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2022, pp. 1–6.
[101] C. Bolchini, L. Cassano, I. Montalbano, G. Repole, A. Zanetti, and G. D. Natale, "HATE: A hardware trojan emulation environment for microprocessor-based systems," in *Proc. IEEE 25th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2019, pp. 109–114.
[102] V. Gohil, H. Guo, S. Patnaik, and J. Rajendran, "ATTRITION: Attacking static hardware trojan detection techniques using reinforcement learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 1275–1289.
[103] K. Hasegawa, S. Hidano, K. Nozawa, S. Kiyomoto, and N. Togawa, "R-HTDetector: Robust hardware-trojan detection based on adversarial training," *IEEE Trans. Comput.*, vol. 72, no. 2, pp. 333–345, Feb. 2023.
[104] A. Sarihi, A. Patooghy, P. Jamieson, and A.-H. A. Badawy, "Trojan playground: A reinforcement learning framework for hardware trojan insertion and detection," 2023, *arXiv:2305.09592*.
[105] Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, vols. C-30, no. 3, pp. 215–222, Mar. 1981.
[106] A. Sarihi, P. Jamieson, A. Patooghy, and A.-H.-A. Badawy, "TrojanForge: Generating adversarial hardware trojan examples using reinforcement learning," in *Proc. ACM/IEEE 6th Symp. Mach. Learn. CAD (MLCAD)*, Sep. 2024, pp. 1–7.
[107] A. Hepp, T. Perez, S. Pagliarini, and G. Sigl, "A pragmatic methodology for blind hardware trojan insertion in finalized layouts," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2022, pp. 1–9.
[108] A. Sarihi, A. Patooghy, P. Jamieson, and A.-H.-A. Badawy, "Hardware trojan insertion using reinforcement learning," in *Proc. Great Lakes Symp. VLSI*, Jun. 2022, pp. 139–142.
[109] D. Sullivan, J. Biggers, G. Zhu, S. Zhang, and Y. Jin, "FIGHT-metric: Functional identification of gate-level hardware trustworthiness," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–4.
[110] S. K. Haider, C. Jin, M. Ahmad, D. M. Shila, O. Khan, and M. van Dijk, "Advancing the state-of-the-art in hardware trojans detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 1, pp. 18–32, Jan. 2019.
[111] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "Bomberman: Defining and defeating hardware ticking timebombs at design-time," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 970–986.
[112] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia controllability/observability analysis program," in *Proc. 17th Design Autom.*, Jun. 1980, pp. 190–1980.
[113] A. Bazzazi, M. T. Manzuri Shalmani, and A. M. A. Hemmatyar, "Hardware trojan detection based on logical testing," *J. Electron. Test.*, vol. 33, no. 4, pp. 381–395, Aug. 2017.
[114] T. Le, L. Weaver, J. Di, S. Zhang, and Y. Jin, "Hardware trojan detection and functionality determination for soft IPs," in *Proc. IEEE 3rd Int. Verification Secur. Workshop (IVSW)*, Jul. 2018, pp. 56–61.
[115] W. Hu, A. Ardeshiricham, and R. Kastner, "Hardware information flow tracking," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–39, May 2022.
[116] M. Tiwari, H. M. G. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in *Proc. 14th Int. Conf. Architectural Support Program. Languages Operating Syst.*, Mar. 2009, pp. 109–120.
[117] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 503–516, May 2015.
[118] W. Hu, B. Mao, J. Oberg, and R. Kastner, "Detecting hardware trojans with gate-level information-flow tracking," *Computer*, vol. 49, no. 8, pp. 44–52, Aug. 2016.
[119] J. Wu, F. Fowze, and D. Forte, "EXERT: EXhaustive IntEgRiTy analysis for information flow security," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2022, pp. 1–6.
[120] J. Wu and D. Forte, "EXERTv2: Exhaustive integrity analysis for information flow security with FSM integration," *J. Hardw. Syst. Secur.*, vol. 7, no. 4, pp. 147–164, Dec. 2023.
[121] Y. Jin and Y. Makris, "Proof carrying-based information flow tracking for data secrecy protection and hardware trust," in *Proc. IEEE 30th VLSI Test Symp. (VTS)*, Apr. 2012, pp. 252–257.
[122] M. Qin, W. Hu, D. Mu, and Y. Tai, "Property based formal security verification for hardware trojan detection," in *Proc. IEEE 3rd Int. Verification Secur. Workshop (IVSW)*, Jul. 2018, pp. 62–67.
[123] Q. Zhang, J. He, Y. Zhao, and X. Guo, "A formal framework for gate-level information leakage using Z3," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2020, pp. 1–6.
[124] Z. Liu, O. Arias, W. Fu, Y. Jin, and X. Guo, "Inter-IP malicious modification detection through static information flow tracking," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 600–603.
[125] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik, "Reverse engineering digital circuits using structural and functional analyses," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 1, pp. 63–80, Mar. 2014.
[126] J. Geist, T. Meade, S. Zhang, and Y. Jin, "RELIC-FUN: Logic identification through functional signal comparisons," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[127] S. Rajendran and M. L. Regeena, "A novel algorithm for hardware trojan detection through reverse engineering," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 4, pp. 1154–1166, Apr. 2022.
[128] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware trojan attacks: Recent advances and challenges," *IEEE Access*, vol. 8, pp. 10796–10826, 2020.
[129] P. Gaikwad, J. Cruz, P. Chakraborty, S. Bhunia, and T. Hoque, "Hardware IP assurance against trojan attacks with machine learning and post-processing," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 3, pp. 1–23, Jul. 2023.
[130] K. Hasegawa, K. Yamashita, S. Hidano, K. Fukushima, K. Hashimoto, and N. Togawa, "Node-wise hardware trojan detection based on graph learning," *IEEE Trans. Comput.*, vol. 74, no. 3, pp. 749–761, Mar. 2023.
[131] K. Huang and Y. He, "Trigger identification using difference-amplified controllability and dynamic transition probability for hardware trojan detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3387–3400, 2020.
[132] M. Tebyanian, A. Mokhtarpour, and A. Shafieinejad, "SC-COTD: Hardware trojan detection based on sequential/combinational testability features using ensemble classifier," *J. Electron. Test.*, vol. 37, no. 4, pp. 473–487, Aug. 2021.
[133] P.-Y. Lo, C.-W. Chen, W.-T. Hsu, C.-W. Chen, C.-W. Tien, and S.-Y. Kuo, "Semi-supervised trojan nets classification using anomaly detection based on SCOAP features," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 2423–2427.
[134] H. Salmani, "The improved COTD technique for hardware trojan detection in gate-level netlist," in *Proc. Great Lakes Symp. VLSI*, Jun. 2022, pp. 449–454.
[135] N. Muralidhar, A. Zubair, N. Weidler, R. Gerdes, and N. Ramakrishnan, "Contrastive graph convolutional networks for hardware trojan detection in third party IP cores," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2021, pp. 181–191.
[136] H. Lashen, L. Alrahis, J. Knechtel, and O. Sinanoglu, "TrojanSAINT: Gate-level netlist sampling-based inductive learning for hardware trojan detection," 2023, *arXiv:2301.11804*.
[137] C. Dong, Y. Liu, J. Chen, X. Liu, W. Guo, and Y. Chen, "An unsupervised detection approach for hardware trojans," *IEEE Access*, vol. 8, pp. 158169–158183, 2020.
[138] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, "Hardware trojans classification for gate-level netlists based on machine learning," in *Proc. IEEE 22nd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2016, pp. 203–206.
[139] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[140] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware trojans classification for gate-level netlists using multi-layer neural networks," in *Proc. IEEE 23rd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2017, pp. 227–232.
[141] B. Cakir and S. Malik, "Hardware trojan detection for gate-level ICs using signal correlation based clustering," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 471–476.
[142] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *ACM SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, Jun. 1999.
[143] S.-Y. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. Al Faruque, "HW2 VEC: A graph learning tool for automating hardware security," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2021, pp. 13–23.

[144] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," 2019, *arXiv:1907.04931*.
[145] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "GNN-RE: Graph neural networks for reverse engineering of gate-level netlists," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2435–2448, Aug. 2022.
[146] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware trojan detection," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, Jan. 2009, pp. 396–410.
[147] Y. Lyu and P. Mishra, "Automated test generation for trojan detection using delay-based side channel analysis," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1031–1036.
[148] S. M. Sebt, A. Patooghy, and H. Beitollahi, "An efficient technique to detect stealthy hardware trojans independent of the trigger size," *J. Electron. Test.*, vol. 35, no. 6, pp. 839–852, Dec. 2019.
[149] Y. Lyu and P. Mishra, "Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1287–1300, Jul. 2021.
[150] C. Nigh and A. Orailoglu, "Adatrust: Combinational hardware trojan detection through adaptive test pattern construction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 544–557, Mar. 2021.
[151] C. Nigh and A. Orailoglu, "Taming combinational trojan detection challenges with self-referencing adaptive test patterns," in *Proc. IEEE 38th VLSI Test Symp. (VTS)*, Apr. 2020, pp. 1–6.
[152] C. Nigh and A. Orailoglu, "Test pattern superposition to detect hardware trojans," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 25–30.
[153] J. Cruz, F. Farahmandi, A. Ahmed, and P. Mishra, "Hardware trojan detection using ATPG and model checking," in *Proc. 31st Int. Conf. VLSI Design 17th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2018, pp. 91–96.
[154] L. Shen, D. Mu, G. Cao, M. Qin, J. Blackstone, and R. Kastner, "Symbolic execution based test-patterns generation algorithm for hardware trojan detection," *Comput. Secur.*, vol. 78, pp. 267–280, Sep. 2018.
[155] S. Saha, R. S. Chakraborty, S. S. Nuthakki, Anshul, and D. Mukhopadhyay, "Improved test pattern generation for hardware trojan detection using genetic algorithm and Boolean satisfiability," in *Proc. 17th Int. Workshop Cryptograph. Hardw. Embedded Syst.*, Saint-Malo, France. Cham, Switzerland: Springer, Sep. 2015, pp. 577–596.
[156] M. A. Nourian, M. Fazeli, and D. Hely, "Hardware trojan detection using an advised genetic algorithm based logic testing," *J. Electron. Test.*, vol. 34, no. 4, pp. 461–470, Aug. 2018.
[157] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *Proc. 26th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2021, pp. 408–413.
[158] H. Chen, X. Zhang, K. Huang, and F. Koushanfar, "AdaTest: Reinforcement learning and adaptive sampling for on-chip hardware trojan detection," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 2, pp. 1–23, Mar. 2023.
[159] V. Gohil, S. Patnaik, H. Guo, D. Kalathil, and J. Rajendran, "DETERRENT: Detecting trojans using reinforcement learning," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 697–702.
[160] X. Guo, R. G. Dutta, Y. Jin, F. Farahmandi, and P. Mishra, "Pre-silicon security verification and validation: A formal perspective," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.
[161] Y. Jin and Y. Makris, "A proof-carrying based framework for trusted microprocessor IP," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2013, pp. 824–829.
[162] M.-M. Bidmeshki and Y. Makris, "VeriCoq: A verilog-to-coq converter for proof-carrying hardware automation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 29–32.
[163] M.-M. Bidmeshki and Y. Makris, "Toward automatic proof generation for information flow policies in third-party hardware IP," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 163–168.
[164] N. Veeranna and B. C. Schafer, "Hardware trojan detection in behavioral intellectual properties (IP's) using property checking techniques," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 4, pp. 576–585, Oct. 2017.
[165] Veripool. *Verilator*. Accessed: Aug. 21, 2023. [Online]. Available: https://www.veripool.org/verilator/
[166] F. Khalid, I. H. Abbassi, S. Rehman, A. M. Kamboh, O. Hasan, and M. Shafique, "ForASec: Formal analysis of hardware trojan-based security vulnerabilities in sequential circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 4, pp. 1167–1180, Apr. 2022.
[167] M. Rathmair, F. Schupfer, and C. Krieg, "Applied formal methods for hardware trojan detection," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 169–172.
[168] F. Farahmandi, Y. Huang, and P. Mishra, "Trojan localization using symbolic algebra," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 591–597.
[169] A. Ito, R. Ueno, and N. Homma, "A formal approach to identifying hardware trojans in cryptographic hardware," in *Proc. IEEE 51st Int. Symp. Multiple-Valued Log. (ISMVL)*, May 2021, pp. 154–159.
[170] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *IEEE Design Test Comput.*, vol. 18, no. 4, pp. 36–45, Apr. 2001.
[171] H. Salmani, M. Tehranipoor, and J. Plusquellic, "New design strategy for improving hardware trojan detection and reducing trojan activation time," in *Proc. IEEE Int. Workshop Hardw.-Oriented Secur. Trust*, Jul. 2009, pp. 66–73.
[172] B. Zhou, W. Zhang, S. Thambipillai, J. Teo Kian Jin, V. Chaturvedi, and T. Luo, "Cost-efficient acceleration of hardware trojan detection through fan-out cone analysis and weighted random pattern technique," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 5, pp. 792–805, May 2016.
[173] Y. Jin, N. Kupp, and Y. Makris, "DFTT: Design for trojan test," in *Proc. 17th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2010, pp. 1168–1171.
[174] M. Banga and M. S. Hsiao, "ODETTE: A non-scan design-for-test methodology for trojan detection in ICs," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust*, Jun. 2011, pp. 18–23.
[175] A. Shabani and B. Alizadeh, "PMTP: A MAX–SAT-based approach to detect hardware trojan using propagation of maximum transition probability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 25–33, Jan. 2020.
[176] S. Wei, K. Li, F. Koushanfar, and M. Potkonjak, "Provably complete hardware trojan detection using test point insertion," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2012, pp. 569–576.
[177] S. Wei and M. Potkonjak, "Malicious circuitry detection using fast timing characterization via test points," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 113–118.
[178] H. Salmani and M. Tehranipoor, "Layout-aware switching activity localization to enhance hardware trojan detection," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 76–87, Feb. 2012.
[179] M. Rithesh, G. Harish, B. B. V. Ram, and S. Yellampalli, "Detection and analysis of hardware trojan using scan chain method," in *Proc. 19th Int. Symp. VLSI Design Test*, Jun. 2015, pp. 1–6.
[180] P.-S. Ba, S. Dupuis, M. Palanichamy, M.-L. Flottes, G. Di Natale, and B. Rouzeyre, "Hardware trust through layout filling: A hardware trojan prevention technique," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 254–259.
[181] 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.
[182] H. Hossein-Talaee and A. Jahanian, "Layout vulnerability reduction against trojan insertion using security-aware white space distribution," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2017, pp. 551–555.
[183] F. Wang, Q. Wang, B. Fu, S. Jiang, X. Zhang, L. Alrahis, O. Sinanoglu, J. Knechtel, T.-Y. Ho, and E. F. Y. Young, "Security closure of IC layouts against hardware trojans," in *Proc. Int. Symp. Phys. Design*, Mar. 2023, pp. 229–237.
[184] X. Wei, J. Zhang, and G. Luo, "GDSII-guard: ECO anti-trojan optimization with exploratory timing-security trade-offs," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2023, pp. 1–6.
[185] B. Khaleghi, A. Ahari, H. Asadi, and S. Bayat-Sarmadi, "FPGA-based protection scheme against hardware trojan horse insertion using dummy logic," *IEEE Embedded Syst. Lett.*, vol. 7, no. 2, pp. 46–50, Jun. 2015.
[186] A. Duncan, G. Skipper, A. Stern, A. Nahiyan, F. Rahman, A. Lukefahr, M. Tehranipoor, and M. Swany, "FLATS: Filling logic and testing spatially for FPGA authentication and tamper detection," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2019, pp. 81–90.
[187] T. M. Supon, M. Seyedbarhagh, R. Rashidzadeh, and R. Muscedere, "Hardware trojan prevention through limiting access to the active region," in *Proc. 14th Int. Conf. Design Technol. Integr. Syst. Nanosc. Era (DTIS)*, Apr. 2019, pp. 1–6.

[188] A. Alaql, M. M. Rahman, and S. Bhunia, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 8, pp. 1529–1542, Aug. 2021.

[189] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "MuxLink: Circumventing learning-resilient MUX-locking using graph neural network-based link prediction," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 694–699.

[190] T. F. Wu, K. Ganesan, Y. A. Hu, H.-S. P. Wong, S. Wong, and S. Mitra, "TPAD: Hardware trojan prevention and detection for trusted integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 4, pp. 521–534, Apr. 2016.

[191] D. K. Pradhan, *Fault-tolerant Computer System Design*, vol. 132. Englewood Cliffs, NJ, USA: Prentice-Hall, 1996.

[192] C. Bao, Y. Xie, and A. Srivastava, "A security-aware design scheme for better hardware trojan detection sensitivity," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 52–55.

[193] S. Bhunia and M. Tehranipoor, *The Hardware Trojan War*. Cham, Switzerland: Springer, 2018.

[194] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *Proc. IEEE 20th Int. On-Line Test. Symp. (IOLTS)*, Jul. 2014, pp. 49–54.

[195] Y.-L. Zhang, Y.-J. Yan, J.-W. Li, and M. Liu, "A novel hardware trojan protection technology based on logic encryption," in *Proc. IEEE 3rd Int. Conf. Integr. Circuits Microsyst. (ICICM)*, Nov. 2018, pp. 396–400.

[196] A. Nejat, D. Hely, and V. Beroulle, "ESCALATION: Leveraging logic masking to facilitate path-delay-based hardware trojan detection methods," *J. Hardw. Syst. Secur.*, vol. 2, no. 1, pp. 83–96, Mar. 2018.

[197] T. Linscott, P. Ehrett, V. Bertacco, and T. Austin, "SWAN: Mitigating hardware trojans with design ambiguity," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–7.

[198] 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.

[199] R. S. Chakraborty and S. Bhunia, "Security against hardware trojan through a novel application of design obfuscation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2009, pp. 113–116.

[200] 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.

[201] A. Sengupta and S. Bhadauria, "Untrusted third party digital IP cores: Power-delay trade-off driven exploration of hardware trojan secured datapath during high level synthesis," in *Proc. 25th Great Lakes Symp.*, May 2015, pp. 167–172.

[202] C. Bobda, J. Mead, T. J. Whitaker, C. Kamhoua, and K. Kwiat, "Hardware sandboxing: A novel defense paradigm against hardware trojans in systems on chip," in *Proc. 13th Int. Symp. Appl. Reconfigurable Comput.*, Delft, The Netherlands. Cham, Switzerland: Springer, Apr. 2017, pp. 47–59.

[203] C. Bobda, T. Whitaker, J. M. Mbongue, and S. K. Saha, "Synthesis of hardware sandboxes for trojan mitigation in systems on chip," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–6.

[204] F. Khormizi, A. Shabani, and B. Alizadeh, "Hardware patching methodology for neutralizing timing hardware trojans using vulnerability analysis and time borrowing scheme," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 6, pp. 2937–2941, Jun. 2022.

[205] M. M. Mashahedur Rahman, S. Tarek, K. Z. Azar, and F. Farahmandi, "EnSAFe: Enabling sustainable SoC security auditing using eFPGA-based accelerators," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2023, pp. 1–6.

[206] S. T. C. Konigsmark, D. Chen, and M. D. F. Wong, "Information dispersion for trojan defense through high-level synthesis," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[207] A. Sengupta, S. Bhadauria, and S. P. Mohanty, "TL-HLS: Methodology for low cost hardware trojan security aware scheduling with optimal loop unrolling factor during high level synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 4, pp. 655–668, Apr. 2017.

[208] S. Rajmohan, N. Ramasubramanian, and N. Naganathan, "Hybrid evolutionary design space exploration algorithm with defence against third party IP vulnerabilities," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2602–2614, Oct. 2020.

[209] A. P. Deb Nath, S. Ray, A. Basak, and S. Bhunia, "System-on-chip security architecture and CAD framework for hardware patch," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 733–738.

[210] W.-K. Liu, B. Tan, J. M. Fung, R. Karri, and K. Chakrabarty, "Hardware-supported patching of security bugs in hardware IP blocks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 1, pp. 54–67, Jan. 2023.

[211] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing computer hardware using 3D integrated circuit technology and split manufacturing for obfuscation," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 495–510.

[212] M. Li, B. Yu, Y. Lin, X. Xu, W. Li, and D. Z. Pan, "A practical split manufacturing framework for trojan prevention via simultaneous wire lifting and cell insertion," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1585–1598, Sep. 2019.

[213] S. Patnaik, M. Ashraf, O. Sinanoglu, and J. Knechtel, "A modern approach to IP protection and trojan prevention: Split manufacturing for 3D ICs and obfuscation of vertical interconnects," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 4, pp. 1815–1834, Oct. 2021.

[214] Y. Yang, Z. Chen, Y. Liu, T.-Y. Ho, Y. Jin, and P. Zhou, "How secure is split manufacturing in preventing hardware trojan?" *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, vol. 25, no. 2, pp. 1–23, 2020.

[215] Defense Microelectronics Activity. *Tapo—mission Statement*. Accessed: Aug. 30, 2023. [Online]. Available: https://www.dmea.osd.mil/TAPOMission.aspx?area=TAPO%20Mission%20Statement

[216] T. D. Perez and S. Pagliarini, "A survey on split manufacturing: Attacks, defenses, and challenges," *IEEE Access*, vol. 8, pp. 184013–184035, 2020.

[217] A. Suresh, S. N. Dhanuskodi, and D. Holcomb, "A secure design methodology to prevent targeted trojan insertion during fabrication," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jun. 2023, pp. 1–6.

[218] H. Salmani and M. Tehranipoor, "Analyzing circuit vulnerability to hardware trojan insertion at the behavioral level," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2013, pp. 190–195.

[219] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP Security and Trust*. Springer, 2017.

[220] H. Salmani and M. M. Tehranipoor, "Vulnerability analysis of a circuit layout to hardware trojan insertion," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1214–1225, Jun. 2016.

[221] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "ICAS: An extensible framework for estimating the susceptibility of IC layouts to additive trojans," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1742–1759.

[222] J. Cruz, P. Gaikwad, and S. Bhunia, "Analysis of hardware trojan resilience enabled through logic locking," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2022, pp. 1–6.

[223] A. Marcelli, E. Sanchez, G. Squillero, M. U. Jamal, A. Imtiaz, S. Machetti, F. Mangani, P. Monti, D. Pola, A. Salvato, and M. Simili, "Defeating hardware trojan in microprocessor cores through software obfuscation," in *Proc. IEEE 19th Latin-Amer. Test Symp. (LATS)*, Mar. 2018, pp. 1–6.

[224] M. Hasan, J. Cruz, P. Chakraborty, S. Bhunia, and T. Hoque, "Trojan resilient computing in COTS processors under zero trust," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 10, pp. 1412–1424, Oct. 2022.

[225] W. Hu, Y. Ma, X. Wang, and X. Wang, "Leveraging unspecified functionality in obfuscated hardware for trojan and fault attacks," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2019, pp. 1–6.

JONATHAN CRUZ received the B.S. (Hons.), M.S., and Ph.D. degrees in computer engineering from the University of Florida, Gainesville, FL, USA, in 2017, 2020, and 2022, respectively. His research interest includes hardware security. He has seven U.S. patents, granted or pending, and has been recognized with awards, such as IEEE HOST Best Hardware Demo, in 2019, and the IEEE HSTTC Top Picks in Hardware Security, in 2021.

JASON HAMLET (Senior Member, IEEE) received B.S. degree in electrical engineering and mathematics from New Mexico Institute of Mining and Technology, Socorro, NM, USA, in 2007, and the M.S. degree in electrical engineering from The University of New Mexico, Albuquerque, NM, USA, in 2009. His research interests include hardware and systems security and reliability. He has been granted over 25 U.S. patents and recognized as an IEEE Region 6 Outstanding Engineer, in 2019.