1-2018

Securing Soft IPs against Hardware Trojan Insertion

Thao Phuong Le
University of Arkansas, Fayetteville

Securing Soft IPs against Hardware Trojan Insertion

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Engineering

by

Thao Le
University of Arkansas
Bachelor of Science in Computer Engineering, 2012

May 2018
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

_____
Jia Di, Ph.D.
Dissertation Director

_____
James P. Parkerson, Ph.D.
Committee Member

_____
Dale Thompson, Ph.D.
Committee Member

_____
Jingxian Wu, Ph.D.
Committee Member

ABSTRACT

Due to the increasing complexity of hardware designs, third-party hardware Intellectual Property (IP) blocks are often incorporated in order to alleviate the burden on hardware designers. However, the prevalence use of third-party IPs has raised security concerns such as Trojans inserted by attackers. Hardware Trojans in these soft IPs are extremely 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) and gate-level soft IP analysis method named Structural Checking, this dissertation 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 high percentage match. A match of the unknown IP to a whitelisted IP advances it to be identified with a known functionality while a match to a blacklisted IP causes it to be detected with Trojan. Examples are given on how this methodology can successfully identify hardware Trojans inserted in unknown third-party IPs. In addition to soft IPs analysis, Structural Checking provides data flow tracking capability to help users discover vulnerable nodes of the soft IPs. Structural Checking is implemented with a graphical user interface, so it does not take users much time to use the tool.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Jia Di. He has been an excellent mentor. He has encouraged me to strive for success during my graduate program. He is a great example for mentorship and leadership. Again, I thank him for his support and guidance throughout my graduate career.

Many thanks to my committee members Dr. Dale Thompson, Dr. James P. Parkerson and Dr. Jingxian Wu for their guidance.

I would also like to thank George Holmes and Jason Crawley for their support.

DEDICATION

To my husband who countlessly listens to presentations, proof-reads my papers. He gives me feedbacks and advices. He supports me through challenging time at work and research.

Finally, to my Mom and Dad and my brother for supporting me to archive this degree.

CONTENTS

1. ..... INTRODUCTION 1

2. ..... BACKGROUND 4

2.1 JTAG System and Scan-chain Structure Overview 4

2.2 Asset 5

2.2.1 Asset Definition 5

2.2.2 External Asset 6

2.2.3 Internal Asset 11

2.2.4 Asset Filtering 12

2.2.5 Asset Trace and Asset Pattern 12

2.3 Functionality 13

2.4 Golden Reference Library 14

3. ..... METHODOLOGY AND IMPLEMENTATION 16

3.1 Asset Pattern Matching 16

3.1.1 Basic asset trace matching 16

3.1.2 Partial asset trace matching 17

3.1.3 Complete asset pattern matching 18

3.1.4 Functionality matching 19

3.2 Enhanced Golden Reference Matching for both RTL and Gate-level IPs 19

3.3 Standard Logic Gate Model and Netlist Pre-processing for Structural Checking .. 23

4. ..... RESULT AND ANALYSIS

..... 25

4.1 Trojan Detection Result for Gate-level versus RTL 25

4.2 Examples 29

4.2.1 Crypto core AES-T1900 29

4.2.2 Communication UART 30

4.2.3 Microcontroller c16 31

5. ..... ASSET APPLICATION IN DATA FLOW TRACKING

..... 34

5.1 Introduction 34

5.2 Data Flow Analysis 34

5.2.1 Malicious Signal Detection 34

5.2.2 Confidential Data Tracking 36

5.2.3 Critical Data Bypass Checking 38

6. .....UPDATED STRUCTURAL CHECKING

..... 39

7. ..... CONCLUSION AND FUTURE WORK

..... 45

REFERENCES 46

GLOSSARY

Abbreviations

RTL	Register Transfer Level
IPs	Intellectual Properties
Soft IPs	Intellectual Properties under RTL or gate-level
GR	Golden Reference
GRL	Golden Reference Library
SC	Structural Checking
DFF	D-Flip Flop
SDFF	Scan enable DFF
DFFSR	D-Flip flop with set and reset
LATSR	Latch with set and reset
HPM	Highest percentage matching

1 INTRODUCTION

As more hardware components are being outsourced to third-party entities due to economic considerations, the concept of hardware security has become a pressing matter in the minds of hardware designers. Since it is not financially efficient to design everything in-house from scratch, the integration of third-party Intellectual Property (IP) blocks has become necessary. However, since these IPs are not designed in-house, their integrity is not 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 insertion of malicious logic into a circuit triggered by a specific event or sequence of events and result in a payload compromising the operation of the circuit. Potential payloads include denial of service, information leakage, and data tampering attacks. A hardware Trojan inserted into a third-party IP can result in great damage to the system incorporating this hardware design and completely compromise its higher-level security mechanism.

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 that of a Trojan-free design. Another technique integrates sensors to the empty space of a layout. Sensors used in the research [4] provide "self-authentication" by measuring circuit delays, while similar research [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 approaches which analyze circuit characteristics, several other methods focus on activating potential Trojans. For example, randomized test vectors generated in a

probabilistic manner are used in [7] . Similarly, test vectors are applied in [8] for activating 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 introduced 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] use formal verification and sequential Automatic Test Pattern Generation (ATPG) for the same purpose. Another technique introduced 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 [14] provides a statistical analysis to determine backdoor signals. Finally, the research performed in [15] 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 signal or a circuit block of hardware Trojan while FANCI [14] flags suspicious primary signals based on their statistically rare activity. Both tools achieve the Trojan detection goal; however, their results are compromised when hardware Trojan is injected in a gate-level netlist. Therefore, another methodology is developed and published in [17] to mitigate hardware Trojan at the gate-level. Since then, the

methodology is improved to not only identify functionality but also detect Trojan of a gate-level netlist.

The rest of this dissertation is organized as following. Section 2 is basic knowledge of JTAG and scan chain structure. Section 2 also includes previous works of assets, Structural Checking (SC) tool and Golden Reference Library (GRL). Section 3 is the methodology of the enhanced GR matching. Section 4 is results and proof of concept examples. Section 5 is the independent data flow tracking research. Section 6 is the updated status of SC. The dissertation is concluded in Section 6.

2 BACKGROUND

2.1 JTAG System and Scan-chain Structure Overview

JTAG, i.e., the Joint Test Action Group, was invented in 1985 [18] as a better and reliable method to test and verify a complex integrated circuit (IC) or even a printed circuit boards (PCBs) after manufacturing. Later, IEEE 1149.1 – 1990 IEEE Standard Test Access Port and Boundary-Scan Architecture [19] became standard for JTAG in integrated circuits. The current supported standard in [20] is IEEE 1149.1 – 2013. This standard allows for assistance in testing, maintaining and supporting ICs. The benefits of JTAG ports offer IC manufacturers high throughput and low-cost testing. A central test access port (TAP) controller can support a daisy-chain of multiple blocks through boundary scan registers (BSRs) of those cores. Three basic modes in JTAG are BYPASS, INTEST and EXTEST. During the BYPASS mode, the block's BSRs allow data to pass through it. When a core is under INTEST mode, the core's BSRs shift test data in, wait for the data to be processed by the block's internal logic, then shift the test data out. The EXTEST mode is used to check the interconnection between boundary scan cells. Figure 1 is a diagram of threes cores with JTAG interface. The signals include TMS (Test Mode Select), TCK (Test Clock), TDI (Test Data In), and TDO (Test Data Out).

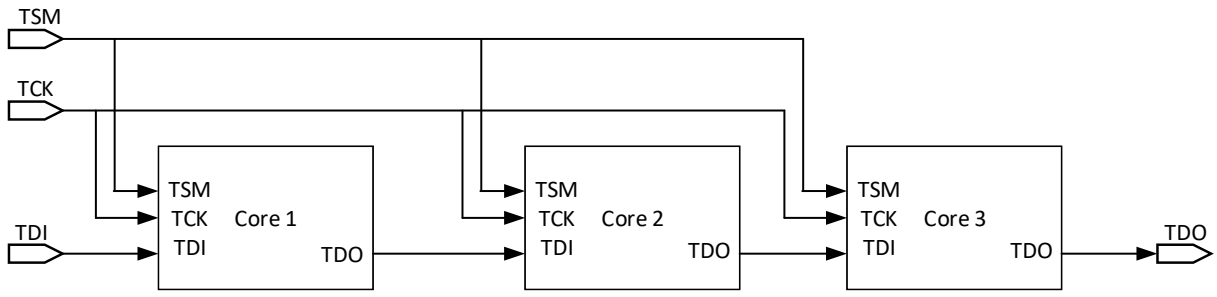

Figure 1. A general diagram of three circuit blocks in daisy-chain structure

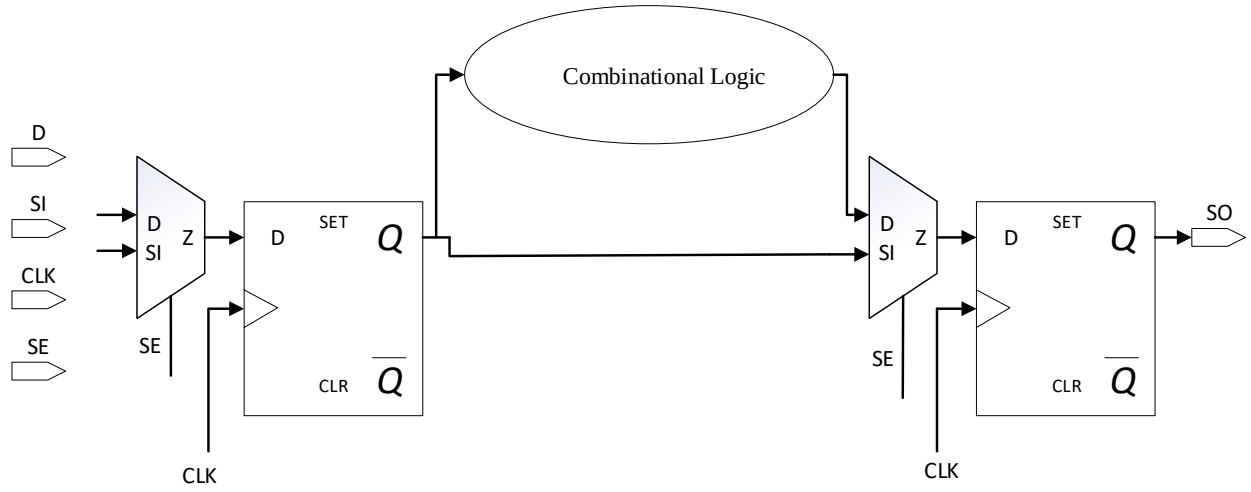

Figure 2 A simple scan-chain diagram

A typical scan-chain structure is as shown in Figure 2 where scan D-type flip flops (SDFFs) are connected as a shift register. A scan-chain netlist operates in 3 modes which are scan-in, capture and scan-out mode. Scan-in mode allows a test value from SI port of the first SDFF flow to the input of the test combinational logic. Once the output of the combinational logic is ready, the second SDFF captures the value. Then, the scan-out mode allows the test value to shift out.

2.2 Asset

2.2.1 Asset Definition

Critical to the Structural Checking process are the concepts of assets and asset patterns of a soft IP. From [18], assets are used to define the roles of a signal while asset pattern is the accumulation of assets in a design. In other words, asset concepts are means to capture the purpose/use/contribution of a signal to a soft IP. It is possible to assign multiple assets to a signal based on its role. In this way, end-users gain a better understanding of the roles of each signal. There are two categories of assets - external and internal.

2.2.2 External Asset

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. Initially, four assets were introduced in [18] serving as a proof-of-concept. Later in [16], fifty external assets were developed and categorized based on functionalities for providing a much broader coverage. Then, additional four TDI, TDO, TCK, TMS and TRST assets are added to provide coverage for JTAG and scan-chain enabled designs. In order to utilize these assets for the research work presented in this dissertation, they are categorized as shown from Table 1 to Table 5.

Table 1 Data external asset category

Data External Asset	Description
DATA_COMPUTATIONAL	Assigned to data signals of RTL or gate-level IPs such as ALUs, adder, multipliers, etc.
DATA_MEMORY	Assigned to data signals of a memory IP at the RTL or at the gate-level
DATA_PERIPHERAL	Assigned to data signals being used by peripheral units at the RTL or at the gate-level
DATA_COMMUNICATION	Assigned to data signals being used for communication purposes by communication units at the RTL or at the gate-level
DATA_ENCRYPTION	Assigned to data signals being used being encrypted by encryption units at the RTL or at the gate-level
TDI	Assigned to a test data-in signal in JTAG or a scan-in signal at the RTL or at the gate-level
TDO	Assigned to a test data-out signal in JTAG or a scan-out signal at the RTL or at the gate-level
DATA_SENSITIVE	A general form of the data assets and should only assigned to a signal of an IP at the RTL or at the gate-level when a more specific asset is not applicable

Table 2 Timing external asset category

Timing External Asset	Description
DONE	Assigned to a signal of an IP at the RTL or at the gate-level indicating that an operation is finished
HOLD	Assigned to a signal of an IP at the RTL or at the gate-level indicating to hold an operation
WAIT	Assigned to a signal of an IP at the RTL or at the gate-level indicating that an operation must wait
READY	Assigned to a signal of an IP at the RTL or at the gate-level indicating that an operation is ready
BUSY	Assigned to a signal a signal of an IP at the RTL or at the gate-level indicating that an operation is busy
STATUS	Assigned to a signal a signal of an IP at the RTL or at the gate-level indicating the status of the system
COUNT	Assigned to a signal of an IP at the RTL or at the gate-level used in a counter
TIMER_CONTROL	Assigned to a signal of an IP at the RTL or at the gate-level controlling a timer
CLOCK_CONTROL	Assigned to a signal of an IP at the RTL or at the gate-level controlling the primary or subsystem clock
TCK	Assigned to a test clock signal of an IP at the RTL or at the gate-level in JTAG system
SYSTEM_TIMING	Assigned to the primary clock signal of an IP at the RTL or at the gate-level
SUBSYSTEM_TIMING	Assigned to a subsystem clock signal of an IP at the RTL or at the gate-level

Table 3 System control external asset category

System Control External Asset	Description
SET	Assigned to a signal of an IP at the RTL or at the gate-level used to set a value
RESET	Assigned to a signal of an IP at the RTL or at the gate-level used to reset a value
READ	Assigned to a signal of an IP at the RTL or at the gate-level used to perform a read operation
WRITE	Assigned to a signal of an IP at the RTL or at the gate-level used to perform a write operation
SELECT	Assigned to a signal of an IP at the RTL or at the gate-level used to perform a select operation
EXECUTE	Assigned to a signal of an IP at the RTL or at the gate-level indicating that an operation is executed
LOAD	Assigned to a signal of an IP at the RTL or at the gate-level indicating that a value is to be loaded
MODE	Assigned to a signal of an IP at the RTL or at the gate-level indicating the mode of an operation
ENABLE	Assigned to a signal of an IP at the RTL or at the gate-level used to perform an enable operation
HANDSHAKING	Assigned to a signal of an IP at the RTL or at the gate-level used in communication by a handshaking operation
SHIFT	Assigned to a signal of an IP at the RTL or at the gate-level indicating that a shift operation
TMS	Assigned to a test mode select signal of a JTAG system at the RTL or at the gate-level
TRST	Assigned to a test reset signal of a JTAG system at the RTL or at the gate-level
INSTRUCTION	A general form of instruction assets and should only assigned to a signal of an IP at the RTL or at the gate-level when a more specific asset is not applicable
SYSTEM_CONTROL	A general form of system control assets and should only assigned to a signal of an IP at the RTL or at the gate-level when a more specific asset is not applicable

Table 4 Specific system control external asset category

Specific System Control External Asset	Description
MEMORY_OP	Assigned to a signal of an IP at the RTL or at the gate-level used to perform an operation within a memory unit
DATA_OP	Assigned to a signal of an IP at the RTL or at the gate-level used to perform an operation within a data processing unit
INTERRUPT_OP	Assigned to a signal of an IP at the RTL or at the gate-level used to perform an operation within an interrupt unit
PROGRAM_COUNTER_OP	Assigned to a signal of an IP at the RTL or at the gate-level used to perform an operation within a program counter unit
INTERRUPT_CONTROL	Assigned to a signal of an IP at the RTL or at the gate-level used as system control within an interrupt unit
PERIPHERAL_CONTROL	Assigned to a signal of an IP at the RTL or at the gate-level used as system control within a peripheral system
REGISTER_FILE_CONTROL	Assigned to a signal of an IP at the RTL or at the gate-level used as system control within a register file unit
COMMUNICATION_CONTROL	Assigned to a signal used as system control within a communication unit
COMMUNICATION_PROTOCOL	Assigned to a signal of an IP at the RTL or at the gate-level used to handle a protocol within a communication unit

Table 5 Miscellaneous external asset category

Miscellaneous External Asset	Description
CRITICAL	Assigned to a signal of an IP at the RTL or at the gate-level that could lead to harm if an attacker gained possession of it
COMPONENT	Assigned to a signal of an IP at the RTL or at the gate-level referring to another component of a system
ADDRESS_SENSITIVE	Assigned to a signal of an IP at the RTL or at the gate-level indicating the address in a memory unit
CONSTANT	Assigned to a signal of an IP at the RTL or at the gate-level indicating a value to be used as a constant
KEY	Assigned to a signal of an IP at the RTL or at the gate-level using as an encryption key in an encryption unit
REGISTER	Assigned to a signal of an IP at the RTL or at the gate-level using to handle data to in a register file unit
PROGRAM_COUNTER	Assigned to a signal of an IP at the RTL or at the gate-level indicating the value being manipulated within a program counter
ERROR_HANDLING	Assigned to a signal of an IP at the RTL or at the gate-level performing error handling
EXCEPTION_HANDLING	Assigned to a signal of an IP at the RTL or at the gate-level performing error handling
STATE	Assigned to a signal of an IP at the RTL or at the gate-level tracking the state of system or FSM

2.2.3 Internal Asset

Internal assets are assigned to mostly but not exclusively internal signals of a soft IP. Some internal assets developed in [18] are assigned to signals automatically. Other internal assets (*OBSERVABLE*, *CONTROLLABLE* and *PROTECTED*) developed in [19] are assigned to internal signals manually due to those signals' unique contribution. The current version of the Structural Checking tool utilized these internal assets to create complete asset patterns. Table 6

Table 6 Internal assets and their descriptions

Asset	Description
PROCESS_SENSITIVE	Assigned to a signal in a RTL process sensitivity list
PROCESS_OPERATION_SENSITIVE	Assigned to a signal being manipulated in a RTL process block
CONDITIONAL_DRIVING	Assigned to a signal in a RTL conditional statement or a conditional statement of a MUX or a DFF model at the gate-level
CONDITIONAL_DRIVEN	Assigned to a signal being modified in a RTL conditional block or a conditional block of a MUX or a DFF model at the gate-level
CONCURRENT_DRIVING	Assigned to a signal driving another signal in a concurrent statement of both RTL and gate-level
CONCURRENT_DRIVEN	Assigned to a signal being driven by another signal in a concurrent statement of both RTL and gate-level
CC_OPERATION_SENSITIE	Assigned to a signal being driven by two or more signals and logic operations of both RTL and gate-level
OBSERVABLE	Assigned to an observable signal under scan/test mode at the gate-level
CONTROLLABLE	Assigned to a controllable signal under scan/test mode at the gate-level
PROTECTED	Assigned to a signal that is protected from data leakage at the gate-level

shows a list of internal assets and their descriptions.

2.2.4 Asset Filtering

The idea of asset filtering is comparable to the taint analysis method introduced in [20]. 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 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. This filtering mechanism was firstly introduced in [15]. The filtering rule for the internal assets is slightly different form that of the external asset. 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 both RTL and gate-level.

2.2.5 Asset Trace and Asset Pattern

Previously introduced in [16], the set of assets assigned to a specific signal is termed an asset trace. The complete collection of asset traces of a design is termed an asset pattern. Asset patterns are generated by asset filtering process and are important for functionality matching. After assets are assigned to the signals, they are filtered along direct connections to populate the complete set of signals with a collection of assets. The asset pattern is an essential element of the design functionality determination.

An asset pattern includes 6 characteristics. Input port signal external asset characteristic is denoted as (>). Input port signal internal asset characteristic is denoted as (>*). Output port signal external asset characteristic is denoted as (<). Output port signal internal asset

characteristic is denoted as (<*). Internal signal external asset characteristic is denoted as (/). Internal signal internal asset characteristic is denoted as (/).

2.3 Functionality

Every soft IP analyzed by the Structural Checking methodology is given a functionality that represents its role in a system. During the formation of the GRL, functionalities were manually assigned to trusted designs based on previous knowledge of that design. However, the matched functionality is automatically assigned to the unknown design by the Structural Checking tool when performing GRL matching. Table 7 lists several possible functionalities that a design could be assigned to, which are categorized into whitelist and blacklist. The whitelist category contains designs that are known to be Trojan-free, while the blacklist category contains Trojan-infested designs. Matching a soft IP to a blacklist functionality is for Trojan detection and will be addressed in detail in a later section.

Table 7 Whitelist and blacklist functionality

Whitelist Functionality	Blacklist Functionality
SHIFT_REGISTER	*TROJAN_ENCRYPTION_UNIT*
INTERRUPT_UNIT	*TROJAN_TRIGGER*
COMMUNICATION	*TROJAN_COMMUNICATION*
ENCRYPTION_UNIT	*TROJAN_SHIFT_REGISTER*
COMPUTATIONAL	
TIMING	
CONTROL_GENERATION	
REGISTER_FILE	
PERIPHERAL	
DECODER_ENCODER	
DEBUG_INTERFACE	
TOP_CONTROLLER	

2.4 Golden Reference Library

An asset of a signal is an essential building block for an asset pattern and the Golden Reference Library. The Golden Reference Library (GRL) is another foundational element of the Structural Checking methodology, which is originated from [16]. The initial entries of GRL are various small designs collected from OpenCores [21] and Trust-Hub [22]. Since they are small, exhaustive verification is feasible. More entries are added from in-house designs. SC is then applied to generate asset patterns for all entries and functionalities are assigned manually.

The GRL contains many trusted GRL files/entries. Each GRL file/entry has a functionality, which represents the purpose of the associated soft IP, and an asset pattern. GRL files representing the whole collection of functionalities as defined in previous section. Figure 3 is an

```
Entity    simple_alu:
        28 port signals
        24 IntraSignals
        4 Port Signal Vectors
        3 Intra-Signal Vectors
        0 SubInstances
        1 Processes
Functionality: COMPUTATIONAL
Secondary Func: NON_SEQUENTIAL
Number of Input bits:      20
Number of Output bits:    8
>[SYSTEM_TIMING]
>*[PROCESS_SENSITIVE, CONDITIONAL_DRIVING]
>[DATA_COMPUTATIONAL]
>[DATA_OP]
>*[CONDITIONAL_DRIVING]
<[DATA_COMPUTATIONAL]
<*[CONCURRENT_DRIVEN]
/[DATA_COMPUTATIONAL]
/*[CONCURRENT_DRIVEN]
/*[CONDITIONAL_DRIVEN, PROCESS_OPERATION_SENSITIVE]
```

Figure 3 A GRL file of a RTL design

example of a GRL file where *simple_alu* is a RTL design. Figure 4 is another example where *modmult_MPWID16_1_DW01_sub_2* is a gate-level netlist. After the previous work discussed in [16] and [17], the GRL of Structural Checking has been substantially updated to a total of 152 files/entries, which are the asset patterns of distinctive designs with and without Trojan inserted. Note that these entries already consider the situation where a port signal may be assigned different assets.

```
Entity    modmult_MPWID16_1_DW01_sub_2:
                56 port signals
                35 IntraSignals
                3 Port Signal Vectors
                0 Intra-Signal Vectors
                37 SubInstances
                0 Processes
Functionality: COMPUTATIONAL
Secondary Func: NON_SEQUENTIAL
Number of Input bits:      37
Number of Output bits:    19
<[DATA_COMPUTATIONAL]
>[DATA_MEMORY, DATA_COMPUTATIONAL]
>[DATA_COMPUTATIONAL]
<[CONSTANT]
>[CONSTANT]
/[DATA_COMPUTATIONAL]
/[DATA  COMPUTATIONAL, DATA  MEMORY]
```

Figure 4 A GRL file/entry of a gate-level netlist

3 METHODOLOGY AND IMPLEMENTATION

3.1 Asset Pattern Matching

3.1.1 Basic asset trace matching

Table 8 Example of basic asset trace matching

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

After asset filtering, the target IP (X) has six asset pattern characteristics as outlined in Section 2.2.5, 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 8to clarify the asset pattern matching methodology. In Table 8, 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. However, in a case that a soft IP does not have any internal signal, the internal signal characteristics are empty. In the case that the same characteristic of both the GRL entry and the unknown IP is empty, the match percentage of the empty characteristics will be left out of the final matching result.

3.1.2 Partial asset trace matching

The partial asset trace matching algorithm was developed to gain more precisions in matching assets between two traces of the same characteristic. This is due to assets in the two asset traces often originate from the same nature. 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. If the basic matching algorithm was applied in this case, the result would yield 0%, which would not present the similar nature of the two assets.

For further explanation, Table 9 below illustrates different scenarios where partial matching is applied. First, as previously stated, case number 1 has the match of 50% because the two assets are in the same asset category. Secondly, in case number 2, 100% match is from the *DATA_MEMORY* asset of both traces while the *DATA_COMPUTATIONAL* only appears in one trace, which yields the match result of that asset 0%. Therefore, the average of 100% and 0% is 50%. In case number 3, even though both *DATA_MEMORY* and *DATA_ENCRYPTION* are in the same data category, they yield the result of 0% because those two assets represent two specific data assets. Case number 4 is the combination of case number 1 and 2 where *SET* asset gives 100% match, as well as, *RESET* and *SYSTEM_CONTROL* give 50% match. Thus, the result of

Table 9 Examples of partial asset trace matching

Case	Unknown Design Asset Traces	GRL Entry Asset Traces	Match
1	*SYSTEM_CONTROL*	*RESET*	50%
2	*DATA_ MEMORY*, *DATA_COMPUTATIONAL*	*DATA_ MEMORY*	50%
3	*DATA_ MEMORY*	*DATA_ENCRYPTION*	0%
4	*RESET, SET*	*SET, SYSTEM_CONTROL*	75%
5	*RESET*	*SET, SYSTEM_CONTROL*	25%

case number 4 is 75%. Finally, in case number 5, since the *RESET* asset is a 50% partial match to the *SYSTEM_CONTROL* asset, and no asset reflects 0% to *SET* asset. When the order of matching reverses, the *RESET* asset is matched with the *SET* asset. This causes 0% matched. No asset reflects the *SYSTEM_CONTROL* asset which leads to 0% match. Thus, the average is 0%. Since algorithm prioritizes the highest percentage match, case number 5 has 25% match.

3.1.3 Complete asset pattern matching

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. The final match value 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

Table 10 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×

internal characteristics. Hence, they have a higher weight. Table 10 shows the weighting applied to the associated characteristic. If new characteristics are added in the future, the weight ratio will be adjusted. 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.

3.1.4 Functionality matching

To aid in the asset pattern matching algorithm, a functionality determination algorithm was 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 the ones that do not. This weight number is calculated based on observation experiments throughout testing various IPs collect from open sources [21] and [22]. The weight ratio can be adjusted if the future experiments suggest differently. For example, if the *DATA_ENCRYPTION* asset is found in an input port signal external asset pattern of the unknown IP, the percentage of the asset pattern that contains *DATA_ENCRYPTION* in all GRL entries of encryption unit functionality is multiplied by 1.5. Other asset pattern characteristics receive the weight of 1 if they do not have any functionality-specific external assets.

3.2 Enhanced Golden Reference Matching for both RTL and Gate-level IPs

The concept of matching an unknown asset pattern to known asset patterns in GRL is termed GR matching. The enhanced GR matching not only addresses port mapping issues, but

also ensures matched results coherent. The enhanced technique is categorized into two states which are the matching state and the evaluating state. The matching state is implemented as matching a top-level pattern and sub-level patterns of an unknown design to known patterns in a GRL. While the evaluating state verifies the top-level and the sub-level matched results.

The matching state analyzes either a RTL or a gate-level netlist which has one top-level entity and many sub-level entities. For matching state to operate, GR files/entries are divided into the top-level GRL and the sub-level GRL. Then, SC compares the top-level asset pattern of the unknown IP (denoted as X) to each entry of the top-level GRL. SC applies all the rules outlined in Section 3.1 to find the highest percentage matching (HPM) result. The functionality of the top-

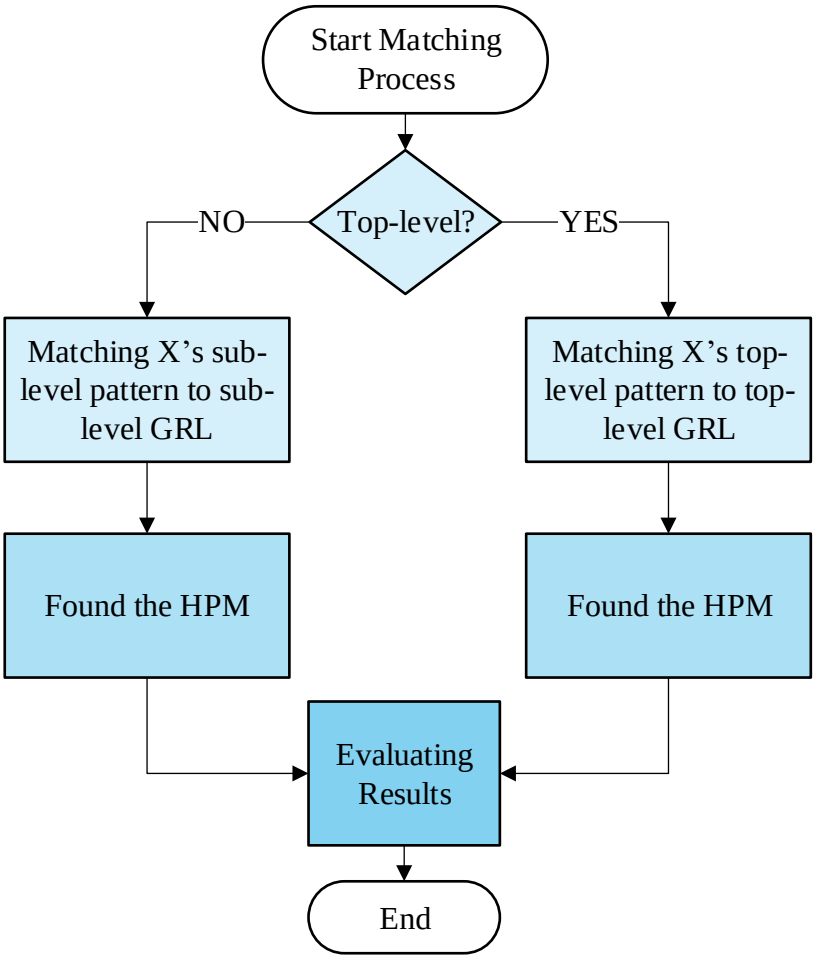

Figure 5 A simplified flow chart for the matching state

level GR entry which has the HPM is assigned to the top-level asset pattern of X. Simultaneously, SC compares each of X's sub-level patterns to all entries of the sub-level GRL. The same rules in Section 3.1 are applied. All the HPM results and assigned functionalities of the sub-level patterns are recorded for the next evaluating state. Figure 5 is a simplified flow chart for the matching state.

The current version of the evaluating state only applies to gate-level netlist analysis. SC gives level 0 to the top-level entity of X, and the maximum level to the lowest level entity of X. Since X is a gate-level netlist, X's standard logic gates usually have the maximum level. The algorithm of the evaluating state is illustrated in

Figure 6. The evaluating state operates based on the total hierarchy depth of X. If the hierarchy is less than 2, X is a flattened netlist. SC presents users the HPM results, other possible matched results and matched functionalities from both the top-level GRL and the sub-level GRL. If the hierarchy ranges from 2 to 4, X is a simple circuitry. Therefore, SC verifies the matched functionalities of X's top-level and sub-level are the same. However, there are exceptions for functionalities in encryption category (ENCRYPTION_UNIT, TROJAN_ENCRYPTION_UNIT), communication category (COMMUNICATION, TROJAN_COMMUNICATION) and generic category (COMPUTATIONAL, TROJAN_TRIGGER). When matching top-level and sub-level patterns of a gate-level netlist, SC does not match standard logic gate patterns because these patterns do not provide valuable information to gate-level netlists. When matched functionalities fall in a combination of encryption and generic category or a combination of communication and generic category, those functionalities are considered valid. Otherwise, SC raise a flag that the matched functionalities are invalid. Finally, the hierarchy is equal or greater than 5, X is a complex circuitry. SC notifies users to analyze the sub-level entity of X prior to proceed the full analysis from the top-level

entity. Only if X's top-level functionality is TOP_CONTROLLER; and X's sub-level functionalities consist of 2 or more different functionalities. SC deems the matches results as valid. Examples in Section 4.2 further explain the methodology.

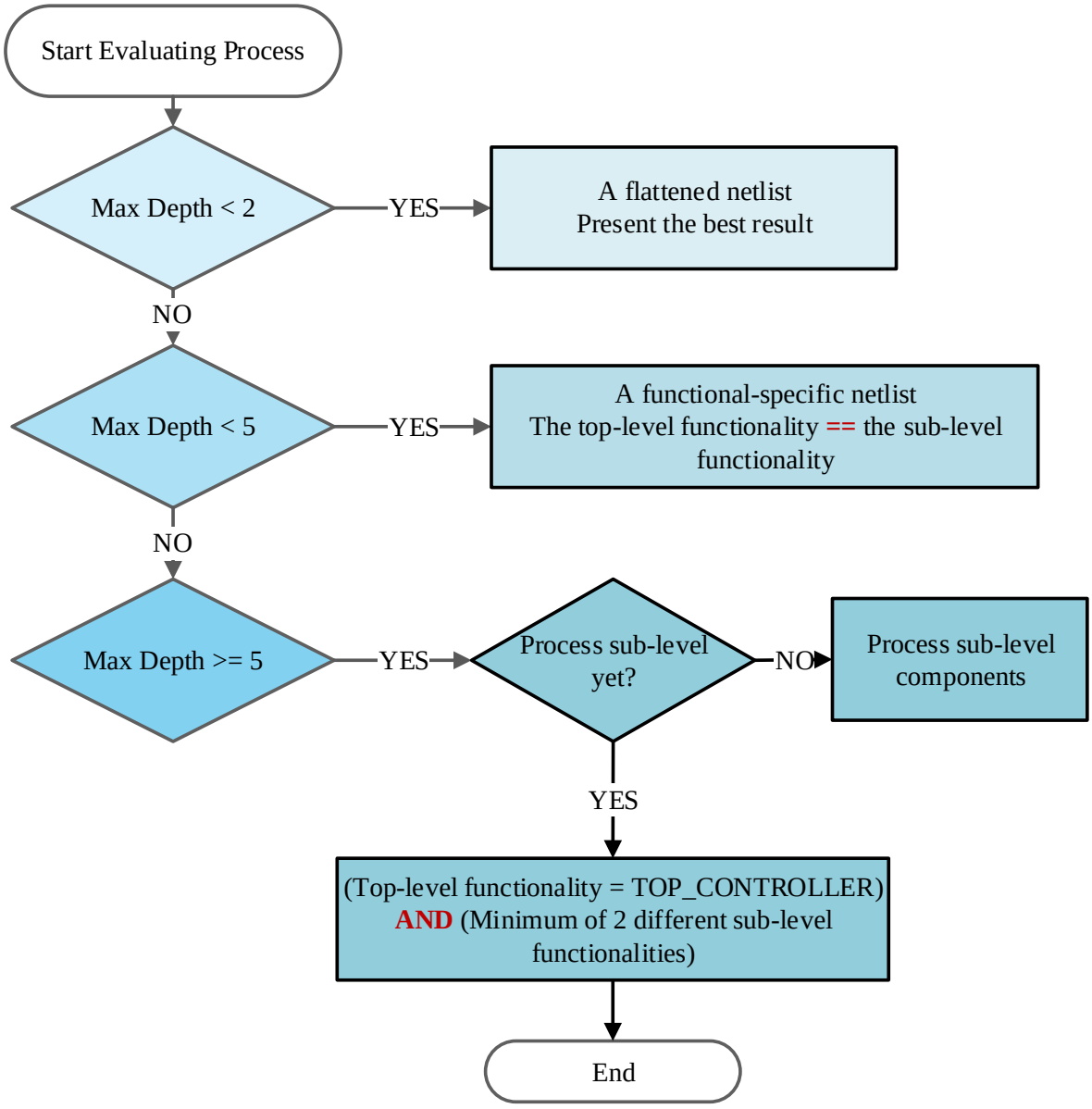

Figure 6 A simplified diagram of the evaluating state

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity AND2 is
  port( Z       : out STD_ULOGIC;
        A       : in STD_ULOGIC;
        B       : in STD_ULOGIC);
end AND2;
architecture ARCH_FUNC of AND2 is
begin
    Z <= ( B )AND( A );
end ARCH_FUNC;
```

Figure 7 An AND2 standard logic gate model for SC

3.3 Standard Logic Gate Model and Netlist Pre-processing for Structural Checking

The standard logic gate library is important while using SC to analyze a gate-level netlist. This naming convention of this dissertation work is based on the IBM 130nm 8RF standard cell library because this library is used in the synthesis process of all testing IPs. Most standard logic gates are modeled in concurrent statements. Multiplexer (MUX) cells and flip flops (DFFs) are modeled with process statements and conditional statements. Each standard logic gate has capitalized name such as AND, OR, XOR, etc., followed with the number of inputs. The inputs are named alphabetically and capitalized (e.g., A, B and C) and the output is named Z. Special cells such as DFF, DFFSR, SDFFAR, LATSR, and MUX have different primary port names. For flip flops and latches, D is data input; RN or RSTB is reset-bar input; S is set input; and Q and QBAR are data output and inverse data output. Figure 7 is an example of standard logic gate AND2. Figure 8 is an example of a DFFSR.

A soft IP gate-level netlist often includes many entities. All entities are grouped in a single VHDL file due to the synthesis tool default format while SC parses an entity in a VHDL file individually. Hence, a Python script named GateSizeRemoval_EntityPartition is developed to partition a synthesized netlist into individual VHDL files. The name of each VHDL file is based

on the entity's name in the file content. In addition, the script removes all sizes of standard logic gates and changes BUFFER to BUFFER1. Changing buffer gate name is necessary because SC's parser considers BUFFER as a compiling error.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity DFFSR is
  port(
    Q          : out  STD_ULOGIC;
    QBAR       : out  STD_ULOGIC;
    CLK        : in   STD_ULOGIC;
    D          : in   STD_ULOGIC;
    RN         : in   STD_ULOGIC;
    S          : in   STD_ULOGIC);
end DFFSR;

ARCHITECTURE BEHAV OF DFFSR IS
 BEGIN
   FF: PROCESS( RN , S , CLK )
   BEGIN
                if RN = '0' then
                        Q <= '0';
                        QBAR <= '1';
                elsif S = '1' then
                        Q <= '1';
                        QBAR <= '0';
                elsif CLK'event and CLK = '1' then
                        Q <= D;
                        QBAR <= NOT D;
                end if;
   END PROCESS;
```

Figure 8 A DFFSR standard cell for SC

4 RESULT AND ANALYSIS

4.1 Trojan Detection Result for Gate-level versus RTL

The GR matching in [16] assigns TOP_LEVEL functionality to the top-level entity of IPs when encountering port-map. It does not perform GR matching at the top-level entity. Examples such as RS232-T400 and BasicRSA-T100 show that it is possible to insert Trojan payload at the top-level of an IP. Hence, enhanced GR matching solves issues for both RTL and gate-level analysis.

There are 38 gate-level netlists and 22 RTL IPs used to test the enhanced GR matching methodology. All IPs used for testing are obtained from both Trust-Hub [22] and OpenCores [21]. Testing IPs used for both RTL and gate-level methodology are mostly encryption cores, UART cores and microcontrollers with and without hardware Trojan inserted. Hardware Trojan payloads from testing IPs are denial-of-service and key/data leakage. Table 12 shows a list of IPs which have their asset patterns and functionalities added to the GRL. Table 13 shows a list of IPs used to challenge the GR matching for RTL. As shown in Table 13, the first 12 IPs are correctly identified as Trojan infested and the last 4 IPs are Trojan free. Even though SC detects Trojan from RS232-T100 correctly, it identifies another entity within RS232-T100 incorrectly as Trojan infested. Therefore, the false positive of the GR matching for RTL is 6.25%.

Comparing to the GR matching for RTL IPs, the GR matching for gate-level IPs improves with a false positive rate of 3.57%. The number of IPs used to challenge the new gate-level GR matching is 27 as shown in Table 14. The first 22 IPs are Trojan infested while the last 5 IPs are Trojan free. Table 11 includes a list of IPs whose asset patterns and functionalities are added to the top-level GRL and the sub-level GRL correspondently. Due to gate-level netlists have simple syntaxes, SC is able to analyze more synthesized IPs from both Trust-Hub [22] and OpenCores

[21]. SC identifies all Trojan infested and Trojan free IPs in Table 14 correctly using the enhanced GR matching methodology outlined in Section 3.2. However, SC incorrectly identified *x_mit* entity of RS232-T800 contained Trojan while it does not. Hence, it yields 3.57% false positive. SC operates on a desktop 3.6GHz CPU with 16GB RAM. Overall, the enhanced GR matching methodology provides a centralized whitelist and blacklist database within the GRL. The end-users can run a quick analysis (15 – 30 minutes) to eliminate the possible presents of these known Trojans.

Table 12 RTL IPs in to the GRL

IPs Name	**Trojan Infest**
BasicRSA-T300	Yes
RS232-T200	Yes
RS232-T500	Yes
RS232-T600	Yes
RS232-T800	Yes
AES-T2000	Yes

Table 11 Gate-level IPs in the GRL

IPs Name	**Trojan Infested**
BasicRSA-T300	Yes
RS232-T100	Yes
RS232-T200	Yes
RS232-T400	Yes
RS232-T600	Yes
AES-T100	Yes
AES-T300	Yes
AES-T1800	Yes
AES-T2000	Yes
debug_interface	No
MSP430	No

Table 13 A list of IPs used to verify the RTL methodology

IPs Name	Trojan	
	Infested	Found
AES-T600	1	1
AES-T1800	1	1
BasicRSA-T100	1	1
BasicRSA-T200	1	1
BasicRSA-T400	1	1
RS232-T100	1	1
RS232-T300	1	1
RS232-T600	1	1
RS232-T700	1	1
RS232-T900	1	1
RS232-T901	1	1
Microcontroller-c16	1	1
RSA - Trojan Free	0	0
AES - Trojan Free	0	0
RS232 - Trojan Free	0	0
RegisterFile - Trojan Free	0	0

Table 14 A list of IPs used to verify the gate-level methodology

IPs Name	Number of Trojan	
	Infested	Found
BasicRSA-T100	1	1
BasicRSA-T200	1	1
BasicRSA-T400	1	1
RS232-T300	1	1
RS232-T500	1	1
RS232-T700	1	1
RS232-T800	1	1
RS232-T900	1	1
RS232-T901	1	1
AES-T200	1	1
AES-T500	1	1
AES-T600	1	1
AES-T700	1	1
AES-T800	1	1
AES-T900	1	1
AES-T1000	1	1
AES-T1100	1	1
AES-T1200	1	1
AES-T1300	1	1
AES-T1400	1	1
AES-T1500	1	1
AES-T1900	1	1
OpenJTAG	0	0
Microcontroller-c16 – Trojan free	0	0
RSA - Trojan free	0	0
AES - Trojan free	0	0
RS232 - Trojan free	0	0

4.2 Examples

4.2.1 Crypto core AES-T1900

The benchmark AES-T1900 obtained from Trust-Hub [22] is used to demonstrate the detection of Trojans using SC. AES-T1900 originally is a RTL 128-bit encryption core. It is infested with a cipher key leakage Trojan. AES-T1900 is synthesized with a reserved hierarchy. The Python script GateSizeRemoval_EntityPartition (Section 3.3) is executed to remove gate sizes and partition AES-T1900 to individual entity files. After the parsing process, assets are assigned to AES-T1900 primary port signals. Table 15 is the asset assignment for AES-T1900. Following the assignment process, the filtering process generates one top-level and sub-level asset patterns for AES-T1900. Those patterns are then used to match to other patterns in the top-level and the sub-level GRL accordingly. In this case, the functionally of top-level AES-T1900 is correctly identified as ENCRYPTION_UNIT. Sub-level entities of AES-T1900 (*aes_128, expand_key_128*, etc.) are correctly identified as ENCRYPTION_UNIT. *TSC* and *TSC_DW01_add_0* are correctly identified as TROJAN_ENCRYPTION_UNIT and TROJAN_TRIGGER, respectively. Figure 9 is a screenshot of the result from SC's log screen. The total analysis time takes 15 minutes and 27 second on average on a 3.6GHz processor PC with 16GB of RAM.

Table 15 Asset assignment for AES-T1900

Signals	Assets
clk	*SYSTEM_TIMING*
key	*KEY*
out_port	*DATA_ENCRYPTION*
rst	*RESET*
state	*STATE*

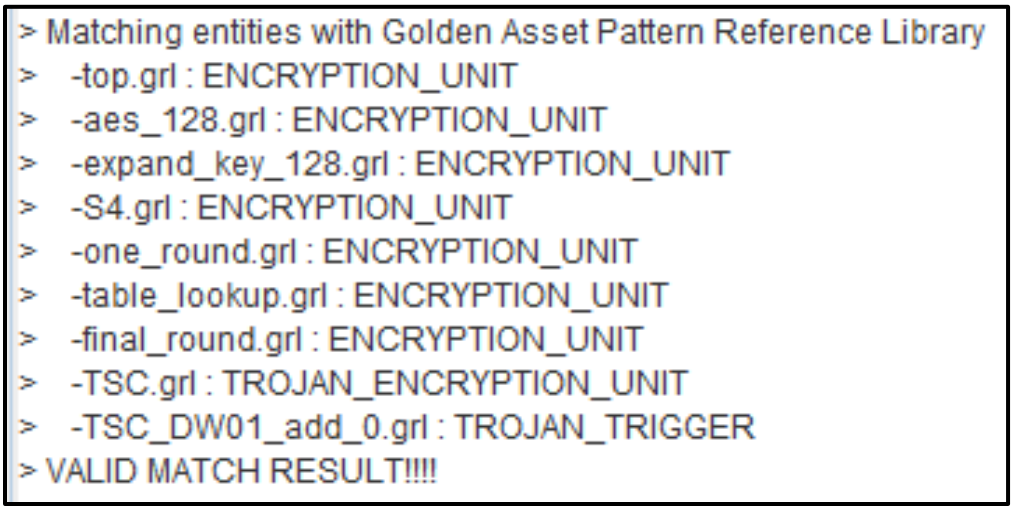

Figure 9 AES-T1900 matched result

4.2.2 Communication UART

The non-flattened *uart_baugen* is a sub-level entity of Trojan-free microcontroller c16 obtained from OpenCores [21]. *uart_baugen* also has sub-level entities such as *uart, UART_TX,* and *UART_RX*. The maximum depth hierarchy of *uart_baugen* is 3. Based on the methodology in Section 3.2, the valid matched results should be COMMUNICATION. Table 16 is the asset assignment for *uart_baugen*. After SC's parsing, asset assigning, asset filtering and GR matching, *uart_baugen* has the HPM as the top-level *uart* of RS323-T100. *uart* of RS323-T100 has a COMMUNICATION functionality, thus *uart_baugen* is assigned the COMMUNICATION functionality. Sub-level entities of *uart_baugen* (*uart, UART_TX* and *UART_RX*) have the HPM as the sub-level *u_rec* of RS232-T600. Since *u_rec* of RS232-T600 has COMMUNICATION functionality, *uart, UART_TX* and *UART_RX* have COMMUNICATION functionality. Note that *u_rec* of RS232-T100 and *u_xmit* of RS232-T600 are Trojan infested, the other entities of those RS232 are Trojan free. Hence, in this example, the matching of *uart_baugen* is correct.

The same *uart_baugen* is flattened. Its maximum hierarchy is 1. The flattened *uart_baugen* is assigned the same set of assets in Table 16 after the SC's parsing process. Applying the

methodology in Section 3.2, SC matches the only top-level *uart_baugen* to both top-level GRL and sub-level GRL. The flattened *uart_baugen* has the HPM as the *uart* entity of the RS232-T100 which has COMMUNICATION functionality.

4.2.3 Microcontroller c16

Microcontroller c16, is a Trojan free RTL IP. It is synthesized and used to challenge the enhanced GR matching. The maximum depth of c16 is 5 when *cpu* is level 0. Figure 10 is a simplified diagram of c16. Since the maximum level is 5, the analysis needs to start with entities that have lower maximum depth. Therefore, *alu8, uart_baugen, memory* and *opcode_decoder* are analyzed first. However, memory and *opcode_decoder* sub-level entities are RTL designs because they are not synthesizable. SC successfully determines *memory* and *opcode_decoder* as REGISTER_FILE and DECODER_ENCODER, respectively. Then, *alu8* is identified as COMPUTATIONAL, and *uart_baugen* is identified as COMMUNICATION (Section 4.2.2). After all lower levels of *cpu* are analyzed, SC starts the analysis from *cpu* entity, the top-level. *cpu*'s asset pattern is correctly recognized as TOP_CONTROLLER. Table 17 is a list of assets assigned to the primary port signal of *cpu*. All conditions such as TOP_CONTROLLER functionality for top-level pattern and more than two different functionalities for sub-level patterns are satisfied. Hence, the matching results in this example are valid. If the sub-level entities are not analyzed prior to the top-level entity, SC incorrectly matches *alu8* and *uart_baugen* to a potential REGISTER_FILE or TIMING functionality.

Table 16 Asset assignment for the uart_baugen of microcontroller c16

Signals	Asset
CLK_I	*SYSTEM_TIMING*
RST_I	*RESET*
RD	*READ*
RX_DATA	*DATA_COMMUNICATION*
RX_READY	*COMMUNICATION_STATUS*
RX_SERIN	*DATA_COMMUNICATION*
TX_BUSY	*COMMUNICATION_STAUS*
TX_DATA	*DATA_COMMUNICATION*
TX_SEROUT	*DATA_COMMUNICATION*
WR	*WRITE*

Table 17 Asset assignment for the cpu of microcontroller c16

Signals	Assets
CLK_I	*SYSTEM_TIMING*
SER_IN	*DATA_SENSITIVE*
SER_OUT	*DATA_SENSITIVE*
SWITCH	*PERIPHERAL_CONTROL*
TEMP_CE	*READY*
TEMP_SCLK	*SUBSYSTEM_TIMING*
TEMP_SPI	*DATA_PERIPHERAL, DATA_COMMUNICATION*
TEMP_SPO	*DATA_PERIPHERAL, DATA_COMMUNICATION*
XM_ADR	*ADDRESS_SENSITIVE*
XM_CE	*READY*
XM_RDAT	*DATA_MEMORY*
XM_WDAT	*DATA_MEMORY*
XM_WE	*READ, WRITE*

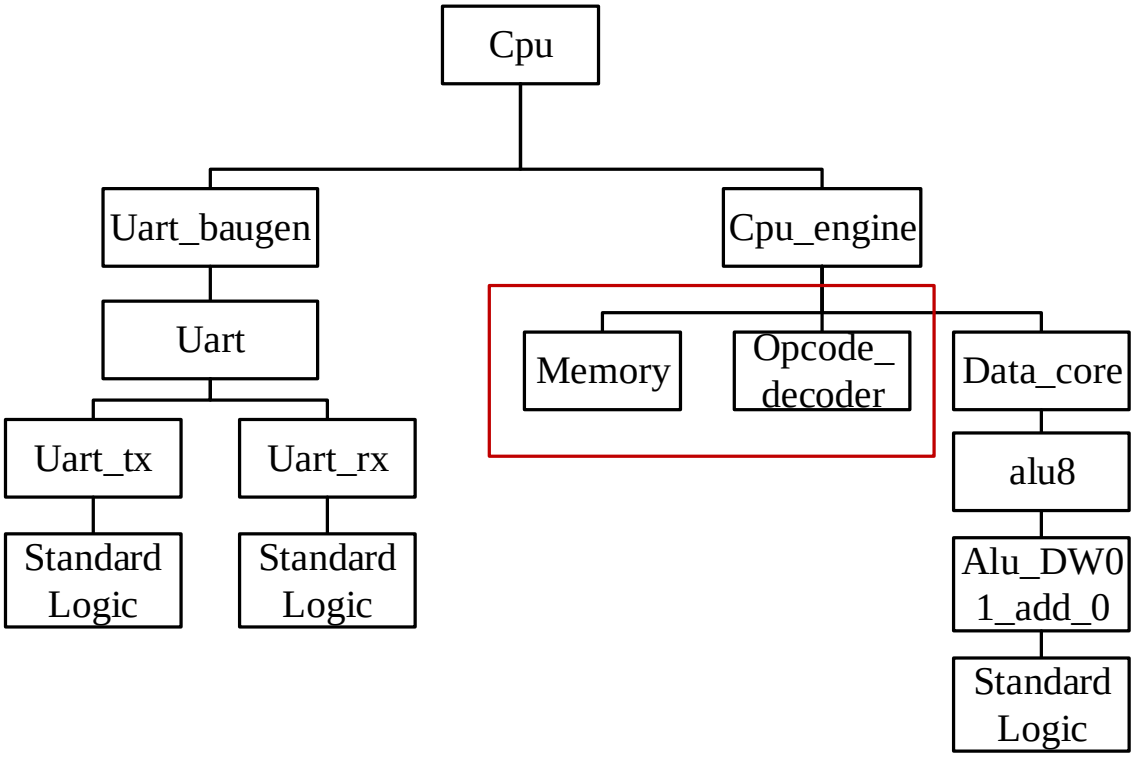

Figure 10 *c16* hierachy diagram

5 ASSET APPLICATION IN DATA FLOW TRACKING

5.1 Introduction

Data flow tracking using SC published in [19] is independent from GR matching methodology. The technique uses asset concepts and asset filtering in SC as building blocks to uncover possible critical data leakage in scan-chain gate-level netlist. Due to static analysis nature of SC, SC does not take much time to analyze a scan-chain netlist. At the same time, SC raise awareness to users about suspicious signals, confidential data flow and critical data bypass.

5.2 Data Flow Analysis

5.2.1 Malicious Signal Detection

The goal of malicious signal detection is discovering suspicious internal signals. Usually, a system is operated by critical control signals. If these critical signals are controlled/driven by other malicious internal signals, the system is compromised. The detection technique is

```
A suspicious signal with M/SE asset: n500
The instance has observable net: valid_out_reg
The observable signal leaks a KEY is : Q
The instance has observable net: valid_out_reg
The observable signal leaks a KEY is : QN
The instance has observable net: data_out_reg_3_inst
The observable signal leaks a KEY is : Q
The instance has observable net: data_out_reg_3_inst
The observable signal leaks a KEY is : QN
The instance has observable net: data_out_reg_2_inst
The observable signal leaks a KEY is : Q
The instance has observable net: data_out_reg_2_inst
The observable signal leaks a KEY is : QN
The instance has observable net: data_out_reg_1_inst
The observable signal leaks a KEY is : Q
The instance has observable net: data_out_reg_0_inst
The observable signal leaks a KEY is : QN
The instance has observable net: data_out_reg_0_inst
The observable signal leaks a KEY is : QN
```

Figure 11 A sample report of malicious signal detection and confidential data tracking

implemented in SC and is illustrated in Figure 12. After the parsing process and asset assigning process, SC performs the detection step. First, SC trace signals (Bs) of the target scan-chain netlist are assigned with TMS assets, previously preferred as SCAN_ENABLE assets. If those signals (Bs) are driven/assigned by other signals (Cs), SC continues to examine Cs if they contain any asset within system control category (Table 3). If this is not true, SC raises a flag to alert users about Cs. For example, *scan_enable* signal and internal signal *n500* signal are inputs of an AND2 gate. If *n500* is assigned by a constant '1', logically *scan_enable* signal is not affected by *n500*. However, the presence of an extra logic gate and an extra signal increases load capacitance and can potentially lead to system malfunction. On the other hand, if *n500* is driven/assigned by a *reset* signal with a RESET asset through an inverter, then *n500* is not suspicious. An output example of the malicious signal detection is highlighted in Figure 11.

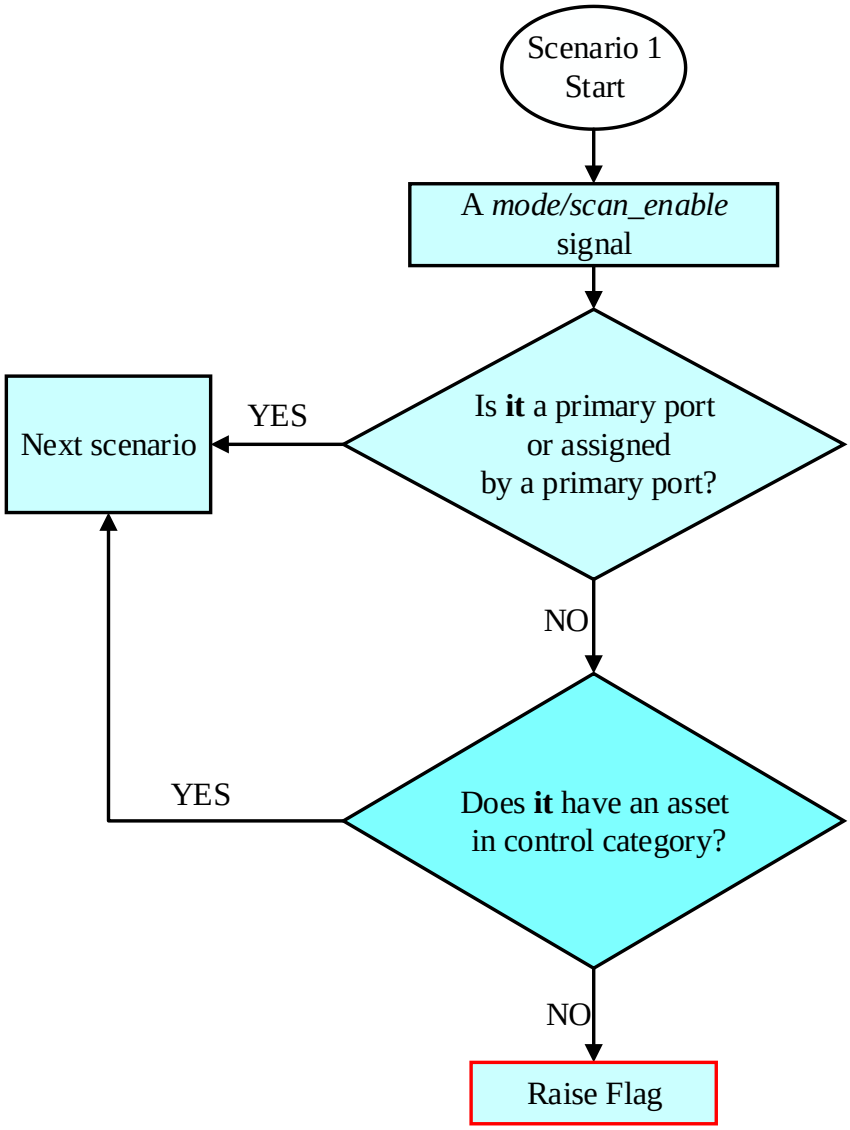

Figure 12 Malicious signal detection flow chart

5.2.2 Confidential Data Tracking

The goal of confidential data tracking technique is to warn users about the observability of the secret data in a scan-chain system. Cypher key is assumed as a type of confidential data; therefore, cypher key signals are assigned with the KEY asset. The scan-in input and the scan-out output of a scan cell are considered as a controllable net and an observable net, respectively. Thus, the scan-in input is assigned with the CONTROLLABLE asset; and the scan-out output is

assigned with the OBSERVABLE asset. During the asset filtering, SC keeps track of where KEY asset is passed to. If the KEY asset is filtered to a net that has an OBSERVABLE asset, SC raises a flag to that net. Figure 11 is an example when SC reports to users about the secret key leakage through observable nets. However, if the observable net also has PROTECTED asset, SC does not raise a flag. The confidential data tracking technique is developed based on an assumption: the end-users who develop a scan-chain system with an integrated encryption unit have knowledge of shielding observable nets from attackers. Figure 13 is a flow chart of confidential data tracking technique.

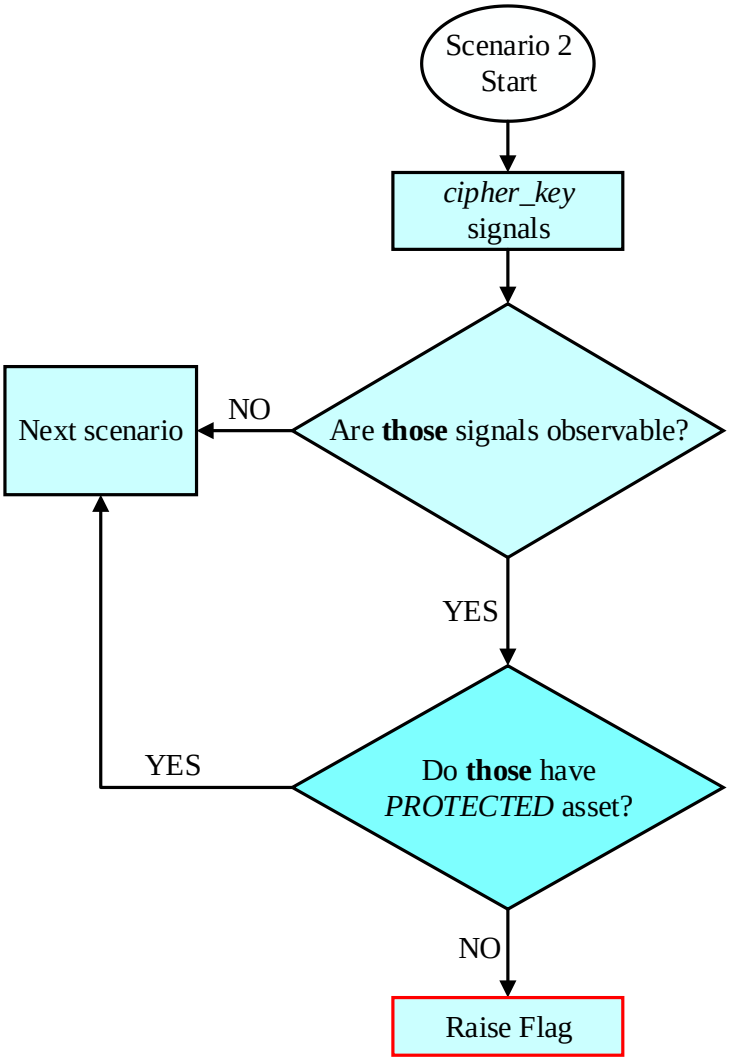

Figure 13 A flow chart of confidential data tracking

5.2.3 Critical Data Bypass Checking

Critical data bypass checking assumes that attackers circumvent critical data away from trusted IPs. In other words, if an end-user integrates his/her trusted encryption core to a 3rd IP, it is important for the critical data go through his/her encryption core. Therefore, it is necessary for the end-user to know all possible data paths. For the critical data bypass checking to operate, users must assign PROTECTED asset to all primary port signals of trusted entities. Then, SC reports all possible data paths which connect to trusted entities. In case users do not assign PROTECTED asset to any signal, SC reports all possible data paths which contain observable nets. Figure 14 is a portion of the data path report SC generates using critical data bypass checking. The report includes instances and signal along each data path.

```
A data path of instances: [Top_Level_Instance, U4, U3, data_out_reg_0_inst]
The data signals path: [data_in(0), IN2, Q, n800, IN4, n670, D, <data_out_0_port>]

A data path of instances: [Top_Level_Instance, U6, U5, data_out_reg_1_inst]
The data signals path: [data_in(1), IN2, Q, n799, IN4, n668, D, <data_out_1_port>]

A data path of instances: [Top_Level_Instance, U10, U9, data_out_reg_3_inst]
The data signals path: [data_in(3), IN2, Q, n797, IN4, n664, D, <data_out_3_port>]

A data path of instances: [Top_Level_Instance, U8, U7, data_out_reg_2_inst]
The data signals path: [data_in(2), Q, n798, IN4, n666, D, <data_out_2_port>]

A data path of instances: [Top_Level_Instance, data_out_reg_0_inst]
The data signals path: [test_si, SI, <Q>, <data_out_0_port>]
```

Figure 14 A portion of data path report

6 UPDATED STRUCTURAL CHECKING

SC is implemented in Java programming language. SC has a graphical user interface for users to navigate through the tool. The home screen is shown in Figure 15. The left side of Figure 15 shows five steps: design parsing, external and internal asset assignment, filtering – matching – functionality analysis, and Trojan detection. The right side is the system log screen to display extra information for users. The round color dot is an indication of each step. Red dot means that the previous step is not complete. Yellow dot means the step is in operation. Finally, green dot indicates the step is complete. Step one – design parsing: users navigate the tool to the needed analysis soft IP through a browse button. Parse design button allows users to start the parsing process. The yellow dot of step 1 turns green as shown in Figure 16 as soon as step 1 is finished. Step two – external asset assignments: users have the option of assigning external assets manually for the first time or assigning external asset from file for repeated analysis. If users wish to assign external assets manually, another window of SC appears as Figure 17 to let users choose signals and assign assets. During this step, users are allowed to select multiple signals to assign or remove assets using highlighted assign asset and remove asset buttons in Figure 17, respectively. Figure 18 reflects the GUI of available external assets in SC. Step three – internal asset assignments: users assigns internal assets appropriately to internal signals through similar interfaces (Figure 19 and Figure 20) as step two. Step four – filtering, matching and functionality analysis: SC first propagates assets from port signals and internal signals to generate asset pattern for the unknown IP. Then, SC compares the unknown pattern to known patterns in the GRL and determines the functionality for the unknown soft IP. The matched results are displayed in the system log screen as highlighted in Figure 21. Finally, step five – Trojan analysis: SC generates a report to alert users the type of potential hardware Trojan or Trojan triggers if the blacklist

functionality does not support. The blacklist functionality appearance at the end of steps four points out the hardware Trojan within an entity boundary. Then, the report in step five points out potential Trojan signals and Trojan trigger signals.

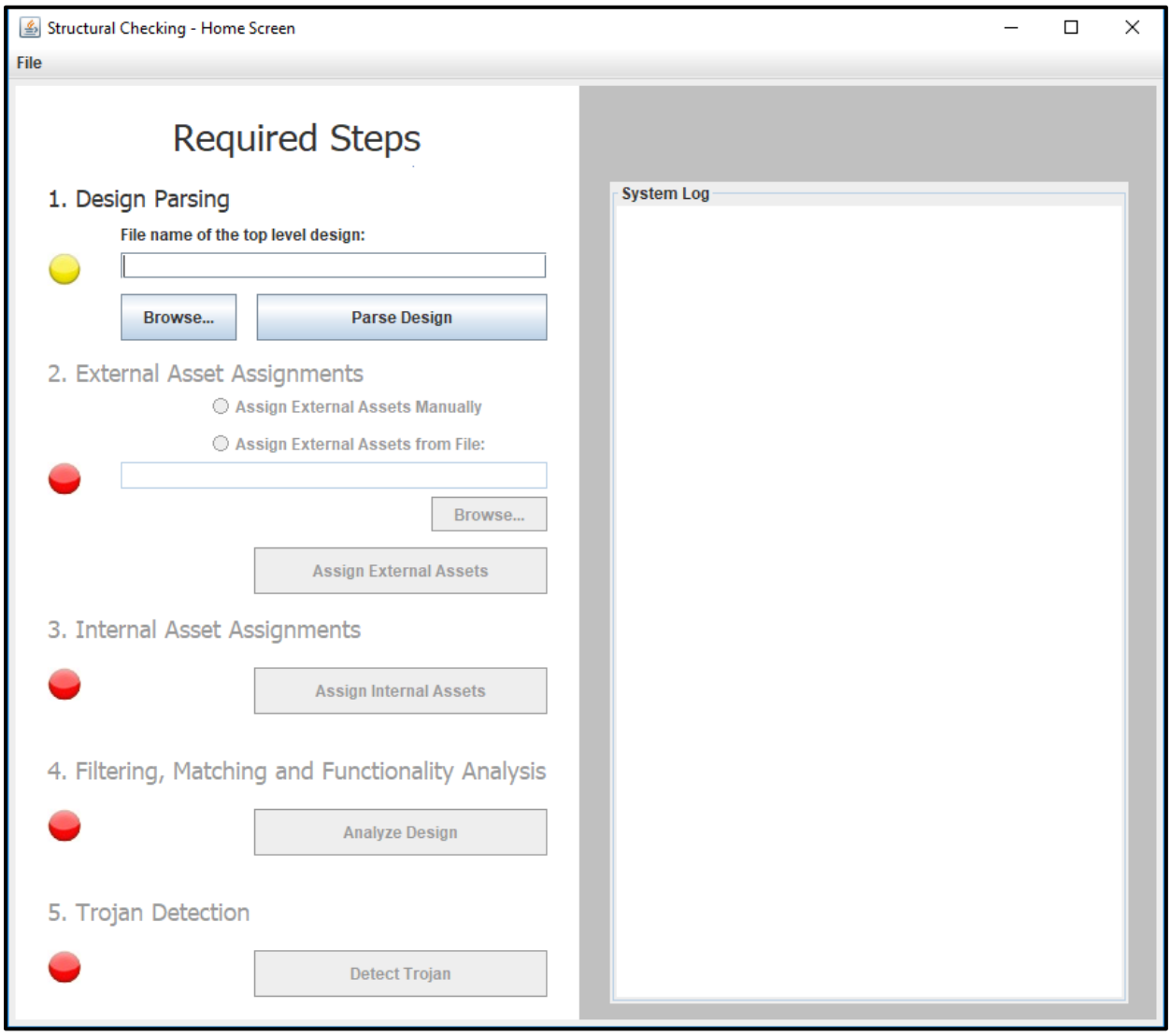

Figure 15 Structural Checking main GUI

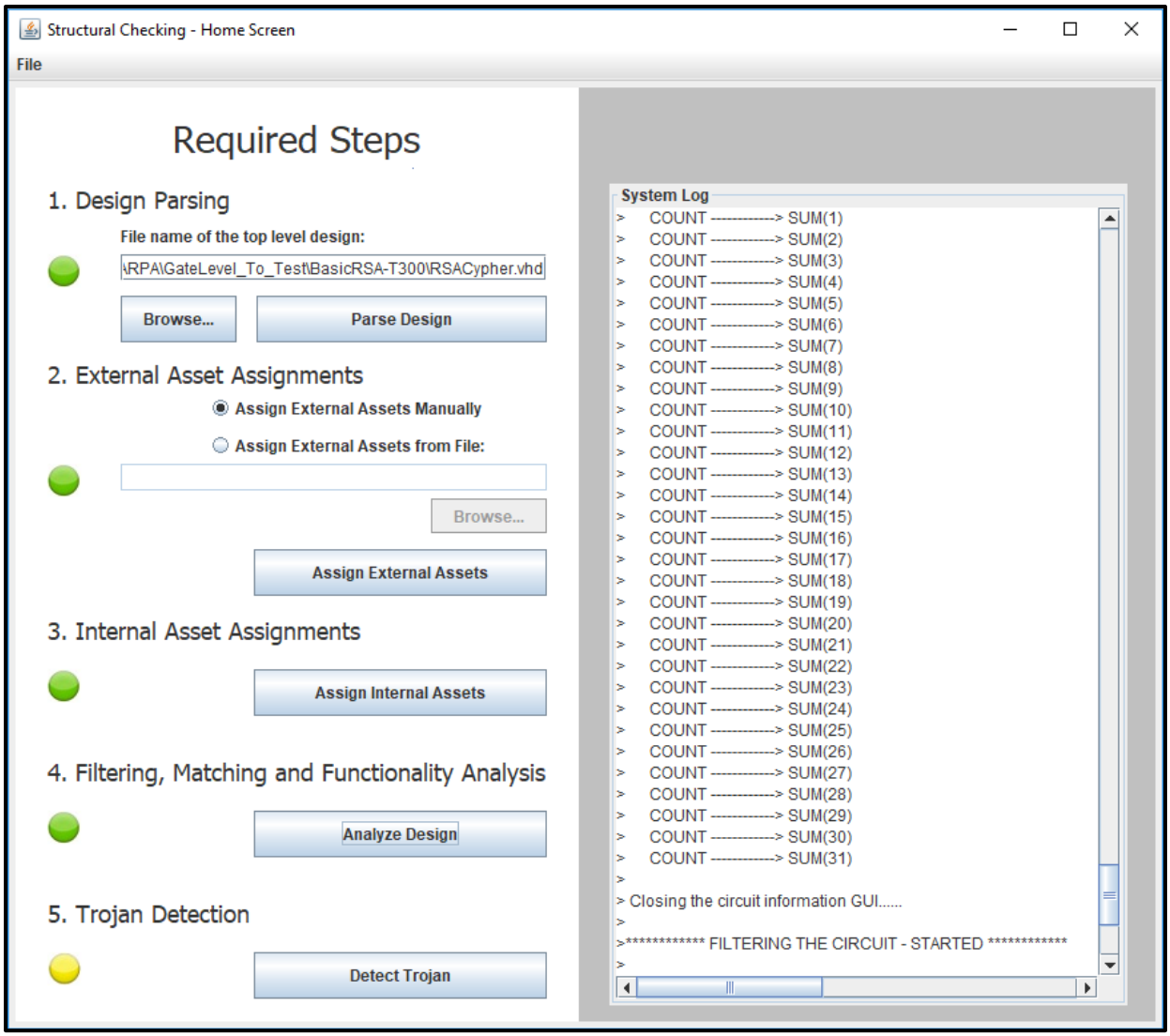

Figure 16 SC main GUI complete status

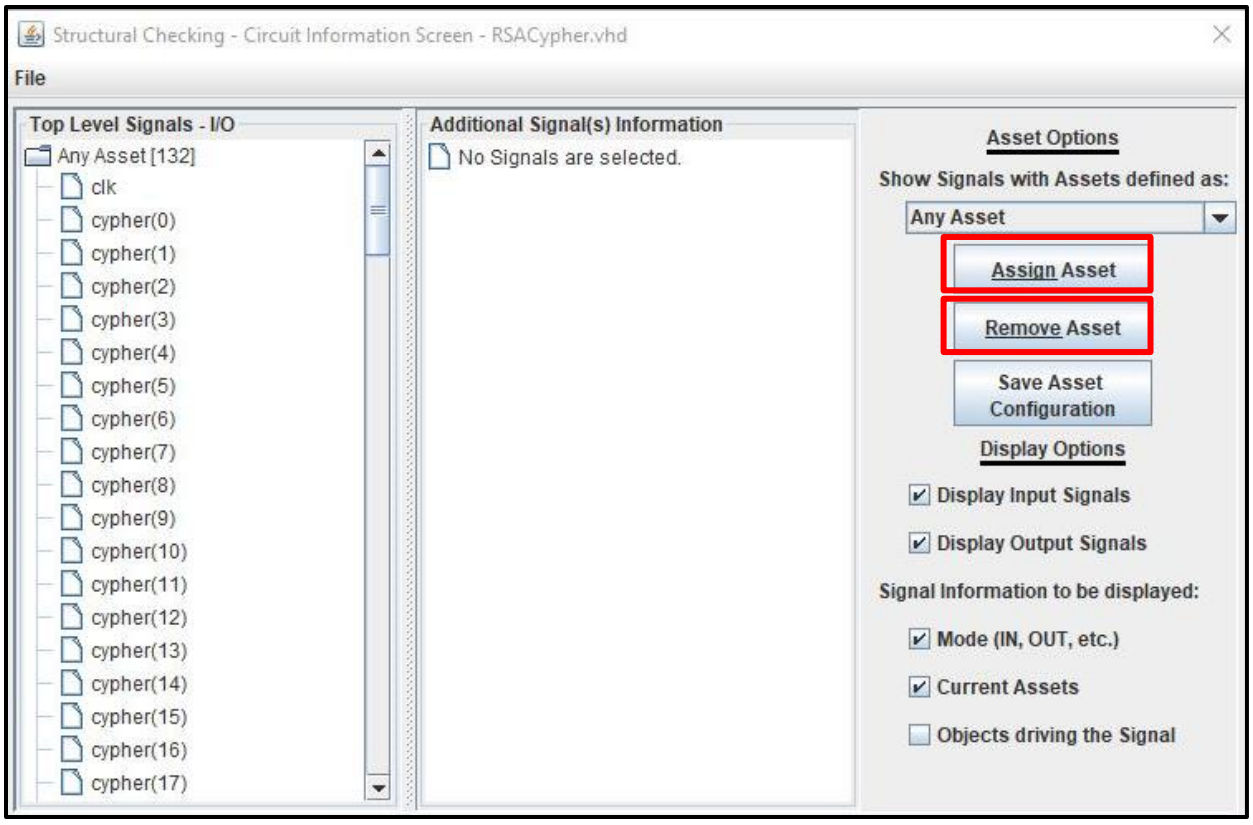

Figure 17 Circuit information GUI

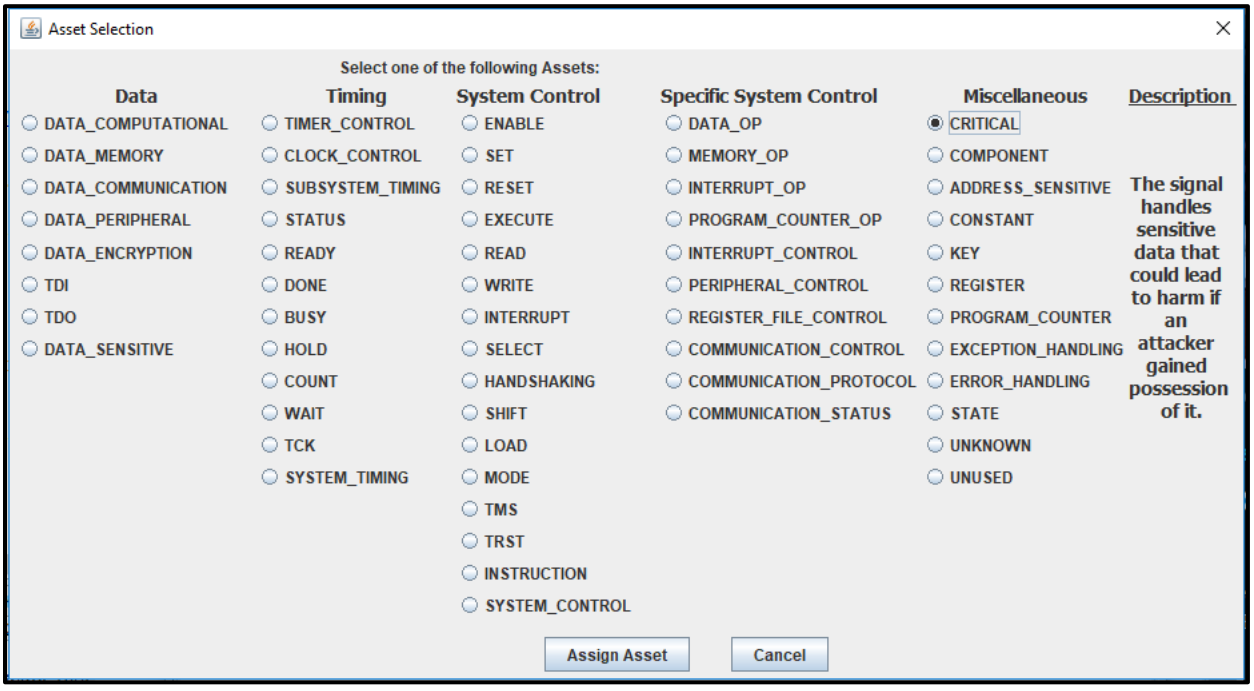

Figure 18 External asset GUI

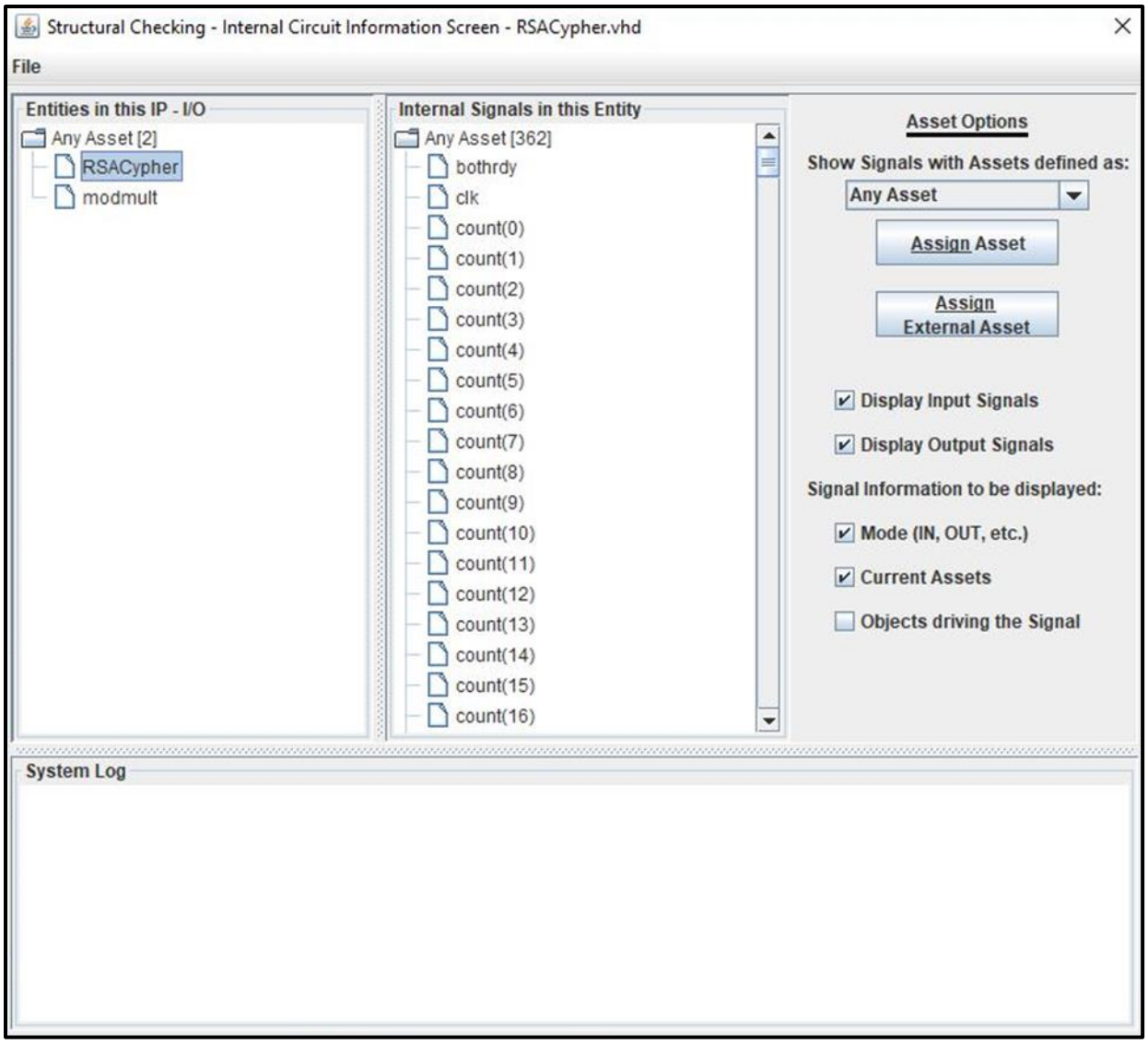

Figure 19 Internal circuit information GUI

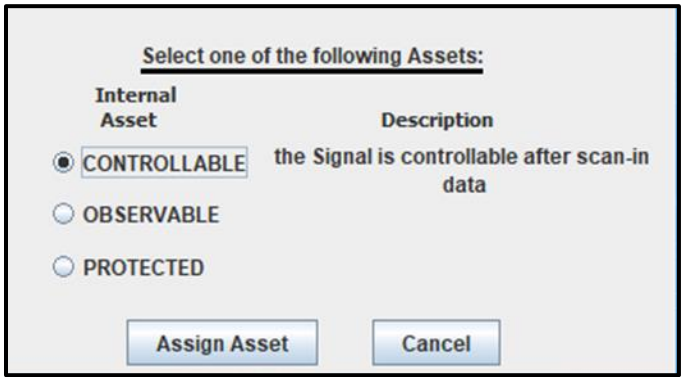

Figure 20 Internal asset GUI

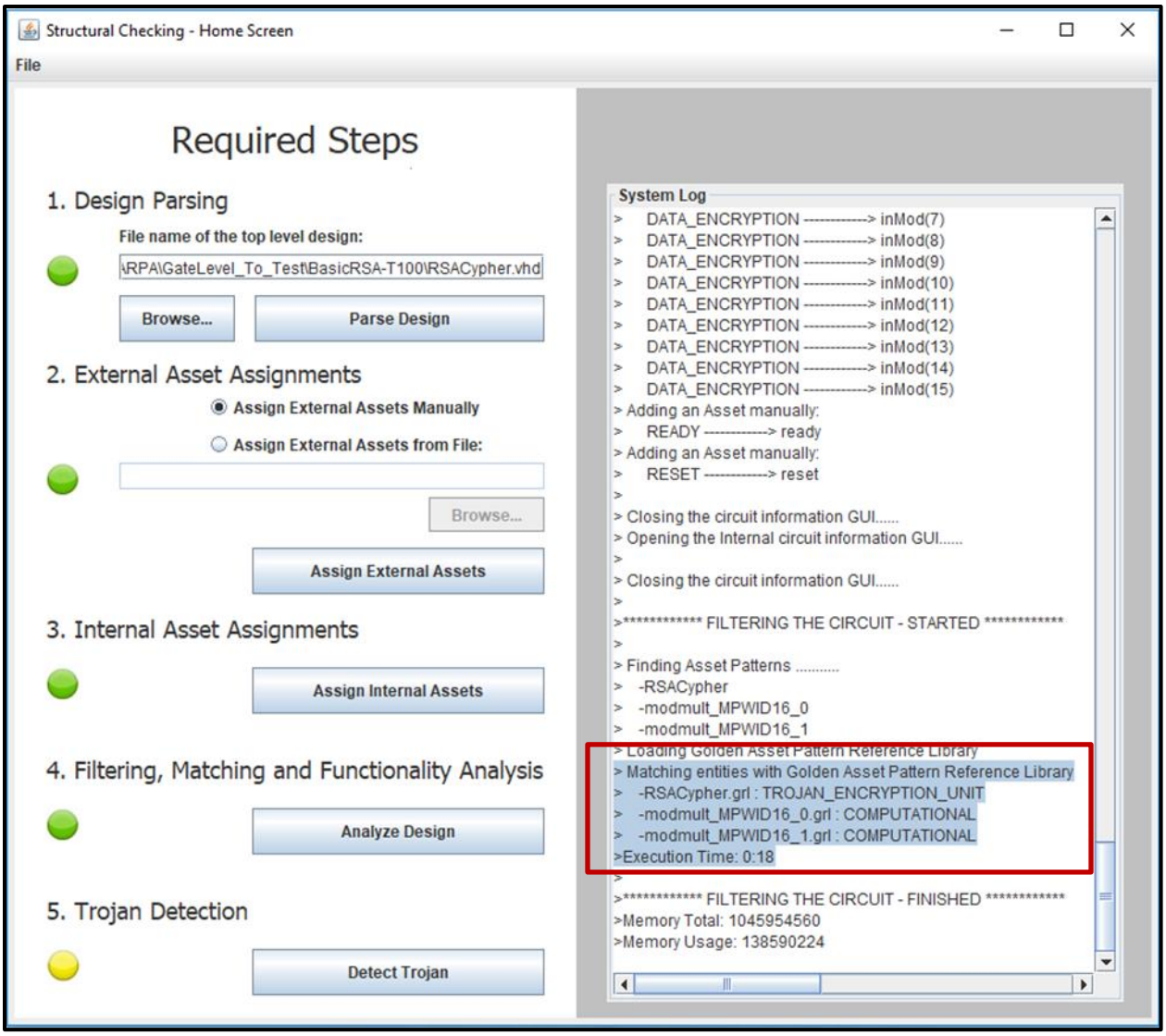

Figure 21 An example of matched result

7 CONCLUSION AND FUTURE WORK

The enhanced GR matching is an effective methodology that allows Structural Checking to detect hardware Trojan in a soft IP. First, the matching process includes basic asset pattern matching, partial asset pattern matching, and functionality matching. 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, matching the top-level and the sub-level of an unknown IP asset pattern to trusted top-level and sub-level of GRL allows the Trojan to be identified efficiently and effectively. The evaluating process of the HPM results ensures the coherence of the matching results. Based on a test vehicle suite, this detection process overall yields 6.23% of false positive rate in testing with RTL IPs and 3.57% of false positive rate in testing with gate-level IPs. The independent data flow tracking feature in SC allows for scan-chain analysis. For future development, both blacklist and whitelist of the GRL 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 internal assets and external assets can be added to improve the resolution of the analysis.

REFERENCES

[1] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC Fingerprinting," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, 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," in *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, 2008, pp. 87-95.

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

[4] A. Davoodi, M. Li, and M. Tehranipoor, "A Sensor-Assisted Self-Authentication Framework for Hardware Trojan Detection," *IEEE Design & Test*, vol. 30, no. 5, 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," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 4, pp. 776-780, 2015.

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

[7] S. Jha and S. K. Jha, "Randomization Based Probabilistic Approach to Detect Trojan Circuits," in *2008 11th IEEE High Assurance Systems Engineering Symposium*, 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," in *2008 Design, Automation and Test in Europe*, 2008, pp. 1362-1365.

[9] M. Banga and M. S. Hsiao, "A region based approach for the identification of hardware Trojans," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, 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, pp. 56-59.

[11] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware Trojans in third-party digital IP cores," in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2011, pp. 67-70.

[12] Y. Jin, N. Kupp, and Y. Makris, "DFTT: Design for Trojan Test," in *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, 2010, pp. 1168-1171.

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

[14] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: identification of stealthy malicious logic using boolean functional analysis," presented at the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, Berlin, Germany, 2013.

[15] J. Yust, M. Hinds, and J. Di, "Structural Checking: Detecting Malicious Logic without a Golden Reference," *Journal of Computational Intelligence and Electronic Systems,* vol. 1, no. 2, p. 8, 2012.

[16] L. Weaver, T. Le, and J. Di, "Golden Reference Library Matching of Structural Checking for securing soft IPs," in *SoutheastCon 2016*, 2016, pp. 1-7.

[17] T. Le and J. Di, "Golden reference matching for gate-level netlist functionality identification," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 567-570.

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

[19] T. Le, J. Di, M. Tehranipoor, and L. Wang, "Tracking data flow at gate-level through structural," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, 2016, pp. 185-189.

[20] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood, and R. Kastner, "Theoretical analysis of gate level information flow tracking," in *Design Automation Conference*, 2010, pp. 244-247.

[21] *OpenCores*. Available: http://opencores.org/

[22] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 471-474.