Journal of Electronic Testing (2022) 38:247–260
https://doi.org/10.1007/s10836-022-06004-z

The Detection of Malicious Modifications in the FPGA

Kamran Zahid[1]

Received: 1 February 2022 / Accepted: 17 May 2022 / Published online: 30 May 2022

Abstract
Field Programmable Gate Arrays (FPGAs) are being widely used in a variety of embedded applications. Due to their programmable feature, FPGAs are the perfect choice for various hardware-based systems. In many of the competing types of FPGAs, the dominant types are Static Random-Access Memory (SRAM) based which can be reprogrammed at any stage of execution of a job. SRAM-based FPGAs are volatile and need an external memory to store configuration bitstream that is vulnerable to attacks. In the development as well as deployment stages, the threat of malicious modifications or inserting Hardware Trojans (HTs) into the bitstream is always present. FPGA's bitstream can be infiltrated or corrupted in a non-invasive manner that may cause fatal consequences. Therefore, a framework is proposed that uses Xilinx Design Language (XDL) or Native Circuit Description (NCD) files that can be extracted from the infected bitstream of FPGA. Xilinx Command Line tools are used to get complete information on hardware primitives, resource utilization, timing constraints, and power summaries from XDL/NCD files in textual form. Further, Natural Language Processing (NLP) has been employed to extract the syntactic features from the descriptive artifact to find the malicious modifications/HTs. The proposed framework also identifies the types of the detected HTs and provides a good understanding to study the behavior of trojans. For logic implementation and testing, Xilinx ISE 14.7 along with PlanAhead^{TM} and FPGA Editor design tools are used. The experimental results show that the proposed framework can be successfully used for the detection of malicious modifications/HTs with optimal accuracy.

Keywords FPGA · Trojan · XDL · NCD · NLP

1 Introduction

Field Programmable Gate Array (FPGA) is an integrated circuit (IC) that consists of a matrix of Configurable Logic Blocks (CLBs) that are connected through programmable interconnects. FPGAs can be reprogrammed to the desired functional requirements and applications. This feature differentiates FPGAs from Application-Specific Integrated Circuits (ASICs) which are designed for specific tasks and can not be reprogrammed. In various types of FPGAs, the SRAM-based FPGAs are widely used due to their advantages in area, speed, and re-programmability. Some common applications are aerospace & defense, medical electronics, ASIC prototyping, automotive, video & image processing, consumer electronics, data center, high performance computing, industrial & scientific instruments, security systems, and wired and wireless communications.

There are two modes of FPGA programming, slave mode, and master mode. In the slave mode, FPGA is programmed by an external master device via a boundary scan (JTAG) or by using a dedicated configuration interface. While in the master mode, configuration data (bitstream) is stored in external nonvolatile memories such as Programmable Read-Only Memory (PROM) and serial & Parallel FLASH, etc... During the configuration process, the bitstream is loaded into the FPGA CLBs to run a specific application. Therefore, unlike ASIC, FPGA bitstream is more vulnerable to various attacks.

The threat of malicious modifications/HTs created by malicious developers or intruders is a serious problem in modern VLSI systems. HT can be defined as an intentional malicious change in the circuit design that leads to undesirable behavior when it is deployed [39]. HT infected FPGAs may experience changes in their functionality and operations that lead to degraded or unreliable performance. HT can be

Responsible Editor: M. Tehranipoor

Kamran Zahid
imkamranzahid@gmail.com

1 Department of Electrical Engineering, National University of Computer and Emerging Sciences, Islamabad, Pakistan

inserted into the FPGA bitstream by modifying the configuration (bit) file stored in onboard memory. An attacker may need to know the logic and internal wires and preferably where the configuration file is physically located. Therefore, FPGA bitstream can be obtained or altered by wire-tap [10] which may bring critical security issues for FPGA-based applications. Meanwhile, Chakraborty *et al.* implanted HTs in FPGA bitstream by finding the empty space in it [5]. By using the HAL framework [13], the bitstream can be identified and relevant netlist components can be manipulated. For the first time, Moradi *et al.* presented the technique to find out the vulnerabilities of bitstream encryption in [25]. They successfully exemplified an attack at bitstream decryption of a Xilinx Virtex-II FPGA. In [11], the authors extended their research against bitstream encryption in the latest Xilinx FPGAs. They launched low-cost successful attacks by exploiting a design flaw that leaks the decrypted bitstream. Swierczynski *et al.* proposed an attack that focuses on unencrypted bitstreams named BiFI to obtain valuable information by manipulating an encrypted bitstream [37] and a successful attack was claimed for different Xilinx FPGAs i.e. Spartan-6 and Virtex-5. Besides the attacks that require physical connectivity, the side-channel attack is another, where no physical access to the hardware is required [47]. The bitstream can be reverse-engineered not only to understand the logical functions but can also be used to insert malicious logic into it. Therefore, to make devices more secure from the aforementioned vulnerabilities, system security attracts more attention and new ways are imperative to find malicious changes.

In this article, a new framework is proposed that can detect not only the existence of malicious modifications or HTs but also identifies the attacker's intention using NCD or XDL files that can be obtained by using reverse engineering (RE) methods. The proposed framework has been demonstrated by employing the Xilinx Command Line tools that help to find out the type of HTs. Moreover, these tools are used to populate the descriptive artifacts that have been used as input into the NLP module to extract the syntactic features. The experimental results show that the proposed scheme can detect HTs with optimal accuracy. Furthermore, required countermeasures are also proposed for the security of FPGA devices.

The rest of the paper is organized as follows: Section 2 gives insight into the background and related work that summarizes current research in FPGA-based Trojan taxonomy and detection. Section 3 demonstrates the case study that describes the types of HTs that are also considered for implementation. The proposed framework is discussed in Sect. 4. This paper ends up with the conclusion and future directions after discussing the countermeasures.

2 Background and Related Work

2.1 FPGA Trojan Taxonomy, Attacks, and Detection Approaches

HT taxonomy, activation or insertion mechanism, physical and functional characteristics have already been proposed in [22, 38]. These articles provide a good concept about HT types and their functionalities. HTs attacks, threat analysis, and relevant countermeasures were explained by [4]. Different types of FPGA-based HTs, their point of entry, and their method of creation were thoroughly discussed in [20] which is a helpful study to understand the behavior of FPGA-based Trojans. Furthermore, the vulnerability analysis flow and benchmarks proposed by [32] provide profound concepts about the Trojans and their implementation. The HTs' functionalities can be classified as follows:

- **Denial of service:** A Trojan either temporarily or permanently stops target devices, resulting in a denial of service which is crucial for digital systems.
- **Performance Degradation:** A Trojan can degrade the performance of the target devices by changing the parameters.
- **Change of functionality:** A Trojan can malfunction the target devices.
- **Information leakage:** A Trojan can leak confidential information from target devices through hidden and open interfaces.

For FPGAs, Iwase et al. [19] used a Support Vector Machine (SVM) to detect HT. Their proposed method utilizes the difference in power consumption between 'with and without Trojan' which is time-consuming and needs additional equipment. An interesting work related to the design file analysis is an Unused Circuit Identification (UCI) technique that detects the never used RTL code and shrinks the code space left for HTs was proposed by Hicks et al. [16]. Guo et al. found additional functions in RTL code that were not defined in the design specification [15]. However, the equivalence checking method takes much time to detect malicious changes. In [46], Zhang et al. proposed an HT detection mechanism by identifying the redundant logic in the netlist. However, the technique has certain limitations to detect trigger-based Trojans.

Although bitstream reverse engineering is difficult, however, following research articles claim certain information extraction from the bitstream, indicating the fact that it is not impossible. FPGA reverse engineering aims at converting the bitstream to a file that contains netlist information. The same process can be extended to netlist reversing that further clarifies the high-level descriptions. Ziener et al. [48] investigated the Virtex II FPGA bitstream and extracted the Look-Up Table (LUT) contents. The extraction lies in partial bitstream reversal techniques that focuse on configurable blocks like LUTs. In the study [7], a netlist was converted to XDL file and designed a tool to generate Hardware Description Language (HDL) code. The study was limited to netlist re-engineering rather than bitstream reverse engineering. An algorithm was proposed to construct the relationship between the configurable elements and the bitstream for multiple Xilinx FPGAs in [26]. Later on, Benz et al. presented a tool called BIL using the same algorithm and successfully recovered a fraction of the netlist from the bitstream [3]. Both of these reverse engineering tools have low accuracy and extract limited tile resources [18]. Further improvements were proposed in [9] using two algorithms, Position Known (PK) and Position Un-Known (PUK) analysis. These analyses were used to obtain the bitstream mapping information and to convert the bitstream to NCD/XDL netlists. The accuracy rate on the XC5VLX50T device was claimed more than 88% even in the worst-case scenario. However, for exact Trojan detection, such accuracy is insufficient. There is a possibility that some malicious circuit information might not get converted completely. The "project IceStorm", a reversing tool was presented in [40] which effectively decompiles the bitstreams of Lattice iCE40 FPGAs that have a minimal architecture with limited function units and types of tiles [18].

In [12], Ender et al. improved the bitstream reversing techniques by simplifying routing extraction mechanisms. There are various research articles [27, 28, 30, 35, 48] that discuss the bitstream analysis, generation, manipulation, and reverse engineering of SRAM-based FPGAs. These research articles are helpful to retrieve and study the content of bitstream. Some tools, like RapidSmith [24] and Torc [36] were presented to improve the flexibility for designers and researchers. Rapidsmith alone cannot be used for reverse engineering, however, it can provide a suitable function for XDL manipulation to assist the other tools used in reverse engineering. Similarly, Torc is helpful in reading, writing, and manipulating FPGA bitstream, but is unsuitable for configuration frame contents. Later, [8] extended Torc's work for reconfigurable computing. Furthermore, previously mentioned articles do not address the types of the detected HTs.

In [43], Yoo et. al used Xilinx ISE design tools and applied reverse engineering techniques to detect malicious modifications with an accuracy of 88%. So there is a possibility that some circuit information may not be fully recovered. This creates uncertainty to find the exact malicious changes in the recovered files. Further, Hassan Salmani [31] proposed the HT detection technique using gate-level netlist and unsupervised clustering analysis, which is a type of Machine Learning (ML) algorithm. His proposed method does not require any golden model, however, timing complexity increases as the number of signals increases. Yoon *et al.* improved the BIL reversing by generating a tool to recover the netlist from bitstream [44] and suggested an ML approach for HT detection. The authors claimed that their proposed BRET method recovery rate was 95.9% for PIPs and 98.4% for LUTs on average. BIL was executed for the same data samples with a recovery rate of 54.4% for PIP. Furthermore, BRET also achieved a recovery rate of 97.9% for the INT tiles only, while BIL had 82.1%, thereby, achieving a good percentage of recovery rates. While Zhang et al. [45] proposed a comprehensive tool-chain for Bit to RTL conversion and claimed that their proposed model can convert bitstream to netlist and the netlist to RTL code with 100% effectiveness. They also suggested the HTs detection method from netlists and converted RTL code. Although [45] is a close source, however, we can take the benefit of the proposed methods. Therefore, their presented results in the literature have been used for reference.

2.2 Xilinx ISE Work Flow

Xilinx (AMD) offers industry-leading and commercially available low-cost FPGAs and captures a larger share of the approximately $135 billion markets [42] The user implements the target design described in the modeling language (VHDL or Verilog) to FPGA using the Xilinx ISE Design Suite. Xilinx ISE provides the support for commercially available low-cost Xilinx FPGAs while Vivado Suite starts with Virtex-7, Kintex-7, Artix-7, and Zynq-7000 FPGAs that usually have a high cost. There are several steps involved in converting the design to the FPGA programming file, as shown in Fig. 1. Their details are given below:

1. Once a design idea and specifications are finalized, design entry is the next step in the Integrated Software Environment (ISE) design where the source files are created. These files can be any of the formats of HDL, such as VHDL or Verilog.
2. Synthesis involves the conversion of HDL sources into architecture-specific netlist files, which are known as NGC files.
3. The implementation consists of Translate and Map processes, where the logical netlist gets converted into a physical file format and downloaded to the target device.
4. Place and route (PAR) takes a mapped NCD file, places & routes the design, and generates the P&Red NCD file that is used for the bitstream generation.
5. Finally, the bit file is generated from the NCD file and is used to program the target FPGA device. The bit file is stored in an external memory as an MCS (configuration memory) file. On power-up, the stored bitstream is fed to the FPGA to perform required tasks.

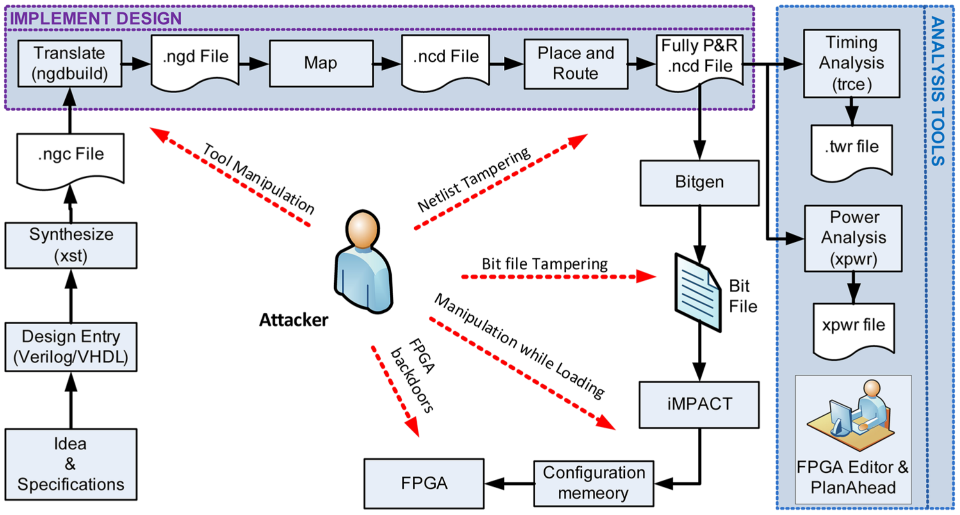


Fig. 1 Xilinx Design Flow

3 Case Study

FPGA-based systems have a unique design flow and supply chain models which bring significant security challenges. Low-cost FPGAs do not have built-in support for encrypted bitstreams [14] and are usually stored as plain text on the same circuit board along with the FPGA. So, the manipulation of the bitstream is possible during different stages of the FPGA's life cycle. It can be intercepted during shipment, sale, and design processes as shown in Fig. 1. The HTs can be added to the bitstream in two different ways [20]:

1. If the Trojan circuit is inserted into the existing bitstream file at locations where resources were not originally used, or unoccupied FPGA resources are configured using an additional malicious circuit, then such type of Trojan is called Type-I Trojan.
2. If the Trojan and original circuits are interconnected, then such type of The Trojan is called Type-II Trojan. These Trojans can either interfere with its operation or extract information from the original scheme.

Given previous sections, it is evident that bitstream can be altered or required information can be extracted by converting it into RTL or some other descriptive formats. After obtaining the necessary data and having sound knowledge, adversaries can get a clear idea of logic implementation in that bitstream. Therefore, the bitstream can be modified whether it is Type-I or Type-II. The same goes for the person who is going to detect HT in the extracted bitstream. Many types of HTs have already been discussed in different research articles. In this paper, the following Trojans [4, 20, 22, 29, 32, 38] are considered for implementation and detection, which:

- Use unallocated resources
- Cause frequency fluctuations
- Increase path delay
- Leak information
- Induce power fluctuations

4 KZ Framework: A Proposed Detection Scheme for Malicious Modifications/HTs

Many HTs proposed so far in the literature are related to the modification of the bitstream or design files. These modifications consist of adding a small circuit that is implemented and added to the project. To find such modifications XDL or NCD file is needed that can be extracted from the infected bitstream using [45]. A framework is proposed, named ***"KZ Framework"*** as shown in Figs. 2 and 3. The main advantage of this model is that NLP has been employed to extract the syntactic features from the descriptive artifact to find the malicious modifications/HTs. It also identifies the types of the detected HTs and provides a good understanding to study the behavior of the trojans. Furthermore, Table 1 shows the comparison of research contributions and limitations of different published techniques with the proposed one. To test the proposed framework, trojans are created by using Xilinx ISE 14.7 and FPGA Editor. All effectuated changes are counter verified by the Xilinx PlanAhead^{TM} tool.

4.1 Xilinx Command-Line Tools

Xilinx provides a suite of "Command-Line Tools" that allows for implementing and verifying FPGA designs. These tools can be run in the standard design flow or can be used by special command-line options. These command-line tools generate files that are helpful to detect HTs or malicious

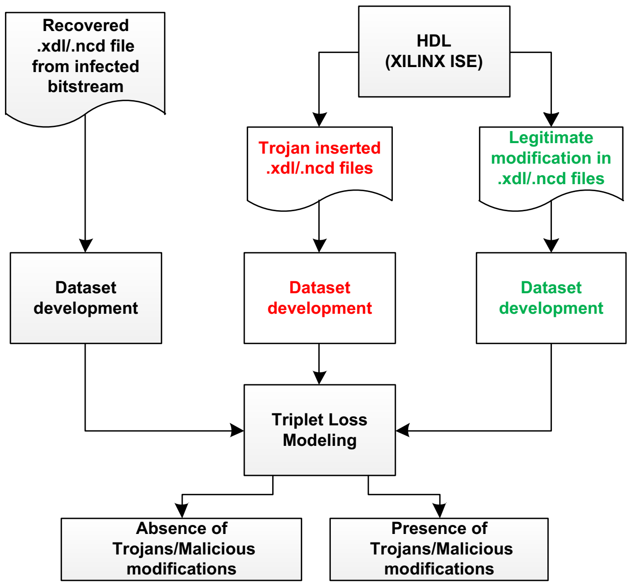

Recovered .xdl/.ncd file from infected bitstream
HDL (XILINX ISE)
Trojan inserted .xdl/.ncd files
Legitimate modification in .xdl/.ncd files
Dataset development
Dataset development
Dataset development
Triplet Loss Modeling
Absence of Trojans/Malicious modifications
Presence of Trojans/Malicious modifications

Fig. 2 KZ Framework: A Proposed Model for Trojan detection

changes. For the proposed design **XDL, PAR, PIN2UCF, TRACE, ReportGen**, and **XPWR** command-line tools are used [41].

4.2 NLP Based Detection Technique

Natural Language Processing (NLP) is the ability of a computer program to understand the written or spoken language referred to as natural language [17]. NLP uses Artificial Intelligence (AI) to take real-world input, process it, and make sense of it in a way that a computer can understand. Syntax and semantic analysis are two main techniques used with NLP. For that Term Frequency (TF)-Inverse Document Frequency (IDF) [34] is a technique that is used to find the meaning of sentences and cancels out the incapability of the Bag of Words technique which is good for text classification. It has many uses, however, in automated text analysis, it is very useful for scoring words in Machine Learning (ML) algorithms.

For the proposed design, information retrieval (IR), evaluation, refinements, and IR scores using the Triplet loss (TL) function are used. TL is a loss function for ML algorithms where a matching input (called positive) and a non-matching input (called negative) are compared to a reference input (called the anchor). The distance from the anchor to the negative input is maximized, and the distance from the anchor to the positive is minimized [6, 33]. For the proposed framework, Xilinx command-line tools are used to generate **".par, .txt, .xdl, .ucf, .pwr, .twr** and **.dly"** files from XDL/NCD netlist files that can be extracted from the infected bitstream using [45]. The generated files have complete information on hardware primitives, resource utilization, timing constraints & delays, and power summaries. Furthermore, using NLP features and energy flow indicators, three types of datasets are produced i.e., "Legitimate modified XDL/NCD", "Trojan injected in XDL/NCD", and "recovered XDL/NCD from infected bitstream". The first two datasets are generated using Xilinx ISE and based on the distance values, these datasets are compared with the third one using the TL function. The experimental results indicate the existence of certain types of trojans or malfunctions in the extracted XDL and NCD files.

Fig. 3 KZ Framework: Dataset Development

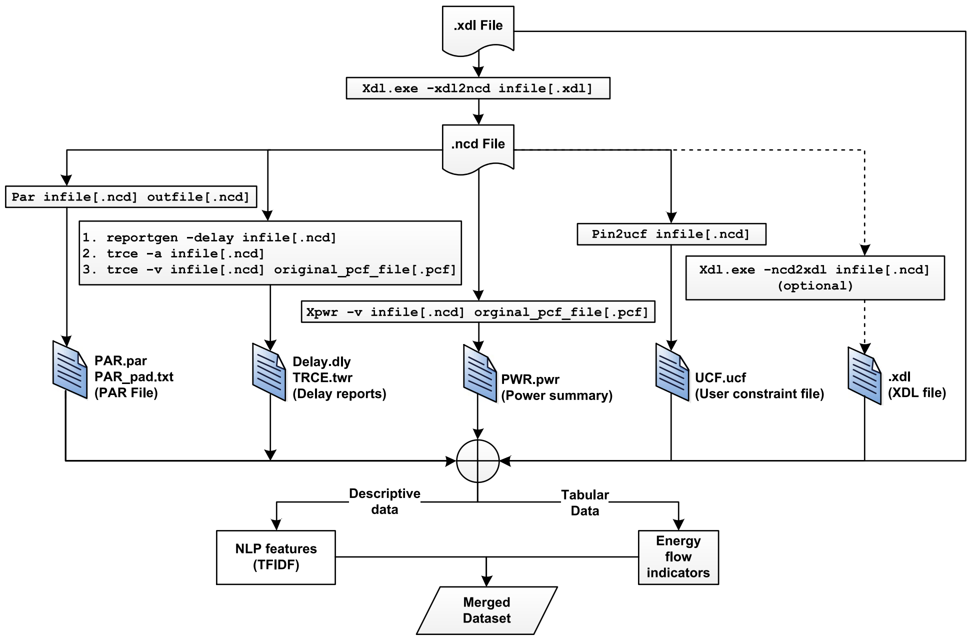

.xdl File
Xdl.exe -xdl2ncd infile[.xdl]
.ncd File
Par infile[.ncd] outfile[.ncd]
1. reportgen -delay infile[.ncd]
2. trce -a infile[.ncd]
3. trce -v infile[.ncd] original_pcf_file[.pcf]
Pin2ucf infile[.ncd]
Xdl.exe -ncd2xdl infile[.ncd] (optional)
Xpwr -v infile[.ncd] orginal_pcf_file[.pcf]
PAR.par PAR_pad.txt (PAR File)
Delay.dly TRCE.twr (Delay reports)
PWR.pwr (Power summary)
UCF.ucf (User constraint file)
.xdl (XDL file)
Descriptive data
Tabular Data
NLP features (TFIDF)
Energy flow indicators
Merged Dataset

Table 1 Research contributions and limitations of reverse engineering & HT detection

Technique	Article	Research Contribution	Limitation	HT detection	Type of HT detection
Feature Analysis	Iwase et al. [19]	Used chip power signature to detect HT by using SVM.	Time-consuming and needs additional equipment.	Yes	No
	Hicks et al. [16]	Proposed Unused UCI technique that detects the never used RTL code and shrinking the code space left for HTs.	Time-consuming.	Yes	No
	Guo et al. [15]	Found additional functions in RTL Code that are not defined in design specification.	Equivalence checking method takes much time to detect malicious changes.	Yes	No
Reverse Engineering	Ziener et al. [48]	Extracted the Look-Up Table (LUT) contents from Virtex II FPGA bitstream.	Partial bitstream reversal.	No	No
	Cheremisinov [7]	Netlist was converted to an XDL file and generated the Hardware Description Language (HDL) code.	Limited to netlist re-engineering rather than bitstream reverse engineering.	No	No
	Note and Rannaud [26]	Construct the relationship between configurable elements and the bitstream.	Low accuracy and extract limited tile resources.	No	No
	Benz et al. [3]	Presented BIL that recovered a fraction of the netlist from bitstream.	Low accuracy and extract limited tile resources.	No	No
	Yoo et al. [43]	Detected harmful modifications using reverse engineering methods.	Less accurate about 88%	No	No
	Yoon et al. [44]	Recovered netlist from the bitstream and suggested an ML approach for HT detection.	The method does not have full recovery rates: 95.9% for PIPs, 98.4% for LUTs, 97.9% for the INT tiles.	Yes	No
	Zhang et al. [45]	Proposed a tool-chain for Bit to RTL conversion with 100% accuracy.	Methods for detection of HTs types must be incorporated.	Yes	No
Reverse Engineering & Feature selection	Proposed	XDL/NCD files (recovered from RE) are converted to textual files and applied NLP/TFIDF/TL techniques to find HTs and their types.	Reference files needed to automate the process.	Yes	Yes

The reason for applying NLP is to automate the Trojan detection process without the need for a golden file. The reference files will no longer be required once the model is trained. It will automatically identify the malicious modifications related to different types of Trojans in extracted files. Moreover, the degree of semantic similarity and dissimilarity would be accomplished through NLP. This would segregate the syntactic similar but semantic dissimilar elements.

To test the proposed framework, the Advanced Encryption Standard (AES-256) [1] core is implemented in Xilinx Virtex-5 (device: xc5vlx50t) and generated two datasets (Legitimate modified (Trojan Free) or Trojans inserted). For testing and evaluation, malicious logic/Trojan circuits are implemented and placed in AES_256 core (considered as recovered from infected bitstream) that are discussed in detail in further sections.

4.3 Detection of the Trojans That Use Unallocated Resources

HTs may be added to the design by utilizing unallocated hardware resources. Adversaries can use the same technique to modify the original design by inserting extra logic into these vacant places. Such HT circuits can be triggered internally or from outside the chip using unallocated I/O ports. Three command-line tools "XDL, PAR, and PIN2UCF" are used to detect aforesaid modifications.

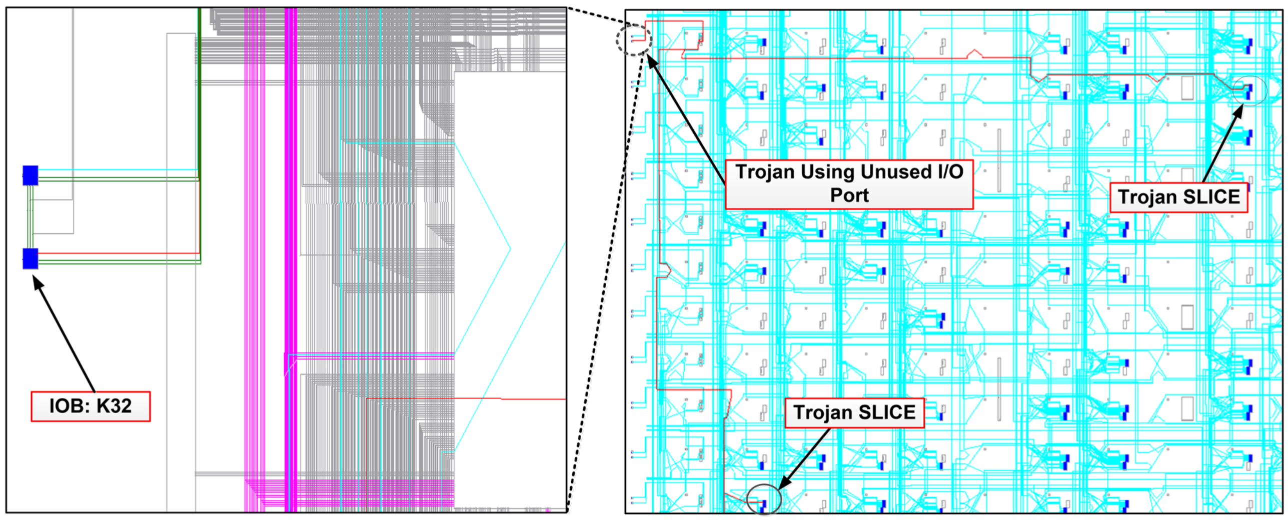


Fig. 4 Trojan using unallocated spaces and I/O pin (FPGA Editor's View)

4.3.1 XDL

The NCD file can be converted to an XDL file and vice versa. The XDL file is a textual representation of the netlist that has the low-level description of the FPGA's internal state. It has complete information on Programmable Logic Points (PLPs) and Programmable Interconnect Points (PIPs). For XDL/NCD file creation, the XDL tool is used as follows:

`xdl.exe -ncd2xdl infile[.ncd]` //ncd2xdl generation
`xdl.exe -xdl2ncd infile[.xdl]` //xdl2ncd generation

4.3.2 PAR

PAR takes a mapped NCD file as input and generates an NCD file along with a PAR report (having utilization summary of all placements and routing iterations) and a text file (containing I/O pins assignments in an ASCII text version). To get these files, the PAR tool is used as follows:

`par infile[.ncd] outfile[.ncd]` //par & text file generation

4.3.3 PIN2UCF

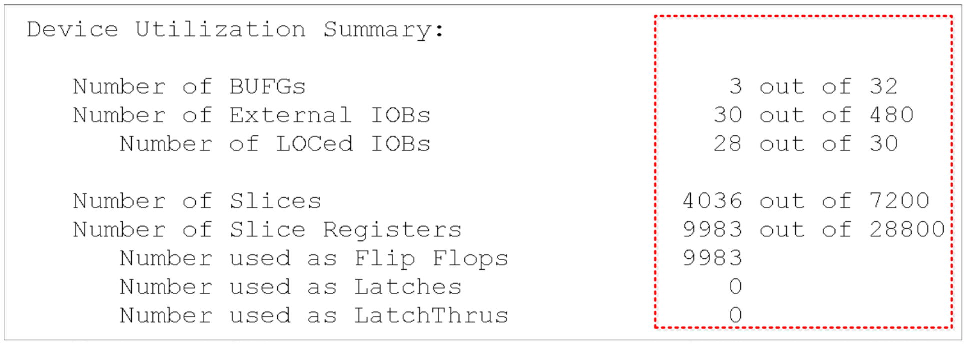
PIN2UCF takes an NCD file as input and writes information to a user constraint file (UCF). For this purpose, the following PIN2UCF command is used:

```
pin2ucf infile[.ncd]
```

For an experiment, a kill switch and its required malicious logic are implemented and placed in the AES_256 core, as shown in Fig. 4. These changes are considered for the dataset creation of "recovered XDL/NCD from the infected bitstream". The function of kill switch circuitry is to damage the whole encryption process by zeroing all the keys. The logic is triggered from an external input signal that is fed into the design using an I/O pin.

By using the PAR command two files are generated *PAR_pad.txt* and *PAR.par*. Similarly, the UCF.ucf file is also generated

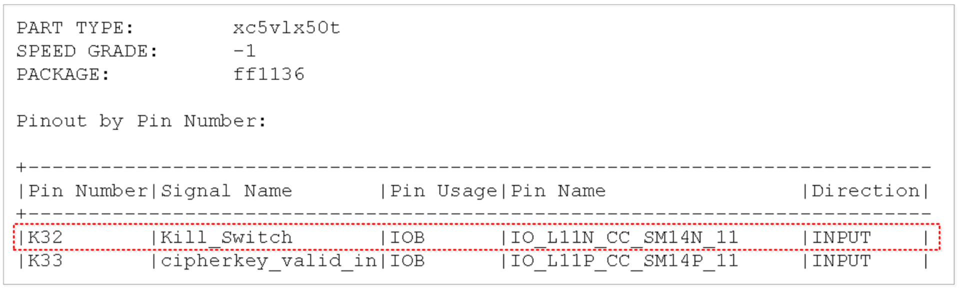
Fig. 5 Detection of malicious resource utilization using Par file



```
Device Utilization Summary:

   Number of BUFGs                            3 out of 32
   Number of External IOBs                   30 out of 480
      Number of LOCed IOBs                   28 out of 30

   Number of Slices                        4036 out of 7200
   Number of Slice Registers               9983 out of 28800
      Number used as Flip Flops            9983
      Number used as Latches                  0
      Number used as LatchThrus               0
```



```
PART TYPE:          xc5vlx50t
SPEED GRADE:        -1
PACKAGE:            ff1136

Pinout by Pin Number:

+-------------------------------------------------------------------------
|Pin Number|Signal Name        |Pin Usage|Pin Name                |Direction|
+-------------------------------------------------------------------------
|K32       |Kill_Switch        |IOB      |IO_L11N_CC_SM14N_11     |INPUT    |
|K33       |cipherkey_valid_in|IOB      |IO_L11P_CC_SM14P_11     |INPUT    |
```

Fig. 6 Detection of malicious I/O details using .ucf or PAR_pad.txt files

by the PIN2UCF command that has the complete information of the pins assignments. The descriptive and tabular information are separated. Later, the descriptive information has been given to the NLP module, wherein, syntactic features are read. The TFIDF measure is used to assign a score to syntactic features. The score of the features represents the relevance of the input file to the Trojan and the clean descriptive file. After that, energy flow indicators (e.g., input, power, frequency, output, etc...) are sorted out. Finally, the data output from the NLP module, and sorted out energy flow indicators are merged. After applying the TL modeling technique, the existence of malicious modifications or Trojan circuits is detected that are using unallocated resources as shown in Figs. 5 and 6. Furthermore, to find out the exact details of malicious logic, the XDL command is used. A *Malicious.xdl* file is generated using the XDL command and further evaluated on the same framework. The same malicious modifications were found, as shown in Fig. 7. The test was performed on real datasets generated by Xilinx ISE. IOB name "Kill Switch" appears only for that reason. In a recovered netlist, it would not be preserved the same as used in the synthetic file. However, the IOB location can be extracted by the proposed method. IOB detail can also be compared with extracted XDL's PIPs for further clarification. Malicious modification can be identified once the IOB location is detected.

4.4 Detection of Trojans Causing Frequency Fluctuations

A Trojan can alter the timing of the circuit by utilizing the path delay effects. Due to the loading effects of Trojan circuits on internal paths, the operating frequency can be changed. To detect such Trojans, the TRACE command-line tool is used.

4.4.1 TRACE

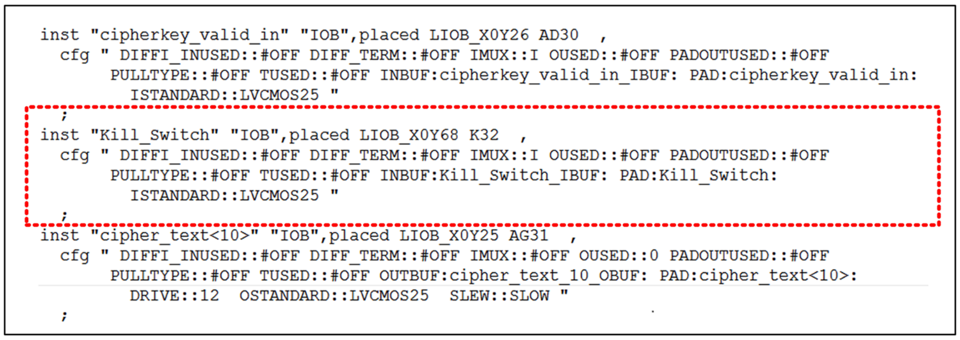
The Timing Reporter and Circuit Evaluator (TRACE) tool makes static timing analysis based on input timing constraints of an FPGA design and generates a report file with a `.twr` extension. In addition, TRACE can also be run on unplaced designs, partially placed & routed designs, and completely placed & routed designs. The following command-line tool is used to create a `.twr` file:

```
trce -a infile[.ncd]
```

To test the proposed model, the position of BUFG is changed from "BUFGCTRL_X0Y17" to "BUFGCTRL_X0Y0", while the input "clk" pad is located at "AH15". It changes CLK_BUFGP's clock period from 7.247ns to 7.497ns. In addition, the corresponding maximum operating frequency is also changed from 137.988MHz down to 133.387MHz. These changes are considered for the dataset creation of "recovered XDL/NCD from the infected bitstream". After applying the KZ framework, Fig. 8 shows the detection of malicious modifications in the .twr file.

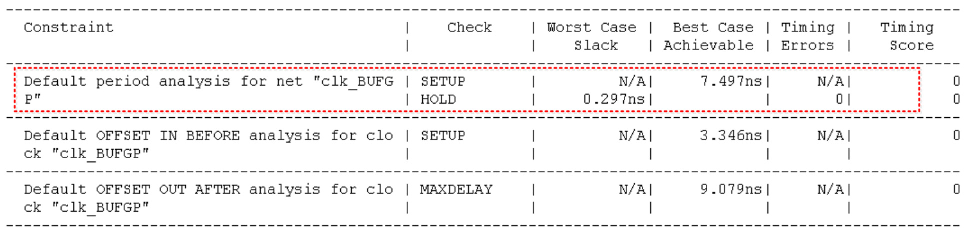
4.5 Detection of Trojans that Increase Path Delay

The Trojan circuits can affect route delay in several ways. These circuits add more gate delays to the original paths. Therefore,



```
inst "cipherkey_valid_in" "IOB",placed LIOB_X0Y26 AD30  ,
  cfg " DIFFI_INUSED::#OFF DIFF_TERM::#OFF IMUX::I OUSED::#OFF PADOUTUSED::#OFF
       PULLTYPE::#OFF TUSED::#OFF INBUF:cipherkey_valid_in_IBUF: PAD:cipherkey_valid_in:
         ISTANDARD::LVCMOS25 "
  ;
inst "Kill_Switch" "IOB",placed LIOB_X0Y68 K32  ,
  cfg " DIFFI_INUSED::#OFF DIFF_TERM::#OFF IMUX::I OUSED::#OFF PADOUTUSED::#OFF
       PULLTYPE::#OFF TUSED::#OFF INBUF:Kill_Switch_IBUF: PAD:Kill_Switch:
         ISTANDARD::LVCMOS25 "
  ;
inst "cipher_text<10>" "IOB",placed LIOB_X0Y25 AG31  ,
  cfg " DIFFI_INUSED::#OFF DIFF_TERM::#OFF IMUX::#OFF OUSED::0 PADOUTUSED::#OFF
       PULLTYPE::#OFF TUSED::#OFF OUTBUF:cipher_text_10_OBUF: PAD:cipher_text<10>:
         DRIVE::12  OSTANDARD::LVCMOS25  SLEW::SLOW "
  ;
```

Fig. 7 Detection of malicious modifications using .xdl file



```
--------------------------------------------------------------------------------------------------------
  Constraint                                |    Check     | Worst Case |  Best Case | Timing |   Timing
                                            |              |    Slack   | Achievable | Errors |    Score
--------------------------------------------------------------------------------------------------------
  Default period analysis for net "clk_BUFG | SETUP        |         N/A|     7.497ns|     N/A|         0
  P"                                        | HOLD         |     0.297ns|            |       0|         0
--------------------------------------------------------------------------------------------------------
  Default OFFSET IN BEFORE analysis for clo | SETUP        |         N/A|     3.346ns|     N/A|         0
  ck "clk_BUFGP"                            |              |            |            |        |
--------------------------------------------------------------------------------------------------------
  Default OFFSET OUT AFTER analysis for clo | MAXDELAY     |         N/A|     9.079ns|     N/A|         0
  ck "clk_BUFGP"                            |              |            |            |        |
--------------------------------------------------------------------------------------------------------
```

Fig. 8 Detection of malicious changes in .twr file

one important parameter that is used to detect the Trojans is the path delay. Such Trojans can be created by modifying interconnects connecting lookup tables (LUTs) across slices of CLBs. To detect such Trojans, the ReportGen command-line tool is used.

4.5.1 ReportGen

ReportGen accepts an NCD file and generates various pad reports and a log file (contains standard usage information) along with the `.dly` file. The `.dly` file contains path delay information on each net of a design. These files are generated by the following tool:

`reportgen -delay infile[.ncd]` //.dly file generation

In AES_256 Core, 256-bit Cipher Key is fed to the design through "cipher_key[7:0]" input ports in multiple cycles. The cipher_key_delay is a flip-flop that stores all the key values before feeding to the design. To test the proposed scheme, the net cipher_key_0_IBUF is selected to insert path delay. The selected ports and components of the schematic are shown in Fig. 9. The net delay is produced by changing the SLICE (name: cipher_key_delay<3>) position from $X0Y75$ to $X22Y105$, while the "cipher_key[0]" IOB is placed at location "AP32" as shown in Figs. 10 and 11. In doing this, the corresponding switch boxes (SB) and PIPs were also changed. These modifications were carried out by using FPGA Editor and the modified file was considered a malicious NCD file. The comparison between PIPs of both Trojan-free or malicious XDL files is shown in Figs. 12 and 13. However, more delay can be inserted by increasing logic levels. In this way, the logic delay (i.e. Tilo) can be added to the overall path delay. To check complete logic and net delays the following command is used:

```
trce -v infile[.ncd]
```

-v option is used to generate a verbose report.

Moreover, by using the ReportGen command, `.dly` file is generated. The comparison between both files is carried out and analyzed the path delays for each signal using the KZ framework. The delay reported for the net cipher_key_0_IBUF to cipher_key_delay<3>is from 2.060 ns to 3.354 ns.

4.6 Detection of Trojans That Leak Information

There are many ways in which a Trojan can leak sensitive or secret information by providing backdoor channels in FPGA. The main source of information leakage is data ports. Therefore, it is a challenging task to check the connectivity of benign or legitimate I/O ports with the ports that are maliciously inserted by some adversaries to leak sensitive information. To detect such changes or internal connections, ReportGen and PIN2UCF tools are used.

To test the proposed technique, the NCD file is changed by using FPGA Editor's "Probes" option to leak sensitive information. The net cipher_key_0_IBUF is manually connected to another IO port (name: my_pin) to exfiltrate cipher_key[0]

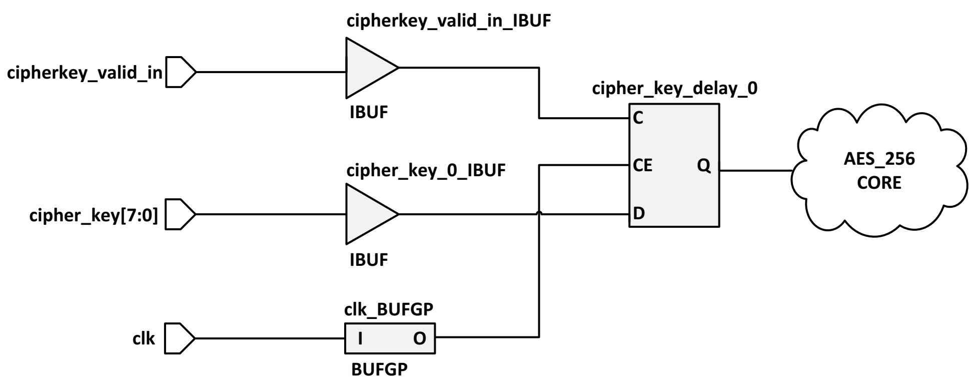


Fig. 9 Selected ports in schematic

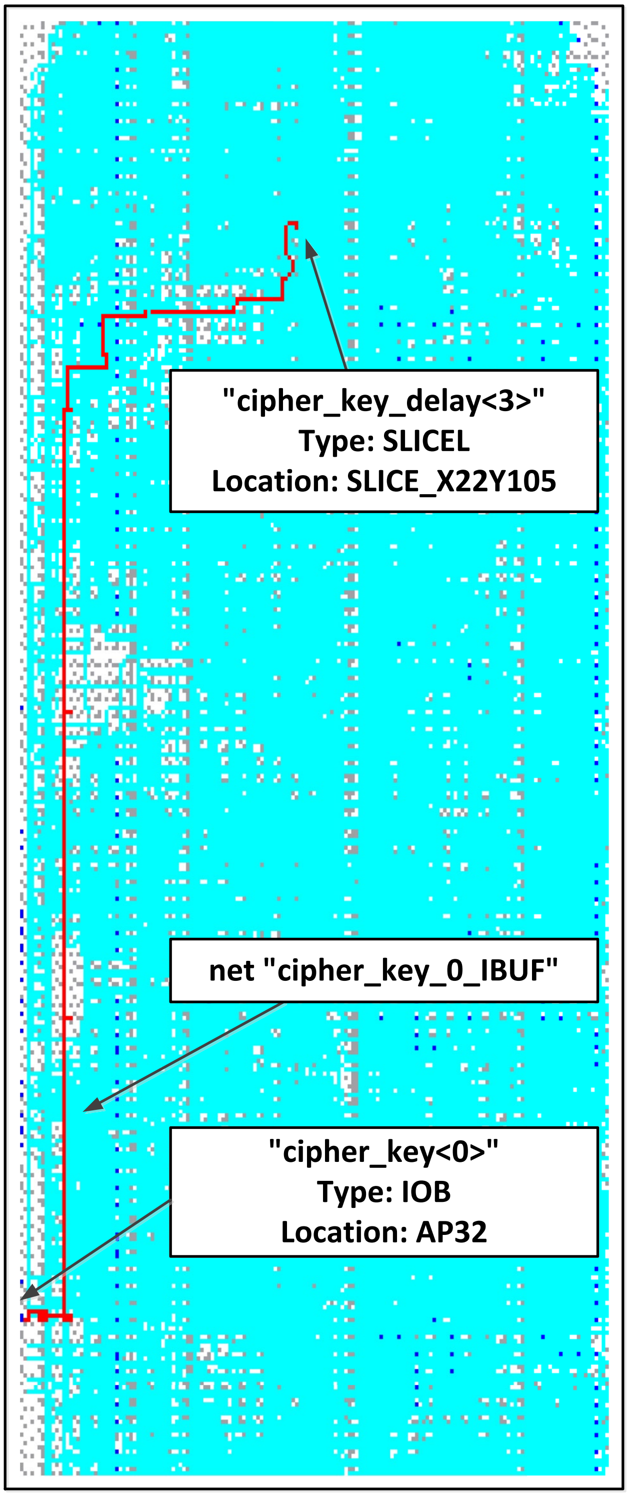


Fig. 10 Trojan free NCD file

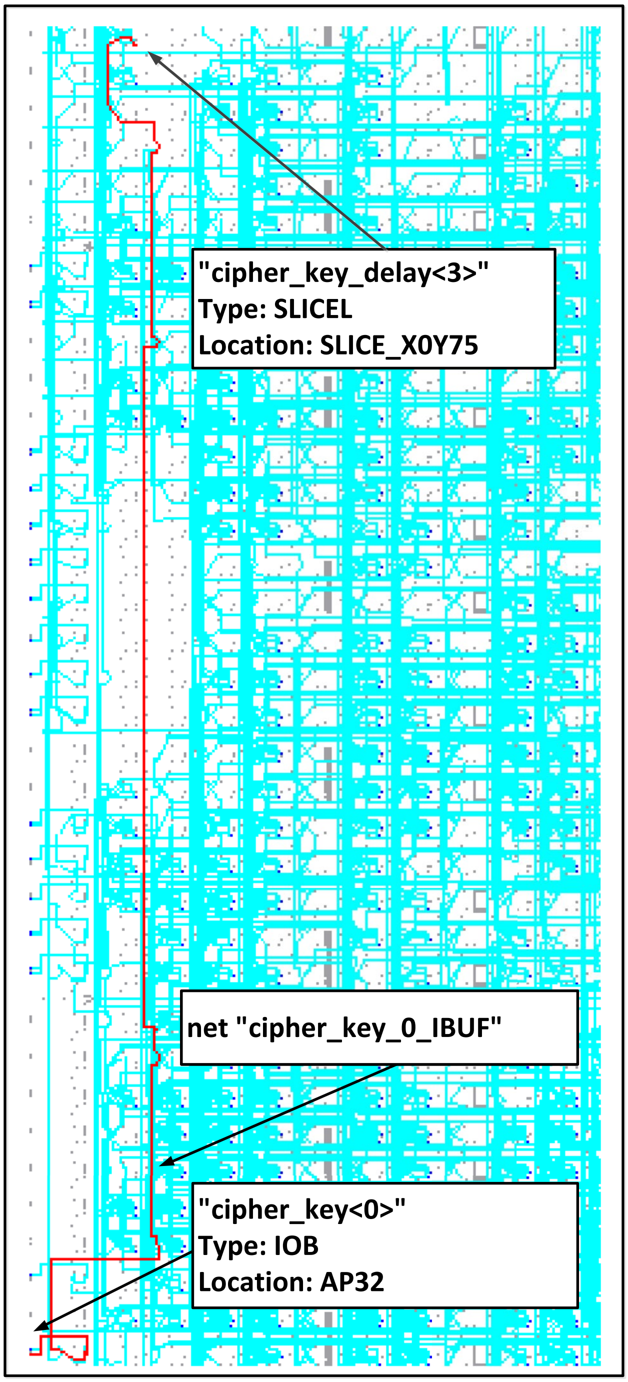
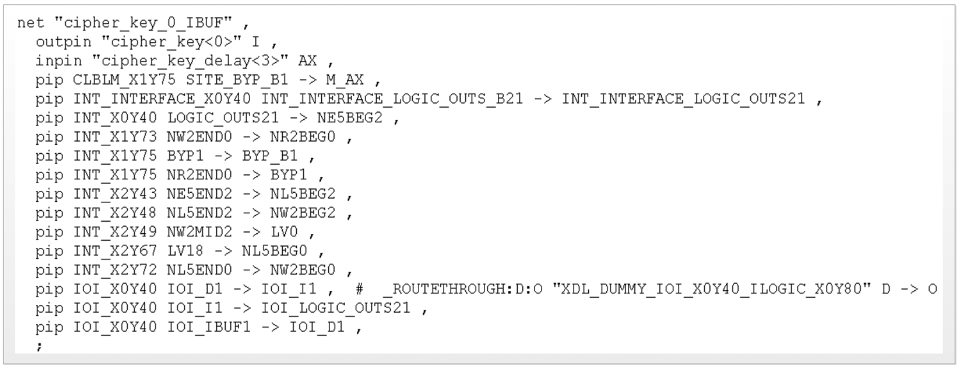


Fig. 11 Malicious NCD file

information. While net cipher_key_0_IBUF is already connected to the input port cipher_key[0]. These changes are made for creating a recovered XDL/NCD dataset from the infected bitstream. The following output is generated by the "ReportGen" command:

```
cipher_key_0_IBUF
cipher_key<0>.I
3.354 cipher_key_delay<3>.AX
2.730 my_pin.O
```



```
net "cipher_key_0_IBUF" ,
  outpin "cipher_key<0>" I ,
  inpin "cipher_key_delay<3>" AX ,
  pip CLBLM_X1Y75 SITE_BYP_B1 -> M_AX ,
  pip INT_INTERFACE_X0Y40 INT_INTERFACE_LOGIC_OUTS_B21 -> INT_INTERFACE_LOGIC_OUTS21 ,
  pip INT_X0Y40 LOGIC_OUTS21 -> NE5BEG2 ,
  pip INT_X1Y73 NW2END0 -> NR2BEG0 ,
  pip INT_X1Y75 BYP1 -> BYP_B1 ,
  pip INT_X1Y75 NR2END0 -> BYP1 ,
  pip INT_X2Y43 NE5END2 -> NL5BEG2 ,
  pip INT_X2Y48 NL5END2 -> NW2BEG2 ,
  pip INT_X2Y49 NW2MID2 -> LV0 ,
  pip INT_X2Y67 LV18 -> NL5BEG0 ,
  pip INT_X2Y72 NL5END0 -> NW2BEG0 ,
  pip IOI_X0Y40 IOI_D1 -> IOI_I1 ,  #  _ROUTETHROUGH:D:O "XDL_DUMMY_IOI_X0Y40_ILOGIC_X0Y80" D -> O
  pip IOI_X0Y40 IOI_I1 -> IOI_LOGIC_OUTS21 ,
  pip IOI_X0Y40 IOI_IBUF1 -> IOI_D1 ,
  ;
```

Fig. 12 PIP details in the Trojan free XDL file

while PIN2UCF shows the following result:

```
NET ”my_pin.OUTBUF.OUT” LOC = AD25;
```

Further, the XDL command is used to get complete detail of the logic associated with the malicious port. After applying the proposed model these changes are detected by TL modeling.

4.7 Detection of Trojans That Induce Power Fluctuations

The Trojans can be implemented by simultaneously switching the signals that use the CLB's interconnect resources. Such switching signals are used to connect to unused PIPs that can be detected using the XDL command. Moreover, by increasing fan-outs or adding extra net delays, there will be more power consumption drawn by the circuit. So, the infected points where these Trojans' interconnections were present are detected by using the XPWR command.

4.7.1 XPWR

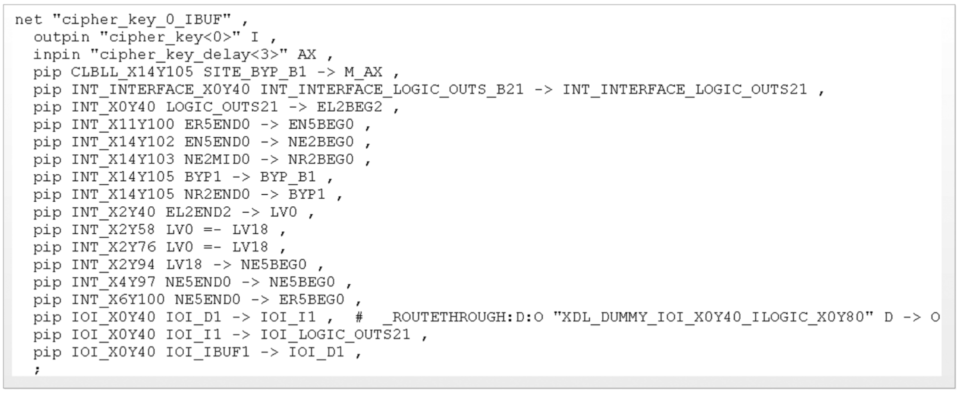
XPWR provides power as well as thermal estimates for FPGA designs. The file generated by the XPWR command gives information about the power of each net or logic element in the design. It also provides the status of junction temperature and a complete on-chip power summary. This information can be acquired by the following tool:

`xpwr -v infile[.ncd] original_pcf_file[.pcf]` //.xpwr file generation

The *original_pcf_file[.pcf]* is the Physical constraint file(PCF) of Trojan free design. To test the proposed scheme, the fanout for the signal cipher_key_0_IBUF is increased from 1 to 129. In doing this, the power for that signal is changed from 0.01 mW to 0.20 mW which is observed after applying the proposed framework to generated files. XPWR can be used to detect Type-II Trojans with 100% accuracy. For Type-I, accuracy varies depending upon resource utilization.

4.8 Bit file Generation

Based on the TL-generated reports, malicious changes are removed in infected XDL/NCD files and then Trojan free Bit file is generated using the Xilinx tool BitGen. BitGen takes an NCD file as input and generates a configuration bitstream file as output with a ".bit" extension. So modified XDL file is converted into an NCD file and then the following command is used:



```
net "cipher_key_0_IBUF" ,
  outpin "cipher_key<0>" I ,
  inpin "cipher_key_delay<3>" AX ,
  pip CLBLL_X14Y105 SITE_BYP_B1 -> M_AX ,
  pip INT_INTERFACE_X0Y40 INT_INTERFACE_LOGIC_OUTS_B21 -> INT_INTERFACE_LOGIC_OUTS21 ,
  pip INT_X0Y40 LOGIC_OUTS21 -> EL2BEG2 ,
  pip INT_X11Y100 ER5END0 -> EN5BEG0 ,
  pip INT_X14Y102 EN5END0 -> NE2BEG0 ,
  pip INT_X14Y103 NE2MID0 -> NR2BEG0 ,
  pip INT_X14Y105 BYP1 -> BYP_B1 ,
  pip INT_X14Y105 NR2END0 -> BYP1 ,
  pip INT_X2Y40 EL2END2 -> LV0 ,
  pip INT_X2Y58 LV0 =- LV18 ,
  pip INT_X2Y76 LV0 =- LV18 ,
  pip INT_X2Y94 LV18 -> NE5BEG0 ,
  pip INT_X4Y97 NE5END0 -> NE5BEG0 ,
  pip INT_X6Y100 NE5END0 -> ER5BEG0 ,
  pip IOI_X0Y40 IOI_D1 -> IOI_I1 ,  #  _ROUTETHROUGH:D:O "XDL_DUMMY_IOI_X0Y40_ILOGIC_X0Y80" D -> O
  pip IOI_X0Y40 IOI_I1 -> IOI_LOGIC_OUTS21 ,
  pip IOI_X0Y40 IOI_IBUF1 -> IOI_D1 ,
  ;
```

Fig. 13 PIP details in the Malicious XDL file

`bitgen -w infile[.ncd]`//.bit file generation

5 Countermeasures

First of all, it is strongly recommended to use bitstream encryption rather than using an unencrypted plain bitstream. Some successful attacks have also been launched against bitstream encryption in different XILINX FPGAs [11]. However, encrypting the bitstream will create an obstacle, and a huge time will require for an attacker to decrypt or alter the encrypted bitstream. Some techniques like validating the design before use, patchable encryption, model checker, Revision Select (RS) pins (used to clear the battery-backed RAM (BBRAM) key storage and reset the FPGA), and obfuscation are some effective countermeasures against bitstream encryption attacks [11]. Moreover, key-based bitstream obfuscation techniques can also be used to generate logically varying bitstreams for the same FPGA architecture that provides good protection against major attacks [21].

Secondly, manually re-route the design and set a proper slice position on the FPGA's die instead of using default settings. Unused I/O pins must be grounded and fill the vacant resources with dummy logic [23] or replicate the same design [2]. These copies will be shuffled and activated at the designer's choice. In this way, the attacker will have a hard time deciding which one is the correct design, when will it activate, and where to place HTs. The built-in self-test (BIST) [20] should be placed on specific logic test points and designers must work on real-time monitoring techniques [4]. For measuring physical operating parameters of FPGAs like on-chip power supply voltages and die temperatures Xilinx "System Monitor wizard" IP core is very effective and helpful in real-time monitoring. It is suggested that design/netlist files to bitstream conversion must be customized in a way that soft logic-locking techniques along with encryption must be incorporated within the design tools. Finally, there must be some additional features like "bitstream reversing to netlist files" and "one-click efficient dummy logic insertion on vacant places" in FPGA's design suites. It will not only be helpful for researchers to detect malicious modifications in the design but beneficial for practitioners and system developers as well.

6 Conclusion and Future Directions

In this study, the major challenges in HTs detection are addressed by introducing a novel "KZ Framework" that utilizes Xilinx command-line tools and NLP techniques. In this work, the detection of malicious modifications/HTs is demonstrated by extracting complete information on hardware primitives, resource utilization, timing constraints, and power summaries from XDL/NCD files in textual form. The XDL or NCD files can be extracted from the infected bitstream by applying reverse engineering techniques. Further, NLP is employed to extract the syntactic features from the descriptive artifact to find the malicious modifications/HTs. The aforementioned NLP method is generic and can be employed on files extracted from different FPGAs, while netlist to textual file conversion will require some tools like the Xilinx Command line.

The proposed work is not restricted to only one type of data matching. The reason for converting bitstream to textual/readable data is to find the types of detected HTs. Furthermore, the reference files are only required to train the model. Once the model is trained, these files are no longer required. Further, the TL function has been employed for being the least hungry for data and most prudent for such problems. The proposed method could pave the way to learning and studying the behavior of different types of HTs and malicious intentions. The KZ framework shows different detection methods and each method targets the particular type of HTs rather than finding out generic malicious changes. Moreover, no additional equipment is required for the simple, time-efficient, and non-invasive detection techniques that make it cost-effective, manageable, and easily operated. Further, experimental results show that the proposed technique can detect HTs with excellent accuracy.

Finally, for future research, such type of detection model should be incorporated in FPGA-based design suites that will ease the developers and researcher to find out malicious changes. This will lead to studying and implementing the best methods of detection to ensure devices that are free of HTs.

Acknowledgements I acknowledge my mentor Dr. Mureed Hussain for his motivation, guidance, and valuable revision of this work. I also acknowledge Dr. M. Rizwan for his valuable suggestions.

Data Availability The data that support the findings of this study are subject to third-party restrictions and so are not publicly available.

Declarations

Conflicts of Interest The author has no competing interests to declare that are relevant to the content of this article.

References

1. Advanced Encryption Standard (AES) (2001). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
2. Beaumont M, Hopkins B, Newby T (2011) Hardware Trojans-prevention, detection, countermeasures (a literature review). Technical report, Defence Science and Technology Organisation Edinburgh (Australia) Command
3. Benz F, Seffrin A, Huss SA (2012) Bil: A tool-chain for bitstream reverse-engineering. In: Proc. 22nd International Conference on Field Programmable Logic and Applications (FPL). IEEE, pp 735–738

4. Bhunia S, Hsiao MS, Banga M, Narasimhan S (2014) Hardware Trojan attacks: threat analysis and countermeasures. Proc IEEE 102(8):1229–1247
5. Chakraborty RS, Saha I, Palchaudhuri A, Naik GK (2013) Hardware Trojan insertion by direct modification of FPGA configuration bitstream. IEEE Design & Test 30(2):45–54
6. Chechik G, Sharma V, Shalit U, Bengio S (2010) Large scale online learning of image similarity through ranking. J Mach Learn Res 11(3)
7. Cheremisinov DI (2013) Design automation tool to generate EDIF and VHDL descriptions of circuit by extraction of FPGA configuration. In: Proc. East-West Design & Test Symposium (EWDTS). IEEE, pp 1–4
8. Couch JD (2011) Applications of TORC: an open toolkit for reconfigurable computing. PhD thesis, Virginia Tech
9. Ding Z, Qiang W, Zhang Y, Zhu L (2013) Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation. Microprocessors and Microsystems 37(3):299–312
10. Drimer S (2008) Volatile FPGA design security–a survey. IEEE Computer Society Annual Volume. pp 292–297
11. Ender M, Moradi A, and Christof Paar (2020) The unpatchable silicon: a full break of the bitstream encryption of Xilinx 7-Series FPGAs. In: Proc. 29th {USENIX} Security Symposium ({USENIX} Security 20)
12. Ender M, Swierczynski P, Wallat S, Wilhelm M, Knopp PM, Paar C (2019) Insights into the mind of a Trojan designer: the challenge to integrate a Trojan into the bitstream. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference. pp 112–119
13. Fyrbiak M, Wallat S, Swierczynski P, Hoffmann M, Hoppach S, Wilhelm M, Weidlich T, Tessier R, Paar C (2018) Hal–the missing piece of the puzzle for hardware reverse engineering, Trojan detection and insertion. IEEE Trans Dependable Secure Comput 16(3):498–510
14. Gören S, Ozkurt O, Yildiz A, Ugurdag HF, Chakraborty RS, Mukhopadhyay D (2013) Partial bitstream protection for low-cost FPGAs with physical unclonable function, obfuscation, and dynamic partial self reconfiguration. Comput Electr Eng 39(2):386–397
15. Guo X, Dutta RG, Jin Y, Farahmandi F, Mishra P (2015) Pre-silicon security verification and validation: A formal perspective. In: Proceedings of the 52nd Annual Design Automation Conference. pp 1–6
16. Hicks M, Finnicum M, King ST, Martin MMK, Smith JM (2010) Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In: Proc. IEEE Symposium on Security and Privacy. IEEE, pp 159–172
17. Hirschberg J, Manning CD (2015) Advances in natural language processing. Science 349(6245):261–266
18. Hoyoung Y, Lee H, Lee S, Kim Y, Lee H-M (2018) Recent advances in FPGA reverse engineering. Electronics 7(10):246
19. Iwase T, Nozaki Y, Yoshikawa M, Kumaki T (2015) Detection technique for hardware Trojans using machine learning in frequency domain. In: Proc. IEEE 4th Global Conference on Consumer Electronics (GCCE). IEEE, pp 185–186
20. Jyothi V, Rajendran JJV (2018) Hardware Trojan attacks in FPGA and protection approaches. In: The Hardware Trojan War. Springer, pp 345–368
21. Karam R, Hoque T, Ray S, Tehranipoor M, Bhunia S (2016) Robust bitstream protection in FPGA-based systems through low-overhead obfuscation. In: Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, pp 1–8
22. Karri R, Rajendran J, Rosenfeld K, Tehranipoor M (2010) Trustworthy hardware: Identifying and classifying hardware Trojans. Computer 43(10):39–46
23. Khaleghi B, Ahari A, Asadi H, Bayat-Sarmadi S (2015) FPGA-based protection scheme against hardware Trojan horse insertion using dummy logic. IEEE Embed Syst Lett 7(2):46–50
24. Lavin C, Padilla M, Lundrigan P, Nelson B, Hutchings B (2010) Rapid prototyping tools for FPGA designs: Rapidsmith. In: Proc. International Conference on Field-Programmable Technology. IEEE, pp 353–356
25. Moradi A, Barenghi A, Kasper T, Paar C (2011) On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs. In: Proceedings of the 18th ACM Conference on Computer and communications security. pp 111–124
26. Note J-B, Rannaud É (2008) From the bitstream to the netlist. In: FPGA, vol 8. pp 264
27. Nguyen J-F (2016) Analysing the bitstream of Altera's MAX-V CPLDS
28. Pham KD, Horta E, Koch D (2017) Bitman: a tool and API for FPGA bitstream manipulations. In: Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, pp 894–897
29. Rai D, Lach J (2009) Performance of delay-based Trojan detection techniques under parameter variations. In: Proc. IEEE International Workshop on Hardware-Oriented Security and Trust. IEEE, pp 58–65
30. Raghavan AK, Sutton P (2002) JPG-a partial bitstream generation tool to support partial reconfiguration in virtex FPGAs. In: Proc. Parallel and Distributed Processing Symposium, International, vol 2. Citeseer, p 0155
31. Salmani H (2016) Cotd: Reference-free hardware Trojan detection and recovery based on controllability and observability in gate-level netlist. IEEE Trans Inf Forensics Secur 12(2):338–350
32. Salmani H, Tehranipoor M, Karri R (2013) On design vulnerability analysis and trust benchmarks development. In: Proc. IEEE 31st International Conference on Computer Design (ICCD). IEEE, pp 471–474
33. Schroff F, Kalenichenko D, Philbin J (2015) Facenet: A unified embedding for face recognition and clustering. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp 815–823
34. Schütze H, Manning CD, Raghavan P (2008) Introduction to information retrieval, vol 39. Cambridge University Press Cambridge
35. SymbiFlow (2017) Project x-ray. https://github.com/SymbiFlow/prjxray
36. Steiner N, Wood A, Shojaei H, Couch J, Athanas P, French M (2011) Torc: towards an open-source tool flow. In: Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays. pp 41–44
37. Swierczynski P, Becker GT, Moradi A, Paar C (2017) Bitstream fault injections (BiFI)–automated fault attacks against SRAM-based FPGAs. IEEE Trans Comput 67(3):348–360
38. Tehranipoor M, Koushanfar F (2010) A survey of hardware Trojan taxonomy and detection. IEEE Des Test Comput 27(1):10–25
39. Tehranipoor M, Wang C (2011) Introduction to hardware security and trust. Springer Science & Business Media
40. Wolf C (2015) Project icestorm. http://www.clifford.at/icestorm/
41. Xilinx command line tools user guide (UG628) (2013). https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/devref.pdf
42. Xilinx Inc. (2022) https://www.xilinx.com/
43. Yoo HY, Choi SY, Park JW (2020) Reverse engineering for Xilinx FPGA chips using ISE design tools. J Integr Circuits Syst 6(1)
44. Yoon J, Seo Y, Jang J, Cho M, Kim J, Kim H, Kwon T (2018) A bitstream reverse engineering tool for FPGA hardware Trojan detection. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp 2318–2320
45. Zhang T, Wang J, Guo S, Chen Z (2019) A comprehensive FPGA reverse engineering tool-chain: From bitstream to RTL code. IEEE Access 7:38379–38389

46. Zhang J, Yuan F, Wei L, Liu Y, Qiang X (2015) Veritrust: Verification for hardware trust. IEEE Trans Comput Aided Des Integr Circuits Syst 34(7):1148–1161
47. Zhao M, Suh GE (2018) FPGA-based remote power side-channel attacks. In: Proc. IEEE Symposium on Security and Privacy (SP). IEEE, pp 229–244
48. Ziener D, Aßmus S, Teich J (2006) Identifying FPGA IP-cores based on lookup table content analysis. In: Proc. International Conference on Field Programmable Logic and Applications. IEEE, pp 1–6

Kamran Zahid received his Master's degree in Electrical Engineering from the National University of Computer and Emerging Sciences, Pakistan. His area of interest is digital system design, Verilog/VHDL programming, hardware security, and vulnerability analysis of embedded systems. He has completed various industrial and commercial projects related to the aforementioned fields. He also taught courses and conducted training workshops on FPGA design for embedded systems and Verilog programming in various institutions.