FASTrust: Feature Analysis for Third-Party IP Trust Verification

Song Yao[1], Xiaoming Chen[2], Jie Zhang[3], Qiaoyi Liu[1], Jia Wang[4], Qiang Xu[3], Yu Wang[1], Huazhong Yang[1]

[1]Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
[1]Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
[2]ECE Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[3]Department of CSE, The Chinese University of Hongkong, Shatin, N.T., Hongkong
[4]Department of ECE, Illinois Institute of Technology, Chicago, IL 60616, USA
[1]{yaos11, liuqy13}@mails.tsinghua.edu.cn, {yu-wang, yanghz}@tsinghua.edu.cn
[2]xchen3@andrew.cmu.edu, [3]{jzhang, qxu}@cse.cuhk.edu.hk, [4]jwang@ece.iit.edu

***Abstract*—Third-party intellectual property (3PIP) cores are widely used in integrated circuit designs. It is essential and important to ensure their trustworthiness. Existing hardware trust verification techniques suffer from high computational complexity, low extensibility, and inability to detect implicitly-triggered hardware trojans (HTs). To tackle the above problems, in this paper, we present a novel 3PIP trust verification framework, named FASTrust, which conducts HT feature analysis on the flip-flop level control-data flow graph (CDFG) of the circuit. FASTrust is not only able to identify existing explicitly-triggered and implicitly-triggered HTs appeared in the literature in an efficient and effective manner, but more importantly, it also has the unique advantage of being scalable to defend against future and more stealthy HTs by adding new features to the system.**

***Index Terms*—Hardware Trojan , third-party intellectual property, feature analysis, hardware security**

I. Introduction

With the continuous globalization of integrated circuit (IC) design and fabrication process, third-party intellectual property (3PIP) cores are widely adopted in IC designs to reduce development cost and time-to-market. However, as both the design and verification of 3PIPs are conducted by the IP providers, adversaries can easily introduce malicious circuits (i.e., hardware Trojans (HTs) [1]) into IP cores to cause untimely chip failure or leak confidential information covertly. Consequently, the trustworthiness of 3PIPs is a serious security concern.

To help obtain trustworthy 3PIPs, system integrators (i.e., IP consumers) can either take proactive preventive measures or passive trust verification methods. When an agreement is reached with the IP vendor, IP consumers can assign specific design rules for 3PIP such as fulfilling some security-related properties [2], [3]. With such design rules, it is possible to verify the trustworthiness of 3PIPs. However, when such an agreement is not available (and it is the common case, at least for today), it is essential to verify whether 3PIPs contains HTs. Knowledge-guided methods analyze the differences between the specification and the procured IP core, and can identify trusted parts in 3PIPs [4], [5]. Redundancy-based methods diversify the sources of 3PIPs and check the equivalence between the results from different version 3PIPs which should have the same functionality [6]–[8]. Signal-based and circuit-based methods exploit the rareness of HT activation to capture HTs: On one hand, signal-based methods directly monitor all the wires to capture HT enable signals [9], [10]. On the other hand, circuit-based methods, such as UCI [11], FANCI [12], and VeriTrust [13], check the dependencies between wires in combinational logic circuits to locate HT trigger inputs or payloads.

This work was supported by 973 project 2013CB329000, National Natural Science Foundation of China (No. 61261160501, 61373026), Tsinghua University Initiative Scientific Research Program, and The Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions.

A. Contributions

HT designs and trust verification techniques are like arms race. As HT designs would adjust their tactics with known HT detection techniques, it is nearly impossible to come up with a technique that is able to defend against all present and future HTs. Feature analysis has been proved to be a scalable solution for malware detection, in which a set of features of malware are extracted and used for identification. In this paper, we bring this concept into the hardware trust verification domain, and propose a novel HT detection framework, named FASTrust, which employs simple yet effective feature analysis to detect HTs in 3PIPs.

FASTrust is different from all existing HT detection methods in three aspects. First, FASTrust extracts a set of features to represent different HT types based on the HT taxonomy rather than distinguishing all HTs with a uniform rule. Besides, FASTrust detects HTs in the flip-flop level control data flow graphs (CDFGs) rather than directly analyzing the functionality of gate-level netlists. Furthermore, in sequential circuits, unlike existing circuit-based methods that examine combinational logic circuit blocks individually, FASTrust models the relationships between different parts. Though we cannot theoretically prove that all HTs can be found, results show that FASTrust is able to detect all HTs from TrustHub benchmarks [14] and DeTrust [15] benchmarks appeared in the literature effectively.

978-1-4673-6578-9/15/$31.00

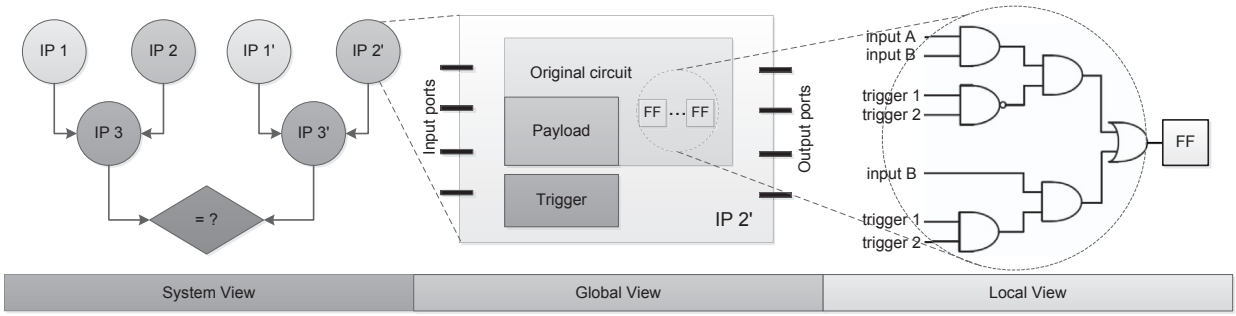

Fig. 1. The perspectives of SoC protection methods with untrustworthy 3PIPs

To be specific, this paper makes the following contributions:

- We propose FASTtrust, a fast and extensible 3PIP trust verification approach based on feature analysis.
- We extract a set of representative features for different HT types in the flip-flop level CDFG based on detailed HT classification. To our best knowledge, this is the first work to detect HTs in the flip-flop level CDFG.
- In this work, we make the first attempt to detect implicitly-triggered HTs, i.e., HTs whose trigger logics spread over multiple combinational logic circuit blocks and sequential levels. Such kind of HTs are shown to be able to evade state-of-the-art HT identification techniques and there are no known solutions to detect them [15].

The rest of the paper is organized as follows. Section II briefly introduces the related work and our motivation. Section III presents the HT classifications. The overall flow of FASTrust and the features for different HT types are proposed in Section IV. Experimental results are presented in Section V. We discuss the limitations and future work of FASTrust in Section VI. Finally we conclude this paper in Section VII.

II. Related Work and Motivation

A. Related work

If an agreement can be reached between the IP vendor and IP consumers, in order to obtain trustworthy 3PIPs, specific design rules can be assigned for 3PIP design when outsourced. In this case, when procuring the 3PIPs from the IP vendor, it will be much easier to verify their trustworthiness. Jin *et. al.* proposed a set of circuit security properties and a proof-carrying based framework for assessing the trustworthiness of third-party microprocessor IPs [2]. Love *et. al.* further developed the proof-carrying framework in [3].

With untrustworthy 3PIPs, the trust verification techniques are essential to find potential malicious parts in the 3PIPs and assist manual check. The perspectives of protection and trust verification techniques can be roughly categorized into three levels: system view, global view, and local view. Examples for all kinds of perspectives are shown in Fig. 1.

- **System view**: Regarding a 3PIP as a black box and protecting the whole on-chip system.
- **Global view**: Considering dependencies between all different parts in a 3PIP.
- **Local view**: Considering only dependencies between wires or circuits within a specific part of a 3PIP.

Existing trust protection and verification techniques can be classified into four categories: redundancy-based methods, knowledge-based methods, signal-based methods, and circuit-based methods. In general, redundancy-based methods take the system view, knowledge-based methods employ the global view, signal-based methods and circuit-based methods are local-view methods. The definitions of different kinds of trust protection and verification methods are listed as follows.

- **Redundancy-based methods:** Using several IPs from different vendors with same functionality to implement runtime protection.
- **Knowledge-based methods:** Making use of knowledge on existing trusted circuits and HTs to identify trustworthy circuits and malicious circuits.
- **Signal-based methods:** Considering no circuit structures, directly monitoring all signals, and determining whether a signal is suspicious based on the property of the signal itself.
- **Circuit-based methods:** Considering circuit structures and detecting suspicious wires and circuits by checking dependencies between wires in combinational logic circuit blocks.

Redundancy-based methods all employ the system view and support runtime protection. The fundamental assumption of these methods is that the 3PIPs procured from different IP vendors have different implementations even when they have the same functionality. In other words, even there are HTs in the 3PIPs obtained from different IP vendors, their trigger conditions and payloads will not be the same, and thus in most cases, the outputs of different version 3PIPs should be the same. Al-Anwar *et al.* proposed a method to determine suspected 3PIPs with a majority voting circuit [8]. Rajendran *et.al.* formulated two security constraints for using 3PIPs with IP duplication [6]. First, the 3PIPs which perform original and redundant operations should be procured from different IP vendors. Second, at least one parent operation of a 3PIP should be performed with an IP from a different IP vendor. Cui *et.al.* further developed the security constraints to save cost and enable runtime recovery [7].

Signal-based methods consider signals individually and take the local view. Banga *et al.* proposed a four-phase method to identify HT trigger signals [9]. They first employed the functional vector simulation and an N-detect full scan ATPG to find hard-to-invert signals. After that, they used the SAT

solver [16] to analyze whether a state that never appeared was reachable. Finally, suspicious gates could be located according to the suspicious signals. Zhang *et al.* first employed coverage analysis and then took several techniques to reduce the suspicious wire set [10].

Existing circuit-based methods all take the local view since they only exploit the dependencies within a combinational logic circuit block. Hicks *et al.* proposed a method named Unused Circuit Identification (UCI) by finding equivalent signal tuples. [11]. For example, in one circuit, if two signals S_1 and S_2 equaled each other in all test cases, they concluded that the circuit between S_1 and S_2 was unused and therefore was suspicious. Besides, Zhang *et al.* proposed the VeriTrust method to detect HTs triggered by complex trigger patterns [13]. They recorded the activation history of minterms and maxterms in the K-map of a circuit during simulation time and detected HTs by finding un-activated malicious minterms and maxterms. Furthermore, Waksman *et al.* proposed FANCI, an HT detection method based on static boolean functional analysis [12]. They analyzed the truth table of combinational logic circuit and calculated the "control value" of each input on the output to evaluate the influence of each input on the output. They concluded that the inputs which weakly affected outputs were suspicious.

Knowledge-based methods are different from the methods mentioned above [4], [5]. These methods build a knowledge base for RTL code and specification and identifies the circuits that exactly match the specification. Results in [4], [5] show that, in most cases, the identification coverage is larger than 60%, and thus the workload for manual check can be reduced.

B. Motivation

Redundancy-based methods have several advantages. For example, they are able to withstand the threat from unverified 3PIPs and no golden model is needed. However, they also increase the implementation cost greatly. Besides, the redundant area and power consumption result in the inapplicability for some usage-critical products.

Compared with signal-based methods, circuit-based methods can achieve much better HT coverage. However, circuit-based methods suffer from near exponential complexity when analyzing the functionality of a combinational logic circuit block: for an n-input combinational logic circuit, the complexity is about $O(2^n)$. Besides, since taking the local view, circuit-based methods examine combinational logic circuit blocks individually. In sequential circuits, combinational logic circuit blocks are separated by flip-flops. Consequently, if an HT consists of multiple combinational logic circuit blocks over many sequential levels, it may evade circuit-based methods.

Since the concept of hardware trojan is borrowed for software trojan, naturally, it is worth investigating the methods in malware detection. In the malware detection area, signature matching, i.e. feature analysis, is a very effective method to detect virus. The overall flow of malware detection using signature matching is shown in Fig. 2. For each kind of virus, a unique signature can be extracted. For untrustworthy data, a matching process is executed to find whether there is a part of data exactly matches the signature. If any part of the data matches the signature, it can be concluded that this piece of data is contaminated by the corresponding virus.

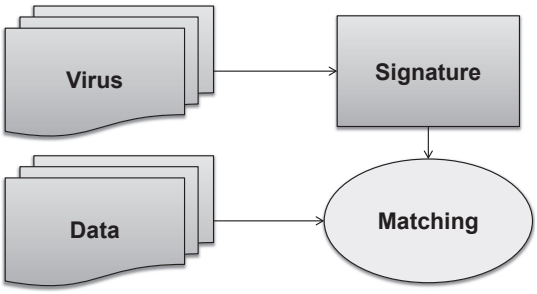

Fig. 2. The overall flow of malware detection using signature matching.

Feature analysis, which is proved to be effective for malware detection, also has potential to be a fast and extensible 3PIP trust verification solution. For known HT types, it is possible to extract a set of high-level features to distinguish them from benign circuits.

In this paper, we propose a novel 3PIP Trust verification method, named FASTrust, by applying feature analysis to hardware trojan detection. Unlike the knowledge-based methods proposed in [4], [5] which built RTL base and spec base, and tried to identify all benign functional modules, FASTrust aims to directly identify malicious circuits. Based on the HT taxonomy, a set of features in the flip-flop level CDFG that can represent different HT types are extracted. Since the flip-flop level CDFGs only contain the information of data transfer between inputs, outputs, and flip-flops, they are on relatively much smaller scale compared with the original 3PIPs. For example, the benchmark AES-T400 from [14] has around 163 thousand units but only about 7 thousand nodes in its corresponding flip-flop level CDFG. Consequently, FASTrust is dealing with a problem on much smaller scale and has potential to achieve fast detection. Besides, the flip-flop level CDFGs contain all connection relations between flip-flops, which exactly model the relations between different combinational logic circuit blocks in sequential circuits, and thus FASTrust is able to take the global view for HT detection in sequential circuits. In this way, all the parts of a 3PIP can be considered altogether, and hence, it is possible to detect implicitly-triggered HTs.

C. Threat Model

The threat model used in this paper follows the same threat model in [13]: The 3PIPs are procured from outside design houses in the form of either RTL code or gate-level netlist without knowledge of their trustworthiness. We assume the HTs (if any) implement certain malicious logic function (e.g., not hidden in the clock/power network), but the the HT trigger mechanisms and malicious payloads are not restricted.

III. HT Classification

FASTrust is based on feature analysis, in which each feature represents a certain HT type. Consequently, a detailed HT clas-

sification method is required. In this section, two classification methods based on HT trigger mechanisms and trigger spread mechanisms are introduced. The HT types which can only be inserted during the fabrication stage are not considered in this paper.

A. Classification based on trigger mechanism

Based on the HT trigger conditions, HTs can be roughly classified into three types: always-on HTs, time-triggered HTs, and data-triggered HTs.

- **Always-on HT**: An always-on HT executes the hidden malicious operation all the time.
- **Time-triggered HT**: A time-triggered HT can also be called as a data-independent HT. An internal trigger circuit should be designed to activate the HT after a certain period.
- **Data-triggered HT**: The trigger condition of a data-triggered HT is related to the inputs or internal signals of the 3PIP. Depending on whether the HT trigger circuit contains a state machine, the data-triggered HTs can be further divided into the single-triggered HTs and the sequential-triggered HTs.
  - **Single-triggered HT**: A single-triggered HT can be triggered when the unique trigger condition is met.
  - **Sequential-triggered HT**: A sequential-triggered HT needs a state machine to record the history and can be activated when a sequence of trigger conditions occur.

For example, a time-triggered HT may contain a 40-bit counter and delay the HT activation for tens of minutes. A single-triggered HT in a CPU may be triggered when the CPU fetches a specific 64-bit instruction. A sequential-triggered HT can be triggered when a CPU fetches a specific instruction for three times.

Sometimes, it is hard to strictly differentiate time-triggered HTs and sequential-triggered HTs. Generally, the trigger circuit of time-triggered HTs should be independent from inputs except the clock and reset signals. In this paper, we do not strictly distinguish between time-triggered HTs and sequential-triggered HTs but adopt the definitions within the benchmark introduction documents.

B. Classification based on trigger spread mechanism

Based on how the trigger logic spreads, the HTs can be differentiated into two categories: explicitly-triggered HTs and implicitly-triggered HTs. We adopt the definitions from [15]:

- **Explicitly-triggered HT**: An explicitly-triggered HT has an input pattern in the HT-affected signal's fan-in logic cone, which uniquely represents the trigger condition,
- **Implicitly-triggered HT**: An implicitly-triggered HT has no input pattern in the HT-affected signal's fan-in logic cone, which uniquely represents the trigger condition.

The trigger logics of an implicitly-triggered HT must spread over multiple combinational logic blocks and sequential stages. For example, assuming there is a 32-bit counter in a time-triggered HT, if

$$Counter[31]_{in} = f(Counter[0]_{out}, ..., Counter[30]_{out})$$

and each trigger logic is implemented into one combinational logic circuit block, then this HT is an explicitly-triggered HT. Otherwise, if

$$Intermeadiate1_{in} = f_1(Counter[0]_{out}, ..., Counter[14]_{out}),$$
$$Intermeadiate2_{in} = f_2(Counter[15]_{out}, ..., Counter[30]_{out}),$$
$$Counter[31]_{in} = f_3(Intermediate1_{out}, Intermediate2_{out}),$$

and all trigger logics are separated into two or more sequential stages, then this HT is an implicitly-triggered HT.

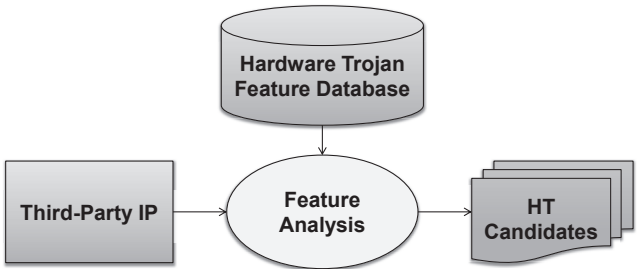

Fig. 3. The overall flow of FASTrust.

IV. FASTrust Framework

In this section, the proposed FASTrust technique is introduced in detail. First, an overview of FASTrust is presented. After that, the process of building CDFG is introduced. Finally, each feature and the corresponding feature analysis algorithm are introduced.

A. Overview

The overall flow of the proposed FASTrust method is shown in Fig. 3. Based on the HT taxonomy, an HT feature database is established in advance. After obtaining the 3PIPs, the feature analysis, the core step in FASTrust, is conducted. For each feature in the HT feature database, a feature matching algorithm is also proposed to detect nodes or node groups using the feature. Finally, all HT candidates that match HT features are reported for further manual examination.

The features of HTs can be extracted at various levels, such as the physical level, the transistor level, the gate level netlist, and the flip-flop level CDFG. A lower level description, for example, transistor level, contains more details of the 3PIP but also suffers from larger size and longer verification time. On the contrary, a higher level description such as flip-flop level CDFG contains less information but enables extraction of simple features and supports for fast detection. In this paper, we only consider the HT features at the flip-flop level CDFG. At this level, all connection relations of flip-flops are maintained but all details of the combinational logic and parasitic parameters are discarded.

It is possible to introduce a preprocessing stage before feature analysis after obtaining the 3PIP. The preprocessing

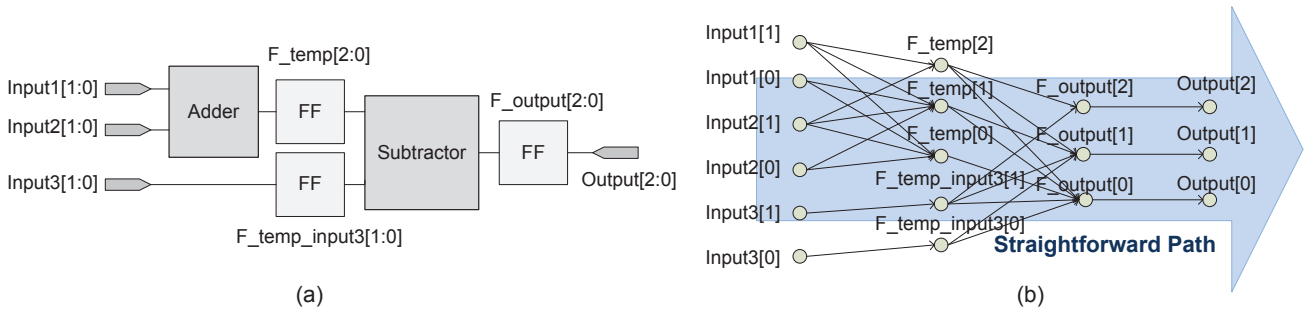

Fig. 4. Build flip-flop level from original circuits: (a) An example circuit and (b) the corresponding flip-flop level CDFG.

stage is aimed to identify all functional modules that exactly match the specification. The objective of this step is to remove those known benign circuits from consideration, which may not only relax the computational effort for the following feature analysis step, but more importantly, reduce the false positive rate for the reported potential HTs. In this paper, the preprocessing stage is not implemented, and all nodes in the flip-flop level CDFG are considered in feature analysis.

B. Building CDFG of a 3PIP

In the flip-flop level CDFG, all flip-flops, input ports, and output ports are modeled as nodes, and the data dependencies are modeled as directed edges. The flip-flop level CDFG contains only relations between flip-flops and all combinational logic circuits are removed. For example, if there is an *n*-input combinational logic circuit block in a sequential circuit:

$$F_{0-in} = f(F_{1-out}, ..., F_{n-out})$$

where F means flip-flop, then there will be n edges from node F_1, ..., F_n to node F_0 in the flip-flop level CDFG. The established CDFG is a directed graph.

In the flip-flop level CDFG, the nodes are classified into two categories: the loop nodes and the normal nodes.

- **Loop Node:** The node that contains a self-loop.
- **Normal Node:** The node that contains no self-loop and passes the data straightforward to the successors in the next stage.

The two kinds of nodes compose two parts in the flip-flop level CDFG, the loop groups and the straightforward paths.

- **Loop Group:** A group of linked loop nodes. Any node in one loop group has at least one linked node which is in the same loop group.
- **Straightforward Path:** A group of linked normal nodes. No node in one straightforward path is a loop node and a straightforward path can be divided into several sequential stages.

An example circuit and its corresponding flip-flop level CDFG are shown in Fig. 4. The example circuit contains 3 sequential stages, reads three 2-bit inputs and outputs the result of ($input_1$ + $input_2$ - $input_3$). Since there is no loop node in Fig. 4 (b), all the nodes compose a straightforward path.

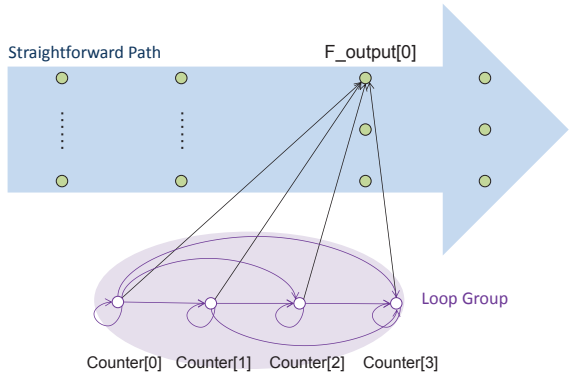

Fig. 5. Example of Feature 1: The flip-flop level CDFG of the example circuit in Fig. 4 with an additional time-triggered HT.

C. Features of time-triggered HTs

For any time-triggered HT, to delay the activation and evade functional verification (FV), an internal large counter must be designed to count the number of cycles. Usually, FV runs millions of cycles. Since $2^{20} \approx 1$ million, to evade FV, a time-triggered HTs should contains a counter whose size is larger than 20 bits. In the flip-flop level CDFG of a 3PIP, we can distinguish large counters to detect time-triggered HTs.

As an example, we insert a 4-bit counter in the example circuit shown in Fig. 4 to count the time and activate the HT. When $Counter[3:0] = 4'b1111$, the HT is activated, and the lowest bit of the output is stuck at 1. As shown in Fig. 5, in the flip-flop level CDFG of the example circuit with a 4-bit counter, there are 4 nodes forming a loop group. If all loop groups that consist of more than 3 nodes are marked as suspicious, then the 4-bit counter can be detected.

The feature for time-triggered HTs can be summarized as:

- **Feature 1 :** All nodes in the trigger circuit of a time-triggered HT form a large loop group.

Based on Feature 1, the matching algorithm can be proposed. A cut-off *threshold* is employed to judge whether a loop group found by the matching algorithm is suspicious.

1) For each node, if it contains a self-loop, mark it as a loop node.
2) For each loop node, if it is linked with other loop nodes, add them into one loop group.
3) For each loop group, if its size is larger than the *threshold*, report it as a suspicious loop group.

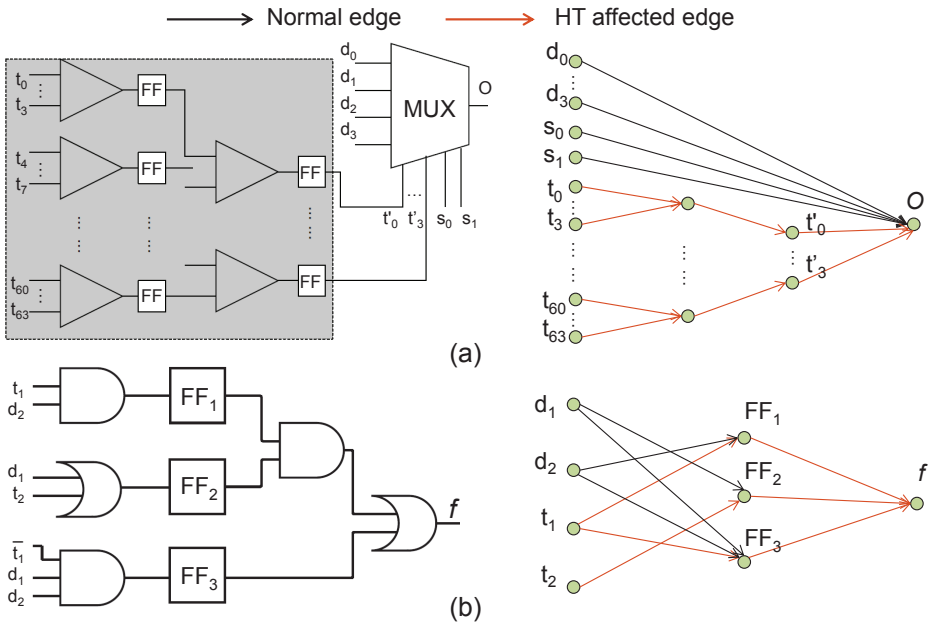

Fig. 7. Two methods to evade circuit-based methods: (a) Splitting large HT payload circuits into smaller ones and spreading them into multiple sequential stages; (b) Separating HT trigger inputs and mixing them with functional inputs [15]. The example circuits are shown on left, and the corresponding flip-flop level CDFGs are shown on right.

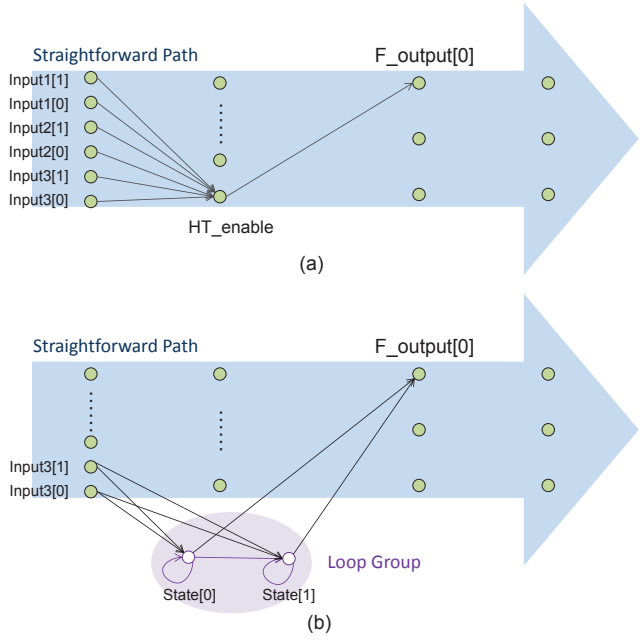

Fig. 6. Example of Feature 2 and Feature 3: The flip-flop level CDFG of the example circuit in Fig. 4 with (a) a single-triggered HT and (b) a sequential-triggered HT.

D. Features of data-triggered HTs

For data-triggered HTs, the trigger inputs should be deliberately selected to achieve an extreme low trigger probability. However, a rare trigger condition is not easy to be implemented. With a small trigger pattern, it is not predictable that one pattern occurs with extremely low probability for all cases. Consequently, the trigger patterns of data-triggered HTs tend to be large: For single-triggered HTs, the number of trigger inputs is usually great; for sequential-triggered HTs, either the depth of the state machine or the number of trigger inputs tends to be great.

The in-degree of a node is the number of directed edges arrive at the node. In the flip-flop level CDFG, the in-degree of one node represents the number of inputs in the fan-in logic cone of the corresponding flip-flop. Consequently, if a combinational logic circuit contains a great number of inputs, the corresponding node has a large in-degree. This can be summarized into the following feature:

- **Feature 2:** A single-triggered HT contains a node with extremely large in-degree.

For sequential-triggered HTs, there may be large state machines. Besides, a small state machine whose state is hard to be changed may also serve as the trigger circuit of a sequential-triggered HT. A state machine can be found as a loop group in the flip-flop level CDFG. To evaluate both the state machine size and the difficulty for state change, the total in-degree from outside nodes of a loop group is selected as a feature since the total in-degree increases along with both the increment of loop group size and the in-degree of each node:

- **Feature 3:** A sequential-triggered HT contains a loop group whose total in-degree from outside nodes is extremely large.

The matching algorithm for Feature 2 is intuitive: For each node, if its in-degree is larger than a *threshold*, then mark it as a suspicious node. For Feature 3, the matching algorithm is similar to the matching algorithm of Feature 1: Finding all loop nodes and then obtaining all loop nodes, finally using a *threshold* of total in-degree of this loop group to judge whether the loop group is suspicious.

Fig. 6 (a) shows the flip-flop level CDFG of the example

circuit with a single-triggered HT. The HT can be triggered when all the inputs are assigned to predetermined values. If the cut-off threshold of in-degree is smaller than 6, node *HT_enable* will be reported as suspicious. The flip-flop level CDFG of the example circuit with a sequential-triggered HT is shown in Fig. 6 (b). If the cut-off threshold of total in-degree from outside nodes is 4, the loop group consists of *state[1]* and *state[0]* will be marked as suspicious.

E. Features of implicitly-triggered HTs

In sequential circuits, since combinational logic circuit blocks are separated by flip-flops, an HT consists of multiple combinational logic blocks may evade the detection of circuit-based methods.

In [15], an implicitly-triggered HT design methodology named DeTrust was proposed to defeat existing circuit-based methods such as FANCI and VeriTrust. DeTrust spreads the trigger logic of HTs into multiple sequential stages and combination logic blocks. As shown in Fig. 7 (a), since FANCI identifies signals with weakly-affecting inputs, DeTrust splits large HT payloads into small ones whose inputs all strongly affect the output. In this way, no input will be marked as suspicious by FANCI. Besides, as shown in Fig. 7 (b), since VeriTrust detects HTs by finding unused minterms and maxterm, DeTrust mixes trigger inputs with normal inputs to avoid unused malicious terms.

If the entire implicitly-triggered HT can be found and combined, it is possible for FANCI and VeriTrust to identify the HT. However, simply extending FANCI and VeriTrust faces several limitations. First, the computational complexity is intolerable due to the exponential increase of space where HT could be inserted. Without a priori knowledge of how many sequential levels that HT trigger logics spread, all possibilities should be enumerated. Besides, a large number of false positives will be introduced. On one hand, for FANCI, by combing several combinational logic circuit blocks together, the size of the combined combinational logic circuit block is larger than previous separated ones. In this case, the number of inputs of the combinational logic circuit increases, and each input affects the output more weakly. Consequently, the false positive rate increases. On the other hand, for VeriTrust, the terms in the K-map increase exponentially with the increment of combinational logic circuit size, and naturally more unused minterms and maxterms will be flagged.

For all HTs, we have the following observations: first, unlike in normal circuits, where a signal may affect a large number of circuits, the HT trigger signals affect only the HT payloads. Besides, the signals in the intermediate stages of an implicitly-triggered HT can only affect the circuits in the next stage of the HT. Consequently, in the flip-flop level CDFG, nodes within an implicitly-triggered HT have small out-degrees. These observations can be summarized into Feature 4.

- **Feature 4:** The nodes in an implicitly-triggered HT have small out-degrees.

An example of Feature 4 is shown in Fig. 7 (a), where all nodes in the implicitly-triggered HT have only one successor.

With Feature 4, by defining a *threshold* for out-degree, separated parts of an implicitly-triggered HT can be found using the following matching algorithm:

1) Initializing candidate list with nodes whose successors are fewer than $threshold$.
2) In the candidate list, if two nodes are connected with an edge, or two nodes share a same successor, then combine them into a group.
3) For each group, scan all nodes in the flip-flop level CDFG. If one node's all predecessors or successors are in the same group, add it into the group.
4) If any new node was added in 3), go to 3).

Since the successors of the HT nodes in second-last stage are exactly the HT payload nodes, the out-degrees of these HT nodes equal to the number of HT payload nodes. If the number of HT payload nodes is smaller than the *threshold*, all the HT nodes in the second-last are found in step 1. Afterwards, in step 2 and step 3, all the nodes of the implicitly-triggered HT can be found. For example, in Fig. 7 (b), nodes named t_2, FF_1, FF_2, and FF_3 can be distinguished by step 1. In step 2, all these four nodes are combined into the same group. After that, in step 3, since all the successors of node d_1, d_2, and t_1 are in the same group, these 3 nodes are also added into the group. Finally, all nodes in the implicitly-triggered HT are combined into one group.

V. Experimental Results

In this section, we evaluate the effectiveness of FASTrust with both explicitly-triggered HT benchmarks from TrustHub benchmark suite [14] and implicitly-triggered HT benchmarks from DeTrust [15]. Since FASTrust reads in gate level netlists, we directly adopt the benchmarks provided in gate level netlists. For HT benchmarks provided in RTL code, we synthesize them with Synopsys Design Compiler before running FASTrust.

FASTrust is a single-thread program implemented in C++ language. The experiment platform is Ubuntu 14.04.1 LTS with an Intel Xeon E5-2690 CPU @2.90GHZ and 48GB RAM.

TABLE I
THRESHOLDS FOR DIFFERENT FEATURES

	Feature 1	Feature 2	Feature 3	Feature 4
Threshold	20	20	50	1
Meaning	Node group size	Node's in-degree	Node group's total in-degree	successor number

A. The Detection Capability

In this subsection, we evaluate the effectiveness of FASTrust on HTs from TrustHub benchmark suite [14] and DeTrust benchmarks [15].

The thresholds for different features used in the experiments are shown in Table I. The threshold for each feature is set based on an assumption that the basic functional verification is employed. Since $2^{20} \approx 1$ million, if a time-triggered HT

TABLE II
DETECTION CAPABILITY OF FASTRUST ON HARDWARE TROJANS

HT type	Benchmark	# of units (k)	# of nodes	Expected feature	Feature 1		Feature 2		Feature 3		Feature 4		Detected?	Runtime		Peak memory (MB)
					Candidates	Hit	Candidates	Hit	Candidates	Hit	Candidates	Hit		Building CDFG (ms)	Feature analysis (ms)	
Time-trig.	RS232-T300	0.3	112	Feature 1	**1**	**1**	33	32	2	1	0	0	Yes	6.181	0.034	2.844
	RS232-T500	0.3	112		**1**	**1**	33	32	2	1	0	0	Yes	5.584	0.084	2.848
	BasicRSA-T300	2.2	718		**4**	**1**	297	15	4	1	0	0	Yes	190.1	0.098	18.15
	BasicRSA-T400	2.4	726		**5**	**1**	394	14	5	1	0	0	Yes	175.1	0.123	17.76
Single-trig.	BasicRSA-T100	2.3	693	Feature 2	4	0	**332**	**32**	4	0	0	0	Yes	165.2	0.113	17.02
	s35932-T100	6.0	2285		0	0	**1**	**1**	0	0	0	0	Yes	232.7	0.121	31.13
	s35932-T200	6.0	2284		0	0	**6**	**6**	0	0	0	0	Yes	238.3	0.120	31.06
	AES-T400	163.5	7391		1	0	**11**	**1**	1	0	1	0	Yes	3598	0.377	1011
	AES-T500	163.6	7391		1	0	**8**	**8**	1	0	1	0	Yes	3608	0.374	1011
	AES-T600	163.1	7236		0	0	**1**	**1**	0	0	0	0	Yes	3532	0.286	1010
	AES-T700	124.6	7255		0	0	**1**	**1**	0	0	1	0	Yes	2855	0.553	757.2
Seq.-trig.	PIC16F84-T100	1.3	534	Feature 3	0	0	125	0	**6**	**1**	17	0	Yes	11.82	11.77	8.004
	PIC16F84-T200	1.3	534		0	0	125	0	**6**	**1**	17	0	Yes	12.43	11.76	7.975
Impli.-trig.	s15850	10.6	773	Feature 4	3	0	168	0	8	0	**1**	**1**	Yes	12.35	0.885	59.80
	Wb_conmax	73.3	6646		1	0	1929	0	16	0	**2**	**2**	Yes	407.6	193.1	359.0
	Or1200_Ctrl	1.4	809		1	0	5	0	6	0	**2**	**2**	Yes	10.55	0.723	6.715

aims to evade the functional verification which runs simulation for 1 million cycles, it must contains more than 20 bits. In this case, the size of the suspicious loop group is larger than 20. Consequently, the threshold for Feature 1 is set to 20. Likewise, to reduce the activation probability and evade the functional verification, the trigger inputs and state machines in single-triggered HT and sequential-triggered HTs tend to be large. In this paper, the threshold for Feature 2 and 3, i.e. the in-degree of a node and the total in-degree of a loop group are set to 20 and 50, respectively. For Feature 4, the threshold actually should be set according to possible HT payloads number. In this paper, the threshold for Feature 4 is set to 1.

Experimental results of FASTrust's detection capability on HTs are shown in Table II. The 3rd column shows the number of gates in each benchmark and the 4th column presents the number of nodes in each benchmark's flip-flop level CDFG. It is apparent that the size of a flip-flop level CDFG is much smaller than its corresponding gate level netlist. In Table II, "Candidates" means the number of nodes identified as suspicious by each feature, and "Hit" means how many nodes of candidates are actually the nodes in HTs. For Feature 1, Feature 3 and Feature 4, the HT candidates are loop groups. For Feature 2, the HT candidates are nodes.

For the RS232 benchmark set, the internal 32-bit counters are implemented which can be reset when counting to a specific number. Therefore, each bit in the counters is affected by all other 31 bits and thus each node in the loop groups has at least 31 predecessors. In this case, all nodes within the counters are detected with both Feature 1, Feature 2, and Feature 3.

For the BasicRSA benchmark set, though the inserted HTs are detected, the false positives with Feature 2 are high. The reason is that there are a large number of adders and multipliers in these benchmarks, and the output nodes of those large adders or multipliers naturally have many predecessors. In this case, a great number of nodes in adders and multipliers have large in-degrees and are marked as suspicious by Feature 2.

For the s35932 benchmark set, the flip-flop level CDFGs are quite sparse and few normal nodes have large in-degree. Consequently, the single-triggered HTs in these benchmarks are identified with no false positive.

For the AES benchmarks, HTs are identified with low or no false positive. Although there are about 163 thousand gates in the gate level netlists of the AES benchmarks, the time consumptions for building flip-flop level CDFGs are still less than 4 seconds. These time consumption results strongly demonstrate the scalability of FASTrust.

Results on the PIC16F84 benchmarks demonstrate the effectiveness of Feature 3: the state machines in HTs are identified. Unlike results of benchmarks mentioned above, analysis with Feature 4 reports a few HT candidates, which means there are some normal nodes in these benchmarks have only one successor.

Finally, for the DeTrust benchmarks which are able to evade the detection of FANCI, the implicitly-triggered HTs in them are identified precisely with Feature 4. However, the analysis with Feature 2 results in a large number of false positives.

In summary, for all evaluated HTs, FASTrust shows to be able to detect them with extremely short runtime. For example, for the AES-T400 benchmark which has about 163k units, the total runtime of FASTrust including building the flip-flop level CDFG and feature analysis is about 3.6 seconds, which is impossible for HT detection using FANCI and VeriTrust. For FANCI and VeriTrust, the typical runtime is about a few hours to tens of hours.

Feature analysis with Feature 2 results in very high false positive rate, while other features shows to be much more effective. This is because in the flip-flop level CDFG, most of information on combinational logic are omitted and the sequential information is emphasized. In this case, when

detecting a single-triggered HT, the relations between different nodes cannot be utilized, since a single-triggered HT consists of exactly one combinational logic circuit block in the gate-level netlist and one node in the flip-flop level CDFG.

Results on benchmarks BasicRSA-T300 and BasicRSA-T400 reveal the reason why circuit-based methods such as FANCI [12] and VeriTrust [13] can detect typical time-triggered HTs. In a counter, each bit is affected by the lower-order bits, and thus, the number of inputs of the fan-in logic cone associated with a higher-order bit tends to be large. In this manner, for a very high-order bit, FANCI can detect several inputs that weakly affect the output in its corresponding combination logic block circuit, and VeriTrust can detect an unused minterm if this bit has not been changed during the functional verification.

B. Detection of implicitly-triggered HTs

In Table II, experimental results show that the implicitly-triggered HTs in three DeTrust benchmarks are detected by Feature 4 with the matching algorithm. For these experiments, detailed experimental results are shown in Table III. In the 4th column, the number of HT nodes in the node groups detected with Feature 4 are presented. The numbers in the 5th column are the number of total nodes in the HT. For each HT candidate node group, all nodes in it are merged into one node and then be analyzed with Feature 2. Results in the 6th column show that these HTs are detected with Feature 2 when merged.

Results show that, for benchmark s15850, all the HT nodes are found. For benchmark Wb_conmax and Or1200_ctrl, two main branches of the inserted implicitly-triggered HTs are founded. Though nodes of the two HTs in two benchmark Wb_conmax and Or1200_ctrl are not merged entirely, detection with Feature 2 also identifies their main branches as suspicious.

The threshold for Feature 4 actually reflects the assumed maximum HT payload number. For example, when the HT payload number is exactly one, feature analysis using Feature 4 with a threshold = 1 can guarantee detecting this HT. Though HT-affected signals have limited usage, it is possible that they affect more than one successive circuit and have multiple payloads. In this manner, we can increase the threshold for feature matching with Feature 4 to detect HTs. With different thresholds, the results of HT candidate numbers are shown in Table IV. The results confirm the intuitive conclusion: the number of suspicious groups increases along with the increase of the threshold. It should be noted that though a smaller threshold introduces fewer false positives, it does not necessarily lead to a better detection result. For example, when the threshold is 2, the branches of the implicitly-triggered HT in Or1200_ctrl are merged into one.

TABLE III
DETAILED DETECTION RESULTS ON IMPLICITLY-TRIGGERED HTS WITH FEATURE 4

Benchmark	Feature 4		# of hit nodes	# of HT nodes	Detected with Feature 2 after merging?
	Candidates	Hit			
s15850	**1**	**1**	13	13	Yes
Wb_conmax	**2**	**2**	13	14	Yes
Or1200_ctrl	**2**	**2**	29	40	Yes

TABLE IV
NUMBER OF IMPLICITLY-TRIGGERED HT CANDIDATES DETECTED WITH FEATURE 4 WITH DIFFERENT THRESHOLDS

Benchmark	Threshold			
	1	2	3	4
s15850	1	14	26	30
Wb_conmax	2	17	16	16
Or1200_ctrl	2	12	15	18

VI. LIMITATIONS AND FUTURE WORK

A. Limitations

Though FASTrust has been shown to be able to detect all the HTs evaluated in this paper, it does not mean that no HTs can defeat it. Actually, attackers can still exploit the weaknesses of FASTrust to evade it. In this section, we analyze the limitations of FASTrust in detail.

First, FASTrust is a static HT identification technique, and how to set the thresholds in order to avoid false negative is not a trivial task. For example, since we initialize the candidate list with the nodes whose successor number is smaller than a threshold when detecting implicitly-triggered HTs with Feature 4, if all nodes in the implicitly-triggered HTs have a large number of successors, the separated nodes of the HT cannot be grouped together. A specific example is that an implicitly-triggered HT changes a 32-bit output at the same time when triggered, then all nodes in its second-last stage have 32 successors. In this case, the implicitly-triggered HT may evade FASTrust when the threshold is also smaller than the number of HT nodes in every intermediate stage.

Second, the false positive rate tends to be another limitation. Unlike in malware detection, where feature of each virus type is unique, the features of HTs have less uniqueness. Consequently, false positives are inevitable. Since HT designs and trust verification techniques are like arm race, not only the defending techniques can update, but new HTs can also be designed. To cover more HT types, more features should be continuously added into the HT feature database in the future, and thus the overall false positive rate will monotonously increase. Though it is possible to add a preprocessing to reduce the false positive rate, preprocessing cannot completely resolve this issue.

Moreover, the selection of features and the selected circuit description level may also limit the development of FASTrust. Since FASTrust is based on feature analysis, the effectiveness of each feature greatly influences the overall performance of FASTrust. For example, Feature 2 may flag a large set of nodes, since it concerns no functionality but only the input number of a combinational logic circuit.

B. Future work

As discussed earlier, FASTrust still suffers from several limitations. We plan to further improve the performance of FASTrust in the future and overcome or alleviate most of these limitations.

First, as mentioned above, the false positive rate will monotonously increase along with the extension of HT feature database and the improper selection of features may introduce more false positives. The preprocessing stage, which can relax later analysis effort and reduce false positives, will be explored.

Second, we plan to propose a method to theoretically evaluate the significance and stability of the selected features to select optimal features. The significance means the capability that a feature can distinguish between benign circuits and malicious circuits. The stability means the capability of a feature that identifies an HT type with different implementations. For example, in this paper, we assume that time-triggered HTs use counters to delay the activation. However, it is also possible for a time-triggered HT to make use of linear feedback shift registers (LFSRs) as its trigger circuit. If Feature 1 is able to detect all different implements of time-triggered HTs, we call that it is stable on this kind of HTs. Otherwise, more detailed classification should be employed and the time-triggered HT type should be divided into several sub-types. The aim for feature selection is achieving both high significance and stability.

In addition, the adoption of cross-level features is also considered for possible future improvement. As mentioned in Section III, HT features can be extracted at various levels. In this paper, we only consider the features at the flip-flop level CDFG, and thus Feature 2 suffers from a natural disadvantage: The flip-flop level CDFG contains no information on the combinational logic, only the input number of each combinational logic circuit block is recorded. Since FASTrust builds the flip-flop level CDFG according to the gate level circuit, it is possible to combine two description levels and use cross-level features to detect certain HTs.

VII. Conclusion

In this paper, we propose a novel 3PIP trust verification framework, named FASTrust, which employs a set of feature analysis in the flip-flop level CDFG of 3PIPs to detect HTs. With the global view, FASTrust models the relationship between different parts of a 3PIP and thus can detect implicitly-triggered HTs. Experimental results demonstrate that FASTrust is able to detect all explicitly-triggered HTs and implicitly-triggered HTs evaluated in this paper within short time. FASTrust is able to not only identify existing explicitly-triggered and implicitly-triggered HTs appeared in the literature in an efficient and effective manner, but also be scalable to defend against future and more stealthy HTs by adding new features to the system.

References

[1] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *Design Test of Computers, IEEE*, vol. 27, no. 1, pp. 10–25, Jan 2010.

[2] Y. Jin and Y. Makris, "A proof-carrying based framework for trusted microprocessor ip," in *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, Nov 2013, pp. 824–829.

[3] E. Love, Y. Jin, and Y. Makris, "Proof-carrying hardware intellectual property: A pathway to trusted module acquisition," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 1, pp. 25–40, Feb 2012.

[4] B. Singh, A. Shankar, F. Wolff, C. Papachristou, D. Weyer, and S. Clay, "Cross-correlation of specification and rtl for soft ip analysis," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–6.

[5] B. Singh, A. Shankar, F. Wolff, D. Weyer, C. Papachristou, and B. Negi, "Knowledge-guided methodology for third-party soft ip analysis," in *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*. IEEE, 2014, pp. 246–251.

[6] J. Rajendran, H. Zhang, O. Sinanoglu, and R. Karri, "High-level synthesis for security and trust," in *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*, July 2013, pp. 232–233.

[7] X. Cui, K. Ma, L. Shi, and K. Wu, "High-level synthesis for run-time hardware trojan detection and recovery," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, June 2014, pp. 1–6.

[8] A. Al-Anwar, Y. Alkabani, M. El-Kharashi, and H. Bedour, "Hardware trojan detection methodology for fpga," in *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, Aug 2013, pp. 177–182.

[9] M. Banga and M. Hsiao, "Trusted rtl: Trojan detection methodology in pre-silicon designs," in *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, June 2010, pp. 56–59.

[10] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, June 2011, pp. 67–70.

[11] M. Hicks, M. Finnicum, S. King, M. Martin, and J. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Security and Privacy (SP), 2010 IEEE Symposium on*, May 2010, pp. 159–172.

[12] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 697–708.

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

[14] Trust-Hub benchmarks. https://www.trust-hub.org/resources/benchmarks.

[15] J. Zhang, F. Yuan, and Q. Xu, "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. ACM, 2014, pp. 153–166.

[16] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th annual Design Automation Conference*. ACM, 2001, pp. 530–535.