Automated Trojan Detection and Analysis in Field Programmable Gate Arrays

by

Nicholas Houghton
B.Eng., University of Victoria, 2015

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

Automated Trojan Detection and Analysis in Field Programmable Gate Arrays

by

Nicholas Houghton
B.Eng., University of Victoria, 2015

Supervisory Committee

ABSTRACT

Electronics have become such a staple in modern life that we are just as affected by their vulnerabilities as they are. Ensuring that the processors that control them are secure is paramount to our intellectual safety, our financial safety, our privacy, and even our personal safety. The market for integrated circuits is steadily being consumed by a reconfigurable type of processor known as a field-programmable gate-array (FPGA). The very features that make this type of device so successful also make them susceptible to attack. FPGAs are reconfigured by software; this makes it easy for attackers to make modification. Such modifications are known as hardware trojans. There have been many techniques and strategies to ensure that these devices are free from trojans but few have taken advantage of the central feature of these devices. The configuration Bitstream is the binary file which programs these devices. By extracting and analyzing it, a much more accurate and efficient means of detecting trojans can be achieved. This discussion presents a new methodology for exploiting the power of the configuration Bitstream to detect and described hardware trojans. A software application is developed that automates this methodology.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My parents and family for the trust and support while I learned what was truly important in life.

Natalie, sometimes, the only thing keeping me going was knowing you would never stop applauding me.

My supervisor, Dr. Fayez Gebali, for the trust that I could be more than what I was. The opportunity to excel may never have come without you.

Co-Supervisor and Friend, Dr. Samer Moein, for the endless guidance and pushing me to always do better

My Committee, Dr. Brian Wyvill, for your precious time and valuable insight.

You want to know the difference between a master and a beginner? The master has failed more times than the beginner has even tried

Yoda

DEDICATION

To my parents, Peter and Valerie. Simply, for everything.

Chapter 1

Introduction

The term *Trojan Horse* or *Trojan* has become a modern metaphor for a deception where by an unsuspecting victim welcomes a foe into an otherwise safe environment [1]. Though modern civilization rarely has need for large walls we are similarly surrounded. Not by stone and mortar but by the technology we so heavily rely on. These days it is more common to come across a piece of equipment with some form of computer in it than without. They provide us entertainment, education, security, monitor our health, grow our food and more. Our reliance make us susceptible to their compromise. Since the dawn of the computer we have dealt with software threats. We are almost as good at protecting ourselves against them as attackers are at making them. In recent years a new incarnation of electronic danger has emerged; in hardware. In this new arena of attack and defend those who seek to defend are far behind.

1.1 Motivation

In the summer of 2007 an Israeli military action referred to as Operation Orchard commenced. A group of F-15I Ra'am fighter jets from the Israeli Air Force 69th Squadron took off to attack a suspected nuclear reactor in neighboring Syria [2]. In the flight path was a Syrian radar station which boasted 'state-of-the-art' aircraft detection and neutralization technology. The Israeli war planes were able to approach and destroy the installation undetected. Though never proven it is commonly accepted that the detection mechanism was deactivated by a back-door circuit inserted into the radar system. In 2011 over 1300 cases of modified ICs were reported to the

Electronic Resellers Association International (ERAI); the occurrence of such cases has since been going up [3]. Integrated Circuits (IC) are a large part of modern life yet it is easy to forget that they drive virtually every piece of technology used today. Ensuring that their ICs run our devices as expected is vital in the digital era. Since their discovery there has been concerted effort to detect these modifications but the innate complexity of ICs makes it difficult [4]. These modifications are commonly known as hardware trojans.

One of the most commonly attempted trojan detection techniques is to apply a series of inputs and look to see if the outputs are as expected. This is known as Functional testing or Test Vectoring [5, 6, 7]. This method has been shown to provide some success but is commonly beaten by attackers. Another common strategy is to search for side effects of modified circuits; these methods are referred to as 'side-channel' analysis methods. Such side effects include, changes to power consumption, temperature differences, radiation differences and more [8, 9, 10, 11]. Again, these methods provide varying success but face a lot of difficulty.

IC designs for Field Programmable Gate-Arrays (FPGA) are made using a software language. The design is then converted to a binary file called a configuration Bitstream which is then downloaded onto the device; this process is known as synthesizing the design. There have been many attempts to develop mechanisms and techniques to determine whether a malicious user has tampered with the design via test vectoring or side-channel analysis. As of yet there has been little effort to directly analyze the configuration Bitstream.

1.2 Contributions

The goal of this work is to prove that analysis of the configuration Bitstream is the most powerful means of detecting trojans in FPGAs. To do so, an application was developed which parses and analyzes the Bitstream, detects the presence of trojans and provides an astute description of the modification.

The contributions of this work can be said to be:

1. Proof that though FPGA manufacturers hide the intimate details of the configuration Bitstream it is the most viable means of trojan detection and analysis.
2. Proof of concept for a new method of detecting and analyzing trojans.
3. Develop a software application which automates the described methodology.

1.3 Organization

This enclosing section presents a map of this work and a short description of each chapter. Chapter 2 provides some background information on trojans themselves and introduces the descriptive taxonomy used. Chapter 3 introduces the new detection and analysis methodology. It gives a brief overview of FPGA architecture and configuration, describes the analysis process of the Bitstream, introduces *Component Mapping*, and discusses how the taxonomic attributes are extracted. Chapter 4 discusses the software implementation which has been named FPGA Trojan Detector. It discusses the technologies used and why they were chosen, presents the User-Interface and discusses its operating procedure. Chapter 5 presents the results of experimentation done using three FPGA-trojan benchmarks These benchmarks were specifically chosen to demonstrate that the application works as expected and is easy to use. Finally, chapter 6 concludes this work and restates its contributions.

Chapter 2

Hardware Trojans

2.1 Background

Integrated Circuits (IC) are continuously decreasing in size whilst increasing in complexity. These trends require ever more people and sophisticated means of manufacture which in turn creates security vulnerabilities. Products developed by semiconductor companies tend to be compromised in one of two ways. First, due to its complexity it is rare for an IC to be entirely manufactured within a single company. Frequently, steps in the production-chain are outsourced. It is within these 'third-party' contributors that products can be maliciously modified. Secondly, for various reasons, employees of trusted contributors have been known to make modifications [12]. ICs are an integral part of every facet of the modern world. Proper application of a trojan can provide information, control of mechanical systems, surveillance and more to an unauthorized party.

2.2 Topology

The discussion, detection and evaluation of hardware trojans requires a comprehensive means of description. Several hardware trojan taxonomies have been proposed [13, 14, 15, 16]. In [13], trojans were organized based solely on their activation mechanisms. A taxonomy based on the location, activation and action of a trojan was presented in [14], [15]. However, these approaches do not consider the manufacturing process. Another taxonomy was proposed in [16] which employs five categories: insertion, abstraction, activation, effect, and location. While this is more extensive than

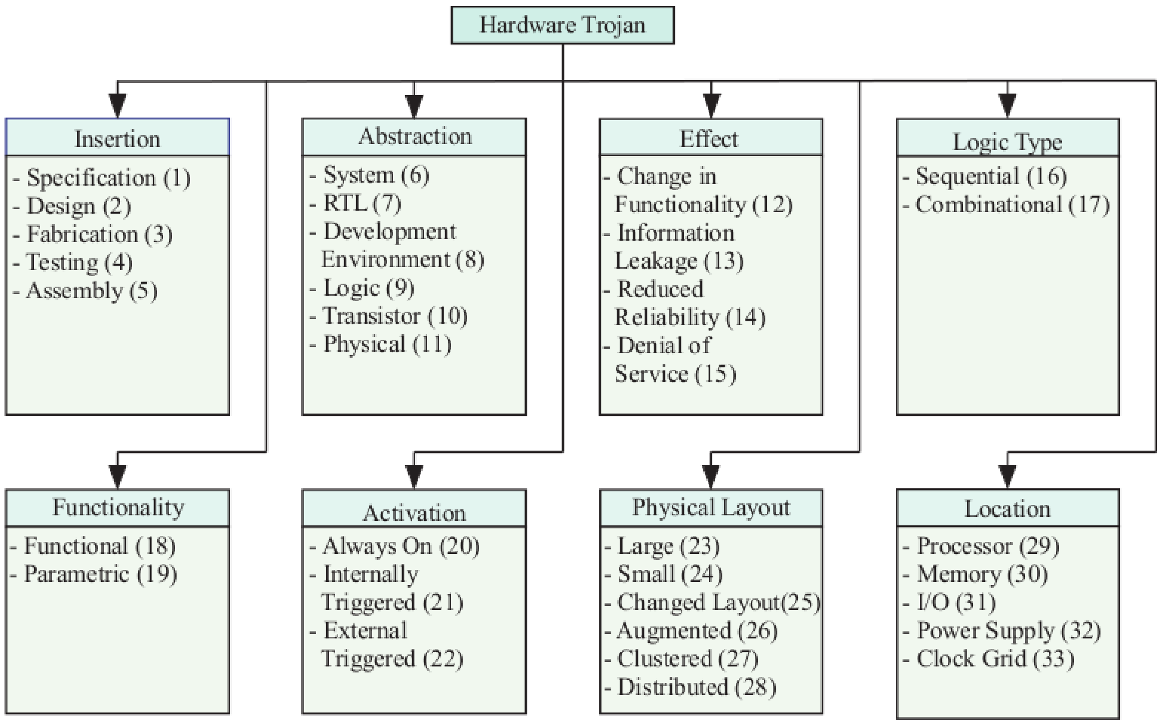


Figure 2.1: The thirty-three attributes of the hardware trojan taxonomy in [17].

previous approaches, it fails to account for the physical characteristics of a trojan. An additional taxonomy was proposed in [17] which considers all attributes a hardware trojan may posses. This taxonomy is the most comprehensive and was selected as the means of description for this work. It is comprised of thirty-three attributes organized into eight categories as shown in Fig. 2.1. These categories can be arranged into the following four levels as indicated in Fig. 2.2.

1. The **insertion** (chip life-cycle) level/category comprises the attributes pertaining to the IC production stages.
2. The **abstraction** level/category corresponds to where in the IC abstraction the trojan is introduced.
3. The **properties** level comprises the behavior and physical characteristics of the trojan.
4. The **location** level/category corresponds to the location of the trojan in the IC.

The properties level consists of the following categories.

- The **effect** describes the disruption or effect a trojan has on the system.
- The **logic type** is the circuit logic that triggers the trojan, either combinational

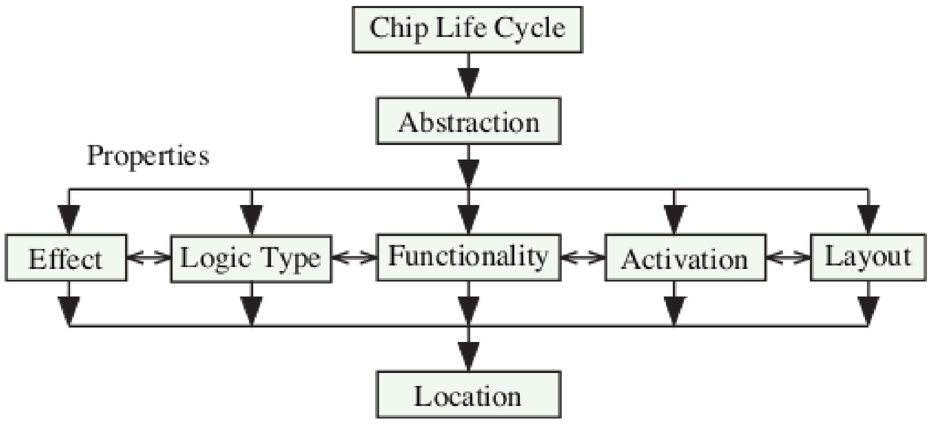


Figure 2.2: The hardware trojan levels [17].

or sequential.

- The **functionality** differentiates between trojans which are functional or parametric.
- The **activation** differentiates between trojans which are always on or triggered.
- The **layout** is based on the physical characteristics of the trojan.

The relationships between the trojan attributes shown in Fig. 2.1 can be described using a matrix $\mathbf{R}$ [17]. Entry $r(i, j)$ in $\mathbf{R}$ indicates whether or not attribute i can lead to attribute j. For example, $r(2, 3) = 1$ indicates that design (attribute 2) can lead to fabrication (attribute 3). This implies that if an IC can be compromised during the design phase (attribute 2), it may influence the fabrication phase (attribute 3).

The matrix $\mathbf{R}$ is divided into sub matrices as follows

$$\mathbf{R} = \begin{bmatrix} \mathbf{R_1} & \mathbf{R_{12}} & 0 & 0 \\ 0 & \mathbf{R_2} & \mathbf{R_{23}} & 0 \\ 0 & 0 & \mathbf{R_3} & \mathbf{R_{34}} \\ 0 & 0 & 0 & \mathbf{R_4} \end{bmatrix}$$

$$\mathbf{R} = \left[\begin{array}{c||ccccc||cccccc||ccccccccccccccccc||ccccc}
A & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 \\ \hline\hline
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & & & & & & & & & & & & & & & & & & & & & & \\
2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & & & & & & & & & & & & & & & & & & & & & & \\
3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & & & & & & & & & & & \\
4 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & & & & & & & & & & & & & & & & & & & & & & \\
5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & & & & & & & & & & & & & & & & & & & & & & \\ \hline\hline
6 & & & & & & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & \\
7 & & & & & & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & & & & & \\
8 & & & & & & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & \\
9 & & & & & & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & \\
10 & & & & & & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & & & & & \\
11 & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & \\ \hline\hline
12 & & & & & & & & & & & & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
13 & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
14 & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
15 & & & & & & & & & & & & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
16 & & & & & & & & & & & & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
17 & & & & & & & & & & & & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
18 & & & & & & & & & & & & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
19 & & & & & & & & & & & & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
20 & & & & & & & & & & & & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
21 & & & & & & & & & & & & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
22 & & & & & & & & & & & & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
23 & & & & & & & & & & & & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
24 & & & & & & & & & & & & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
25 & & & & & & & & & & & & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
26 & & & & & & & & & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
27 & & & & & & & & & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
28 & & & & & & & & & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ \hline\hline
29 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & \\
30 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & \\
31 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & \\
32 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & \\
33 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & &
\end{array}\right]$$

where $\mathbf{R_1}$, $\mathbf{R_2}$, $\mathbf{R_3}$ and $\mathbf{R_4}$ indicate the attribute relationships within a category. For example, $\mathbf{R_1}$ is given by

$$\mathbf{R_1} = \left[\begin{array}{c|ccccc} A & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 1 \\ 5 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Submatrix $\mathbf{R_{12}}$ relates the attributes of the insertion category to the attributes of the abstraction category. An example of this submatrix is

$$\mathbf{R_{12}} = \left[\begin{array}{c|cccccc} A & 6 & 7 & 8 & 9 & 10 & 11 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 & 1 & 0 & 0 \\ 5 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

2.3 Recent Work

Hardware trojans are a new and exciting field of study. As FPGAs take a larger portion of the Integrated Circuit (IC) market the need for security becomes greater. With this, detection and analysis of trojans in FPGAs has become a growing topic and has seen some development in recent years. The majority of work focused on FPGA trojans has employed either a means of reverse engineering, functional testing or 'side-channel' analysis. The configuration Bitstream contains all of the information one would want to know about what is happening on the device. There has been little effort to directly analyze the Bitstream because manufacturers are reluctant to provide intimate details of its format.

2.3.1 Cyclic Redundancy Check (CRC) Detection

Because FPGA design is done using a software language it is possible for designers to share component designs easily. It is common for designers to employ 'third-party'

Intelectual Property (IP). This sharing of component design provides the opportunity for attackers to add trojans to commercial products by sharing trojan-containing IP. In 2013 researchers at Cairo University proposed a method of insulating externally sourced IP with Cyclic Redundancy Check (CRC) defense modules [18]. According to the authors their method is capable of detecting leaked information with a 99.95% accuracy. This method employs a methodology referred to as 'built-in-self-test" (BIST) where additional hardware is added to the design. The additional hardware performs run-time checking of circuit output. In general, the BIST method is only capable of detecting modifications the designers have foreseen. In this case, this method is only capable of detecting trojans that possess the attribute Information Leakage (13) from the taxonomy presented ins section 2.2. In addition the authors report considerable detriment to power consumption and performance.

2.3.2 Ring Oscillator (RO) Detection

Researchers at the Technological Educational Institute of Western Greece and Industrial Systems Institute/RC Athena jointly proposed a method of using Ring Oscillators (RO) as a mechanism for detecting hardware trojans [19]. A RO is a circuit composed of inverters formed into a loop. Electric current looping through the RO does so at an inherent frequency. By configuring the circuit paths of the user's design into a RO it is possible to create a 'signature'. This signature is an expected frequency emitted from the desired design. The authors claim that modifications to the design will alter the frequency emitted by its circular configuration. The experimental results showed that modifications did in-fact alter the frequency enough to reliably detect modifications. This method can reliably detect hardware trojans but is incapable of providing any details regarding its effect. Further, the stipulation that the desired design must be such that it forms an oscillating ring is impractical. It is impossible to guarantee that all real-world designs can form an RO whilst maintaining desired functionality and performance.

2.3.3 The Multi-Faceted Approach

Researchers from Iowa State University proposed a multi-faceted approach to trojan detection in FPGAs [20]. Their method composed of three approaches:

- Functional Testing: A means of feeding test vectors and comparing the output to expected results.

- Power Analysis: Using an oscilloscope, the difference in power consumption between the desired design and the modified devices performing the same operations were recorded. Differences were used to discern the presence of a trojan.
- Bitfile Analysis: The authors attempted to employ a binary file analysis library named *deBit* [21] to reverse engineer a netlist from the Bitstream.

The functional testing method attempted provided reasonable results. Test vectors used showed unexpected behavior; this provided only the information that a trojan was present. The power analysis method again provided results of moderate quality. With careful placement of the oscilloscope probes the authors were able to infer the physical location on the device where modifications occurred. This provided no information however as to the relation between the modifications and the design. Finally, the authors were able to only partially able to convert the Bitstream to its netlist description. Only descriptions of the primary logic circuit elements were achieved. This could be used to discern some information regarding modifications discovered but is far from creating a complete description of a trojan.

Chapter 3

Automated Trojan Detection

3.1 Methodology

An Integrated Circuit (IC) belongs to one of two categories. An Application Specific Integrated Circuit (ASIC) or a Field Programmable Gate-Array (FPGA). An ASIC is manufactured once and is immutable; its hardware is permanently printed into its silicon. FPGAs are reconfigurable because they are comprised of an array of Programmable Logic Devices (PLD). A PLD is a component whose functionality is dependent on a set of configuration options; in other words, a user can define how it behaves. Each PLD receives configuration instructions from the user that defines its functionality; these instructions are in the form of a binary message.

The methodology proposed in this chapter focuses on FPGA devices manufactured by *Xilinx* . In *Xilinx* terminology a PLD is referred to as a tile. A single FPGA can contain hundreds, or even thousands of tiles; a device is usually made up of over a hundred different types. Different types are used for different functions, such as Input-Output (IO), Logic, Memory...etc. The set of messages sent to all of the tiles in the device from the user is referred to as the configuration Bitstream.

FPGA users create designs using a programming language referred to as a Hardware Description Language (HDL). The design is then compiled and synthesized into a configuration Bitstream which is then downloaded or ”configured” onto the device. Creating HDL designs for FPGAs is considerably cheaper and easier then designing ASIC chips. Additionally, if the user wishes to make modifications the design on the device can be updated when out in the field; this is another large advantage over ASICs. Because of this, FPGAs are becoming the IC of choice in larger scale productions. These same features, however, increase their vulnerability to attack.

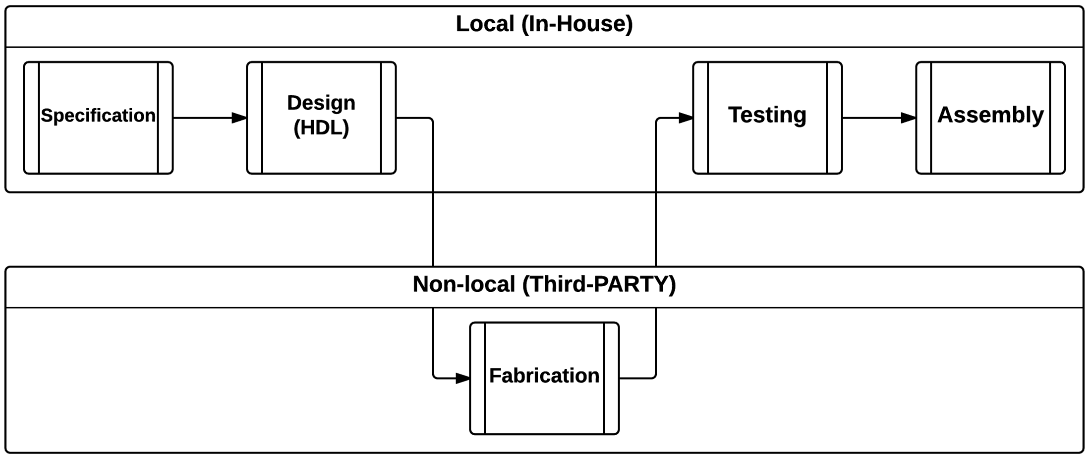


Figure 3.1: FPGA Life-Cycle

Figure 3.1 provides a visual representation of the use-case assumed for the purposes of this work. With the exception of the fabrication process, all stages of production of an FPGA implementation are assumed to have been done "in-house". Any trojan discovered is inserted in the fabrication phase; all other stages are trusted. The method of automated trojan detection described in this work would take place in the 'testing' phase of the life-cycle.

Figure 3.2 shows an overview of the trojan detection methodology. As mentioned, FPGA designs are written in a Hardware Description Language (HDL). *Xilinx* provides a series of User-Interface (UI) and command line tools to process the HDL known as the 'tool-chain'. The tool chain generates a series of files that are used for a variety of purposes as shown in the 'Resultant Files' box in Figure 3.2. The NGC file is a non-human readable semantic description of the design known as a netlist. This file can be converted into a human-readable version known as *Xilinx* Design Language (XDL) which will be described in section 4.2.1. The Bit file is the binary representation of the design to be implemented. It is referred to as the Bitstream or 'configuration' Bitstream and is the final form that is loaded into the FPGA. This Bit file is the primary file sent to the fabrication house where it will be implemented onto the batch of devices ordered. The resultant files, produced 'in-house' are to be kept in secure storage while a copy is sent to be fabricated; these stored copies are referred to as Golden and assumed to be trojan-free. Though it is known that the

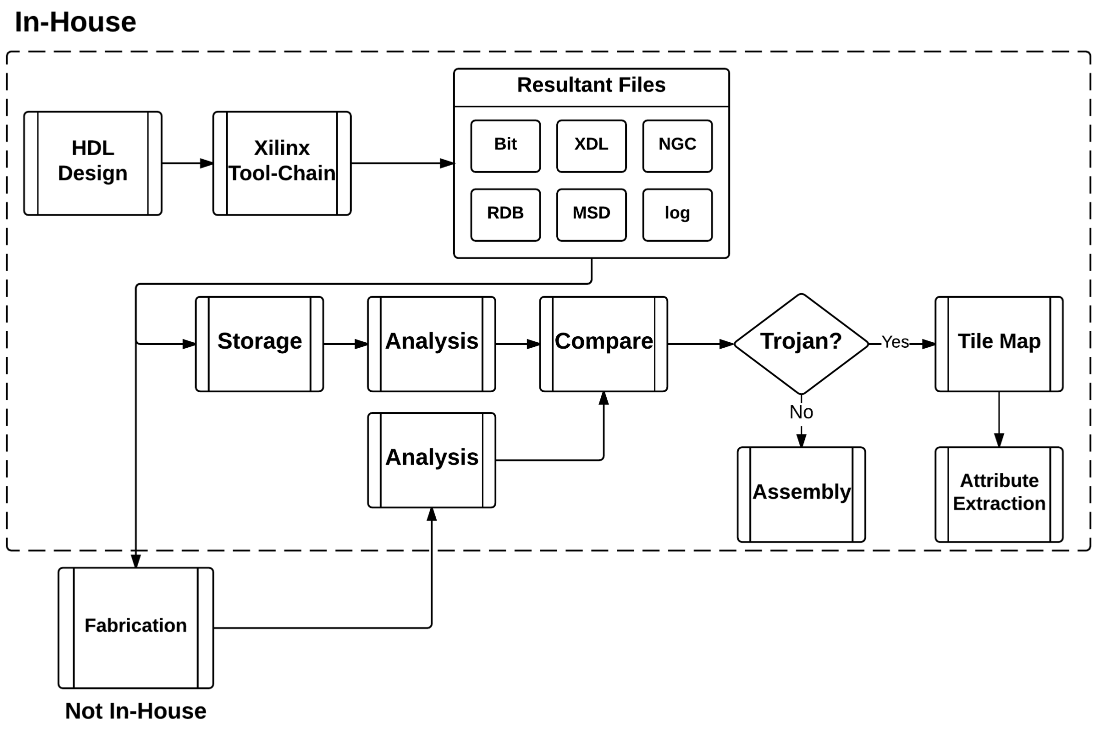


Figure 3.2: Methodology Overview

fabrication houses will often attempt to make optimizations on designs, this methodology requires that no such efforts be made. When the completed batch of fabricated chips are returned the Bitstream is extracted from a sample via the method described in section 3.3. That which is extracted is referred to as the Target Bitstream. The Golden and Target Bitstreams are analyzed in conjunction to detect differences. This technique is described in section 3.4. Any discovered differences are then attributed to the corresponding component in the architecture, described in section 3.5. Finally, the descriptive attributes, originally presented in section 2.2, are returned to the user. This is described in section 3.6.

3.2 FPGA Architecture and Configuration

A *Xilinx* Field Programmable Gate-Array (FPGA) is comprised of a matrix of blocks referred to as the 'gate-array' and is shown in Figure 3.3 [22]. A device can contain anywhere from a couple hundred to a few thousand blocks and are arranged into columns by type. A block is not a physical device but a conceptual grouping of

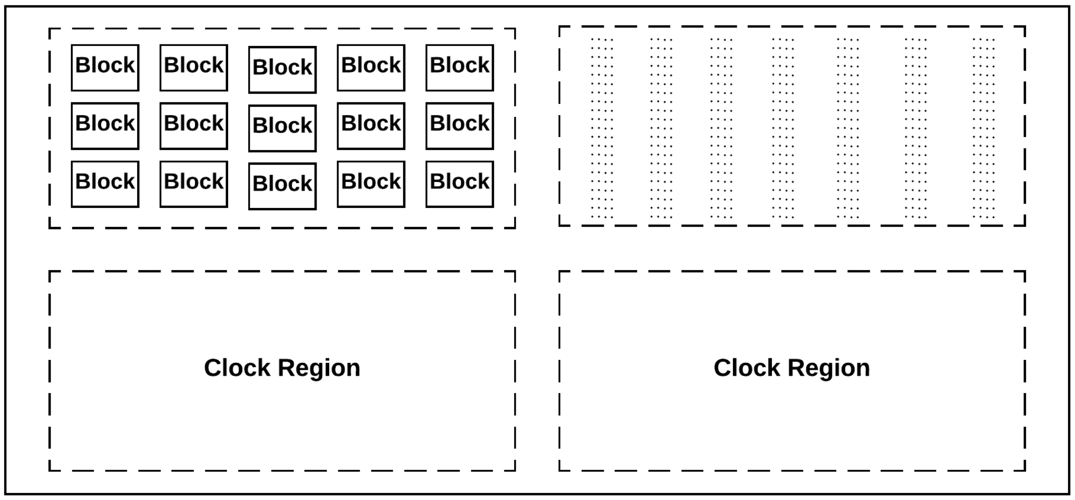


Figure 3.3: Gate-Array of Block Columns Separated into Clock Regions

tiles. A block will consist of one or multiple tiles depending on its type. A tile is a component specific to a particular function such as Input-Output (IO), design logic, memory...etc but their detailed functionality can be configured by the user. Though an FPGA may have over one-hundred different types of tiles each column is

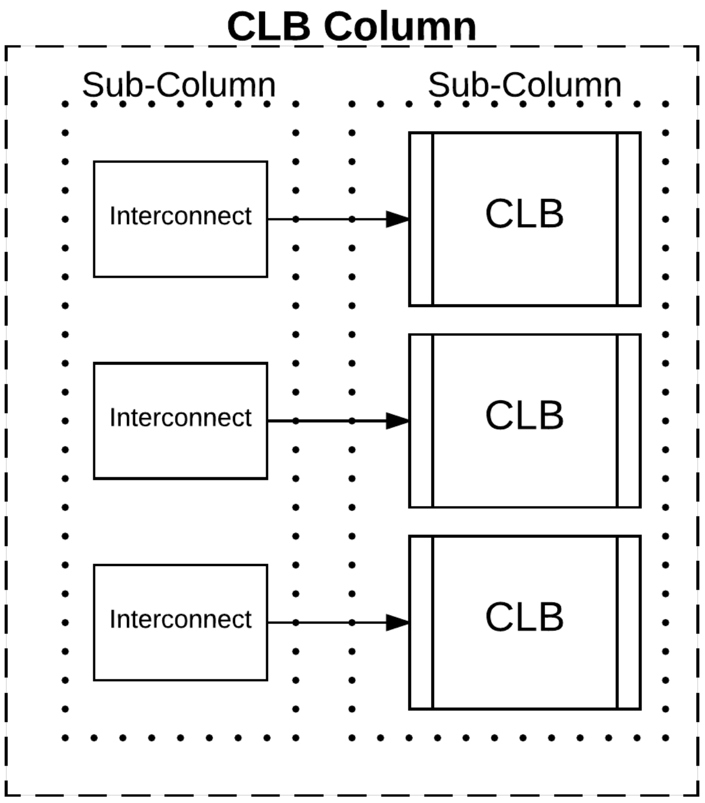


Figure 3.4: Column Composition

comprised entirely by a single block type. Columns are separated into regions shown by the dashed lines in Figure 3.3. These regions each use a separate clock mechanism and are referred to as 'Clock Regions'.

Figure 3.4 depicts a Configurable Logic Block (CLB) column. As mentioned, blocks may contain multiple tiles. Each row in Figure 3.4 (i.e. the interconnect/CLB tile-pair) is a block. In this column the CLB tile is the primary feature of each block, hence, it is referred to as a CLB block. Consequentially, since this column is made up of CLB blocks it is referred to as a CLB column. A column is comprised of blocks stacked vertically. In the case where blocks contains multiple tiles, columns can be thought to contain sub-columns. The stack of interconnects can be considered the 'interconnect' sub-column while the stack of CLB tiles can be thought of as the CLB sub-column. Combined, the two sub-columns make up this CLB column. Though each tile has a designated purposes (ex. Input-Output (IO), Configurable Logic (CL), memory...etc) their functionality can be configured by the user; this is how designs are implemented on a device. The configuration of each tile is dictated by the Bitstream.

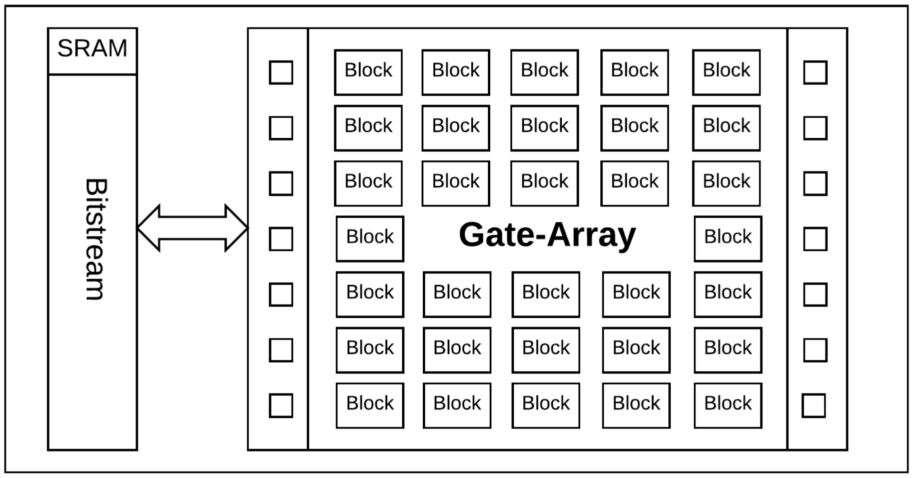


Figure 3.5: FPGA Device Layout

To improve performance the blocks of the gate-array are dynamic memory components. A dynamic device is unable to retain the contents of its memory when it looses power. To prevent having to plug in a device and download the configuration every time it is powered on, an external static memory device (i.e. retains its contents with loss of power) holds the Bitstream. When an FPGA is powered on the Bitstream is loaded from the external memory (often Static Random Access Memory (SRAM)) into the gate-array, as can be seen in Figure 3.5.

3.3 Bitstream Extraction

In order to detect any trojans in the Target device the configuration Bitstream will need to be recovered. As mentioned in section 3.2 the Bitstream is stored in a memory unit external to the gate-array [23]. All *Xilinx* devices provide a feature known as Readback. There are two styles of Readback; Readback verify and Readback capture. The Readback capture method provides a large quantity of debug information which is not needed; Readback verify will be used. Readback verify is the process where the device is put into a 'frozen' state during run-time and all of the configuration bits are returned from the gate-array to the SRAM. The results can then be uploaded to a Personal Computer (PC) for analysis. This process overwrites the original frame data in the SRAM with the values which actually configured the device. By using this method rather than simply reading the SRAM it ensures that what is tested in section 3.4 is actually what configured the device. This minimizes risk of tampered external memory units or configuration mechanics.

3.4 The FPGA Bitstream Analysis

The *Xilinx* Bitstream is a binary file composed of a series of 32-bit words organized into 'frames'. A frame is a string of single bits that span from the top to the bottom of a clock region of a device as seen in the top-right quadrant of Figure 3.3. A frame affects every block in a column and multiple horizontally adjacent frames are required to configure an entire column. Each frame is uniquely identified by a 32-bit address and is the smallest addressable element. The composition of the frame address is fairly consistent across the *Xilinx* catalog however there are small differences between device families. The following is the structure of the Virtex-5 family frame address scheme according to [23]. The make-up of a frame address is shown in Table 3.1.

Table 3.1: Frame Address

Unused								BA			T	Row Address					Major Address								Minor Address						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0

The Block Address (BA) identifies the block type.

- BA 0: Logic type.
- BA 1: Block Random Access Memory (BRAM).
- BA 2: BRAM Interconnect.
- BA 3: BRAM non-configuration frame.

The logic block contains the columns which provides the primary configuration for the device (CLBs, IOBs... etc). The BRAM columns initialize the memory for the device while the BRAM Interconnect columns configure how the logic of the design interacts with the BRAM.

Each clock region is given a row value in its address that increments away from the center of the device starting at 0. The frame address includes a Top indicator bit in position 20 that indicates whether the specified row is above or below the center of the

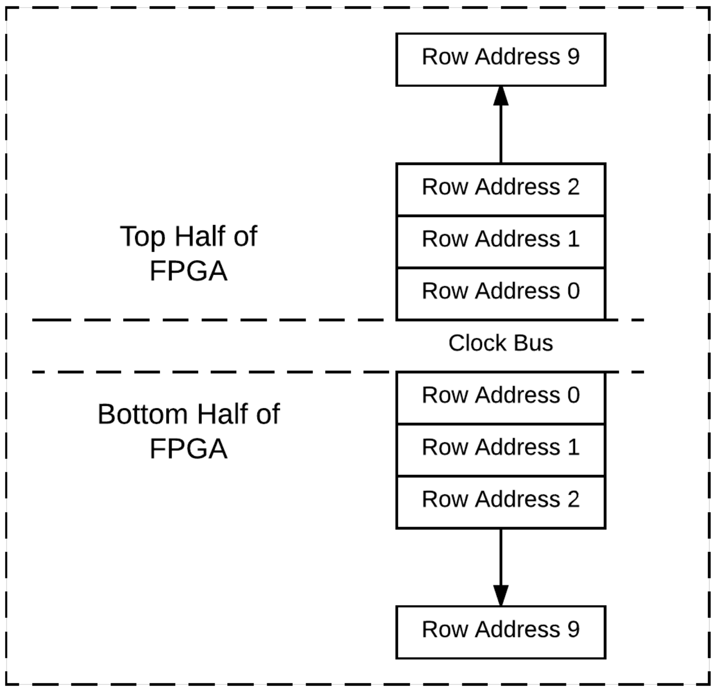


Figure 3.6: Row Order of Virtex-5 Clock Region

device [23]. The major address specifies the column within the row. These addresses are numbered from left to right and begin at 0. The minor address indicates the frame number within a column. Table 3.2 provides the number of frames per column type. As described in section 3.2 a block may contain multiple tiles. In a CLB column a

Table 3.2: Number of Frames (minor addresses) per Column [23]

Block	Number Of Frames
CLB	36
DSP	28
BRAM	30
IOB	54
Clock	4

block consists of an interconnect tile, also known as a Switching Matrix (SM) and a CLB. Frames are numbered from left to right, starting with 0. For each block, except in a clock column, frames numbered 0 to 25 access the interconnect tile for that column. For all blocks, except the CLB and the clock column, frames numbered 26 and 27 access the Interface for that column. All other frames are specific to that block [23]. To further understand how frames configure tiles a mapping must be made between each frame and the corresponding tile. This is described in section 3.5.

3.5 Component Mapping

The FPGA Trojan Detector employs a method referred to as Component Mapping to create a mapping between each word in a configuration frame and the component on the device that it configures. This information is not publicly released by *Xilinx* as a means of providing security through obscurity.

3.5.1 Frame to Column Mapping

The configuration Bitstream is stored in an external memory device as described in section 3.2. When powered-on the Bitstream is transmitted in frame address order to populate the dynamic memory in the tiles of the gate-array. The frame addressing scheme describes where in the gate-array the frame is destined fairly directly. Frames with a BA value of 1 are clearly destined to configure the BRAM and do not need further analysis for the purposes of this method. Frames with a BA value of 0 or 2 must be mapped more finitely. The row address specifies which row of clock-regions the frame is destined. Figure 3.3 shows four clock regions organized into two rows.

As an example the Virtex-5 240T has 12 rows; its row address spans from 0-5 and the Top bit in the address indicates whether it is in the top or bottom half of the device in accordance with Figure 3.6. Once the correct clock region is discerned the major address is used to determine which column the frame configures. The major address begins at 0 on the left and counts up towards the number of columns in the row. Finally, the minor address is used to determine which sub-column has been modified according to Table 3.1.

3.5.2 Word to Block Mapping

In the case of Virtex-5 devices a frame is composed of 41 words that can be thought of as a vertical stack that aligns with a column. As described in section 3.4 a row consists of a stack of basic blocks; there are 20 CLB blocks per column, 40 IOBs, 4 BRAM...etc for Virtex-5 devices. As can be seen in Figure 3.7 the central word in a frame configures the horizontally running clock bus. The remaining words are

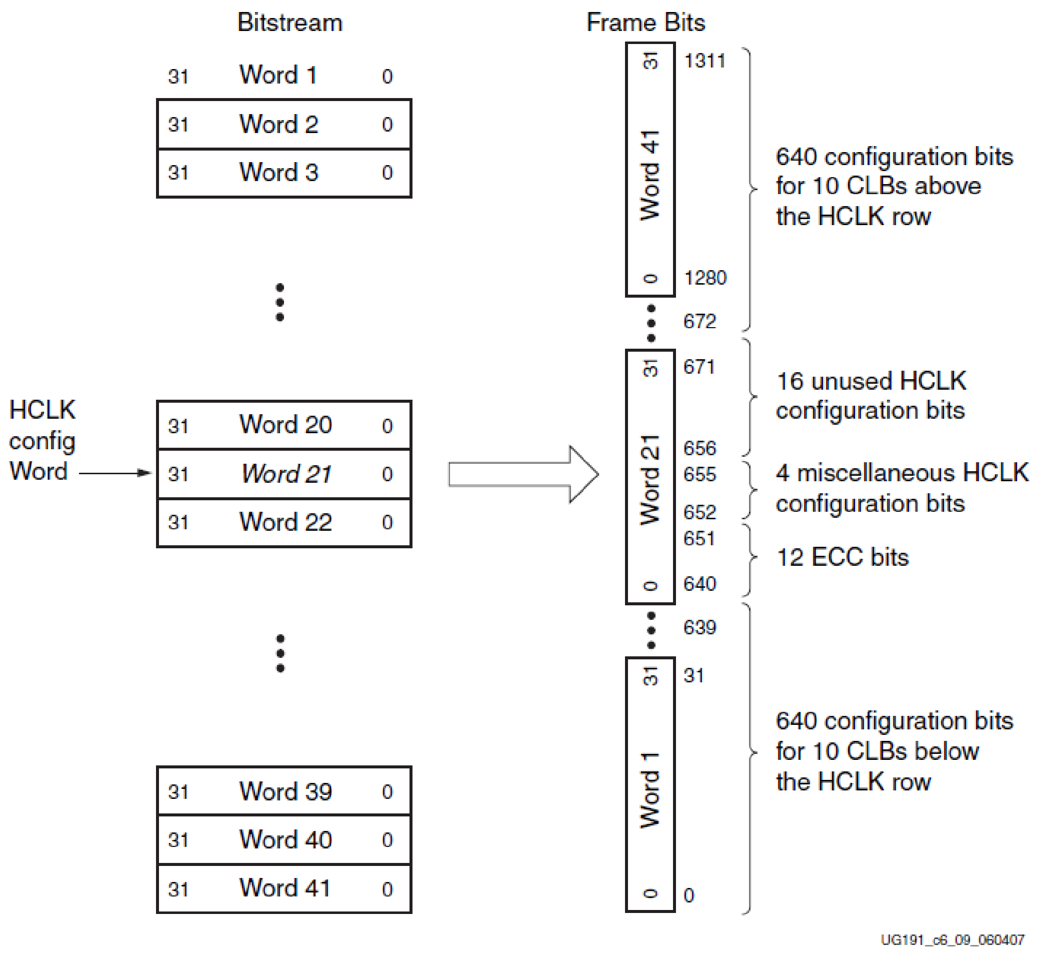


Figure 3.7: Configuration Words in the Bitstream [23]

used to configure the blocks in the column. The purpose of the central word in the column is known to be mapped to the clock bus. For the purposes of the following computations it is considered removed from the frame. From this, equation 3.1 can be deduced which is used to compute the number of 32-bit words that span each block.

$$n = (W - C) + B \tag{3.1}$$

where:

n	= Number of Words per Block
W	= Number of 32-bit words per frame
C	= Number of clock words per frame
B	= Number of blocks per column

As shown in Figure 3.7 words are addressed from the 'top' of a device down.

Equation 3.2 can be used to map a particular word in a frame to a block on the device.

$$i = B - \left\lfloor \frac{w}{n} \right\rfloor \tag{3.2}$$

where:

i = Word Number in frame
B = Number of blocks per column
w = Word number
n = Number of Words per Block

With equations 3.1 and 3.2 it is now possible attribute any modifications in the Bitstream to its corresponding block.

3.6 Determining Trojan Attributes

The complexity of Integrated Circuit designs and their corresponding trojans requires a more human-friendly scope. The taxonomy described in Chapter 2 provides a series of thirty-three attributes which a trojan may or may not posses. These attributes allow us to readily describe, interpret and discuss trojans in a more comfortable and useful way. To employ this taxonomy a trojan must be analyzed; characteristics observed can either infer or provide direct evidence of possession of attributes. Though it is desirable to be able to observe and directly extract the attributes a trojan possesses it is not always possible. In [17] a matrix, **R**, is provided which describes the relationships between each of the thirty-three attributes in the taxonomy. When it is not possible to directly determine the presence of certain attributes, this relation matrix is used to infer their existence. The analysis stage of the automated trojan detection technique provided by this work begins by extracting those attributes that are directly observable then using matrix **R** to infer the existence of the remainder.

3.6.1 Extraction Methods

Observed Location Attributes

The presence of attributes in the *Location* category are directly observable from the results of the component mapping method described in section 3.5. *Xilinx* tiles conform to purpose-specific groups or block types which were discussed in section 3.4. These block types contain sub-types that perform actions which pertain to the *Location*, category.

1. The **Processor** attribute pertains to the core functionality of the design logic. It can be awarded for presence of a modified CLB tile or Interconnect tile.
2. The **Memory** attribute can be awarded for the presence of modified BRAM components.
3. The **IO** attribute can be awarded for presence of modified IOB tiles.
4. The **Power Supply** attribute can be awarded for the presence of modified interface or configuration tiles.
5. The **Clock Grid** attribute can be awarded for modified clock tiles.

Scatter Score Method

The gate-array configuration of components in *Xilinx* FPGAs allows for an analytical method of determining attributes in the *Physical Layout* category. The "Scatter Score" method uses the grid coordinates of components to derive a numerical score rating for the size, position, and augmentation of configured tiles. Tiles are assigned global coordinates that represent their horizontal and vertical positions within the gate array denoted x and y respectively. These values can then be used to strongly infer the presence of *Physical Location* attributes.

The golden chip is first analyzed. The set of all tiles which are configured in the golden design is found and a series of numerical descriptors are computed.

$$n = \sum_{x=0}^{X} \sum_{y=0}^{Y} T_{xy} \tag{3.3}$$

where:

n = Number of all **configured** tiles
X = The column width of the gate-array
Y = The number of rows of the gate-array
T = A configured tile

$$a_x = \frac{1}{n} \sum_{x=0}^{n} T_x \tag{3.4}$$

$$a_y = \frac{1}{n} \sum_{y=0}^{n} T_y \tag{3.5}$$

$$\sigma_x = \sqrt{\frac{1}{n} \sum_{x=0}^{X} (x_i - a_x)^2)} \tag{3.6}$$

$$\sigma_y = \sqrt{\frac{1}{n} \sum_{y=0}^{Y} (y_i - a_y)^2)} \tag{3.7}$$

where:

a_x = The average x coordinate of configured tiles
a_y = The average y coordinate of configured tiles
T_x = The x coordinate of a configured tile
T_y = The y coordinate of a configured tile
σ_x = The standard deviation of the x coordinate of configured tiles
σ_y = The standard deviation of the y coordinate of configured tiles

Equations 3.4 and 3.5 are used to create a rating known as the Position Median in Equation 3.8. The Position Median value provides a simple descriptor for where in the gate array the design is centralized. The Scatter Score in Equation 3.9 describes how spread out or, *clustered* the design is.

$$M_{xy} = (a_x,\ a_y) \tag{3.8}$$

$$S_{xy} = (\sigma_x,\ \sigma_y) \tag{3.9}$$

where:

M_{xy} = The Position Median
S_{xy} = The Scatter Score

The results of the component mapping method described in section 3.7 are used to generate the set of all tiles reconfigured by the trojan. The set of reconfigured tiles can be said to contain three subsets: the subset of tiles activated by the trojan, those deactivated and those modified. The results of the golden design analysis, the subsets, and the numeric descriptors can be used to discern which of the *Physical Location* attributes the trojan possesses. The *Physical Location* category contains six attributes. These six can be considered three pairs; a trojan exhibits one attribute from each pair.

1. **Large or Small** (attributes 23 or 24): According to [17], small trojans are defined as those that are nearly impossible to detect via power consumption. From this it can be said that 'small' trojans occupy minimal resources. Trojans where the number of reconfigured tiles is less than 5% of the number of tiles in the golden design are considered small. Other wise they are attributed as large.
2. **Changed Layout or Augmented** (attributes 25 or 26): A 'changed layout' trojan is such that only tiles that are configured by the golden design are reconfigured. An augmented trojan is where additional layout is added. The presence of 'activated' or 'deactivated' tiles indicates an augmented trojan.
3. **Clustered or Distributed** (attributes 27 or 28): The trojan is considered to be clustered when the standard deviation of the reconfigured tile positions is less than 15%; distributed otherwise.

Insertion and Abstraction Attributes

The linear nature of the manufacturing life-cycle implies a propagation of effects. Design decisions, flaws and improvements affect later stages. As discussed in section 3.1, for the purposes of this automated FPGA trojan detection method it is assumed that the only non-trustworthy stage in the life-cycle is fabrication. In other words, the trojan was inserted in the third-party fabrication stage. Due to the propagating nature of the life-cycle the effects of the modifications made in the fabrication stage (attribute 3) are felt in the testing (attribute 4) and assembly (attribute 5) stages. Hence, due to the assumptions made in this method it can be said that this trojan possesses insertion category attributes 3, 4 and 5.

The abstraction category pertains to the level at which a trojan resides [24, 25]. Due to the nature of FPGA design a few additional assumptions can be made when selecting abstraction category attributes. Hardware Description Language (HDL) is used to generate a "programming", or "configuration" file which is downloaded into the FPGA. The configurable functionality is dictated by this file. The hardware of an FPGA chip can be modified to hide trojans at a lower abstraction level. The detection of such trojans requires advanced reverse engineering techniques and is beyond the scope of this method. Hence, it can be said that trojans discovered by this method are not able to obtain abstraction category attributes that pertain to the physical hardware of a chip. This removes the possibility of *Logic* (attribute 9), *Transistor* (attribute 10), and *Physical* (attribute 11). The possession of the remaining abstraction category attributes are determined by analysis of the relation matrix discussed in section 3.6.2.

3.6.2 Relation Matrix Use

The relation matrix $\mathbf{R}$ provided in chapter 2 can be summarized by a series of submatrices.

$$\mathbf{R} = \begin{bmatrix} \mathbf{R_1} & \mathbf{R_{12}} & 0 & 0 \\ 0 & \mathbf{R_2} & \mathbf{R_{23}} & 0 \\ 0 & 0 & \mathbf{R_3} & \mathbf{R_{34}} \\ 0 & 0 & 0 & \mathbf{R_4} \end{bmatrix}$$

Each submatrix is a matrix of binary values which describe the relationship between two attributes. Refer to section 2.1 for a detailed description. Submatrix $\mathbf{R_{34}}$ relates the *Location* category attributes to *Properties* attributes. Since the *Location* attributes are directly observable they can be used to generate a list of *Property* attributes which the trojan may possess. This list is compared to results generated by any additional property attribute extraction methods, such as the *Scatter Score* method described in section 3.6.1.

Once a list of *Property* attributes has been solidified submatrix $\mathbf{R_{23}}$ is used to infer the *Abstraction* category attributes. As mentioned in section 3.6.1 *Abstraction* category attributes nine, ten and eleven are assumed inaccessible. Attributes six, seven and eight can be inferred based on the presence *Property* attributes.

Chapter 4

Software Implementation

4.1 Introduction

The Integrated Circuit (IC) life-cycle was introduced in section 3.1. It is the process where ICs begin as ideas and end as products to be sold. The process is expensive, delicate, and consumes thousands of man hours. To streamline this process and minimize risk the industry is more frequently relying on third-party sources for a variety of steps [26]. For manufacturers it is imperative to ensure that their products are not at risk while in the hands of these outsourced services. Conversely, it is undesirable for the the required security and quality assurance measures to mitigate the benefit gained by using these services.

The FPGA Trojan Detector can provide manufacturers additional security that takes only minutes to complete; the analysis requires only a few button clicks. It was built to be a stand-alone application. Manufacturers and researchers alike are already required to purchase large and expensive pieces of equipment; much of which requires arduous installation and configuration. The FPGA Trojan Detector is light-weight, cross platform and does not require installation. Its application can greatly improve security at little to no cost.

4.2 Technologies Used

4.2.1 *Xilinx*

Xilinx is one of the two largest manufacturers of FPGAs; their devices are considerably more popular than their competitors. The configuration of devices employs a well known series of steps referred to as the *Xilinx* 'tool-chain' [22]. The 'tool-chain' not only compiles user designs and constraints into the configuration Bitstream but performs a series of complex operations to optimize designs and effectively implement them on any *Xilinx* model of the user's choosing.

1. **sch2hdl**: used to convert schematic designs to Hardware Description Language (HDL). This stage is optional as many choose to develop directly in HDL
2. **XST**: *Xilinx* Synthesis Tools, a package that parses, optimizes and compiles the HDL code.
3. **MAP**: A *Xilinx* tool used to calculate the optimal routing for the finalized design.
4. **PAR**: used to place and route the design in the specific *Xilinx* device.
5. **ngdbuild**: A tool used to build a NETLIST.
6. **trce**: used to perform timing and performance analysis.
7. **Bitgen**: The final stage used to generate the configuration Bitstream which will configure the device.

Each step produces a series of files that are used for a variety of purposes. Often the resultant files are used by the subsequent step but some are intended for user information. The ngdbuild tool generates what is known as the netlist for the design. The netlist is a description of the connectivity of the circuit implemented on the device. The generated netlist is in a non human-readable format in an 'ndc' file. Fortunately, *Xilinx* provides an additional tool called xdl2ndc which allows the conversion to the human-readable *Xilinx* Design Language (XDL).

***Xilinx* Design Language (XDL)**

The *Xilinx* Design Language (XDL) is a human-readable ASCII format; though it is not actively part of the 'tool-chain' it is considered a native netlist format for describing and representing FPGA designs [27]. In a netlist, a 'part' is a human-defined component which is to be implemented. Netlists either contain or refer to descriptions of the parts used and where in the device they are implemented. When a part is used it is called an 'instance'; thus each 'instance' has a definition, sometimes referred to as a master.

An XDL file contains two sections, the instance placement and configuration section, and the net routing section. The placement and configuration section provides a list of every instance in a design. Their descriptions include all of the configuration details required for their implementation on a component (power settings, logic, timing configurations... etc). In the *Xilinx* jargon, a net refers to any electrical path between two components; more specifically, a net describes a communication channel. The gate-array of a *Xilinx* device is composed of a grid of wires. These wires can be fused together by Programmable-Interconnect-Points (PIP) to make a useful connection between two components, thus creating a net. The output-pin of a net receives the signal from the transmitting component while the input-pin delivers the signal to the corresponding receiver. The PIPs in-between the end-points dictate the path the signal takes between the two components. The net routing section of the XDL file describes every path in the design. Combined, these two sections completely describe a design and how it is implemented. The information provided by the XDL file is used by the FPGA Trojan Detector to relate any of the effects of a discovered trojan to the user's intended operation.

XDLRC

The XDL file describes the design implemented on a device. From this a lot of information regarding the composition of a *Xilinx* device can be learned but it does not provide the entire description; only what has been used by the design. The xdl2ndc command line tool provides an option to generate a resource report file referred to as an XDLRC file. A XDLRC file describes the entire architecture of a *Xilinx* FPGA.

Listing 4.1: A hierarchical XDLRC resource description of a Spartan 6 FPGA consisting of a header, a tile section, and a trailing device summary [27]

```
#Header section
(xdl_resource_report v0.2 xc6slx16csg324-3 spartan6
# Device Level Dimensions
(tiles 73 62
...
   #Configurable logic block with two slices
   (tile 4 6 CLEXL_X1Y61 CLEXL 2
      (primitive_site SLICE_X0Y61 SLICEL internal 45
         (pinwire A1 input L_A1)
      ...
      (primitive_site SLICE_X1Y61 SLICEX internal 43
      ...
      (pinwire D output XX_D)
   ...)
   # Interconnect tile
   (tile 4 5 INT_X1Y61 INT 1
   ...
      (wire EE2B0 2
         (conn CLEXM_X2Y61 CLEXM_EE2M0)
         (conn INT_BRAM_X3Y61 EE2E0)
      ...
      # Programmable Interconnect Points
      (pip INT_X1Y61 EE2E0 -> EE2B0)
      (pip INT_X1Y61 EE4E0 -> EE2B0)
      (pip INT_X1Y61 EL1E_S0 -> LOGICIN_B9)
   ...)
# summary
(summary tiles=4526 sites=5378 sitedefs=46 numpins=157962 numpips=5782505))
```

Listing 4.1 provides an example of the XDLRC format. The report begins with a header describing the device. It then reports that the overall architecture contains a matrix of tiles, 73-wide and 62-tall. Further down is a configurable logic block at coordinate (4,6) and an interconnect tile at coordinate (4,5). Each of these tile descriptions provide details of the subcomponents they contain, pinwires, PIPs, slices...etc. The xdl2ndc tool is capable of generating a variety of different XDLRC files for a device, ranging in the level of detail. The smaller files can be around 10MB while the fully detailed descriptions can reach 7GB in size.

4.2.2 Java

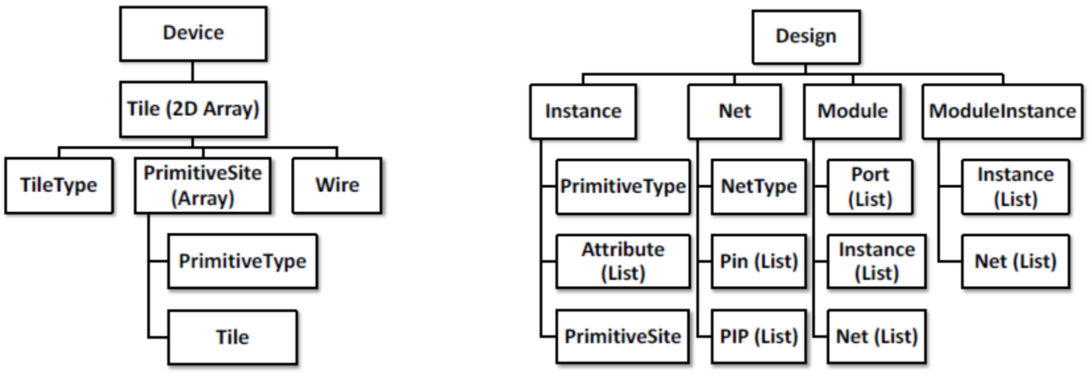
Java is a powerful and general-purpose programming language. It is specifically designed to be as independent as possible. The original developer, James Gosling, intended the language to allow developers a comfortable implementation experience with seamless deployment. The custodians of the Java language, Sun Microsystems, promote the slogan 'write-once, run anywhere'. FPGA Trojan Detector was written in Java primarily in order to interface with the API known as *RapidSmith* which is described in section 4.2.3. However, Java additionally allows the FPGA Trojan Detector to be a compact, cross-platform application. The Java language provides a native Graphical User-Interface (GUI) toolkit known as *Swing*. *Swing* is an API that is part of Oracle's Java Foundation Classes; in other words it is readily available to all users of Java. It provides a simple to use programming structure for creating sophisticated UI. The FPGA Trojan Detector employs Java and Java *Swing* . This implies that the only requirement for a user to take advantage of the power of the FPGA Trojan Detector is to have Java installed; which most modern machines do.

4.2.3 ***RapidSmith***

RapidSmith is an Application Programming Interfaces (API) written in Java that enables Computer Aided Design (CAD) tool creation for *Xilinx* FPGAs [28]. Its purpose is to be used as a rapid prototyping platform for experimentation and research. The code is free to use and readily accessible. It was chosen as a supporting library for the FPGA Trojan Detector for several reasons. First, the code base provides a series of class structures that astutely mirror the architecture of *Xilinx* devices. Secondly, it provides ready-made tools for extracting the configuration frames from Bitstream files. Bitstream files are long binary sequences, without the tools provided by *RapidSmith* the analysis of these files becomes an arduous task. Finally, and most importantly, the creators of *RapidSmith* have developed a means of condensing XDLRC files into a greatly compressed format referred to as a 'database' file. The FPGA Trojan Detector requires considerable detail of an FPGA's architecture in order to accurately described the effects a trojan has on a design. As mentioned in section 4.2.1, the fully detailed XDLRC files can reach 7GB in size. Working with text-based files of this size detriments performance to an unacceptable level. The makers of *RapidSmith* have developed a compression scheme which converts the XDLRC files into a non human-readable format. This scheme reduces the size of the XDLRC files from 7GB to 1.3MB. The API provides the necessary interface classes for querying the compressed files. With these compressed fies, *RapidSmith* gives the FPGA Trojan Detector the ability to analyze the effect any discovered trojan has on a device.

Class Structure

Building off of the information provided by the XDLRC architecture files, *RapidSmith* provides a class structure that enables developers an easy to use interface for working with FPGAs. As described in section 3.2, the gate-array of an FPGA is composed of blocks made up of tiles. These tiles are further composed of sub-components collectively referred to as ’primitives’. Figure 4.1 shows the top-down construction of the *RapidSmith* class structure. The FPGA Trojan Detector employs both the ’Device’ and ’Design’ classes shown in Figures 4.1a and 4.1b respectively. The Bitstream for the Golden and Target devices are read into memory. A utility class which parses the Bitstream into its configuration frames determines the model of the devices used. The ’Device’ class is configured to the correct model and then is loaded with the architecture details from the compressed XDLRC file. The complete description of the architecture is read and loaded; every tile, primitive, wire, and how they all interconnect is stored in memory. The Golden XDL file is then loaded. The XDL file is read and the ’Design’ class is similarly loaded with the user-defined design. Not



(a) The Device Class-Structure (b) The Design Class-Structure

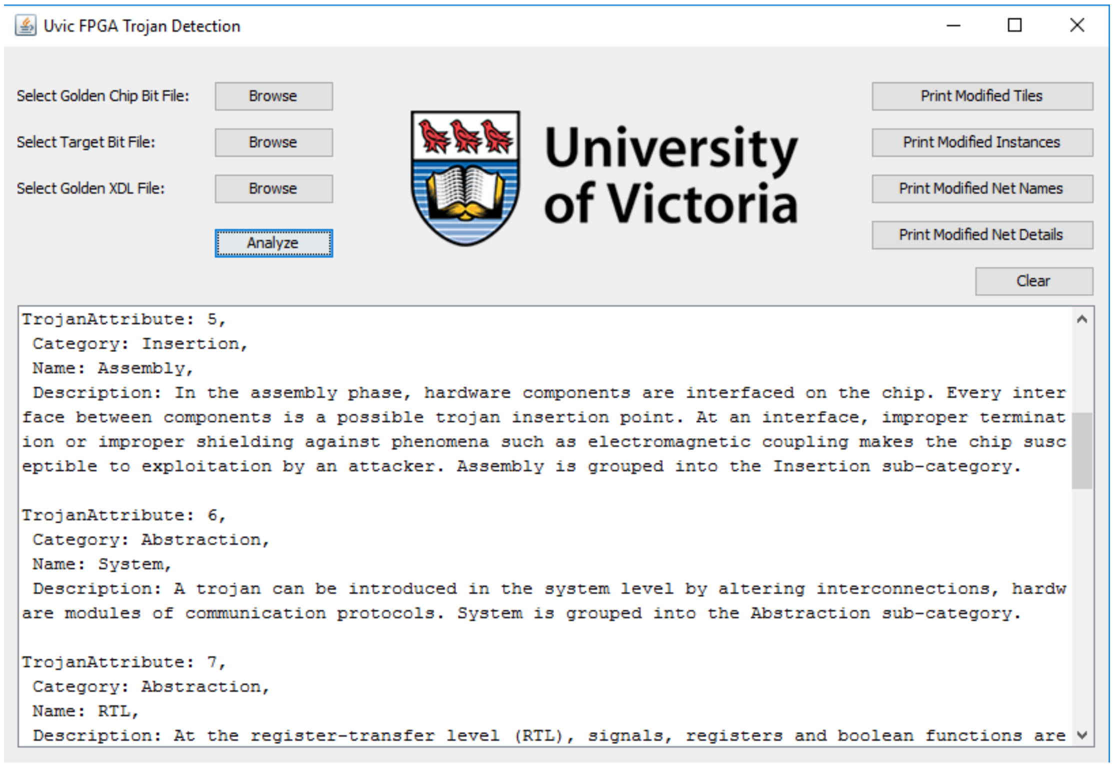
Figure 4.1: The *RapidSmith* Class Hierarchy [29]

shown in Figure 4.1 are a series of utility classes. The Bitstream parser class reads and interprets the Bitstream files. It uses a ’Frame’ class to organize the Bitstream file into an array of objects that adhere to the configuration pattern described in section 3.2. The frame objects are populated with the frame’s 32-bit address, an array of 32-bit words which make up the frame and a series of helper methods.

4.3 FPGA Trojan Detector

4.3.1 User-Interface (UI)

Java and Java *Swing* provide an easy to use User-Interface development system which produces light-weight, portable and cross platform applications. The FPGA Trojan Detector employs *Swing* to provide a very simplistic and easy to use interface which



```
TrojanAttribute: 5,
 Category: Insertion,
 Name: Assembly,
 Description: In the assembly phase, hardware components are interfaced on the chip. Every inter
face between components is a possible trojan insertion point. At an interface, improper terminat
ion or improper shielding against phenomena such as electromagnetic coupling makes the chip susc
eptible to exploitation by an attacker. Assembly is grouped into the Insertion sub-category.

TrojanAttribute: 6,
 Category: Abstraction,
 Name: System,
 Description: A trojan can be introduced in the system level by altering interconnections, hardw
are modules of communication protocols. System is grouped into the Abstraction sub-category.

TrojanAttribute: 7,
 Category: Abstraction,
 Name: RTL,
 Description: At the register-transfer level (RTL), signals, registers and boolean functions are
```

Figure 4.2: The User-Interface of the FPGA Trojan Detector

can be seen in Figure 4.2. To perform an analysis the user must use the UI to navigate to and select the Golden Bitstream, the Target Bitstream and the Golden XDL file via the three browse buttons on the left. Once loaded the detection process can be launched via the 'Analyze' button. On the right the user is able to review information specific to the trojan, which will be described in section 4.3.2.

If a modification is discovered, the FPGA Trojan Detector analyzes it and reports a list of attributes from the taxonomy of Chapter 2 which describe it. Results are printed textually in the text window. The attribute id-number, name, category and a brief description of the attribute are provided. After the analysis the user is able to use the 'Print Modified Tiles' feature. Any tile in the architecture which corresponds to a modified tile is printed to the text window. This provides the user with the exact location on the their device where modification took place. The 'Print Modified Instances' feature corresponds each modified tile to the design-instance it belongs to. *Xilinx* provides schematic design views which correspond instance names to the user's design. This can be used by the user to refer back to their design and view the trojan's effects from a high abstraction perspective. The 'Print Modified Net Names' provides only the names of each channel that has been modified. Again this allows the user to relate modifications to their design. The 'Print Modified Net Details' button provides both the names of modified nets and their corresponding list of primitives. Finally, the 'Clear' button is a manual method of removing the text from the text-window.

4.3.2 Operation

Figure 4.3 gives a visual representation of the operating procedure of the FPGA Trojan Detector. To perform analysis the user must submit the Golden and Target Bitstream files. As seen on the left of Figure 4.3, these large binary files are parsed by a utility class provided by *RapidSmith* . The parsing utility is able to detect the model of the device by scanning the 'header' information in the files. Different *Xilinx* models use different configuration patterns. The Bitstream of different models may differ in a variety of features including frame size, ordering, address structure and more.

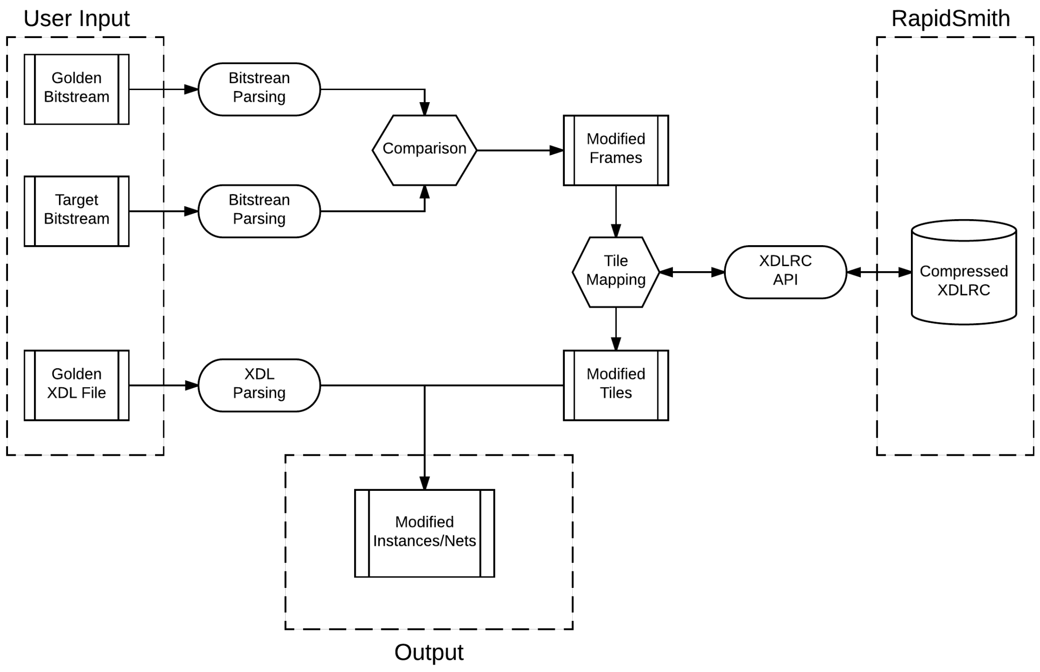


Figure 4.3: Overview of Functional Operation

Knowing the device, the parser is able to refer to additional utility files and accurately extract the frames. The Golden and Target frames are extracted and stored in memory as arrays. The two arrays are then passed to a comparison process. Each frame is compared bit by bit. A trojan-free circuit will have no differences between the Golden and Target files. Any modified frames discovered are stored in a 'Modified_Frame' class. The 'Modified_Frame' object stores the Golden frame data, the Target frame data, the address of the frame and more. They are then used by the 'Component Mapping' method described in section 3.5 to determine where in the architecture the modifications have been made.

The component mapping procedure relies heavily on the compressed XDLRC files and the corresponding API described in section 4.2.1 as well as the 'Device' class structure shown in Figure 4.1a. Component mapping produces a list of objects of the 'Modified_Tile' class. This class contains a reference to the corresponding tile in the XDLRC file, the Golden and Target words (which differ), the frame address in which it occurred, the sub-column type and the column type. The user is also required to enter the Golden XDL file. As described in section 4.2.1 the XDL file contains the Netlist for the Golden design. This file describes not only how each of the users components are built and interconnected but where on the device they are placed. The modified tiles, as shown in Figure 4.3, must be correlated to the users design to extract useful information. The instances and nets described in the XDL file list the exact tile and primitive in which they are placed. The *RapidSmith* XDL parser reads the user's design and populates the 'Design' class shown in Figure 4.1b. The list of 'Modified_Tile' objects are compared to the tile placings of the instances and nets in the populated 'Design' object. Any instances or nets that are placed in 'modified' tiles are flagged and stored. The user is able to view which instances and nets of their design have been modified by the four buttons on the right of the UI shown in Figure 4.2.

Chapter 5

Results and Discussion

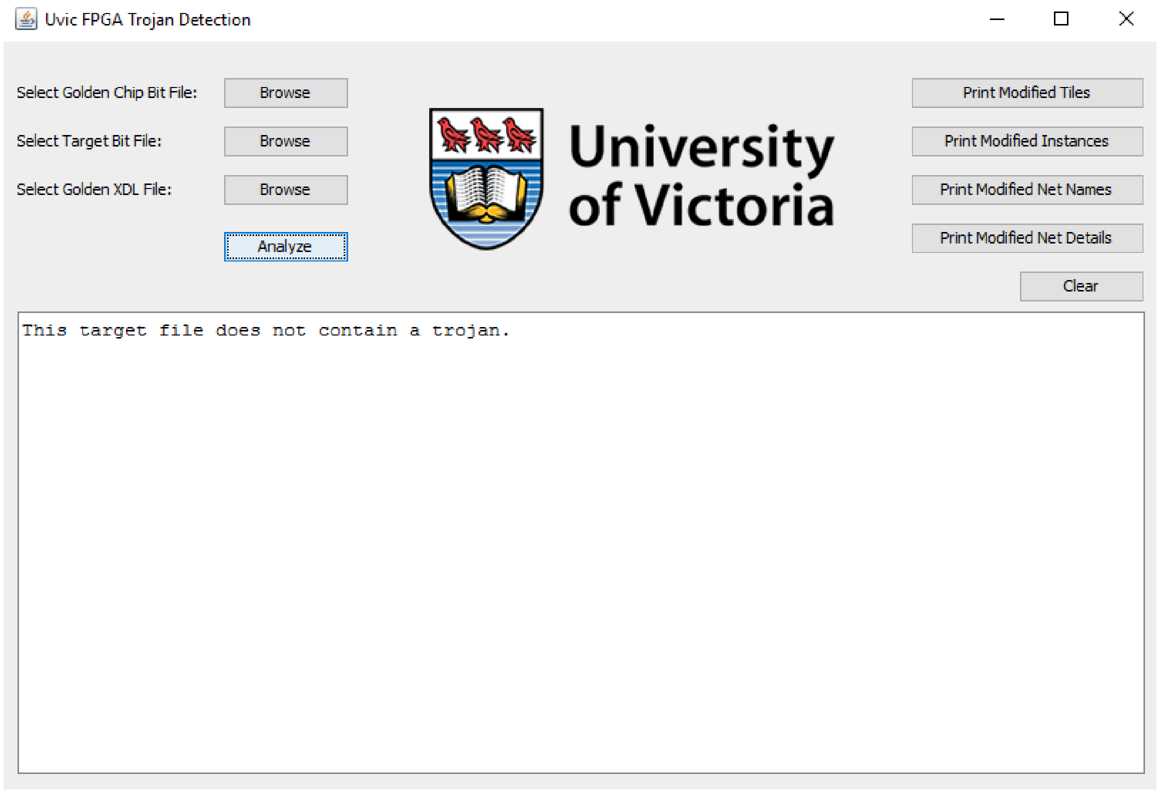
5.1 Results

5.1.1 Methodology

Integrated Circuit manufacturers distribute their product to devices which control everything from cell phones to fighter jets. Those who employ Field Programmable Gate-Arrays require the means to ensure that their chips function as intended. The FPGA Trojan Detector provides manufacturers a quick means of ensuring that they are providing a safe and secure product to their customers. To demonstrate its potential, it has been tested using a series of benchmarks. These benchmarks are included in the FPGA Trojan Detector application package as an example for users.

5.1.2 Priority Decoder

At the IEEE International Symposium on Circuits and Systems (ISCAS) in 1989, F. Brglez and H. Fujiwara presented a series of FPGA benchmark circuits originally written in C [30]. These benchmarks were accessed and a small 'Priority Decoder' circuit was chosen [31]. The provided verilog code for the decoder benchmark was synthesized on a Virtex-5 XC5VLX155 and the generated Bitstream and XDL files were acquired. The priority decoder Bitstream file was fed to both the Golden and Target inputs to the FPGA Trojan Detector. Feeding the same Bitstream file to both inputs replicates the occurrence where the third-party fabrication house made no modifications; the Bitstream extracted from the Target device is exactly the same as the Golden. It is expected that the FPGA Trojan Detector returns a result in-



```
This target file does not contain a trojan.
```

Figure 5.1: Priority Decoder Analysis Results

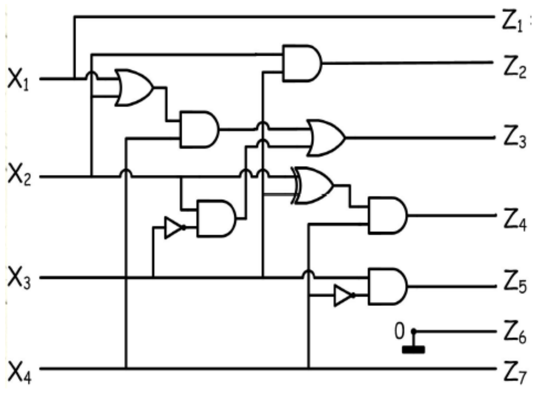
dicating that there is no trojan present. As can be seen in Figure 5.1, the FPGA Trojan Detector successfully analyzed the Bitstreams and determined that there was no trojan present, as expected.

5.1.3 User Authentication Circuit

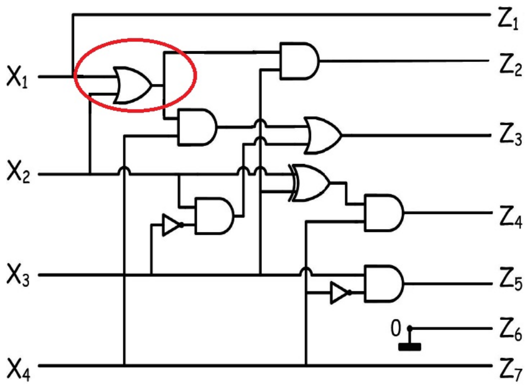
Consider a circuit designed to compute a function $F(x)$ for a system to authenticate user-password pairs x and $F(x)$. The system performs the arithmetic operation $F(x) = x^2$ to validate users. The customer wishes to provide access to ten users labeled I_0 to I_9. To identify all ten users, four input bits are required, x_1 to x_4. The largest function output is 81 meaning seven bits are required for output, Z_1 to Z_7, as illustrated in Fig. 5.2a. A trojan can be inserted into this circuit as shown in Fig. 5.2b (called a backdoor trojan). The outputs of the original and infected circuits are compared in Table 5.1. A simple test will show that the circuit outputs the desired $F(x) = x^2$ for each of the users. However, upon closer inspection it is noted that the inputs corresponding to $x = 10$ to $x = 15$ are not used; there are no clients

Table 5.1: Outputs of the Circuits in Figs. 5.2a and 5.2b [17]

		Inputs					Circuit A								Circuit B							
		X_1	X_2	X_3	X_4	X	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	$F(x)$	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	$F(x)$
Inputs	I_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	I_1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
	I_2	0	0	1	0	2	0	0	0	0	1	0	0	4	0	0	0	0	1	0	0	4
	I_3	0	0	1	1	3	0	0	0	1	0	0	1	9	0	0	0	1	0	0	1	9
	I_4	0	1	0	0	4	0	0	1	0	0	0	0	16	0	0	1	0	0	0	0	16
	I_5	0	1	0	1	5	0	0	1	1	0	0	1	25	0	0	1	1	0	0	1	25
	I_6	0	1	1	0	6	0	1	0	0	1	0	0	36	0	1	0	0	1	0	0	36
	I_7	0	1	1	1	7	0	1	1	0	0	0	1	49	0	1	1	0	0	0	1	49
	I_8	1	0	0	0	8	1	0	0	0	0	0	0	64	1	0	0	0	0	0	0	64
	I_9	1	0	0	1	9	1	0	1	0	0	0	1	81	1	0	1	0	0	0	1	81
Undefined	I_{10}	1	0	1	0	10	1	0	0	0	1	0	0	68	1	1	0	0	1	0	0	100
	I_{11}	1	0	1	1	11	1	0	1	1	0	0	1	89	1	1	1	1	0	0	1	121
	I_{12}	1	1	0	0	12	1	1	1	0	0	0	0	112	1	0	1	0	0	0	0	80
	I_{13}	1	1	0	1	13	1	1	1	1	0	0	1	121	1	0	1	1	0	0	1	89
	I_{14}	1	1	1	0	14	1	1	0	0	1	0	0	100	1	1	0	0	1	0	0	100
	I_{15}	1	1	1	1	15	1	1	1	0	0	0	1	113	1	1	1	0	0	0	1	113



(a) A Simple User Authentication Circuit



(b) The Back-door Trojan

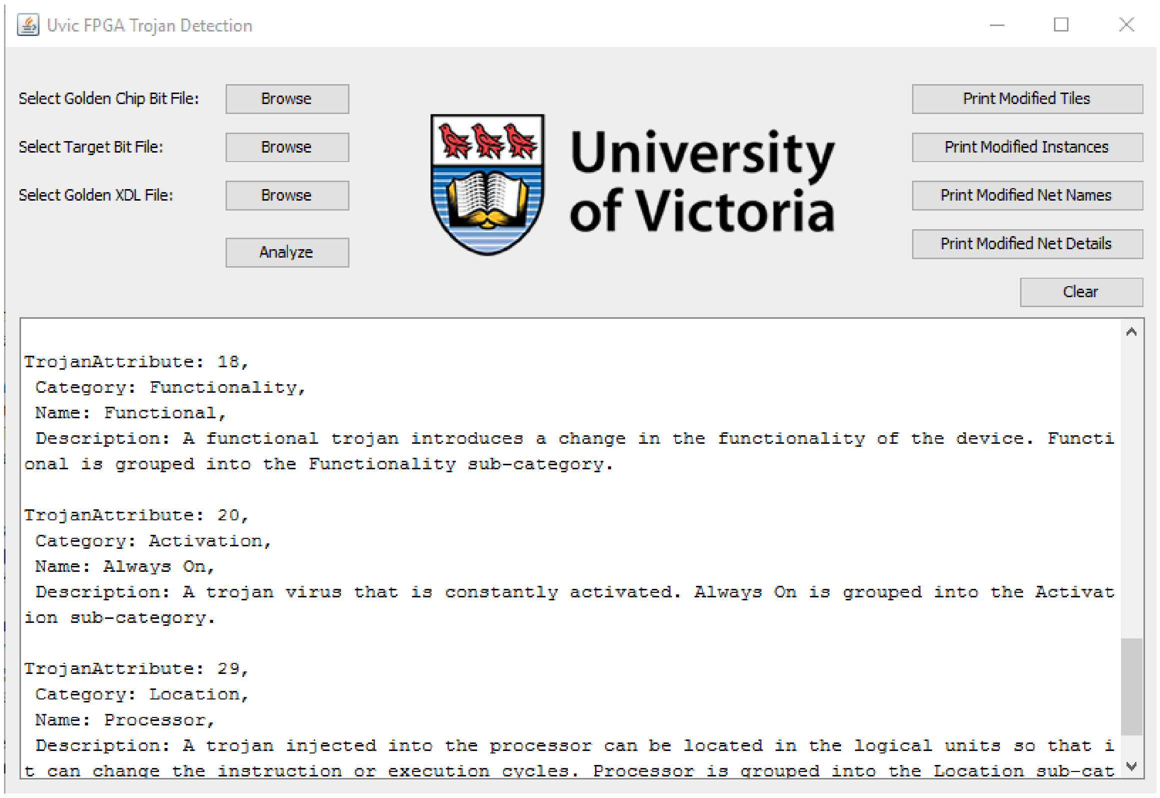
Figure 5.2: The User Authentication Circuit: Clean and Trojan

occupying those identifications. These unused inputs are referred to as 'dont-cares', meaning that it is not important to the function of the circuit what their corresponding output is. Don't-care cases are a typical vulnerability which can be exploited by an attacker. Under the 'Circuit A' column of Table 5.1 it can be seen that I_{10} and I_{11} produce results 68 and 89 respectively. These results are not correct according to $F(x) = x^2$. This is an intended result by the circuit designer to add security for these don't care conditions. If an attacker is able to make the modification in Fig. 5.2b, inputs 10 and 11 will now produce results 100 and 121 which are correct according to $F(x) = x^2$. This can be seen under the 'Circuit B' column of Table 5.1. Now, the attacker has built a 'back-door' into the circuit. The original 10 users' authentication is not altered but inputs 10 and 11 provide unauthorized access to the attacker.

The circuits shown in Figure 5.2 were implemented using *Xilinx* 's schematic designer and placed on a Virtex-5 240T (XC5VSX240T). The Bitstreams and XDL files were generated and input to the FPGA Trojan Detector. The analysis detected a trojan and output the list of determined attributes as can be seen in Figure 5.3.

The system output the attributes:

- Attribute 3: Fabrication
- Attribute 4: Testing
- Attribute 5: Assembly
- Attribute 6: System
- Attribute 7: RTL
- Attribute 12: Change in Functionailty
- Attribute 17: Combinational



```
TrojanAttribute: 18,
 Category: Functionality,
 Name: Functional,
 Description: A functional trojan introduces a change in the functionality of the device. Functi
onal is grouped into the Functionality sub-category.

TrojanAttribute: 20,
 Category: Activation,
 Name: Always On,
 Description: A trojan virus that is constantly activated. Always On is grouped into the Activat
ion sub-category.

TrojanAttribute: 29,
 Category: Location,
 Name: Processor,
 Description: A trojan injected into the processor can be located in the logical units so that i
t can change the instruction or execution cycles. Processor is grouped into the Location sub-cat
```

Figure 5.3: Results of The Authentication Circuit Trojan Detection

- Attribute 18: Functional
- Attribute 20: Always On
- Attribute 24: Small
- Attribute 26: Augmented
- Attribute 27: Clustered
- Attribute 29: Processor

As expected the results state that the trojan was inserted in the Fabrication phase (3); it is the earliest stage in the 'insertion' phase produced indicating it as the source. The effects of this modification propagate to the Testing (4) and Assembly (5) phases as expected. The modifications reach both the System (6) and Registry Transfer Level (RTL) (7) abstraction levels. Since the modifications were made using the schematic designer provided by *Xilinx* which works in the RTL level, these results are as expected. The results indicate that the trojan Changes Functionality (12). This agrees with the modification to the values listed in Table 5.1. The trojan does not take affect over multiple clock cycles; this indicates it is composed of only Combinational (17) circuitry which is reflected in the results. The trojan did not modify power levels or operation configurations, only design configurations. This indicates that the trojan can be described as Functional (18), not Parametric (19); this agrees with the results. Since the modification made a permanent alteration to the internal wiring of the circuit it can be said to be Always On (20). The modified values for input $x = 10$ or $x = 11$ are always available and not activated. This is consistent with the returned results. The trojan changed only minor routing configurations in the circuit designs, this required the alteration of only a few tiles. The new route required the activation of tiles which were not active in the Golden design; further, all of the modified tiles are nearby those in the Golden design. With these observations it can be said that this trojan exhibits Physical Layout attributes Small (24), Augmented (26) and Clustered (28). All of the expected Physical Layout attributes were correctly determined by the Scatter Score method of section 3.6.1. Finally, all of the tiles modified by the trojan belong to major block type 0: Logic Type. These tiles only affect the internal processing of the circuit. No IOB, Clock or BRAM tiles were modified. This is reflected by the fact that only Location attribute, Processor (29), was returned by the analysis. The results observed by the experiment conformed with the experiments expectations demonstrating the accuracy of the method. The entire analysis takes less than a minute to perform.

5.1.4 AES-T100

In 2013 H. Salmani, M. Tehranipoor and R. Karri published a discussion on the design and development of FPGA trojan benchmarks. They collaboratively developed a series of verilog, VHDL and virtual machines that demonstrated effective creation of testable benchmarks. They took the benchmarks they created and published them on a website they created called *Trust-Hub*. The benchmarks provided are organized by a taxonomy they proposed in [32]. Their taxonomy is slightly different than the one used by the FPGA Trojan Detector which was proposed in [17], however the two are conceptually similar. To demonstrate the efficacy of the FPGA Trojan Detector a benchmark named ’AES-T100’ was chosen. The supporting documents describe this trojan as follows:

> The Trojan leaks the secret key from a cryptographic chip running the AES algorithm through a covert channel. The channel adapts the concepts from spread spectrum communications (also known as Code-Division Multiple Access (CDMA)) to distribute the leakage of single bits over

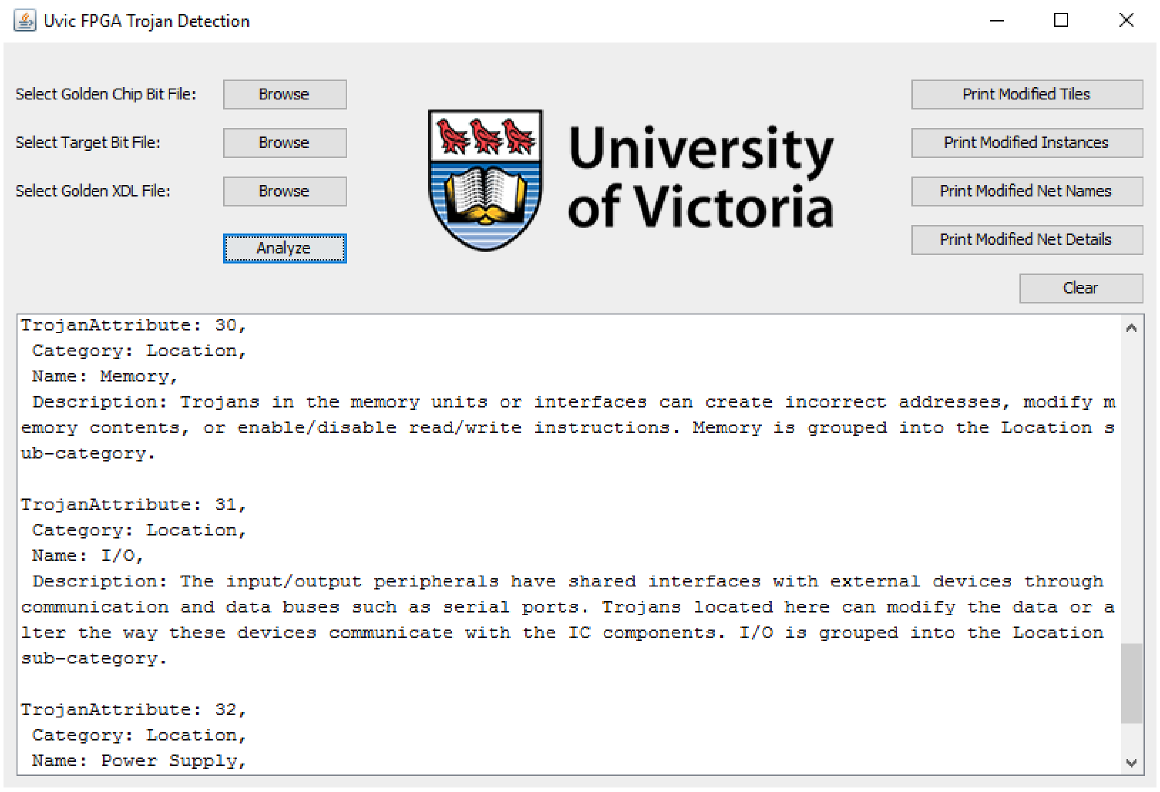
> many clock cycles. The Trojan employs this method by using a pseudo-random number generator (PRNG) to create a CDMA code sequence, the PRNG initialized to a predefined value. The code sequence is then used to XOR modulate the secret information bits. The modulated sequence is forwarded to a leakage circuit (LC) to set up a covert CDMA channel in the power side-channel. The LC is realized by connecting eight identical flip-flop elements to the single output of the XOR gate to mimic a large capacitance. [33]

From the description it is reasonable to expect certain results from the FPGA Trojan Detector. The description states that the trojan 'leaks the secret key'. From this we should expect our results to contain Effect attribute Information Leakage (13). Information being leaked from a device will need a means to be transmitted to the attacker. Location attribute IO (31) may be observed. It then states that it leaks 'single bits over many clock cycles'. This suggests that the trojan exhibits some form of Sequential Logic (16). This may or may not require modification to clock tiles; Location attribute Clock Grid (33) may be observed. It then states that the PRNG is initialized to a predefined value; initialization requires the value be stored in memory. Location attribute Memory (30) should be expected. The trojan then uses a 'power side-channel' as a communication channel. This will require modification to power tiles; Location attribute Power Supply (32) should be expected.

The source code for the Golden and Target designs were downloaded from *trust-hub.org* and synthesized on a Virtex-5 240T (XC5VSX240T). The analysis results were successfully found by the system as seen in Figure 5.4.

The FPGA Trojan Detector output the following attributes:

- Attribute 3: Fabrication
- Attribute 4: Testing
- Attribute 5: Assembly
- Attribute 6: System
- Attribute 7: RTL
- Attribute 13: Information Leakage
- Attribute 16: Sequential
- Attribute 18: Functional
- Attribute 20: Always On
- Attribute 24: Large
- Attribute 26: Augmented



```
TrojanAttribute: 30,
 Category: Location,
 Name: Memory,
 Description: Trojans in the memory units or interfaces can create incorrect addresses, modify m
emory contents, or enable/disable read/write instructions. Memory is grouped into the Location s
ub-category.

TrojanAttribute: 31,
 Category: Location,
 Name: I/O,
 Description: The input/output peripherals have shared interfaces with external devices through
communication and data buses such as serial ports. Trojans located here can modify the data or a
lter the way these devices communicate with the IC components. I/O is grouped into the Location
sub-category.

TrojanAttribute: 32,
 Category: Location,
 Name: Power Supply,
```

Figure 5.4: Results of Analysis on the AES-T100 Benchmark

- Attribute 27: Distributed
- Attribute 29: Processor
- Attribute 30: Memory
- Attribute 31: IO
- Attribute 32: Power Supply
- Attribute 33: Clock Grid

Again, it is assumed that this trojan was inserted during the fabrication phase; due to this the effects propagate to the Testing (4) and Assembly (5) phases. Again, the trojan resides in the System (6) and RTL (7) abstraction levels as was expected. Attributes 13, 16, 30, 31, 32 and 33 appear in the analysis results as expected. The Scatter Score method describes this trojan as Large, Augmented and Distributed. These results seem to correspond well with the description. The addition of the leakage circuit and the PRNG would be non-trivial addenda likely requiring the activation of considerable resources resulting in attribute Augmented (26). Due to their complexity these added circuits will most likely need to be placed away from the Golden resources causing attribute Distributed (27).

Chapter 6

Contributions

Configuration Bitstreams are enormous strings of binary data. To the human reader this information means nothing. To an FPGA, however, this data is everything. Every conceivable design, and every possible trojan is contained within the Bitstream. Yet, due to the shear volume of information within it and the lack of details on its format it has not previously been a common subject of study.

The purpose of this work is to provide an easy-to-use application that automates the processes of discovering and analyzing malicious modifications. With its success, Integrated Circuit manufacturers that use FPGAs will have an additional tool to ensure that their products operate as expected. Using the FPGA Trojan Detector takes only a few button clicks on the User-Interface. Its simple construction does not require any additional software or complicated install procedures and it can be used on any major operating system. Ensuring chips that have returned from Fabrication operate as expected takes no more than a few minutes. With the use of the FPGA Trojan Detector manufacturers will not need to train employees, buy expensive equipment or waste man-hours on additional testing.

The contributions of this work are:

1. Developed a method of extracting all of the information necessary to detect and analyze trojans directly from the configuration Bitstream.
2. Described methods for directly observing the presence of attributes in detected trojans.
3. Provided a method for inferring attributes that are not directly observable using a known taxonomy.
4. Developed a user-friendly application which automates the described methodology.

5. Succeeded in ensuring the application is cross-platform and does not require a complicated install procedure.

Bibliography

[1] Michael Wood. *"In search of the Trojan war*. 1st ed. The British Broadcasting Corporation, 1998. ISBN: 978-0-520-21599-3.

[2] H.-S. Philip Wong Subhasish Mitra and Simon Wong. *Stopping Hardware Trojans in Their Tracks*. IEEE Spectrum: Technology, Engineering, and Science News. url: http://spectrum.ieee.org/semiconductors/design/stopping-hardware-trojans-in-their-tracks. Jan. 2015.

[3] Celia Gorman. *Counterfeit Chips on the Rise*. IEEE Spectrum: Technology, Engineering, and Science News. url: http://spectrum.ieee.org/computing/hardware/counterfeit-chips-on-the-rise. May 2012.

[4] H. Li et al. "A Survey of Hardware Trojan Detection, Diagnosis and Prevention". In: *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*. 2015, pp. 173–180. DOI: `10.1109/CADGRAPHICS.2015.41`.

[5] N. Lesperance, S. Kulkarni, and Kwang-Ting Cheng. "Hardware Trojan detection using exhaustive testing of k-bit subspaces". In: *The 20th Asia and South Pacific Design Automation Conference*. 2015, pp. 755–760. DOI: `10.1109/ASPDAC.2015.7059101`.

[6] X. Mingfu et al. "Monte Carlo Based Test Pattern Generation for Hardware Trojan Detection". In: *Dependable, Autonomic and Secure Computing (DASC), 2013 IEEE 11th International Conference on*. 2013, pp. 131–136. DOI: `10.1109/DASC.2013.50`.

[7] P. Kitsos and A. G. Voyiatzis. "Towards a hardware Trojan detection methodology". In: *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*. 2014, pp. 18–23. DOI: `10.1109/MECO.2014.6862687`.

[8] A. Nejat, D. Hely, and V. Beroulle. "Facilitating side channel analysis by obfuscation for Hardware Trojan detection". In: *2015 10th International Design Test Symposium (IDT)*. 2015, pp. 129–134. DOI: 10.1109/IDT.2015.7396749.

[9] S. K. Rao et al. "Post-layout estimation of side-channel power supply signatures". In: *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. 2015, pp. 92–95. DOI: 10.1109/HST.2015.7140244.

[10] N. B. Gunti and K. Lingasubramanian. "Efficient static power based side channel analysis for Hardware Trojan detection using controllable sleep transistors". In: *SoutheastCon 2015*. 2015, pp. 1–6. DOI: 10.1109/SECON.2015.7132948.

[11] C. He et al. "A novel hardware Trojan detection method based on side-channel analysis and PCA algorithm". In: *Reliability, Maintainability and Safety (ICRMS), 2014 International Conference on*. 2014, pp. 1043–1046. DOI: 10.1109/ICRMS.2014.7107362.

[12] N. Jacob et al. "Hardware Trojans: current challenges and approaches". In: *IET Computers Digital Techniques* 8.6 (2014), pp. 264–273. ISSN: 1751-8601. DOI: 10.1049/iet-cdt.2014.0039.

[13] F. Wolff et al. "Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme". In: *2008 Design, Automation and Test in Europe*. 2008, pp. 1362–1365. DOI: 10.1109/DATE.2008.4484928.

[14] R. M. Rad et al. "Power supply signal calibration techniques for improving detection resolution to hardware Trojans". In: *2008 IEEE/ACM International Conference on Computer-Aided Design*. 2008, pp. 632–639. DOI: 10.1109/ICCAD.2008.4681643.

[15] R. Karri et al. "Trustworthy Hardware: Identifying and Classifying Hardware Trojans". In: *Computer* 43.10 (2010), pp. 39–46. ISSN: 0018-9162. DOI: 10.1109/MC.2010.299.

[16] Xiaoxiao Wang, M. Tehranipoor, and J. Plusquellic. "Detecting malicious inclusions in secure hardware: Challenges and solutions". In: *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. 2008, pp. 15–19. DOI: 10.1109/HST.2008.4559039.

[17] S. Moein et al. "An attribute based classification of hardware trojans". In: *Computer Engineering Systems (ICCES), 2015 Tenth International Conference on*. 2015, pp. 351–356. DOI: 10.1109/ICCES.2015.7393074.

[18] A. Al-Anwar et al. “Hardware Trojan detection methodology for FPGA”. In: *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*. 2013, pp. 177–182. DOI: 10.1109/PACRIM.2013.6625470.

[19] P. Kitsos and A. G. Voyiatzis. “FPGA Trojan Detection Using Length-Optimized Ring Oscillators”. In: *Digital System Design (DSD), 2014 17th Euromicro Conference on*. 2014, pp. 675–678. DOI: 10.1109/DSD.2014.74.

[20] M. Patterson et al. “A multi-faceted approach to FPGA-based Trojan circuit detection”. In: *VLSI Test Symposium (VTS), 2013 IEEE 31st*. 2013, pp. 1–4. DOI: 10.1109/VTS.2013.6548925.

[21] J.B. Note and E. Rannaud. “From the bitstream to the netlist”. In: *the International Symposium on Field Programmable Gate Arrays (FPGA)*. 2008.

[22] *Development System Reference Guide*. v10.1. Xilinx. 2008.

[23] *Virtex-5 FPGA Configuration User Guide*. v3.11. Xilinx. 2012.

[24] S. Adee. “The Hunt For The Kill Switch”. In: *IEEE Spectrum* 45.5 (2008), pp. 34–39. ISSN: 0018-9235. DOI: 10.1109/MSPEC.2008.4505310.

[25] M. Banga and M. S. Hsiao. “A region based approach for the identification of hardware Trojans”. In: *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. 2008, pp. 40–47. DOI: 10.1109/HST.2008.4559047.

[26] B. N. Hwang and M. H. Chen. “Key selection criteria for third party logistics in the IC manufacturing industry”. In: *Service Operations and Logistics, and Informatics (SOLI), 2013 IEEE International Conference on*. 2013, pp. 445–449. DOI: 10.1109/SOLI.2013.6611456.

[27] C. Beckhoff, D. Koch, and J. Torresen. “The Xilinx Design Language (XDL): Tutorial and use cases”. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*. 2011, pp. 1–8. DOI: 10.1109/ReCoSoC.2011.5981545.

[28] C. Lavin et al. “RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs”. In: *2011 21st International Conference on Field Programmable Logic and Applications*. 2011, pp. 349–355. DOI: 10.1109/FPL.2011.69.

[29] C. Lavin et al. *RapidSmith: A Library for Low-level Manipulation of Partially Placed-and-Routed FPGA Designs*. NSF Center for High Performance Reconfigurable Computing (CHREC) Department of Electrical and Computer Engineering Brigham Young University. Jan. 2014.

[30] F. Brglez, D. Bryan, and K. Kozminski. "Combinational profiles of sequential benchmark circuits". In: *Circuits and Systems, 1989., IEEE International Symposium on*. 1989, 1929–1934 vol.3. DOI: `10.1109/ISCAS.1989.100747`.

[31] *Collection of Digital Design Benchmarks*. http://ddd.fit.cvut.cz/prj/Benchmarks/. Accessed: 2016-10-24.

[32] H. Salmani, M. Tehranipoor, and R. Karri. "On design vulnerability analysis and trust benchmarks development". In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 2013, pp. 471–474. DOI: `10.1109/ICCD.2013.6657085`.

[33] *Trust-Hub.org*. http://trust-hub.org/. Accessed: 2016-10-24.