Possible Organization for Writing a Thesis including a LaTeXFramework and Examples

by

A Graduate Advisor
B.Sc., University of WhoKnowsWhere, 2053
M.Sc., University of AnotherOne, 2054

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

DOCTOR OF PHILOSOPHY

in the Department of Whichever

Possible Organization for Writing a Thesis including a LaTeXFramework and Examples

by

A Graduate Advisor
B.Sc., University of WhoKnowsWhere, 2053
M.Sc., University of AnotherOne, 2054

Supervisory Committee

Dr. R. Supervisor Main, Supervisor
(Department of Same As Candidate)

Dr. M. Member One, Departmental Member
(Department of Same As Candidate)

Dr. Member Two, Departmental Member
(Department of Same As Candidate)

Dr. Outside Member, Outside Member
(Department of Not Same As Candidate)

Supervisory Committee

Dr. R. Supervisor Main, Supervisor
(Department of Same As Candidate)

Dr. M. Member One, Departmental Member
(Department of Same As Candidate)

Dr. Member Two, Departmental Member
(Department of Same As Candidate)

Dr. Outside Member, Outside Member
(Department of Not Same As Candidate)

ABSTRACT

This document is a possible Latex framework for a thesis or dissertation at UVic. It should work in the Windows, Mac and Unix environments. The content is based on the experience of one supervisor and graduate advisor. It explains the organization that can help write a thesis, especially in a scientific environment where the research contains experimental results as well. There is no claim that this is the *best* or *only* way to structure such a document. Yet in the majority of cases it serves extremely well as a sound basis which can be customized according to the requirements of the members of the supervisory committee and the topic of research. Additionally some examples on using LaTeXare included as a bonus for beginners.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:
my cat, Star Trek, and the weather, for supporting me in the low moments.
Supervisor Main, for mentoring, support, encouragement, and patience.
Grant Organization Name, for funding me with a Scholarship.

I believe I know the only cure, which is to make one's centre of life inside of one's self, not selfishly or excludingly, but with a kind of unassailable serenity-to decorate one's inner house so richly that one is content there, glad to welcome any one who wants to come and stay, but happy all the same in the hours when one is inevitably alone.

Edith Wharton

DEDICATION

Just hoping this is useful!

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. [17] 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 as protecting ourselves against them as they are at attacking us. In recent years a new incarnation of 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. 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. [14] In 2011 over 1300 cases were reported to the Electronic Resellers Association International (ERAI) of modified ICs and the occurrence has been going

up. [6] 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 their ICs run our devices as expected is vital in the digital era. Since their discovery there has been concerted effort to detect hardware trojans but the innate complexity of ICs makes it difficult. [11]

1.2 Contributions

1.3 Organization

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 generally compromised in one of two ways. First, due to the complexity it is rare for a product to be managed 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. [7] These modifications are known as hardware Trojans. 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 [8, 13, 15, 16]. In [16], trojans were organized based solely on their activation mechanisms. A taxonomy based on the location, activation and action of a trojan was presented in [13], [8]. However, these approaches do not consider the manufacturing process. Another taxonomy was proposed in [15] which employs five categories: insertion, abstraction, activation, effect, and location. While this is more extensive than

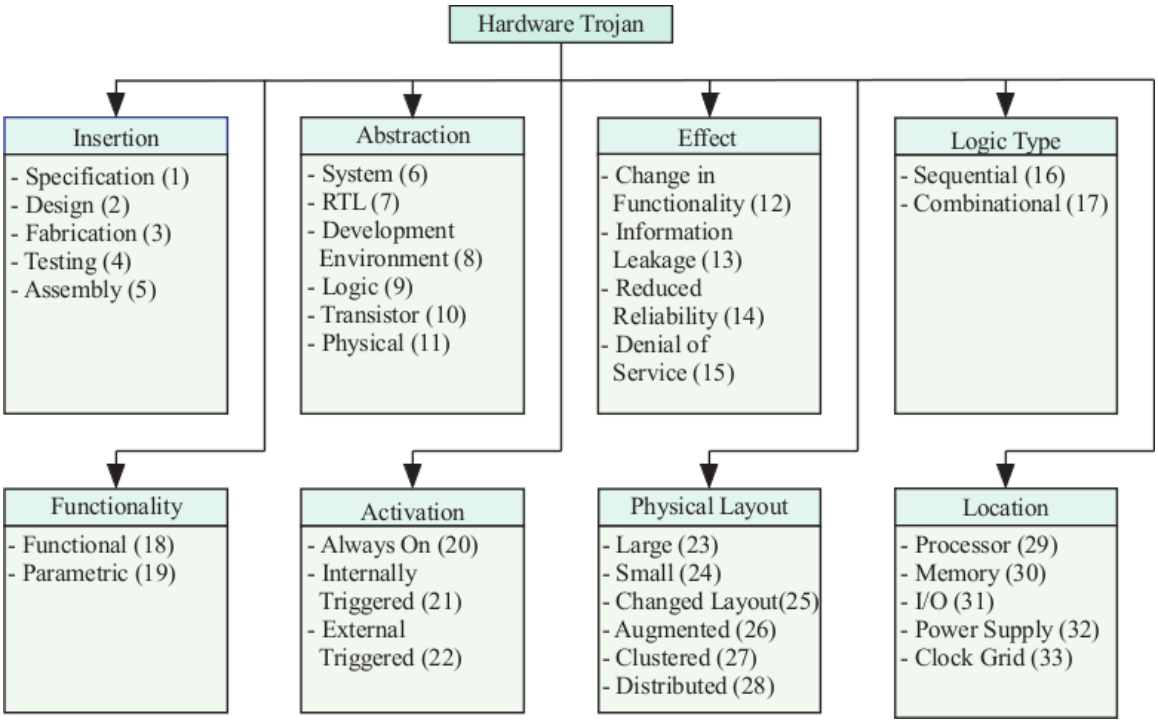


Figure 2.1: The thirty-three attributes of the hardware trojan taxonomy in [12].

previous approaches, it fails to account for the physical characteristics of a trojan. An additional taxonomy was proposed in [12] 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

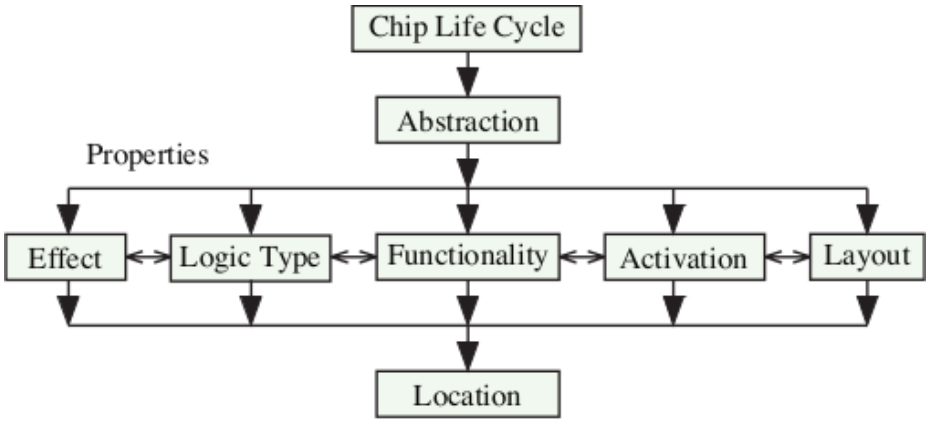


Figure 2.2: The hardware trojan levels [12].

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 $\mathbf{R}$ [12]. 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

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

$$\mathbf{R} =$$

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
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																						
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					
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
29																																	
30																																	
31																																	
32																																	
33																																	

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

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

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. 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 PLDs in the device from the user is referred to as the configuration Bitstream. FPGA users create designs using a programming language; the design is then downloaded or "configured" onto the device via the Bitstream. 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 provides a visual representation of the use-case assumed for the purposes

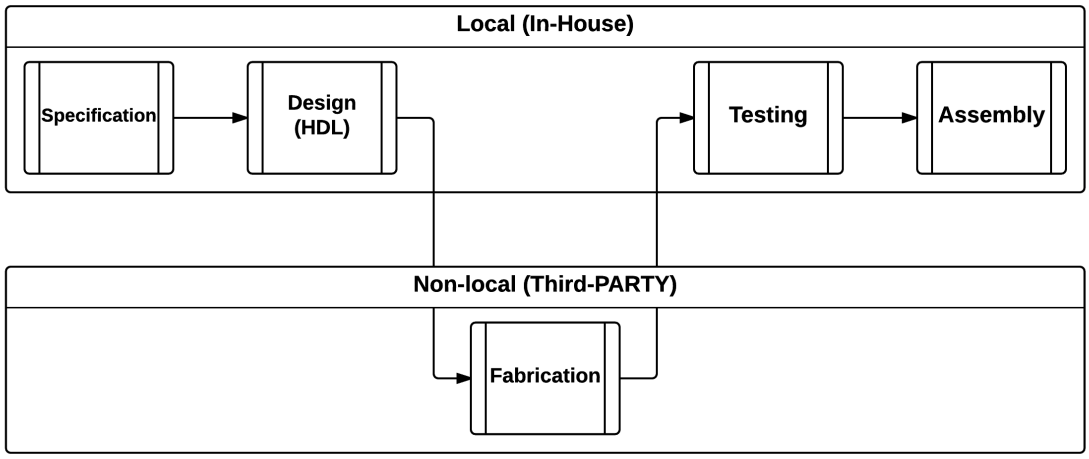


Figure 3.1: FPGA Life-Cycle

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 are to be kept in secure storage while a copy is sent to be fabricated; these 'clean' copies are referred to as Golden. Though it is known that the fabrication houses will often attempt to make optimizations on designs, this methodology requires that no such efforts are made. When the completed batch of fabricated chips are returned the Bitstream is extracted via the

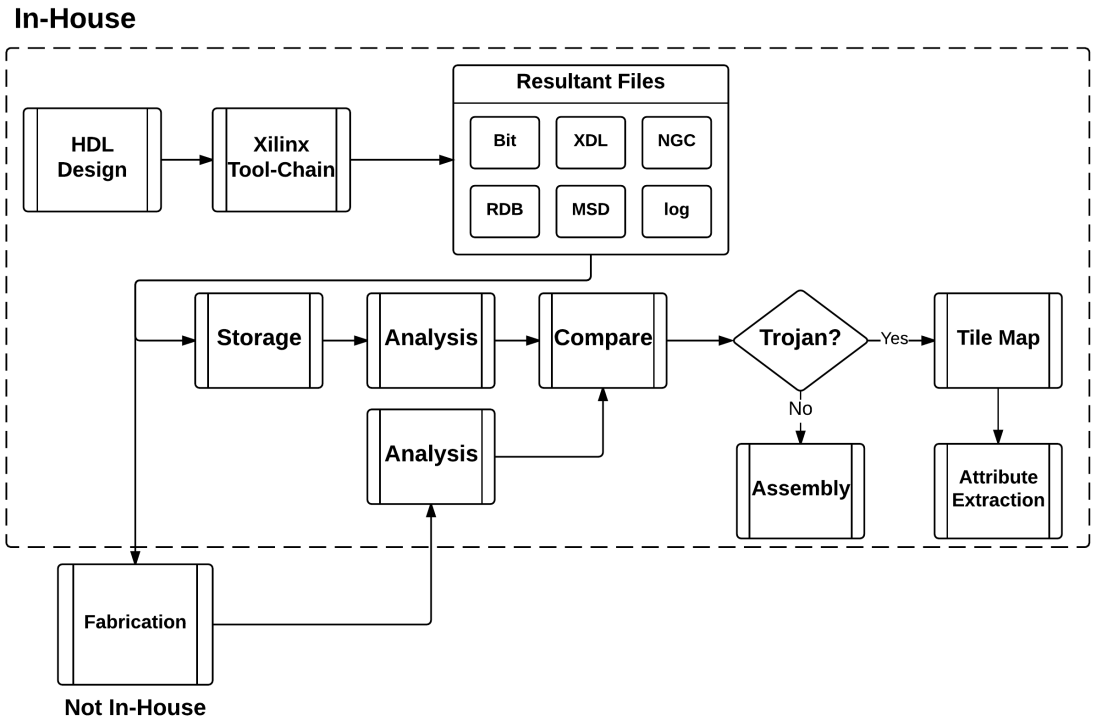


Figure 3.2: Methodology Overview

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, descriptive attributes presented in section 2.2 are returned to the user, 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. 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 tiles. A

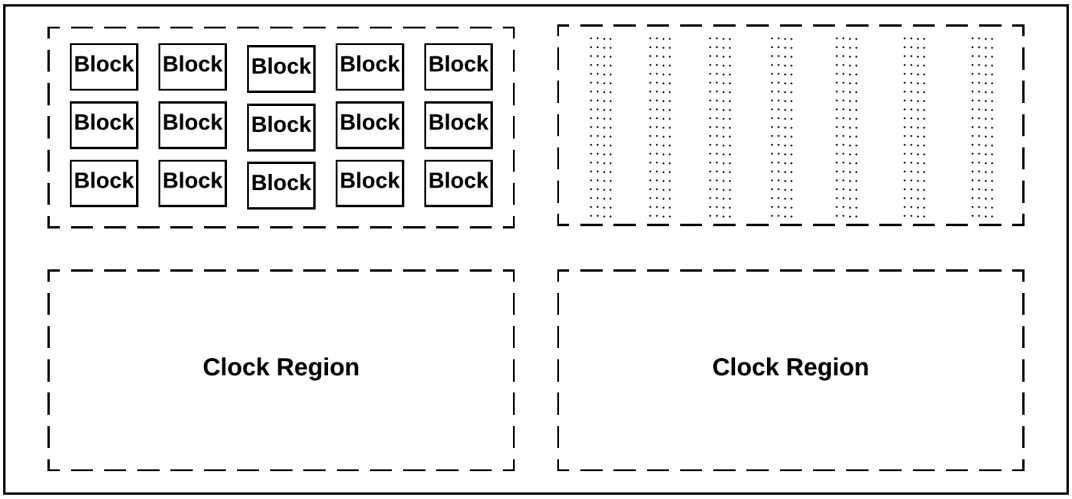


Figure 3.3: Rudimentary Layout of a Virtex Gate-Array

block will consist of one or multiple tiles depending on the 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 comprised entirely by a single block type. Columns are separated by clock regions shown by the dashed lines in Figure 3.3. Each region is an independent array of blocks that uses a dedicated clock resource; this minimizes clock skew from causing undesired timing delays. Figure 3.4

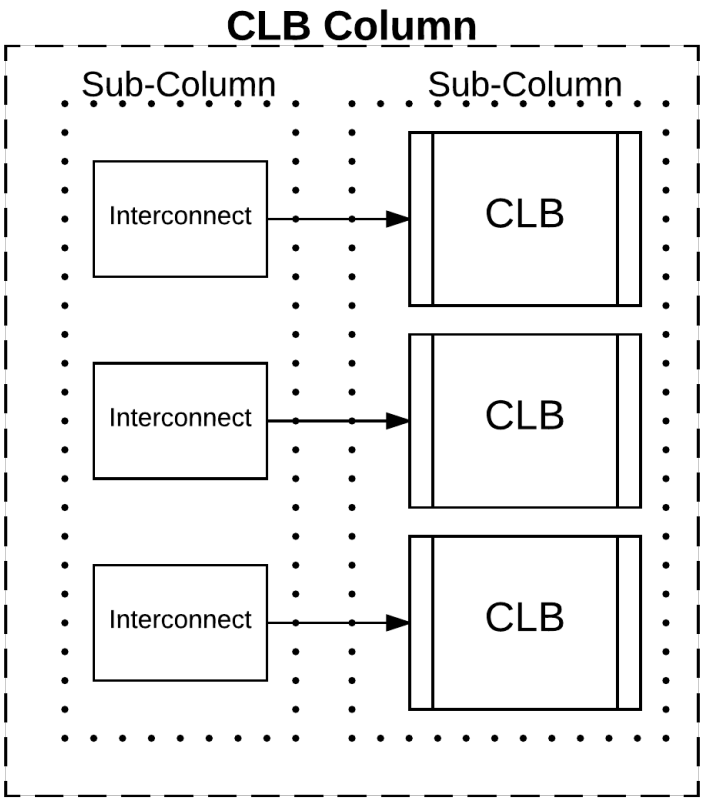


Figure 3.4: Column Composition

depicts a Configurable Logic Block (CLB) column. A column is comprised of blocks stacked vertically; CLB blocks contain a CLB tile and an interconnect tile. 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 the 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.

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

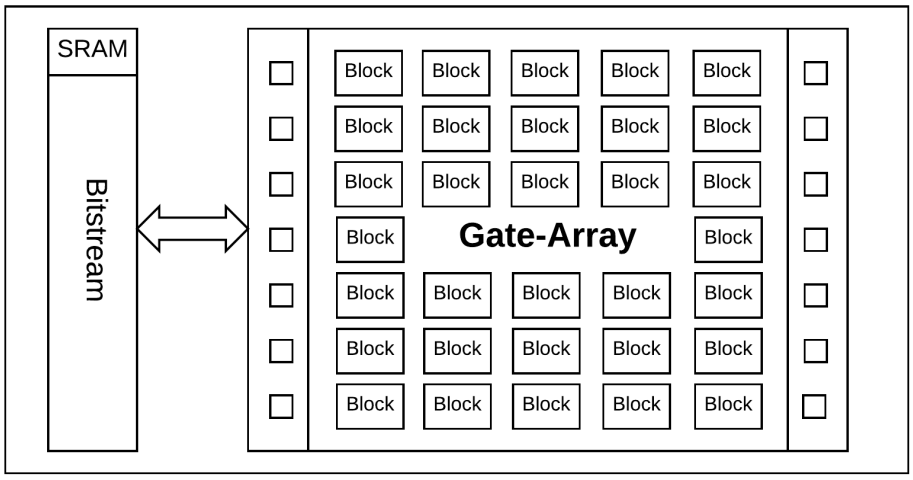


Figure 3.5: FPGA Device Layout

loaded from the external memory (often Static RAM (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. 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 [19]. 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 RAM (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 device [19]. 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 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

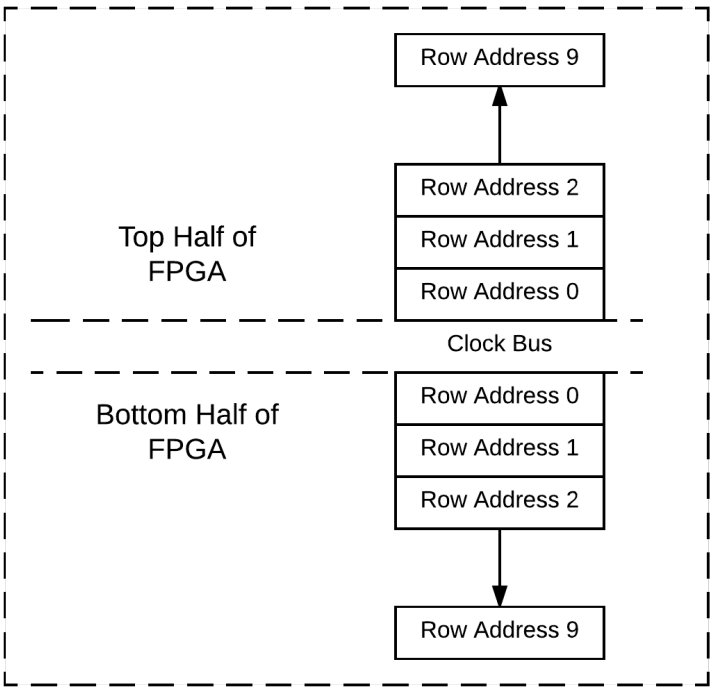


Figure 3.6: Row Order of Virtex-5 Clock Region

Table 3.2: Number of Frames (minor addresses) per Column [19]

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

26 and 27 access the Interface for that column. All other frames are specific to that block. [19] 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 Automated Hardware Trojan Systememploys 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 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 can be thought of as the Y value for the clock-region in which the frame resides. 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. Once the correct clock region is discerned the major address is used to determine the block column. The major address begins at 0 on the left and counts up towards the number of columns in the clock region. 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 device. As can be seen in Figure 3.7 the central word in a frame configures the horizontally running clock bus. The remaining words are 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:

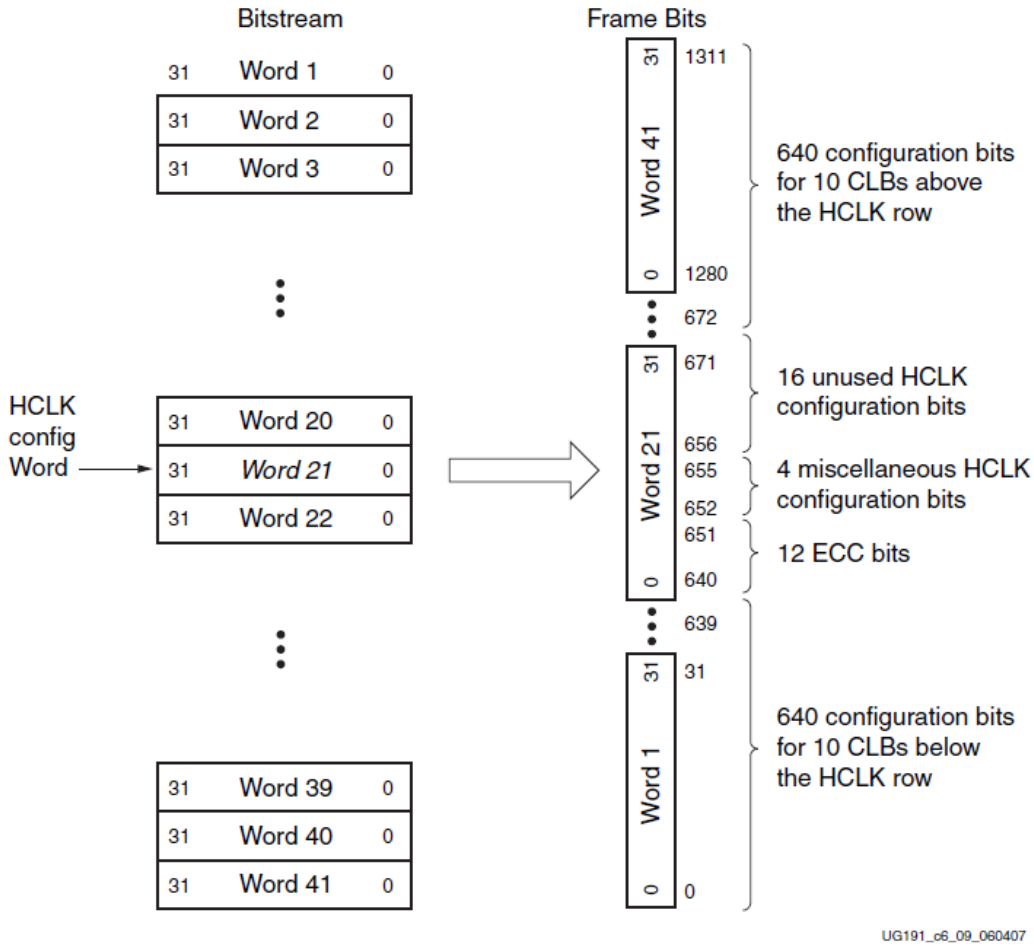


Figure 3.7: Configuration Words in the Bitstream [19]

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 [12] 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 extracting the attributes that are directly observable then using them to infer the existence of the remaining attributes.

3.6.1 Extraction Methods

Observed Location Attributes

The presence of attributes in the *Location* category are directly observable from the results of the tile 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 row height 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 tile 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 [12], 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 [2, 3]. 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 **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 Automated Hardware Trojan System was built to be a stand-alone application. In order to allow it to be a powerful yet simple tool the technologies used were selected with the idea of portability in mind.

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' [18]. 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 [4]. 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 is used by Automated Hardware Trojan System to relate any 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 [4]

```
#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 we can see 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'. Automated Hardware Trojan System was written in Java primarily in order to interface with the API known as *RapidSmith* which is described in section 4.2.3. However, the added benefits of allowing Automated Hardware Trojan System to be a compact, cross-platform application can not be understated. 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. Automated Hardware Trojan System employs Java and Java *Swing* to make it as user-friendly as possible.

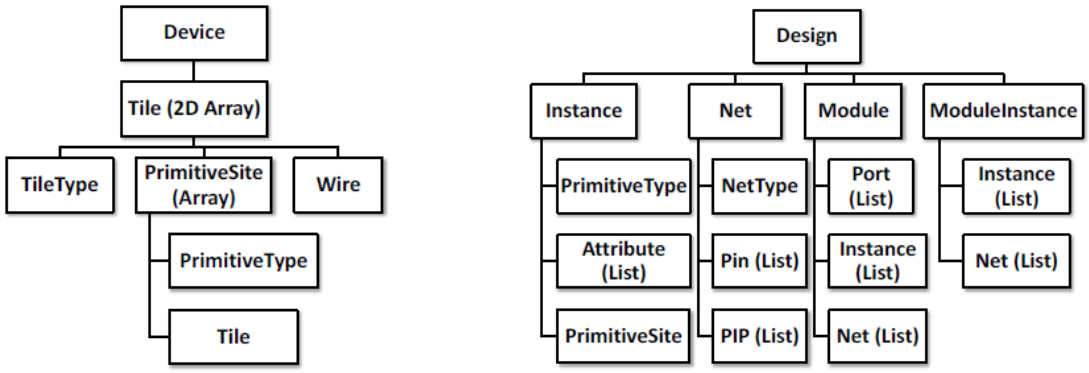
4.2.3 *RapidSmith*

RapidSmith is a set of tools and Application Programming Interfaces (API) written in Java that enable Computer Aided Design (CAD) tool creation for *Xilinx* FPGAs [10]. 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 support-

ing library for Automated Hardware Trojan System 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. Automated Hardware Trojan System 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 unusable level. The makers of *RapidSmith* have developed a compression scheme which converts the XDLRC files into a non human-readable format. The compression scheme reduces the size of the XDLRC files from 7GB to 1.3MB. The API provides interface classes that support querying the compressed files. With these compressed fies, *RapidSmith* enables 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 Automated Hardware Trojan System 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 cofngiured to the correct class 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 files is read and the 'Design' class is similarly loaded with the user-defined design. Not shown in Figure 4.1 are a series of utility classes. The Bitstream parser class



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

Figure 4.1: The *RapidSmith* Class Hierarchy [9]

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 Automated Hardware Trojan System

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 Automated Hardware Trojan System employs *Swing* to provide a very simplistic and easy to use interface which 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, described in section 4.3.2. Results are printed textually in the text window. If a trojan is discovered Automated Hardware Trojan System reports the list of attributes which describe it. 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

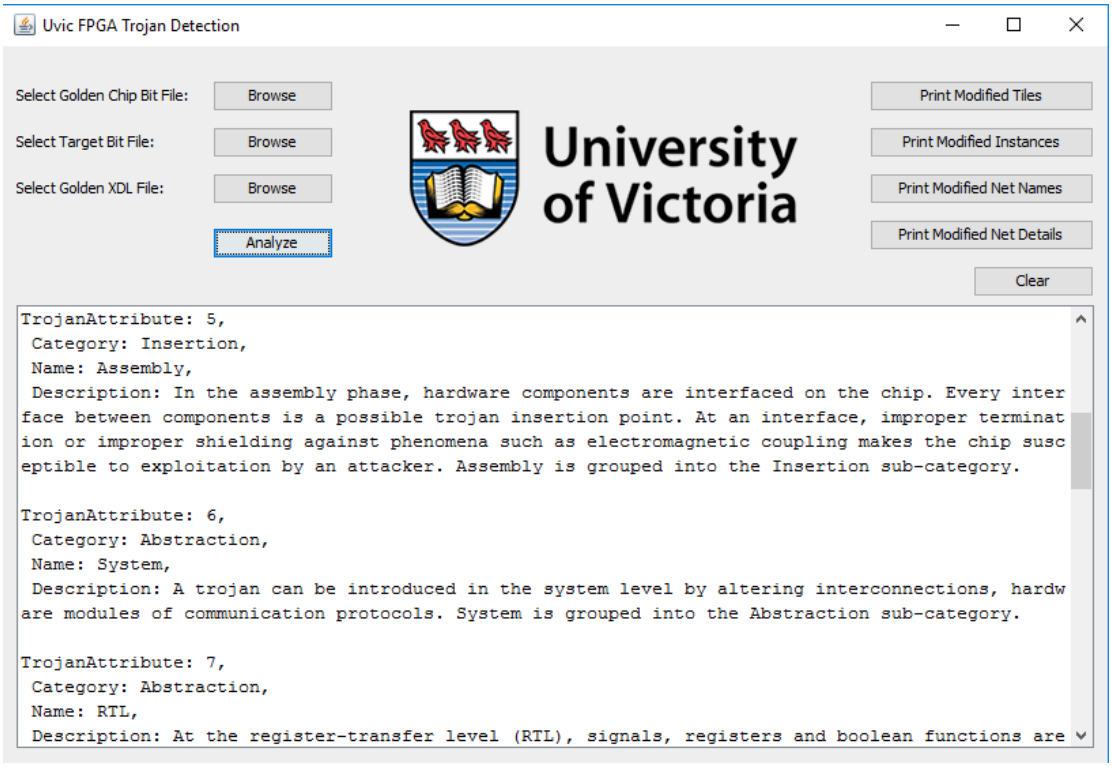


Figure 4.2: The User-Interface of the Automated Hardware Trojan System

place. The 'Print Modified Instances' feature corresponds each modified tile to the instance it belongs to. *Xilinx* provides schematic design views which correspond instance names to the user's design. This allows 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 which has been modified. Again this allows the user relate modification 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 Automated Hardware Trojan System. 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. The *RapidSmith* API provides a Bitstream parsing utility. It 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 different pattern result in different features of the Bitstream files including frame size and ordering ordering. 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 'Tile Mapping' method described in section 3.5 to determine where in the architecture the modifications have been made.

The tile-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. Tile 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

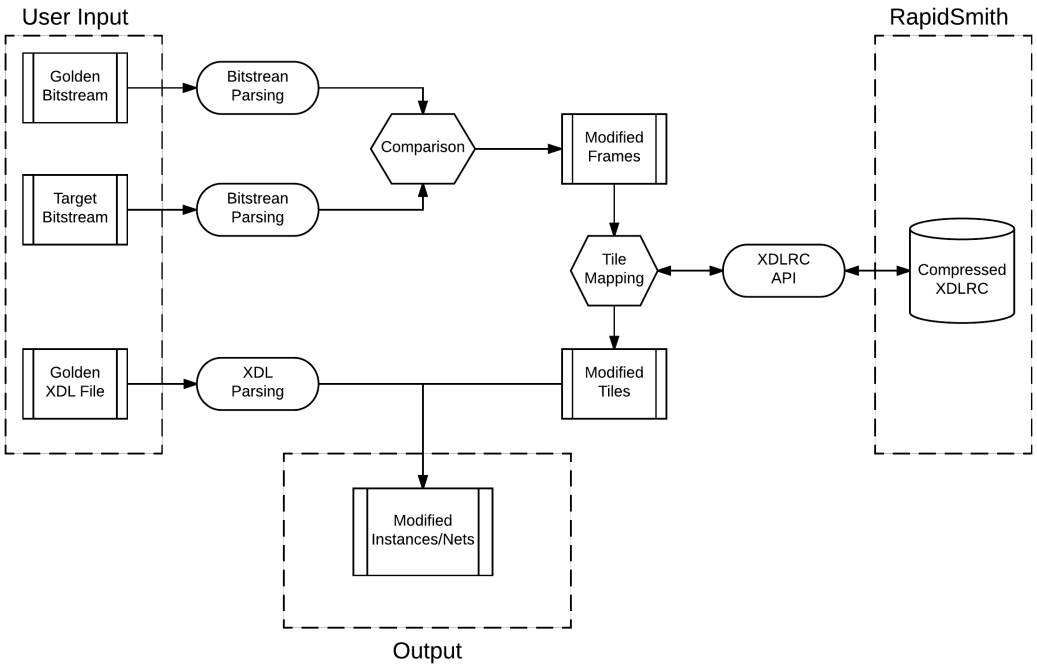


Figure 4.3: Overview of Functional Operation

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 Automated Hardware Trojan System 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.

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 [5]. These benchmarks were accessed and a small 'Priority Decoder' circuit was chosen [1]. 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 Automated Hardware Trojan System. 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 Automated Hardware Trojan System returns a result indicating that there is no trojan present. As can be seen in Figure 5.1,

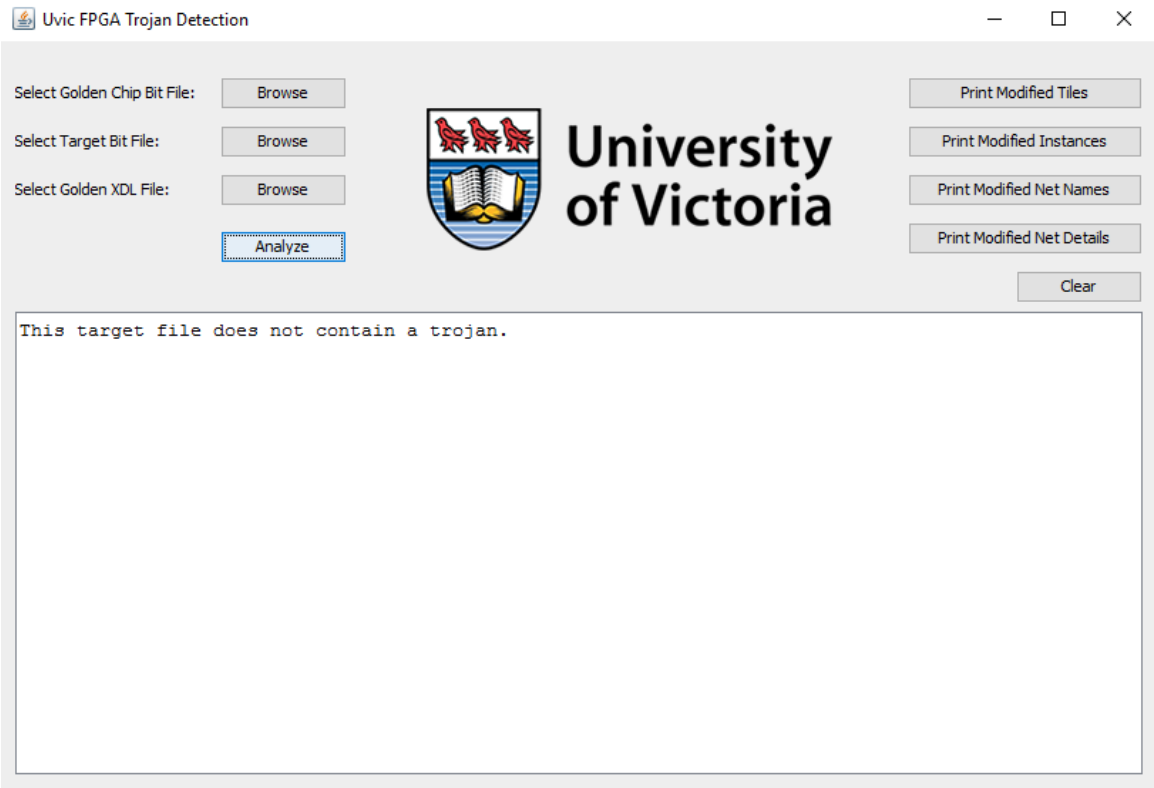


Figure 5.1: Priority Decoder Analysis Results

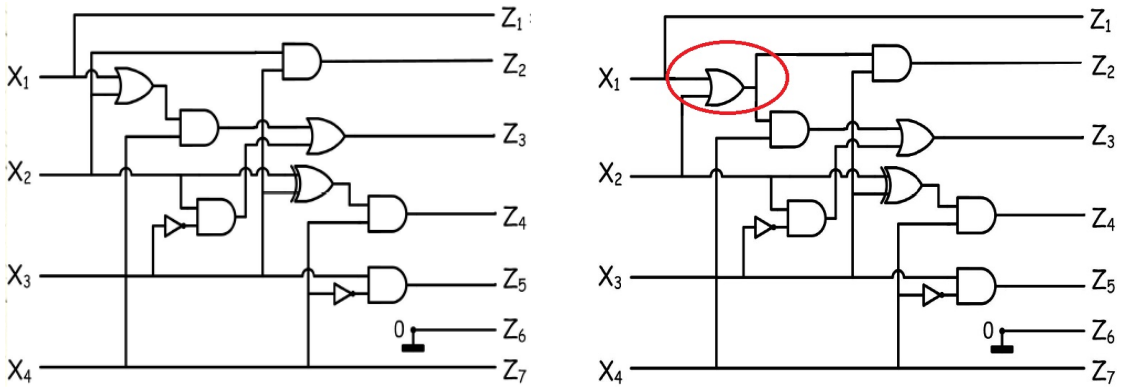
the Automated Hardware Trojan System successfully analyzed the Bitstreams and determined that there was no trojan present, as expected.

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. These unused inputs are referred to as 'dont-cares', meaning that it is not important to the func-

Table 5.1: Outputs of the Circuits in Figs. 5.2a and 5.2b [12]

		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

tion 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 input to the Automated Hardware Trojan System. 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

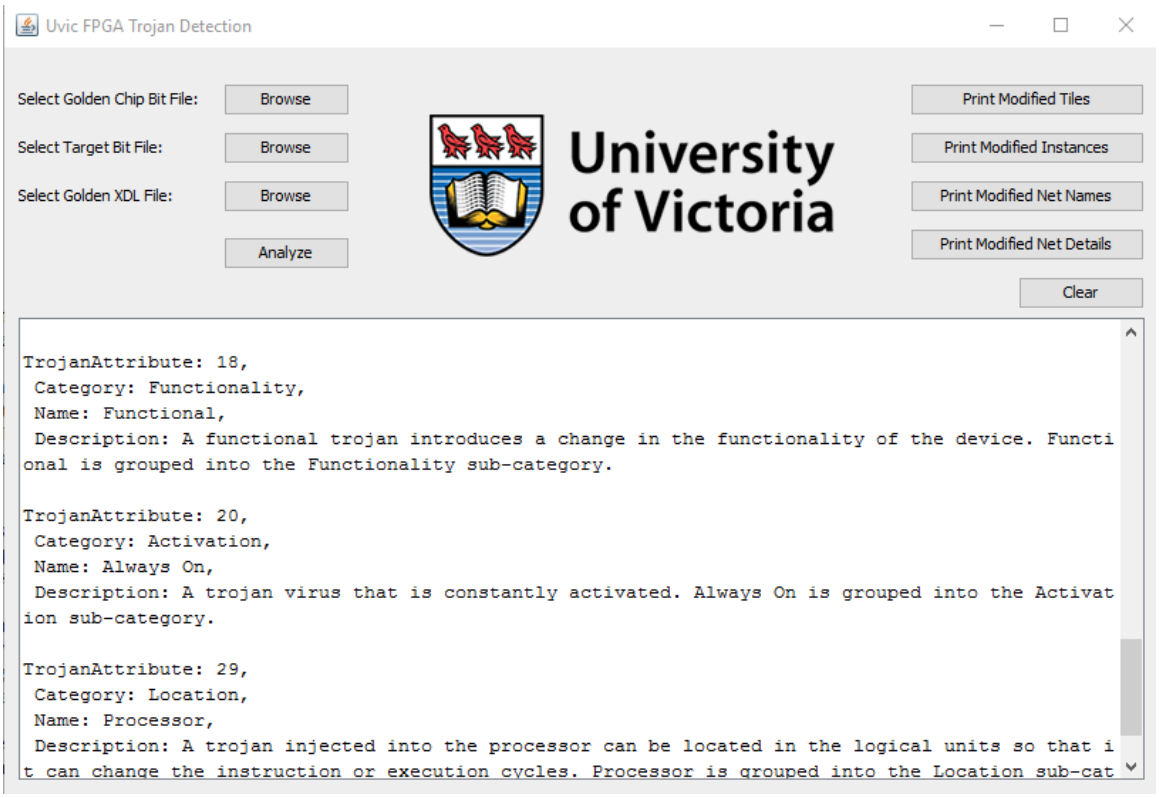


Figure 5.3: Results of The Authentication Circuit Trojan Detection

- Attribute 18: Functional
- Attribute 20: Always On
- Attribute 29: Processor

As expected the results state that the trojan was inserted in the Fabrication phase; it is the earliest stage in the 'insertion' phase produced indicating it as the source. The effects of this modification propagate to the testing and assembly phases as expected. The modifications reach both the System and Registry Transfer Level (RTL) 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 properties attributes listed state that the trojan changes the functionality of the circuit. 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 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, not Parametric; 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'. The modified values for input $x = 10$ or $x = 11$ are always available and not activated. This is consistent with the returned results. Finally, all of the tiles modified by the trojan began 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 29, Processor, 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

5.2 Discussion

Chapter 6

Conclusions

My first rule for this chapter is to avoid finishing it with a section talking about future work. It may seem logical, yet it also appears to give a list of all items which remain undone! It is not the best way psychologically.

This chapter should contain a mirror of the introduction, where a summary of the *extraordinary* new results and their wonderful attributes should be stated first, followed by an executive summary of how this new solution was arrived at. Consider the practical fact that this chapter will be read quickly at the beginning of a review (thus it needs to provide a strong impact) and then again in depth at the very end, perhaps a few days after the details of the previous 3 chapters have been somehow forgotten. Reinforcement of the positive is the key strategy here, without of course blowing hot air.

One other consideration is that some people like to join the chapter containing the analysis with the only with conclusions. This can indeed work very well in certain topics.

Finally, the conclusions do not appear only in this chapter. This sample mini thesis lacks a feature which I regard as absolutely necessary, namely a short paragraph at the end of each chapter giving a brief summary of what was presented together with a one sentence preview as to what might expect the connection to be with the next chapter(s). You are writing a story, the *story of your wonderful research work*. A story needs a line connecting all its parts and you are responsible for these linkages.

Appendix A

Additional Information

This is a good place to put tables, lots of results, perhaps all the data compiled in the experiments. By avoiding putting all the results inside the chapters themselves, the whole thing may become much more readable and the various tables can be linked to appropriately.

The main purpose of an Appendix however should be to take care of the future readers and researchers. This implies listing all the housekeeping facts needed to continue the research. For example: where is the raw data stored? where is the software used? which version of which operating system or library or experimental equipment was used and where can it be accessed again?

Ask yourself: if you were given this thesis to read with the goal that you will be expanding the research presented here, what would you like to have as housekeeping information and what do you need? Be kind to the future graduate students and to your supervisor who will be the one stuck in the middle trying to find where all the stuff was left!

Bibliography

[1] Collection of digital design benchmarks. http://ddd.fit.cvut.cz/prj/Benchmarks/. Accessed: 2016-10-24.

[2] S. Adee. The hunt for the kill switch. *IEEE Spectrum*, 45(5):34–39, May 2008.

[3] 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*, pages 40–47, June 2008.

[4] 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*, pages 1–8, June 2011.

[5] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Circuits and Systems, 1989., IEEE International Symposium on*, pages 1929–1934 vol.3, May 1989.

[6] Celia Gorman. Counterfeit chips on the rise. IEEE Spectrum: Technology, Engineering, and Science News, 5 2012. url: http://spectrum.ieee.org/computing/hardware/counterfeit-chips-on-the-rise.

[7] N. Jacob, D. Merli, J. Heyszl, and G. Sigl. Hardware trojans: current challenges and approaches. *IET Computers Digital Techniques*, 8(6):264–273, 2014.

[8] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor. Trustworthy hardware: Identifying and classifying hardware trojans. *Computer*, 43(10):39–46, Oct 2010.

[9] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and M. Wirthlin B. Hutchings. 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, 1 2014.

[10] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings. Rapidsmith: Do-it-yourself cad tools for xilinx fpgas. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 349–355, Sept 2011.

[11] H. Li, Q. Liu, J. Zhang, and Y. Lyu. A survey of hardware trojan detection, diagnosis and prevention. In *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, pages 173–180, Aug 2015.

[12] S. Moein, S. Khan, T. A. Gulliver, F. Gebali, and M. W. El-Kharashi. An attribute based classification of hardware trojans. In *Computer Engineering Systems (ICCES), 2015 Tenth International Conference on*, pages 351–356, Dec 2015.

[13] R. M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic. Power supply signal calibration techniques for improving detection resolution to hardware trojans. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 632–639, Nov 2008.

[14] H.-S. Philip Wong Subhasish Mitra and Simon Wong. Stopping hardware trojans in their tracks. IEEE Spectrum: Technology, Engineering, and Science News, 1 2015. url: http://spectrum.ieee.org/semiconductors/design/stopping-hardware-trojans-in-their-tracks.

[15] 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*, pages 15–19, June 2008.

[16] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty. Towards trojan-free trusted ics: Problem analysis and detection scheme. In *2008 Design, Automation and Test in Europe*, pages 1362–1365, March 2008.

[17] Michael Wood. *"In search of the Trojan war.* The British Broadcasting Corporation, 1 edition, 1998.

[18] Xilinx. *Development System Reference Guide*, v10.1 edition, Jan 2008.

[19] Xilinx. *Virtex-5 FPGA Configuration User Guide*, v3.11 edition, Oct 2012.