Turning the Table: Using Reverse Engineering Techniques to Detect FPGA Trojans

Wafi Danesh, Joshua Banago, Mostafizur Rahman
Computer Science Electrical Engineering
University of Missouri Kansas City, Kansas City, Missouri, USA
rahmanmo@umkc.edu

ABSTRACT

According to recent reports [1]-[3], FPGAs have increasingly become the prime targets for carrying out extremely sophisticated cyber-attacks. Reconfigurability, a core feature of FPGAs, which allow modular designs to be mapped, consequently makes them vulnerable, since it allows intruders to gain control of the hardware during run-time for Trojan insertion. In most cases, the attackers start with reverse engineering a reference hardware design before Trojan insertion. In this paper, we turn the tables and utilize bitstream reverse engineering for Trojan detection. Our approach is bottom-up: it starts with bitstream inspection, FPGA component identification from the layout and follows up with identification of malicious configurations. Due to this bottom-up approach, any stealthy malicious circuit can be detected, which is very hard to do through conventional design analysis methods [16] . We leverage open-source tools, publicly available datasets and vendor provided tools. The method is generic and can be applied to a wide range of FPGAs. An example of Trojan detection is shown for Xilinx Virtex5 VLX50T FPGA. Once the reference database is created, the Trojan detection is very fast, takes less than a minute on a general-purpose Intel Core i5 processor-based computer. The proposed Trojan detection approach is, to the best of our knowledge during publication, the only open-source, software-based, low-cost alternative to time consuming design re-construction and analysis based approaches. It can be applied at the customer end without requiring any hardware alteration/IP updates from FPGA vendors.

KEYWORDS

FPGA, Bitstream Reverse Engineering, Trojan Insertion, Bitstream Recovery, Trojan Detection

ACM Reference format:

FirstName Surname, FirstName Surname and FirstName Surname. 2018. Insert Your Title Here: Insert Subtitle Here. In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1. Introduction

By 2020, experts predict that there will be an estimated 50 billion connected devices [4], which amounts to approximately 6 connected devices per person on the planet. With this rapid increase in connectivity, cyber criminals have become more nefarious and cyberattacks grown more sophisticated. In particular, there has been a surge in new types of cyberattacks that target the network infrastructure such as high-speed routers, switches, firewalls and Intrusion Detection Systems (IDSs). FPGAs, whose core feature is reconfigurability, are a type of programmable logic device, widely used in network infrastructures to implement critical network functionality [5]-[12]. The very advantage of reconfigurability is consequently a curse for FPGAs, leaving them vulnerable to being exploited by cyber-attacks [1]-[3], [13]. In a recent case involving Cisco in 2019, a security firm exposed a security vulnerability in millions of Cisco routers, where an attacker can remotely gain root access to the router and reprogram the FPGA configuration bitstream to disable the Trust Anchor Module (TAm), which protects against boot-time exploits [1]. A few years back it was reported [14] that even military grade FPGAs (with readback disabled) can be compromised and back door methods can be found to insert malwares in FPGAs. In the literature [15]-[24], numerous authors report reverse engineering or Trojan insertions that compromise FPGA security.

A hardware Trojan is an electronic circuit that serves a shadow purpose besides its intended functionality and is usually triggered by rare input events. State-of-the-art techniques to detect hardware Trojans target detection of trigger circuits [16], [25]. However, it was shown in [25] that stealthy Trojans can be designed by reconfiguring the high-level design and bitstream that can avoid detection by well- known techniques. We propose an automated bitstream Trojan detection framework that can be applicable for modern FPGAs (irrespective of vendor, model) given the bitstream is not encrypted. Towards that objective our contribution is three-fold:

a) Generation of frame database for correlation mapping between bitstream and FPGA architectures

b) De-compiling of the bitstream to extract key components such as frame address, site location and configuration data

c) Correlation of the extracted data with the frame database for detailed layout (with or without Trojan) retrieval

For database creation, the placed and routed files from reference designs along with FPGA architectures from vendors are being used as inputs. Details of the static resources, routing and both programmable interconnects and logics are being considered. De-compiling of bitstreams is based on the information provided in datasheets and multiple synthesis runs to check what changes occur in bitstreams due to deterministic variations in design files. Once the database is complete with exhaustive references and reasonable confidence is built on bitstream De-compiling , any arbitrary bitstream can be interpreted back to the layout through the correlation approach. Once the layout is interpreted, any miniscule changes can be identified by comparison with the original layout (without Trojan) from the vendor. We demonstrate the effectiveness of our method by showing detection of stealth circuits

that are undetectable at HDL level shown in [25]. Our demonstration is with the widely used Xilinx Virtex 5 family FPGA (V5VLX50T). The rest of the paper is organized as follows: in Section 2, the background and methodology of our Trojan detection scheme is described in detail. In Section 3, we implement a hardware Trojan detection example using our methodology on a Xilinx FPGA. Conclusion and future works are presented in Section 4.

2. Background & Methodology

The overall method is shown in Fig. 1. The idea is to convert a bitstream (with or without Trojan) to a layout level representation and compare that with another file that is Trojan free. If the comparison reveals no new information, then the bitstream under

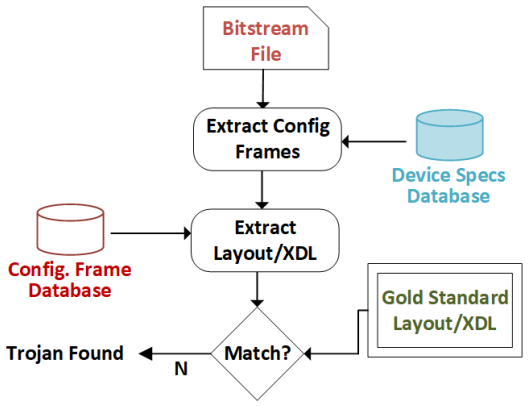

Figure 1. Bitstream reverse engineering based Trojan detection framework

test is Trojan free, and is infected otherwise. In case of infection, the layout file will also reveal the location of the Trojan within the FPGA fabric and its logic/configuration information. To convert a bitstream to an intermediate representation with layout level information, two databases and correlation of information between the bitstream frames and databases are required.

2.1 Frame database for correlation mapping between bitstream and FPGA architectures:

Extraction of information from any arbitrary bitstream requires information related to an FPGA design and architecture, namely (1) FPGA design architecture database and (2) bits to programmable logic and interconnect map database. We can collect FPGA architecture related information for databases from device specs. For Xilinx FPGAs, device specs can be collected from device-specific XDLRC files and through Xilinx's PARTGEN tool. For Altera FPGAs, such information can be obtained from RPF files. There are several open-source partial reconfiguration platforms such as Rapidsmith [26], Torc [20], HAL [19]**Error! Reference source not found.**, BIL[27], BitstreamToNCD [28] that allow access to such a database. For our demonstration, we used Rapidsmith. A wide range of devices including Artix, Kintex, Spartan (2, 2e, 3, 3a, 3adsp, 3e), Virtex (e, 2, 2p, 4, 5, 6, 7) and Zynq family devices (totaling over 200 devices) are supported in this platform. For later generation of FPGAs RapidSmith2 [31], RapidWright [32] provide similar interfaces with device specs. After the databases are prepared for device specs, the next step in the flow (Fig. 1) is to prepare configuration frame (which bit programs what logic as shown in Fig. 2) database, and extract configuration frames from a bitstream and map them to relative FPGA configuration architecture. An FPGA can be perceived as a rectangular grid of tiles with each tile containing configurable logic and interconnects. To map to these grids of tiles, there is a grid of memory [30] on to which configuration frames are mapped. A physical site maps to a unique subset of tiles, which denotes a rectangular area enclosed by surrounding wire channels. Each tile has a grid position, a unique name, a type, and may contain an arbitrary number of entities such as LUTs, MUXes, together they are called Programmable Logic Points (PLPs). For Xilinx FPGAs, switch matrix is called Programmable Interface Points (PIPs). PIPs control configurable routing within and in-between tiles. Apart from PIPs, there are static wires that form internal connections. After configuration frames are extracted, we map them to PLPs and PIPs using reference databases. Once the configuration points are recovered, we will create an intermediate representation of the design that was implemented in the bitstream and map it to a virtual FPGA device. That intermediate representation can then be transformed to XDL file, which can be easily converted to schematic and HDL files using Xilinx's BITGEN and NETGEN utilities.

The key information needed from bitstream analysis are the configuration data and location (which tile, PIP, etc. will be

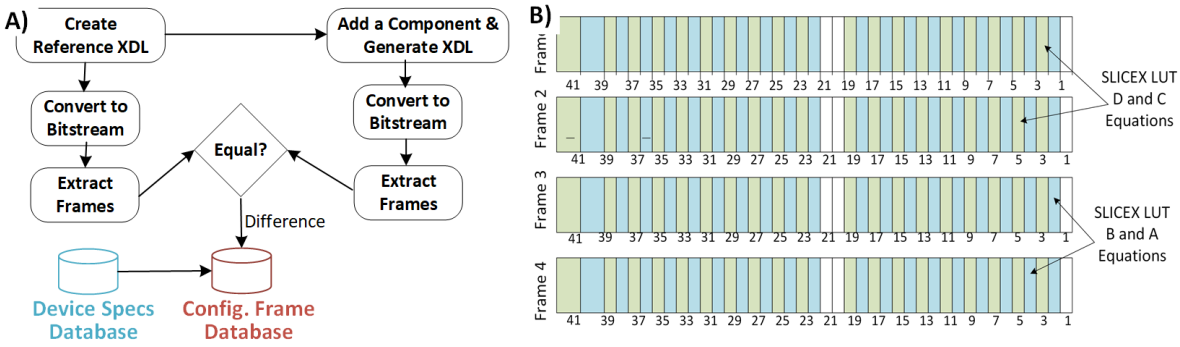

Figure 2. (A) Methodology for configuration frame database creation, (B) example of frame mapping to components

configured). A bitstream is a binary file composed of a series of words organized into "frames" [30]. A frame affects every block in a column of FPGA grid structure, and multiple horizontally adjacent frames are required to configure an entire column. Each frame is uniquely identified by an address and is the smallest addressable element. We have derived the composition of the frame address from the vendor datasheets [30]. The row address specifies which row in the grid will be written. The major address specifies the column within that row and the minor address identifies the frame number within the column. The BA bits specify the Random Access Memory, interconnect and/or logic type that will be programmed with configuration data. The frame address includes a Top indicator bit in position 20 that indicates whether the specified row is above or below the center of the device. Usually, only the first frame address is specified, and it is automatically incremented/decremented. The regions in the FPGA are separated based on clocks. Each clock region is given a row value in its address that increments away from the center of the device starting at 0. The row, column, frame organization information can be found from vendor's datasheet.

To identify meaningful application of the frames it is necessary to properly partition the configuration frame data into small segments to isolate the bits responsible for a specific PIP or element configuration. Our methodology for identifying frame configuration is shown in Fig. 3. We first create a reference design and then keep adding components to that reference design (by modifying the XDL file) and generate bitstreams. We use the XDL force method mentioned in [29]. Once the bitstreams are generated, we compute the difference between reference and new to identify meaningful application of those bits in FPGA. Through an iterative process, we build the database for all of the frames.

Extraction of CLB/PIP/IOB mapping information: We partition the data in a per tile style, as tiles are the smallest building blocks of the FPGA. Because tiles of the same type contain the same resources, we start with the assumption that they exhibit the same mapping of configuration bits. Using the lookup table generated during bitstream analysis, corresponding configuration data chunk can be fetched for tiles of all types. After the data fetch, the next task is to map the configuration data with components inside the tile and construct logic.

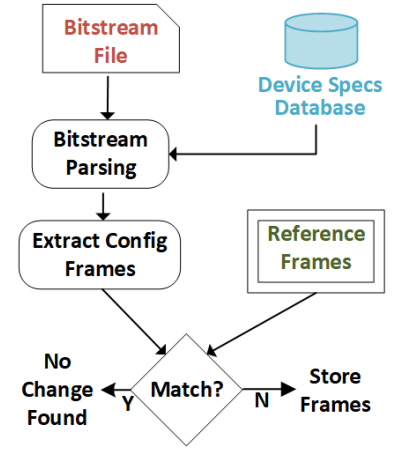

Figure 3. Extraction of configuration frames from the target bitstream

Following our proposed methodology, we were successful in recovery of the bitstream. We were able to decode logic information from bit frames. FPGAs typically constitute of Configuration Logic Blocks (CLBs), Block RAMs, Programmable Interconnect Points (PIPs), Interconnect Ports, and SoCs such as DSPs, Co-Processors, etc. At the heart of reconfigurability is the programming of CLBs to implement any arbitrary logic. CLB programming occurs through programming Look Up Tables (LUTs) inside CLBs. A Virtex 5 FPGA's CLB is divided into 2 Slices [30], and each Slice has four 64-bit LUTs. To program a logic (i.e., AND, OR, XOR, etc.), the 64 bits of a particular LUT's output needs to be programmed. To recover logic (whether the LUT is programmed with AND/OR/XOR, etc.), one needs to identify the location of the 64 bits that program a particular LUT (e.g., CLB1764, SliceL, LUTA) in sea of bits in the bitstream. Using the database that we developed, we were able to map bit frames to rows and columns in an FPGA fabric. Once the mapping was done, we used a bit correlation approach to recover the bits that program a particular LUT. In this correlation method, a particular LUT was programmed using an intermediate interface (i.e., XDL file) and the change was observed in corresponding bit frames.

Through correlation between expected LUT output bits and observed bits, we can infer the location of configuration bits for a particular LUT and the arrangement of bits to program the LUT. To prevent this inference, a security mechanism is added by Xilinx, which we were able to decode. Instead of expected output from AND operation (i.e., 0000 0000 0000 0001), we observed the output to be (1111 1111 1111 1111) and spread across 4 frames. The reason behind this disparity is because Xilinx stores a set of constant values in input registers on which the desired operation is performed and finally the resultant bits are stored. As a result of this manipulation, expected outputs are not observed. A challenge was first to determine the constant values, and then to correlate the outputs with the observed ones from bit frames. To recover the constants, we repeatedly fed inputs to outputs and observed changes. It is notable that the constants change for type of Slices and for device family. Using the methodology described, we can determine constants for almost all devices.

Using the constants and our methodology described earlier, we were able to create a map of LUTs, IOBs, and PIPs from bitstreams as shown in Figs. 2 and 4. This mapping reveals valuable information about bit correlation with FPGA components. Since the FPGA fabric consists of regular array of tiles, the same bit mapping information is relevant for all the CLB/INT/IOB tiles. Therefore, we can use this information to recover any arbitrary logic from any location in FPGA.

2.2 De-compiling bitstream to extract key information

After frame configuration database, our next step is to extract configuration information from the target bitstream. A typical bitstream file consists of header, authentication and data blocks [30]. To reverse the logic from bitstream, most important portion is the configuration data blocks, which are encapsulated in configuration data frames. To extract those frames, we need to first XOR bitstream frames with reference frames[26]. We first generate

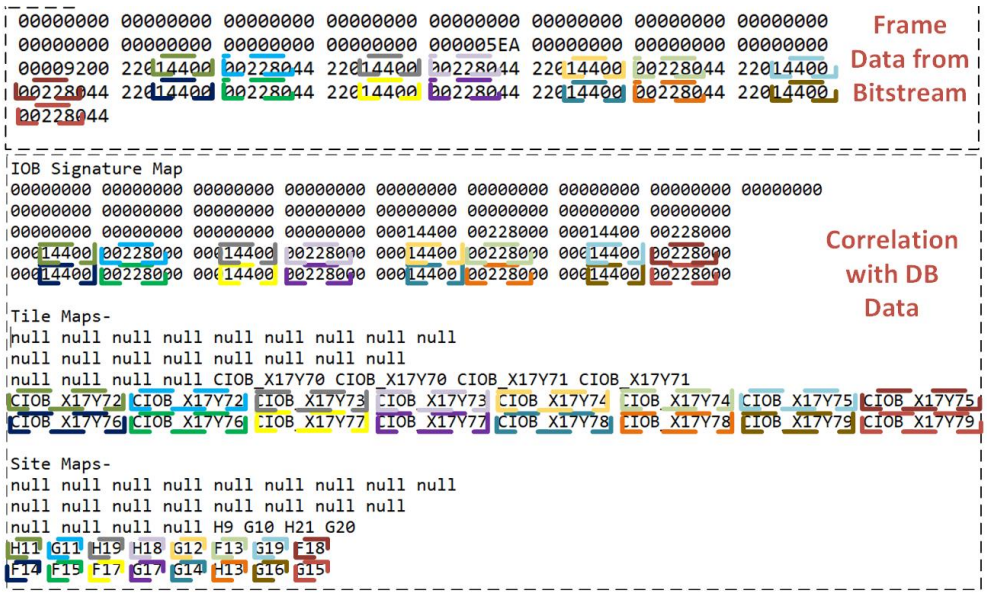

Figure. 4. Frame data from the target bitstream and corresponding results from our frame reference database is shown for a IOB frame. With different colors, matching of a frameword with a particular signature in database, corresponding site and tile names are shown.

the reference frames (with header, checksum and data information) for an empty design such that when we compare with the target bitstream, the comparison will reveal only the configuration frames from target bitstream. During the bitstream parsing process we associate the extracted frames with relevant FPGA to identify the frame address, column, row, minor and top/bottom information. Once the configuration frames are extracted by XORing with reference frames, the next task is to interpret the contents of those frames by associating the bits with configuration database that we generated in Section 2.1. An example of correlation between extracted frame data from target bitstream and the configuration database is shown in Fig. 4.

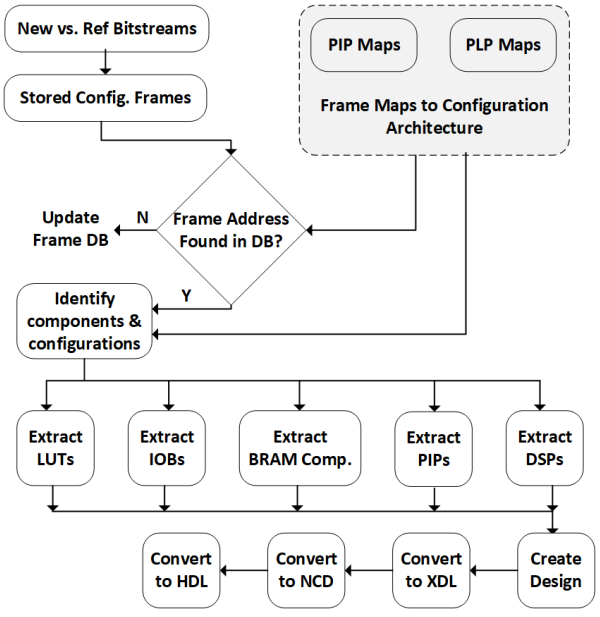

Figure 5. Correlation of configuration frames with information from database and netlist retrieval.

2.3 Correlation of extracted bits with configuration database and layout retrieval

The bitstream reversal process begins with database lookups. In the database we store all the information about frames, and the frame address works as the identifier. Once we find a matching frame, then we need to correlate bits within the frame with database information. A frame in V5VLX50T consists of 41 words with each word consisting of 32 bits. For a particular frame, the database contains mapping information for all 41x32 = 1312 bits. By direct correlation, we can determine tile names, site names, pip names, etc., and recover instances and interconnects within a design. For interconnect recovery, we first extract individual PIPs and then connect them from the source to destination. If the PIP connections emanate from an IOB, we will call that IOB a source, and stitch the interconnects to reach the destination. For Trojan detection though, full recovery of interconnects is not necessary as will be shown in Section 3. Upon recovery of all components through correlations with databases, an intermediate representation of the bitstream file is created which contains detailed layout and configuration information. We then use the layout file to retrieve the XDL file, which then can be used to generate NCD and HDL files (Fig. 5).

Using the proposed methodology, we were successful in recovering all CLB, IOB, BRAM and PIP information (ILogic, OLogic, IODelay) including configuration details, dummy instances for routing and port connections. Our interconnect stitching was 70% successful for a 16 NAND based trigger circuit. Since our objective was Trojan discovery, in most cases full interconnect recovery was not essential. Once the configuration database is generated, which

is a one time requirement, the bitstream reversal process is very fast. In most cases it took less than 1 minute in an average Intel Core i5 processor-based computer. To the best of our knowledge, this is the fastest bitstream recovery demonstration reported in comparison to bitstream reverse engineering [26], [20], [19], [27], [28]. Key specs are listed below:

Tasks	Time	Comparison
Configuration database generation	72-120 hours*	
Bitstream extraction & correlation to generate intermediate layout file	30-60seconds*	[26], [20], [19], [27], [28]/ 40-72hrs

*Software platform: Eclipse IDE, Java, Xilinx ISE 14.5
*Opensource device specs & partial programming: Rapidsmith
*Hardware platform: Intel i5, 8GB RAM, general purpose computer

3. Trojan Detection Example

To demonstrate effectiveness of our method to detect Trojan, we use here an example of a decoder. In our case, the decoder behaves as a normal decoder when there is no Trojan present, and when there is a Trojan, for certain input combinations it outputs 0 regardless of the desired output. To insert Trojan, we use an LUT with additional circuitry. The Trojan model is similar to [25], where a malicious LUT was inserted to showcase shadow behavior and incorrect operation. The Trojan insertion and activation in [25] was a two-step process where a malicious front-end first created a design with additional circuitry (i.e., LUT and a decode logic) to be hidden during placement, routing and timing checks, and a malicious back-end then replaced the bitstream with another synthesized bitstream with similar characteristics to trigger the Trojan. To emulate the same behavior, we synthesized and generated bitstreams for both benign decoder and Trojan infected decoder shown in Fig 6A & B. Since our intention is to identify differences between benign and malicious bitstreams, two step Trojan insertion and activation is irrelevant for our case and we can only work with the final bitstreams.

Output XDL snippets after reverse engineering both benign and infected decoders are shown in Fig. 6A &B (bottom). Only the differences are shown. The XDL file shows lowest level physical layout results; the tiles and interconnects that are configured in the FPGA fabric are shown in detail in the XDL file. In the FPGA fabric, there are columns designated for I/O pads, CLBs (Configuration Logic Blocks), BRAM, DSPs, and BRAMInterconnects. Inside each column, there are tiles which are further divided into sites. After synthesis, each I/O pad is given a specific tile and site address, and their configuration (input/output). In Fig. 6A (bottom) CLBLL_X16Y77 implies L type CLB in the 16th column and 77th row of the fabric. Within that particular CLB, the Slice X27Y77 was configured with a logic which is shown next to LUT O6 for the circuit above. The decoder circuit had 8 inputs and 1 output. The inputs and outputs are all assigned to IOBs in the FPGA fabric. In order to connect to those IOBs, dummy configuration blocks are used which are defined as OLogic (for outputs) and ILogic (for inputs). It is noticeable from Fig. 6 that due to the insertion of malicious logic, the original equation in SLICE_X27Y77 was changed and it also had an effect on the routing; the Ologic places in IOI_X17Y79 was removed in the malicious bitstream. Our tool was successful in reversing the bitstreams and identifying each component within the bitstreams.

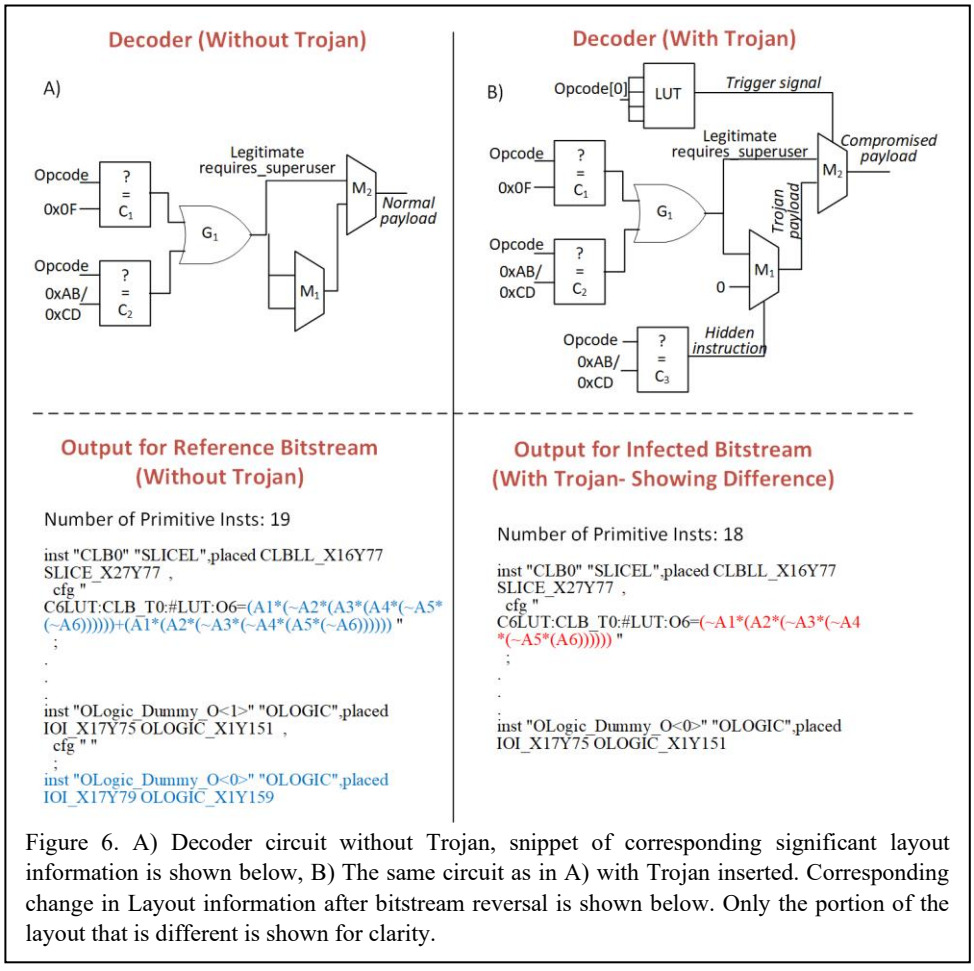

Figure 6. A) Decoder circuit without Trojan, snippet of corresponding significant layout information is shown below, B) The same circuit as in A) with Trojan inserted. Corresponding change in Layout information after bitstream reversal is shown below. Only the portion of the layout that is different is shown for clarity.

From those identifications, it is clear that the Trojan is located in the CLBLL_X16Y77 location under SLICE_X27Y77 and the changed logic equation for the output LUT is (~A1*(A2*(~A3*(~A4*(~A5*(A6)))))) where A1 to A6 are inputs to that SLICE. In Fig. 6B (bottom), CLB recovery, we were also able to pinpoint whether A/B/C/D type LUT is configured. This bitstream reversal process is completely automated regardless of the bitstream size and type.

4. Conclusion & Future Work

This paper introduces a new method for Trojan detection in FPGA bitstreams. The reverse engineering methods, which are normally considered adversarial and hackers' tools, are used in the positive context. Our framework for Trojan detection leverages open-source tools, publicly available information and vendor provided software. We show an example of Trojan insertion and detection through a decoder circuit. The bitstream extraction and Trojan detection process was very fast: it took ~30 seconds in a general-purpose Intel Core i5 processor-based computer. To our knowledge, this is the fastest bitstream information extraction demonstration and the first usage of reverse engineering of FPGA bitstreams for Trojan detection. This research is meant to open new ways for low cost Trojan detection and removal at the customer end without requiring any new hardware or IP changes. Its successful implementation can significantly enhance security for reconfigurable hardware. Our future work will explore software-hardware approaches for Trojan detection and removal at run-time in a networked environment with many connected nodes to prevent man-in-the-middle and denial-of-service type attacks due to reconfigurable router infiltrations.

REFERENCES

[1] Kataria, J., Housley, R., Pantoga, J., & Cui, A. (2019). Defeating Cisco Trust Anchor: A Case-Study of Recent Advancements in Direct {FPGA} Bitstream Manipulation. In *13th {USENIX} Workshop on Offensive Technologies ({WOOT} 19)*.

[2] Govindan, V., Koteshwara, S., Das, A., Parhi, K. K., & Chakraborty, R. S. (2019, December). ProTro: A Probabilistic Counter Based Hardware Trojan Attack on FPGA Based MACSec Enabled Ethernet Switch. In *International Conference on Security, Privacy, and Applied Cryptography Engineering* (pp. 159-175). Springer, Cham.

[3] Krautter, J., Gnad, D. R., & Tahoori, M. B. (2018). FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 44-68.

[4] Lee, B., Amaresh, S., Green, C., & Engels, D. (2018). Comparative study of deep learning models for network intrusion detection. SMU Data Science Review, 1(1), 8.

[5] Yazdinejad, A., Parizi, R. M., Bohlooli, A., Dehghantanha, A., & Choo, K. K. R. (2020). A high-performance framework for a network programmable packet processor using P4 and FPGA. *Journal of Network and Computer Applications*, *156*, 102564.

[6] Ricart-Sanchez, R., Malagon, P., Salva-Garcia, P., Perez, E. C., Wang, Q., & Calero, J. M. A. (2018). Towards an FPGA-Accelerated programmable data path for edge-to-core communications in 5G networks. *Journal of Network and Computer Applications*, *124*, 80-93.

[7] Bonati, L., Polese, M., D'Oro, S., Basagni, S., & Melodia, T. (2020). Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead. *arXiv preprint arXiv:2005.10027*.

[8] De Villiers, D. B. B. (2020). *FPGA implementation of a network coding capable switch* (Doctoral dissertation).

[9] Kiat, W. P., Mok, K. M., Lee, W. K., Goh, H. G., & Achar, R. (2020). An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications. *Microprocessors and Microsystems*, *73*, 102966.

[10] Wang, X., Niu, Y., Liu, F., & Xu, Z. (2020). When FPGA Meets Cloud: A First Look at Performance. *IEEE Transactions on Cloud Computing*.

[11] Caulfield, A., Costa, P., & Ghobadi, M. (2018, June). Beyond SmartNICs: Towards a fully programmable cloud. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)* (pp. 1-6). IEEE.

[12] Li, J., Sun, Z., Yan, J., Yang, X., Jiang, Y., & Quan, W. (2020). DrawerPipe: A Reconfigurable Pipeline for Network Processing on FPGA-Based SmartNIC. *Electronics*, *9*(1), 59.

[13] Ender, M., Moradi, A., & Paar, C. (2020). The Unpatchable Silicon: A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*.

[14] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. In Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems, CHES'12, pages 23–40, Berlin, Heidelberg, 2012. Springer-Verlag

[15] Ender, M., Swierczynski, P., Wallat, S., Wilhelm, M., Knopp, P. M., & Paar, C. (2019, January). 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).

[16] Zhang, J., & Qu, G. (2019). Recent attacks and defenses on FPGA-based systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, *12*(3), 1-24.

[17] Aldaya, A. C., Sarmiento, A. J. C., & Sánchez-Solano, S. (2016). AES T-Box tampering attack. *Journal of Cryptographic Engineering*, *6*(1), 31-48.

[18] Chakraborty, R. S., Saha, I., Palchaudhuri, A., & Naik, G. K. (2013). Hardware Trojan insertion by direct modification of FPGA configuration bitstream. *IEEE Design & Test*, *30*(2), 45-54.

[19] Fyrbiak, M., Wallat, S., Swierczynski, P., Hoffmann, M., Hoppach, S., Wilhelm, M., ... & Paar, C. (2018). HAL—The missing piece of the puzzle for hardware reverse engineering, Trojan detection and insertion. *IEEE Transactions on Dependable and Secure Computing*, *16*(3), 498-510.

[20] Steiner, N., Wood, A., Shojaei, H., Couch, J., Athanas, P., & French, M. (2011, February). Torc: towards an open-source tool flow. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays* (pp. 41-44).

[21] Horta, E. L., Lockwood, J. W., & Kofuji, S. T. (2002, September). Using PARBIT to implement partial run-time reconfigurable systems. In *International Conference on Field Programmable Logic and Applications* (pp. 182-191). Springer, Berlin, Heidelberg.

[22] Moradi, A., Kasper, M., & Paar, C. (2012, February). Black-box side-channel attacks highlight the importance of countermeasures. In *Cryptographers' Track at the RSA Conference* (pp. 1-18). Springer, Berlin, Heidelberg.

[23] Moradi, A., & Schneider, T. (2016, April). Improved side-channel analysis attacks on Xilinx bitstream encryption of 5, 6, and 7 series. In *International Workshop on Constructive Side-Channel Analysis and Secure Design* (pp. 71-87). Springer, Cham.

[24] Wallat, S., Fyrbiak, M., Schlögel, M., & Paar, C. (2017, July). A look at the dark side of hardware reverse engineering-a case study. In *2017 IEEE 2nd International Verification and Security Workshop (IVSW)* (pp. 95-100). IEEE.

[25] Krieg, C., Wolf, C., & Jantsch, A. (2016, November). Malicious LUT: a stealthy FPGA trojan injected and triggered by the design flow. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (pp. 1-8). IEEE.

[26] Lavin, C., Padilla, M., Lundrigan, P., Nelson, B., & Hutchings, B. (2010, December). Rapid prototyping tools for FPGA designs: RapidSmith. In *2010 International Conference on Field-Programmable Technology* (pp. 353-356). IEEE.

[27] Benz, F., Seffrin, A., & Huss, S. A. (2012, August). Bil: A tool-chain for bitstream reverse-engineering. In *22nd International Conference on Field Programmable Logic and Applications (FPL)* (pp. 735-738). IEEE.

[28] Ding, Z., Wu, Q., Zhang, Y., & Zhu, L. (2013). Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation. *Microprocessors and Microsystems*, *37*(3), 299-312.

[29] Ender, M., Swierczynski, P., Wallat, S., Wilhelm, M., Knopp, P. M., & Paar, C. (2019, January). 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).

[30] Virtex-5 FPGA Configuration User Guide, V3.12, May 2017, Available at- https://www.xilinx.com/support/documentation/user_guides/ug191.pdf

[31] Nelson, B., Townsend, T., and Haroldsen, T., Rapidsmith2: A Library for Low-Level Manipulation of Vivado Designs at the Cell/BEL Level, Feb 23, 2018, available at- https://github.com/byuccl/RapidSmith2/blob/master/docs/TechReport/TechReport.pdf

[32] Lavin, C., and Kaviani, A., "RapidWright: Enabling Custom Crafted Implementations for FPGAs," *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Boulder, CO, 2018, pp. 133-140, doi: 10.1109/FCCM.2018.00030.