University of Arkansas, Fayetteville
ScholarWorks@UARK

Theses and Dissertations

5-2016

Hardware Trojan Detection via Golden Reference Library Matching

Lucas Weaver
University of Arkansas, Fayetteville

Follow this and additional works at: http://scholarworks.uark.edu/etd

Part of the Hardware Systems Commons, and the Information Security Commons

Recommended Citation
Weaver, Lucas, "Hardware Trojan Detection via Golden Reference Library Matching" (2016). *Theses and Dissertations*. 1460.
http://scholarworks.uark.edu/etd/1460

This Thesis is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, ccmiddle@uark.edu.

المنارة للاستشارات

www.manaraa.com

Hardware Trojan Detection via Golden Reference Library Matching

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Engineering

by

Lucas Weaver
John Brown University
Bachelor of Science in Engineering, 2014

May 2016
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

____
Dr. Jia Di
Thesis Director

Dr. James Parkerson Committee Member	Dr. Dale Thompson Committee Member

المنارة للاستشارات

www.manaraa.com

ABSTRACT

Due to the proliferation of hardware Trojans in third party Intellectual Property (IP) designs, the issue of hardware security has risen to the forefront of computer engineering. Because of the miniscule size yet devastating effects of hardware Trojans, few detection methods have been presented that adequately address this problem facing the hardware industry. One such method with the ability to detect hardware Trojans is *Structural Checking*. This methodology analyzes a soft IP at the register-transfer level to discover malicious inclusions. An extension of this methodology is presented that expands the list of signal functionalities, termed *assets*, in addition to introducing a methodology for matching soft IPs to a functionality category, termed *Golden Reference Library Matching*. Trojan detection methods are introduced that utilize the results of *Golden Reference Library Matching* as well as internal characteristics of the IP. This methodology is verified using benchmarks developed by a trusted third party.

المنارة للاستشارات

www.manaraa.com

ACKNOWLEDGEMENTS

Many thanks to my advisor, Dr. Jia Di, for his advising on this research. Thanks to Dr. Di's entire lab for everyone's support and mentorship. Particular thanks to Thao Le for her guidance in this research. Thanks also to my remaining committee members, Dr. James Parkerson and Dr. Dale Thompson, for their support of this research.

للاستشارات المنارة

www.manaraa.com

DEDICATION

To Dad, Mom, Syd, Eli and of course Dublin.

To Dad for giving me the heart and mind of an engineer and to Mom for everything else.

المنارة للاستشارات

www.manaraa.com

TABLE OF CONTENTS

1. INTRODUCTION **1**

1.1 MOTIVATION 1

1.2 OBJECTIVE 2

1.3 APPROACH 2

1.4 THESIS ORGANIZATION 3

2. BACKGROUND **4**

2.1 INTRODUCTION 4

2.2 HARDWARE TROJAN CATEGORIZATION 4

2.3 HARDWARE TROJAN DETECTION SURVEY 5

3. METHODOLOGY **9**

3.1 INTRODUCTION 9

3.2 ASSETS 10

3.2.1 INTERNAL ASSETS 10

3.2.2 EXTERNAL ASSETS 12

3.2.3 ASSET ASSIGNMENT 15

3.2.4 ASSET FILTERING 16

3.2.5 ASSET OPTIMIZATION 17

3.2.6 ASSET PATTERN 19

3.2.7. ASSET PATTERN FUNCTIONALITY 21

3.3 GOLDEN REFERENCE LIBRARY 22

3.3.1 GOLDEN REFERENCE LIBRARY CREATION AND CHARACTERISTICS 22

3.3.2 GOLDEN REFERENCE LIBRARY MATCHING 24

3.3.2.1 GRL MATCHING ALGORITHM 25

3.3.2.2 GRL PARTIAL MATCHING ALGORITHM 27

المنارة للاستشارات

www.manaraa.com

3.3.2.3 Functionality Considerations 30
3.3.2.4 Final Matching 31
3.3.3 Golden Reference Library Results 32
3.4 Trojan Detection Algorithms 33
3.4.1 Asset Pattern Algorithms 34
3.4.2 Functionality Assignment Algorithms 36
3.4.2.1 Blacklisted Functionalities 37
3.4.2.2 Suspicious Connections 38
3.4.2.3 Functionality Detection with Asset Pattern Recognition 39
3.4.3 RTL Characteristics 39
3.4.4 Trojan Detection Report 43
3.5 GUI Implementation 44
4. RESULTS 48
4.1 Introduction 48
4.2 GRL Matching 48
4.2.1 Asset Pattern Weighting 48
4.2.2 GRL Matching Example 51
4.3 Trojan Detection 53
4.3.1 Trust-Hub Benchmarks 54
4.3.2 Additional Trojan Examples 59
4.3.3 Trojan-Infested Crypto Core Example 63
4.3.4 Trojan-Infested Microcontroller 66
4.4 Analysis 71
5. CONCLUSION 76
5.1 Summary 76

المنارة للاستشارات
www.manaraa.com

5.2 Future Work 76

REFERENCES 78

المنارة للاستشارات

www.manaraa.com

LIST OF FIGURES

Figure 1: System Diagram 9

Figure 2: Asset Optimization Diagram 19

Figure 3: SPI Module Asset Pattern 21

Figure 4: SPI Module Asset Pattern with Delimiters 24

Figure 5: Golden Reference Library Matching High-Level Diagram 25

Figure 6: Golden Reference Library Match File 33

Figure 7: Trojan Detection High-Level Diagram 34

Figure 8: Trojan Detection Report 43

Figure 9: Trojan Detection Report with Driving Signals 44

Figure 10: GUI Home Screen 45

Figure 11: External Asset Assignment Dialog Box 46

Figure 12: First ALU Port Signals 49

Figure 13: Second ALU Port Signals 49

Figure 14: UART I/O Port Signals 51

Figure 15: UART Asset Pattern 52

Figure 16: Encryption Unit Key Leak VHDL Example 54

Figure 17: Encryption Unit Denial of Service VHDL Example 55

Figure 18: Time Bomb Counter VHDL Example 56

Figure 19: Encryption Unit Denial of Service VHDL Example 56

Figure 20: Counter Instance in Interrupt Unit 61

Figure 21: Interrupt Unit Denial of Service Attack 61

Figure 22: Trigger Assignment Attack 62

المنارة للاستشارات

www.manaraa.com

Figure 23: Crypto Core Port Signals 64

Figure 24: AES Time Bomb Trigger 64

Figure 25: AES Time Bomb Key Leakage 65

Figure 26: Microcontroller Primary Port Signals 67

Figure 27: ALU Denial of Service 68

Figure 28: Trojan-free Memory FSM 69

Figure 29: Trojan-infested Memory FSM 69

Figure 30: UART Trojan Attack 70

Figure 31: Trojan Shift Register Port Map 71

Figure 32: VHDL Concurrent Statements 73

المنارة للاستشارات

www.manaraa.com

LIST OF TABLES

TABLE 1: Internal Assets 10

TABLE 2: Data Assets 12

TABLE 3: Timing Assets 12

TABLE 4: System Control Assets 13

TABLE 5: Specific System Control Assets 13

TABLE 6: Miscellaneous Assets 14

TABLE 7: Types of Asset Patterns 20

TABLE 8: Functionalities 21

TABLE 9: Asset Pattern Delimiters 23

TABLE 10: General Asset Trace Matching Examples 26

TABLE 11: Partial Matching Asset Categories 28

TABLE 12: Partial Asset Trace Matching Examples 29

TABLE 13: Asset Pattern Characteristic Weighting 32

TABLE 14: Blacklist Functionalities 37

TABLE 15: UART Asset Assignment 52

TABLE 16: Microcontroller Asset Assignment 67

المنارة للاستشارات

www.manaraa.com

1. INTRODUCTION

1.1 Motivation

Many semiconductor manufacturers in today's world utilize the availability of third party IPs rather than create an entire system in-house. This scenario poses a threat to the system's security as third party IPs are vulnerable to the inclusion of hardware Trojans. Hardware Trojans take the form of unwanted or malicious logic included within a hardware design. The insertion of hardware Trojans allows the attacker to potentially gain possession of valuable information, such as encryption keys, or prevent the correct operation of the design by a denial of service attack. Hardware Trojans are often nearly impossible to detect during the testing and verification process, as they are triggered by a very specific set of circumstances that only the attacker knows.

The subject of hardware Trojan detection is an emerging field with few viable solutions currently in place. The methodologies that have proven to be more successful involve side-channel signal analysis in order to determine whether additional logic has been added to the design. However, a hardware Trojan inserted to a design has the potential to be as small as only a few logic gates and therefore cannot always be revealed through such analysis. Therefore, a more thorough examination of the design is required to reveal the inclusion of hardware Trojans.

Previous work published in [10] has been performed in the area of *Structural Checking*, and this research seeks to extend its capabilities. This method of analyzing soft IPs involves the parsing of a design at the register-transfer level in order to create a representation of the internal structure of the unknown IPs. Then, inclusions of malicious logic are identified by comparing the internal structure of the unknown design to trusted designs as well as examining the internal structure for suspicious connections. This strategy for detecting hardware Trojans has more

المنارة للاستشارات
www.manaraa.com

advantages than other methodologies currently in use for multiple reasons. First of all, by analyzing the design and detecting Trojans at the register-transfer level, the Trojan threats can be prevented early in the manufacture process. This allows the semiconductor companies to reduce the considerable amount of time and testing costs involved in the Trojan detection method using side-channel signal analysis. Additionally, by parsing the internal structure of a design, the *Structural Checking* methodology can detect smaller inclusions of malicious logic that the side-channel signal analysis cannot.

1.2 Objective

The objective of this research is to significantly increase the number of assets used to represent the role of a signal to provide differences among designs, as well as to create a matching methodology where an unknown design matches to a trusted design from a *Golden Reference Library*. In addition, for scalability, a hardware Trojan detection methodology is developed by using the matching methodology and the characteristics of a soft IP as a specific set of benchmarks.

1.3 Approach

Hardware designs, written in VHDL, are represented in the form of a collection of assets used to describe the role of signals. External assets are manually assigned to primary port signals of the design while internal assets are automatically assigned to signals immediately after the parsing of the VHDL design. Following asset assignment, assets are filtered throughout the designs along connections between signals. The result of the filtering process is a collection of assets assigned to each signal, which combine to form an asset pattern. Asset patterns effectively describe the characteristics of those designs, thus they can be used to compare to similar types of designs.

المنارة للاستشارات
www.manaraa.com

The representations of designs in terms of an asset pattern are collected and utilized in the form of a Golden Reference Library (GRL). The GRL is composed of the asset patterns of trusted designs that have been assigned a functionality. The asset patterns of unknown designs are compared against the asset patterns of trusted designs in the GRL in order to determine the functionality for the unknown design. Following the functionality assignment to an unknown design, the unknown design is analyzed for hardware Trojans. The methodology of hardware Trojan detection leverages multiple aspects to determine whether a Trojan is included in the design. The asset pattern, the functionality matching and the characteristics of the register-transfer level (RTL) code are all utilized in the identification of hardware Trojans.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 discusses background information regarding hardware Trojans along with similar research performed in the area of hardware Trojan detection. Chapter 3 presents the unique methodology of hardware Trojan detection, which is the thrust of the thesis. Chapter 4 summarizes the results of this methodology of Trojan detection and provides analysis for the testing results. Finally, chapter 5 summarizes the entire thesis in the form of a conclusion as well as provides potential areas of future work.

المنارة للاستشارات
www.manaraa.com

2. BACKGROUND

2.1 Introduction

Hardware Trojans must first be evaluated prior to an in-depth analysis of hardware Trojan detection via the *Structural Checking* methodology. Specifically, an introduction to the concept and classification of hardware Trojans is presented in order to understand the threat model for the *Structural Checking* methodology. Additionally, existing Trojan detection methodologies are evaluated in addition to show how the *Structural Checking* methodology advances the capability of hardware Trojan detection.

2.2 Hardware Trojan Categorization

Before surveying strategies of hardware Trojan detection, the concept of hardware Trojans needs to be explained. As [1] notes, "hardware Trojans are modifications to original circuitry inserted by adversaries to exploit hardware or to use hardware mechanisms to gain access to data or software running on the chip." As noted previously, many semiconductor companies rely on untrusted third party IPs. Therefore, even if they can ensure the trust of circuitry developed in-house, the third party IPs included in the final design are susceptible to Trojan insertion.

Characteristics exhibited by hardware Trojans allow for clear organization of hardware Trojans. The three main categories of hardware Trojans as identified by [1] and [15] are *physical*, *activation*, and *action*. The first category, *physical*, is broken down into multiple subcategories, some of which are self-explanatory. The first subcategory describes the type of Trojan, whether it involved gates added to the design or if gates were modified. The remaining categories are self-explanatory and consist of the size, distribution and structure of the hardware Trojan [15].

المنارة للاستشارات
www.manaraa.com

The second major category, *activation*, is also described as the trigger for the hardware Trojan. This category is further divided into *externally-activated* and *internally-activated* Trojans. In the case of *externally-activated* Trojans, the adversary alone knows a very specific input sequence resulting in the Trojan activation. *Internally-activated* Trojans are manifested in multiple, self-explanatory forms, such as *always-on* and *condition-based* [15].

The final major category, *action*, is also described as the payload of the hardware Trojan [15]. The malicious characteristics exhibited by the payloads of Trojans are further divided into three categories—*modify-function*, *modify-specification,* and *transmit-information* [15]. The first subcategory, *modify-function*, consists of attacks focused on augmenting the logic of the circuit in order to change its intended behavior. The *modify-specification* subcategory describes attacks intended to adjust certain properties of the circuit, such as clock frequency. The final subcategory, *transmit-information*, involves attacks focused on leaking important information to an attacker.

The *Structural Checking* methodology analyzes designs at the RTL and therefore is limited to certain types of hardware Trojans. Specifically, detecting Trojans in the physical characteristics subcategory is outside the scope of the *Structural Checking* methodology. Therefore, the hardware Trojans detected by the *Structural Checking* methodology are Trojans found at the RTL within the action and activation categories.

2.3 Hardware Trojan Detection Survey

Many Trojan detection methods have been proposed in previous research. The methodologies proposed range from Trojan activation techniques to side-channel analysis. This section gives a brief overview of the major Trojan detection methods.

المنارة للاستشارات
www.manaraa.com

A major technique utilized in the detection of hardware Trojans involves the analysis of side-channel characteristics of the circuit. Two of the major side-channel characteristics of a circuit to be analyzed in the detection of hardware Trojans are power consumption and current. A foundational work in the development of power analysis for the purposes of hardware Trojan detection is presented in [17]. By establishing a power signature for a particular type of circuit, the researchers found that hardware Trojans could be detected by identifying significant deviations from the power signature. The research performed in [20] similarly focused on side-channel characteristics, but limited the analysis to the current in isolated portions of the circuit. An additional side-channel characteristic measured the register-to-register path delay. The research performed in [18] establishes a technique for using the path delay measurement to verify the absence of a hardware Trojan. All of these methodologies achieved success in identifying larger hardware Trojans, but found difficulty in detecting smaller Trojans.

Another prominent methodology of Trojan detection involves the integration of sensors to available space of a design layout. The research performed by [2] proposes the sensors measuring the delays as a form of "self-authentication" to ensure that a design is Trojan-free. This is very similar to the research performed by [3] using sensors to measure the variability of path delays, although this research does not explicitly discuss using the on-chip sensors to detect hardware Trojans. Measurements performed by a ring oscillator network measuring power consumption on-chip coupled with statistical analysis allow the researchers in [4] to identify malicious inclusions to hardware designs. The methodologies of using on-chip monitors yield positive results in identifying specific types of Trojans, such as Trojans described by their physical characteristics.

المنارة للاستشارات
www.manaraa.com

An additional strategy of Trojan detection involves the purposeful activation of hardware Trojans. By performing various activation techniques, one can find the designs with Trojan inclusions by observing the Trojan payload. The researchers in [12] employ a probabilistic approach to Trojan activation through applying randomized test sequences to activate hardware Trojans. Another strategy of Trojan activation, as presented in [21], analyzes the circuit to determine nets that are rarely activated and in turn using test vectors that activate those same nets. A final strategy of Trojan activation found in [14] involves narrowing down the area of potential Trojan inclusion to a specific region and testing that region thoroughly for Trojans. While each of these methodologies achieved reasonable success, the strategy of Trojan activation has limitations since Trojans often require a very unique and complex activation sequence.

Another category of Trojan detection involves providing greater trust to third party IPs. The first of these methodologies, presented in [5], utilize functional vectors to remove trusted signals from consideration in order to isolate Trojans to a specific region of the design. The researchers in [6] employ an assortment of methodologies to identify hardware Trojans in third party IPs, such as formal verification and sequential ATPG. Finally, the research performed in [7] proposes a Design-for-Trojan-Test methodology that reduces the likelihood of Trojan insertion by making potential Trojan trigger sequences difficult to implement. While the research presented in this category yielded positive results, there are limitations to the number and size of Trojans that they can detect.

Additional methodologies focus on the security of third party IPs from a software analysis perspective rather than from a testing and verification perspective. The research performed by [8] compares functionally similar IP blocks to determine if malicious logic is present. The procedure by which this methodology compares the two IP blocks borrows from

المنارة للاستشارات
www.manaraa.com

the concept of loop unrolling in order to represent the internal logic states of both designs. Another methodology presented in [9] uses statistical analysis to assign values to signals based on their vulnerability to Trojan insertion. The vulnerability values are determined by the level of weight assigned to a statement as well as the observability of the statement. In doing so, a value for the level of trust of an entire IP block can be determined. While the research performed by both parties produce positive initial results, more advances must occur for these to be viable options for hardware Trojan detection.

As discussed previously, the methodology of hardware Trojan detection employed in this thesis is derived from the methodology of *Structural Checking* as originally presented in [19]. Previous work had been done in the area of modeling hardware threats in [16] and subsequently incorporated into the *Structural Checking* methodology presented in [19]. Research performed in [10] advanced the *Structural Checking* methodology through the creation of a software tool performing VHDL parsing and expression analysis as well as an initial conceptualization of hardware Trojan detection from a *Structural Checking* perspective. This research advances the *Structural Checking* methodology further to include a more robust collection of assets, a Golden Reference Library for functionality matching and hardware Trojan detection capability.

المنارة للاستشارات
www.manaraa.com

3. METHODOLOGY

3.1 Introduction

The Trojan detection methodology presented in this research is derived from the *Structural Checking* methodology developed in [10]. This methodology analyzes hardware designs at the register-transfer level (RTL) in order to determine the presence of hardware Trojans in the form of malicious logic. A high-level system diagram for this methodology can be found in Figure 1 below.

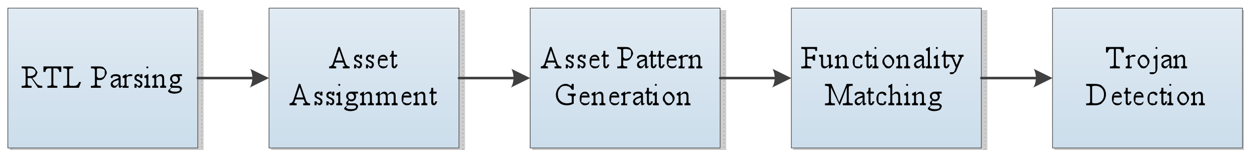

Figure 1: System Diagram

The design in question, written in VHDL, is first parsed in order to create a hierarchical structure of the data paths within the design. The VHDL parser was available as an open-source parser [22] and has been developed and implemented in a previous project as noted in [10]. Next, the primary port signals of the design are assigned assets, which denote the role of the signal within the design. The collection of assets assigned to primary port signals was greatly increased from the amount of assets found in the previous *Structural Checking* project. The assigned assets are filtered throughout the circuit through direct connections of the primary inputs and primary outputs determined by the VHDL parsing. Depending on the assets and their locations, the design is categorized based on its functionality by comparing it to trusted RTL designs in a Golden Reference Library (GRL). The GRL feature for categorizing designs was a completely new innovation added to the *Structural Checking* project. Finally, potential hardware Trojans within the design are identified by analyzing the results of the GRL categorization as well as characteristics of the RTL code. The Trojan detection process was also a new addition to

المنارة للاستشارات
www.manaraa.com

the project from the previous *Structural Checking* methodology. This entire methodology has been implemented and tested in the form of a graphical user interface that allows users to analyze a potentially malicious design for hardware Trojans. The tool produces output reports that present the results of hardware Trojan detection for user readability.

3.2 Assets

The concept of an asset was previously introduced in [10]. As [10] explains, an asset describes the purpose of a signal within a design. This concept is a foundation of Trojan detection, as it allows for signals to be represented in terms of a collection of assets. A signal's collection of assets will become important in later sections when dealing with Trojan detection.

Assets can be divided into two broad categories. The first category is termed internal assets. Primary port signals and internal signals both receive internal assets via automatic assignments. These assets are termed internal due to the fact that they refer to the way the signals are internally used within a system. The other category of assets is termed external assets. External assets are manually assigned to the primary port signals of a design.

3.2.1 Internal Assets

The first category of assets, internal assets, includes a set of assets that are automatically assigned to all signals within the designs. The assignment of internal assets occurs following the parsing of the RTL code by first looping through the code and searching for all logical expressions. The signals used in each logical expression are then identified and assigned an internal asset based on the role that it plays within the expression. Table 1 below shows a list of the 16 internal assets that have been developed along with a description of each asset.

TABLE 1: Internal Assets

Asset	Description
PROCESS_SENSITIVE	Assigned to a signal contained within a process sensitivity list.

المنارة للاستشارات
www.manaraa.com

TABLE 1: Internal Assets (Cont.)

Asset	Description
PROCESS_OPERATION_SENSITIVE	Assigned to a signal being modified within a process statement.
CONDITIONAL_DRIVING	Assigned to a signal contained within a conditional statement.
CONDITIONAL_DRIVEN	Assigned to a signal being modified within a conditional statement.
CONCURRENT_DRIVING	Assigned to a signal driving another signal via a concurrent statement.
CONCURRENT_DRIVEN	Assigned to a signal being driven by another signal via a concurrent statement.
CC_OPERATION_OR	Assigned to a signal being driven by a concurrent statement performing an OR operation.
CC_OPERATION_AND	Assigned to a signal being driven by a concurrent statement performing an AND operation.
CC_OPERATION_XOR	Assigned to a signal being driven by a concurrent statement performing an XOR operation.
CC_OPERATION_NOR	Assigned to a signal being driven by a concurrent statement performing a NOR operation.
CC_OPERATION_NAND	Assigned to a signal being driven by a concurrent statement performing a NAND operation.
CC_OPERATION_XNOR	Assigned to a signal being driven by a concurrent statement performing an XNOR operation.
CC_OPERATION_NOT	Assigned to a signal being driven by a concurrent statement performing a NOT operation.
CC_OPERATION_A_ADD	Assigned to a signal being driven by a concurrent statement performing an addition operation.
CC_OPERATION_MULT	Assigned to a signal being driven by a concurrent statement performing a multiplication operation.
CC_OPERATION_SENSITIVE	Assigned to a signal being driven by a concurrent statement using another type of logic than the types previously listed.

The assets above can be broken down into three categories. The first category deals with assets assigned to signals used in process statements. Certain internal assets are assigned to signals based on whether the signal appears in a process sensitivity list or whether it is used inside the process itself. The second category deals with assets assigned to signals used in conditional statements. Similarly to the assets dealing with process statements, conditional assets can be assigned to signals found within a conditional statement or to signals being

www.manaraa.com

modified within a conditional statement. Finally, numerous assets are assigned to signals being used within concurrent statements. These assets are assigned based on the logic used within the concurrent statement. As a signal can be used in multiple types of expressions in RTL code, multiple internal assets can be assigned to a signal.

3.2.2 External Assets

The second category of assets is external assets, and these assets are manually assigned to primary port signals. As opposed to internal assets that describe how a signal is used internally in the RTL code, external assets describe how a primary port signal is used. There are a total of 51 external assets that are distributed among several broad categories.

The first category contains assets describing data signals. Table 2 below shows the assets within this category along with the definition of the specific asset.

TABLE 2: Data Assets

Asset	Description
DATA_COMPUTATIONAL	Assigned to data signals within ALUs, adder, multipliers, etc.
DATA_MEMORY	Assigned to data signals being stored in memory.
DATA_PERIPHERAL	Assigned to data signals being used by peripheral units.
DATA_COMMUNICATION	Assigned to data signals being used for communication purposes by communication units.
DATA_ENCRYPTION	Assigned to data signals being used being encrypted by encryption units.
DATA_SENSITIVE	This is the most general of the data assets and can be assigned to signals containing data that does not fit any other category.

The second category consists of assets related to the timing of a system. Table 3 below shows the assets within this category along with the definition of the specific asset.

TABLE 3: Timing Assets

Asset	Description
SYSTEM_TIMING	Assigned to the primary clock signal.
SUBSYSTEM_TIMING	Assigned to a subsystem clock signal.
STATUS	Assigned to a signal indicating the status of the system.
DONE	Assigned to a signal indicating that an operation is finished.
HOLD	Assigned to a signal indicating to hold an operation.
READY	Assigned to a signal indicating that an operation is ready.

www.manaraa.com

TABLE 3: Timing Assets (Cont.)

Asset	Description
BUSY	Assigned to a signal indicating that an operation is busy.
COUNT	Assigned to a signal used as a counter.
WAIT	Assigned to a signal indicating that an operation must wait.
TIMER_CONTROL	Assigned to a signal controlling a timer.
CLOCK_CONTROL	Assigned to a signal controlling the primary or subsystem clock.

The next category involves assets assigned to signals used for system control. Table 4 below shows the assets within this category along with the definition of the specific asset.

TABLE 4: System Control Assets

Asset	Description
SET	Assigned to a signal used to set a value.
RESET	Assigned to a signal used to reset a value.
READ	Assigned to a signal used to perform a read operation.
WRITE	Assigned to a signal used to perform a write operation.
SELECT	Assigned to a signal used to perform a select operation.
EXECUTE	Assigned to a signal indicating that an operation is to be executed.
LOAD	Assigned to a signal indicating that a value is to be loaded.
MODE	Assigned to a signal indicating the mode of an operation.
ENABLE	Assigned to a signal used to perform an enable operation.
HANDSHAKING	Assigned to a signal used in communication by way of a handshaking operation.
SHIFT	Assigned to a signal indicating that a shift operation is to occur.
INSTRUCTION	Assigned to a signal used as an instruction. This is the most general form of this asset and should only be used when a more specific asset does not describe the signal.
SYSTEM_CONTROL	Assigned to a signal that is used in system control. This is the most general system control asset and should only be used when a more specific asset does not describe the signal.

The next category of assets is a subset of the previous category of system control assets. These assets are specific to a certain type of subsystem. Table 5 below shows the assets within this category along with the definition of the specific asset.

TABLE 5: Specific System Control Assets

Asset	Description
MEMORY_OP	Assigned to a signal used to perform an operation within a memory subsystem.
DATA_OP	Assigned to a signal used to perform an operation within a subsystem dealing with data.

المنارة للاستشارات
www.manaraa.com

TABLE 5: Specific System Control Assets (Cont.)

Asset	Description
INTERRUPT_OP	Assigned to a signal used to perform an operation within an interrupt unit subsystem.
PROGRAM_COUNTER_OP	Assigned to a signal used to perform an operation within a program counter.
INTERRUPT_CONTROL	Assigned to a signal used as system control within an interrupt unit subsystem.
PERIPHERAL_CONTROL	Assigned to a signal used as system control within a peripheral subsystem.
REGISTER_FILE_CONTROL	Assigned to a signal used as system control within a register file subsystem.
COMMUNICATION_CONTROL	Assigned to a signal used as system control within a communication subsystem.
COMMUNICATION_PROTOCOL	Assigned to a signal used to handle a protocol within a communication subsystem.
COMMUNICATION_STATUS	Assigned to a signal indicating the status of an operation within a communication subsystem.
INTERRUPT	Assigned to a signal used to handle an interrupt requests.

The final category of assets is simply a miscellaneous category. These assets do not clearly fit into any one category and are therefore grouped together in the miscellaneous category. Table 6 below shows the assets within this category along with the definition of the specific asset.

TABLE 6: Miscellaneous Assets

Asset	Description
CRITICAL	Assigned to an asset that could lead to harm if an attacker gained possession of it.
COMPONENT	Assigned to an asset that refers to another component of a system.
ADDRESS_SENSITIVE	Assigned to an asset that describes the address used in a memory subsystem.
CONSTANT	Assigned to a signal that describes a value to be used as a constant.
KEY	Assigned to a signal used as an encryption key in an encryption unit.
REGISTER	Assigned to a signal used to handle data to be used in a register file subsystem.
PROGRAM_COUNTER	Assigned to a signal used as the value being manipulated within a program counter.
ERROR_HANDLING	Assigned to a signal that performs error handling.
EXCEPTION_HANDLING	Assigned to a signal that performs error handling.

المنارة للاستشارات
www.manaraa.com

TABLE 6: Miscellaneous Assets (Cont.)

Asset	Description
STATE	Assigned to a signal that tracks the state of system.

3.2.3 Asset Assignment

As mentioned in the previous section, users assign external assets to the primary port signals of a design. This requires the user to understand how the signal of a design is used in the system. Oftentimes, the choice of external asset is very simple, as there is a direct correlation between the external asset to be assigned and the signal it is assigned to. However, there are cases in which the user must deduce the external asset to be used based on the closest match to the functionality of the signal. Therefore, several rules should be considered when assigning assets.

First of all, the most important rule in assigning external assets is that the most specific asset appropriately describing the signal should be assigned. As the descriptions of the assets in the previous section show, there are certain assets that are general in nature, such as *DATA_SENSITIVE* and *SYSTEM_CONTROL*. These assets should only be assigned in the case that no other assets best describe the functionality of a signal. For instance, when considering the assignment of an asset to a data signal in an ALU, the more specific external asset *DATA_COMPUTATIONAL* should be used rather than the general *DATA_SENSITIVE* asset. Additionally, if there is not an exact asset describing a signal's role within the system but there is an asset functionally similar to the signal under consideration, then that asset should be assigned. For example, when considering the asset assignment of a *clear* signal, the functionally similar asset, *RESET*, should be assigned to the *clear* signal.

The second rule to consider when assigning assets involves the number of assets assigned to a signal. Primary port signals can have multiple assets assigned to the same signal in the case

www.manaraa.com

the signal cannot be appropriately described by a single asset. For example, a signal may be used to perform *read/write* operations. In that case, both the *READ* and *WRITE* assets should be assigned to the signal in order to appropriately describe its functionality. However, the ideal scenario is that a single asset can be assigned that appropriately describes the functionality of that signal. This is especially important when considering the first rule above stating that the most specific asset to describe a signal should be used. For example, if a signal is a data signal within a communication unit, only the *DATA_COMMUNICATION* asset should be assigned, rather than additionally assigning another data asset that may only partially describe the functionality of the signal.

The final rule to consider when assigning assets involves asset assignment for system specific assets. This rule is similar to the first in that it requires a user to assign the most specific asset possible to describe a signal. More specifically, this rule involves assigning assets specific to a type of system if the functionality of the system is known. For example, if an asset is used as an instruction within an ALU, the general *INSTRUCTION* asset should not be assigned. Instead, the more system-specific asset *DATA_OP* should be assigned. Even in the case that an asset describes the functionality of the signal, the system specific asset should be used. For example, if a *read* signal is being analyzed in memory unit, the *MEMORY_OP* asset should be chosen rather than the *READ* asset. The assignment of system-specific assets aids in the future step of functionality matching.

3.2.4 Asset Filtering

Following the VHDL parsing and assignment of assets, the next step in the process is to filter the assets throughout the circuit. This involves passing the assets previously assigned to the primary port signals to lower level signals based on the internal connections of the circuit.

المنارة للاستشارات
www.manaraa.com

For example, if a primary input port signal directly drives an internal signal via a concurrent statement, the internal signal would receive the assets assigned to the input signal. Filtering occurs both from input to output and vice versa, meaning that assets can be passed from input to output and from output to input. This process is repeated recursively until every connection within the circuit has been reached. At this point, every signal within the circuit has been populated with assets that describe all possible functionalities of the signal. The information necessary to filter the assets is collected during the VHDL parsing. The development of the methodology and implementation of asset filtering was previously described by [10].

3.2.5 Asset Optimization

An additional step following the filtering of assets through the circuit, termed asset optimization, occurs in order to ensure a precise asset pattern. Asset optimization involves the analysis of each individual port signal's external and internal assets to ensure that there are no redundant assets. As past analysis of asset filtering has shown, asset patterns for large designs can be extremely large, resulting in every signal essentially having the exact same asset pattern. Asset optimization corrects this issue by removing filtered assets that do not actually contribute to the functionality of the circuit.

Asset optimization is only performed on the primary port signals of the circuit that have been assigned assets. The internal assets of these signals are then analyzed to determine if the signal in question is driving another signal or being driven by a signal. Certain assets are removed based on whether the signal is driving another signal or being driven by another signal. In the case of the signal being driven by another signal and the original signal contains a data asset, the system control assets relevant to data operations are removed. For example, if a data signal assigned a *DATA_MEMORY* asset has received a *MEMORY_OP* asset during the filtering

المنارة للاستشارات
www.manaraa.com

process, the *MEMORY_OP* asset is removed from the collection of assets assigned to the data signal. Conversely, in the case that a signal is driving another signal and the original signal contains a system control asset, the data assets relevant to the system control operations are removed. For example, if a system control signal assigned a *MEMORY_OP* asset has received a *DATA_MEMORY* asset during the filtering process, the *DATA_MEMORY* asset is removed from the collection of assets assigned to the system control signal. This process allows the true functionality of the signal to represented with assets rather than allowing the filtered assets to distort the functionality of the signal.

There are also certain optimizations that are performed regardless of whether the signal is driving or being driven by another signal. As mentioned previously there are certain assets that are very general in nature, such as *DATA_SENSITIVE* and *SYSTEM_CONTROL*. In the case where one of these assets is present and other more specific assets are present, the general assets are removed. For example, in the case that a signal is assigned a *DATA_COMPUTATIONAL* asset and a *DATA_SENSITIVE* asset is filtered to the signal, the *DATA_SENSITIVE* asset will be removed from the signal's collection of assets. However, it is important to note that only filtered assets can be removed through optimization while the assigned assets are permanent. The entire process of asset optimization is illustrated in Figure 2 below.

المنارة للاستشارات
www.manaraa.com

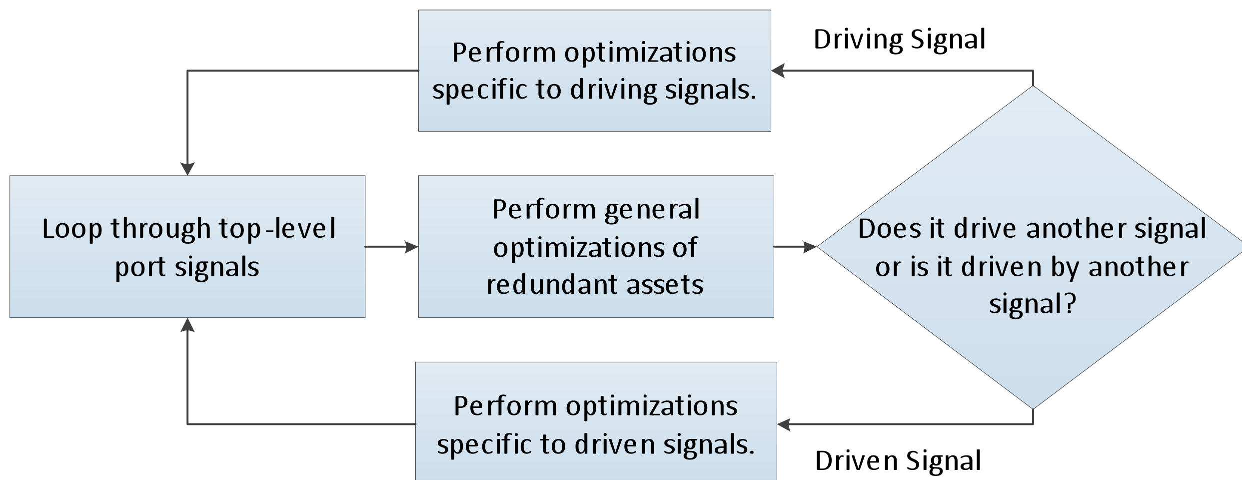

Figure 2: Asset Optimization Diagram

3.2.6 Asset Pattern

The results of the filtering and optimizing of assets is termed an asset pattern. Asset patterns describe the system in terms of a collection of internal and external assets that have been filtered throughout the circuit. This is accomplished by listing the internal and external assets that have been assigned to individual signals. The collection of assets assigned to a specific signal is termed an asset trace. The collection of asset traces forms an asset pattern for a system.

Asset traces originate from the external assets that are manually assigned to a signal and the internal assets that are automatically assigned to a signal. Once the assets have been filtered throughout the circuit, each individual signal contains a collection of assets that have been assigned initially as well as assets that have been filtered to the signal. Each asset within the asset trace is unique. This means that the asset traces do not repeat assets even in the case that an asset has been both assigned and filtered to the signal. Additionally, each signal has both an external asset trace as well as an internal asset trace. An example of an external asset trace found in a communication unit can be seen below:

[DATA_COMMUNICATION, COMMUNICATION_PROTOCOL]

An example of an internal asset trace can be seen below:

المنارة للاستشارات
www.manaraa.com

[CONDITIONAL_DRIVEN, PROCESS_SENSITIVE, CONDITIONAL_DRIVING]

Both of these examples show the list of external and internal assets that were both assigned and filtered to a signal.

Asset traces are grouped into six separate categories based on the type of assets being assigned and the type of signals the assets are being assigned to. The two types of assets are internal and external, while the types of signals that the assets are being assigned to are primary input/output signals and internal signals. Therefore, the categories can be seen in Table 7 below:

TABLE 7: Types of Asset Patterns

Asset Pattern Type	Description
input port signal external asset pattern	Collection of external asset traces assigned to top level input port signals
output port signal external asset pattern	Collection of external asset traces assigned to top level output port signals
internal signal external asset pattern	Collection of external asset traces assigned to internal signals
input port signal internal asset pattern	Collection of internal asset traces assigned to top level input port signals
output port signal internal asset pattern	Collection of internal asset traces assigned to top level output port signals
internal signal internal asset pattern	Collection of internal asset traces assigned to internal port signals

The combination of these six sets of asset traces forms the asset pattern of a single circuit. The asset pattern represents the unique combination of assets that are used to describe a circuit design. An example of an asset pattern of a SPI module can be seen below in Figure 3.

المنارة للاستشارات
www.manaraa.com

[SYSTEM_TIMING]
[PROCESS_SENSITIVE, CONDITIONAL_DRIVING]
[RESET]
[COMMUNICATION_CONTROL]
[CONDITIONAL_DRIVING]
[READ, WRITE]
[ADDRESS_SENSITIVE]
[DATA_COMMUNICATION, COMMUNICATION_PROTOCOL]
[DATA_COMMUNICATION]
[CONDITIONAL_DRIVEN]
[INTERRUPT]
[SUBSYSTEM_TIMING]
[CONCURRENT_DRIVEN]
[COMMUNICATION_PROTOCOL, DATA_COMMUNICATION]
[COMMUNICATION_CONTROL]
[COMMUNICATION_PROTOCOL]
[PROCESS_SENSITIVE, CONDITIONAL_DRIVING, CONDITIONAL_DRIVEN]
[COMMUNICATION_PROTOCOL, DATA_COMMUNICATION]

Figure 3: SPI Module Asset Pattern

As is indicated in this asset pattern, many of the assets are related to communication. It could then be inferred that the circuit described by this asset pattern belongs to the category of communication without the previous knowledge that it is a SPI module. This fact will be used in future sections to match asset patterns to functionalities.

3.2.7. Asset Pattern Functionality

Following the creation of an asset pattern for a system, a functionality is assigned to the design. As the name implies, the functionality assignment is intended to effectively describe the purpose of the design. Table 8 below lists the types of functionalities that the design could be assigned to.

TABLE 8: Functionalities

Functionality	Description
SHIFT_REGISTER	Assigned to a circuit being used to shift data in and out.
INTERRUPT_UNIT	Assigned to a circuit handling interrupt requests.
COMMUNICATION	Assigned to a circuit handling communication.
ENCRYPTION_UNIT	Assigned to a circuit being used to encrypt or decrypt data.

للاستشارات المنارة
www.manaraa.com

TABLE 8: Functionalities (Cont.)

Functionality	Description
COMPUTATIONAL	Assigned to a circuit being used to manipulate data, such as an ALU, adder or multiplier
TIMING	Assigned to a circuit responsible for controlling the timing of a system.
CONTROL_GENERATION	Assigned to a circuit used to handle system control.
REGISTER_FILE	Assigned to a circuit used to store data
PERIPHERAL	Assigned to a circuit handling peripherals other than communication.
DECODER_ENCODER	Assigned to a circuit used to encode or decode data.

As the list indicates, many functionalities have direct correlations to assets that were presented previously. This was intentional, as this aids in determining the functionality of a particular design.

3.3 Golden Reference Library

The asset patterns generated by asset assignment and filtering are essential in the analysis of unknown designs. A Golden Reference Library (GRL) is formed by obtaining asset patterns from trusted IPs that in turn can be used to compare against unknown IPs to determine their level of trust. The GRL contains files with characteristics of the individual designs that have been deemed to be trusted. The unknown design is then compared to the GRL designs and assigned a functionality based on the closest GRL design match. The resulting data is outputted in user-readable format.

3.3.1 Golden Reference Library Creation and Characteristics

The Golden Reference Library was created by first obtaining trusted designs for each type of functionality. The asset pattern for each of the trusted designs are generated and added to the GRL to be used as the golden references. Due to the fact that these are the basis of matching unknown designs, a sufficient amount of trusted designs for each category must be chosen in order to guarantee that an unknown design could match sufficiently to a trusted design or else be

المنارة للاستشارات
www.manaraa.com

deemed untrusted. Obtaining designs for the GRL is an ongoing process as more trusted designs can always be added to represent more types of designs. However, the designs currently present in the GRL are sufficient to match many unknown designs.

The trusted designs were obtained through numerous sources. The main source of designs was the website OpenCores [13], an open-source repository for hardware designs. Many of their trusted designs were incorporated into the GRL. The remaining trusted designs were either collected from additional online repositories or were simply created during the course of the project. One potential issue that arose when creating the GRL is that there are numerous ways to implement designs of a specific category, and it is impossible to find and implement all possible designs. However, this issue is addressed by the addition of assets specific to a functionality as well as allowing the GRL to be constantly updated with new designs. The specific assets allow designs to be matched with functionalities closely related to those assets.

GRL files contain several important characteristics used in the matching process. The main information used in this process is the complete asset pattern for that design. In order for the asset pattern to be parsed by the tool, certain delimiters were used to identify the specific portions of the asset pattern. The delimiters for each type of asset pattern can be seen in Table 9 below.

TABLE 9: Asset Pattern Delimiters

Asset Pattern Type	Delimiter
input port signal external asset pattern	>
output port signal external asset pattern	<
internal signal external asset pattern	/
input port signal internal asset pattern	>*
output port signal internal asset pattern	<*
internal signal internal asset pattern	/*

As the table shows, the addition of the symbol '*' indicates that an asset pattern is an internal asset pattern, while the absence of that symbol indicates that it is an external asset

المنارة للاستشارات
www.manaraa.com

pattern. An example asset pattern with delimiters inserted can be seen in Figure 4 below. This figure is the same asset pattern as the SPI module from before with delimiters.

```
>[SYSTEM_TIMING]
>*[PROCESS_SENSITIVE, CONDITIONAL_DRIVING]
>[RESET]
>[COMMUNICATION_CONTROL]
>*[CONDITIONAL_DRIVING]
>[READ, WRITE]
>[ADDRESS_SENSITIVE]
>[DATA_COMMUNICATION, COMMUNICATION_PROTOCOL]
<[DATA_COMMUNICATION]
<*[CONDITIONAL_DRIVEN]
<[INTERRUPT]
<[SUBSYSTEM_TIMING]
<*[CONCURRENT_DRIVEN]
<[COMMUNICATION_PROTOCOL, DATA_COMMUNICATION]
<[COMMUNICATION_CONTROL]
<[COMMUNICATION_PROTOCOL]
/*[PROCESS_SENSITIVE, CONDITIONAL_DRIVING, CONDITIONAL_DRIVEN]
/[COMMUNICATION_PROTOCOL, DATA_COMMUNICATION]
```

Figure 4: SPI Module Asset Pattern with Delimiters

The asset patterns generated by the asset filtering process are stored in files termed *GRL files* with a ".grl" extension. Each GRL file is stored under the Golden Reference Library directory. This directory is located such that it is easily found in order to be used later in the GRL matching process. If additional trusted designs were generated by the asset filtering process, the associated GRL file could simply be added to this directory.

3.3.2 Golden Reference Library Matching

The creation of the Golden Reference Library is used as the basis for matching unknown designs to a functionality. Functionality matching is necessary for determining the level of trust to be assigned to the unknown design. Therefore, a matching methodology was developed that allows the unknown design to be matched to a known design in the GRL. The asset patterns of the trusted designs found within GRL files provided a characteristic to be used for comparison

www.manaraa.com

between the unknown designs and the trusted designs. However, many factors of the asset patterns must be analyzed in order to create an algorithm for matching. This section discusses the various characteristics of the GRL matching algorithm along with examples of how portions of unknown designs would be matched to trusted designs in the GRL.

3.3.2.1 GRL Matching Algorithm

As discussed in previous sections, the asset pattern generated by asset filtering contains six separate characteristics based on the type of signal (input and output port signals and internal signals) and the type of asset (internal and external). Therefore, during the matching of asset patterns, each characteristic is analyzed and matched to a potential design. As Figure 5 below indicates, the beginning of the matching process is to loop through each entry of the GRL in order to compare the characteristics of each GRL entry to the unknown design. Each characteristic is analyzed individually and assigned a percentage match that is then used to determine the total match.

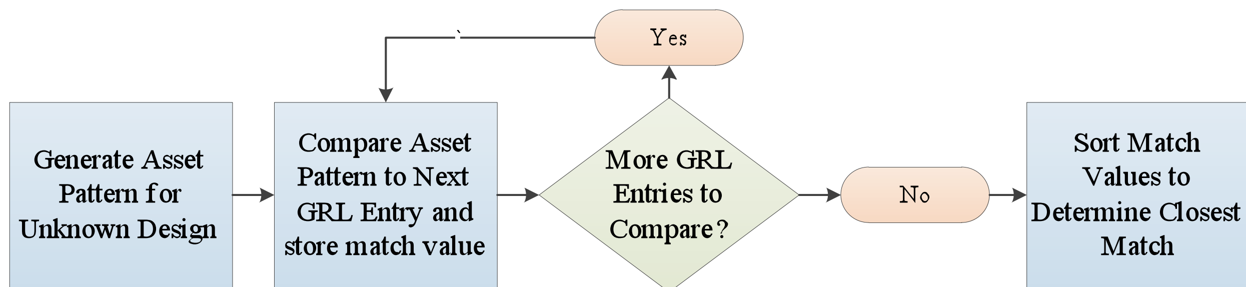

Figure 5: Golden Reference Library Matching High-Level Diagram

The match percentage of each individual characteristic is contained in a hash map using the name of the GRL entry as the key and the match percentage as the value. There are six hash maps as there are six characteristics used in the matching process. The values are obtained while looping through the GRL entries and comparing the individual asset pattern characteristics of the GRL entry to the unknown design. The matching of the individual characteristics is essentially

www.manaraa.com

an intersection of sets in order to determine the percentage of assets contained within the unknown design with respect to each GRL entry.

While looping through the GRL entries, the individual asset traces of the asset pattern characteristics are taken individually and compared to the asset traces of the same asset pattern characteristics of the unknown design. For example, the first asset pattern characteristic of the GRL entry, the input port signal external asset pattern, is broken up into individual asset traces and compared to the input port signal external asset traces of the unknown designs. Each asset trace is looped through individually in order to find the largest match between asset traces. The largest match is found by finding the intersection of the two asset traces (many examples are given below for clarification). Once the largest match is assigned to the individual asset traces, they are added together and divided by the total number of asset traces in order to find the asset pattern percentage match. This is summarized in the Equation 1 below.

(Eq. 1)

In order to clarify the matching process, several examples have been given in Table 10. These examples show how each asset trace of an unknown design matches to an asset trace of a GRL entry.

TABLE 10: General 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%

www.manaraa.com

Case 1 in Table 10 gives the simple scenario that both asset traces are identical. In this case, the match is clearly 100%. Case 2 is also a simple example showing that the match will be 0% when the asset traces have no intersection. Case 3 shows the scenario in which the two asset traces intersect on two assets; however, the GRL entry contains a third asset not found in the unknown design asset trace, causing the final match to be 67%. Finally, Case 4 considers the scenario in which one asset is shared while the other asset is different. This case yields a 50% asset trace match. If these asset traces where combined to form an asset pattern for a specific asset pattern characteristic, the final match for that characteristic could be easily determined by averaging the percentages listed. Therefore, the asset pattern characteristic match would be 54.25%.

In certain instances, one of the characteristics of a design may be empty. For example, if a design does not contain internal signals, then there will be no assets assigned to internal signals and those asset pattern characteristics will be empty. In the case that both the GRL entry and the unknown design both have the same characteristic as empty, the hash map containing the match percentages is marked with a -1 value indicating that this characteristic will be left out of the final matching.

3.3.2.2 GRL Partial Matching Algorithm

In order to gain a more precise indication of the match between asset traces, an algorithm for partially matching asset traces was developed. This algorithm takes into account the fact that there is occasional overlap among assets and therefore there must be a way to assign a percentage match to these assets greater than zero. For example, when matching the *DATA_SENSITIVE* asset to the *DATA_COMPUTATIONAL* asset using the normal intersection set matching, the result would be a 0% match. However, due to the fact that these assets are

المنارة للاستشارات
www.manaraa.com

similar in nature, the matching percentage should be greater than 0%. This is the problem addressed by the partial matching algorithm.

The partial matching algorithm is implemented during the matching of asset traces. Using general intersection set matching, the assets are analyzed and assigned a value by determining whether or not the assets found within one set matches the assets found in the other set. Instead, partial matching runs each individual asset through a method that analyzes the asset to determine if it is part of a subset of similar assets. The set of similar assets are then searched for within the asset trace intended for matching to determine whether or not any are found. In the case that another asset that is similar in nature to the asset in question is found in the other asset trace, a 50% match is assigned for that individual asset. This 50% match is then factored in with the remaining assets that are a part of the asset trace.

The partial matching algorithm uses certain subsets of assets to determine whether or not an asset should receive a partial match. These subsets of assets can be found in Table 11 below.

TABLE 11: Partial Matching Asset Categories

SYSTEM_CONTROL	***DATA_SENSITIVE***	***INSTRUCTION***	***STATUS***
SELECT	*DATA_ ENCRYPTION*	*DATA_OP*	*READY*
READ	*DATA_ COMMUNICATION*	*MEMORY_OP*	*DONE*
WRITE	*DATA_ COMPUTATIONAL*	*PROGRAM_ COUNTER_OP*	*HOLD*
INSTRUCTION	*DATA_MEMORY*	*INTERRUPT_OP*	*STATUS*
MODE	*DATA_PERIPHERAL*		*BUSY*
SET			*WAIT*
RESET			
ENABLE			
EXECUTE			
HANDSHAKING			
LOAD			

المنارة للاستشارات
www.manaraa.com

TABLE 11: Partial Matching Asset Categories (Cont.)

SYSTEM_CONTROL	*DATA_SENSITIVE*	*INSTRUCTION*	*STATUS*
SHIFT			
INTERRUPT_ CONTROL			
PERIPHERAL_ CONTROL			
REGISTER_FILE_ CONTROL			
COMMUNICATION_ CONTROL			
CLOCK_CONTROL			
TIMER_CONTROL			

As the table shows, there are four categories of assets that can be partially matched. The four categories are represented by the assets listed in the top row of each column. The partial matching algorithm consists of first identifying whether one of the assets listed in the top row is contained within either the GRL entry asset trace or the unknown design asset trace. If one of the assets is found in either asset trace, the other asset trace is parsed for one of the assets in the column of the original asset found. If one of those assets is found, a 50% match is assigned for this set of assets.

In order to clarify the partial matching algorithm, several examples are shown in Table 12 below, listing the two asset traces along with a partial match percentage.

TABLE 12: Partial Asset Trace Matching Examples

Case	Unknown Design Asset Traces	GRL Entry Asset Traces	Match
1	*DATA_MEMORY*	*DATA_ SENSITIVE*	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, SYSTEM_CONTROL*	25%

Case 1 provides the simple example of two assets that can be partially matched at 50%. The matching scenario in Case 2 shows that the two *DATA_ SENSITIVE* assets are matched at

المنارة للاستشارات
www.manaraa.com

100%; however, since the unknown design contains an additional asset, *DATA_ MEMORY*, the final match is 50%. It is important to note with this example that since the *DATA_SENSITIVE* asset in the GRL entry was matched with the *DATA_SENSITIVE* asset in the unknown design, it could not be partially matched with the *DATA_MEMORY* asset. Case 3 shows that even though the two assets are data assets, they are not partially matched. For a partial match to occur, one of the assets had to be a *DATA_SENSITIVE* asset. Case 4 describes a scenario in which the *SET* assets are matched at 100% while the *ENABLE* and *SYSTEM_CONTROL* assets are matched at 50%, resulting in a final match of 75%. Finally, Case 5 describes a scenario in which the *SYSTEM_CONTROL* asset is partially matched to the *RESET* asset at 50%. Since the GRL entry has one additional asset, the final match is 25%.

3.3.2.3 Functionality Considerations

In addition to the partial asset matching, an algorithm that considers the functionality of the GRL entries was developed in order to take advantage of the precise nature of many of the assets. The external assets of the unknown design can give an indication as to what functionality that design may be. By searching for the functionality-specific assets assigned to the unknown design, the algorithm takes into consideration the GRL entries with that same functionality and weights them greater than the entries without that functionality assignment.

As mentioned, this aids in the final matching of the unknown design to a functionality. During the analysis of individual asset pattern characteristics, the algorithm recognizes whether or not one of these assets is contained within the asset pattern characteristic. In the case that a functionality-specific asset is present, all of the GRL entries with the corresponding functionality are given a weight of 1.5 in order to give these entries precedence. Therefore, the match percentage for only that characteristic will be multiplied by 1.5. For example, in the case that the

المنارة للاستشارات
www.manaraa.com

input port signal external asset pattern contains a *DATA_ENCRYPTION* asset, the match percentage for the input port signal external asset pattern of all GRL entries of functionality type *encryption unit* are multiplied by 1.5. However, the other asset pattern characteristics do not necessarily receive the multiplier unless other functionality-specific assets are also contained in those characteristics.

3.3.2.4 Final Matching

Once each individual asset pattern characteristic hash map has been filled with values representing the individual asset pattern matches, the final asset pattern match for each GRL entry can be determined. Each individual asset pattern characteristic is taken into account during the final asset pattern match; however, not all of the asset pattern characteristics are weighted equally. There are several reasons for this. The most important reason is that certain designs can be implemented in several different ways. For example, the internal implementation of one ALU may be completely different from that of another ALU even though they accomplish the same purposes. However, both of the ALUs will have similar I/O port signal external assets. Therefore, the I/O port signal external assets should be weighted higher than the internal characteristics of the design, including the internal signal assets. An additional reason for having larger weight for I/O port signal external assets is that the internal assets of multiples designs are similar to each other even if the designs have different functionalities.

Now that it has been established that not all asset pattern characteristics should be weighted the same, the next question to address is the amount of weight that each characteristic should be given. After performing extensive testing in order to determine the proper weight, the values seen in Table 13 below were finalized.

المنارة للاستشارات
www.manaraa.com

TABLE 13: Asset Pattern Characteristic Weighting

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

As the table shows, the port signal external asset patterns receive the largest weight, three times larger than any other characteristic (the weighting of the asset pattern characteristics and the testing results will be discussed further in subsequent sections). Once each individual asset characteristic values have been determined, the above weighting is applied to each characteristic, producing the final match value for each GRL entry to the unknown design. As mentioned previously, certain designs do not contain every asset pattern characteristic. Therefore, during the final matching these characteristics are simply omitted and the remaining characteristics are used to match an unknown design to the GRL entry.

3.3.3 Golden Reference Library Results

Following the generation of matching values of a GRL entry to an unknown design, an output file is generated containing the results of the matching analysis. This file contains the match percentage along with the average match for each functionality. An example of the output file can be seen in Figure 6 below.

المنارة للاستشارات
www.manaraa.com

```
        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
```

Figure 6: Golden Reference Library Match File

The first line of this file gives the name of the unknown design being analyzed, which in this case is *i2c_master*. The second line gives the best GRL entry match, which in this case is another i2c unit matching at 92.5%. This line also gives the matching percentage for each asset pattern characteristic. The remaining lines list the average percentage match among all functionalities with respect to the unknown design. As one would expect, the communication functionality has the largest percentage match at 62%. Additional information can be added to this file at the user's discretion, such as the match value for every GRL entry.

3.4 Trojan Detection Algorithms

The final and most important component of analyzing unknown designs is the process of Trojan detection. Several methodologies are employed in the detection of hardware Trojans. First, the results of asset filtering are utilized in order to identify suspicious asset patterns. Asset filtering can reveal suspicious connections between signals within the design and therefore is essential in the identification of hardware Trojans. Another methodology by which hardware Trojans may be identified consists of utilizing the functionality matching accomplished by the Golden Reference Library in order to match an untrusted design to a blacklisted design. Functionality matching also provides the opportunity of identifying suspicious connections

المنارة للاستشارات
www.manaraa.com

among otherwise trusted instances. Finally, characteristics of the RTL code itself are analyzed in order to detect specific hardware Trojans embedded within the logic of the circuit.

In order to implement the methodologies of Trojan detection, the entire design is parsed and each signal is analyzed individually by applying certain heuristics for determining the presence of a Trojan. A diagram of this process can be seen below in Figure 7.

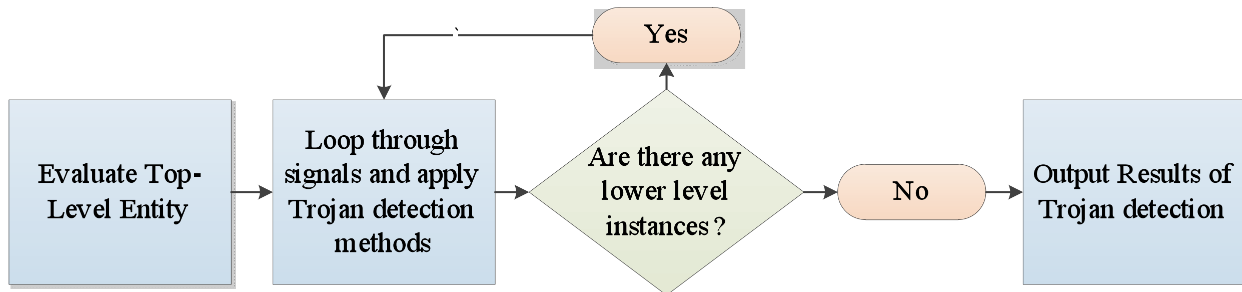

Figure 7: Trojan Detection High-Level Diagram

Of course not all possible hardware Trojans are intended to be detected using these methodologies, as new hardware Trojans are constantly being developed. However, since the implementation of these methodologies was done in modular fashion, additional Trojan detection methods may be added in the future with relative ease. The hardware Trojan designs used as the basis of these methodologies can be found on Trust-Hub's website [11].

3.4.1 Asset Pattern Algorithms

The first of the methodologies of Trojan detection, asset pattern recognition, consists of evaluating the external and internal asset traces assigned to an individual signal. The internal and external asset traces are used in conjunction to determine whether or not the signal is being affected by a hardware Trojan. For instance, the external asset trace of a signal can be analyzed to determine the external assets that have been filtered to a signal. In certain cases, a Trojan can be identified by simply identifying that an external asset has been filtered to the signal, exposing a suspicious connection between internal signals. Additionally, the internal asset trace assigned

www.manaraa.com

to a signal can be useful in identifying suspicious driving assets by determining the internal logic affecting that signal.

The first type of Trojan detected using asset pattern recognition involves the analysis of signals dealing with the timing of a design. One type of attack on a timing signal consists of a *SET* or *RESET* asset being filtered to a signal that had been originally assigned a *SYSTEM_ TIMING* or *SUBSYSTEM_TIMING* asset. This scenario is an example of a denial of service attack against a timing signal of a design, as the *set* or *reset* signal could disable the timing of the circuit. This attack is detected by analyzing the asset traces of the timing signals and searching for filtered assets such as *SET* and *RESET*.

Additional Trojan detection methods involving asset pattern recognition identify malicious logic being implemented within encryption units. The first method involves identifying the modification of an encryption unit key by an attacker. The benchmarks developed by Trust-Hub specifically address this potential Trojan [11]. If an attacker were able to modify the encryption key, then the he or she would be able to decrypt any message being encrypted by the encryption unit. This attack is identified by detecting the internal asset trace assigned to a signal with a *KEY* external asset. If the internal asset trace contains any assets indicating that the signal has been modified, such as a *CONCURRENT_DRIVEN* asset, the signal is known to have been modified. Therefore, the signal containing these assets is flagged as malicious. Another attack involving encryption units consists of the encryption key being leaked to the output. In this scenario, an attacker could gain access to the key of the encryption unit by using a certain trigger sequence. This type of attack is detected by analyzing the output port signal external asset pattern to determine if a *KEY* asset has been filtered to any of the outputs. If this scenario has occurred, the signal that has received a *KEY* asset is flagged as malicious.

المنارة للاستشارات
www.manaraa.com

Another asset pattern algorithm involves the analysis of signals containing *CRITICAL* assets. Since *CRITICAL* assets are intended for signals that are to be protected from leaking information to attackers, these signals are extremely important to protect. Therefore, one of the Trojan detection methods checks the outputs of all circuits to see if a *CRITICAL* asset has been passed to it. If this is the case, then that output is marked as susceptible to critical information leakage. A similar Trojan detection method analyzes the input and output signals of a top-level design. For each of these signals, this method checks the assigned assets and compares them to the filtered assets. In the case that the set of filtered assets are not contained within the set of tagged assets, it is possible that a suspicious connection has been made with one of the signals in question. In this case, the signal is marked as suspicious and presented to the user for further inspection.

The final Trojan detection method pertaining to the analysis of asset patterns involves analyzing the primary port signals of the unknown design. This method checks the assets that have been assigned to the primary signals and compares them to the filtered assets of the same signal. Due to the fact that malicious logic could be inserted into the unknown design, unexpected assets could be filtered to the primary signals. In the case that the filtered assets are not contained within the set of assets that have been assigned to the signal, the port signal is marked as being connected to malicious logic within the unknown design.

3.4.2 Functionality Assignment Algorithms

The next set of Trojan detection algorithms consists of analyzing the functionality assigned to a design during GRL matching. Functionality assignment can be leveraged in several ways in order to detect the inclusion of a hardware Trojan. First, the process of GRL functionality matching can reveal that a design has been compromised by Trojans by matching

المنارة للاستشارات
www.manaraa.com

the design to a blacklisted functionality. Additionally, functionality assignment can reveal suspicious connections between instances that otherwise would seem harmless. Finally, the functionality assignment, coupled with asset pattern recognition, can reveal information leakages in large designs.

3.4.2.1 Blacklisted Functionalities

The first category of functionality algorithms used to detect Trojans incorporates a blacklist of Trojan-infested designs. The use of a blacklist to detect Trojans is a natural extension of the Golden Reference matching presented previously. Rather than matching an unknown design to a trusted design that is a part of the whitelisted GRL, the Golden Reference matching analyzes the unknown design to determine if it matches to a blacklisted functionality. The blacklist contains designs that are known to contain Trojans. Therefore, if an unknown design matches to a design containing Trojans rather than a trusted design, the unknown design is labeled as suspicious. The blacklisted designs were accumulated by creating asset patterns from the Trojan-infested examples developed by Trust-Hub [11].

A list of the blacklist functionalities can be seen in Table 14 below. As the table indicates, several of the blacklisted functionalities are similar to the whitelisted functionalities noted previously. Therefore, the blacklisted functionalities are necessary to detect small differences in implementation between legitimate designs and Trojan-infested designs.

TABLE 14: Blacklist Functionalities

Functionality	Description
TROJAN_ENCRYPTION_UNIT	Assigned to Trojan-infested encryption units leaking
TROJAN_TRIGGER	Assigned to Trojan triggers designed to initiate Trojan attacks in other entities
TROJAN_SHIFT_REGISTER	Assigned to Trojan-infested shift registers
TROJAN_COMMUNICATION	Assigned to Trojan-infested communication units

www.manaraa.com

3.4.2.2 Suspicious Connections

Another aspect of functionality detection that can be utilized to detect Trojans is the identification of suspicious connections between instances. After first assigning functionalities to unknown entities and sub-entities of a design, the connections between the instances of the entities can be analyzed to determine whether or not a Trojan is present. Occasionally, an instance found in a legitimate design can be used as a trigger to leak information.

Several suspicious types of connections are specifically searched for during the Trojan detection process. The first two involve suspicious connections involving an encryption unit. Certain instances within an encryption unit can allow information leakage if an activation sequence is coded. The first of these connections involves the use of a shift register to leak data from the encryption unit. This scenario has been implemented in one of Trust-Hub's benchmarks. Another connection that allows information leakage from an encryption unit involves a counter instance being used as a trigger.

Another functionality that is susceptible to this type of attack is the register file. Since the register file stores data, an attacker may seek to gain access to the data via a hardware Trojan. Lower level connections of a register file could be used as a trigger to gain access to this data. Similarly to the encryption unit Trojan detection methods, the register file entities are searched to uncover possible lower level connections with shift register or counter functionalities. In all of these cases, the malicious connection is identified by first starting with the top-level entity of an analyzed design and recursively searching for all connections between entities of the design. If a higher level entity, such as the encryption unit or register file, contains a lower-level connection between entities that could serve as a trigger, such as a counter or a shift register, then the lower-level entity is marked as malicious and presented to the user for inspection.

المنارة للاستشارات
www.manaraa.com

3.4.2.3 Functionality Detection with Asset Pattern Recognition

The final category of functionality-based Trojan detection methods involves the additional information of the asset pattern in order to identify Trojans. This additional information assists in the detection of information leakages as it allows for identification of specific assets found within suspicious functionalities. Due to the fact that the external assets were developed and intended for specific functionalities, if one of the specific assets was found outside of the designated functionality, malicious connections between instances could be present in the RTL code. One of the Trojan detection methods of this category involves the leakage of an encryption unit key. If an encryption key were found outside of an encryption unit, there would be a strong indication that an attacker is trying to obtain the key and decrypt messages from the encryption unit. Additionally, other types of assets are verified to be contained only within designated functionalities in order to guarantee that there are no suspicious connections between instances. By analyzing the individual signals of a design and comparing them to the functionality of that design, these types of attacks can be recognized and reported. For example, another Trojan detection method within this category consists of the analysis of interrupts being found outside interrupt units. Interrupt signals found within certain functionalities, such as *TIMING* and *CONTROL_GENERATION*, could be used to disrupt the control of the design by a malicious attacker. The attacker could send a false interrupt to trigger an unwanted state. Therefore, if an interrupt is filtered to an entity that typically does not process interrupts, the interrupt signal is marked as being malicious.

3.4.3 RTL Characteristics

The final category of Trojan detection algorithms consists of parsing the RTL code to discover malicious logic inserted by attackers. Oftentimes such logic will consist of various

المنارة للاستشارات
www.manaraa.com

Trojan triggers, such as time bomb counters or finite state machines. These triggers initiate the leakage of data or denial of service. After recognizing potential triggers during the parsing of the RTL code, certain algorithms are applied to the signal in question in order to determine its legitimacy. These algorithms also consist of analyzing the external and internal asset trace assigned to the signal.

The first category of Trojans detected using RTL code consists of denial of service attacks. Denial of service attacks are extremely prevalent in hardware against such signals as clocks and interrupts and result in a portion of the circuit becoming unavailable due to malicious logic. Detecting denial of service attacks involves the analysis of individual signal assignments. A common scenario in which a signal is the subject of a denial of service attack consists of the attacker substituting a Trojan signal for the actual signal. Consequently, the intended signal does not have an internal asset pattern as it is a floating signal in the design and the Trojan signal is taking its place. Therefore, denied signals can be recognized by identifying the internal asset trace for internal assets. If there are no internal assets indicating that the signal drives or is driven by another signal, then that signal is being denied and is marked as such. Additional denial of service attacks can be applied to encryption keys. Attackers could potentially modify the encryption key and could use a different key known only to them. This is detected by analyzing the encryption key signal and determining whether it has been modified.

Another category of Trojans detected using RTL code involves time bomb counters. Time bomb counters are used as a trigger for many types of Trojan attacks, such as information leakage or denial of service. The signals used to implement time bomb counters can be identified by examining the structure of the RTL code. Each signal is parsed to determine whether it serves as a counter within the design. If so, additional verifications are applied to

المنارة للاستشارات
www.manaraa.com

determine whether the signal is being used as a Trojan trigger. This is accomplished by checking the internal asset trace of the signal to determine whether it conditionally drives another signal. If this is the case, the signal is marked as a potential time bomb counter and the signal that is triggered by the counter is marked as a susceptible to a time bomb counter. A separate Trojan detection method related to time bomb counters also performs a verification to determine whether the signals being conditionally driven by the counter is connected to the output. While the previous time bomb counter detection method targets all types of Trojan attacks, this detection method specifically targets information leakage attacks that result in a suspicious driving assignment to an output signal. If the time bomb counter is conditionally driving an output, the counter is marked as a trigger for an information leakage attack and the susceptible signals are also noted. Trust-Hub provides several examples dealing with time bomb counters, and these were used for testing this methodology.

An additional category of Trojan detection methods involving RTL code consists of finite state machine detection. Trojans can be implemented by attackers in the form of a finite state machine (FSM) with an unwanted state that is only entered in rare conditions. The result of the trigger condition can be information leakage or denial of service. The first detection method involving an FSM examines the conditional signal used to determine the state of an FSM. The number of states in the FSM is also examined to determine if it matches the number of cases possible based on the size of the conditional signal. In the case that there is not an *OTHERS* state listed, a Trojan could be implemented in the RTL code by having an unwanted state that would typically be sent to the *OTHERS* state. Therefore, the FSM is marked as suspicious. Additionally, the FSM is checked to guarantee that every possible state is accounted for as defined by the size of the conditional signal. In the case that there are fewer states than the

www.manaraa.com

signal allows, the FSM is marked as suspicious. This is due to the fact that an attacker could gain access to the gate-level netlist and insert an additional state to perform an unwanted task. These FSM detection methods are used as warnings against potentially malicious logic being implemented by an attacker and indicate to the user that the FSM should be verified as secure.

Several more Trojan detection methods involve analysis of RTL code. First, the assignment statements of individual signals are checked for trigger sequences. For example, an assignment such as "*X <= Y(0) AND Y(1) AND Y(2)*" is a potential Trojan activation sequence for the signal *X* as it only goes high in the case that the *Y* vector reaches the value "111." This type of signal assignment could be a Trojan trigger sequence implementing a denial of service or information leakage attack. However, additional criteria must be met in order to mark the assignment as suspicious. The signal *X* must also be triggering a process or driving a conditional statement. This can easily be verified by analyzing the internal asset trace assigned to the potential trigger signal. If the signal *X* contains an internal asset indicating that it triggers another signal, the signal assignment of *X* is marked as suspicious. Additionally, any other signal being driven by *X* is marked as susceptible to a Trojan attack.

One final Trojan detection method using RTL code involves the detection of extra circuitry added to a design. An attacker often inserts extra circuitry in order to increase the payload and/or perform unnecessary switching activities. Several examples provided by Trust-Hub implement attacks of this nature. While not every scenario of adding circuitry can be detected by this methodology, the examples provided by Trust-Hub have been tested and detected. In Trust-Hub's examples, an entire instance is being used as additional circuitry within a design. However, the instance used as additional logic does not contain outputs as it only performs unnecessary switching. Therefore, the detection method analyzes the instances to

المنارة للاستشارات
www.manaraa.com

determine if output signals are present. If no outputs are present, the instance is marked as an addition of malicious logic.

3.4.4 Trojan Detection Report

After performing all Trojan detection methods, a list of Trojans is compiled and presented in user-readable format. An example of a portion of a Trojan detection report can be found in Figure 8 below.

```
Trojan Information for:
  RSACypher:

Type of Trojan found: KEY_LEAK
Entity:
        RSACypher
Instance:
        Top_Level_Instance
Signal:
        inExp

Type of Trojan found: ENCRYPTION_UNIT_LEAK
Entity:
        RSACypher
Instance:
        Top_Level_Instance
Signal:
        cypher
```

Figure 8: Trojan Detection Report

The Trojan detection report contains specific information about each Trojan present in the design. In this example, the top of the report lists the name of the entity being analyzed, *RSACypher*. This entity is a Trojan-infested encryption unit implementing the RSA algorithm. A portion of the report shown lists two different Trojans present in the design, *KEY_LEAK* and *ENCRYPTION_UNIT_LEAK*. In addition to the Trojan type, the output report also lists the entity, instance and signal that are being affected by the Trojan. In this example, both Trojans

www.manaraa.com

are being applied to the *RSACypher* entity. In some cases, additional information is added to the output report. An example of this can be seen in Figure 9 below.

```
Type of Trojan found: TIME_BOMB_COUNTER
Entity:
        input_output
Instance:
        Top_Level_Instance
Signal:
        TIMER

Type of Trojan found: TIME_BOMB_SIGNAL
Entity:
        input_output
Instance:
        Top_Level_Instance
Signal:
        SECRETKey
Driving Signal:
        TIMER(15)
```

Figure 9: Trojan Detection Report with Driving Signals

In this example, a time bomb counter has been detected by the methodology. Therefore, the output report notes the signals that contribute to the implementation of the Trojan. As the report indicates, the *TIMER* signal is used as the time bomb counter to leak the *SECRETKey*. Therefore, the Trojan detection report also lists the “Driving Signal” used to leak the *SECRETKey* signal for the user to know which signal is being affected by the time bomb counter. In this case, multiple Trojans are present in the same design.

3.5 GUI Implementation

The entire methodology of hardware Trojan detection is implemented using a Java-based GUI. This tool allows a user to easily navigate to a VHDL file to be analyzed for Trojans. The main GUI home screen can be seen below in Figure 10.

المنارة للاستشارات
www.manaraa.com

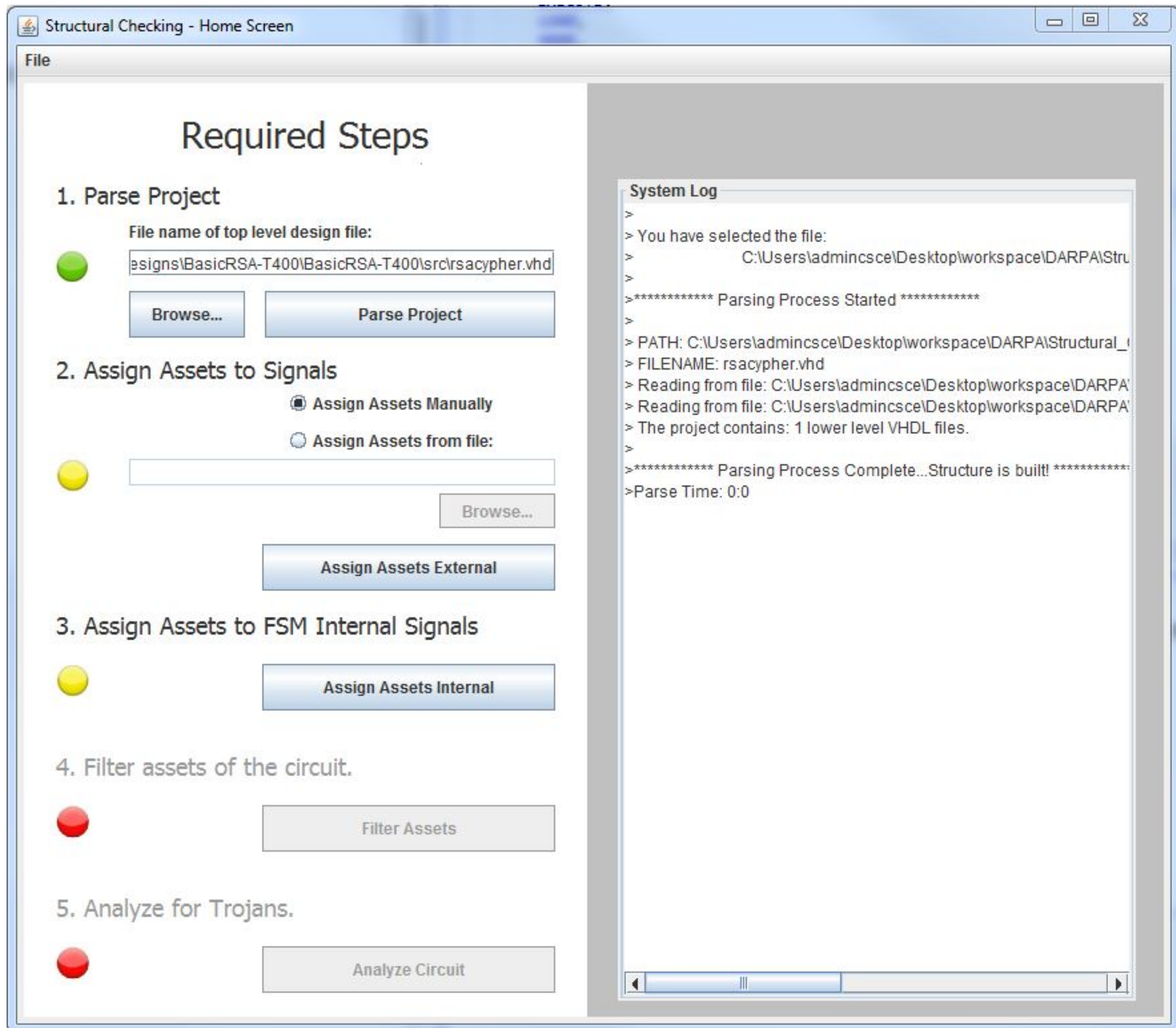

Figure 10: GUI Home Screen

The left side of the screen indicates the steps that must be initiated by the user in order to perform the methodology while the right side keeps a log of the entire process. Each individual step is fairly self-explanatory. The colored dots next to the step indicate the status of each step. A green dot indicates that the process has been finished, a yellow dot indicates that the step is ready to be initiated and a red dot means that previous steps must be completed before the step can be initiated. Step 1 allows the user to browse to the top-level file and parse the VHDL code. In the screenshot of the GUI shown above, this step has already been initiated and logged on the right side of the screen. The second step allows the user to assign external assets to the primary

المنارة للاستشارات
www.manaraa.com

port signals of the chosen design. The external assets can either be assigned manually or imported from an asset assignment file (the third step is used to assign specific internal assets for another project). After assets have been assigned, the fourth step initiates both asset filtering and GRL matching. At this point, the output report giving the results of GRL matching is generated. The final step in the process is to analyze the design for Trojans. The results of the Trojan detection are also written to a file for the user to examine.

During the second step of the tool, the user has the option of assigning assets manually. If this option is chosen, the dialog box found in Figure 11 appears.

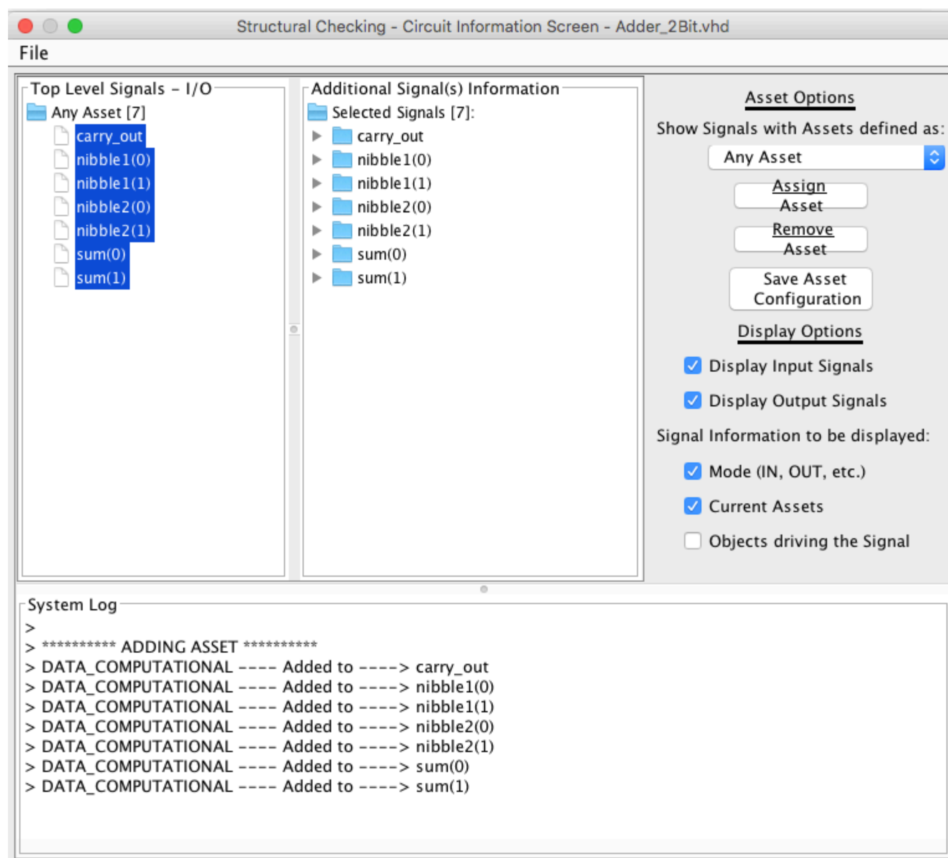

Figure 11: External Asset Assignment Dialog Box

As the screenshot shows, the user has the option of choosing any primary I/O port signal from the list on the left side of the screen. On the right side of the screen, the user has the option of assigning an asset to the selected signal with the “Assign Asset” button. If this button is

المنارة للاستشارات
www.manaraa.com

selected, an additional dialog box appears giving the entire list of assets for the user to choose from. Additionally, the user could remove an asset that was accidentally assigned with the "Remove Asset" button. Once all assets are assigned the user has the option to save the asset assignments to a ".asset" file for future usage. Finally, a log is shown at the bottom of the dialog box to allow the user to track the assets that have been assigned.

Several output reports are generated while using the tool. Following the fourth step in the process, the percentage match file is generated along with a GRL file for each entity and sub-entity of the design being run. Both of these files are in the "OutputFiles" directory, which is in the same location of the tool. The other generated file is the Trojan detection report, which can be found in the "TrojanFiles" directory in the same location as the tool.

المنارة للاستشارات
www.manaraa.com

4. RESULTS

4.1 Introduction

The individual portions of the Trojan detection methodology were tested and analyzed concurrently with their development. The majority of the testing focused on collecting results of the GRL matching and Trojan detection methods and guaranteeing that they performed as expected. The Trust-Hub benchmarks were instrumental for the testing of the individual Trojan detection methods. After ensuring the correctness of these methods, they were tested on larger designs to ensure that they could detect multiple Trojans in the same large design.

4.2 GRL Matching

The Golden Reference Library methodology for matching unknown designs to entries in the GRL required significant testing in order to ensure that the matching produced correct results. The first step in testing the matching was to optimize the matching specifications, particularly with regard to the weighting of the asset pattern characteristics of the design. Once the matching methodology was finalized, many unknown designs were tested by matching to a GRL entry. An example is given to show the process by which an unknown design is matched to a GRL entry along with the results of the matching.

4.2.1 Asset Pattern Weighting

The weighting of the individual asset pattern characteristics was discussed in detail in the methodology section. As noted, the external asset patterns of the input and output port signals were given precedence over all other asset patterns by multiplying their match value by three. This value was obtained through two guiding principles as well as trial and error.
The first principle in establishing a weighting value for each asset pattern characteristic is that port signals are the most important in determining the functionality of an unknown design.

المنارة للاستشارات
www.manaraa.com

While the internal implementation of a functionality type can vary, the port signals often share the same characteristics. For example, Figure 12 below shows the port signals of an ALU. As the list of port signals show, there are two input data vectors and one output data vector. Additionally, the design contains an operation signal as well as a carry out and a flag. For comparison, Figure 13 shows the port signals of a different ALU.

```
entity ALU_VHDL is
  port
  (
    Nibble1, Nibble2 : in std_logic_vector(3 downto 0);
    Operation : in std_logic_vector(2 downto 0);
    Carry_Out : out std_logic;
    Flag : out std_logic;
    Result : out std_logic_vector(3 downto 0)
  );
end entity ALU_VHDL;
```

Figure 12: First ALU Port Signals

```
entity simple_alu is
port(   Clk : in std_logic; --clock signal
        A,B : in signed(7 downto 0); --input operands
        Op : in unsigned(2 downto 0); --Operation to be performed
        R : out signed(7 downto 0)  --output of ALU
        );
end simple_alu;
```

Figure 13: Second ALU Port Signals

As the port signals of the second design show, there are many similarities between the two ALUs. Each ALU contains an operation signal as well as two data input vectors and a result output vector. If the specific assets of *DATA_COMPUTATIONAL* and *DATA_OP* are assigned to these signals, the port signal asset pattern matches will be nearly identical. However, the internal logic of the ALU designs consists of vastly different implementations resulting in completely different internal asset patterns. If the internal asset patterns of the designs were weighted the same as the port signal asset patterns, the close match of the port signal asset

المنارة للاستشارات
www.manaraa.com

patterns would cancel with the poor match of the internal asset patterns. Therefore, it would not indicate that each design belongs to the same category. Thus, the weighting for the port signal asset patterns should be greater than the internal asset patterns.

After establishing that the port signal asset patterns should be given greater weights than the internal asset patterns, the next question is how much weight they should be given. Based on the previous principle, there is no identifiable limit to the weight given to a port signal. Therefore, the second principle gives guidance to an approximate limit for the weight of port signals. The second principle states that the internal asset patterns reveal the inclusion of malicious logic, and therefore must also carry weight in order to match to blacklist designs containing Trojans.

The testing of values for the weight of the individual characteristics consisted of trial and error with multiple weighting values. After implementing multiple weighting values, the two options that resulted in the best matches were between two and three as the weight factor to the port signal asset patterns. The two options were compared by running the same design through the matching process using both values and comparing the results. Rather than only comparing the value of the closest matched design, the average match value for entire functionality categories were examined. In this way, the improvement of using certain values as a weight factor was easier to determine. After further comparison between these two options, the weighting value of three produced the closest match. Noticeable improvement was seen across multiple types of designs in the desired functionality category. More importantly, the average match values for the categories other than the matched functionality were significantly lower with the weight value of three than that of two.

للاستشارات المنارة
www.manaraa.com

4.2.2 GRL Matching Example

The following example proves the effectiveness of the GRL matching methodology as well as illustrating the process of matching an unknown design to a GRL entry. As mentioned in the methodology section, an unknown design is analyzed first through the parsing process followed by asset assignment and filtering to produce an asset pattern. The unknown design chosen to illustrate the matching methodology is a basic UART communication module. The primary I/O signals of the design can be seen in Figure 14 below.

```
entity uart is
port (
   clk: in std_logic;
   reset: in std_logic;
   rx_data: out std_logic_vector(7 downto 0);
   rx_enable: out std_logic;
   tx_data: in std_logic_vector(7 downto 0);
   tx_enable: in std_logic;
   tx_ready: out std_logic;
   rx: in std_logic;
   tx: out std_logic
 );
end uart;
```

Figure 14: UART I/O Port Signals

Following the parsing process, assets were assigned to each port signal. The assignment of the assets was fairly straightforward. The assigned assets are in Table 15 below. After filtering the assets throughout the circuit, the asset pattern found in Figure 15 was produced. Due to the specificity of the assets, the majority of the assets assigned to this design are directly involved in the process of communication. Therefore, during the matching phase of the methodology, the communication functionalities should theoretically have the highest match value.

المنارة للاستشارات
www.manaraa.com

TABLE 15: UART Asset Assignment

Signal	Asset
clk	*SYSTEM_TIMING*
reset	*RESET*
rx_data	*DATA_COMMUNICATION*
rx_enable	*COMMUNICATION_CONTROL*
tx_data	*DATA_COMMUNICATION*
tx_enable	*COMMUNICATION_CONTROL*
tx_ready	*COMMUNICATION_STATUS*
rx	*COMMUNICATION_CONTROL*
tx	*COMMUNICATION_CONTROL*

```
>[SYSTEM_TIMING]
>*[PROCESS_SENSITIVE, CONDITIONAL_DRIVING]
>[RESET]
<[DATA_COMMUNICATION]
>[DATA_COMMUNICATION]
>*[PROCESS_SENSITIVE]
>[COMMUNICATION_CONTROL]
<[COMMUNICATION_STATUS]
<[COMMUNICATION_CONTROL]
/*[PROCESS_SENSITIVE, CONDITIONAL_DRIVEN]
/*[CONDITIONAL_DRIVEN]
/*[CONDITIONAL_DRIVEN, CONDITIONAL_DRIVING]
/[DATA_COMMUNICATION]
/[COMMUNICATION_CONTROL]
/*[CONDITIONAL_DRIVEN, PROCESS_OPERATION_SENSITIVE]
/*[CONDITIONAL_DRIVING]
/[COMMUNICATION_STATUS]
/*[PROCESS_SENSITIVE, CONDITIONAL_DRIVEN, CONDITIONAL_DRIVING]
/*[CONDITIONAL_DRIVEN, CONDITIONAL_DRIVING,
PROCESS_OPERATION_SENSITIVE]
```

Figure 15: UART Asset Pattern

The matching process iteratively compared the asset patterns of GRL entries to the asset pattern of the UART design. As expected, the communication functionalities matched as closely as the unknown design. In fact, the top five closest matches have communication functionalities and matched with a value of 80% or greater. The GRL entry with the closest match is an interesting case study for understanding the matching process. All of the asset pattern characteristics dealing with external assets (input port signal, output port signal and internal

المنارة للاستشارات
www.manaraa.com

signal) received a 100% match. This occurs because of the specific assets and the functionality multiplier.

However, the internal signals have a variety of matching values. The internal asset pattern of the input signals matched at 75%, the internal asset pattern of the output signals matched at 0%, and the internal asset pattern of the internal signals matched at 78%. The 0% match for the internal asset pattern characteristic of the output signals illustrates an important point in understanding the matching algorithm and the weighting of the asset pattern characteristics. Although the remaining asset pattern characteristics have a high percentage match, the internal asset pattern of the output port signals has a 0% match because the two designs were coded differently in HDL. While the output signals of the basic UART entity assigns values to the primary outputs inside a process block, the design matched with the UART design uses concurrent statements and conditional statements to assign values to its outputs. Therefore, even though the functionality of the designs is clearly the same, the internal asset patterns are greatly different. As a result, the weighting of the asset pattern characteristics heavily favors the external asset pattern characteristics. After the weighting of the asset pattern characteristics is applied, the final match is 85%, which gives a fairly certain indication that the functionality of the design is the communication functionality as was expected.

4.3 Trojan Detection

Thorough testing of the Trojan detection strategies presented in the methodology section required designs with Trojans inserted. The Trust-Hub benchmarks [11] were used as a basis to perform unit testing of individual Trojan detection methods. These benchmarks focused on encryption units and communication units containing Trojans performing information leakage or denial of service attacks. In addition to the benchmarks created by Trust-Hub, several custom

المنارة للاستشارات
www.manaraa.com

benchmarks were developed in order to test the remaining Trojan detection methods that could not be tested with the examples provided by Trust-Hub. After verifying the functionality of each Trojan detection method through unit testing, large designs with multiple Trojans inserted were used to further test the Trojan detection methods and guarantee that all Trojans in the design could be detected. The first of the large designs was a crypto core developed by Trust-Hub with Trojans already inserted. The second was an open-source microcontroller that had several custom Trojans that were inserted into the previously Trojan-free design.

4.3.1 Trust-Hub Benchmarks

The Trust-Hub benchmarks provide a useful tool for the unit testing of Trojan detection methods. While not all of the Trust-Hub benchmarks are useful as they only contain gate-level netlists, a significant portion contain the RTL code required to test the Trojan detection methods of this methodology. Along with a Trojan-infested design, Trust-Hub includes a design without the Trojan included so that the two designs can be compared. Some of the benchmarks were written in Verilog rather than VHDL. By using the XHDL tool [22], the Verilog designs could be converted into VHDL in order to be useful for testing.

The first category of the Trust-Hub benchmarks consists of four RSA encryption units each containing a different type of Trojan. The first of these benchmarks, called *BasicRSA-T100*, leaks the encryption key when a specific plaintext string is entered. The Trojan-infested portion of the design can be seen in Figure 16 below.

```
if indata = x"44444444" then
        cypher <= key;          -- Trojan leaks the private key through cypher output bus
else
        cypher <= tempout;              -- set output value
end if;
```

Figure 16: Encryption Unit Key Leak VHDL Example

المنارة للاستشارات
www.manaraa.com

As the VHDL code shows, the key leaks when the string "44444444" is entered. Using traditional testing methods, this scenario is very difficult to discover, as it is a very rare condition. However, by performing asset assignment and filtering, the asset pattern of the ciphertext output is shown to contain a *KEY* asset. Therefore, the Trojan detection method designed to detect encryption unit leaks identifies the Trojan in this design.

This benchmark also serves as a unit test for another Trojan detection method. Since the Trojan is inserted into the design, the encryption key, which is assigned the external asset *KEY*, is directly driving the ciphertext, which is assigned the external asset *DATA_ENCRYPTION*. Therefore, the Trojan detection method ensuring that the filtered assets are contained within the tagged assets identifies that a suspicious connection occurs between the encryption key and the ciphertext. The ciphertext output's asset trace contains a *KEY* external asset due to filtering while the encryption key receives a *DATA_ENCRYPTION* asset. Therefore, both of the signals are noted to be vulnerable to attacks.

The next RSA Trojan benchmark, *BasicRSA-T200*, contains a denial of service attack against the encryption key. Similar to the previous Trojan, the attack is triggered by a specific input sequence of plaintext. The Trojan-infested portion of the design can be seen in Figure 17 below.

```
Trojan: process (indata) is
begin
        if indata = x"01fa0301" then
                trojanKey <= x"00000001";
        end if;
end process Trojan;
```

Figure 17: Encryption Unit Denial of Service VHDL Example

As the VHDL code shows, the signal *trojanKey* is assigned a value when a certain string of plaintext is inserted. This signal is then used as the encryption key for the *BasicRSA-T200*,

www.manaraa.com

while the original key is being denied. As in the previous example, it is nearly impossible to test the scenario in which the specific string of plaintext is inserted, resulting in the denial of service attack. However, the Trojan detection method identifying denial of service attacks detect this attack by identifying the internal asset pattern of the original encryption key. Since the original key signal is replaced by the *trojanKey* signal, its internal asset pattern is empty, indicating the presence of a denial of service attack against the encryption unit key.

Another RSA benchmark, *BasicRSA-T300*, has a similar payload to *BasicRSA-T100* in that it leaks the encryption key. However, it is triggered by a time bomb counter rather than a specific plaintext string. The time bomb counter process can be seen in Figure 18 below. Additionally, the payload causing the leakage of the encryption key can be seen in Figure 19 below.

```
TrojanTrigger: process (ds, reset) is
begin
        if reset='1' then
                TrojanCounter <= x"00000000";
        elsif rising_edge(ds) then
                TrojanCounter <= TrojanCounter + 1;
        end if;
end process TrojanTrigger;
```

Figure 18: Time Bomb Counter VHDL Example

```
if TrojanCounter = x"00000002" then
        cypher <= key;
else
        cypher <= tempout;
end if;
```

Figure 19: Encryption Unit Denial of Service VHDL Example

As the two portions of the design show, the time bomb counter will increment on the rising edge of the *ds* signal. After incrementing twice, it will leak the encryption key. Two separate Trojan detection methods identify this attack. First, the same Trojan detection method

www.manaraa.com

used to identify the Trojan in *BasicRSA-T100* detects the leak by identifying the asset pattern of the output. Second, the Trojan detection method designed to identify time bomb counters detects the attack by first identifying the fact that *TrojanCounter* is a counter. After identifying this fact, the detection method recognizes that it is contained within a conditional statement used to leak information. Therefore, the *TrojanCounter* signal is identified as a time bomb counter used as a trigger for information leakage.

The final RSA benchmark, *BasicRSA-T400*, contains the same trigger as the *BasicRSA-T300* benchmark as it uses a time bomb counter. Additionally, the payload of this Trojan is the same as *BasicRSA-T200* as it results in a denial of service attack. Therefore, the detection methods used for detection in the previous attacks are used to identify the Trojan in this benchmark.

The next set of Trust-Hub benchmarks consists of several AES encryption units containing various types of Trojans. These designs are much larger than the RSA examples and contain several Trojans for each individual example. One particular example in this set of benchmarks is a large crypto core, which is analyzed for numerous Trojans. However, other benchmarks in this category are analyzed for Trojans in order to test the detection methods. Many of the designs in this category contained the same type of Trojans, so not all designs in this category will be discussed.

The first Trojan attack found in this set of benchmarks consists of the encryption key being leaked to an instance that is not for encryption. By doing so, the encryption key is leaked through a shift register acting as a leakage circuit. This attack is found in *AES-T600* and *AES-T2000*. By identifying the instance inside the encryption unit is a shift register and the

المنارة للاستشارات
www.manaraa.com

encryption has been leaked to it, the circuit is marked as susceptible to an attack leaking the encryption unit key.

Another Trojan attack found in *AES-T600* and *AES-T2000* as well as *AES-T1800* implement a Trojan intended to drain the battery of the circuit by adding unnecessary circuitry. The shift register in the previous discussion and in *AES-T1800* is the instance used as extra circuitry. This is detected by noting that there are no outputs found in the instance, as it is being used only to perform unnecessary computations designed to drain the battery of the circuit. The Trojan detection method utilizing functionality matching also is used to recognize this Trojan, as it detects shift registers being contained within encryption units. Therefore, multiple detection methods are used to identify the same Trojan.

The final major category of Trust-Hub benchmarks consists of the RS232 UART communication units. The designs consist of a top-level entity with a transmitter instance and a receiver instance. Several Trojans implemented in this category are identified using previously discussed Trojan detection methods. However, there are a few new Trojans within this category that are detected with additional Trojan detection methods.

Several designs in the RS232 category—namely *RS232-T600*, *RS232-T700*, and *RS232-T900*—contain malicious FSMs used to activate a Trojan. After a certain sequence is inputted to the communication unit, it reaches a state that causes the output information to be modified. The Trojan detection methods pertaining to FSMs are utilized to detect Trojans in these designs. As the methodology section mentioned, the Trojan FSM detection methods indicate the possibility of a Trojan and require user interaction for verification. The fact that an input sequence is used to trigger a specific state could potentially be legitimate. However, in this particular case, once

www.manaraa.com

the user has been notified of a potential Trojan present, it can be easily verified that the FSM is being used as a Trojan.

An additional method of Trojan detection utilized in this category of benchmarks is the matching of blacklisted designs. Due to the fact that these designs use an FSM to implement Trojan triggers, the internal asset patterns of the Trojan-infested design are significantly different than the internal asset patterns of the Trojan-free version. Therefore, the GRL matching algorithm matches the unknown design to a blacklisted functionality when analyzing other designs with a similar Trojan trigger.

4.3.2 Additional Trojan Examples

After performing unit tests with the Trust-Hub benchmarks for the relevant Trojan detection methods, the remaining Trojan detection methods were tested by producing designs containing the corresponding Trojans. The first of the remaining Trojan detection methods tested is the Trojan resetting the timing signal of the design. This is a simple example to test, as an internal clock signal assignment was modified to include a reset signal. A sample assignment statement can be seen below.

Internal_Clock <= Clock AND Reset;

In this scenario, the internal clock is clearly reset rather than being directly driven by the clock port signal. Therefore, anything driven by the internal clock will experience a denial of service attack. This attack is detected by analyzing the external asset trace of the timing signals of the design. In this scenario, the asset trace of the timing signals also contains a filtered *RESET* asset. Therefore, the timing signal with a *RESET* in its asset trace is noted as being vulnerable to a Trojan. In addition to the clock-reset Trojan detection method, another Trojan detection method can be used to identify this attack. This Trojan detection method deals more generally with

www.manaraa.com

denial of service attacks to timing signals by analyzing the internal asset trace of the timing signals. The denial of service attacks are detected by identifying internal assets indicating that the signal has been modified, such as *CONCURRENT_DRIVEN*, *CONDITIONAL_DRIVEN* and *PROCESS_OPERATION_SENSITIVE*. In this example, the timing signal's internal asset trace contains a *CONCURRENT_DRIVEN* asset due to the concurrent assignment statement. Therefore, the signal is also flagged as being vulnerable to a timing denial of service attack.

The Trojans related to suspicious entities contained within register files were tested by creating connections to malicious instances in a previously secure register file. The first type of Trojan tested is a counter instance inside a register file. After creating a connection in the register file, the design was run through the tool, resulting in correct functionality matches for the register file and the counter. Following the matching of the functionalities, the Trojan detection method identifies that a *TIMING* instance is found in a *REGISTER_FILE* top-level entity. Therefore, the *TIMING* instance is correctly flagged as a Trojan. Next, a malicious shift register was inserted in the register file. Following the same process, both entities are correctly matched using the GRL. The Trojan detection method consequently flags the shift register as a suspicious connection, correctly identifying the inserted Trojan.

The next set of Trojan detection methods tested involved attacks on interrupt units. The first of these attacks involved an interrupt signal being leaked to another component of the design, resulting in a number of possible failures in the circuit. The unit test that was developed to test this detection method involved a counter instance being found within an interrupt unit. The port mapping of this instance can be seen in Figure 20 below.

المنارة للاستشارات
www.manaraa.com

```
trojanCounter: counter
Port Map(C => CLK_I,
         CLR => IR(0),
         Q => tjCounter);
```

Figure 20: Counter Instance in Interrupt Unit

When a specific interrupt signal arrives to the interrupt unit, it triggers the counter, eventually resulting in a denial of service attack against the interrupt unit. However, this same detection method is valid for detecting other types of attacks to the design. The detection method identifies an interrupt was sent to a *TIMING* instance and therefore flags the signal that is sent to the instance.

Another Trojan detection method deals more generally with denial of service attacks against interrupt units. In the unit test for this method, the input interrupt signal is being denied as a result of a Trojan signal being used in its place. The Trojan signal is preset to a value, as the line below shows.

```
signal trojanInterrupt : std_logic := '0';
```

The *trojanInterrupt* signal is then used in place of the intended interrupt signal. A portion of the VHDL code can be seen below in Figure 21.

```
if( trojanInterrupt = '1' ) then
        int_pt <= pt(0);
        INTR_O <= '1';
        next_s <= tx_int_info_priority;
elsif( trojanInterrupt = '1' ) then
        int_pt <= pt(1);
        INTR_O <= '1';
        next_s <= tx_int_info_priority;
end if;
```

Figure 21: Interrupt Unit Denial of Service Attack

As this example shows, the value of the interrupt signal is preset and not modified, resulting in a denial of service attack. This attack is detected by analyzing the internal asset

المنارة للاستشارات
www.manaraa.com

pattern of the denied interrupt signal and identifying as not being used in the design. Therefore, the interrupt signal is flagged as being denied, meaning that another signal has taken its place.

The next Trojan detection method tested involved the identification of Trojan triggers in the form of concurrent assignment statements. These statements are suspicious as they are involved in activating a signal under a rare circumstance. Because of the limitations of testing, these rare conditions are often untested, ultimately resulting in the activation of the Trojan by an attacker. A portion of the example designed to test the Trojan detection method can be seen below in Figure 22.

```
b <= A(3) AND A(2) AND A(1) AND A(0);
if (b='1') then
        out_data <= '0';
else
        out_data <= in_data;
end if;
```

Figure 22: Trigger Assignment Attack

As this portion of the code shows, the signal *b* is triggered only when the *A* vector reaches its terminal value. As a result, *b* goes high and the *out_data* is set to ‘0’. In this scenario, trigger assignments have the ability to leak information through outputs or cause denial of service attacks. Therefore, the assignment of *b* is marked as malicious and presented to the user.

The next Trojan detection method to be tested involved the leakage of any signals assigned a *CRITICAL* output. Users assign *CRITICAL* assets to signals that should be kept secure, such as an encryption key or other sensitive data. Therefore, this Trojan detection method was tested by assigning a *CRITICAL* asset to an encryption key that was leaked through an output. Many of the Trust-Hub designs implement this type of Trojan and were therefore used to test this method. The method detects the information leak by analyzing the asset pattern

المنارة للاستشارات
www.manaraa.com

of the output signals. Since the signal with the *CRITICAL* asset directly drives an output signal, the output signal contains a filtered *CRITICAL* asset. Therefore, the output signal is marked as being susceptible to a leakage of critical information.

4.3.3 Trojan-Infested Crypto Core Example

In order to demonstrate how the Trojan detection methods extend to larger designs, multiple larger designs were analyzed for Trojans. The first large design example involved the analysis of a crypto core found in the Trust-Hub benchmarks. Specifically, the design is an AES core called *AES-T600* in the Trust-Hub benchmark. This design contained multiple Trojans intended to leak the encryption key. Additional Trojans were also added to the design in order to demonstrate the abilities of multiple Trojan detection algorithms. There were a total of 11 lower level entities in the design. Although typically negligible, the filtering time for this design took significantly longer as it was a larger design using many rounds of encryption. The total parse time took 6 seconds, the total filtering time took 5 minutes and 41 seconds and the Trojan detection time was negligible. This process was performed on an Apple MacBook Pro with a 2.4 GHz Intel Core 2 Duo processor and 16 GB of RAM.

The first step in analyzing the design was to assign external assets to the primary port signals. The port signals for the design can be seen in Figure 23. The asset assignment for the listed port signals was fairly self-explanatory. The *clk* signal was assigned a *SYSTEM_TIMING* external asset, the *rst* signal was assigned a *RESET* asset, the *in* signal was assigned a *DATA_ENCRYPTION* asset, the *key* signal was assigned a *KEY* asset and the *out* signal was assigned a *DATA_ENCRYPTION* asset.

المنارة للاستشارات
www.manaraa.com

```
ENTITY top IS
  PORT (
    clk      : IN STD_LOGIC;
    rst      : IN STD_LOGIC;
    in       : IN STD_LOGIC_VECTOR(127 DOWNTO 0);
    key      : IN STD_LOGIC_VECTOR(127 DOWNTO 0);
    out      : OUT STD_LOGIC_VECTOR(127 DOWNTO 0)
  );
END ENTITY top;
```

Figure 23: Crypto Core Port Signals

Following asset assignment, the assets of the design were filtered throughout the circuit in order to obtain an asset pattern for the design. The asset pattern was then used in the functionality matching for the individual entities of the design. The functionality of the top-level design was correctly matched to an encryption unit. Therefore, the multiple encryption unit Trojan detection algorithms were used to identify Trojans.

The list of Trojans that were previously inserted into the design began with a time bomb counter inserted in the final round of encryption. The trigger for this attack can be seen in Figure 24 below. As the figure shows, the Trojan counter increments on the rising edge of the clock. The counter is used to trigger an information leakage in the final round of encryption. The payload of the attack is a leakage of the encryption key through the ciphertext output. Therefore, by knowing the point at which the Trojan counter causes leakage, an attacker can observe the output of the encryption unit and obtain the key. The information leakage portion of the VHDL code can be seen in Figure 25 below.

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

Figure 24: AES Time Bomb Trigger

المنارة للاستشارات
www.manaraa.com

```
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;
```

Figure 25: AES Time Bomb Key Leakage

As the portion of code shows, the encryption key is leaked through the output once the counter reaches the value of "44444444". This attack is detected by first applying the algorithm for identifying time bomb counters. By identifying a counter was used in conjunction with a leakage of information, the *TrojanCounter* signal is flagged as a trigger for an information leakage attack while the *state_out* signal is flagged as the output port signal used to leak the information. Additionally, this attack is also identified by applying the algorithm of detecting encryption keys leaking directly to an output. By identifying the asset traces of both the *state_out* signal and the *key_in* signal, the Trojan attack is detected by recognizing the direct driving assignment between these two signals and that an encryption key could directly leak through an output port signal.

The next attack found in the AES core involved a Trojan shift register. The concept of a Trojan shift register was previously addressed when discussing the Trust-Hub benchmarks. The Trojan shift register is used as additional logic performing a shift operation on the encryption key. In doing so, it allows attackers to perform power analysis side-channel attack on this portion of the circuit and thus obtain the value of the encryption key. It is initially triggered by a time bomb counter incrementing up to a certain value before performing the attack. This attack is detected in multiple ways. First, since the entity does not contain any outputs, it is immediately flagged as an inclusion of additional logic. Second, the attack is identified as a

المنارة للاستشارات
www.manaraa.com

Trojan shift register through the functionality matching. Finally, the trigger for the attack is identified as a time bomb counter through the time bomb counter Trojan detection method. Through the combination of all these Trojan detection methods, the evidence indicates that the entity in question is malicious.

4.3.4 Trojan-Infested Microcontroller

The final large design intended to test Trojan detection methods involves a microcontroller with Trojans inserted. This example is different than the previous example in that the Trojans were not inserted by Trust-Hub or another third party. Instead, the Trojan-free version of this design was obtained through OpenCores [13] then the Trojans were personally inserted in various positions throughout the design. This was intended to show the capability of the Trojan detection methods of finding Trojans in a large design. The design chosen from OpenCores to be evaluated was termed *c16*, which is an open-source 16-bit microcontroller. The design has 19 lower level entities, such as a register file, an ALU, and a communication unit. This design was chosen due to the fact that it was already written in VHDL and required minimal modifications to be parsed by the tool. The total parse time for the design was 18 seconds, the filtering time was 11 seconds and the Trojan detection time was negligible.

Following the parsing process of the design, assets were assigned to the primary port signals. Since the design was significantly larger than the unit tests, it was imperative to assign correct assets to the signals in order to adequately represent the functionality of the signal. This is needed because the signal is filtered throughout a much larger design; and, if the signal is assigned to an incorrect asset then that asset propagates throughout a much larger space than if the design was smaller. The port signals of the microcontroller can be seen in Figure 26 below.

المنارة للاستشارات
www.manaraa.com

Additionally, the asset assignment of the primary port signals from Figure 26 can be seen in Table 16 below.

```
entity board_cpu is
PORT (CLK40              : in  STD_LOGIC;
      SWITCH             : in  STD_LOGIC_VECTOR (9 downto 0);
      SER_IN             : in  STD_LOGIC;
      SER_OUT            : out STD_LOGIC;
      TEMP_SPO           : in  STD_LOGIC;
      TEMP_SPI           : out STD_LOGIC;
      CLK_OUT            : out STD_LOGIC;
      LED                : out STD_LOGIC_VECTOR (7 downto 0);
      ENABLE_N           : out STD_LOGIC;
      DEACTIVATE_N       : out STD_LOGIC;
      TEMP_CE            : out STD_LOGIC;
      TEMP_SCLK          : out STD_LOGIC;
      SEG1               : out STD_LOGIC_VECTOR (7 downto 0);
      SEG2               : out STD_LOGIC_VECTOR (7 downto 0);
      XM_ADR             : out   STD_LOGIC_VECTOR(15 downto 0);
      XM_CE_N            : out STD_LOGIC;
      XM_OE_N            : out STD_LOGIC;
      XM_WE_N            : in STD_LOGIC;
      XM_DIO             : in STD_LOGIC_VECTOR(7 downto 0)
      );
end board_cpu;
```

Figure 26: Microcontroller Primary Port Signals

TABLE 16: Microcontroller Asset Assignment

Signal	Asset
CLK40	*SYSTEM_TIMING*
SWITCH	*DATA_PERIPHERAL*
SER_IN	*DATA_COMMUNICATION*
SER_OUT	*DATA_COMMUNICATION*
TEMP_SPO	*DATA_PERIPHERAL*
TEMP_SPI	*DATA_PERIPHERAL*
CLK_OUT	*SYSTEM_TIMING*
LED	*DATA_PERIPHERAL*
ENABLE_N	*ENABLE*
DEACTIVATE_N	*STATUS*
TEMP_CE	*PERIPHERAL_CONTROL*
TEMP_SCLK	*SUBSYSTEM_TIMING*
SEG1	*DATA_PERIPHERAL*
SEG2	*DATA_PERIPHERAL*

المنارة للاستشارات
www.manaraa.com

TABLE 16: Microcontroller Asset Assignment (Cont.)

Signal	Asset
XM_ADR	*ADDRESS_SENSITIVE*
XM_CE_N	*ENABLE*
XM_OE_N	*ENABLE*
XM_WE_N	*COMMUNICATION_CONTROL*
XM_DIO	*DATA_COMMUNICATION*

The first of the Trojans inserted in the design is a denial of service attack found in the ALU. The attack targets the ALU instruction signal used to determine which operation is to be performed. By controlling the instruction signal, an attacker has the ability to control which operation is to be performed by the ALU, therefore rendering it useless. The Trojan was inserted as an additional signal, called *ALU_OP_Trojan*, that could be pre-programmed by an attacker while the original ALU instruction signal was denied. A portion of the FSM controlled by the Trojan signal can be seen in Figure 27 below.

```
case ALU_OP_Trojan is
        when     ALU_MD_FIN =>      -- mult/div
          if (QP_NEG = '0') then
                    ZZ <= PROD_REM(15 downto 0);
          else
                    ZZ <= X"0000" - PROD_REM(15 downto 0);
          end if;

        when     others =>      -- modulo
          if (RM_NEG = '0') then
                    ZZ <= PROD_REM(31 downto 16);
          else
                    ZZ <= X"0000" - PROD_REM(31 downto 16);
          end if;
end case;
```

Figure 27: ALU Denial of Service

With the Trojan signal controlling the FSM, the outputs from the ALU are untrusted. This attack was detecting by identifying that the original ALU instruction signal, called

www.manaraa.com

ALU_OP, was being denied since the design was being controlled by a Trojan instruction signal. Therefore, the original ALU instruction signal was flagged as a part of a denial of service attack.

The next attack involved the insertion of a malicious state to an FSM contained within the memory component of the design. The Trojan-free FSM typically contains an "OTHERS" state that handles remaining states that are not listed. The Trojan-free version of the FSM can be seen in Figure 28 below.

```
case LADR is
      when "0001" =>          RDAT <= OUT_1;
      when others =>          RDAT <= OUT_0;
end case;
```

Figure 28: Trojan-free Memory FSM

As the example shows, when the *LADR* signal has a value of "0001", the *OUT_1* signal is assigned to the *RDAT* output signal. When the *LADR* has any other value, the *OUT_0* signal is assigned to the *RDAT* output signal. However, the inserted Trojan results in the modification of the default value for the FSM. The Trojan-infested version of the FSM can be seen in Figure 29 below.

```
case LADR is
      when "0001" =>          RDAT <= OUT_1;
      when "0010" =>          RDAT <= trojanOut;
end case;
```

Figure 29: Trojan-infested Memory FSM

As the Trojan-infested version shows, the "OTHERS" case has been removed and replaced with a state occurring when the *LADR* value is "0010". When this state is reached, an incorrect value is assigned to the *RDAT* signal. Additionally, any case other than the two listed will not result in a change in the value of the *RDAT* output. Therefore, this attack has modified the output data that is assigned to the *RDAT* signal. This attack was detected using the FSM

المنارة للاستشارات
www.manaraa.com

Trojan detection methods. First, because the *case statement* does not contain an "OTHERS" state and not all states are listed, it is flagged as susceptible to the insertion of a Trojan as a malicious state. Additionally, it is marked as susceptible to a gate-level Trojan inserted since not all states are accounted for. Therefore, this attack was detected by identifying the missing states in the FSM.

The next attack was inserted inside the UART communication unit resulting in a transmission of incorrect data. The attack was triggered by a counter signal that counted on the rising edge of the clock. The transmission of serial data was disrupted when the Trojan counter reached a certain value. The Trojan payload occurred when the most significant bit of the Trojan counter is equal to zero. The VHDL code can be seen in Figure 30 below.

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

Figure 30: UART Trojan Attack

This scenario only occurs when the most significant bit of the Trojan counter goes high, and therefore it takes a significant amount of time to occur. This scenario is also very time consuming to test, thus it is rarely caught through functional testing. However, the time bomb counter Trojan detection method identifies the inclusion of the time bomb counter and subsequently identifies any signals triggered by the counter signal. Therefore, both the time bomb counter signal and the serial out signal are flagged as a part of a Trojan attack.

The final Trojan inserted into the microcontroller is another instance of additional logic in the form of a shift register. The shift register was similar to the one presented in the previous example of the crypto core. Even though this example was previously tested, it was important to

المنارة للاستشارات
www.manaraa.com

test it in a larger design in order to guarantee the identification of the malicious logic when there were significantly more entities to analyze. The shift register was inserted in the data core entity of the design and was intended to leak data from memory to an attacker. The port mapping of the Trojan shift register can be seen in Figure 31 below.

```
--Trojan Shift Register
ShiftRegister: TSC
Port Map(clk => CLK_I,
         rst => CLR,
         data_leak => RDAT,
         Tj_Trig => tjTrigger);
```

Figure 31: Trojan Shift Register Port Map

As the VHDL code shows, the *RDAT* signal of the data core was leaked to the shift register and therefore leaked to an attacker. The Trojan shift register's internal characteristics are the same as the previous example's shift register's characteristics. The shift register entity does not contain an output and was used as additional logic designed to leak the input information. The Trojan shift register was detected using the same algorithms as before. It was analyzed and found to contain no outputs, and therefore was marked as additional logic that could be utilized by a malicious attacker to leak information.

4.4 Analysis

As the previous sections noted, the Golden Reference Library matching as well as the Trojan detection methods produced successful results. The matching methodology was subjected to numerous rounds of testing to determine its ability to match an unknown design to a functionality successfully. Before specific assets were developed, the matching results were occasionally incorrect when dealing with general assets. However, after incorporating the specific assets, the matching methodology was successful in matching an unknown design to a functionality. Additionally, the output report of the percentage match gives the user an

المنارة للاستشارات
www.manaraa.com

indication of the other designs that matched closely in the case of an outlier as well as presenting the user with the average match of a functionality. These provide the user with additional information that can assist in the evaluation of an unknown design. Overall, the matching methodology is very successful in evaluating unknown designs due to the specificity of asset assignment and the large GRL used in comparison.

Additionally, the Trojan detection portion of the project successfully analyzes unknown designs for potential Trojans. As the previous sections have described, the Trust-Hub benchmarks were the major source of verification for the Trojan detection methods. After implementing Trojan detection methods for all the Trojans found in the benchmarks, additional test vehicles were developed to illustrate the effectiveness of the remaining Trojan detection methods. Finally, the Trojan detection methods were tested using large designs to ensure that Trojans could still be identified. During all of the testing phases, the Trojan detection methods successfully identified the entire set of Trojans. Therefore, the false negative rate for the Trojan detection methods was 0%.

A limitation of the matching methodology involves matching a large design with many level of hierarchy. Large designs incorporate internal signals mapped to lower level instances and the internal signals do not receive external assets during the asset assignment phase. Therefore, as the internal signals are connected to lower levels of the hierarchy, the asset pattern of the lower level entities includes the assets of signals that were not assigned an external asset. Additionally, the asset assignment at the top level of a design often requires more general assets since the signal at the top level is mapped to multiple lower level entities. The result of these issues is that the functionality assignment of the lowest level entities is not as accurate as entities higher in the hierarchy of the design.

المنارة للاستشارات
www.manaraa.com

Although the Trojan detection methodology accurately identifies the Trojans with a false negative rate of zero, the Trojan detection methods consist of a non-zero false positive rate. The ultimate goal of the Trojan detection methods was to produce a false negative rate of zero and in that respect the Trojan detection methods were successful. In order to allow a false negative rate of zero, the false positive rate was non-zero. However, the false positive rate is not a significant hindrance in determining the location of Trojans. Additionally, the false positive rate for the Trojan detection varied among the individual Trojan detection methods. The overall final false positive rate for all Trojan detection methods is 4.4%.

Of the set of Trojan detection method false positives, certain false positives were to be expected and are due to the nature of the Trojan detection methods. First of all, the limitations of the parser cause occasional false positives to occur in Trojan detection methods involving the use of internal assets. Certain VHDL syntax cannot be recognized by the parser in order to assign appropriate internal assets to the signal. For example, the use of the "when" keyword in a concurrent statement cannot be recognized by the parser. An example of this type of VHDL statement can be seen in Figure 32 below.

```
WR_0  <= '1' when (WR = '1' and ADR(15 downto 12) = "0000"  ) else '0';
WR_1  <= '1' when (WR = '1' and ADR(15 downto 12) = "0001"  ) else '0';
```

Figure 32: VHDL Concurrent Statements

In the statement above, the *WR* signal ideally should receive a conditionally driving asset since it is used to determine the outputs *WR_0* and *WR_1*. However, the parser does not recognize signals found in the condition of the "when" statement resulting in no internal assets assigned to the *WR* signal. Therefore, detection methods that involve analyzing the internal assets of a signal, such as the denial of service methods, are susceptible to attacks involving the "when" statements that are not identified by the parser. Since the priority of the Trojan detection

المنارة للاستشارات
www.manaraa.com

process was to ensure a zero false negative rate, the Trojan detection methods assumed a worst-case scenario. The result is that the false negative rate is indeed zero, but the false positive rate incurs a penalty.

Additionally, some Trojan detection methods incur a larger false positive rate because of how the detection method is implemented. Some of the Trojan detection methods are treated as warnings against possible malicious intentions by attackers rather than certainties of a Trojan insertion. For example, one of the FSM Trojan detection methods identifies FSMs that are vulnerable to Trojan insertion at the gate level. Even if no attack occurs at the gate level, this Trojan detection method identifies the potential for a Trojan to be inserted. The driving objective behind these Trojan detection methods, as has been stated frequently, is identifying all Trojans in the design. Therefore, some of the Trojan detection methods, such as the FSM method mentioned, present the user with the possibility that a Trojan could have been inserted. The Trojan detection report allows the user to quickly identify the locations of the potential Trojans and determine the validity of the detection method.

Determining the false positive rate of the Trojan detection methods requires finding all false positives within the sample space then dividing this number by the sum of the false positives and the true negatives. The true negatives are the cases when the Trojan detection methods correctly identified the absence of a Trojan. There was a range of false positive rates found among the Trojan detection methods. Some Trojan detection methods received a 0% false positive rate as no false positives had been detected. The highest false positive rate found among the Trojan detection method was the time bomb counter detection method, which received a 20% false positive rate. While this number is high, the Trojan detection method was designed to identify all Trojans, necessitating the presence of false positives. Additionally, this Trojan

www.manaraa.com

detection method, along with other detection methods that have a high false positive rate, would require analyzing the intention for a signal, which is very difficult to perform. Consequently, the false positive rates for these types of Trojans are higher than others that are more distinctly malicious. Therefore, for the Trojans with higher false positive rates, the user is required to analyze the design further to determine whether a Trojan has actually been inserted. When all of the false positive rates were averaged together, the previously presented value of 4.4% was determined as the overall false positive rate.

المنارة للاستشارات
www.manaraa.com

5. CONCLUSION

5.1 Summary

The objectives presented at the outset were successfully attained throughout the course of the project. The number of assets was significantly increased in order to provide greater diversity to asset patterns. Furthermore, a Golden Reference Library and functionality matching methodology were created, allowing an unknown design to be analyzed and matched to a known design. Finally, hardware Trojan detection methods were introduced to the *Structural Checking* methodology by utilizing the Golden Reference Library matching as well as analyzing internal characteristics of a hardware design. Each of these key objectives was implemented and thoroughly tested.

This project provides significant progress in the area of hardware Trojan detection, specifically with regard to the *Structural Checking* methodology. Within the larger realm of hardware Trojan detection, this methodology detects very small hardware Trojans, a limitation of many other hardware Trojan detection methods. Additionally, by using this methodology, hardware Trojans are detected pre-fabrication, saving IC design companies the significant cost of testing for hardware Trojans. With respect to the *Structural Checking* methodology, this project has increased the capabilities of asset assignment as well as introduced the implementation of the completely new features of a Golden Reference Library and hardware Trojan detection.

5.2 Future Work

Several steps can be taken in order to further improve the *Structural Checking* methodology. First, as more trusted designs are analyzed, the asset patterns associated with these designs can be added to the GRL. This allows for the matching methodology to be more robust as more designs are added. Additionally, as more Trojan attacks are discovered,

المنارة للاستشارات
www.manaraa.com

corresponding Trojan detection methods can be developed to identify these attacks. Additionally, the Trojan detection methods currently being used can be modified to further reduce the false positive rate associated with each method. Finally, additional steps can be taken to better match lower level entities of large designs to a functionality.

المنارة للاستشارات

www.manaraa.com

REFERENCES

[1] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," Design & Test, IEEE, vol. PP, pp. 1-1, 2013.

[2] 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.

[3] 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.

[4] Xuehui 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, 2011, pp. 1-6.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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. 169177, 2012.

[11] https://www.trust-hub.org/resources/benchmarks.

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

المنارة للاستشارات
www.manaraa.com

[13] http://www.opencores.org.

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

[15] Xiaoxiao Wang, M. Tehranipoor and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on, Anaheim, CA, 2008, pp. 15-19.

[16] J. Di and S. Smith, "A Hardware Threat Modeling Concept for Trustable Integrated Circuits," Region 5 Technical Conference, 2007 IEEE, Fayetteville, AR, 2007, pp. 354-357.

[17] 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.

[18] Jie 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.

[19] S. C. Smith and J. Di, "Detecting Malicious Logic Through Structural Checking," Region 5 Technical Conference, 2007 IEEE, Fayetteville, AR, 2007, pp. 217-222.

[20] 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.

[21] 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, 2008. DATE '08, Munich, 2008, pp. 1362-1365.

[22] http://www.x-tekcorp.com/xhdl.html

المنارة للاستشارات
www.manaraa.com