5-2022

Framework of Hardware Trojan Detection Leveraging Structural Checking Tool

Rafael Dacanay Del Carmen
University of Arkansas, Fayetteville

Citation

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

Framework of Hardware Trojan Detection
Leveraging Structural Checking Tool

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

by

Rafael Del Carmen
University of Arkansas
Bachelor of Science in Computer Engineering, 2020

May 2022
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

____
Jia Di, Ph.D.
Thesis Director

______________________________	______________________________
John Gauch, Ph.D.	Brajendra Panda, Ph.D.
Committee Member	Committee Member

ABSTRACT

Since there is a significant demand for obtaining third-party soft Intellectual Property (IP) by first-party integrated circuit (IC) vendors, it is becoming easier for adversaries to insert malicious logic known as hardware Trojans into designs. Due to this, vendors need to find ways to screen the third-party IPs for possible security threats and then mitigate them. The development of the Structural Checking (SC) tool provides a solution to this issue. This tool analyzes the structure of an unknown soft IP design and creates a network of all the signals within the design and how they are connected to each other. In addition, these signals will be assigned with assets. Assets describe the central role of a signal in the entire design. These assets are then used to create asset patterns, which will be crucial for this thesis research. Previous research on SC tool focuses on Trojan detection by comparing and matching an unknown design to a trusted design in a Golden Reference Library. In this thesis research, another method of Trojan detection has been implemented in the SC tool, which focuses on recognizing specific asset patterns that mainly exist in Trojan-infested designs. These specific asset patterns can then be used to check against unknown designs for Trojans without using a Golden Reference Library. This thesis improves this method by creating a new framework for easily identifying the unique Trojan asset patterns.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Jia Di, for supporting me with this research. He has helped me in many ways during my graduate degree and he has guided me to become a better person.

I would like to thank my Structural Checking group members, especially Hunter Nauman, Derek Taylor, and Noah Waller. They have always been helpful to me.

Lastly, thank you to my other committee members, Dr. John Gauch and Dr. Brajendra Panda. I appreciate your time for accepting and serving as my committee member.

DEDICATION

To my former teachers, counselors, and other faculty of Little Rock Hall High School. They are the ones who believed in me and motivated me to become a successful engineering student and future Computer Engineer. Without their encouragement, I would be pursuing a different career right now.

CONTENTS

1. INTRODUCTION

Hardware Trojans can be inserted in different stages of integrated circuit (IC) production. One stage of IC production that many researchers focus on is the design stage. Nowadays, it is getting more difficult for an IC vendor to design every IC component in house. Because of this, many third parties are given the responsibility for designing certain components known as intellectual properties (IPs). However, an issue that can arise from letting third parties design components is whether they are trustable. Due to the vast number of such components, malicious logic or hardware Trojan can be easily hidden in any of the components.

There have been multiple research efforts about Trojan detection in various stages of IC production. One approach utilizes machine learning to detect hardware Trojans at the register-transfer level [1]. There is also research about detecting Trojans by measuring combinational delays to check whether the designs got altered [2]. Some researchers [3] used randomization-based probabilistic techniques to verify the legitimacy of the circuit design.

The Structural Checking Tool (SC Tool) is another research on hardware Trojan Detection. The SC Tool was first introduced in [4]. Unlike other hardware Trojan detection approaches, the SC Tool focuses on analyzing the structure of a register-transfer level (RTL) IPs and then matches against entries in a Golden Reference Library (GRL) to see whether they match a known design with Trojan or not.

The next chapter of this thesis provides all the relevant background information for the SC Tool. It provides important details of assets and describes the basic concept of the GRL. Moreover, this chapter also discusses hardware Trojans that can be inserted during the design phase of IC production. Chapter 3 proposes an improvement of the tool’s Trojan detection by focusing on the unique asset pattern method of Trojan detection. Additionally, this chapter introduces three

Trojans that can be detected using the asset pattern method. Chapter 4 discusses the results of implementing the detection for the three types of Trojans mentioned in Chapter 3. Lastly, Chapter 5 summarizes the thesis and provides insights on how the research done in this thesis can be improved.

2. BACKGROUND

2.1 Structural Checking Tool

2.1.1 Overview

The goal of the SC tool is to determine whether an unknown design contains hardware Trojans or not. The tool takes in unknown soft IP designs, such as in VHDL, as inputs, and then outputs a report regarding its Trojan status determination. The Structural Checking tool consists of five main steps, i.e., 1) design parsing, 2) asset assignment, 3) asset filtering, 4) matching analysis, and 5) Trojan detection.

2.1.2 Assets

The concept of assets is an integral part of the Structural Checking tool. Assets were introduced in [5]. The function of an asset is to describe the primary purpose or contribution of a signal in a soft IP design. A signal can have multiple purposes; therefore, multiple assets may also be assigned to the same signal. Depending on the asset type, assets can be manually assigned by the user or automatically by the tool. In addition, asset assignment for a signal may not be the same in every design since signals may serve different purposes in other designs. The two major types of assets are External assets and Internal assets. Moreover, assets are divided into various categories under each of those major types.

2.1.3 External Assets

An external asset is one of the two major types of assets that are primarily assigned to describe the contribution of primary port signals of an RTL design. The user of the Structural Checking tool manually assigns this type of asset. For example, a soft IP design that takes in a clock signal as one of its primary inputs, that signal can be assigned a SYSTEM TIMING asset. Currently, there are 75 external assets which are divided into seven categories. The number of external assets is expected to increase as more signal roles are identified in the future. A past version of external assets is listed in [6]. Tables 1-7 contain a list of all the current iterations of assets divided into their respective categories.

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.
DECODING	The signal contains information about data used within a decoding process.
ENCODING	The signal contains information about data used within an encoding process.
ADDRESS	The signal controls address 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.

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
SYSTEM_TIMING	The signal controls timing for the entire system, such as timing between synchronous components of the circuit
SUBSYSTEM_TIMING	The signal controls timing for a particular subsystem
STATUS	The signal indicates the status of the system
READY	The signal indicates whether or not an operation is ready
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 indicated an operation with a state of readiness without being immediately involved
TEST_CLOCK	The TCK (Test Clock) signal of the circuit

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

Table 3 (Cont.)

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

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

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
DESTINATION	This signal describes the destination for data output by an instruction or the destination 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
PIPELINE_LOCK	This signal locks the instruction pipeline

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

Table 7: Miscellaneous asset category

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

2.1.4 Internal Assets

The second primary type of asset is the Internal asset. Internal assets are mainly assigned to internal signals as opposed to external assets. In addition, most internal assets are automatically assigned by the tool. However, the tool also allows users to manually set three assets: *Observable*, *Controllable*, and *Protected*.

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 Filtering, Asset Trace, and Asset Pattern

After all assets are assigned to primary port signals and internal signals, the next step for the tool is asset filtering. In this step, assets assigned to a signal will propagate to every signal it is connected to. For example, suppose a primary input signal is connected to a primary output signal through a series of intermediate signals. In that case, all assets assigned for that primary input will also propagate to the primary output and all the intermediate signals. This allows the tool to determine how the signals are associated with each other and if they share some similar functionalities. These assets are saved into an asset trace. An asset trace is a set of assets that were assigned or filtered into a signal.

An asset pattern is generated by the tool, and it consists of all the asset traces formed within that design. Asset Pattern consists of six 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 and Golden Reference Library (GRL)

The formation of an asset pattern is used to approximate the overall functionality of the unknown design as well as the functionalities of each of its components. The information stored in the asset pattern will be compared to other asset patterns in each entry in the GRL during matching analysis, which will be explained later. Suppose there is a GRL entry that matches the closest to the unknown design. In that case, the functionality of that GRL entry will be given as the suggested functionality of the unknown design. Currently, there are 18 functionalities in the Structural Checking Tool, and they are categorized as either Trojan-free or Trojan-infested functionalities. The main difference between the two is that the Trojan-infested category contains functionalities that are commonly found in designs with Trojans.

The Golden Reference Library is a library of trusted and analyzed soft IP designs that are either Trojan-free or Trojan-infested. Each GRL entry is a single file currently stored as a JSON file. In addition, A GRL entry contains both the functionalities and the asset pattern of the soft IP design. The information stored in a GRL entry will be helpful for the matching analysis step.

2.1.7 Golden Reference Matching

Golden reference matching was first developed in [7]. The matching process for the Golden Reference Library includes algorithms such as basic matching and partial matching. Basic matching uses each asset characteristic formed from the unknown design's asset traces and creates a percentage match against the designs from the GRL. However, since there are assets that might be associated with or similar to other assets, partial matching is developed to take that into account. Slightly similar assets can be assigned a 50% match instead of 100%. Statistical matching, introduced in [8], is developed to create a more accurate percentage match than both basic and partial matches. Moreover, Champion GRL and Functionality GRL were added in [9], which further improves the matching accuracy while also making the matching process more efficient in computational resources.

2.2 Hardware Trojan

2.2.1 Overview

Hardware Trojans can be inserted in various stages of IC design flow. There are many existing strategies for detecting these Trojans in every stage. However, the Structural Checking tool only focuses on analyzing hardware Trojans inserted into a design at the register-transfer level. [10] proposes a framework for classifying hardware Trojans. In addition, the framework includes various categories and subcategories that can characterize a Trojan. However, the only categories relevant to the Structural Checking tool are the Trojan activation and Trojan action. In this thesis, these two categories can also be called a Trojan trigger and a Trojan payload. Currently, there are two main ways hardware Trojans can be detected with the tool [11]. The first method uses matching analysis, while the second method uses assets and asset traces to analyze patterns from certain hardware Trojans. This thesis work focuses on Trojan detection using the second method.

2.2.2 Trojan Trigger

An essential component of a hardware Trojan is how they are activated. Classifying Trojan triggers were further explored in [12], where an expanded Trojan taxonomy was proposed based on [13], and Trojan triggers were classified into different types. In this taxonomy, Trojan triggers can be broken down between *digitally triggered* or *physically triggered*. The Structural Checking tool can only detect digitally triggered Trojans as the tool analyzes Trojans in RTL designs. One subtype of *digitally triggered trojan* that can be detected with the tool is called a time-bomb or *sequentially triggered Trojans,* according to [12]. An example of this is a counter that operates like a regular counter, but once the counter reaches a specific value, the Trojan will be activated.

2.2.3 Trojan Payload

The malicious action being made by the Trojan is called the Trojan payload. According to [15], the hardware Trojan payload can be broken down into four different types: *Denial-of-service*, *leak information*, *change the functionality*, and *degraded performance*. As described by [14], a Trojan with a *denial-of-Service* payload prohibits a service from working correctly. A Trojan with *leak information* leaks sensitive information through covert channels or overt output interfaces, like an encryption key. Additionally, a *change functionality* payload manipulates the original functionality of the target device to a malicious functionality. Lastly, [14] describes the *degraded performance* as a Trojan payload that affects the performance of a device by modifying parameters.

2.2.4 Trojan Detection

There are two methods for detecting possible Trojan-infested designs using the Structural Checking tool, as elaborated in [7]. The first method for detecting Trojans is through functionality matching. In this method, the unknown design is matched through each champion file [9] listed in a GRL to assign its functionality. After functionality is set, the unknown design will match against each design in the champion's functionality category. If it matches closest to a GRL entry flagged with a Trojan, this design will be marked as possible Trojan-infested. The second method is through asset pattern recognition. As asset patterns consist of connections between signals and their functionalities, they can yield helpful information that can be used to suggest possible Trojans.

[11] introduced Trojan detection using the second method. The first Trojan involved using a timing signal as a Trojan trigger. This Trojan can cause denial-of-service as signals, like *set* or *reset,* can disable pertinent timing signals from a synchronous design. To detect this Trojan, the asset trace of a timing signal was analyzed. If the asset trace contains SET or RESET assets, it can be flagged as a possible Trojan. In this thesis, the second Trojan was called a key leak Trojan. This Trojan utilizes a *leaking information* payload, which, in this case, is an encryption key. An encryption key is not meant to be connected to a primary output of the design. Therefore, to detect this type of Trojan, the tool can identify asset traces that contain the key asset. If that same asset trace is also connected to an output, that can be flagged as a key leak Trojan.

3. METHODOLOGY AND IMPLEMENTATION

3.1 Overview

The current iteration of the tool mainly relies on the first method of Trojan detection mentioned earlier, which is through matching. This thesis research addresses this issue by extending the work done in [11], which utilizes the first method of Trojan detection that relies on asset pattern recognition. This thesis introduces a framework for this method of Trojan detection. Three Trojans, named Counter Trojan, Power Drain Trojan, and FSM Trojan, respectively, are implemented and detected using this method.

3.2 Trojan Detection Through Asset Pattern Framework

There are many ways a hardware Trojan can be implemented in soft IPs. Thus, implementing a Trojan detection for all of them through the use of asset patterns can be difficult and tedious. It is impractical to analyze every signal's asset traces in a design. Creating a framework can be beneficial for tackling this issue. This framework consists of locating the Trojan in an unknown design first. A Trojan-infested design does not necessarily mean that most circuits of the design consist of Trojan. Instead, only a tiny portion of the design contains Trojan in order for it to remain hidden. The hidden Trojan is usually split between a Trojan trigger and a Trojan payload hidden in separate sections of a design, but sometimes they are also integrated. The sections of the designs that contain Trojan trigger and payload will then be analyzed for possible unique asset traces that might only exist in Trojan-infested designs. Depending on the type of Trojan, these unique asset traces may only be found in either the Trojan trigger section or the Trojan payload section of the design. If the Trojan is designed in a way that it can provide different types of payloads, then a unique asset trace may not be found in the Trojan payload section of that design. This is when it is recommended to analyze the Trojan trigger section of the design instead.

On the other hand, if the Trojan trigger is not unique, then it is recommended to look for the unique Trojan asset trace in the Trojan payload section of the design. After identifying these unique asset patterns, the developer of the SC tool can then add an implementation within the SC tool that detects these specific asset traces.

3.2.1 Counter Trojan

This Trojan utilizes a counter to deliver its payload. A counter, in general, can be implemented in multiple ways. A counter can also increment or decrement in different numbers. However, no matter how a counter is implemented, they all share one similar aspect: they use a signal that holds the current value of the counter. Usually, if this signal reaches a specific value, the Trojan payload will be activated. This Trojan can be called a Counter Trojan in this thesis. To detect this type of Trojan using the asset pattern method, the SC tool can look for signals containing COUNT assets. This type of asset is only used for signals being used in a counter. Unfortunately, searching for a counter signal may not be enough to detect a Trojan since the presence of a counter, in general, is not considered suspicious.

However, many Trojans that utilize counters usually leak essential information out of the design, such as an encryption key, or modify a signal that holds sensitive data, such as addresses and memory data. To distinguish this, the tool can then analyze if a signal with the COUNT asset drives any signals with data-related assets. These data-related assets can be DATA ENCRYPTION, DATA MEMORY, DATA SENSITIVE, DATA ADDRESSES, or DATA KEY. This implementation may not guarantee that all signals in these asset patterns will always accurately detect a Trojan presence. However, it can flag all the designs that hold these conditions.

3.2.2 Power Drain Trojan

This type of Trojan performs a *denial-of-service* attack on a device by activating an enable signal on a Trojan-infested sequential circuit within the design. A sequential circuit can get stuck on an infinite loop if it is implemented in a certain way and then enabled. As a result, this trojan can consume power indefinitely. In this type of Trojan, the sequential circuit is the Trojan payload. In this thesis, this Trojan is called a Power Drain Trojan. Unlike the Counter Trojan, the unique asset trace exclusive to this Trojan can only be found in the Trojan payload section of the design. The trigger for this type of Trojan is not unique and can be implemented in different ways. To detect this Trojan, the tool can check for three conditions. The first condition is to check whether a signal or set of signals consists of a loop or cycle. This is the first indication that this circuit is capable of running indefinitely. The second condition is to check if an enable signal activates the same set of signals. Enable signals can be labeled using ENABLE asset. The enable signal suggests that it could be the Trojan trigger. Lastly, the third condition is to check if it does not have any output. This hints that the cycle may not serve any other purpose than running endlessly. Once all these conditions are met, then it is a good indication that the unknown design contains a Power Drain Trojan.

3.2.3 FSM-Triggered Trojan

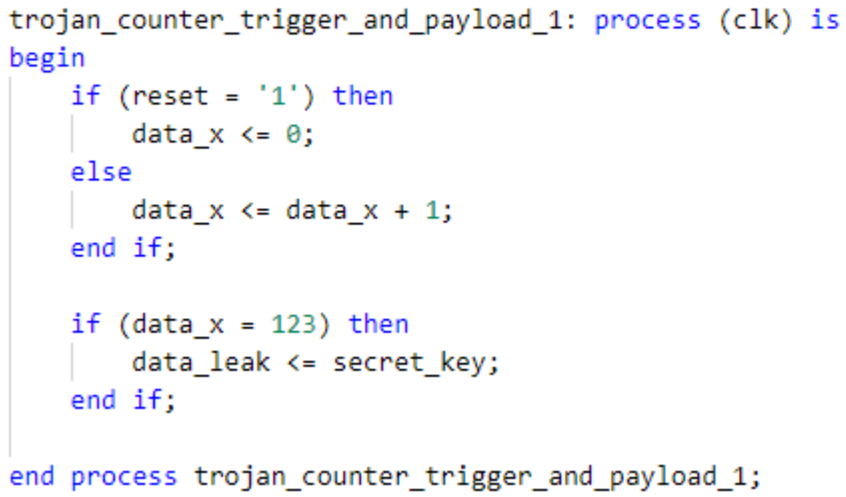
Adversaries can hide a Trojan inside a Finite State Machine or FSM. This hidden Trojan can only be executed when the FSM transitions to a specific state activated with a particular condition. Attackers can insert specific input data in one of the primary input ports of the design. If the FSM observes this particular input value, it will activate the Trojan-infested state. This type of Trojan is called an FSM-Triggered Trojan. Similar to the Counter Trojan, the unique asset trace for this Trojan can be found in the Trojan trigger section of the design, which in this case is a specific state of the FSM. Usually, only a particular state of the FSM is infested with a Trojan, and other states of the FSM can still function normally. This Trojan should contain a signal with the CONTROLLABLE asset. The CONTROLLABLE asset is one of the internal assets that users can assign to a signal being used in an FSM. In addition, if another signal with a DATA asset is indirectly driving this signal, this would suggest that a specific input data is activating this FSM signal. The tool will flag this as a potential FSM-Triggered Trojan if these conditions are met.

4. RESULTS AND ANALYSIS

Test designs written in VHDL were used as inputs to the tool to evaluate the effectiveness of the framework in detecting the three new Trojans. Each design is processed by the SC tool. First, they were parsed by the tool to extract all the relevant signal information from the designs. Second, a combination of manual and automatic asset assignments was done for all primary signals for each design. Third, asset filtering was done to filter the asset from each signal into the signals that it is connected to. Lastly, the matching process was skipped in this experiment since this research focuses on the use of asset pattern Trojan detection. The last step is to start the Trojan detection using all the asset pattern information collected from the designs. If the tool detects a possible Trojan, a list of signals associated with the Trojan will be reported.

4.1 Counter Trojans

To evaluate the implementation of the Trojan detection against the Counter Trojan, a test design written in VHDL was created from scratch. The test design contains multiple implementations of counters to evaluate the accuracy of the Trojan detection. Figure 1 below shows a VHDL process block that represents a simple counter.



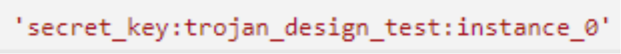
```
trojan_counter_trigger_and_payload_1: process (clk) is
begin
    if (reset = '1') then
        data_x <= 0;
    else
        data_x <= data_x + 1;
    end if;

    if (data_x = 123) then
        data_leak <= secret_key;
    end if;

end process trojan_counter_trigger_and_payload_1;
```

Figure 1: Simple counter with Trojan

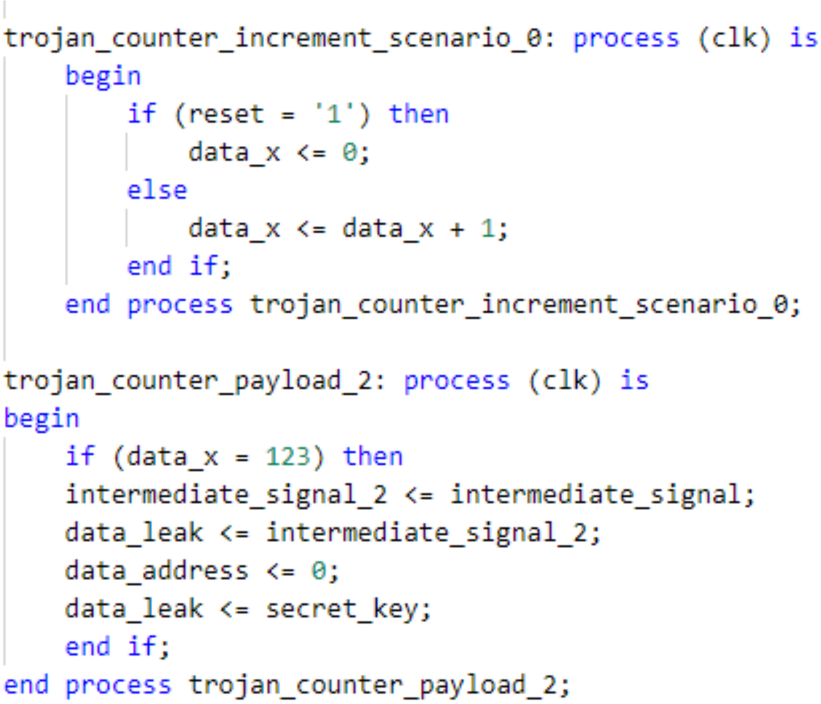
In this counter, the *data_x* signal is a primary input, while the *data_leak* signal is a primary output. *data_x* will keep incrementing by one until it is reset. The Trojan is inserted during the line *“data_leak <= secret_key;."* This Trojan is activated when the value of *data_x* reaches “123”. Since *data_leak* is a primary output and is set with the value of the secret key, it is considered suspicious, and the tool flags this as a possible Trojan. Figure 2 lists the signal that got flagged by the tool as a potential Trojan. The word before the first colon specifies the signal name and the second word after the first colon specifies which design file this signal is located.



```
'secret_key:trojan_design_test:instance_0'
```

Figure 2: Signal associated with Trojan

Some adversaries might even attempt to hide a Counter Trojan by making the counter more complex. Figure 3 below shows a counter that is split into two process blocks.

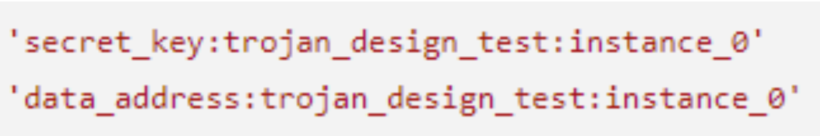


```
trojan_counter_increment_scenario_0: process (clk) is
    begin
        if (reset = '1') then
            data_x <= 0;
        else
            data_x <= data_x + 1;
        end if;
    end process trojan_counter_increment_scenario_0;

trojan_counter_payload_2: process (clk) is
begin
    if (data_x = 123) then
    intermediate_signal_2 <= intermediate_signal;
    data_leak <= intermediate_signal_2;
    data_address <= 0;
    data_leak <= secret_key;
    end if;
end process trojan_counter_payload_2;
```

Figure 3: Another implementation of counter

This implementation of a counter is separated into two different process blocks. The first process block consists of the counter signal *data_x* being incremented, while the second process block contains the Trojan payload. In addition, two intermediate signals were added between the counter signal and the primary output signal *data_leak.* The tool was still able to detect this implementation of the Counter Trojan since it relies on asset patterns. No matter where the signals are placed in the design nor if they put as many intermediate signals as they can to separate the distance between the counter signal and the output signal, the asset trace of the input signal will still show that it is driving an output signal that contains a DATA ADDRESS asset. Figure 4 shows the two signals that got flagged for potential Trojan. Besides secret keys, signals that hold address data are not usually connected to a primary output.

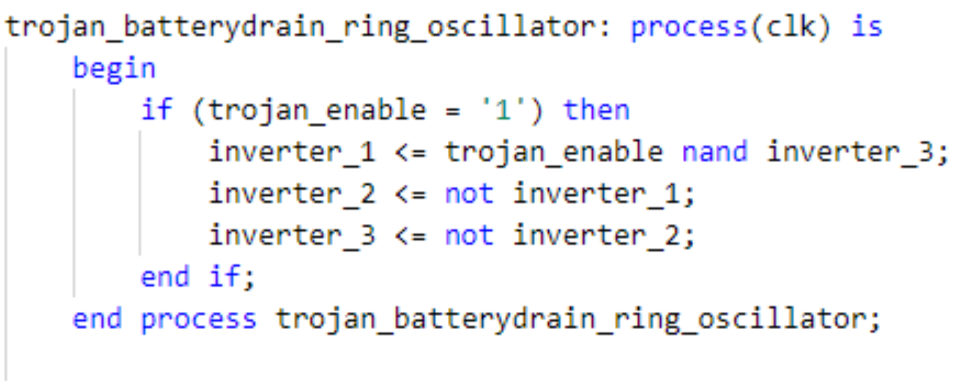


```
'secret_key:trojan_design_test:instance_0'
'data_address:trojan_design_test:instance_0'
```

Figure 4: flagged signals from the Counter Trojan

4.2 Power Drain Trojans

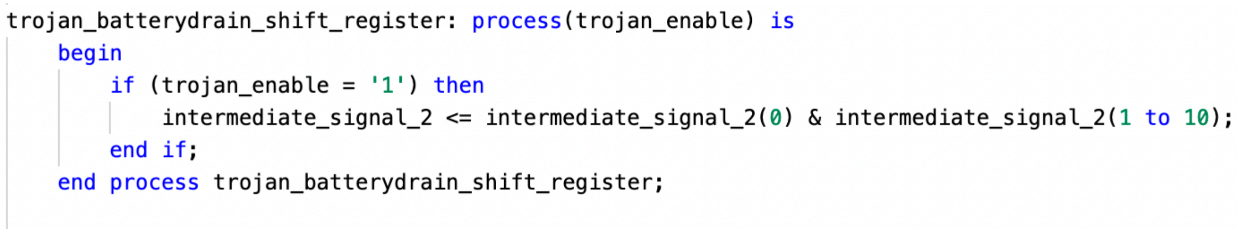
Power Drain Trojans are commonly found in sequential circuits. Two simple sequential circuits were created in the testing VHDL design to assess the Trojan detection for power drain Trojans. The first circuit is a ring oscillator, and it is shown below.



```
trojan_batterydrain_ring_oscillator: process(clk) is
    begin
        if (trojan_enable = '1') then
            inverter_1 <= trojan_enable nand inverter_3;
            inverter_2 <= not inverter_1;
            inverter_3 <= not inverter_2;
        end if;
    end process trojan_batterydrain_ring_oscillator;
```

Figure 5: Trojan-infested ring oscillator

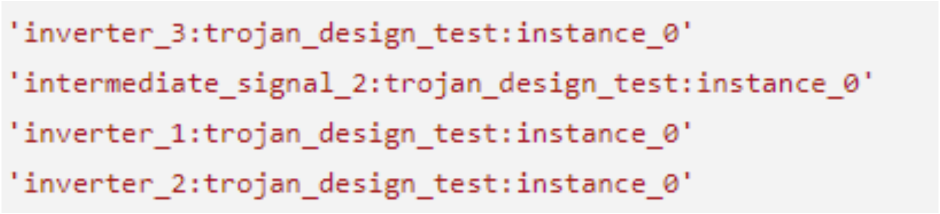
This ring oscillator consists of three inverters. These inverters are only used within this process block. This circuit is enabled by a signal called *trojan_signal*. During the asset assignment, this signal was assigned to ENABLE asset. To detect this Trojan, the tool should pass the three conditions for detecting Power Drain Trojans mentioned in the previous section. The first condition is the existence of a signal with ENABLE asset, which in this case, is the *Trojan_enable*. The second condition relies on the presence of a cycle between a set of signals. In this specific ring oscillator, *inverter_2* depends on *inverter_1*, *inverter_3* depends on *inverter_1,* then *inverter_3* loops back to *inverter_1*. This means that it satisfies the second condition. Lastly, the third condition states that if the sequential circuit does not have any output. Since the asset pattern in this circuit can show that the four inverters are not connected with any output signals, it satisfies the third condition.



```
trojan_batterydrain_shift_register: process(trojan_enable) is
    begin
        if (trojan_enable = '1') then
            intermediate_signal_2 <= intermediate_signal_2(0) & intermediate_signal_2(1 to 10);
        end if;
    end process trojan_batterydrain_shift_register;
```

Figure 6: Trojan-infested shift register

Another sequential circuit created to test this power drain Trojan is a simple shift register. The general purpose of a shift register is to allow the bits of a signal to shift. In Figure 6 above, intermediate_signal_2 is an internal signal and acts like a shift register. This circuit also contains an enable signal, thus passing the first condition. The line *"intermediate_signal_2 <= intermediate_signal_2(0) & intermediate_signal_2(10 downto 1);"* is recognized by the tool as a cycle, therefore passing the second condition. *Intermediate_signal_2* is not connected with any primary output; therefore, it finally satisfies the last condition. Figure 7 lists the flagged from both trojan-infested ring oscillator and shift register

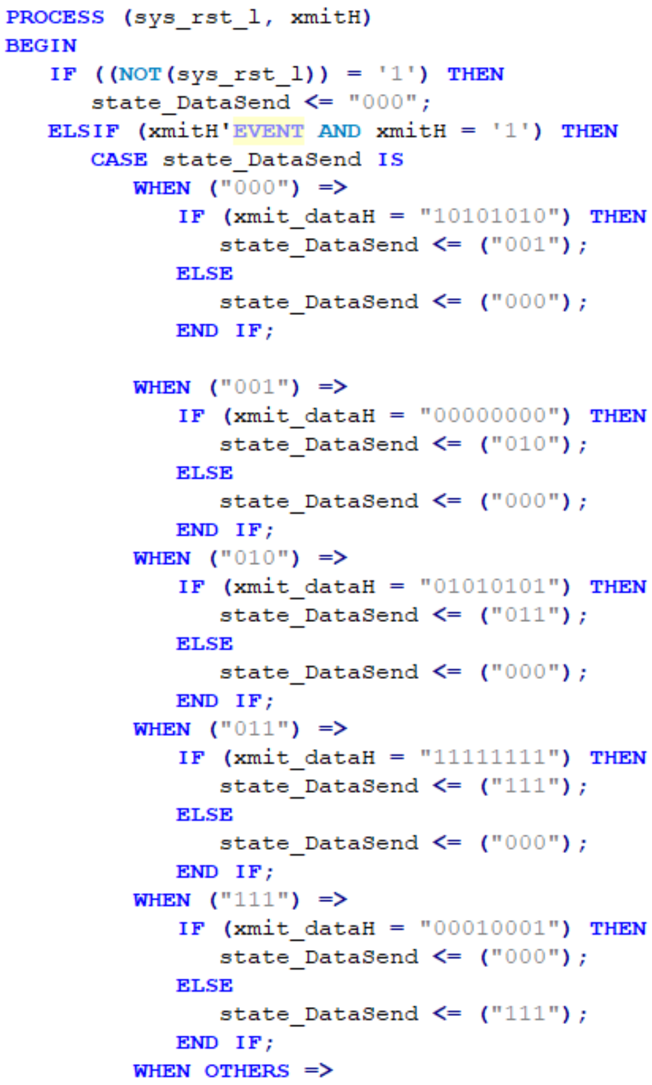


```
'inverter_3:trojan_design_test:instance_0'
'intermediate_signal_2:trojan_design_test:instance_0'
'inverter_1:trojan_design_test:instance_0'
'inverter_2:trojan_design_test:instance_0'
```

Figure 7: flagged signals with potential Power Drain Trojan

4.3 FSM-Triggered Trojans

To evaluate the effectiveness of the Trojan detection for FSM Trojans, a few designs were collected from Trust-hub [15] to test the implementation. One of them is RS232-T600, which is a Trojan-infested communication design that contains a top-level file and two sub-level designs for its receiver and transmitter components. The current version of the tool only supports VHDL files. However, the RS232-T600 design was written in Verilog. Therefore, the design was first converted into VHDL before being parsed by the tool, as shown in Figure 8. The Trojan is hidden within an FSM under the transmitter component.

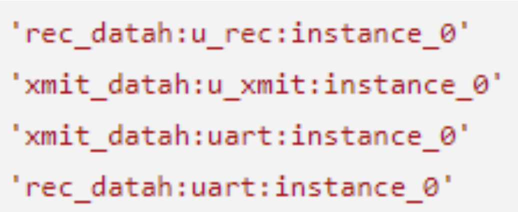


```
PROCESS (sys_rst_l, xmitH)
BEGIN
   IF ((NOT(sys_rst_l)) = '1') THEN
      state_DataSend <= "000";
   ELSIF (xmitH'EVENT AND xmitH = '1') THEN
      CASE state_DataSend IS
         WHEN ("000") =>
            IF (xmit_dataH = "10101010") THEN
               state_DataSend <= ("001");
            ELSE
               state_DataSend <= ("000");
            END IF;

         WHEN ("001") =>
            IF (xmit_dataH = "00000000") THEN
               state_DataSend <= ("010");
            ELSE
               state_DataSend <= ("000");
            END IF;
         WHEN ("010") =>
            IF (xmit_dataH = "01010101") THEN
               state_DataSend <= ("011");
            ELSE
               state_DataSend <= ("000");
            END IF;
         WHEN ("011") =>
            IF (xmit_dataH = "11111111") THEN
               state_DataSend <= ("111");
            ELSE
               state_DataSend <= ("000");
            END IF;
         WHEN ("111") =>
            IF (xmit_dataH = "00010001") THEN
               state_DataSend <= ("000");
            ELSE
               state_DataSend <= ("111");
            END IF;
         WHEN OTHERS =>
```

Figure 8: Trojan-infested FSM from RS232-T600

In this specific Trojan, the payload is activated when the FSM observes specific sequence inputs. Once the FSM transitions to the last state, it will trigger the Trojan, which will ultimately cause a *leak information* attack. If the Trojan detection observes a signal with the CONTROLLABLE asset, it will then analyze its asset trace for all the signals that drive this signal. Since the asset pattern for this design shows that the *state_DataSend* signal is set with a CONTROLLABLE asset, it will then check if it is being driven by any signals that contain any assets from the DATA asset category. Finally, the tool will then report all the signals associated with this potential Trojan. There are two *rec_datah* and *xmit_data_h* listed in Figure 9 since both were used in multiple design files.



```
'rec_datah:u_rec:instance_0'
'xmit_datah:u_xmit:instance_0'
'xmit_datah:uart:instance_0'
'rec_datah:uart:instance_0'
```

Figure 9: Flagged signals from RS232-T600

5. CONCLUSION AND FUTURE WORK

Hardware Trojan has become an important security threat to IC vendors incorporating third-party IPs. Therefore, detecting these Trojan earlier in the IC design flow is critical. There are two existing methods of Trojan detection for the Structural Checking tool. The first method is functionality, which compares an unknown soft IP design to a list of trusted designs in the Golden Reference Library to see whether it matches closely with a Trojan-free or a Trojan-infested design. The second method of Trojan detection uses the concept of asset patterns, which is done by analyzing an unknown design to determine if it contains a unique asset pattern commonly found in Trojan-infested designs. Much of the research done to improve the Structural Checking tool mainly focused on the functionality matching method. This thesis research expanded on the other method of Trojan detection. A framework was developed to help future researchers identify the unique asset patterns hidden in Trojan-infested designs so that they can implement the proper detection for those Trojans. In addition, three Trojan detection examples were introduced using this framework.

Many future works can be done to improve the framework introduced in this thesis. The idea of Trojan detection using asset patterns relies on signals having been assigned with proper assets. However, the current asset assignment implemented in the tool is mainly tailored for external signals. There are limited ways an internal signal can be assigned with assets. For example, some implementation of the Counter Trojan consists of mostly internal signals. Because of this, those internal signals cannot be assigned with a COUNT asset, which is a crucial part of detecting a Trojan using asset patterns. Another limitation is that many Trojan-infested designs were written in Verilog, and by the time this thesis was written, the tool's Verilog support has not been completed yet. Because of this, there are limited testing designs used in this research; thus, accuracy is not the main focus of this research. However, the three Trojan detections introduced in this thesis can be improved once more testing designs becomes available to be parsed by the tool.

6. REFERENCES

[1] 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.

[2] Jie Li and J. Lach, "At-speed delay characterization for IC authentication and Trojan Horse detection," 2008 IEEE International Workshop on Hardware-Oriented Security and Trust, 2008, pp. 8-14, doi: 10.1109/HST.2008.4559038.

[3] S. Jha and S. K. Jha, "Randomization Based Probabilistic Approach to Detect Trojan Circuits," 2008 11th IEEE High Assurance Systems Engineering Symposium, 2008, pp. 117-124, doi: 10.1109/HASE.2008.37.

[4] 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.

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

[6] Le, T. P. (2018). Securing Soft IPs against Hardware Trojan Insertion. *Graduate Theses and Dissertations* Retrieved from https://scholarworks.uark.edu/etd/2694

[7] 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.

[8] 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), 2019, pp. 1-6, doi: 10.1109/HPEC.2019.8916225.

[9] 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), 2021, pp. 861-864, doi: 10.1109/MWSCAS47672.2021.9531813. Signal Assets - a Useful Concept for Abstracting Circuit Functionality

[10] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," in IEEE Design & Test of Computers, vol. 27, no. 1, pp. 10-25, Jan.-Feb. 2010, doi: 10.1109/MDT.2010.7.

[11] 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.

[12] 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.

[13] Wolff, Francis & Papachristou, Ch & Bhunia, Swarup & Chakraborty, Rajat. (2008). Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme. 1362-1365. 10.1145/1403375.1403703.

[14] 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.

[15] trust-hub, [online] Available: https://www.trust-hub.org/.