Automated Hardware Trojan Detection in FPGAs

Nicholas Houghton, *Member, IEEE,* Samer Moein, *Member, IEEE,* and Fayez Gebali, *Life Senior Member, IEEE*

***Abstract*—Embedded Systems have become ubiquitous in modern life; we are just as affected by their vulnerabilities as they are. Ensuring that the processors that control them are secure is paramount to the safety of commercial, transportation and military infrastructure. 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 for *Xilinx* FPGAs.**

***Keywords*—Hardware trojans, chip life cycle, hardware trojan taxonomy, hardware trojan attributes. ... Put your list**

I. Introduction

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. Integrated Circuit (IC) designs for Field Programmable Gate-Arrays (FPGA) are made using Hardware Description Language (HDL). 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.

In 2013 researchers at Cairo University proposed a method of insulating externally sourced Intelectual Property (IP) with Cyclic Redundancy Check (CRC) defense modules [?]. According to the authors their method is capable of detecting leaked information with a 99.95% accuracy. It was designed specifically for detecting trojans that leak information; trojans which exhibit other behaviors can not be detected. In addition the authors report considerable detriment to power consumption and performance.

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. 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.

Researchers from Iowa State University [?] proposed a multi-faceted approach to trojan detection in FPGAs. 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* [?] 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 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.

A new method of extracting and analyzing the configuration Bitstream to determine the presence of hardware trojans has been developed and is presented in this work. This method is able to meaningfully read the long binary file which configures an FPGA and extract modifications. Any discovered changes are located on the device using a technique referred to as 'Component Mapping'. Further, these changes are then mapped to the user's original design. Knowing which components of the device have been modified, and the instances of the synthesized design allows for a powerful description to be built. A software tool known as the FPGA Trojan Detector,

N. Houghton, S. Moein, and F. Gebali are with the Department of Electrical and Computer Engineering, University of Victoria, Victoria, Canada.

Manuscript received April 19, 2005; revised January 11, 2007.

which implements this new method has been built using Java and an open-source application programming interface (API) known as RapidSmith. It is able to automatically detect and analyze trojans in *Xilinx* FPGAs. Once analyzed, a meaningful description is provided using the trojan taxonomy presented in [3].

The contributions of this paper are:

1) A new method mapping configuration Bitstream words to device components named 'Component Mapping'.
2) A systematic process of detecting and analyzing hardware trojans in FPGAs
3) A software tool which automates these new methods.

The remainder of this paper is organized as follows. Section II provides some useful background on the trojan taxonomy used by the new method, as well as the architecture of FPGAs. Section III describes the overall process of the new trojan detection technique. Section IV presents how the analysis results of Section III are used to generate a comprehensive description of a discovered trojan. Section V presents the software tool which demonstrates the efficacy of the new methodology. Section VI presents three case studies using benchmark designs containing trojans. Finally, Section VII provides some concluding remarks.

II. Background

A. Hardware Trojan Taxonomy

The evaluation of hardware trojans requires a comprehensive means of their description. Several hardware trojan taxonomies have been proposed [**?**], [**?**], [**?**], [**?**]. An additional taxonomy was proposed in [**?**] 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 four levels of description as shown in Fig. 1. There are eight categories comprised of a total of thirty-three attributes as shown in Fig. 2.

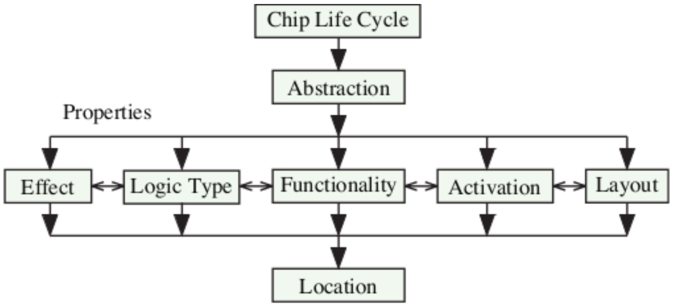


Fig. 1: Hardware trojan life-cycle levels [**?**].

1) The **insertion** (chip life-cycle) level comprises the attributes pertaining to the IC production stages.
2) The **abstraction** level corresponds to where in the IC abstraction the trojan is introduced.
3) The **properties** level comprises the behavior and physical characteristics of the trojan. It contains the taxonomy categories *effect*, *logic type*, *functionality*, *activation* and *layout*.
4) The **location** level corresponds to the location of the trojan in the IC.

The properties level has the following categories.

- The **effect** category describes the disruption or effect a trojan has on the system.
- The **logic type** category describes the circuit logic that triggers the trojan, either combinational logic or sequential.
- The **functionality** category differentiates between trojans which are functional or parametric.
- The **activation** category differentiates between trojans which are always on or are triggered.
- The **layout** category is based on the physical characteristics of the trojan.

The relationships between the trojan attributes shown in Fig. 2 can be described using a matrix $\mathbf{R}$ [**?**]. 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

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

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

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

B. Field Programmable Gate-Array Organization

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. FPGA users create designs using a programming language referred to as a Hardware Description Language (HDL). The design is then compiled and synthesized into the Bitstream which is then downloaded or "configured" onto the device.

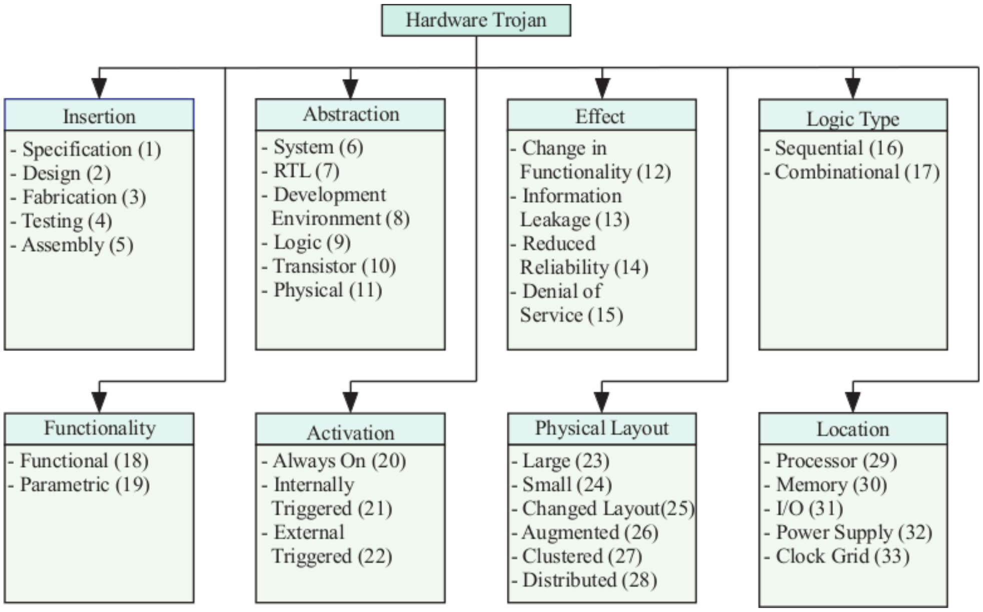


Fig. 2: The hardware trojan attribute taxonomy [**?**].

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 and so on. A *Xilinx* device is organized into a matrix of blocks referred to as the ’gate-array’ as shown in Fig. 3 [**?**]. A block is not a physical device but a conceptual grouping of tiles. A block will consist of one or multiple tiles depending on its type. A device can contain anywhere from a couple hundred to a few thousand blocks and are arranged into columns by type. Columns are separated into regions shown by the dashed lines in Fig. 3. These regions each use a separate clock mechanism and are referred to as ’Clock Regions’. The set of messages sent to all of the tiles in the device from the user is referred to as the configuration Bitstream.

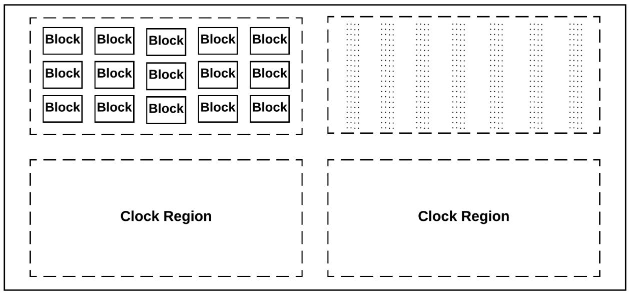


Fig. 3: Rudimentary Layout of a Virtex FPGA

III. Proposed Trojan Detection

A. Overview

Fig. 4 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. Fig. 5 shows an overview of the trojan detection methodology. As mentioned, FPGA designs are written in 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 Fig. 5. This includes but is not limited to:

- Bit: the configuration Bitstream
- XDL: *Xilinx* Design Language
- NGC: a non-human readable semantic description of the design known as a netlist
- RDB: the readback data text file
- MSD: a mask file used to hide unwanted data during the a readback procedure
- log: a log file which collects compiler output.

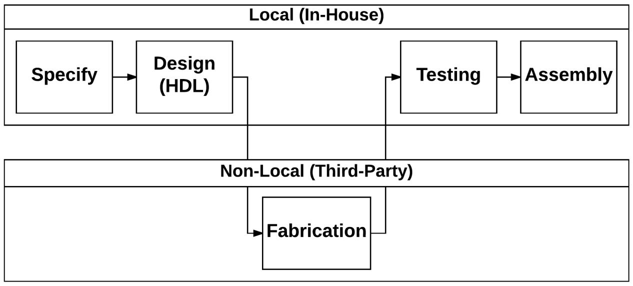


Fig. 4: FPGA Life-Cycle

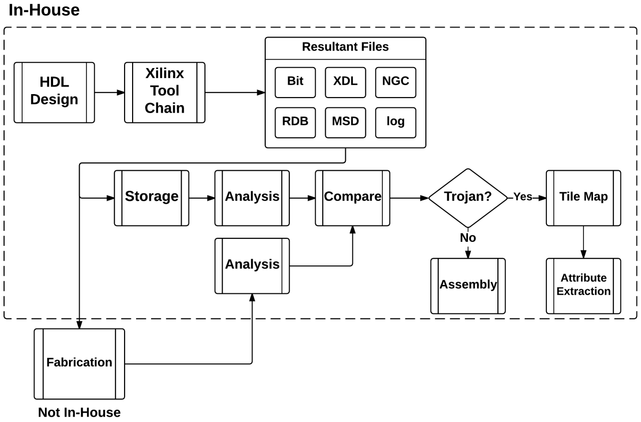


Fig. 5: Methodology Overview

The majority of these files are used for specific purposes that do not pertain to the new hardware trojan detection method. The netlist (.ngc) file, however, contains a thorough description of the user's design and how it relates to the device. It can be converted into a human-readable version known as a *Xilinx* Design Language (XDL) file which will be used to relate any discovered modifications to the user's design. The Bit file is the binary representation of the design implementation. It is referred to as the Bitstream or 'configuration' Bitstream and is the final form that is loaded into the FPGA. 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 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 using the *Xilinx* feature Readback. That which is extracted is referred to as the Target Bitstream. The Golden and Target Bitstreams are analyzed in conjunction to detect differences. Any discovered differences are then attributed to the corresponding component in the architecture, described in section III-C. Modified components are then related to the user's design using the XDL file. Finally, the resultant taxonomic description is built and returned to the user.

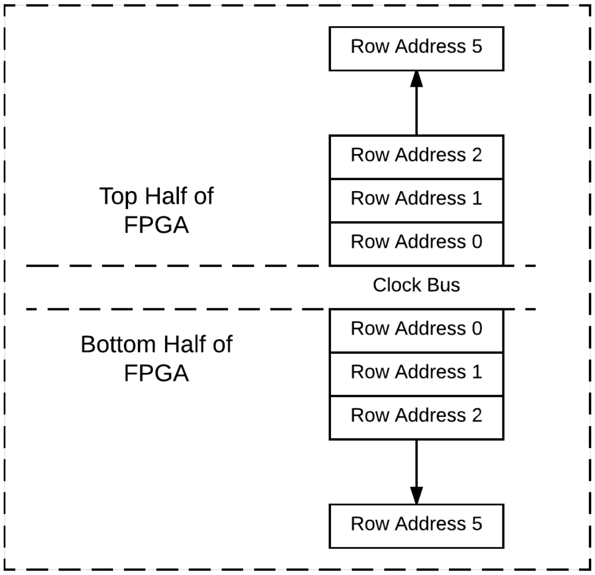


Fig. 6: Row Order of Virtex-5 Clock Region

B. Bitstream Analysis

An FPGA is composed of a matrix of blocks referred to as the gate-array. Blocks perform functions specified by the configuration they receive from the synthesis process. 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. 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 shown in Table I and is the smallest addressable element.

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

The Block Address (BA) discerns whether the block type is a logic block, a memory block or memory interconnect. The Logic block (BA: 0) contains the columns which provide the primary configuration for the device (CLBs, IOBs... etc). The BRAM columns (BA: 1) initialize the memory for the device while the BRAM Interconnect columns (BA: 2) configure how the logic of the design interacts with the BRAM.

In the case of the Virtex-5 family each clock region is composed of twenty blocks in a column separated by a horizontal clock bus as shown in Fig. 6. 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

TABLE I: Bitstream Frame Address Structure

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

a Top indicator bit in position 20 that indicates whether the specified row is above or below the center of the device [?]. 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 II provides the number of frames per column type. In the *Xilinx* jargon, a tile is a conceptual encapsulation of components in the gate-array which perform a specific task. A block may contain multiple tiles. A Logic (BA: 0) column of blocks may be one of many types. As an example, in a Configurable Logic Block (CLB) column, each block consists of an interconnect tile (also known as a Switching Matrix (SM)) on the left, and a CLB tile on the right, as shown in Fig 7.

TABLE II: Number of Frames (minor addresses) per Column [?]

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

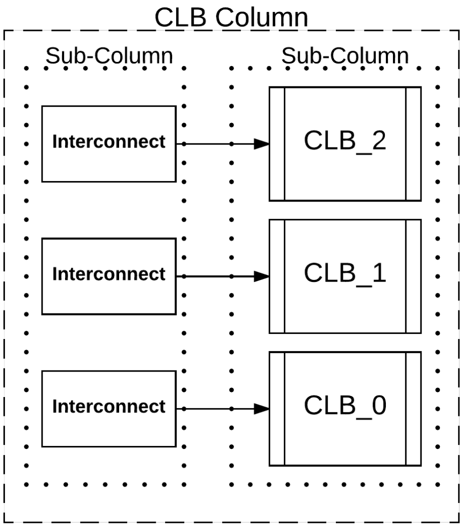


Fig. 7: Column Composition

The Frames which configure a column are numbered from left to right, starting with 0. For the Configurable Logic Block (CLB) column, frames with a minor address of 0 to 25 configure the interconnect (SM) tile. Frames numbered 26 and 27 access the Interface for that column. The remainder configure the CLB tile [?]. To further understand how frames configure tiles a mapping must be made between each frame and the corresponding tile. When analyzing the Golden and Target Bitstreams, by understanding how frames configure the blocks of the gate-array, the address of any modified frames can point to exactly which tile has been affected. A process know as 'Component Mapping' has been developed.

C. Component Mapping

The format of the configuration Bitstream describes the link 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. The FPGA Trojan Detector employs a method referred to as Component Mapping to create a discern which how any discovered modifications affect the device.

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. As described in section III-B, the Block Address (BA) value discerns whether the column type is a logic block column (BA: 0), a Block Random Access Memory (BRAM) column (BA: 1), or a memory interconnect block (BA: 2). 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, as was seen in Fig. 6. 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 Fig. 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. In the *Xilinx* jargon, a tile is a conceptual encapsulation of components in the gate-array which perform a specific task. A block may contain multiple tiles. As an example, in a Configurable Logic Block (CLB) column, each block consists of an interconnect tile (also known as a Switching Matrix (SM)), an interface tile, and a CLB tile. The Frames which configure a column have a minor address numbered from left to right, starting with 0. Frames with a minor address from 0 to 25 configure the interconnect (SM). Frames numbered 26 and 27 access the interface tile and the remainder configure the CLB tile [?].

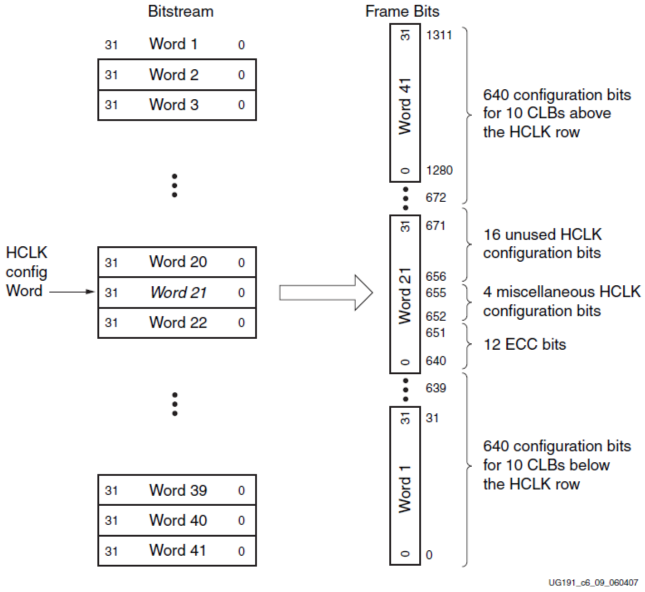


Fig. 8: Configuration Words in the Bitstream [?]

Figure 8 shows how a frame is composed of 41 words that can be thought of as a vertical stack that aligns within a column. Similarly, a component column consists of a stack of blocks; there are 20 blocks per CLB column, 40 per IOB column, 4 per BRAM and so on. The configuration words within a frame align with the physical blocks of the column. To compute which word corresponds to which component a series of equations has been developed. First, Equation 1 is used to compute the number of 32-bit words that span each block.

$$n = (W - C) + B \tag{1}$$

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

Referring to Fig 7, physical components are numbered from the bottom up while, as seen in Fig 8, configuration words are numbered from the top of the column down. This ordering pattern results in Equation 2. The result of Equation 1 and the modified word's position in the frame are used to determine which block the modification affected.

$$i = B - \left\lfloor \frac{w}{n} \right\rfloor \tag{2}$$

where:

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

By using equations 1 and 2 and knowing how the minor address relates to the column it is possible to link any modifications in the Bitstream to its corresponding tile in the device.

IV. Determining Trojan Attributes

The complexity of integrated circuit designs and their corresponding trojans requires a more human-friendly scope. The taxonomy in [?] provides a series of thirty-three attributes which a trojan may or may not posses. Though it is desirable to be able to observe and directly extract the attributes a trojan possesses it is not always possible. In [?] 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.

A. *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 III-C. *Xilinx* tiles conform to purpose-specific groups or block types. Blocks contain subtypes that perform actions which pertain directly to *Location* category attributes as follows:

1) The **Processor** attribute pertains to the core functionality of the design logic. It can be awarded for the presence of a modified CLB tile or Interconnect tile in a trojan.
2) The **Memory** attribute can be awarded for the presence of modified BRAM components.
3) The **Input-Output** 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.

B. *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 following series of steps are first performed on the Golden design, followed by the Target; a comparison is then performed and a score is determined.

The Scatter Score method begins by determining the total number of configured tiles in the design. Equation 3 is used and the total is stored for subsequent use.

$$n = \sum_{x=0}^{X} \sum_{y=0}^{Y} T_{xy} \tag{3}$$

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

Then, the average horizontal and vertical positions are computed using Equations 4 and 5. Combined, these averages are used in conjunction to form the Position Median as seen in Equation 6. The Position Median provides an easy to interpret centralization of the design. Activation or deactivation of tiles by a trojan will shift the position median; this provides valuable insight into which *Physical Layout* category attributes the trojan possesses.

$$a_x = \frac{1}{n}\sum_{x=0}^{n} T_x \tag{4}$$

$$a_y = \frac{1}{n}\sum_{y=0}^{n} T_y \tag{5}$$

$$M_{xy} = (a_x,\ a_y) \tag{6}$$

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
M_{xy} = The Position Median

Once the Position Median has been computed the average values a_x and a_y can be used to determine the standard deviation of the position of tiles via Equations 7 and 8. These values are then combined to create the Scatter Score for the design, shown in Equation 9. Again, the activation or deactivation of tiles will alter the value of the Scatter Score providing valuable insight into the clustering of a trojan.

$$\sigma_x = \sqrt{\frac{1}{n}\sum_{x=0}^{X}(x_i - a_x)^2)} \tag{7}$$

$$\sigma_y = \sqrt{\frac{1}{n}\sum_{y=0}^{Y}(y_i - a_y)^2)} \tag{8}$$

$$S_{xy} = (\sigma_x,\ \sigma_y) \tag{9}$$

where:

σ_x = Standard deviation of the x coord. of configured tiles
σ_y = Standard deviation of the y coord. of configured tiles
T_x = The x coordinate of a configured tile
T_y = The y coordinate of a configured tile
S_{xy} = The Scatter Score

Equations 4 and 5 are used to create a rating known as the Position Median in Equation 6. The Position Median value provides a simple descriptor for where in the gate array the design is centralized. The Position Median in Equation 9 describes how spread out or, *clustered* the design is.

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 [**?**], 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.

C. Insertion and Abstraction Attributes

The linear nature of the manufacturing life-cycle implies a propagation of effects. For the purposes of this 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, it can be said that this trojan possesses insertion category attributes 3, 4 and 5. FPGA designs are made with a HDL. These languages dictate component arrangement in the Registry Transfer Level (RTL) abstraction level. Hence it can be said that trojans occurring in FPGAs take place in the System (attribute 6) and RTL (attribute 7).

D. Relation Matrix Use

Attributes which are not directly observable can be inferred using a systematic method of analyzing the rows and columns of the relation matrix presented in [**?**]. The FPGA Trojan Detector takes the attributes it is able to directly observe and uses them as input to this process. A thorough description of this method is given in [**?**] and [**?**].

V. FPGA Trojan Detector

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 standalone application; it is light-weight, cross platform and does not require installation.

A. *Technologies Used*

1) Xilinx: Xilinx is one of the two largest manufacturers of FPGAs; their devices are considerably more popular than their competitors.

2) Java: FPGA Trojan Detector was written in Java [**?**] primarily in order to interface with the Application Programming Interface (API) known as RapidSmith which is described in section V-A3. However, Java additionally allows the FPGA Trojan Detector to be a compact, cross-platform application. The custodians of the Java language, Sun Microsystems, promote the slogan 'write-once, run anywhere'. 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.

3) RapidSmith: RapidSmith is an APIs written in Java that enables Computer Aided Design (CAD) tool creation for *Xilinx* FPGAs [**?**]. Its purpose is to be used as a rapid prototyping platform for experimentation and research. 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.

VI. Results

To demonstrate its potential, FPGA Trojan Detector has been tested using a series of benchmarks. These benchmarks are included in the FPGA Trojan Detector application package as an example for users. In addition, a website known as the Hardware Trojan System [**?**] (HTS) was used to further visualize the results of the benchmark analysis. The HTS provides a unique tool know as the Classification web tool which accepts a list of taxonomic attributes as input and generates a directed graph. This graph allows for easy observation of each attribute inter-relates and affects one another.

A. *Priority Decoder*

The FPGA Trojan Detector was first tested using a small 'Priority Decoder' circuit presented by F. Brglez and H. Fujiwara in [**?**]. 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 indicating that there is no trojan present. The FPGA Trojan Detector successfully analyzed the Bitstreams and determined that there was no trojan present, as expected.

B. *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 clients labeled I_0 to I_9. To identify all ten clients, 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. 9. A trojan can be inserted into this circuit as shown in Fig. 10 (called a backdoor trojan).

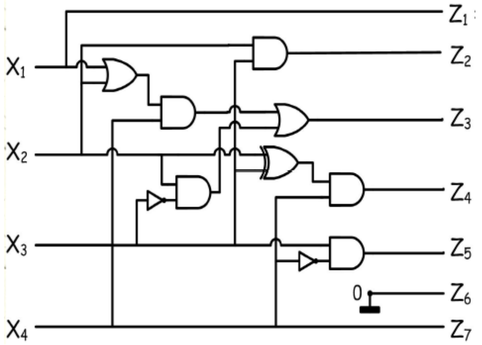


Fig. 9: A Simple User Authentication Circuit

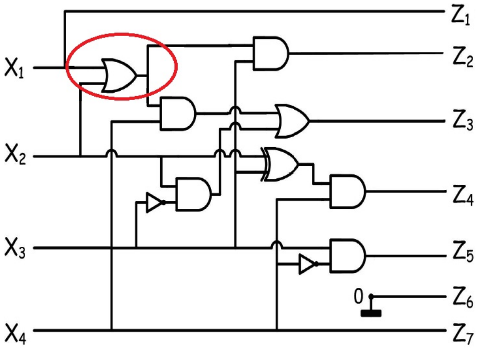


Fig. 10: The Back-door Trojan

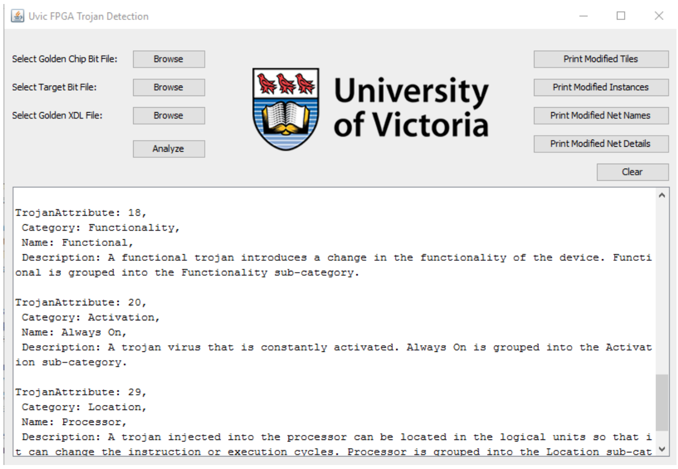
The outputs of the original and infected circuits are compared in Table III. A simple test will show that the circuit outputs the desired $F(x) = x^2$ for each of the users. However, upon closer inspection it is noted that the inputs corresponding to $x = 10$ to $x = 15$ are not used; there are no clients occupying those identifications. These unused inputs are referred to as 'dont-cares' (DC), 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 III 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. 10, 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 III. 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 Figures 9 and 10 were implemented and synthesized on a Virtex-5 240T (XC5VSX240T). The resultant Bitstreams were input to the FPGA Trojan Detector and the analysis detected a trojan. It output the following list of attributes as can be seen in Figure 11.

- Attribute 3: Fabrication
- Attribute 4: Testing
- Attribute 5: Assembly
- Attribute 6: System
- Attribute 7: RTL
- Attribute 12: Change in Functionality
- Attribute 17: Combinational
- 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 III. 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 IV-B. 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



```
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
```

Fig. 11: Results of The Authentication Circuit Trojan Detection

TABLE III: Outputs of the Circuits in Figs. 9 and 10 [**?**]

		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

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.

The attributes found to describe the back-door trojan in the User Authentication Circuit are entered into the Hardware Trojan System in the Classification tool. The matrix $\mathbf{R}$ is automatically analyzed and the visualization shown in Figure 12 is presented.

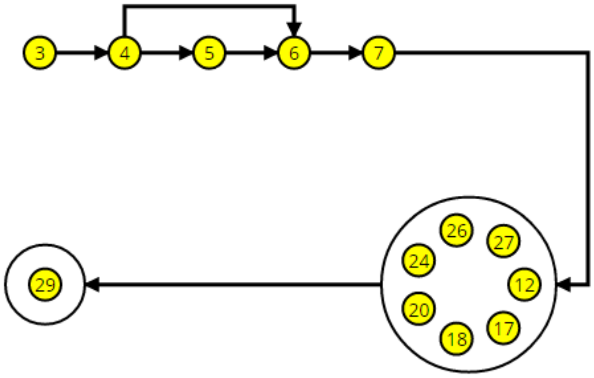


Fig. 12: Directed Graph of Back-Door Trojan generated by the Hardware Trojan System

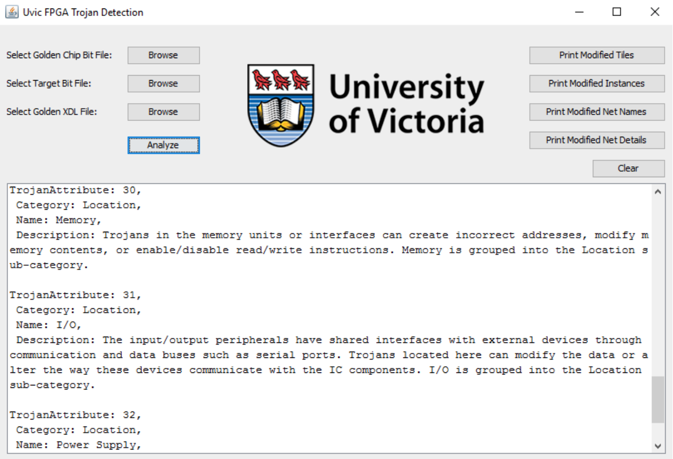
C. AES-T100

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 [**?**]. Their taxonomy is slightly different than the one used by the FPGA Trojan Detector which was proposed in [**?**], 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. [**?**]

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 13.



```
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,
```

Fig. 13: Results of Analysis on the AES-T100 Benchmark

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

- 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).

The resultant attributes are input to the Hardware Trojan System. Again, the matrix $\mathbf{R}$ is automatically analyzed and the visualization shown in Figure 14 is presented.

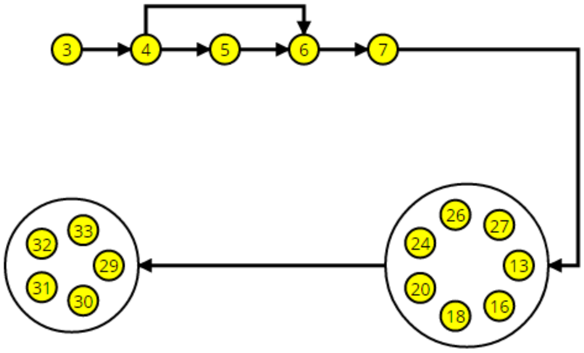


Fig. 14: Directed Graph of AES Circuit Trojan generated by the Hardware Trojan System

VII. Conclusion

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. 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 authors would like to thank...

References

[1] S. Adee, “The hunt for the kill switch,” *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, May 2008.

[2] Defense Science Board. (2005) Task force on high performance microchip supply. [online]. Available: http://www.acq.osd.mil/dsb/reports/ADA435563.pdf

[3] S. Moein, S. Khan, T. A. Gulliver, F. Gebali, and M. W. El-Kharashi, “An attribute based classification of hardware trojans,” in *Proc. Int. Conf. on Computer Eng. and Sys.*, Cairo, Egypt, Dec. 2015, pp. 351–356.

Nicholas Houghton received his masters in Applied Science from the University of Victoria, BC, Canada in 2016. He received the B.Eng degree in Computer Engineering in 2015 also from the University of Victoria.

Samer Moein received the Ph.D. degree in Computer Engineering from the University of Victoria, Victoria, BC, Canada, in 2015. Currently, he is a Post Doctoral Fellow in the Department of Electrical and Computer Engineering at the University of Victoria. He received the B.Sc. degree in Computer Engineering from Kuwait University, Kuwait in 2004, and the M.Sc. degree in Computer Engineering from Kuwait University, Kuwait in 2011. His research interests include computer security, cryptography, and crypto-processors.

Fayez Gebali received the B.Sc. degree in Electrical Engineering (first class honors) from Cairo University, the B.Sc. degree in Mathematics (first class honors) from Ain Shams University, and the Ph.D. degree in Electrical Engineering form the University of British Columbia where he held an NSERC postgraduate scholarship. Dr. Gebali is a Professor in the Department of Electrical and Computer Engineering at the University of Victoria, and is currently Department Chair. His research interests include parallel algorithms, networks-on-chip, three-dimensional integrated circuits, digital communications, and computer arithmetic.