Hardware Trojan Detection and Functionality Determination for Soft IPs

Thao Le
Computer Science and Computer Engineering Department
University of Arkansas
Fayetteville, U.S.A.
tpl001@uark.edu

Lucas Weaver
Computer Science and Computer Engineering Department
University of Arkansas
Fayetteville, U.S.A.
lcweaver@uark.edu

Jia Di
Computer Science and Computer Engineering Department
University of Arkansas
Fayetteville, U.S.A.
jdi@uark.edu

Shaojie Zhang
Computer Science Department
University of Central Florida
Orlando, U.S.A.
shzhang@cs.ucf.edu

Yier Jin
Electrical and Computer Engineering Department
University of Florida
Gainesville, U.S.A.
yier.jin@ece.ufl.edu

***Abstract*— Due to the increasing complexity of hardware designs, third-party hardware Intellectual Property (IP) cores are often incorporated to alleviate the burden on hardware designers. However, the prevalent use of third-party IPs has raised security concerns such as hardware Trojans. These Trojans inserted in the soft IPs are very difficult to detect through functional testing and no single detection methodology has been able to completely address this issue. Based on a Register-Transfer Level (RTL) soft IP analysis method named Structural Checking, this paper presents a hardware Trojan detection methodology and tool by detailing the implementation of a Golden Reference Library for matching an unknown IP to a functionally similar Golden Reference. The matching result is quantified in percentages so that two different IPs with similar functions have a higher percentage match. A match of the unknown IP to a whitelist IP advances it to be identified with a known functionality, while a match to a blacklist IP causes it to be detected as Trojan-infested.**

Keywords— Asset, Structural Checking, Golden Reference Library, Hardware Trojan Detection

I. INTRODUCTION

As the complexity of integrated circuits (ICs) keeps increasing, it is no longer financially efficient to design everything in-house from scratch. Acquiring and integrating third-party Intellectual Property (IP) blocks have become common practice. However, since these IPs are not designed in-house, their security and integrity cannot be guaranteed. Hardware Trojans may be inserted into these soft IPs, which pose a great threat to a large number of important applications, such as defense and financial systems. Hardware Trojans are the malicious insertion or modification triggered by a specific event or sequence of events, resulting in a payload compromising the operation of the circuit. Potential payloads include denial of service, information leakage, or data tampering, inducing great damage to the system incorporating this IP and completely compromising the higher-level security mechanisms.

This work is supported by NSF award CNS-1703602 and CNS-1812071.

Many solutions have been proposed focusing on hardware Trojan detection. One approach is to analyze side-channel signals in order to identify the impact of hardware Trojans. Multiple side-channel characteristics have been analyzed in research, such as power [1], current [2], and timing [3]. Trojans are revealed by comparing each of these characteristics to those of a Trojan-free design. Another technique integrates sensors to the empty space of a layout. Sensors used in [4] provide "self-authentication" by measuring circuit delays, while similar research in [5] measures path delays. Additionally, an on-chip ring oscillator network discussed in [6] performs power analysis that aids in Trojan detection.

In contrast to those solutions analyzing circuit characteristics, several other methods focus on activating potential Trojans. For example, the research in [7] utilizes randomized test vectors generated in a probabilistic manner. Similarly, the research in [8] applies test vectors designed to activate nets that are rarely activated, as they could be the targets of a Trojan. Also, by narrowing down the potential regions for Trojan detection and testing these regions thoroughly, the research in [9] finds some success in identifying Trojans.

Another strategy for Trojan detection focuses specifically on the security of third-party IPs and how to provide improved trust to these designs. For example, in [10] researchers use testing methods to identify vulnerable portions of the third-party IP. Additionally, the research in [11] uses formal verification and sequential Automatic Test Pattern Generation (ATPG) for the same purpose. Another technique in [12] presents a strategy of Design-for-Trojan-Test in order to limit the abilities of an attacker to insert Trojan triggers. The research in [13] involves the comparison of IP blocks with a similar function in order to identify malicious logic. FANCI tool in [21] provides a statistical analysis to determine backdoor signals. Last but not least, the research in [14] identifies vulnerable signals by applying statistical analysis to determine the observability of the signal.

Different from the research in [13] which compares two untrusted IPs to detect Trojans, the Golden Reference Library Matching method in [16] compares an untrusted Register-Transfer Level (RTL) IP asset pattern and functionality with those of a collection of trusted IPs in a Golden Reference Library (GRL). In term of hardware Trojan scenarios, Trojan detection methods in [16] uncover case-specific hardware Trojan signals or components while FANCI [21] flags suspicious primary signals based on their statistically rare activity. While both tools achieve the Trojan detection goal, the method in [16] gives the users extra benefits by identifying the functionality of the unknown IP. This methodology is based off the *Structural Checking* methodology [15]. Since *Structural Checking* does not perform any simulation, it serves as an assistant tool to the end user for Trojan detection.

II. BACKGROUND AND DEFINITION

A. Structural Checking and GRL Matching Flow

The *Structural Checking* methodology for detecting hardware Trojans consists of several distinct steps. The first step, VHDL Parsing, is to analyze the VHDL code of a RTL soft IP and create a structure in memory of the internal connections and expressions. The next step consists of the assignments of roles to all signals of the design. In the context of *Structural Checking*, these roles are termed *assets*. This is performed through a graphical user interface, allowing users to easily assign assets to the primary I/O ports. Following the asset assignment, filtering is performed by passing assets along direct connections extracted during the VHDL parsing. The results of asset filtering are used in the determination of the IP's functionality. This step is termed *Golden Reference Library (GRL) Matching*. Finally, the results of GRL matching are used for functionality determination and Trojan detection.

TABLE I. EXTERNAL ASSETS IN EACH CATEGORY

Asset				
Data	***Timing***	***System Control***	***Specific System Control***	***Miscellaneous***
DATA_COMPUTATIONAL	*STATUS*	*SET*	*MEMORY_OP*	*CRITICAL*
DATA_MEMORY	*DONE*	*RESET*	*DATA_OP*	*COMPONENT*
DATA_PERIPHERAL	*HOLD*	*READ*	*INTERRUPT_OP*	*ADDRESS_SENSITIVE*
DATA_COMMUNICATION	*READY*	*WRITE*	*PROGRAM_COUNTER_OP*	*CONSTANT*
DATA_ENCRYPTION	*BUSY*	*SELECT*	*INTERRUPT_CONTROL*	*KEY*
DATA_SENSITIVE	*COUNT*	*EXECUTE*	*PERIPHERAL_CONTROL*	*REGISTER*
	WAIT	*LOAD*	*REGISTER_FILE_CONTROL*	*PROGRAM_COUNTER*
	TIMER_CONTROL	*MODE*	*COMMUNICATION_CONTROL*	*ERROR_HANDLING*
	CLOCK_CONTROL	*ENABLE*	*COMMUNICATION_PROTOCOL*	*EXCEPTION_HANDLING*
	SYSTEM_TIMING	*HANDSHAKING*	*COMMUNICATION_STATUS*	*STATE*
	SUBSYSTEM_TIMING	*SHIFT*	*INTERRUPT*	
		INSTRUCTION		
		SYSTEM_CONTROL		

TABLE II. INTERNAL ASSETS AND THEIR DESCRIPTION

Asset	**Description**
PROCESS_SENSITIVE	Assigned to a signal in a process sensitivity list.
PROCESS_OPERATION_SENSITIVE	Assigned to a signal being modified in a process block.
CONDITIONAL_DRIVING	Assigned to a signal in a conditional statement.
CONDITIONAL_DRIVEN	Assigned to a signal being modified in a conditional block.
CONCURRENT_DRIVING	Assigned to a signal driving another signal in a concurrent statement.
CONCURRENT_DRIVEN	Assigned to a signal being driven by another signal in a concurrent statement.
CC_OPERATION_SENSITIE	Assigned to a signal being driven by two or more signals and logic operations.

B. Assets

Critical to the *Structural Checking* process are the concept of assets and asset patterns of a soft IP. Assets define the role of a signal while asset pattern is the accumulation of assets in a design. Asset patterns are generated by asset filtering and are important for functionality matching.

1) Asset Definition

Introduced originally in [15], assets mean to capture the purpose/use/contribution of the signal using descriptions of common functionalities that a signal may possess. In this way, end-users gain a better understanding of the roles of each signal. There are external and internal assets.

a) External Assets

External assets are the set of potential functionalities assigned to the primary port signals of a soft IP by the user. They were created with the purpose of encompassing all possible roles that a port signal may assume in a design. A total of 50 external assets have been developed and categorized based on functionalities, as shown in Table I.

b) Internal Assets

Internal assets are automatically assigned by the tool to both primary port signals and internal signals. Internal assets are assigned based on a signal's logical role in the VHDL code and are broken down into three major categories — *concurrent*, *process*, and *conditional* [20]. Table II shows a list of internal assets and their descriptions.

2) Asset Pattern

After assets are assigned to the signals, they are filtered along direct connections in order to populate the entire set of signals with a collection of assets. The set of assets assigned to a specific signal path is termed an asset trace. The entire collection of asset traces of a design is termed an asset pattern.

C. Asset Filtering

The idea of asset filtering is comparable to the taint analysis proposed in [17]. The taint value propagates from the input bit to the dependent output bit of a logic gate in the gate-level netlist. Similarly, the external assets assigned to primary inputs are filtered to the next signal connections until they reach the dependent primary outputs. Then, the external assets previously assigned to primary outputs are filtered backward to the primary inputs that they are dependent on. The filtering rule for the internal assets is slightly different form the external asset filtering rule. The internal assets in the *process* category propagate within the process block boundary of the VHDL code. Similarly, the *conditional* internal asset category propagates within the conditional block boundary of the VHDL code. Finally, the internal assets in the *concurrent* category follow the same filtering rule as external asset filtering. The entire filtering process operates at RT-Level.

D. Golden Reference Libaray (GRL)

The initial entries of GRL are various small designs collected from OpenCores [19]. Since they are small, exhaustive verification is feasible. More entries are added from in-house designs. *Structural Checking* is then applied to generate asset patterns for all entries and functionalities are assigned manually.

TABLE III. ASSET PATTERN CHARACTERISTIC WEIGHT

Asset Pattern Characteristic	Weight
input port signal external asset	3×
output port signal external asset	3×
internal signal external asset	1×
input port signal internal asset	1×
output port signal internal asset	1×
internal signal internal asset	1×

```
i2c_master
Best Match: i2c : 92.5 (75.0, 100.0, 100.0, 100.0, 100.0, 100.0)
Communication Match: 62
Computational Match: 26
Decoder/Encoder Match: 27
Interrupt Unit Match: 50
Control Generation Match: 13
Peripheral Match: 3
Register File Match: 17
Encryption Unit Match: 20
Shift Register Match: 17
Timing Match: 20
```

Fig. 1. A portion of a matching report file

The GRL contains asset patterns of trusted soft IPs collected as GRL files. Each GRL file has an associated functionality. The functionality matching of *Structural Checking* utilizes the set of GRL files to match an unknown design to a functionality associated with the closest matched asset pattern. The current GRL contains a total of 122 files/entries, which are the asset patterns of distinctive designs with (blacklist) and without (whitelist) Trojan inserted. Note that these entries already consider the situation where a port signal may be assigned different assets.

Each GRL file contains six characteristics that are analyzed independently. The asset pattern can first be broken down into the external asset pattern characteristic and the internal asset pattern characteristic, depending on whether the asset pattern characteristic contains external or internal assets. Each of these asset pattern characteristics can be further broken down into primary inputs, primary outputs, and internal signals. It is necessary for each of these components to be analyzed independently in order to detect possible Trojans aimed at each specific portion of the design.

III. GOLDEN REFERENCE MATCHING METHODOLOGY

The GRL is the standard merit for identifying an unknown design's functionality, which is critical for Trojan detection. In order to determine the functionality of an unknown IP precisely, the asset pattern of that IP needs to be matched properly to GRL entries.

A. Basic Asset Trace Matching

After asset filtering, the target IP has six asset pattern characteristics, the same as a GRL entry. These characteristics are compared in pairs. For example, the input port signal

external asset trace of the unknown IP is compared to the same characteristic of each GRL entry. The matching result is the percentage of the identical portion between two characteristics. The same process is applied to other characteristics. Several examples are included in Table III. In Table III, case number 1 is the 100% match because the asset traces of both the GRL entry and the unknown IP are the same. In case number 2, 0% is the result of two completely different asset traces. Case number 3 shows the result of 67% because two out of three assets in the unknown IP's asset trace are identical to the asset trace of the GRL entry. The last case presents the scenario where one out of the two assets is the same on both asset traces, so the result yields 50%. If each case represents an asset pattern characteristic, the final matching result is 54.25% as the average of the four cases.

B. Partial Asset Trace Mathcing

The partial asset trace matching algorithm is developed to gain more precisions in matching assets between two traces of the same characteristic. This is due to the fact that assets in the two asset traces often originate from the same category. An asset that represents a specific role is considered as 50% match to an asset that represents a general role in the same asset category. For instance, the match result of a *SYSTEM_CONTROL* asset and a *CLOCK_CONTROL* asset is 50% because they are listed in the same system control category, which would be 0% using the basic matching algorithm and would not present the similar nature of the two assets. Table IV illustrates different scenarios where partial matching is applied.

C. Pattern Mathcing Compilation

Once all asset pattern characteristics of the unknown IP have been matched to the corresponding asset pattern characteristics of the GRL entry, a final match value is determined, which is the main factor for the functionality of the matched GRL entry to be assigned to the unknown IP. Even though each asset pattern characteristic contributes to the overall match value, not all characteristics are weighted equally. The weighting for each characteristic is performed experimentally by first recognizing that there are multiple implementations representing the same functionality. The internal characteristics of a functionality, which includes all internal asset characteristics along with the external assets filtered to internal signals, have the potential to be vastly different from another design with the same functionality. For this reason, the asset pattern characteristics related to internal characteristics are weighted less than the port signal external asset characteristics. In addition, the experimental results show the external asset pattern characteristics have a greater influence in the functionality determination than the internal characteristics. Hence, they have a higher weight. Table V shows the weighting applied to the associated characteristic. After applying these weights to the asset pattern characteristics, a final highest match value is determined for each GRL entry to the unknown IP. The functionality of the GRL entity with the highest match value is then assigned to the unknown IP.

D. Functionality Matching

To aid in the asset pattern matching algorithm, a functionality determination algorithm is developed to precisely identify the functionality of an unknown IP. The functionality-specific external asset is considered as the major indication of the potential functionality for the unknown IP. Functionality-specific assets are assets that have a clear link to a functionality category as defined previously. Any general-purpose assets are disregarded for consideration in this matching method. Thus, for any unknown IP containing a functionality-specific external asset, the GRL entries with the corresponding functionality are weighted 1.5 times higher than others. This weight number is calculated based on observation experiments throughout testing various IPs collect from open source website [18-19]. For example, if the *DATA_ENCRYPTION* asset is found in an input port signal external asset pattern, the percentage of the asset pattern that contains *DATA_ENCRYPTION* in all GRL entries of *encryption unit* functionality is multiplied by 1.5.

E. Matching Report

Structural Checking provides a detailed breakdown of the asset pattern characteristics match along with the results from other categories of designs. The results are output in the form of a human-readable file, including percentage matches for all functionality categories. This gives the user the opportunity to understand the matching procedure as well as the rationale for a design being matched to a specific functionality. An example of matching result file is in Fig. 1. The first line indicates the

TABLE IV. BASIC ASSET TRACE MATCHING EXAMPLES

Case	Unknown Design Asset Traces	GRL Entry Asset Traces	Match
1	*DATA_MEMORY, CRITICAL*	*DATA_ MEMORY, CRITICAL*	100%
2	*DATA_ MEMORY, CRITICAL*	*SYSTEM_CONTROL*	0%
3	*DATA_ MEMORY, CRITICAL*	*DATA_ MEMORY, CRITICAL, SYSTEM_CONTROL*	67%
4	*DATA_ MEMORY, SYSTEM_CONTROL*	*DATA_ MEMORY, SYSTEM_TIMING*	50%

TABLE V. PARTIAL ASSET TRACE MATCHING EXAMPLES

Case	Unknown Design Asset Traces	GRL Entry Asset Traces	Match
1	*SYSTEM_CONTROL*	*CLOCK_CONTROL*	50%
2	*DATA_ MEMORY, DATA_ SENSITIVE*	*DATA_ SENSITIVE*	50%
3	*DATA_ MEMORY*	*DATA_COMPUTATIONAL*	0%
4	*ENABLE, SET*	*SET, SYSTEM_CONTROL*	75%
5	*RESET*	*SET, SYSTE*	

unknown entity being analyzed. The second line presents the highest percentage match GRL entry within current GRL, and the percentage match of each asset pattern characteristic. The remainder lists the average percentage match of each functionality in the GRL with respected to the entity.

IV. TROJAN DETECTION USING GOLDEN REFERENCE LIBRARY MATCHING

A. Detection Algorithm

Trojan detection in the context of *Structural Checking* is a gradual process that analyzes the individual components of the target IP before determining the presence of a hardware Trojan. Each of these steps contributes to the identification of Trojan triggers and payloads.

1) Abnormal Asset Pattern Identification

This method is the analysis of external and internal asset traces. Certain combinations of assets found within asset traces expose the inclusion of a Trojan. Moreover, a Trojan can be suspected if a filtered asset of this signal is not in the same set as initially assigned.

One type of Trojan attacks that can be detected using this method is the denial of service attack involving a timing signal. For instance, a *set* or *reset* signal may disable the clock signal of a synchronous design. This can be detected by searching for the *SET* or *RESET* asset in the clock's asset trace after the filtering process. Another type of attack that can be uncovered by this method is cipher key leakage, for which a Trojan directs the secret key to the primary output. This can be detected by checking if the output signal's asset trace of the encryption unit has the *KEY* asset.

2) Trojan functionality identification

The second Trojan detection method is through functionality identification and assignment. During the process of GRL matching, *Structural Checking* finds the GRL entry that has the highest match. If the unknown IP contains a known Trojan in the blacklist, a blacklisted functionality will be assigned to that unknown IP. Otherwise, a whitelist functionality will be assigned. In addition, *Structural Checking* uses the GRL to identify all the sub-entities of the IP in order to reveal suspicious nets between those sub-entities. The suspicious nets are often found to be triggers for information-leakage Trojans.

3) Trojan Detection Report

In a situation where one or more potential Trojans are identified, a readable text file is generated to inform the user. This Trojan detection report includes the target design's entity name, Trojan types, instance name and signals affected by the Trojan. Fig. 2 shows a portion of a Trojan detection report of the entity *RSACypher* which is a blacklist Trojan-infested RSA encryption unit. The Trojan found in this *RSACypher* is *KEY_LEAK* meaning that the cipher key is leaked from the design and the signal affected by this Trojan is *inExp*.

B. Results and Various Trojan Detection Examples

1) Overall Result Analysis

A total of 21 RTL IPs from both Trust-Hub [18] and OpenCores [19] are included in the tests. Among these, for 17 Trojan-infested IPs, there are 27 implementations of denial-of-service and key/data leakage payload. All Trojan-infested IPs are correctly identified Trojan-infested which yields the false negative rate 0%. Only one Trojan-free testing IP is identified as Trojan-infested. Therefore the false positive rate is 4.7%. Two out of the 21 testing IPs are described below to further explain the Trojan detection methods.

2) Crypto Core AES-T600

AES-T600 is a Trojan-infested design obtained from Trust-Hub. In addition to the original cipher key leakage Trojan, two other Trojans are added, i.e., a time bomb counter and an additional shift register. After the parsing process, assets are assigned to *AES-T600* primary port signals. Following the assignment process, the filtering process generated the asset pattern for *AES-T600*. This pattern is then used to match to other patterns in the GRL in order to obtain a functionality for the design. In this case, the functionally of *AES-T600* is correctly identified as *ENCRYPTION_UNIT*. Then, multiple Trojan detection algorithms are used to detect Trojans. The parsing process, filtering process and Trojan detection process takes on average 6 second, 341second, and less than 1 second, respectively, on a 2.4GHz Duo processor PC with 16GB of RAM.

The first Trojan detected is the time bomb through the blacklisted *TROJAN_TRIGGER* functionality. *Structural Checking* compares asset pattern of this *AES-T600* to both whitelisted and blacklisted functionalities in GRL to get the highest percentage match. Fig. 2a illustrates the time bomb trigger which is inserted to the final encryption round of AES. After the time bomb is triggered, the cipher key, which is one of the two Trojan inserted, is leaked through the primary output of this *AES-T600*. Fig. 2b illustrates the cipher key leakage of the design. The last Trojan is the additional shift register. This shift register not only is used to perform shift operations, but also allows the attacker to perform power analysis side-channel attack on the AES. The shift register is triggered by a time bomb counter, which is the first Trojan. This shift register is identified as *TROJAN_SHIFT_REGISTER* via blacklist functionality matching.

```
TrojanTrigger: process (clk) is
  begin
    if rising_edge(clk) then
      TrojanCounter <= TrojanCounter + 1;
    end if;
  end process TrojanTrigger;
```

(a)

```
process (clk)
  begin
    if (clk'event AND clk = '1') then
      state_out <= (z0 & z1 & z2 & z3);
    elsif (trojancounter = x"44444444") then
      state_out <= key_in;
    end if;
  end process;
```

(b)

Fig. 2. A portion of (a) time bomb trigger code and (b) time bomb key leakage in the AES-T600 entity

3) Microcontroller c16

Another example is a Trojan-free microcontroller *c16* acquired from OpenCores [19]. The Trojan is inserted inside the UART communication unit, causing the transmission of incorrect data. The Trojan counter triggers the attack when it reaches a certain value as shown in Fig. 3. *Structural Checking* detected this Trojan using the "suspicious connection" algorithm to identify a counter in a communication unit. A portion of this report is in Fig. 4. It takes on average 18 seconds to parse, 11 seconds to filter, and less than 1 second to detect Trojans on the same computer.

```
if TrojanCounter(31) = '1' then
        SER_OUT <= "1";
else
        SER_OUT <= BUF(0);
end if;
```

Fig. 3. A Trojan inserted in UART unit of the microcontroller c16

```
Type of Trojan found:
TIME_BOMB_LEAKAGE_TRIGGER
Entity:
        UART_TX
Instance:
        tx
Signal:
        TrojanCounter
...
Blacklist Trojan Detected in Entity: TSC
  Trojan Functionality Type:
TROJAN_SHIFT_REGISTER
```

Fig. 4. A portion of Trojan detection report of c16

V. CONCLUSION

Golden Reference Library matching is an effective methodology that allows *Structural Checking* to detect hardware Trojan in a soft IP. The percentage of matching result is determined by the similarity of unknown and known asset pattern characteristics. The functionality is then determined by the functionality of the matched GRL entry. Hence, the matching process between an unknown IP asset pattern and trusted GRL asset patterns allows the Trojan detection process to be performed efficiently and effectively. The Trojan detection process using the GRL includes abnormal asset pattern identification and Trojan functionality identification. For future development, both blacklisted and whitelisted of the GRL, as well as the assets, can be easily expanded to improve the accuracy and resolution of the matching process, which is feasible because the process of creating a GRL entry is automated. More Trojan detection methods can be included to *Structural Checking* when those Trojan attacks are discovered.

REFERENCES

[1] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar, "Trojan Detection using IC Fingerprinting," Security and Privacy, 2007. SP '07. IEEE Symposium on, Berkeley, CA, 2007, pp. 296-310.

[2] X. Wang, H. Salmani, M. Tehranipoor and J. Plusquellic, "Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis," Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08. IEEE International Symposium on, Boston, MA, 2008, pp. 87-95.

[3] J. Li and J. Lach, "At-speed delay characterization for IC authentication and Trojan Horse detection," Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on, Anaheim, CA, 2008, pp. 8-14.

[4] A. Davoodi, Min Li and M. Tehranipoor, "A Sensor-Assisted Self-Authentication Framework for Hardware Trojan Detection," Design & Test, IEEE, vol. 30, pp. 74-82, 2013.

[5] F. Saqib, D. Ismari, C. Lamech and J. Plusquellic, "Within-Die Delay Variation Measurement and Power Transient Analysis Using REBEL," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 23, pp. 776-780, 2015.

[6] X. Zhang and M. Tehranipoor, "RON: An on-chip ring oscillator network for hardware trojan detection," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, pp. 1-6.

[7] S. Jha and S. K. Jha, "Randomization Based Probabilistic Approach to Detect Trojan Circuits," High Assurance Systems Engineering Symposium. HASE 2008. 11th IEEE, Nanjing, 2008, pp. 117-124.

[8] F. Wolff, C. Papachristou, S. Bhunia and R. S. Chakraborty, "Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme," Design, Automation and Test in Europe, Munich, 2008, pp. 1362-1365.

[9] M. Banga and M. S. Hsiao, "A region based approach for the identification of hardware Trojans," Hardware-Oriented Security and Trust. IEEE International Workshop on, Anaheim, CA, 2008, pp. 40-47.

[10] M. Banga and M. S. Hsiao, "Trusted RTL: Trojan Detection Methodology in Pre-Silicon Designs," in 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2010.

[11] X. Zang 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, San Diego, CA, 2011.

[12] Y. Jin, N. Kupp and Y. Makris, "DFTT: Design for Trojan Test," in Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on, Athens, 2010.

[13] T. Reece and W. H. Robinson, "Detection of Hardware Trojans in Third-Party Intellectual Property Using Untrusted Modules," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 3, pp. 357-366, March 2016.

[14] H. Salmani and M. Tehranipoor, "Analyzing circuit vulnerability to hardware trojan insertion at the behavioral level," in Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on, 2013, pp. 190-195.

[15] J. Yust, M. Hinds and J. Di, "Structural Checking: Detecting Malicious Logic without a Golden Reference," in Journal of Computational Intelligence and Electronic Systems, vol. 1, no. 2, p. 169177, 2012.

[16] L. Weaver, T. Le and J. Di, "Golden Reference Library Matching of Structural Checking for Securing Soft IPs," IEEE SoutheastCon, Norfolk, VA, 2016.

[17] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood and R. Kastner, "Theoretical analysis of gate level information flow tracking," in Proceedings of the 47th Design Automation Conference (DAC '10), New York, NY, USA, 2010, pp. 244-247.

[18] H. Salmani, M. Tehranipoor, and R. Karri, "On Design vulnerability analysis and trust benchmark development" IEEE Int. Conference on Computer Design (ICCD), 2013.

[19] http://opencores.org/

[20] M. Hinds, J. Brady and J. Di, "Signal Assets - a Useful Concept for Abstracting Circuit Functionality," in 2013 Government Microcircuit Applications & Critical Technology Conference (GOMACTech), 2013.

[21] A. Waksman, M. Suozzo, S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13). ACM. New York, NY, USA, 2013, pp. 697-708.