Modeling and Test Generation for Combinational Hardware Trojans

Ziqi Zhou, Ujjwal Guin, and Vishwani D. Agrawal
Department of Electrical and Computer Engineering
Auburn University, AL 36849, USA
{zhouziq, ujjwal.guin, agrawvd}@auburn.edu

Abstract—Due to globalization of semiconductor manufacturing, appearance of malicious circuitry known as hardware Trojan is now a recognized security threat. A Trojan may be added to the verified netlist without the knowledge of the designer or user causing unexpected malfunction or data theft when the device is in use. In this research we devise tests that would detect a Trojan in a manufactured chip. We recognize that a Trojan must escape manufacturing tests provided with the netlist by the designer. Based on the two parts of a Trojan, namely, a trigger derived as a Boolean function of any set of signals and a payload (typically, an XOR gate) inserted on a signal line, we develop a test generation model. A single-line trigger combined with a single payload line gives a set of $2K \times (K - 1)$ Trojans in this model for a circuit with $K$ signal lines. Tests for these are shown to be vectors that detect “conditional stuck-at” faults, for which we give a test generation algorithm using standard ATPG tools. The model allows us to define and measure a Trojan coverage metric for tests. Results show scalability of these tests, besides being more effective in detecting real Trojans than N-detect stuck-at test vectors or random vectors.

Index Terms—Hardware Trojans, modeling, logic testing, N-detect ATPG, verification

I. INTRODUCTION

Ensuring the security of integrated circuits (ICs) becomes a major challenge due to the globalization of the semiconductor industry. Majority of system-on-chip (SoC) design companies outsource their production across the world to fabrication units (fabs or foundries) due to a massive cost (several billion dollars [40]) for building and maintaining such foundries. This creates the threat of hardware Trojans (HT), which is a leading security concern for government and industry [3], [9], [19], [28], [30]–[33], [38]. A hardware Trojan is a malicious alteration to the original design to modify its functionality such that an adversary can gain control of the system. An adversary may insert a hardware Trojan into a design to interrupt its normal operation in the field. The Trojan would act like a “silicon time bomb” [19]. It can also create a backdoor in a secure system to give access to critical system functionality or leak secret information to an adversary.

Researchers have proposed numerous techniques to detect and prevent HTs. These techniques are broadly classified into two groups, namely, solutions targeted for the detection of HTs, and solutions designed for preventing an adversary to insert a HT in a design. The detection methods for HTs can further be classified into logic testing [8], [13], [15], [20], [35], and side-channel analysis [4], [6], [7], [21], [22], [25]. Prevention methods can be grouped into design-for-trust measures [12], [23], [26], [29], [39] and split manufacturing [27], [34], [36].

The overall aim is to detect HTs in chips manufactured in an untrusted environment and, thus, prevent Trojan infected devices from getting into the electronics supply chain. Logic testing can be used to detect these Trojans, where we apply stimuli to primary inputs (PIs) and observe responses at primary outputs (POs) [8], [9], [13], [15], [20], [30]. Detection of a HT occurs when there is a mismatch between the observed and expected responses. Such detection of a HT through logic testing does not have any impact from the process and environmental variations. On the other hand, the side-channel analysis uses physical characteristics such as power [37], temperature [24], delay [18], and radiation [16] to detect the HT. Side-channel detection methods primarily rely on the availability of Trojan-free golden circuits, which may not be available in reality. Moreover, process and environmental variations may mask the side channel leakage, if the Trojan circuitry is small. Despite significant research performed on HT, we still lack methods for modeling and test generation to detect them.

A. Contribution

We propose a generalized model of a combinational hardware Trojan. We believe this is the first time such a model for a Trojan is being presented. We then propose a generalized method based on conditional stuck-at faults to detect the modeled Trojans. The contributions of this paper are:

- **Design of a combinational hardware Trojan:** We have proposed a generalized model of hardware Trojan based on the circuit netlist. We call this Type-$n$ Trojan, where $n$ is the number of the trigger inputs. The payload of this Trojan can be delivered to a location where a stuck-at fault (SAF) is detectable by a Trojan activation pattern (TAP). Because TAPs may detect several stuck-at faults, the location of the Trojans is not unique. Any such fault site inserted with payload will result in Trojan behavior for the TAP. We believe this is the first approach to model a generalized Type-$n$ Trojan.

- **Detection based on conditional SAFs:** We have proposed a hardware Trojan detection technique based on conditional detection of SAFs. With reasonable test length, we can detect all Type-1 Trojans. These conditional SAF patterns (CSP) also detect higher order Trojans with reasonable confidence. It is reasonable to assume that an adversary will not have access to the CSP since they would not be included in manufacturing data.

The rest of this paper is organized as follows: Section II describes the generalized model for a combinational hardware Trojan, termed as Type-$n$ Trojan. Section III details our approach for detecting Type-$n$ Trojans. Section IV describes simulation results to demonstrate the effectiveness of our pro-
Trojan must have the following properties:

Property 1: A Trojan modifies the logical function of a chip, although the modification may be subtle. For certain inputs, termed as activation vectors or activation patterns, the output of the chip then deviates from its correct value. This incorrect result may help an adversary to fulfill his/her malicious purpose.

Property 2: A Trojan must not be activated by production (scan-based structural or functional) tests. This leads the Trojan circuitry to remain undetected during production testing of the chip.

Property 3: Although the effect of a Trojan may appear similar to a design error, the Trojan distinctly differs from a design error. In case of a remaining error in a completed design, production tests are generated for the chip with error and these tests aim at preserving the error in the manufactured chip. Since the Trojan is inserted in the chip design after the production tests were generated, the Trojan circuitry is, by design, made transparent to tests. Thus, the design of a Trojan must consider its function (activation inputs and modified outputs), the chip function (typically, a netlist), and the production tests.

B. Hardware Trojan Model

The Trojan circuitry may be designed for malicious purposes, such as, expose some secret key to an adversary, transmit unencrypted data to an unsecured channel, disable a circuitry, or incorrectly execute an intended function. A hardware Trojan modifies the input-output characteristics of the chip and thus provides an adversary to gain undue advantage. In this paper, we assume that the chip is sequential, is implemented with flip-flops and combinational logic, and is tested through the scan technique [11]. Typically, manufacturing tests are generated for single stuck-at faults of the combinational logic. These tests are digital vectors applied to primary inputs (PI) of the combinational logic and the results at primary outputs (PO) are verified against expected responses. Functional tests and delay tests are also performed at the manufacturing site. Without loss of generality, we focus our discussion to stuck-at fault (SAF) manufacturing tests for designing a Trojan and its detection.

Two single stuck-at (SSA) faults, namely, stuck-at-0 (sa0) and stuck-at-1 (sa1), are modeled on every signal or line, where a signal can be a primary input (PI), a gate output, or a fanout branch. Thus, the number $K$ of fault sites is given by:

$$K = \#PI + \#Gates + \#Fanout\ branches$$

A test for a fault on a signal assumes all other signals to be fault-free. The test activates the fault by setting the signal to an appropriate value, for example, 0 for a sa1 fault, and propagates the state of the signal to a primary output (PO). In addition, there are specific test sequences to verify the function of the scan shift register [11].

To facilitate the testing of a hardware Trojan, we propose a model shown in Figure 1 with following attributes:

1) Trigger: The objective of a Trojan designer is to evade manufacturing tests, otherwise every chip will fail at the testing site. The trigger circuit must remain quiet (e.g., output of the trigger $x$ remains “0”) during the tests. The selection of the trigger inputs ($T_i$) can be from the primary inputs or internal nets of the circuit.

2) Payload: A net ($s$) is selected in the circuit to deliver the payload of the Trojan. The original signal $s$, shown with broken line in Figure 1, is rerouted through a two-input XOR gate whose other input is either the trigger $x$ as shown in Figure 1 or $\overline{x}$. We define this net as Trojan location and assume that it is distinctly different from the set $\{T_i\}$ used to generate the trigger. Two conditions must be satisfied by a vector at PI to activate the Trojan. First, the vector should activate a path from $s$ to a PO, hence it should be a test for either a sa0 or sa1 fault on $s$. Second, this vector should place a logic 1 on $x$ (or logic 0 if $\overline{x}$ is connected to the XOR). As a result the PO will experience a signal inversion, changing the true function of the circuit.

Definition 1. A Type-$n$ Trojan is defined as a combinational hardware Trojan of order $n$ and has $n$ trigger inputs.

Definition 2. The location of a Type-$n$ Trojan is defined as a site (signal or line) in the circuit where the payload is delivered.
A Type-1 Trojan has only one trigger input that can come from any part of the netlist or a primary input. Similarly, a Type-2 Trojan has two trigger inputs. We note that for lower order Trojans, in general, the trigger inputs may come from low switching nets to keep the Trojans mostly quiet.

Figure 2 shows an example of a Type-4 Trojan inserted in a 6-input, 2-output circuit, where we select trigger inputs directly from PI’s. We specify this Trojan as \((n_7|x_1, x_2, x_3, x_5)\), where \(n_7\) is the payload site and \(x_1, x_2, x_3\) and \(x_5\) are the trigger input signals. Note that, in general, triggers can be tapped from any line in the circuit. Nine test vectors \(\{P1, P2, \ldots, P9\}\) are generated using the ATPG tool TetraMax [1] as manufacturing tests to provide 100% stuck-at fault (SAF) coverage. As long as the Trojan is not activated by these test vectors, the circuit will pass the production test. For this example, we have Trojan activation pattern, \(TAP = (101X00X)\) \(\notin \{P1, P2, \ldots, P9\}\), where “X” denotes don’t care state. As the order of this Trojan \((n = 4)\) is less than the number of PI (6), the Trojan may be activated by multiple input patterns. It is thus necessary to verify that the trigger output \((x)\) remains 0 for all test vectors \((P1, P2, \ldots, P9)\).

One can deliver the payload at any site, where a SAF is detected by the TAP. Because the two X’s can be enumerated in four ways, the TAP 101X0X corresponds to four vectors. Simulating [1] two SAFs at the payload site \(n_7\) we find that 101001 detects sa0 and the other three patterns, 101000, 101100 and 101101 detect sa0 at \(n_7\). Thus, the Trojan in Figure 2 will produce four errors at \(y_2\), \(0 \rightarrow 1\) for 101001 input and \(0 \rightarrow 0\) for 101000, 101100 and 101101.

In this example, the Trojan delivers the payload at signal \(n_7\), where a SAF fault is detected by one or more TAPs. By simulating any TAP, alternative locations for delivering the payload can be found. Figure 2 shows SAFs detectable at \(x_1, x_2, x_4, n_1, n_2, n_3,\) or \(n_5\) the TAP 101000. Thus, any of these lines can be used as an alternative payload site.

### C. Finding All Type-n Trojans

An upper bound on the number of Type-n Trojans \((T_n)\) is given by:

\[ T_n \leq \binom{K}{n} \times 2^n \times (K - n) \]  

(2)

where \(K\) is the number of lines in the Trojan-free circuit. From Equation 1, \(K = 18\) for the circuit of Figure 2. Numbers of all possible Trojans of types 1 through 4, computed from Equation 2, are shown in Table I. Note that the number of Trojans goes up by one order for each higher type. However, all Trojan structures (payload and trigger) do not modify the truth table of the circuit; when trigger is active, a path from payload site to PO may or may not be sensitized. Those modifying the truth table are shown as feasible Trojans in Table I. These were determined using the exhaustive set of 2^6 = 64 patterns, a fault simulator [1] to identify sensitized paths, and a logic simulator [2] to examine the trigger states. We find that a significant number of the feasible Trojans is detectable by the set of nine SAF manufacturing test patterns \(P1\) through \(P9\) shown in Figure 2. We do not consider those as valid Trojans because chips containing them will be eliminated during production testing. Removing them from feasible Trojans gives us the number of valid Trojans.

Although the Type-n Trojan model seems general, even for a small circuit \((K = 18)\), the number of valid Trojans grows rapidly with \(n\). For generating tests for Trojan detection and for coverage analysis, we will use Type-1 Trojans assuming their number equals the upper bound of Equation 2:

\[ T_1 = 2K(K - 1) \]  

(3)

This number of target Trojans is \(O(K^2)\), or quadratic in circuit size, \(K\). The number would be \(O(K^{n+1})\) for Type-n Trojans, where \(n \leq K - 1\). Thus, our methodology parallels SAF whose tests are known to detect multiple stuck-at and many other types of faults [11].

Trigger circuitry of a Type-n Trojan model is an \(n\)-inputs AND gate. The payload is delivered through an XOR gate to any site activated by the Trojan activation pattern (TAP).
Algorithm 1: Design of a Type-\(n\) Trojan.

**Input**: Circuit Netlist \((C)\), Manufacturing test patterns \((P)\), Order of a Trojan \((n)\)

**Output**: Trojan activation pattern \((TAP)\), Trigger Inputs \((T)\)

1. Read the netlist;
2. Read manufacturing test patterns \((P)\);
3. Select a random pattern as Trojan activation pattern, \(TAP \notin P\);
4. Perform logic simulation using \(P\) to obtain all internal node values \((M_K)\);
5. Perform logic simulation with \(TAP\) to obtain all internal node values \((S_T)\);
6. Select \(n\) random locations of the netlist to form the trigger inputs;
7. Form a new matrix \(M_n, M_n \leftarrow \text{mod}(M_K)\);
8. If \(S_n \in M_n\) then:
   - Drop the selection as it will activate the Trojan;
   - Go to Step 6;
9. Else:
   - Choose \(T\) as trigger input;
10. Perform fault simulation and logic simulation with \(TAP\);
11. Select a fault site (from Step 14) for delivering the payload, i.e., Trojan location.

Algorithm 1 designs a Type-\(n\) Trojan that will not be activated the manufacturing tests. The inputs are original netlist \((C)\), manufacturing test patterns \((P)\), and the order of the Trojan \((n)\). The algorithm reports the \(TAP\) and trigger inputs \((T)\). It reads the Trojan free circuit netlist and manufacturing test patterns (Lines 1-2). A random \(TAP\) is selected (Line 3), which is not present the manufacturing test pattern set \((TAP \notin P)\). Logic simulation gives the internal node values \((M_K)\) of the circuit for all manufacturing test patterns (Line 4). \(M_K\) is a \(K \times p\) matrix where \(K\) and \(p\) denote number of circuit nodes and number of manufacturing test patterns, respectively. We simulate the circuit with \(TAP\) for all internal node values \((S_T)\). Here, \(S_T\) is a \(K \times 1\) vector. Select \(n\) locations from \(K\) nets, either randomly or by some given criterion, to form an \((n \times 1)\) vector \(S_n\) from \(S_T\) (Line 6). The \(\text{mod}(\cdot)\) function returns a new matrix \(M_n\) corresponding to the selected \(n\) nodes (Line 7). Selection of these \(n\) nets is not valid if \(S_n \in M_n\), as one of the test pattern will trigger the Trojan (Line 9), so select a new set of \(n\) nodes (go to Step 6). Finally, fault simulation with \(TAP\) gives possible sites for payload (Lines 14-15).

### III. Test Generation for Type-\(n\) Trojans

**Definition 3.** CSP-\(n\), CSP-1 or CSP, and CSP-0: For a signal \(s\) in a digital circuit, two type-\(n\) conditional stuck-at fault (SAF) patterns (CSP-\(n\)) detect \(sa0\) and \(sa1\) faults, respectively, while setting specified \([0,1]\) values on \(n\) other signals \(t_1, \ldots, t_n\). CSP-1, or simply CSP, are conditional tests with a condition on a single signal. CSP-0 are tests without any condition and are identical to the classical SAF tests.

Type-\(n\) HT’s with signal \(s\) as payload are detectable by CSP-\(n\) if we denote as \((s \sa0 | C_1, \ldots, C_n)\) or \((s \sa1 | C_1, \ldots, C_n)\), where \(C_i = t_i\) or \(T_i\). CSP-\(n\) is a generalization of the CSP-1 defined in the literature [14].

Clearly, a Type-\(n\) Trojan is detectable by any of the two CSP-\(n\)’s for which \(s\) is the payload site and \(t_i’s\) are trigger inputs. For example, the Type-4 Trojan in Figure 2 is detected by CSP-4 \((n7 \ sa0 | x_1, x_2, x_3, x_5)\) and \((n7 \ sa0 | x_1, x_2, x_3, x_5)\). Considering complexity, we restrict to test generation for CSP-1 and evaluate their coverage for higher type of Trojans.

#### A. Conditional SAF Pattern (CSP-1 or CSP) Generation

Algorithm 2 generates conditional SAF patterns (CSP) for detecting hardware Trojans. It is necessary to determine the total number of nets \((K)\) according to Equation 1 (Line 2). The algorithm initializes CSP-\(n\) as an empty set (Line 3) and then iterations with their signal values \((Nets)\) are stored in \(\text{TempNets}\). A CSP-1 is generated for a specific SAF with signal values for specific nets (Line 11). Note that, a CSP-\(n\) generation capability in the ATPG tool TetraMAX [1] is invoked by specifying an SAF target with \(n\) other signals and their values. Logic simulation is performed with this pattern to find the internal node values (Line 12). All \{net, signal value\} pairs corresponding to this pattern are dropped from \(\text{TempNets}\) (Line 13). Repeat this process until \(\text{TempNets}\) is empty (Line 10). Once all iterations are complete, the algorithm reports CSP-\(n\)’s for Type-\(n\) Trojan detection.

#### B. An Example: Circuit of Figure 2

Once again, considering the high complexity due to large number of higher type of Trojans, explained in Section II C, we generated tests for all 612 Type-1 Trojans using Algorithm 2. As a result, 48 CSP-1 detected 605 feasible Trojans confirming the data in Table I. Without confusion, we simply call them CSP. Assuming that 9 SAF vectors would have already tested, generated tests for all 612 Type-1 Trojans using Algorithm 2. Manufacturing tests are applied during production to all chips...
to eliminate defective ones. Assuming that a Trojan remains undetected (we define this as a valid Trojan), all passing chips must have the same Trojan. Hence it is sufficient to test just one chip for Trojans and the Trojan tests can be much longer than the manufacturing tests. For any type \( n \), the quality of Trojan tests is their coverage of valid Trojans, which, in turn, depends on the manufacturing tests. Thus,

\[
\text{Trojan Coverage} = \frac{\# \text{ of detected valid Trojans}}{\# \text{ of all valid Trojans}} \times 100 \%
\] (4)

We generated two other sets, each with 48 vectors, an \( N \)-detect set and a random set. Valid Trojans of types 1 through 4 (Table I) were simulated. Results are given in Table II (Rows 1, 4, 7 and 10). Coverage of CSP was always higher and dropped slower with increasing \( n \). Four Type-2 Trojans, \((x_5 | \overline{T_2}, x_4)\), \((x_5 | x_4, n_5)\), \((n_4 | \overline{T_1}, x_4)\) and \((n_3 | x_1, x_4)\), were only detected by CSP. Their payloads are closer to PI. Besides, they indicate superior capability of CSP in covering trigger combinations.

### IV. BENCHMARK CIRCUITS

To study the effectiveness of the proposed conditional SAF patterns (CSP), we used a simulation setup for ISCAS 85 benchmark circuits [10]. TetraMax [1] provided manufacturing tests for each circuit covering 100% of all detectable SAFs. Next, we generate the CSP for each circuit using Algorithm 2. As the number of valid Trojans is large, we perform the coverage analysis of CSP based on four random sample sets of 20,000 Trojans of Type-1 through Type-4, respectively. Some Trojans cannot be triggered from inputs, nor do they affect outputs. Excluding these, we get feasible Trojans. Feasibility within each sampled set was assessed using the TetraMax conditional ATPG capability [1]. Some feasible Trojans are detectable by the manufacturing tests. Excluding those, we get valid Trojans \( v_n \), any of which an adversary may insert in the netlist. It is economical to estimate the total number of valid Trojans \( V_n \) in a circuit based on the 20,000-Trojan sample. This sample size is large enough for reasonable accuracy [11]. Total number of valid Trojans is

\[
V_n = \frac{v_n}{20,000} \times T_n
\] (5)

Table II shows the results. Trojan sampling was not used for the circuit of Figure 2 (Rows 1, 4, 7 and 10). Next, Rows 2, 5, 8 and 11 show data for c432 benchmark. For \( K = 307 \), Equation 2 gives \( T_1 = 187,884 \) Type-1 Trojans (Column 4). This circuit has 712 SAFs detected by 69 manufacturing test patterns (Column 5). Algorithm 2 generated 99,991 CSP, beyond 69 SAF patterns, shown as HT tests in Column 7. From 187,884 Type-1 Trojans, we take a random sample of 20,000 Type-1 Trojans to estimate the Trojan coverage. Among these, 513 Trojans could not be triggered from inputs, leaving 19,487 feasible Trojans. In addition, 17,711 Trojans were detected by 69 SAF patterns. Hence, number of valid Trojans, \( v_n = 19,487 - 17,711 = 1,776 \). From Equation 5, number of valid Type-1 Trojans, \( V_n = 16,684 \) (Column 7). Next three columns of Row 2 give Type-1 Trojan coverage by CSP, \( N \)-detect patterns, and random patterns, respectively, each containing 99,991 patterns. Similarly, results for Trojans of Type 2 (Row 5), Type 3 (Row 8) and Type 4 (Row 11) were obtained. Notably, the CSP coverages are consistently higher.

Results for c880 benchmark in Rows 3, 6, 9 and 12 were obtained in a similar manner with one exception. The number of HT tests is 531,846 and will grow significantly larger for bigger circuits. We randomly sampled 5,000 patterns from 531,846 HT tests to estimate the coverage of Trojans of Types 1 through 4 [17]. The results are given in Columns 8-10 (rows for c880). Once again, CSP coverages are higher.

### V. CONCLUSION

The Type-\( n \) Trojan is a generalized model that facilitates test generation and coverage analysis. A Consideration of the complexity issue leads to the Type-1 Trojan model and its test by conditional stuck-at fault patterns (CSP). Thus, the number of Trojans to be modeled is \( O(K^2) \) for a circuit with \( K \) signal lines. Although the detection coverage is measured over valid Type-1 Trojans, not detectable by manufacturing tests, tests are generated for all Type-1 Trojans. This is because our “real” targets include higher types as well. We find that both Trojan sampling and vector sampling are beneficial for coverage estimates. For larger circuits, CSP generation for a random sample of Type-1 Trojans may also be used [5].

In the future, the scope of modeling and test generation should be expanded to solve diagnostic problems. Another
aspect to explore is the minimization of Trojan tests. Despite the fact that the Trojan tests need not be applied to all chips, the numbers of Type-1 Trojans and their CSP for large circuits can be enormous. A third aspect to explore is the behavior of CSP in detecting Trojans with \( n > 4 \).

**Acknowledgment:** This research was supported by an internal grant from the ECE Department at Auburn University.

**REFERENCES**