5-2019

Hardware IP Classification through Weighted Characteristics

Brendan McGeehan
University of Arkansas, Fayetteville

Hardware IP Classification through Weighted Characteristics

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

by

Brendan McGeehan
University of Arkansas
Bachelor of Science in Computer Engineering, 2017

May 2019
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____
Jia Di, Ph.D.
Thesis Director

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

ABSTRACT

Today's business model for hardware designs frequently incorporates third-party Intellectual Property (IP) due to the many benefits it can bring to a company. For instance, outsourcing certain components of an overall design can reduce time-to-market by allowing each party to specialize and perfect a specific part of the overall design. However, allowing third-party involvement also increases the possibility of malicious attacks, such as hardware Trojan insertion. Trojan insertion is a particularly dangerous security threat because testing the functionality of an IP can often leave the Trojan undetected. Therefore, this thesis work provides an improvement on a Trojan detection method known as Structural Checking which analyzes Register-Transfer Level (RTL) and gate-level soft IPs. Given an unknown IP, the Structural Checking tool will break down the design primary ports and internal signals into assets that fall into six characteristics. These characteristics organize how the IP is structured and provide information about the unknown IP's overall function. The tool also provides a library of known designs referred to as the Golden Reference Library (GRL). All entries in the library are also broken down into the same six characteristics and are either known to be clean or known to have a Trojan inserted. An overall percent match for each library entry against the unknown IP is calculated by first computing a percent match within each characteristic. A weighted average of these percent matches makes up the final percentage. If the library entry with the best match is known to have a Trojan inserted, then the unknown design is likely to have a Trojan as well and vice versa. Due to the structural variability of soft IP designs, it is vital to provide the best possible weighting of the six characteristics to best match the unknown IP to the most similar library entry. This thesis work provides a statistical approach to finding the best weights to optimize the Structural Checking tool's matching algorithm.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Jia Di. He has provided me with great support and guidance throughout my college career as both an undergraduate and graduate student. He is a perfect example of a great mentor, and I'm glad I got the opportunity to work with him.

I would also like to thank Dr. Thao Le for her help and support during my graduate career.

DEDICATION

To my Mom, Dad, and brothers for supporting me throughout my college career. They have always given me the support I needed to be successful.

To my girlfriend, Kristen Tilley, for all her love and support. She provided me with great advice to help me achieve this degree.

CONTENTS

GLOSSARY

Abbreviations

RTL	Register Transfer Level
IPs	Intellectual Properties
Soft IPs	Intellectual Properties under RTL or gate-level
GR	Golden Reference
GRL	Golden Reference Library
SC	Structural Checking
HPM	Highest percentage matching
EEPROM	Electrically Erasable Programmable Read-Only Memory
RAM	Random Access Memory
VHDL	VHSIC Hardware Description Language
ALU	Arithmetic Logic Unit

1 INTRODUCTION

Due to the growing number of third-party hardware IP vendors world-wide, the importance of securing hardware designs has grown significantly. By outsourcing components of a hardware design to other parties, the integrity of the overall design can be compromised. One example of how a design can be compromised is through the insertion of a hardware Trojan into a third-party component. Trojan-infested components often work as intended to hide the Trojan inserted. Consequently, hardware Trojans are difficult to detect and can lead to very damaging payloads. Some of these effects include leaking a secret key or shutting down a part of the hardware during operation. Any compromised design can then end up in applications where security is vital, such as defense applications. As a result, developing a method for hardware Trojan detection is very important to guarantee the integrity of all hardware.

A significant area of research for hardware Trojan detection comes from side-channel analysis. Side-channel analysis takes advantage of naturally occurring emissions of a circuit, such as power and timing delays, to detect modifications to the circuit. Power analysis can be effective when the Trojan infested circuit emits significantly different power readings compared to the same circuit that is known to be clean. A drawback from this approach is that Trojans can be very small and thus do not produce a significant power consumption to raise concerns about the integrity of the circuit. Introduced in [1], different circuits were first partitioned and then power analysis was performed. By observing the circuit in smaller portions, the difference in power readings is more significant and leads to better detection of even small Trojans. Path delay in [2], or timing analysis, seeks to find significant differences in how long a signal takes to travel through a specific path within the circuit. However, similar to power analysis, the Trojan inserted may not be very large which means that path delays alone may not be enough to indicate

an inserted Trojan. While power and timing analysis are both valid methods of Trojan detection, they both focus on detecting Trojans on hard IPs, or already manufactured chips. Even with correct detection at the hard IP level, the infested chips become unreliable and there is a need for a trusted hardware.

Hardware Trojan detection methods at the soft IP level are also under research. Soft IPs include Register-Transfer Level (RTL) code and gate-level netlists. In [3], hardware Trojans are detected in gate-level netlists using a Random Forrest Classifier. This machine learning approach takes advantage of features that are commonly known in Trojan nets to then classify unknown nets. Another machine learning technique used to detect Trojans from netlists is through a support vector machine classifier [4]. The research conducted in [4] breaks down each net into 5 characteristics and classifies each net as either Trojan infested or Trojan free based on the knowledge of known Trojan free and infested nets. In addition to these approaches, there is research of using Golden Reference Matching for Trojan detection as done in [5]. Golden Reference Matching breaks down RTL code by labeling primary ports and internal signals with assets, the signals' contribution to the overall design. Upon completion of assigning assets, an unknown IP is compared against the Golden Reference Library, a collection of soft IPs that are known to be either be clean or Trojan-infested. If the unknown IP matches best with a Trojan-infested library entry then the unknown IP is likely to contain a Trojan, and vice versa. Building on the work of [5], [6] introduces a way to use Golden Reference Matching with gate-level netlists.

The rest of the thesis is organized in the following manner. Chapter 2 will cover background information on assets, Structural Checking, and the process of Golden Reference Matching with a Golden Reference Library. Chapter 3 will cover the design and implementation

of a statistical based improvements on the Golden Reference Matching algorithm. Chapter 4 provides example soft IPs to prove the effectiveness of the improved algorithm. Chapter 5 will then conclude the thesis and provide details on future work.

2 BACKGROUND

2.1 Assets

2.1.1 Overview

A key component from the Structural Checking tool with Golden Reference Matching is the concept of assets. Assets provide a description to primary ports and internal signals of soft IPs. More specifically an asset provides a label to a signal about its purpose/function to the overall design. For instance, a clock signal would be assigned what is known as a SYSTEM_TIMING asset because a clock provides timing for the overall circuit. Each signal can have multiple assets assigned to it to refine how it fits in the overall design. There are two main categories of assets defined in the Structural Checking tool, internal assets and external assets.

2.1.2 Internal Assets

Internal assets are intended to describe the function of internal signals in a soft IP, but they can also be used for primary port signals. Most internal assets used in the tool were developed in [7] and [8]. The research conducted in [8] added three internal assets specifically for a scan-chain structure (OBSERVABLE, CONTROLLABLE, and PROTECTED). These three assets differ from the rest of the internal assets because they require to be manually assigned to signals. Most internal signals are assigned automatically as the Structural Checking tool parses the Register-Transfer Level (RTL) code. Some examples of internal assets include PROCESS_SENSITIVE and CONDITIONAL_DRIVEN. The PROCESS_SENSITIVE asset describes a signal that is included in the sensitivity list of process from RTL code. CONDITINAL_DRIVEN describes a signal that is within an “if/case” block in the RTL code because its value depends on a certain condition to be met.

2.1.3 External Assets

External assets are used to describe the function/purpose of primary ports in soft IPs. Unlike internal assets, all external assets must be manually assigned to each primary port signal through the use of the Structural Checking tool. These assets are broken up into 5 main categories: *Data*, *Timing*, *System Control, Specific System Control,* and *Miscellaneous.* An example from the *Data* category includes DATA_MEMORY which is assigned to signals that are intended to store data for any type of memory. COUNT is an example from the *Timing* category and this asset is assigned to signals that keep track of a count value for the IP. An example from *System Control* includes HANDSHAKING which is assigned to signals that will handle any type of handshaking operations for the IP. An example of *System Specific Control* includes COMMUNICATION_CONTROL asset and it is assigned to a signal that controls transmission with another component (such as a UART module). Finally, an example from the *Miscellaneous* category includes ADDRESS_SENSITIVE and is assigned to signals that will store any type of address for the IP, such as a memory address. From the work done in [5] and [7] the Structural Checking tool currently has 58 external assets available for assignment to primary ports.

2.1.4 Asset Filtering

The idea of asset filtering is to allow assets assigned on any signal of an IP to propagate through connected signals. By propagating assets, the tool finds correlations between signals and potentially finds signals that have conflicting assets. This allows the Structural Checking tool to raise a flag about a potential Trojan in the circuit. Asset filtering was added to the Structural Checking tool in [9]. External assets assigned to primary inputs propagate to all signals that complete the path from the primary input to the dependent primary output.

Likewise, external assets assigned to primary outputs propagate backwards through connected signals to their dependent primary input. For internal assets there are a few exceptions to this type of filtering process. When filtering a process sensitive asset, the propagation only traverses to signals that are connected to the original signal and are contained within the same process block. Another exception to traditional filtering comes from conditional assets (CONDITIONAL_DRIVEN and CONDITIONAL_DRIVING). Similar to the process sensitive asset, these assets only propagate within their conditional statements. All other internal assets work with concurrent statements in soft IPs, and they follow the same asset filtering process as external assets.

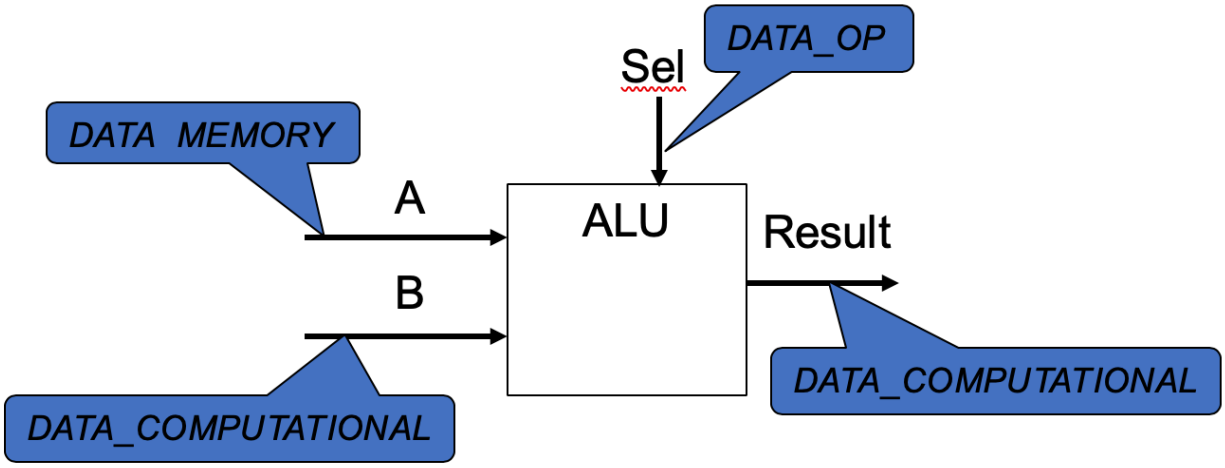

Figure 1: Simple ALU Asset Filtering

Figure 1 illustrates asset assignment to a simple ALU prior to asset filtering. In the diagram input "A" contains data from some type of memory which is why it is assigned a DATA_MEMORY asset. Input "B" stores data intended for computation which is why it is assigned a DATA_COMPUTATINAL asset. The output of the ALU, "Result", is assigned a DATA_COMPUTATIONAL asset because the "Result" signal is the result of the ALU's computation. "Sel" is assigned a DATA_OP asset because it controls the operation that is performed on the data of the ALU. Furthermore, signals "A", "B", and "Result" will have

PROCESS_SENSITIVE assets assigned to them. This occurs because "Sel" requires a process block, due to syntax rules in soft IPs, and each of these signals are contained within the same process block as "Sel". Finally, a CONDITIONAL_DRIVING asset is assigned to "Sel" and a CONDITIONAL_DRIVEN asset is assigned to "Result" because "Result's" value depends on the value of "Sel". After asset filtering, "Result" will also be assigned DATA_MEMORY because the asset comes from filtering input A's asset to the output. Input "A" will add DATA_COMPUTATIONAL to its assets because the output's computational asset filters to A. Input B will not have an additional DATA_COMPUTATIONAL asset after filtering from the output to B because there is no need to duplicate assets on one signal. As a result, each signal becomes more refined in how it fits into the design by filtering assets.

2.1.5 Asset Pattern and Characteristics

Resulting from the work done in [5], an asset pattern, which is written out to an asset file, is a compilation of all asset traces of a soft IP. Asset traces are created for every port/internal signal in a design and contain all assets assigned to it. Assets are broken down into six characteristics. External assets that are assigned to primary input port signals form a characteristic and they are denoted by ">" within a GRL file. Internal assets that are assigned to primary input port signals form another characteristic and are denoted by ">*" within a GRL file. Another characteristic includes external assets that are assigned to primary output ports and they are denoted by "<" within a GRL file. Internal assets that are assigned to primary output port signals form another characteristic and they are denoted by "<*" within a GRL file. The final two characteristics are external assets assigned to internal signals and internal assets assigned to internal signals. These two characteristics are denoted by "/" and "/*", respectively, within a GRL file.

2.2 Golden Reference Matching

2.2.1 Overview

The concept of Golden Reference Matching is to take an unknown soft IP, compare it against a Golden Library of soft IPs, which are known to be Trojan-free or Trojan-infested, and determine if it contains a Trojan. For each library entry, the algorithm behind the matching process calculates a percent match against the unknown IP by comparing the similarity of assets between the designs. Based on the best percent match of the unknown IP against the Golden Reference entries, Golden Reference Matching can provide a probabilistic result on whether or not the unknown design contains a hardware Trojan and determine the overall functionality of the design. Developed in [5], the SC tool uses a Golden Reference Library, which contains a list of known Trojan free and Trojan infested IPs.

2.2.2 Basic Matching Process

Table 1: Basic Matching Example

Trace	Unknown IP Assets	GRL Entry Assets	Percent Match
1	DATA_COMMUNICATION	DATA_COMMUNICATION	100%
2	DATA_SENSITIVE, COUNT, STATUS	DATA_SENSITIVE, HANDSHAKING, MEMORY_OP	33%
3	DATA_SENSITIVE	DATA_MEMORY	0%

Table 1 provides a simple example of the matching process. Each row of Table 1 contains the assets assigned to a single signal from the unknown IP and assets assigned to a single signal from a GRL entry. In the first row the assets are identical which gives a 100% match. Trace number 2 can only match 1 out of the 3 assets which produces a 33% match. Finally, trace 3 has no identical assets between the unknown IP and the GRL entry which results in a 0% match. These three assets traces come from the same characteristic which would result in a 44.33% match for that characteristic. The same process of matching would be done for the other 5 characteristics with the unknown IP and GRL entries. After computing a percent match for all

characteristics, the overall percent match is calculated. To calculate the overall match, an average is taken of the 6 percent matches from the characteristics. When calculating percent matches for each of the characteristics, there are special cases such as either the unknown IP or the GRL entry, or both do not have any assets in a given characteristic. In these special cases, the characteristic is left out of the overall percent match calculation.

2.2.3 Partial Matching

In [5] the idea of partial matching was added to the Structural Checking tool's matching algorithm. Partial matching involved applying a 50% match between assets that were not identical but shared a similar purpose in a design. For instance, in the *Data* category of assets, there is a DATA_SENSITIVE asset that generically classifies a signal to be dependent on some type of data. Within this same category there are assets such as DATA_MEMORY, DATA_COMMUNICATION, etc. All of these other assets in the same category are specific versions of the generic DATA_SENSITIVE asset. Consequently, the algorithm was altered to provide a partial match if an asset from either the unknown or the Golden Reference Library entry was generic while the other was specific.

Table 2: Partial Matching Example

Trace	Unknown IP Assets	GRL Entry Assets	Percent Match
1	DATA_COMMUNICATION	DATA_COMMUNICATION	100%
2	DATA_SENSITIVE, COUNT, STATUS	DATA_SENSITIVE, HANDSHAKING, MEMORY_OP	33%
3	DATA_SENSITIVE	DATA_MEMORY	50%

Table 2 provides the same example from Table 1. However, with partial matching, trace 3 now has a 50% match because DATA_SENSITIVE is a generic version of the DATA_MEMORY asset. In this case the overall percent match for this characteristic is 61% instead of 44.33%.

2.2.4 Golden Reference Library

The GRL is a collection of soft IPS designs that have been collected from Trust-Hub [10,11], OpenCores [12], and some in-house designs. All entries in the library first go through the Structural Checking tool to generate an asset pattern for the design. After generating an asset pattern, a functionality is added to the file to label the overall function of the soft IP. The combination of the asset pattern and the functionality encompasses all information needed for a library entry. All GRL entries are guaranteed to be correctly labeled in terms of Trojan-free (whitelist) or Trojan-infested (blacklist) functionality because all designs come from trusted sources. Table 3 below lists the functionalities that exist in the GRL.

Table 3: Functionalities

Whitelist Functionality	Blacklist Functionality
SHIFT_REGISTER	TROJAN_ENCRYPTION_UNIT
INTERRUPT UNIT	TROJAN_TRIGGER
COMMUNICATION	TROJAN_COMMUNICATION
ENCRYPTION_UNIT	TROJAN_SHIFT_REGISTER
COMPUTATIONAL	
TIMING	
CONTROL_GENERATION	
REGISTER_FILE	
PERIPHERAL	
DECODER_ENCODER	
DEBUG_INTERFACE	
TOP_CONTROLLER	
MICROPROCESSOR	

Unknown IPs that match best with GRL entries with a “whitelist” functionality are given that same functionality and are labeled as being clean. However, if an unknown IP matches best with a GRL entry that has a “blacklist” functionality, then the unknown IP is given the same functionality and is flagged as potentially contains a Trojan.

```
Entity    simple_pic:
    »»    33 port signals
    »»    60 IntraSignals
    »»    4 Port Signal Vectors
    »»    7 Intra-Signal Vectors
    »»    0 SubInstances
    »»    9 Processes
Functionality: INTERRUPT_UNIT
Secondary Func: NON_SEQUENTIAL
Number of Input bits:»  23
Number of Output bits:» 10
>[SYSTEM_TIMING]
>*[PROCESS_SENSITIVE, CONDITIONAL_DRIVING]
>[RESET]
>[INTERRUPT_CONTROL, HANDSHAKING]
>[ADDRESS_SENSITIVE]
>*[CONDITIONAL_DRIVING]
>[INTERRUPT_CONTROL]
>[DATA_SENSITIVE]
>[DATA_SENSITIVE, INTERRUPT]
<[DATA_SENSITIVE, INTERRUPT]
<*[CONDITIONAL_DRIVEN]
<[HANDSHAKING, INTERRUPT_CONTROL]
<*[CONCURRENT_DRIVEN]
<[INTERRUPT, DATA_SENSITIVE]
<*[CONDITIONAL_DRIVEN, PROCESS_OPERATION_SENSITIVE]
>[INTERRUPT, DATA_SENSITIVE]
/[DATA_SENSITIVE]
/*[CONDITIONAL_DRIVEN, PROCESS_OPERATION_SENSITIVE]
/[DATA_SENSITIVE, INTERRUPT]
/*[CONDITIONAL_DRIVEN]
/[HANDSHAKING, INTERRUPT_CONTROL]
/*[CONCURRENT_DRIVEN, CC_OPERATION_AND]
/[INTERRUPT_CONTROL]
/*[CONDITIONAL_DRIVING, CONCURRENT_DRIVEN, CC_OPERATION_AND]
```

Figure 2: Simple PIC GRL Entry

Figure 2 provides an example of a GRL entry. At the top of the file the entity's name is provided along with a breakdown of the type and number of signals used in the design. Then the file provides a labeled functionality which is "INTERRUPT_UNIT" in the case of Figure 2. The remainder of the file contains the asset pattern of this entry which is used for the matching process.

3 METHODOLOGY AND IMPLEMENTATION

3.1 Asset Reassignment

The Structural Checking process described in Chapter 2 of this thesis leads to bias that negatively impacts matching results. All designs that make up the Golden Reference Library, described in Section 2.2.4, require manual assignment of external assets. Additionally, the library has continued to grow in the lifespan of the Structural Checking tool. With this in mind, several developers have contributed entries to the GRL at different stages of the tool's development. As a result, designs from later stages of the tool's development were assigned assets that did not exist in earlier stages. The manual process of assigning external assets causes differences in what assets are assigned to signals within soft IPs. One developer may not fully understand a design due to lack of documentation. Consequently, there could be an abundance of generic assets assigned in this situation. Previously mentioned in Section 2.2.3, the Structural Checking tool can handle matching generic assets to their specific asset counterparts with partial matching. However, this way of matching always reduces the percent match to 50% even when the assets could theoretically be identical but are not due to bias in asset assignment. In order to alleviate these issues, the idea of asset reassignment was added to the tool's matching algorithm. When comparing external assets, the algorithm will perform a check on both the target (unknown) IP's assets and the library entry's assets to see if one asset is a generic version of the other. In the case that one asset is a generic version of the other, the more specific asset is reassigned to the generic asset. For instance, if the target IP has a DATA_MEMORY asset assigned to a signal and is compared to a DATA_SENSITIVE asset from the GRL entry, then the target IP's asset is reassigned to be DATA_SENSITIVE. This replaces partial matching in the algorithm by now giving a 100% match to assets that are similar instead of 50% because the

more specific asset has been reassigned. The process of reassignment does not damage any original intent of assets assigned because each reassigned asset keeps the same general purpose as the original assignment. The only change that occurs is that a specific asset changes to its generic equivalent.

3.2 Statistical Weighting

3.2.1 Overview

As explained in Section 2.2, the overall percent match between the target IP and a GRL entry is calculated by taking the average of the six percent matches from the six characteristics that make up an asset pattern. Figure 3 below illustrates the equation used for this process and denotes the six characteristics as "A" through "F".

$$Overall\ \%\ Match = \frac{\sum_{i=A}^{F} \%\ Match_i}{6}$$

Figure 3: Equal Weight Percent Match

One drawback to averaging all percent matches from the characteristics is that each characteristic then contributes equal weight to the overall percent match. However, external assets provide more information to a soft IP's functionality because external assets currently offer more specific descriptions of how a signal functions within an IP. While internal assets do contribute to the signal's overall asset trace, they do not provide information on what function the signal provides to the overall design. On the other hand, an external asset, such as DATA_MEMORY, provides a specific description that the signal is both data and contributes to some type of memory. In theory, external asset characteristics should be weighted more when calculating the overall percent match.

3.2.2 Assessing Asset Quantity

A common practice in statistical analysis is to weight final results when working with a subgroup that is either underrepresented or overrepresented relative to the size of the full group. For example, if the population of the Earth is known to be 51% female and 49% male, but a survey involving just a small group is made up of 60% male and 40% female, then the final results should be weighted. The female results should be multiplied by 51 over 40 (greater than 1) to account for females being underrepresented in the poll. Similarly, the male results should be multiplied by 49 over 60 (less than 1) to account for males being overrepresented in the poll.

Tying in with asset pattern matching, a similar approach was considered to help weight the six characteristics. For each GRL entry and the target IP, the number of assets per characteristic was recorded. Next, the GRL entries were split into groups based on their functionalities (Table 3).

Table 4: Asset Quantity Example

GRL Entry #	Input External	Input Internal	Output External	Output Internal	Signal External	Signal Internal
1	6	2	4	3	3	5
2	2	2	0	3	0	5
3	5	2	3	3	3	5
4	5	4	3	1	4	10
5	9	5	5	5	5	5
6	8	2	5	3	3	5
7	7	2	5	2	3	8
8	9	7	5	5	2	7
9	4	2	2	4	1	6
10	7	2	6	1	4	2

Table 4 provides an example of breaking down the GRL entries into their functionalities. Each row in Table 4 is a different GRL entry with a COMMUNICATION functionality. The remaining columns to the right of the GRL entry number column represent the six characteristics: input ports assigned external assets, input ports assigned internal assets, external ports assigned external assets, external ports assigned internal assets, internal signals assigned

external assets, and internal signals assigned intern assets. The numbers underneath each characteristic column represent the number of assets defined in the characteristic for these 10 GRL entries.

In the statistical example provided earlier in this section, weight was determined by taking the known value divided by the sampled value. In the case of soft IPs, there is no known number of assets that should be in each characteristic because soft IPs can be designed in numerous ways but still retain the same overall functionality. Therefore, for each GRL entry in Table 4, weight was determined for each characteristic by taking the larger number of assets (comparing number of assets in a characteristic between a GRL entry and the target IP) divided by the smaller number. After calculating individual sets of weights for each GRL entry, a final set of weights was determined for each functionality by taking an average weight for each characteristic.

This process of determining a set of weights for each functionality in the GRL proved to harm the overall percent match by matching target IPs to library entries that did not have the same functionality. The inefficient nature of this approach relates to the idea that soft IPs can be designed differently but still retain the same overall functionality. This fact indicates that the number of assets in a characteristic provides no correlation to an IP's functionality. The idea of weighting characteristics was added to the matching process in order to account for the fact that certain assets provide more information to an IP's functionality. Consequently, the method for determining weight should seek to use data that reflects how well assets in the GRL are matching to the target IP as opposed to the number of assets. In other words, weight should consider the quality of assets in the GRL.

3.2.3 Assessing Asset Quality

Assessing the quality of assets in the GRL focuses on the frequency in which each asset appears in the library. Matching assets that appear in all entries of the library provides little information regarding which entry is most similar to the target IP as compared to assets that only appear in a few entries. For instance, the SYSTEM_TIMING asset is commonly found in most library entries because most entries have some type of timing component. On the other hand, the DATA_ENCRYPTION asset is more commonly found in entries with an ENCRYPTION functionality but is not common to all entries. Therefore, a DATA_ENCRYPTION asset can provide more information about which entries are most similar to the target instead of a more common asset, such as SYSTEM_TIMING. With this in mind, weighting should emphasize the differences in frequency of each asset in the GRL.

In order to determine weight for the six characteristics described in Section 2.1.5, the first step involves calculating the probability of each asset among all assets of the GRL.

$$P(Asset) = \frac{\sum_{i=1}^{n} GRLEntry_i.contains(Asset)}{Total\ \#\ of\ GRL\ Entries}$$

Figure 4: Probability of Asset in GRL

The equation in Figure 4 runs through all "n" GRL entries and will either add one to the numerator if the GRL entry contains the asset, or zero if the GRL entry does not contain the asset. If multiple instances of the same asset exist in any one GRL entry, only one is added to the total because the probability checks only for the presence of an asset. The number of entries that contain the asset is then divided by the total number of GRL entries to obtain the probability that a library entry has that asset.

Table 5: Example GRL

GRL Entry #	Assets
1	SYSTEM_TIMING, DATA_ENCRYPTION, PROCESS_SENSITIVE
2	SYSTEM_TIMING, DATA_MEMORY, PROCESS_SENSITIVE
3	SYSTEM_TIMING, DATA_MEMORY, CONDITIONAL_DRIVEN
4	CONDITIONAL_DRIVEN, ADDRESS_SENSITIVE
5	SYSTEM_TIMING, CONDITIONAL_DRIVEN

Table 5 provides an example GRL to demonstrate calculating asset probability. There are six unique assets defined in this GRL: SYSTEM_TIMING, DATA_ENCRYPTION, PROCESS_SENSITIVE, DATA_MEMORY, CONDITIONAL_DRIVEN, and ADDRESS_SENSITIVE. The probability for SYSTEM_TIMING is equal to four divided by five because SYSTEM_TIMING is present in four of the five GRL entries. The probability for the remaining assets would be calculated in the same manner.

With the probability of each asset calculated, the next step involves calculating a weight for each asset.

$$Weight_{Asset} = 1 - P(Asset)$$

Figure 5: Asset Weight Calculation

As shown in Figure 5, an asset weight is determined by subtracting the probability of an asset from one. This type of calculation is thus determining the probability that an asset will not be in a GRL entry. By assigning weight in this manner, assets that are not commonly found in GRL entries will have higher weights compared to assets that show up frequently in the GRL.

After determining the weight for each asset in the GRL, one final set of weights for the six characteristics can be calculated.

$$Average\ Asset\ Weight = \frac{\sum_{i=1}^{n} MatchedAsset_i.weight}{Total\ \#\ Matched\ Assets}$$

Figure 6: Characteristic Average Asset Weight

Figure 6 describes how to calculate the average weight of the matched assets in an arbitrary characteristic. The numerator keeps a running total of asset weights for each asset that was matched within a characteristic. The sum of matched asset weights is then divided by the total number of matched assets in the characteristic to obtain an average weight. If the calculation shows that a characteristic has a relatively high weight, this indicates that assets matched within this characteristic tended to have higher weight. As a result, the assets within this characteristic are less common in the GRL. Therefore, the characteristic should receive a higher weight relative to the other characteristics when calculating the overall percent match because the assets within the characteristic provide a more unique identification to the functionality of the target IP.

$$Weight_{char} = \frac{Characteristic_{char}.AverageAssetWeight}{\sum_{i=A}^{F} Characteristic_i.AverageAssetWeight} * 100$$

Figure 7: Characteristic Weight Calculation

Figure 7 illustrates the final step in calculating the weight for an arbitrary characteristic, "char." The average asset weight, determined from the equation in Figure 6, of "char" is divided by the sum of all characteristics' average asset weights. The quotient is then converted into a percentage based on each characteristic's contribution to the total average asset weight from all six characteristics. As a result, characteristics with higher average asset weight are weighted more in the overall percent match calculation, which reflects the idea of weighting characteristics based on the weight, or quality, of assets matched.

4 RESULTS AND ANALYSIS

4.1 Overview

During testing, results from [13] were used to confirm the tool's ability to maintain correct classification with the changes made to the matching algorithm. The tested IPs include RS232, Basic-RSA, and AES. In addition to these relatively smaller designs, a few microcontrollers were used to test the improved algorithm. Due to the fact that the current state of the GRL contains very few IPs similar in size to a microcontroller, the statistical based algorithm should help extract important asset matches to obtain the best classification for each microcontroller.

4.2 Examples

4.2.1 RS232, Basic-RSA, AES Modules

A RS232 Trojan-infested module used during testing includes RS232-T700. RS232-T700 contains a Trojan in its transmitter which produces a denial-of-service attack on the module by forcing the transmitter's done signal to be stuck at 0.

Table 6: RS232-T700 Matching Results

	Equal Weight Matching		Statistical Based Matching	
Target IP	**Functionality**	**% Match**	**Functionality**	**% Match**
Uart.vhd	COMMUNICATION	100%	COMMUNICATION	100%
U_xmit.vhd	TROJAN_COMMUNICATION	99.206%	TROJAN_COMMUNICATION	99.490%
U_rec.vhd	COMMUNICATION	90%	COMMUNICATION	94.641%

Table 6 shows results from both the original and improved matching algorithms. In both instances the algorithm correctly classifies the transmitter as containing a Trojan and thus demonstrates the new algorithm's ability to keep correct results from previous work.

The Basic-RSA module tested was BasisRSA-T200. This module is another denial-of-service attack which disables encoding on the transmitter and decoding on the receiver.

Table 7: BasicRSA-T200 Matching Results

	Equal Weight Matching		Statistical Based Matching	
Target IP	**Functionality**	**% Match**	**Functionality**	**% Match**
RSACypher.vhd	TROGAN_ENCRYPTION_UNIT	74.900%	TROGAN_ENCRYPTION_UNIT	83.426%
Modmult.vhd	COMPUTATIONAL	92.5%	COMPUTATIONAL	100%

Similar to the results of RS232-T700, this design was correctly classified by both versions of the matching algorithm.

Finally, AES-T600 was used to further confirm the improved algorithm's ability to maintain correct classification. The secret key of the module can be discovered after a certain plaintext is read. As shown in Table 8, the algorithm correctly classifies the IP as containing a Trojan.

Table 8: AES-T600 Matching Results

	Equal Weight Matching		Statistical Based Matching	
Target IP	**Functionality**	**% Match**	**Functionality**	**% Match**
Top.vhd	TROJAN_ENCRYPTION_UNIT	45%	TROJAN_ENCRYPTION_UNIT	44.017%
Aes_128.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
Expand_key_128.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
S4.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
S.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
One_round.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
Table_lookup.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
T.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
xS.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
Final_round.vhd	ENCRYPTION_UNIT	100%	ENCRYPTION_UNIT	100%
Trojan_trigger.vhd	TROJAN_TRIGGER	66.667%	TROJAN_TRIGGER	98.694%
TSC.vhd	TROJAN_SHIFT_REGISTER	93.75%	TROJAN_SHIFT_REGISTER	94.129%

4.2.2 PIC16F84-T100

The benchmark PIC16F84-T100, acquired from Trust-Hub [10, 11], demonstrates improvement in the overall percent match using the statistical weighting method described in Section 3.2.3. Additionally, this microcontroller offers the GRL an additional trusted IP that is relatively larger than most entries in the top-level section of the library. This microcontroller is made up of two different types of memory (EEPROM and RAM), a watchdog timer, interrupt ports, and I/O ports. Each of these components of the design are contained within one VHDL

file. Once parsed by the SC tool, assets are assigned to the primary input and output ports. These ports, and their corresponding assets, are provided in Table 9.

Table 9: PIC16F84-T100 Asset Assignment

Signal	Assets
clk_i	SYSTEM_TIMING
clk_o	SYSTEM_TIMING
eep_adr_o	ADDRESS_SENSITIVE
eep_dat_i	DATA_MEMORY
eep_dat_o	DATA_MEMORY
existeprom_i	MEMORY_OP
int0_i	INTERRUPT
int4_i	INTERRUPT
int5_i	INTERRUPT
int6_i	INTERRUPT
int7_i	INTERRUPT
mclr_n_i	RESET
pon_rst_n_i	RESET
porta_dir_o	PERIPHERAL_CONTROL
porta_i	DATA_PERIPHERAL
porta_o	DATA_PERIPHERAL
portb_dir_o	PERIPHERAL_CONTROL
portb_i	DATA_PERIPHERAL
portb_o	DATA_PERIPHERAL
powerdown_o	CLOCK_CONTROL
prog_adr_o	ADDRESS_SENSITIVE
prog_dat_i	DATA_MEMORY
ram_adr_o	ADDRESS_SENSITIVE

Table 9 (Cont.)

Signal	Asset
Ram_dat_i	DATA_MEMORY
Ram_data_o	DATA_MEMORY
Rbpu_o	UNUSED
Rd_eep_ack_i	HANDSHAKING, MEMORY_OP, READ
Rd_eep_req_o	HANDSHAKING, MEMORY_OP, READ
Readram_o	HANDSHAKING, MEMORY_OP
Startclk_o	CLOCK_CONTROL
T0cki_i	TIMER_CONTROL
Wdt_clk_i	SUBSYSTEM_TIMING
Wdt_ena_i	CLOCK_CONTROL
Wdt_full_o	CLOCK_CONTROL
Wr_eep_ack_i	HANDSHAKING, MEMORY_OP, WRITE
Wr_eep_req_o	HANDSHAKING, MEMORY_OP, WRITE
Writeram_o	HANDSHAKING, MEMORY_OP

After completing asset assignment, the SC tool filters these assets to connected signals as described in Section 2.1.4. Finally, the matching process was carried out both with equal characteristic weights (no asset reassignment) and statistical characteristic weights (with asset reassignment).

Table 10: PIC Original Matching Results

GRL Entry	Overall Percent Match
Simple_pic	52.553%
Lcd16x2_ctrl	48.233%
Lcd_controller	44.148%
RSACypher_T100	43.414%
Spi_master_1	40.750%

Table 11: PIC Statistical Matching Results

GRL Entry	Overall Percent Match
Simple_pic	47.149%
Lcd16x2_ctrl	36.591%
RSACypher_T100	36.514%
Lcd_controller	31.785%
Spi_master_1	30.211%

Table 10 provides the top five overall percent matches from equal weighting of the characteristics and no asset reassignment. Table 11 provides the top five overall percent matches from statistical weighting of the characteristics with asset reassignment. While statistical weighting did lower all percent matches, the overall results indicate better matching with GRL entries that are most similar to the target's functionality. The calculated weights for the characteristics when matching this microcontroller were 20.908 for "input external", 8.037 for "input interal", 26.276 for "output external", 8.156 for "output internal", 26.406 for. "signal external", and 9.345 for "signal internal". As expected, characteristics with external assets were weighted more due to the fact that they provide assets that produce a better description of each IP's functionality. Analyzing the results of Table 10, most GRL entries were within a few percentage points of the next best match. However, the results from Table 11 show greater

disparity among the results which indicates that the algorithm provided more separation based on assets that were within each library entry. The "simple_pic" entry matched the most assets of high weight with PIC16F84-T100 which reflects the idea of matching based on quality of assets. This also proves the effectiveness of the statistical approach because the GRL entry that is most similar in functionality to the PIC microcontroller is indeed the "simple_pic".

4.2.3 MC8051-T500 Core

The 8051-microcontroller core tested is known to be Trojan-free. The core is made up of control units for a finite state machine (FSM) and memory, an ALU (with several specialized blocks for computations), a serial interface unit (SIU), and a timing unit (also handles interrupt signals). External assets were assigned to the core's top module, "MC8051_core.vhd". Additionally, external assets were assigned to some of the core's internal signals because not all subcomponents of the IP were fully connected to the primary ports of the top module. As aforementioned, asset filtering would not be able to fully define the signals of subcomponents without the manual assignment of internal signals. Once asset assignment was complete, asset filtering was performed, and the matching process was done on the core using both equal weight and statistical weight for the characteristics.

Table 12: MC8051-T500 Core Matching Results

	Equal Weight Matching		**Statistical Based Matching**	
Target IP	**Functionality**	**% Match**	**Functionality**	**% Match**
MC8051_core.vhd	COMMUNICATION	35.321%	INTERRUPT_UNIT	50.899%
MC8051_control.vhd	COMPUTATIONAL	44.871%	REGISTER_FILE	54.689%
Control_fsm.vhd	COMPUTATIONAL	47.767%	REGISTER_FILE	38.913%
Control_mem.vhd	INTERRUPT_UNIT	61.576%	INTERRUPT_UNIT	62.274%
MC8051_alu.vhd	COMPUTATIONAL	22.244%	COMPUTATIONAL	29.564%
Alumux.vhd	COMPUTATIONAL	55.565%	COMPUTATIONAL	46.519%

Table 12 (Cont.)

	Equal Weight Matching		Statistical Based Matching	
Target IP	**Functionality**	**% Match**	**Functionality**	**% Match**
Alucore.vhd	COMPUTATIONAL	50.297%	COMPUTATIONAL	42.133%
Addsub_core.vhd	COMPUTATIONAL	44.250%	COMPUTATIONAL	41.169%
Addsub_cy.vhd	COMPUTATIONAL	46.875%	COMPUTATIONAL	44.748%
Addsub_ovcy.vhd	COMPUTATIONAL	46.875%	COMPUTATIONAL	44.748%
Comb_mltplr.vhd	COMPUTATIONAL	45.833%	COMPUTATIONAL	38.863%
Comb_divider.vhd	COMPUTATIONAL	37.500%	COMPUTATIONAL	35.399%
Dcml_adjust.vhd	COMPUTATIONAL	31.718%	COMPUTATIONAL	34.492%
MC8051_siu.vhd	COMMUNICATION	77.152%	COMMUNICATION	70.793%
MC8051_tmrctr.vhd	REGISTER_FILE	52.257%	INTERRUPT_UNIT	48.587%

When matching the top-level module, “MC8051_core.vhd”, the equal weighting matching process determined the functionality of the core to be COMMUNICATION while the statistical based process determined the functionality to be INTERRUPT_UNIT. The INTERRUPT_UNIT functionality comes from the “simple_pic” GRL entry. Matching the 8051-microcontroller with another microcontroller proves that the statistical algorithm found the best way to match the top-level module. The 8051-core and “simple_pic” are still very different which is reflected in the 50% match at the top-level. The next three control files in Table 12 all matched with functionalities that differed from expected. In theory, each design should match best with a CONTROL_GENERATION functionality because each is intended to generate control signals for the microcontroller. However, the GRL contains very few entries within this functionality and the entries that do exist are related to program counters. The next few files in Table 12, starting from “MC8051_alu.vhd” and going to “dcml_adjust.vhd”, are shown to be correctly classified as COMPUTATIONAL by both matching approaches. Next, “MC8051_siu.vhd” is a

serial interface component of the microcontroller which confirms the correct classification of COMMUNICATION by both matching processes. Finally, "MC8051_tmrctr.vhd" contains control signals for both timing and interrupt components of the 8051-core. Consequently, this design theoretically matches best with a TIMING or INTERRUPT_UNIT functionality. In the equal weight example, this component of the 8051 matches incorrectly with a REGISTER_FILE functionality due to the large number of data assets that inflated its percent match with register files in the GRL. However, the statistical based approach extracted a better match with an INTERRUPT_UNIT functionality which reflects the designs interrupt control signals. The low percent match in the statistical approach demonstrates that the best match is still different from the component of the 8051-core.

5 CONCLUSION AND FUTURE WORK

The statistical based matching algorithm, with asset reassignment, proved to enhance the matching algorithm for the Structural Checking tool. By calculating weights for individual assets, the tool can better determine how well an asset can uniquely identify a soft IP. Using asset weights helped facilitate a way to weight the six characteristics by providing a numerical representation of how important each characteristic is to classification. Characteristics with relatively high weights contained more assets that provide a more unique identification for a target IP. The tests done in this thesis indicate that the statistical based algorithm is an effective approach to matching. In the tests performed in Section 4.2.1, all Trojan test cases were not only classified correctly but were also given a higher percent match using the statistical matching process. Furthermore, the microcontrollers tested were able to match with similar library entries but with relatively low percent matches. The low percent matches reflect the fact that the GRL does not contain very many TOP_CONTROLLER entries and thus makes it difficult for a

microcontroller to find a high percent match. To increase the percent match of soft IPs similar in size to a microcontroller, more designs of that size can be added to the GRL in the future. In addition to more GRL entries, the matching algorithm can continue to improve with additional assets in order to better refine the purpose of each signal within an IP. Finally, future work can also continue to grow the list of functionalities to provide the matching algorithm with more options to classify unknown designs.

REFERENCES

[1] Hossain, F. S., Shintani, M., Inoue, M., & Orailoglu, A. (2018). Variation-Aware Hardware Trojan Detection through Power Side-channel. *2018 IEEE International Test Conference (ITC)*. doi:10.1109/test.2018.8624866

[2] Esirci, F. N., & Bayrakci, A. A. (2017). Hardware Trojan detection based on correlated path delays in defiance of variations with spatial correlations. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. doi:10.23919/date.2017.7926976

[3] Hasegawa, K., Yanagisawa, M., & Togawa, N. (2017). Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier. *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. doi:10.1109/iscas.2017.8050827

[4] Inoue, T., Hasegawa, K., Yanagisawa, M., & Togawa, N. (2017). Designing hardware trojans and their detection based on a SVM-based approach. *2017 IEEE 12th International Conference on ASIC (ASICON)*. doi:10.1109/asicon.2017.8252600

[5] Weaver, L., Le, T., & Di, J. (2016). Golden Reference Library Matching of Structural Checking for securing soft IPs. *SoutheastCon 2016*. doi:10.1109/secon.2016.7506737

[6] Le, T., & Di, J. (2017). Golden reference matching for gate-level netlist functionality identification. *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. doi:10.1109/mwscas.2017.8052986

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

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

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

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

[11] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, M. Tehranipoor, “Benchmarking of Hardware Trojans and Maliciously Affected Circuits”, Journal of Hardware and Systems Security (HaSS), April 2017.

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

[13] Le, T., Weaver, L., Di, J., Zhang, S., & Jin, Y. (2018). Hardware Trojan Detection and Functionality Determination for Soft IPs. *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*. doi:10.1109/ivsw.2018.8494891