12-2023

Trojan Detection Expansion of Structural Checking

Zachary Chapman
University of Arkansas-Fayetteville

Trojan Detection Expansion of Structural Checking

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

by

Zachary Chapman
University of Arkansas
Bachelor of Science in Computer Science, 2022
University of Arkansas
Bachelor of Science in Computer Engineering, 2022

December 2023
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

Jia Di, Ph.D.
Thesis Director

Brajendra Panda, Ph.D.
Committee Member

John Gauch, Ph.D.
Committee Member

ABSTRACT

With the growth of the integrated circuit (IC) market, there has also been a rise in demand for third-party soft intellectual properties (IPs). However, the growing use of such Ips makes it easier for adversaries to hide malicious code, like hardware Trojans, into these designs. Unlike software Trojan detection, hardware Trojan detection is still an active research area. One proposed approach to this problem is the Structural Checking tool, which can detect hardware Trojans using two methodologies. The first method is a matching process, which takes an unknown design and attempts to determine if it might contain a Trojan by matching the unknown design to designs in a Golden Reference Library (GRL). The other method is interpreting structural elements of specific Trojan taxonomies via the use of Trojan detection functions, which is what this thesis research expands upon. The objective of this research is to enhance the tool's capabilities by incorporating three additional Trojan taxonomies into the list of detectable Trojans through the implementation of new Trojan detection functions. This expansion to the Structural Checking tool is achieved through the study of sensitive data leakage Trojans, data modification Trojans, and denial-of-service Trojans.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Jia Di, for allowing me to conduct research under him. He has helped me grow in my academic career and helped me achieve great things.

I would like to thank my committee members Dr. John Gauch and Dr. Brajendra Panda. Dr. Gauch gave me the opportunity to be a teaching assistant for his course and helped me grow as a teacher. Dr. Panda has given me countless pieces of advice while being my advisor during my undergraduate career. I appreciate both of their time for accepting and serving as my committee members.

I would also like to thank Hunter Nauman. He is the lead on the Structural Checking tool project and has helped me grow and thrive in research. I could not have done this without his mentorship and guidance.

DEDICATION

To my former teachers and mentors, they were always there to help me when I needed it. Thanks to them I was able to become a successful engineering student, without their enthusiasm and encouragement it's possible I would have pursued a different career.

To my Mom and Dad for supporting me throughout my life. They have always encouraged me to never give up and always do my best. Their support has always helped me no matter how big or small it was.

Lastly to my partner Chloe, who supports me every day. She always brings a smile to my face and helps me solve my problems. Without her being there for me I would easily struggle far more than I should on a task. I couldn't have gone on this journey without her.

CONTENTS

1. INTRODUCTION

With integrated circuits (ICs) being everywhere today, it is becoming increasingly important to make sure they are developed in a cost-effective manner. Due to this, many designers of ICs have started to use third-party soft intellectual properties (IPs) to save cost. Today a malicious user could compromise these third-party soft IPs with many different types of malicious code. The focus of this research is on detecting hardware Trojans that appear in soft IP. A hardware Trojan, as defined in [1], is any malicious component or code that is designed to appear to perform a certain action, but in reality is performing a hidden task that can harm the circuit. This could be due to the Trojan leaking sensitive data, causing a denial-of-service, degrading the performance of the circuit, or a plethora of other undesired effects. Even if a hardware Trojan only compromises a small aspect of a design, the whole design is at risk of failing. For these reasons, research in hardware Trojan detection plays a crucial role in ensuring the security of the IC.

Over the years there have been many different methods proposed to detect hardware Trojans. One group of researchers used machine learning to detect hardware Trojans at the register-transfer level (RTL) [2]. This research trained a machine learning algorithm to search through the abstract syntax tree (AST) developed from RTL code. Based upon learned aspects of a Trojan-infested design's AST, the machine learning algorithm could detect Trojans in an unknown design. Another team looked at side-channel analysis to determine if a Trojan had been inserted into the design [3]. With the use of cross-validation and a support vector machine the authors were able to train a machine learning algorithm to detect Trojan's in designs based upon their power traces captured via side-channels. Some have even gone so far as to investigate destructive Trojan detection by opening a chip and analyzing it layer by layer to determine the

inclusion of Trojan logic [4]. By removing each layer of the chip at a time a user could see the inner workings of the chip and notice any malicious components that should not be in the chip.

The Structural Checking tool, first introduced in [5], is developed to detect hardware Trojans using two different methodologies. The Structural Checking tool focuses on hardware Trojans that are inserted into a design at the register-transfer level. Along with this tool is a Golden Reference Library (GRL) that contains a number of (144 as for now) designs both clean and Trojan-infested spanning over eighteen different functionalities. These designs are used in one of the tool's hardware Trojan detection methods known as GRL matching. This is the process of comparing an unknown design against all designs within the GRL, using statistical matching, in an attempt to detect potential Trojans [6]. The second method of Trojan detection employed by the Structural Checking tool is structure-based Trojan detection functions. These Trojan detection functions analyze circuits in the search of structural elements and key asset signifiers common within various Trojan taxonomies. Currently within the tool there are four Trojan detection functions for the following Trojan taxonomies: counter Trojans, battery drain Trojans, key leakage Trojans, and clock Trojans. This research expands upon this aspect of the tool with the addition of three new Trojan detection functions for the following Trojan taxonomies: sensitive data leakage Trojans, data modification Trojans, and denial-of-service Trojans.

The following chapter will go into more detail pertaining to the specific operations within the Structural Checking tool. This includes the process that a design will undergo while being processed through the tool and the pertinent information needed to describe Trojan detection functions. Chapter 3 provides the methodology and implementation of three new Trojan detection functions to the tool. This expands the second method of Trojan detection within the tool, bringing the total number of Trojan detection functions for specific Trojan taxonomies up to

seven. Chapter 4 discusses the results of these new Trojan detection functions. Results are gathered by testing how the functions perform against designs with their respective Trojan targets and clean designs with no Trojans inserted. The final chapter will provide some possible expansions to the new Trojan detection functions and improvements to the Structural Checking tool.

2. BACKGROUND

2.1 Structural Checking Tool

2.1.1 Overview

The Structural Checking Tool was designed to target hardware Trojans in soft IPs at the RTL. This is achieved via two main Trojan detection methods, a matching process and Trojan detection functions. One major component of the Structural Checking tool is its GRL. The GRL is used in the statistical matching process that first determines an unknown design's approximate functionality via champion matching, then determines the presence of malicious logic by comparing the unknown designs against all designs of that functionality. Both stages of this matching process are performed by utilizing asset patterns within both the unknown designs and the designs being matched against. Trojan detection functions focus on structural elements and asset patterns. These Trojan detection functions are created to target specific Trojan taxonomies and do so by looking for specific structural elements and asset patterns common to that specific Trojan.

Each design processed through the Structural Checking tool must undergo the same steps. First, a pre-processing step is performed by the Structural Checking tool. The objective of this pre-processing step is to parse all structural information from each hardware description language (HDL) file within a given design. This step allows for the extraction of ports, generics, internal signals, component declarations, and much more. After the design has been parsed for all structural information, asset assignment must be performed on all primary port/generic signals within the design. The Structural Checking tool then performs asset filtering to distribute asset assignments across driving/driven connections to give a more representative view of each

signal's purpose within the circuit. The next step of the Structural Checking tool is to perform the matching process. The final step is then to perform structure-based Trojan detection. During this step, all structure-based Trojan detection algorithms are run to find any possible Trojan signals and report them back to the user.

2.1.2 Assets

One of the major components of the Structural Checking tool is assets. The goal of assets is to describe a signal's purpose and functionality within the circuit. A single signal can be assigned as many assets as needed to complete that description. The Structural Checking tool contains two distinct asset types: external assets and internal assets. External assets are assigned to primary port/generic signals of each component within a given design. Internal assets are applied to all signals withing the design, primarily automatically, to describe syntactic information parsed from the HDL. There are additional structure-focused, manually assignable internal assets that are also available for assignment. These assets are crucial in the Golden Reference Matching process and the Trojan detection functions.

2.1.3 External Assets

As described by the author of [8], external assets are manually assigned to the primary port/generic signals of an RTL design. As mentioned, these assets must be assigned manually by a user of the Structural Checking tool. A common example is that in every cryptographic soft IP design there is a signal that contains information on the key. This signal should be assigned the *"key"* data asset. As new designs are developed, new external assets could be created to better describe the signals in these new designs. Within the last five years, the tool has expanded from fifty external assets in five categories to now seventy-five in seven categories [9]. These

categories are based on signal functionalities and are designed to cover a wide range of varying design types. The seven categories that are currently in the Structural Checking tool are data, timing, system control, specific system control, instruction set, parameter, and miscellaneous. Table 1-7 contains the current full list of external assets with a table for each category.

The data asset category contains external assets that correlate to the transfer of data such as sensitive data, computational data, or critical data.

Table 1: Data asset category

ANY	The signal may accept any type of data depending on configuration or settings
COMPUTATIONAL	The signal contributes to the flow of computational data, such as data used within arithmetic units
MEMORY	The signal contributes to the flow of memory data
COMMUNICATION	The signal handles transmission with external components (ex. UART)
PERIPHERAL	The signal contains data used within the peripherals of the design (ex. Display, Temp, etc.)
ENCRYPTION	The signal contains information about data to be encrypted
DECRYPTION	The signal contains information about data to be decrypted
HASH	The signal contains information about data to be hashed
ENCODING	The signal contains information about data used within an encoding process

Table 1 (Cont.)

DECODING	The signal contains information about data used within a decoding process
ADDRESS	The signal controls addresses used in memory units of the system
KEY	The signal controls a key within an encryption/decryption unit
SENSITIVE	The signal contributes to data that should remain confidential to the circuit
CRITICAL	The signal contributes to data important to the operation of the circuit and could cause issues were it to be tampered with
TEST_IN	The TDI (Test Data In) signal of the circuit
TEST_OUT	The TDO (Test Data Out) signal of the circuit

The timing asset category is to accommodate any signals that relate to timing, e.g., clock signals. This category also includes external assets for counters and other signal indicators.

Table 2: Timing asset category

CLOCK	The signal is the system\’s primary clock
CLOCK_CONTROL	The signal contributes to the control of the system\’s primary clock
SYSTEM_CLOCK_CONTROL	The signal is a subsystem\’s primary clock
SUBSYSTEM_CLOCK_CONTROL	The signal contributes to the control of a subsystem\’s primary clock

Table 2 (Cont.)

SYSTEM_TIMING	The signal controls timing for the entire system, such as timing between synchronous components of the circuit
STATUS	The signal indicates the status of the system
READY	The signal indicates the status of the system
DONE	The signal indicates whether or not an operation has finished
BUSY	The signal indicates whether or not an operation is busy
HOLD	The signal indicates whether or not to hold an operation
COUNT	The signal is used as a counter within the design
WAIT	The signal indicates whether or not an operation must wait
STANDBY	The signal indicates an operation with a state of readiness without being immediately involved
TEST_CLOCK	The TCK (Test Clock) signal of the circuit

The system control asset category is to contain signals that are related to more broad control of systems such as enable, flag, and reset signals.

Table 3: System Control asset category

ENABLE	The signal controls a structure by enabling its operation
_SET	The signal controls a set operation on part of the circuit
RESET	The signal controls a reset operation on part of the circuit
EXECUTE	The signal controls execution of an operation
READ	The signal controls a read operation
WRITE	The signal controls a write operation
SELECT	The signal controls a select operation
LOAD	The signal controls a load operation
SHIFT	The signal controls a shift operation
INTERRUPT	The signal controls an interrupt signal
MODE	The signal controls the mode of a data processing block
ACKNOWLEDGE	The signal is used to acknowledge that an event of some sort has occurred
HANDSHAKING	The signal contributes to communication by way of a handshaking operation
DATAFLOW	The signal controls where data will be sent to
FLAG	The signal is used as a flag bit to control the operation of something
REQUEST	The signal is used for making requests to other modules
TEST_MODE_SELECT	The TMS (Test Mode Select) signal of the circuit
TEST_RESET	The TRST (Test Reset) signal of the circuit

The specific system control asset category relates to signals that are for specific control, e.g., systems like UARTs. These signals usually only apply to one type of system.

Table 4: Specific System Control asset category

INTERRUPT_CONTROL	The signal controls an interrupt unit
PERIPHERAL_CONTROL	The signal controls the peripherals of the design (ex. Display, Temp, etc.)
MEMORY_CONTROL	The signal controls memory information
COMMUNICATION_CONTROL	The signal controls transmission with external components (ex. UART)
COMMUNICATION_PROTOCOL	The signal handles protocol bit from an external component (ex. UART)
COMMUNICATION_STATUS	The signal handles a transmit ready signal from external components (ex. UART)
BUS_CONTROL	The signal controls access to a bus
DUTY_CYCLE	The signal controls duty-cycle-related operations
PHASE	The signal controls phase-related operations
EXCEPTION_HANDLING	The signal handles exceptions within the system
ERROR_HANDLING	The signal handles errors within the system

The instruction set asset category is for signals that relate to instruction set architecture. This would be signals that handle instructions or different operations for processing units and microcontrollers.

Table 5: Instruction Set asset category

INSTRUCTION	This is a generalized instruction asset that should be applied to signals that don\'t fit a more specific instruction description
OPERAND	This signal is an operand used for an instruction
OPERATION_TYPE	This signal sets the type of operation performed by an instruction
SOURCE	This signal describes the location of source data for use with the instruction or is the source data itself
PROGRAM_COUNTER	The signal manipulates the value within a program counter
BRANCH	This signal is used for branch operations
OFFSET	This signal describes offsets used for instruction decoding, encoding, and manipulation
PROGRAM_COUNTER_OP	The signal controls change within a program counter
DATA_OP	The signal controls the operation of a unit dealing with data such as ALU/Data operations
MEMORY_OP	The signal controls operations of a memory unit
INTERRUPT_OP	The signal controls operations of an interrupt unit
PRIORITY	This signal sets the priority or importance of an instruction
AVAILABILITY	This signal sets the availability stage of an instruction for bypassing
PIPELINE_CLEAR	This signal clears the instruction pipeline
PIPLINE_LOCK	This signal locks the instruction pipeline

The parameter asset category covers signals that act as the parameters for different functions and modules in HDL code.

Table 6: Parameter asset category

CONFIGURATION	This is a more generic asset used for when the other parameter assets do not fit properly but another asset is unnecessary
INITIALIZATION	Parameter related to the initialization of some data structure or component
FREQUENCY	Parameter that specifies values related to frequency
TIMING	Parameter that specifies values related to timing
PHASE	Parameter that specifies values related to phase
DATA_WIDTH	Parameter that specifies the data width of some data structure or component
GENERATE_CONTROL	Parameter that specifies how some generate statement should operate
ENABLE	Parameter that enables or disables some feature or features of the design

The Miscellaneous asset category is used to hold the assets that do not properly fall into any of the previously mentioned categories.

Table 7: Miscellaneous asset category

COMPONENT	The signal controls components not defined by other assets
UNKOWN	Cannot define any asset
UNUSED	The signal is not used in the circuit

2.1.4 Internal Assets

Internal assets describe the syntactic information of all signals within the design. Internal assets are primarily automatically assigned by the Structural Checking tool; however there are a few that can be assigned by a user found in Table 8. The three internal assets that a user can assign to signals are *"controllable"*, *"observable"* and *"protected"*. These three internal assets are to provide additional structural information on the signals. The internal assets that are automatically assigned by the Structural Checking tool are given their assignment during the parsing step of the tool.

Table 8: Assignable internal assets

CONTROLLABLE	The signal controls an FSM
OBSERVABLE	The signal is observable after a scan-in operation
PROTECTED	The signal is protected from known attacks

2.1.5 Asset Assignment/Asset Filtering

One very important aspect of the Structural Checking tool is asset assignment. This is a process where a user will launch the tool and assign assets to primary port/generic signals of a design. During this phase the Structural Checking tool will list all components and their

instances starting with the top level and working downwards in a tree like structure as shown in Figure 1.

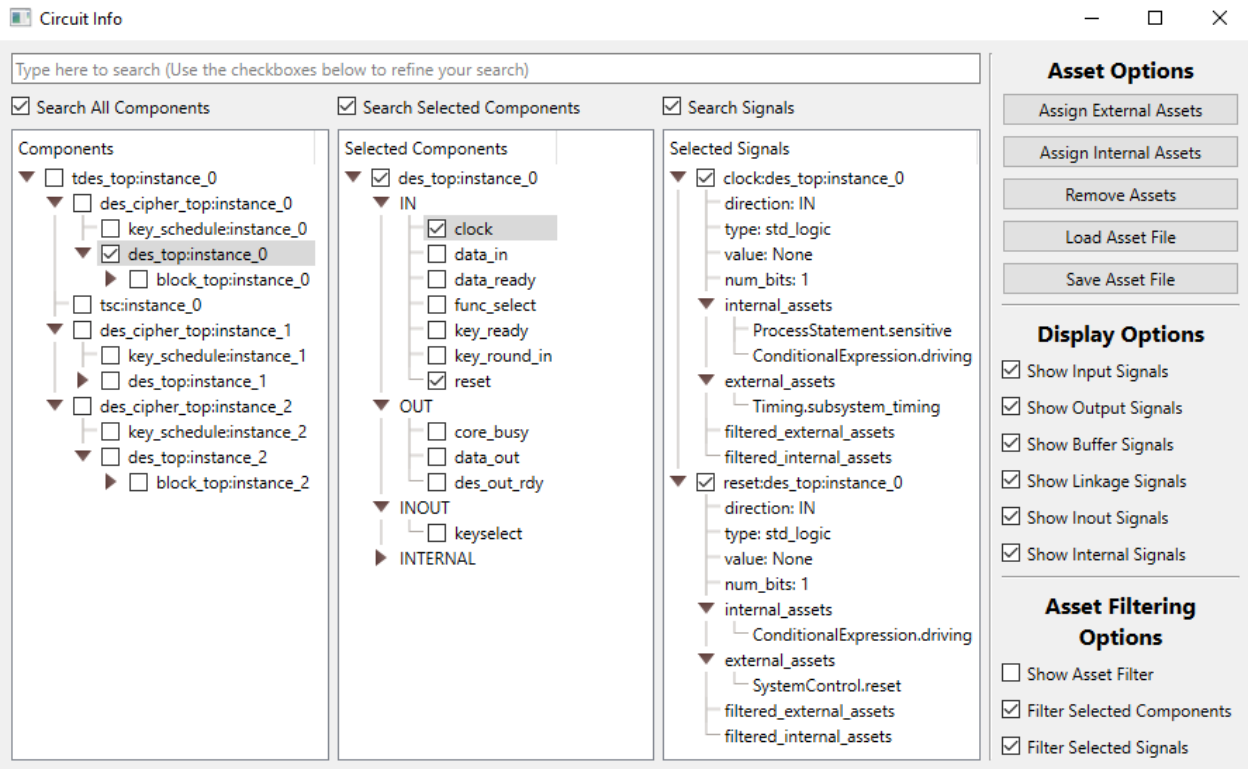

Figure 1: Circuit Info User Interface

From here a user will select the component that they want to work with and begin assigning external or internal assets to the primary port/generic signals of that component instance. During this process a user can also see information the tool has collected on any given signal. The user then has the option to save an asset file, containing all user-defined asset assignments, or they can write those assets to a csv file. Within the csv file, all signals and their assets for each component instance will be listed. CSV files also require the inclusion of functionality assignments. These functionality assignments are made for both the design and each independent component within the design. These functionality assignments are required as CSV files are an integral part of building the GRL.

Asset filtering is another crucial aspect of the Structural Checking tool. The asset filtering step is used to propagate each signal's user-defined asset information across all driving/driven connections. This step is essential in providing a more detailed view into each signal's purpose within the circuit. For example, found in many designs, there is an input signal that will be connected to an output signal by any number of intermediate signals. The assets that were assigned to that input signal will begin to propagate until they reach the output signal moving through as many intermediate signals as needed. This process can be used to enable the detection of conflicting asset assignments or suspicious signals. Once asset filtering has been completed, asset traces will be created for every primary port/generic and internal signal in a design which will contain all assets assigned to it [6]. Additionally, an asset pattern is created for the design. The asset pattern, as described in [10], is the collection of all asset traces for a soft IP. An asset pattern contains six defining characteristics; *"input port signal external asset"*, *"input port signal internal asset"*, *"output port signal external asset"*, *"output port signal internal asset"*, *"internal signal external asset"*, and *"internal signal internal asset"*.

2.1.6 Functionality Assignments and Golden Reference Library (GRL)

Functionality is important as it is a key factor in building the GRL. As briefly discussed in Section 2.1.5, functionality is how designs are sorted within the GRL and used during the matching process to determine possible Trojans. When adding a new design to the GRL the component functionality can be determined by interpreting asset patterns. The asset pattern of an unknown design can be compared to other asset patterns of designs within the GRL to determine what component functionality should be assigned. This component functionality is determined at the beginning of the matching process. Currently within the Structural Checking tool there are eighteen component functionalities where seventeen of them contain a Trojan counterpart. If a

Trojan-infested design is assigned a component functionality, then the functionality has Trojan appended to its name. These Trojan component functionalities can be described the same way as their clean counterparts with the addition of a Trojan in the design. A list of all component functionalities and a brief description can be found in Table 9.

Table 9: Component Functionality

UNASSIGNED	Default functionality used when no other functionality has been assigned
CANNOT_DETERMINE	Functionality used when a device's functionality cannot properly be determined
PROCESSOR	Used to perform operations (usually defined via an instruction set) or an external data source
HARDWARE_ACCELERATOR	Used to quickly perform specific functions to aid in computation
MEMORY	Used to store information for immediate or late use
DATAPATH	Used to perform data processing operations and control the flow of data
POWER_MANAGEMENT	Used to control the distribution and flow of power
CLOCK_GENERATION	Used as the origin of a clock signal for distribution throughout the device
TIMING	Used to configure or process the clocks signal generated by a clock generation functionality device
PWM	Used as a method to reduce average power delivered by an electrical signal by chopping it into discrete parts

Table 9 (Cont.)

CRYPTOGRAPHY	Used to implement cryptographic functions or perform cryptographic operations
COMPUTATIONAL	Used to implement mathematic functions or perform mathematic operations
CONTROL_UNIT	Used to manage operations of other devices
DEBUGGING	Used to display or output internal debugging information
PORT	Used as a physical interface to external devices
COMMUNICATION	Used to facilitate the transfer of data internally or externally over wired or wireless connections between devices
DECODER_ENCODER	Used to either decoder information from a specific format or encode information into a specific format
LIBARARY	A package or library from which components or other signals are loaded from

The GRL is a collection of soft IP designs gathered from Trust-Hub [15], OpenCores [16] and members of the Structural Checking team. Each design has been assigned an overall functionality from Table 9 that best describes the design. Each design in the GRL has a GRL entry file associated with it. This GRL entry file is a JSON file that contains information on the design that is later used in the matching process. As introduced in [14], the golden reference matching process includes two stages. The first stage is champion matching. This process is

where an unknown design is compared against a champion design to determine the unknown design's functionality [7]. These champion designs include a single, representative, design for each functionality within the Structural Checking tool. After comparing the unknown design to all champion designs, the Structural Checking tool will report back the highest functionality match. The second stage during the matching process, functionality matching, occurs after an unknown design has been assigned an approximate functionality based on the results of the champion matching process. This process will then match the unknown design to all designs within the GRL that were assigned the same functionality. This is to learn which design within the GRL best resembles the unknown design. During this process the unknown design is matched against both clean and Trojan-infested designs. Should the unknown design match highest to a Trojan-infested design, it signals that the unknown design may include Trojan logic.

2.2 Hardware Trojan

2.2.1 Overview

Hardware Trojans are a major threat to IC vendors incorporating third-party IP. There are many different Trojan taxonomies that can be defined by two main characteristics of the Trojan. First is how the Trojan is activated, this is known as the Trojan trigger. The second is the specific action that the Trojan performs, this is known as the Trojan payload. The Trojan trigger is the part of a Trojan that activates the Trojan. There are many different types of Trojan triggers that are described in more detail later in this chapter. The Trojan payload is the malicious action that the Trojan performs. There are many different types of Trojan payloads, broken into four main categories, that are discussed later in this chapter. Trojans can be inserted during various stages of the IC design flow; however, the Structural Checking tool only focuses on hardware Trojans that are inserted into an IC design at the RTL.

2.2.2 Trojan Trigger

One major component of a Trojan is the Trojan trigger. As defined in [11], there are two main types of Trojan triggers *"digital"* and *"analog"*. Due to the Structural Checking tool working on RTL designs, it is unable to detect *"analog"* Trojan triggers. This is because *"analog"* Trojan triggers are based on things like on-chip sensors and activity, for example having a Trojan activate when a certain temperature is reached on the chip. On the other hand, *"digital"* Trojan triggers can be detected by the Structural Checking tool. *"Digital"* Trojan triggers can be categorized into two groups *"always on"* and *"conditional"*. The *"always on"* group is for Trojans that are triggered from the moment they have been implemented into a design. The *"conditional"* group can be further defined into two subgroups *"logic"* and *"sensor"*. A common trigger type is known as *"time bombs"*, which are activated by a sequential event occurring or a period of continuous operations occurring. In its simplest form a Trojan triggered by a counter would be considered a *"time bomb"* that would fall into the *"logic"* subgrouping.

2.2.3 Trojan Payload

The specific malicious action performed by a Trojan is considered the Trojan payload. Trust-Hub categorizes Trojan payloads into four main effects; *"transmit information"*, *"modify specification"*, *"change function"*, and *"disable function"*. As defined in [12], the reason a Trojan payload falls into the *"change function"* category is when a Trojan causes the functionality of a target device to change. This renders the device useless as it will be unable to generate the correct result while the Trojan is active. The *"modify specification"* category is for Trojans that affect the performance of the target device in some manner [12]. This Trojan payload can still allow for the correct result to be generated, but at a much lower quality or

speed. The next category, *"transmit information"*, refers to Trojans that will transmit confidential data from the targeted device by any means [12]. This type of Trojan payload can affect the output of the target device by leaking sensitive data. The last Trojan payload category is *"disable function"*. Trojans will fall into this category when the Trojan payload causes the device to stop performing its desired function [12]. This stop in performance could be temporary or permanent depending on the severity of the Trojan.

2.2.4 Trojan Detection

Within the Structural Checking tool there are two Trojan detection methods. The first method is GRL statistical matching, a process that will compare an unknown design against designs within the GRL. This process was described in more detail at the end of Section 2.1.6.

The second Trojan detection method uses structural elements and asset pattern recognition to create Trojan detection functions that can recognize specific Trojan taxonomies. Currently within the Structural Checking tool there are four Trojan detection functions: *"trojan_counter_detection"*, *"trojan_battery_drain_detection"*, *"trojan_key_leak_detection"*, and *"trojan_clock_detection"*. The first function, *"trojan_clock_detection"*, checks a clock/timing signal to determine if there are any possible Trojan signals that could be modifying the clock/timing signals. The second function, *"trojan_key_leak_detection"*, looks for any possible key leakage Trojans by checking if a signal that has been assigned a data *"key"* asset is driving any output signals or any signals that do not contain encryption related assets. The next two Trojan detection functions come from [13], starting with *"trojan_battery_drain_detection"*. This function looks for any possible battery drain Trojans by checking for the following three conditions.

1. If a set of signals consists of a closed loop or cycle. This hints that it is capable of running forever.
2. If the same set of signals is activated by an enable signal. This hints that the enable signal might be a Trojan trigger.
3. If the same set of signals does not have an output. This hints that the cycle does not serve any other purpose but to run endlessly.

The last Trojan detection function is *"trojan_counter_detection"*. This Trojan consists of a counter signal, which can be represented by the *"count"* asset. This function looks for any possible Trojans being triggered by a counter, which can be narrowed by checking signals with the *"count"* asset, then checking if they are driving any signals with specific data assets.

3. METHODOLOGY AND IMPLEMENTATION

3.1 Overview

The Structural Checking tool currently has two methods to detect Trojans: GRL-based statistical matching and structure-based Trojan detection functions. This research expands upon the work done in [13], focusing on the structure-based Trojan detection approach. This research introduces three new Trojan detection functions that utilize structural elements defined within HDL code as well as asset patterns. These Trojans are sensitive data leakage Trojans, data modification Trojans, and denial-of-service Trojans. Each of these Trojan detection functions were designed and implemented to recognize structural elements and asset signifiers that correspond to their respective Trojan taxonomy.

3.2 Trojan Detection Functions

Hardware Trojans can be implemented in many ways inside of soft IPs. Therefore, creating a general-purpose Trojan detection method for all hardware Trojans through the recognition of structural elements and asset patterns would not be feasible. That is why this research expands upon the creation of functions that focus on specific taxonomies to catch Trojans. These Trojan detection algorithms can be beneficial in locating possible Trojan signals in large designs where it would not be feasible to search for Trojan signals manually. The two components of Trojans are traditionally contained in two different sections of the design. The objective of the Trojan detection functions is to analyze the design and identify the sections that include Trojan components to report possible Trojan signals. The function will report signals based on unique structural elements and asset traces that might be more prevalent in Trojan-infested designs.

Depending on the Trojan taxonomy, Trojan signals could be found in either the Trojan trigger component, Trojan payload component or both. Certain Trojan taxonomies could be defined by their Trojan trigger and could have many different Trojan payloads. These Trojan taxonomies can be detected by looking for unique identifiers in the Trojan trigger component of the design. In some cases, a Trojan trigger might not be unique to that specific Trojan taxonomy, in this case it is important to look for unique identifiers in the Trojan payload section of the design. This research attempts to find unique identifiers in both sections of the design for the three Trojan detection functions added. With these unique identifiers a Trojan detection function can be implemented within the Structural Checking tool to detect these specific Trojans.

3.2.1 Sensitive Data Leakage Trojan

Sensitive data leakage Trojans occur when a Trojan payload leaks signals that contain sensitive data to the output of a design. Signals are considered sensitive data signals when the signal contributes to data that should remain confidential to the circuit. Two examples of why signals might be considered sensitive would be if the signal contains proprietary data that the author of the design does not want users to know, or the signal contains some intermediate data that should remain hidden and is vital for the functionality of the design. The leakage of this confidential data is a major threat, and detecting a Trojan that could be leaking this type of data is a positive contribution to the Structural Checking tool's structure-based Trojan detection functions. For the sensitive data leakage detection function, possible Trojan signals will be reported in two categories *"activation_signals"* and *"sensitive_data_leaking_signals"*.

"Activation_signals" correspond to the signals that relate to the Trojan trigger of a sensitive data leakage Trojan payload. The process to detect an activation signal involves first iterating through each signal in a cycle to check if they are being indirectly driven by an

"enable" signal. Since this function's goal is to detect sensitive data leaking from the circuit, the function will only look at cycles that include at least one signal with a *"sensitive"* data asset. Then by iterating through the cycles that contain a *"sensitive"* data asset the detection function will look for signals containing system control asset values that might trigger a Trojan; *"enable"*, *"reset"* or *"flag"*. The next step is to look at the signals that are being driven by these possible Trojan trigger signals to see if they are driving an output signal. If these possible Trojan trigger signals are driving an output signal, then they are added to the *"activation_signals"* category.

The *"sensitive_data_leaking_signals"* category corresponds to the signals that contribute to the Trojan payload of a sensitive data leakage Trojan. Signals can be added to this category in two ways. First, when a possible Trojan activation signal is found it will also report the signal that is being driven by the activation signal as a *"sensitive_data_leaking_signals"*. The second way a signal is added to the category starts by creating a list of the signals a sensitive data signal is driving. This list is then checked for either an output signal, or non-internal signal. If a terminal node is found, it is then flagged as a possible Trojan signal and added to the *"sensitive_data_leaking_signals"*. However, if a non-internal signal is found then that signal is checked to see if it was assigned external asset values of *"encryption"* , *"decryption"* or *"sensitive"*. If so, then that signal will be added to the *"sensitive_data_leaking_signals"* category. This is because if a *"sensitive"* data signal is found then in most cases that signal should drive a signal with asset values of *"encryption"*, *"decryption"* or *"sensitive"* , if it is not driving a signal with those asset values then it is possibly a Trojan signal.

With the implementation of the two separate categories, the detection function can catch possible Trojan signals that apply to the Trojan trigger and payload. This is to catch these Trojan components even when they are contained in separate parts of the design. This implementation

will flag the designs that contain the structural elements and asset patterns that are commonly found in sensitive data leakage Trojans.

3.2.2 Data Modification Trojan

This type of Trojan will modify important data in the design to prevent the design from functioning properly. Data modification Trojans primarily focus their implementations within the payload component of the Trojan. For this reason, many data modification Trojans either contain obscure triggers, which exhibit high variability in their implementation, or have no trigger at all, relying on preset activation intervals. This complexity renders the detection of triggers for this Trojan taxonomy particularly challenging. This Trojan detection function focusses on targeting unique identifiers in the Trojan payload. The use of a data modification Trojan is mainly to reduce system reliability, this is caused by modifying different signals in the design. For example, modifying a key, or other crucial signals in a cryptographic circuit could cause the design to fail. The possible Trojan signals relating to this Trojan taxonomy will appear in the report generated by the Structural Checking tool under the *"data_modification_signals"* category.

Signals are added to the *"data_modification_signals"* category when they contain unique identifiers that have been determined to describe data modification Trojan signals. Those unique identifiers are based on asset patterns and structural elements. An initial scan of all signals takes place before running the Trojan detection function. This scan locates all signals that contain data related assets indicative of the data modification Trojan taxonomy. These assets are as follows: data *"sensitive"*, data *"critical"*, and data *"address"*. After a signal with these data-related assets has been found there are two different ways a signal can be declared as a possible Trojan signal.

The first method will look at the cycle that the signal containing data-related assets is contained in. This cycle will be checked to see if any signals are being indirectly driven by an enable signal. If this is true, it will check to make sure that the signal contains data-related filtered external assets. After that, the function will iterate through the remaining cycles looking for signals that contain system control asset values that might trigger a Trojan; *"enable"*, *"reset"* or *"flag"*. This is to ensure that data *"sensitive"*, data *"critical"*, and data *"address"* that are driving signals with control asset values of *"enable"*, *"reset"* or *"flag"* are flagged as possible Trojan signals. While it is normal to have control assets drive those data related assets, the presence of the opposite is considered Trojan behavior. After all conditions have been checked, if the function deems a signal to be contributing to possible data modification, the signal will be added to the *"data_modification_signals"* category in the possible Trojan report.

The second method by which a signal can be flagged as a possible data modification Trojan signal is as follows. The first step is to find all descendants of the signals that contain data-related assets. This process is done by recursively iterating through each data-related signal's driving connections until an output signal is found. Then each descendant found will be checked to see if they too contain data-related assets. If a descendant signal is found to contain data-related assets, then the internal edges of the signal will be checked to see if any signals driving this descendant also contain data-related assets. This is to make sure that a descendant both contains data-related assets, and a direct ancestor contains data-related assets. This is indicative of a Trojan due to confidential data driving more confidential data that is at risk of being modified. If these conditions are met, it will be added to the *"data_modifciation_signals"* possible Trojan category.

3.2.3 Denial-of-Service Trojan

Denial-of-Service Trojans can be difficult to detect due to having multiple different types of Trojan triggers and Trojan payloads. For this research, when referring to a denial-of-service Trojan, it will be referring to those where the Trojan trigger is activated by a rare sequence of bits within the design. These bits can come from the input directly or indirectly in the form of signal data that is dependent on the input of the design. This type of Trojan trigger was the focus due to it providing the greatest growth to the Structural Checking tool. Currently in the tool there are Trojan detection functions for other types of triggers that can relate to denial-of-service Trojans, for example the counter Trojan detection function and the power drain Trojan detection function. Due to those reasons this Trojan detection function was designed to look for unique identifiers like structural elements and asset patterns for a Trojan trigger that is dependent on a rare sequence of bits in a design. When a possible denial-of-service Trojan signal is found it will be added to the *"denial_of_service_signals"* category in the report generated by the Structural Checking tool.

The Trojan detection function's first step is to check if a signal contains either the system control *"enable"* asset or the system control *"flag"* asset. This is because these assets are likely to be assigned to signals that contain a rare sequence of bits gathered from a larger signal or input value. From here the Trojan detection function will look at descendants of the signal to determine if any descendants contain pertinent assets like system control *"enable"* or system control *"flag"*. The Trojan trigger of a denial-of-service Trojan tends to have multiple enable or flag signals driving each other before activating the Trojan payload. It is this pattern that the function is attempting to detect. Once a descendant is found to contain the pertinent assets then the function will look at the ancestors directly driving the descendant to check that it too contains

the pertinent assets. These pertinent assets are system control *"enable"* and system control *"flag"* assets. If these conditions are met, both the ancestor and its direct descendant will be flagged as possible Trojans.

While it can be difficult to find all possible denial-of-service Trojans, this function will detect a wide range of such Trojans that have been evading the Structural Checking tool's other Trojan detection functions. This function while focusing on the Trojan trigger can still sometimes flag possible Trojan signals that are related to the Trojan payload. This function, when combined with other Trojan detection functions already in the Structural Checking tool, should be able to cover a wide range of Trojans that could cause a denial-of-service attack.

4. RESULTS AND ANALYSIS

To test the Trojan detection functions, designs were created to evaluate the effectiveness of each function. These designs ranged from modified clean HDL designs to complete Trojan infested designs gathered from Open Cores [16] and TrustHub [15], respectively. For each of these designs, a pre-processing step is performed by the Structural Checking tool. The first pre-processing step is to parse all structural information from each HDL file within a given design. This step allows for the extraction of ports, generics, internal signals, component declarations, and much more. After the design has been parsed for all structural information, asset assignment must be performed on all primary signals within the design. Once asset assignment has been performed, the Structural Checking tool performs asset filtering to distribute asset assignments across driving/driven connections to give a more representative view of each signals purpose within the circuit. The next step of the Structural Checking tool is to perform the matching process; however, for this research that step was skipped, as there would be no benefit from analyzing the results of the matching process. This is due to the Trojan detection functions being separate from the matching process, and therefore are not the focus of this research. The final step was then to perform structure-based Trojan detection. During this step, all structure-based Trojan detection algorithms are run to find any possible Trojan signals and report them back to the user. After confirming that the Trojan detection functions can detect the correct Trojan signals in their respective design, an additional script was run to test the Trojan detection functions against other clean designs in the GRL to achieve accurate false positive rates.

4.1 Sensitive Data Leakage Trojan

To evaluate the sensitive data leakage Trojan detection function, a test design was modified from a clean design found on Open Cores [16]. This design is a cryptographic

algorithm that uses triple DES to encrypt data with a sensitive data leakage Trojan added. This Trojan will leak the first cipher text generated after one round of DES. With this leaked partial cipher text, a user could then perform a basic brute force attack to find the plaintext and key information. This attack methodology is common among other cryptographic algorithms making it vital to be able to detect this Trojan via Structural Checking. This design was named *"3des_vhdl_T100"*.

```
if falling_edge(reset) then
    TSC_Trigger <= '1';
end if;
```

Figure 2: Trojan Trigger in top-level

In the top-level design file, the Trojan trigger is activated on the falling edge of the reset signal. This is shown in Figure 2 with *"TSC_Trigger"* acting as the Trojan trigger. Once the Trojan trigger is set to high, additional logic added to the cryptographic circuit in the form of a Trojan-inserted component activates. This Trojan-inserted component, named *"TSC.vhdl"* can be seen in Figure 3. The Trojan-inserted component takes in two input signals and returns a single output; those signals are *"indata"*, *"trigger"*, and *"outdata"*, respectively. The port map showing how these signals are connected to the top-level is shown in Figure 4. Inside the Trojan-inserted component is the *"indata"* signal, which is the result of the first round of DES. *"indata"* is then pushed to drive the *"outdata"* signal, which is the output of the top-level design. This causes the design to output the results of the first round of DES once the Trojan is triggered.

```
entity TSC is
Port
    (
        indata: in std_logic_vector(0 to 63);
        outdata: out std_logic_vector(0 to 63);
        trigger: in std_logic
    );
end TSC;

architecture Behavioral of TSC is

begin
    process (trigger)
    begin
        if (trigger = '1') then
            outdata <= indata;
            trigger <= '0';
        end if;
    end process;
end Behavioral;
```

Figure 3: TSC.vhd

```
Trojan: TSC
port map
(
    indata => data_out_internal1,
    outdata => data_out_internal,
    trigger => TSC_Trigger
);
```

Figure 4: Port map for TSC.vhd

After the Trojan-inserted test design was created and validated via simulation, it was processed through the Structural Checking tool. During the Trojan detection section of this process, the Structural Checking tool generated a report that is as follows:

```
"sensitive_data_leakage_trojans": {
    "sensitive_data_leaking_signals": [
        "trigger:tsc:instance_0",
        "outdata:tsc:instance_0"
    ],
    "activation_signals": [
        "tsc_trigger:tdes_top:instance_0",
        "trigger:tsc:instance_0"
    ]
}
```

Figure 5: Results of sensitive data leakage Trojan detection function

From the results in Figure 5, three different possible Trojan signals can be seen; *"trigger"*, *"tsc_trigger"*, and *"outdata"*. The *"trigger"* and *"tsc_trigger"* signals are listed as activation signals due to their relation to the Trojan trigger. This is because those signals have system control asset values and are driving sensitive data signals to an output. *"outdata"* is listed as a sensitive data leaking signal because it relates to the Trojan payload. This is because *"outdata"* contains sensitive data assets and is positioned on the outer edge of a cycle. *"trigger"* also happens to be listed as a sensitive data leaking signal; this is because this signal is driving the *"outdata"* signal and is helping leak sensitive data.

From testing this Trojan detection function against all clean designs in the GRL, it was found that this detection function has a mean false positive rate of 3.8%. From those designs, when looking at only the designs reporting possible Trojan signals, this function has a mean false positive rate of 7.4%. This removes the clean designs that report 0 Trojans and gives a more detailed view of false positives by only looking at designs that are falsely reporting Trojan signals. Overall, the false positive rates of this function are low and reasonable for any end user.

4.2 Data Modification Trojan

To evaluate the data modification Trojan detection function, a test design was created starting with a clean design found on Open Cores [16]. This design is a cryptographic algorithm that uses triple DES to encrypt data with a data modification Trojan added. This Trojan will modify the first ciphertext generated after one round of DES. With this modification of the ciphertext, the original plain text will be lost, because the next two rounds of DES will be performed on this new modified ciphertext. When a user attempts to decrypt ciphertext generated by this design, it cannot be decrypted back to the original plaintext and instead returns some

arbitrary combination of bits, thus rendering the triple DES VHDL design unusable. This design is renamed *"3des_vhdl_T200"*.

```
if falling_edge(reset) then
    TSC_Trigger <= '1';
end if;
```

Figure 6: Trojan Trigger in top-level

In the top-level of the design, the Trojan trigger signal, *"TSC_Trigger"*, is activated on the falling edge of the reset signal. Once the Trojan trigger is set to high, the modifications to the design in the Trojan-inserted component activates. The Trojan-inserted component, named *"TSC.vhd"*, can be seen in Figure 8. The Trojan-inserted component takes in three input signals and returns a single output. Those signals are *"indata"*, *"trigger"*, *"clk"*, and *"outdata"*, respectively. The port map of how the signals connect to the top-level is shown in Figure 7. The Trojan-inserted component will check every clock cycle to see if the Trojan trigger has been activated. If the Trojan trigger is activated, some arbitrary value is added to the *"indata"* signal, which is the result of the first round of DES. If the Trojan trigger is not activated, then the Trojan-infested component will output the *"outdata"* signal, and the triple DES will continue to work as normal, sending the results to the next round of DES. This now causes the design to generate an incorrect ciphertext that, when attempting to decrypt, will not result in the expected plaintext.

```
Trojan: TSC
port map
(
    indata => data_out_internal1,
    outdata => data_out_internal1_modified,
    trigger => TSC_Trigger,
    clk => clock
);
```

Figure 7: Port map for TSC.vhd

```
entity TSC is
    Port
    (
        indata: in std_logic_vector(0 to 63);
        outdata: out std_logic_vector(0 to 63);
        trigger: in std_logic;
        clk: in std_logic
    );
end TSC;

architecture Behavioral of TSC is

begin
    process (clk, trigger)
    begin
        if (trigger = '1') then
            outdata <= indata + "10101010101010101010101010101010101010101010101010101010101010";
            trigger <= '0';
        else
            outdata <= indata;
        end if;
    end process;
end Behavioral;
```

Figure 8: TSC.vhd

The next step was to process this new Trojan infested design through the Structural Checking tool. After parsing the design and performing asset assignment, the Trojan detection report was returned.

```
"data_modification_signals": [
    "tsc_trigger:tdes_top:instance_0",
    "data_out_internal1_modified:tdes_top:instance_0",
    "trigger:tsc:instance_0"
],
```

Figure 9: Results of data modification Trojan detection function

Looking at the results in Figure 9, three different possible Trojan signals are reported. *"tsc_trigger"* and *"trigger"* are important due to these signals relating to the Trojan trigger. These signals contain system control asset values and drive other possible data modification signals. *"data_out_internal1_modified"* is the Trojan payload. It was flagged because it was assigned either a sensitive data or decryption data asset, and was positioned on the outer edges of a cycle.

For this design, the mean false positive rate, when looking at all clean designs in the GRL, is 2%. When looking at just the designs reporting any possible Trojan signal, the mean

false positive rate increases to 3.89%. This occurs after removing those designs with a false positive rate of 0%. Overall, this Trojan detection function also results in a low false positive rate.

4.3 Denial-of-Service Trojan

The denial-of-service Trojan was evaluated by using a Trojan infested test design that was found on Trust Hub [15]. This Verilog design, named *"AES_T2400"*, is an implementation of the AES cryptographic core with a denial-of-service Trojan inserted. This Trojan is activated when the 89th and 127th bit of two states are set to '*1*' causing the first bit of the output to become flipped. This causes the design to produce the wrong output when generating ciphertexts and causes the design to fail.

```
module TSC(
    input r1,
    input r2,
    output trigger
    );

    assign trigger = r1 && r2;


    always @ (posedge trigger)  $display("Trojan Activated");

endmodule
```

Figure 10: TSC_and.v

```
final_round
    rf (clk, s9, k9b, out2);

TSC Trojan(s2[89], s7[127], Tj_Trig);

assign out[127:1] = out2[127:1];
assign out[0] = out2[0] ^ Tj_Trig;
```

Figure 11: Trojan trigger in top-level

As shown in Figure 11, *"s2[89]"* and *"s7[127]"* are intermediate state values that are dependent on the rounds of AES. *"s2"* is the result after round two and *"s7"* is the result of round seven. The statement *"r1 && r2"* in the Trojan-infested component, shown in Figure 10 is true when *"s2[89]"* and *"s7[127]"* are *'1'*, thus activating the Trojan trigger. The Trojan trigger signal in the top-level then becomes *'1'* causing the bitwise XOR operation shown in Figure 11 to negate the correct output.

The next step was to process this Trojan infested design through the Structural Checking tool. After parsing the design and performing asset assignment the Trojan detection report was returned. Three possible Trojan signals, as shown in Figure 12, can be seen within the report.

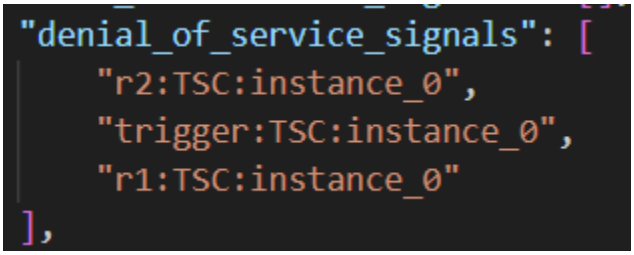

Figure 12: Results of denial-of-service Trojan detection function

The three signals shown in the report from Figure 12 are *"r1"*, *"r2"*, and *"trigger"*. The *"r1"* and *"r2"* signals drive the Trojan trigger and were flagged because they contained system control asset values of *"enable"* and *"flag"* while driving other system control asset signals. *"trigger"* is the output of the Trojan-infested component module and is related to the Trojan payload. This signal is shown in the report due to it being a system control enable signal that is being driven by other enable or flag signals while it itself is driving an output signal. While this did catch three Trojan signals, it failed to catch *"Tj_Trig"*. However, *"trigger"* is directly driving *"Tj_Trig"* and a user should be able to see that connection and inspect the *"Tj_Trig"* signal to determine if it is a Trojan signal.

When running this design against all the clean designs in the GRL the mean false positive rate is 0.6%. When removing the designs that did not report any possible Trojan signals the mean false positive rate increases to 1.17%. Again, this Trojan detection function results in a low false positive rate.

5. CONCLUSION AND FUTURE WORK

The Structural Checking tool currently offers two methods to detect Trojans: GRL-based statistical matching and structure-based Trojan detection functions. GRL-based statistical matching first determines an unknown design's approximate functionality via champion matching, then determines the presence of malicious logic by comparing the unknown designs against all designs of that functionality. Structure-based Trojan detection algorithms utilize structural elements defined within HDL code as well as asset signifiers to detect and report possible Trojan signals. This research was an expansion of the structure-based Trojan detection functions currently offered by the Structural Checking tool. By adding three new Trojan detection functions, it has now expanded the Trojans that can be detected. Hardware Trojans are a major threat to IC vendors incorporating third-party IP and it is for this reason that Trojan detection methods, such as those discussed in this research, are essential.

While the new Trojan detection functions do have low false positive rates, they can still be improved, as there are some outlier designs that yield somewhat high false positive rates. From inspecting the results of false positive testing, it seems that designs with control unit functionalities have a higher false positive rate than others. Additionally, the inclusion of more designs with different implementations of these Trojan taxonomies could allow for further improvement of the Trojan detection functions.

6. REFERENCES

[1]. Y. Alkabani, F. Koushanfar, “Extended Abstract: Designers Hardware Trojan Horse”, IEEE International Workshop Hardware-Oriented Security and Trust (HOST08), IEEE CS Press 2008, pp. 82-83.

[2]. T. Han, Y. Wang and P. Liu, "Hardware Trojans Detection at Register Transfer Level Based on Machine Learning," 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1-5, doi: 10.1109/ISCAS.2019.8702479.

[3]. T. Hu, L. Wu, X. Zhang, Y. Yin and Y. Yang, "Hardware Trojan Detection Combine with Machine Learning: an SVM-based Detection Approach," 2019 IEEE 13th International Conference on Anti-counterfeiting, Security, and Identification (ASID), 2019, pp. 202-206, doi: 10.1109/ICASID.2019.8924992.

[4]. J. Francq and F. Frick, "Introduction to hardware Trojan detection methods," *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2015, pp. 770-775, doi: 10.7873/DATE.2015.1101.

[5]. S. C. Smith and J. Di, "Detecting Malicious Logic Through Structural Checking," 2007 IEEE Region 5 Technical Conference, 2007, pp. 217-222, doi: 10.1109/TPSD.2007.4380384

[6]. B. McGeehan, F. Smith, T. Le, H. Nauman and J. Di, "Hardware IP Classification through Weighted Characteristics," 2019 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2019, pp. 1-6, doi: 10.1109/HPEC.2019.8916225

[7]. N. Waller, H. Nauman, D. Taylor, R. Del Carmen and J. Di, "Character Reassignment for Hardware Trojan Detection," *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Lansing, MI, USA, 2021, pp. 861-864, doi: 10.1109/MWSCAS47672.2021.9531813.

[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]. T. Le, L. Weaver, J. Di, S. Zhang and Y. Jin, "Hardware Trojan Detection and Functionality Determination for Soft IPs," 2018 IEEE 3rd International Verification and Security Workshop (IVSW), 2018, pp. 56-61, doi: 10.1109/IVSW.2018.8494891.

[10]. L. Weaver, T. Le and J. Di, "Golden Reference Library Matching of Structural Checking for securing soft IPs," SoutheastCon 2016, 2016, pp. 1-7, doi: 10.1109/SECON.2016.7506737.

[11]. R. S. Chakraborty, S. Narasimhan and S. Bhunia, "Hardware Trojan: Threats and emerging solutions," 2009 IEEE International High Level Design Validation and Test Workshop, 2009, pp. 166-171, doi: 10.1109/HLDVT.2009.5340158.

[12]. K. Inaba, T. Yoneda, T. Kanamoto, A. Kurokawa and M. Imai, "Hardware Trojan Insertion and Detection in Asynchronous Circuits," 2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2019, pp. 134-143, doi: 10.1109/ASYNC.2019.00025.

[13]. R. Del Carmen (2022). Framework of Hardware Trojan Detection Leveraging Structural Checking Tool. *Graduate Theses and Dissertations* Retrieved from https://scholarworks.uark.edu/etd/4462/

[14]. N. Waller (2021). Characteristic Reassignment for Hardware Trojan Detection. *Graduate Theses and Dissertations* Retrieved from https://scholarworks.uark.edu/etd/3988/

[15]. Trust-Hub. Available: https://www.trust-hub.org/

[16]. OpenCores. Available: http://opencores.org/