

Review

Recent Advances in FPGA Reverse Engineering

Hoyoung Yu ¹, Hansol Lee ², Sangil Lee ², Youngmin Kim ¹  and Hyung-Min Lee ^{2,*} 

¹ School of Computer and Information Engineering, Kwangwoon University, Seoul 01897, Korea; hoyung94@nate.com (H.Y.); youngmin@kw.ac.kr (Y.K.)

² School of Electrical Engineering, Korea University, Seoul 02841, Korea; godth8035@korea.ac.kr (H.L.); lsl3541@korea.ac.kr (S.L.)

* Correspondence: hyungmin@korea.ac.kr; Tel.: +82-2-3290-3219

Received: 18 September 2018; Accepted: 10 October 2018; Published: 12 October 2018



Abstract: In this paper, we review recent advances in reverse engineering with an emphasis on FPGA devices and experimentally verified advantages and limitations of reverse engineering tools. The paper first introduces essential components for programming Xilinx FPGAs (Xilinx, San Jose, CA, USA), such as Xilinx Design Language (XDL), XDL Report (XDLRC), and bitstream. Then, reverse engineering tools (Debit, BIL, and Bit2ncd), which extract the bitstream from the external memory to the FPGA and utilize it to recover the netlist, are reviewed, and their limitations are discussed. This paper also covers supplementary tools (RapidSmith) that can adjust the FPGA design flow to support reverse engineering. Finally, reverse engineering projects for non-Xilinx products, such as Lattice FPGAs (Icestorm) and Altera FPGAs (QUIP), are introduced to compare the reverse engineering capabilities by various commercial FPGA products.

Keywords: reverse engineering; FPGA; hardware security; bitstream; netlists; Xilinx

1. Introduction

Field programmable gate arrays (FPGAs) have been widely used for embedded systems in various applications, such as consumer electronics, medical devices, security systems, and defense industry applications. FPGAs allow agile development of the embedded system because their functions can be continuously updated [1,2]. As FPGAs occupy larger portions of the embedded system market, their hardware security becomes an increasingly important issue.

Reverse engineering is generally defined as a way to analyze an existing system and to design and create copies by identifying component and interrelationships [3–10]. The reverse engineering in the FPGA is a way of re-configuring the netlist inside the FPGA in various ways. The reverse engineering methods introduced in this paper improve vendor and user awareness of security issues and also prevent FPGAs from being reversed in advance.

One potential approach for FPGA reverse engineering is to utilize the bitstream from an external memory to the FPGA [4–6]. Although there are a variety of memory schemes for programming, such as programmable read only memory (PROM), anti-fuse, static random access memory (SRAM), and Flash, most FPGAs have used the SRAM because of its low price, re-programmability, and so on. However, since the SRAM is a volatile memory, the FPGA needs an additional non-volatile memory outside the FPGA to store the programming codes [2]. Figure 1 shows the block diagram of the Xilinx Virtex FPGA connected with the external PROM [11]. When the FPGA board is powered up, the bitstream stored in the PROM is downloaded to the FPGA. Then, the bitstream can be extracted on the downloading path for reverse engineering. Reverse engineering by extracting bitstream files from the external memory is a non-invasive approach that does not damage the FPGA. Therefore, this method can be widely used for criminal investigation, military defense, and security industry that should not damage the target

hardware. While FPGA companies have supported bitstream encryption for security, there have also been attempts to extract encryption keys in order to decrypt the bitstream [12]. It should be noted that these reverse engineering methods assumed that the bitstream is not encrypted.

Several research groups have attempted reverse engineering to recover the netlist from the extracted bitstream [4–6]. This paper reviews recent advances in FPGA reverse engineering, particularly those using Xilinx FPGA products that occupy a dominant portion in FPGA markets, and also discusses their performance and limitations through hands-on experiments. In addition, this paper also reviews several supplementary tools that can support the reverse engineering [13–15].

The rest of this paper focuses on detailed techniques for FPGA reverse engineering using the bitstream. Section 2 describes the Xilinx design flow, tools, and related files generated during compilation. Section 3 reviews several reverse engineering tools, targeting Xilinx products. Section 4 reviews the other reverse engineering projects that aim for Altera and Lattice FPGAs. Section 5 discusses advantages and limitations of reverse engineering tools, followed by concluding remarks in Section 6.

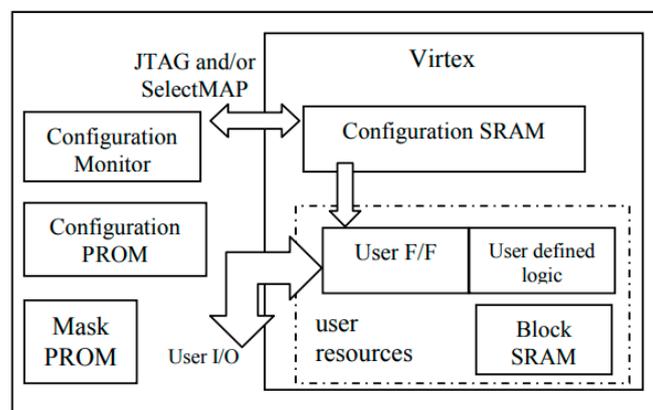


Figure 1. Block diagram of the Xilinx Virtex FPGA and its external PROM [11].

2. Xilinx FPGA Design

2.1. ISE Design Flow

Xilinx supports two integrated development software packages, ISE and Vivado, for FPGA design synthesis, simulation, and configuration. The ISE design suite supports the Spartan-6 and Virtex-6 devices as well as their previous generation families. Recently, Xilinx has recommended the Vivado design suite for new designs with Virtex-7, Kintex-7, and Artix-7. While Xilinx discontinued support for the ISE design tool in 2014, the ISE has continued to be widely used for FPGA programming because the Vivado mainly supports high-performance and high-end devices.

The conceptual diagram of the Xilinx ISE design flow is shown in Figure 2. First, the register-transfer level (RTL) design (Verilog, HDL file) is converted to the Xilinx internal netlist (NGD file) using `-ngd build`. Then, it performs technology mapping of the NGD file to circuit primitives (NCD file) using `-map`. The `-par` command places circuit primitives on the actual FPGA chip and connects each primitive in order to generate the placed and routed (P&R'd) NCD file. The fully P&R'd primitives (P&R'd NCD file) are then translated to the bitstream (BIT file) using `-bitgen`. The BIT file is converted to the MCS file and saved in the PROM. Finally, the stored bitstream (MCS file) in the PROM is downloaded to the FPGA for the purpose of programming when the FPGA board is powered up.

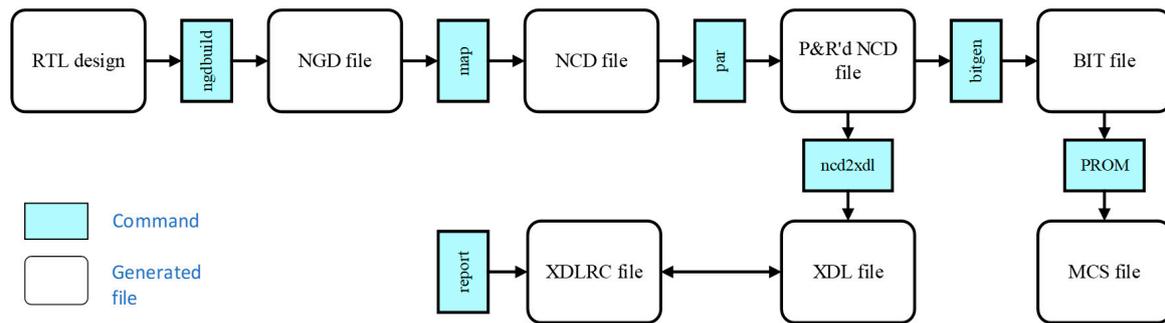


Figure 2. Conceptual diagram of the Xilinx ISE design flow.

The P&R'd NCD file can be also converted to the XDL file using `-ncd2xdl`. The XDL file is a text document that displays the netlist in a readable form. This XDL file can be interpreted through the XDLRC file, which describes all of the FPGA architecture. Both XDL and XDLRC files play important roles in bitstream-based reverse engineering, the concepts of which will be explained in detail in Sections 2.2 and 2.3.

2.2. XDLRC

The XDLRC is a text document detailing the Xilinx FPGA chip architecture that can interpret the XDL file. The XDLRC for each chip can be generated through the `-report` (chip name) of the XDL function provided by ISE. Figure 3 shows a graphical representation of the XDLRC structure, which is divided into tile units. Each site consists of primitives, such as slices. The elements contained within these primitives are described in detail in the primitive definition [3].

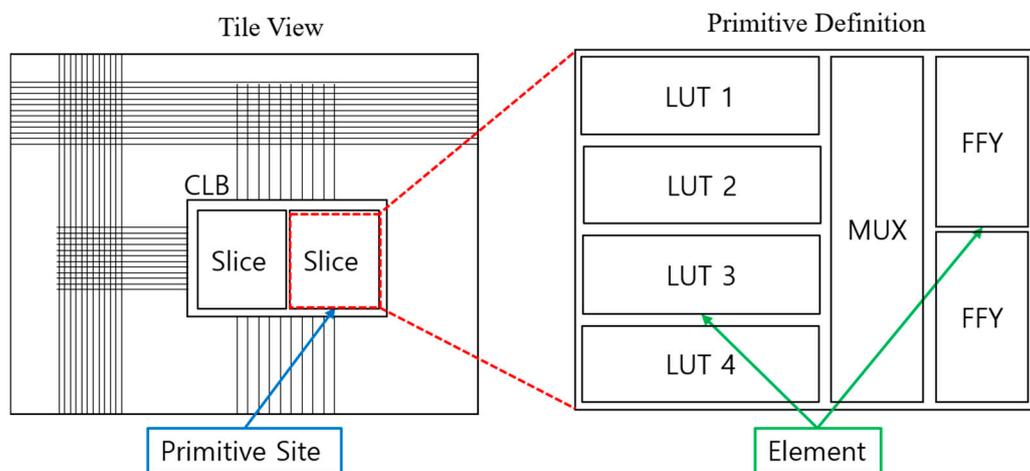


Figure 3. Graphical representation of the XDLRC structure.

An example code of the XDLRC text file is shown in Figure 4. The structure consists of four sections: First, the header section includes information such as the name of each chip, the size of the tile in the chip, etc. Second, the tile section describes the detailed position information of each tile, the types of the tiles (e.g., configurable logic block (CLB), interconnect (INT), input/output buffer (IOB), and digital signal processor (DSP)), the name of the element, and the location of the primitive present in the element. Third, the primitive definitions section describes the elements in detail. Finally, the summary section summarizes the information of the tiles and sites. Since the XDLRC file differs slightly from chip to chip, it must be well understood in order to fully explain the XDL file. Since the XDLRC can construct a mapping table by correlating with the bitstream, the XDLRC files have been utilized for various reverse engineering works.

```

(xdl_resource_report v0.2 xc5vlx30ff324-3 virtex5
(tiles 89 97
-----
(tile 52 76 INT_X29Y32 INT 1
  (primitive_site TIEOFF_X29Y32 TIEOFF internal 3)
(tile 46 67 CLBLL_X25Y38 CLBLL 2
  (primitive_site SLICE_X42Y38 SLICEL internal 45)
-----
(primitive_defs 53
  (primitive_def BSCAN 8 10
    (pin CAPTURE CAPTURE output)
    (element TDI 1
      (pin TDI input)
      (conn TDI TDI <== BSCAN TDI)
    )
-----
(summary tiles=8633 sites=9632 sitedefs=53)

```

Header

Tile section

Primitive definition part

Summary

Figure 4. Example code of the XDLRC file.

2.3. XDL

The XDL file is a textual representation of the netlist; it describes the internal state of the FPGA and is equivalent to the NCD file. While the XDL format is not officially documented, it can be easily understood by using the XDLRC file [4]. The XDL file can be created from the NCD file with the `xdl-ncd2xdl` option, and an XDL file can be reverted back to the NCD file with the `-xdl2ncd` option.

For example, the slice statement **FXMUX:: F** in the XDL file means that the **F** is the output of the **FXMUX** logic. In this way, the instance is created by connecting one logic to the other inside a block, and the point connecting these logics is defined as a programmable logic point (PLP). The wires connecting created instances to one another are considered to be nets. A user-controllable net is defined as a program interconnect point (PIP). In general, the PIP in the FPGA is called the switch box.

For example, the PIP statement, **pip INT_X0Y74 SR5END2 -> EL5BEG2** in the XDL file, indicates that the **SR5END2** wire in the INT block located in X0Y74 is connected to the **EL5BEG2** wire. Since this XDL file is equivalent to the NCD file, the netlist information on the target FPGA can be obtained by extracting the XDL file from the bitstream.

2.4. Bitstream Structure

Virtex family devices in Xilinx FPGA chips are divided into top and bottom parts, as shown in Figure 5. The rows are then further divided into columns and made up of frames, e.g., the highlighted region in yellow, which are the smallest spatial unit [16]. The bitstream has a unique frame address: frame (0–6), column (7–14), row (15–19), top/bottom (20), and block (21–23). The different devices have different frame column widths of each tile, and the bitstream column width of the Virtex-5 family is summarized in Table 1. There are no separate frame column widths for the INT tile with nets, because it is not possible to individually address the bitstream for INT tiles. Therefore, INT information for each tile is included within the tile itself, and the bitstream can be generated by arranging column sets of the tiles side by side.

Table 1. Bitstream column width of the Virtex-5 devices [16].

Column Type	Width in Frames
CLB	36
DSP	28
Block RAM	30
IOB	54

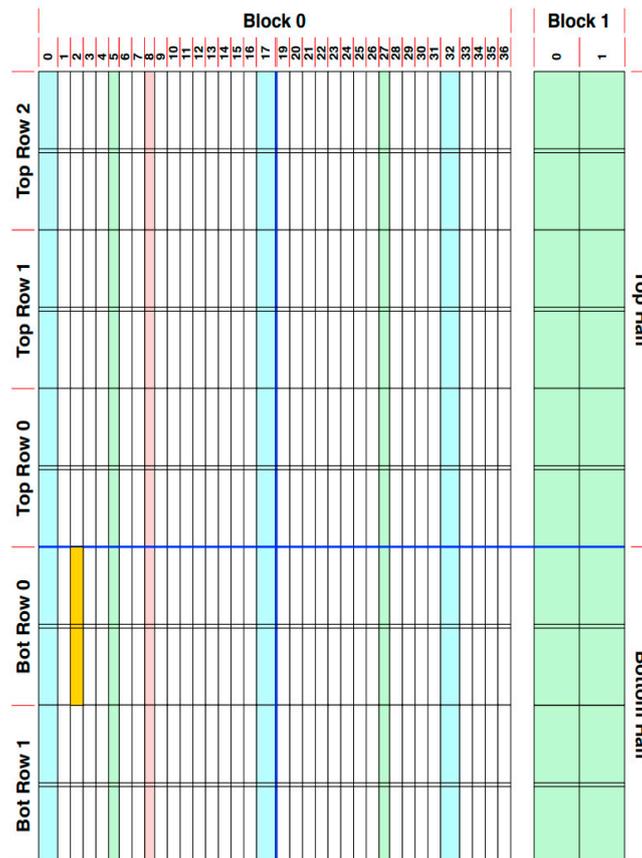


Figure 5. Graphical representation of the bitstream structure [16]. The region highlighted in yellow is located at tile 0, bottom half, row 0, and column 2, which has a frame width of 36.

The bitstream file consists of headers and packets. The header part contains information such as the chip name of the target FPGA, the creation date, and the number of bits used. The packet part contains the command data and the configuration data.

Writing in the FPGA configuration memory starts at the indicated frame of the Frame Address Register (FAR). The FAR in the actual bitstream is generated only once, as shown in Figure 6 [6]. Once the FAR is set for the first frame, the next FAR is generated by the auto-generate function to download the data to the correct location.

The configuration data, which depends on the HDL-generated circuit design, constitutes programmable points, i.e., PIP and PLP, in the FPGA. The bitstream mapping information is not disclosed in the official documents, but the total number of bits in the generated bitstream is listed in the Xilinx official documents [2]. Since the command and configuration data in the bitstream include the mapping information, the bitstream can be used to restore the netlist with reverse engineering tools, which will be introduced in Section 3.

Configuration data includes three types of configuration bits, which consist of the look-up table (LUT), other slice components, and switch boxes (i.e., PIP). The LUT and other slice components configuration occupy only about 10% of the data, the rest of it being used to represent the PIP configuration.

Using the dumped data, we can collect bits for PIP through the correlation algorithm presented in the Debit tool [4]. The correlation algorithm introduced in the Debit tool assumes that the layout of the configuration data is very regular, based on the regular structure of the FPGA. Typically, the FPGA has a regular structure, as shown in Figure 5, and the PIP bit positions within the site are consistent at each of the same sites.

The Debit is an open source tool available from Github, and is only supported by the Linux platform. It supports Virtex-2, Virtex-4, Virtex-5, and Spartan-3 products. In order to examine the reverse engineering performance of the Debit tool, we created a bitstream file with a Spartan-3 FPGA and applied it to the Debit tool to generate the XDL file. Then, the reversed XDL file (by Debit) was compared with the original XDL file (by Xilinx ISE) in order to verify the accuracy of the Debit tool. Figure 8 shows the original XDL file and the XDL file reverse-engineered from the bitstream by the Debit tool. Because only part of the pip was restored in the INT tile, Debit was not able to regenerate the complete XDL code from the bitstream. Moreover, the current ISE tool cannot be used to convert the reversed XDL file into the netlist file (i.e., NCD) because the XDL format is not compatible.

<pre>net "a_IBUF" . outbin "a" 1, inpin "d_OBUF" A1, pip CLBLL_X30Y72_SITE_IMUX_B29 -> M_A1, pip INT_INTERFACE_X31Y74_INT_INTERFACE_LOGIC_OUTS_B11 -> INT_INTERFACE_LOGIC_OUTS11, pip INT_X30Y72_SW2MID2 -> IMUX_B29, pip INT_X30Y73_WS2END2 -> WS2BEG2, pip INT_X31Y74_LOGIC_OUTS11 -> WS2BEG2, pip IOL_X31Y74_IOL_D0 -> IOL_I0, # _ROUTETHROUGH:D:O *XDL_DUMMY_IOL_X31Y74_ILOGIC_X2Y149" D -> O pip IOL_X31Y74_IOL_I0 -> IOL_LOGIC_OUTS11, pip IOL_X31Y74_IOL_IBUF0 -> IOL_D0, ; net "b" . cfg "_BELSIG:PAD,FAD,b'b",</pre>	<pre>pip INT_X31Y69 LOGIC_OUTS11 -> NL2BEG_S0 pip INT_X31Y69 LOGIC_OUTS21 -> WN2BEG2 pip INT_X30Y70 WN2END2 -> NR2BEG1 pip INT_X30Y71 LOGIC_OUTS_S12 -> NE2BEG2 pip INT_X31Y71 NL2MID0 -> NW2BEG0 pip INT_X30Y72 NW2END0 -> IMUX_B26 pip INT_X30Y72 NR2END1 -> IMUX_B27 pip INT_X30Y72 SW2MID2 -> IMUX_B29 pip INT_X31Y72 NE2END2 -> IMUX_B47 pip INT_X30Y73 WS2END2 -> WS2BEG2 pip INT_X31Y74 LOGIC_OUTS11 -> WS2BEG2</pre>
(a)	(b)

Figure 8. Reverse engineering results with debit: (a) Original XDL part, (b) Reversed XDL part: As a result of reverse engineering using the Debit tool, only some pip of the INT tile was reversed.

3.2. Bitstream Interpretation Library (BIL)

The Debit tool was the interesting first step that introduced the possibility of FPGA reverse engineering with the bitstream. Later, the BIL was proposed in order to improve the correlation method of the Debit tool by using the XDLRC, leading to more accurate reverse engineering [5]. The BIL tool separates the configuration data into tiles, which are the smallest units of the FPGA, when obtaining the PIP data. Tiles of the same type have identical configurations, so they can be separated safely and quickly. Since the bitstream mapping table is not officially released, it must be inferred through the XDLRC in order to separate the configuration data. Using the XDLRC file, which describes the FPGA architecture, the correlation between tiles can be easily estimated. The Debit tool cannot reverse all the PIP information of the FPGA by using the XDL file as a reverse source. In contrast, the BIL tool can completely reverse the PIP information using the XDLRC as a reverse source.

The BIL tool was released to analyze the FPGA and reverse the bitstream. Figure 9 shows the conceptual diagram of the BIL flow. The analysis tool includes a basic data generation function, which creates an address layout that processes the configuration memory of a specific device to be structured and a list that describes the name and ID of the Virtex-5 device. It also provides functions to compress XDLRC and create a table that points to all tile addresses in compressed XDLRC. The XDLRC compression function deletes redundant data and compresses large amounts of data without loss by converting data to the binary format.

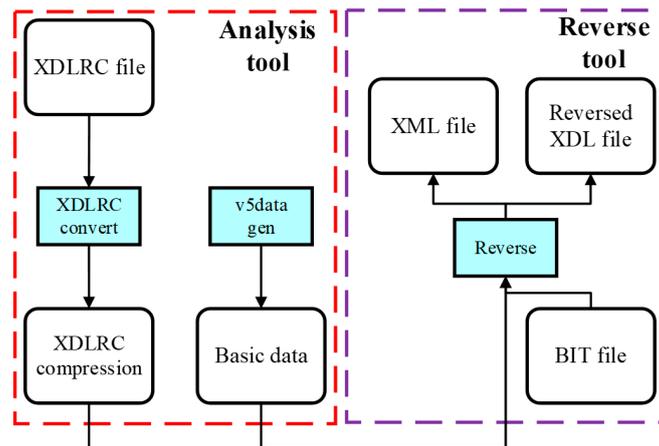


Figure 9. Conceptual diagram of the BIL flow.

Reverse engineering can be conducted using the data generated by the FPGA analysis tool. The FPGA Reverse tool additionally allows for the conversion of the bitstream into an assembly code for visualization.

Like the Debit tool, the BIL tool has been released as an open source tool, providing a reverse engineering function that can be used in the same environments as Debit. We also used the BIL tool to verify its accuracy in reverse engineering. The XDL file reversed through the BIL was compared with the original XDL file in Figure 10. It can be observed that the BIL can generate all the PIPs of the INT tiles, but not the PIP data of the primitive site and other tiles. The results indicate that the correlation algorithm introduced in the BIL does not provide the mapping information of the primitive sites and the PIP of other tiles beyond the INT tile.

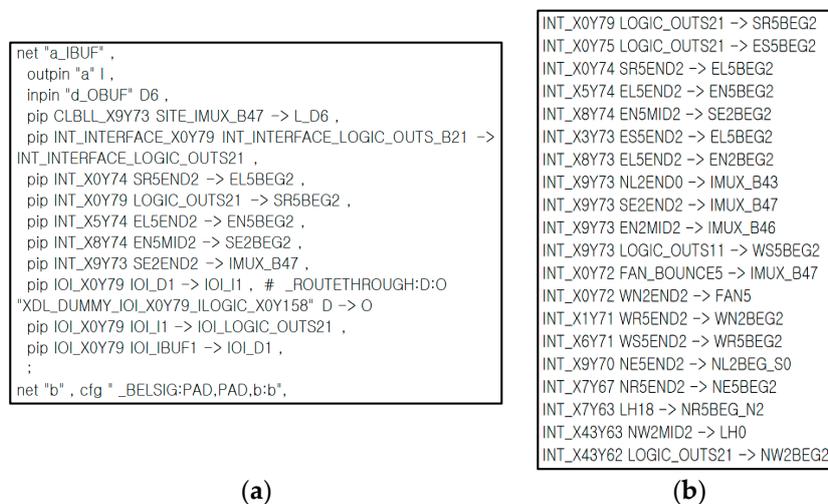


Figure 10. Reverse engineering results with the BIL tool: (a) Original XDL file generated by Xilinx ISE, (b) Reversed XDL file generated by BIL.

3.3. Bit2ncd

The previous Debit and BIL tools were unable to provide the pip mapping information of the primitive sites and other tiles, resulting in incomplete recovery. In [6], it has been experimentally verified that PIP and PLP share the same control bit, and neither the Debit nor BIL can reverse the PLP parts, because they do not consider such redundant data.

Figure 11 shows a conceptual diagram of the Bit2ncd flow. One example of the seed XDL file is shown in Figure 12a. The circuit must be created with the primitives in the FPGA devices such as LUTs, Muxes, and FFs, and they must be fixed at specific locations using a UCF file specifying I/O pins to find the pattern when creating the bitstream. Once the seed XDL has been generated,

CLB information can be found in XDLRC, and it must be modified for only one PLP (e.g., FFY in the example), as highlighted in Figure 12b. When the XDL is modified with all possible PLPs of the FFY, a set of modified XDL is then ready. The generated XDL set, i.e., analysis script group (ASG), is converted into an NCD file through -xdl2ncd provided by the ISE design suite, and then the NCD file is converted into the bitstream through -bitgen. This process is very similar to constructing a mapping table of PLP and PIP, as explained in [6].

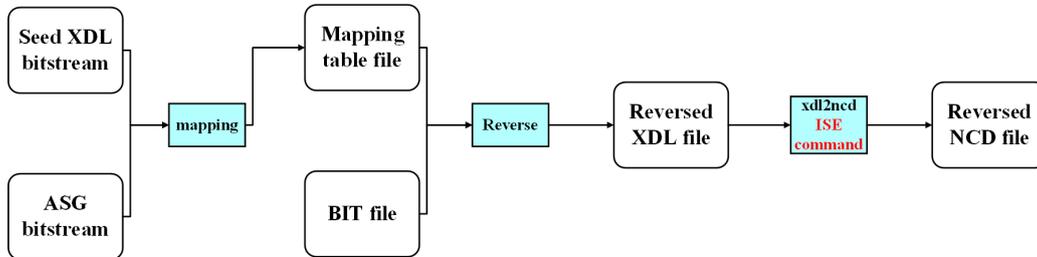


Figure 11. Conceptual diagram of the Bit2ncd flow.

<pre> Case #FF choice: inst "inst7" "SLICE", placed R2C7 CLB_R2C7.S0 cfg "BXMUX::BX.B BYMUX::#OFF CEMUX::CE CKINV::1 COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::0 DYMUX::1 F::#OFF F5USED::#OFF FFX::#FF FFY::#FF FXMUX::#OFF G::#LUT:D=(A3)*A1+A3*(A1) GYMUX::G INITX::LOW INITY::LOW RAMCONFIG::#OFF REVUSED::#OFF SRFFMUX::0 SRMUX::SR SYNC_ATTR::SYNC" </pre> <p style="text-align: center;">(a)</p>	<pre> Case #LATCH choice: inst "inst7" "SLICE", placed R2C7 CLB_R2C7.S0 cfg "BXMUX::BX.B BYMUX::#OFF CEMUX::CE CKINV::1 COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::0 DYMUX::1 F::#OFF F5USED::#OFF FFX::#FF FFY::#LATCH FXMUX::#OFF G::#LUT:D=(A3)*A1+A3*(A1) GYMUX::G INITX::LOW INITY::LOW RAMCONFIG::#OFF REVUSED::#OFF SRFFMUX::0 SRMUX::SR SYNC_ATTR::SYNC " </pre> <p style="text-align: center;">(b)</p>
--	---

Figure 12. Example codes of the analysis script group (ASG) generation for the FFY option [6]: (a) Seed XDL, (b) FFY option modified XDL.

In order to efficiently generate a mapping table, a distributed processing system is used. Its design is introduced in a separate paper on the Bit2ncd software [6]. A server generates the seed XDL and distributes it to each node. The node then generates the ASG, creates a mapping table, and transmits it to the server. The server consolidates the mapping tables generated by each node into a single one. As a result, it takes 50 days to create a mapping table of one FPGA chip with 50 PCs. Also, the mapping table must be reconfigured for each FPGA chip.

Figure 13a,b show the netlist view by the original NCD and the reversed NCD, respectively. As shown, the original NCD and the reversed NCD are the same. The simulation results also indicate that the original XDL and the reversed XDL are identical. In addition, Bit2ncd can be applied to the xdl2ncd option provided by ISE while keeping ISE XDL format, so it can easily create NCD file.

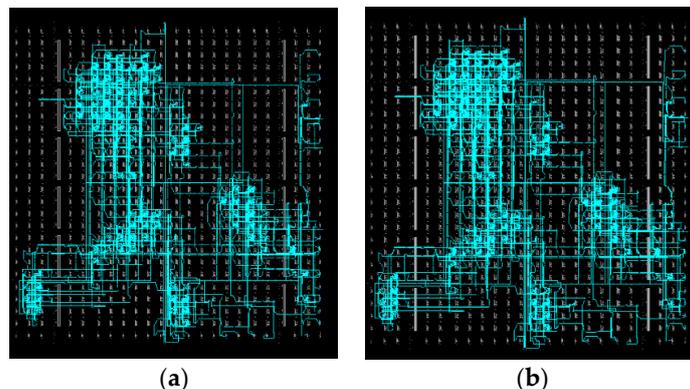


Figure 13. Reverse engineering results with the Bit2ncd tool [6]: (a) Original NCD, (b) Reversed NCD.

3.4. Supplementary Tool Using XDL: Rapidsmith

Commercial FPGAs offer several advantages for programming, but researchers have often found them inconvenient, due to the restricted functions of FPGA design tools. FPGA databases are typically proprietary, and the tools generally have rigid design flows, limiting various experiments with FPGAs. In order to overcome these issues, the Rapidsmith tool was developed for Xilinx FPGAs [13,14]. Even though Rapidsmith is not a tool for reverse engineering, it provides valuable tips and information using XDL files.

Figure 14 shows how the Rapidsmith works, along with the Xilinx CAD tool. While the configuration process of the Xilinx FPGA is identical, as explained in Figure 2, Rapidsmith can support translation, mapping, placing, and routing functions for the XDL file.

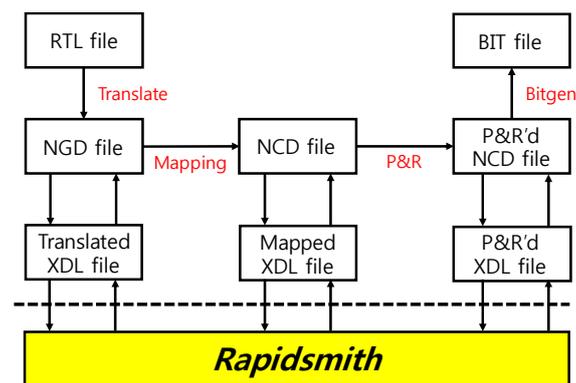


Figure 14. Conceptual diagram of the Xilinx ISE and Rapidsmith flow.

The previously introduced XDL tools use the output files of the FPGA design tool as the input files and convert them to XDL files. On the other hand, Rapidsmith uses the converted XDL file as an input and manipulates it as desired by the user. The manipulated XDL file is then fed back into the XDL tools and converted to files that are compatible with the Xilinx design tools.

From a reverse-engineering perspective, it is paramount to consider hacking attack scenarios to prevent hacking of the FPGAs. A few papers have introduced ways to utilize Rapidsmith for FPGA security. Tavaragiri et al. have demonstrated that on-chip antennas can be built by using programmable interconnects as a pallet [17]. They developed a unique flow to concatenate a large network of unused routing resources in the FPGA, forming a two-dimensional curve to work as an on-chip antenna built with the programmable interconnects of the FPGA. The authors used their own tool to manipulate XDL files, but Rapidsmith can perform the same function to support XDL files. Söll et al. demonstrated that malicious circuits, such as hardware Trojans, can be detected through side-channel analysis [18]. They used Rapidsmith to implement the denial-of-service Trojan into an AES block by manipulating P&R'd XDL files. They then used localized electro-magnetic (EM) measurements to find the Trojan-infiltrated circuit in the FPGA chip. Figure 15 shows the location of the malicious Trojan circuit (black area) on the FPGA chip and its EM measurement result in order to detect the Trojan circuit.

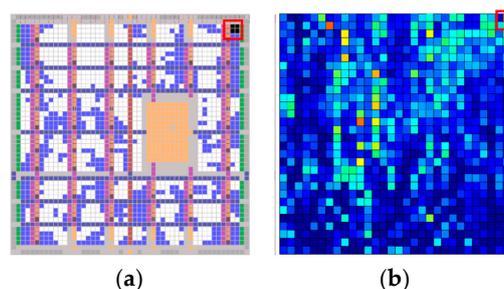


Figure 15. (a) Trojan placed at the top-right corner on the FPGA chip (highlighted by a red square) and (b) its EM measurement [18].

4. Reverse Engineering Tools with Non-Xilinx FPGAs

4.1. Icestorm with Lattice FPGAs

Icestorm is a project related to the reverse engineering that utilizes the bitstream from the Lattice ice40 FPGA with a custom-designed programming tool [7]. Its FPGA board has a minimal and regular architecture with limited types of tiles and function units to facilitate reverse engineering [19]. The Icestorm project targets the Ice40 LP/HX 1K/4K/8K chips.

Figure 16 shows conceptual diagrams meant to compare the iCEcube2 design flow officially supported by the Lattice semiconductor [20] and the custom-designed Icestorm design flow [8]. In Figure 16a, the iCEcube 2 design flow consists of several steps, as follows (1) Add RTL design files and constraint files; (2) Synthesize the design and make netlists; (3) Perform placement and routing process using place & route tools in iCEcube2 and generate programming and configuration files; (4) Program the FPGA with the bitstream.

The design flow of the Icestorm project, shown in Figure 16b, is similar to the iCEcube2 flow, while the Icestorm uses custom-designed tools. Icestorm is an open source project that provides three main tools: Yosys, Arachne-pnr, and Icepack. Yosys is a framework that converts Verilog code to a different format, such as FLIG, EDIF, BTOR, SMT-LIB, and simple RTL Verilog [21]. It has a Verilog 2005 support and provides a basic synthesis set [9]. Yosys performs functions similar to the iCEcube2 synthesis. Arachne-pnr supports placement and routing functions using BLIF files, physical constraints, and place & route scripts, ultimately generating the Icestorm TXT file. It operates in the same way as the iCEcube2 Place & Route tool. Then, the IcePack converts the TXT file into the bitstream [21].

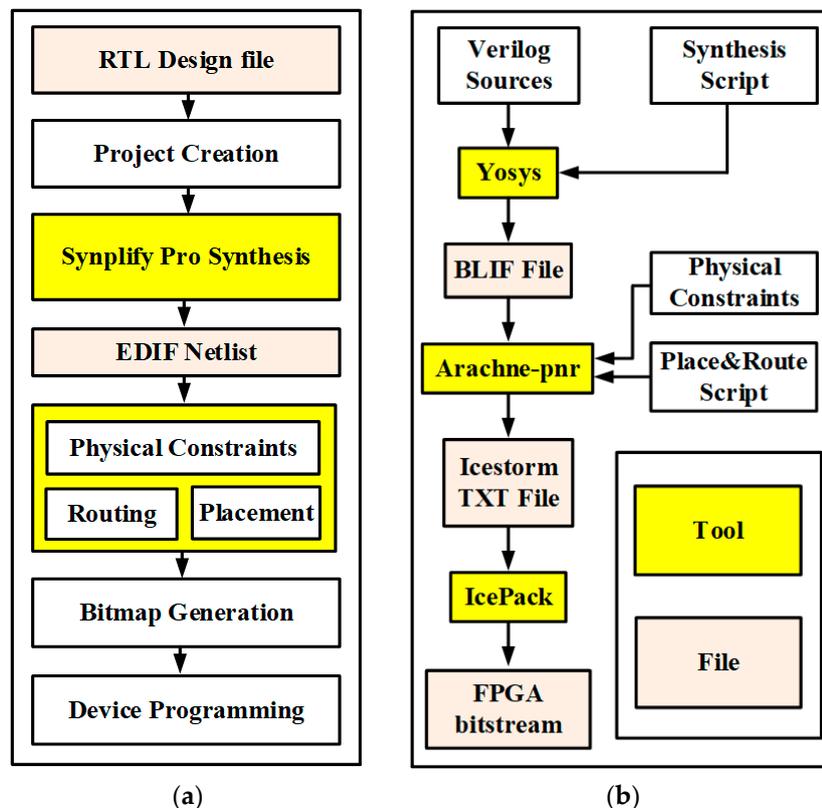


Figure 16. Design flow comparison: (a) iCEcube2 design flow, (b) Icestorm design flow.

The Icestorm project has reported that the ice40 HX 1K and 8K boards can be reverse-engineered, especially for Ice40 HK 1K-TQ144 and Ice40 HX8K-CT256 products [7]. For reverse engineering, the Icestorm utilizes the Icebox_vlog tool, which is an extension software used to reverse the BIN file to

the Verilog file. Figure 17 shows an example code for bitstream conversion to the Verilog file using the Icebox_vlog tool. The file “example.v” implements an AND logic module, and the output y is assigned as a logical result of a AND b. This example uses the Yosys, Arachne-pnr, and Icepack tools to create the bitstream. The bitstream is then applied to the Icebox_vlog tool, generating the recovered Verilog code, which includes the conditional operator, assign y = b ? a:0. This means that if b is 1, y becomes a, and if b is 0, y becomes 0, indicating that y is a logical result of both a AND b. This example verifies that reverse engineering is possible for specific Lattice FPGA products. Since the Icestorm utilized custom-designed tools for Verilog-to-bit file conversion, its reverse engineering tool (i.e., Icebox_vlog) for bit-to-Verilog file conversion was also easily implemented and customized [10].

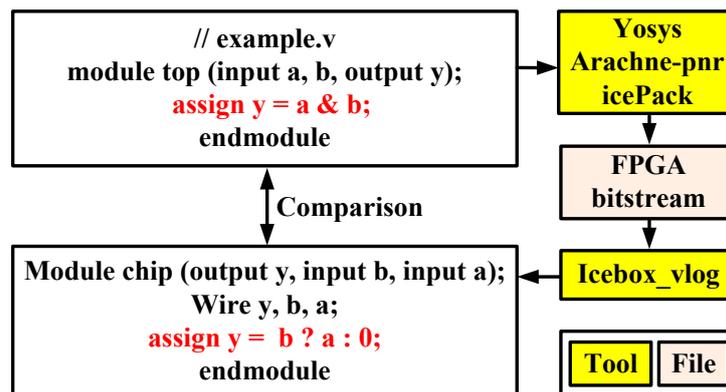


Figure 17. Example code of the bitstream conversion to the Verilog file.

4.2. QUIP with Altera FPGA

The Quartus University Interface Program (QUIP) supplied by Altera provides FPGA researchers with several design capabilities: It can create a netlist of technology specific entities as well as access the device architecture and key timing information. It can also specify the placement of logic elements and routing for FPGA design [15]. The QUIP provides documentation about a Verilog Quartus module (VQM) file format as well as a grammar that researchers can use to create VQM files with a technology specific listing. Accessing the architecture and key timing information obtained using the QUIP can support the development of FPGA flow. For example, the intra-cell delay shows time information about port to port, so that researchers can interpret the timing numbers.

The QUIP can modify the placement and routing information, which can be applied to the Quartus II FPGA. The QUIP is convenient and flexible because it can change certain parts of placement and routing while enabling the replacement of parts in the FPGA flow with custom tools. However, the QUIP does not provide the detailed information file to construct a router, as does the XDL [14]. Moreover, the QUIP only supports the Quartus II FPGA, which significantly limits its usage in reverse engineering for Altera FPGAs.

5. Discussion

Table 2 summarizes reverse engineering tools for FPGAs. Debit attempts reverse engineering using the correlation method between XDL and Bitstream files, but it suffers from low accuracy due to the lack of information in XDL. BIL complements Debit’s correlation method using XDLRC and completely reverses the PIP information of INT tiles. However, the PLP information cannot be extracted due to the lack of information in XDLRC.

Debit and BIL were released as an open-source tool, and we directly tried those tools to experimentally verify their performance and limitations. Reversed XDL files extracted by Debit and BIL were compared with the original XDL generated by the ISE design flow as shown in Figure 8 (Debit) and 10 (BIL). Both reverse engineering tools show low accuracy as a result of extracting limited tile resources.

Bit2ncd is not an open-source tool and has not been tested directly. Bit2ncd compares the ASG bitstream with the original bitstream as shown in Figure 12 to create a mapping table. Figure 13 shows a complete reverse engineering result using Bit2ncd. However, it takes a quite long time (50 days) to create a mapping table for reverse engineering, while requiring huge computational resources (50 PCs), according to the paper.

Table 2. Performance comparison of the reverse engineering tools for FPGAs.

Tool	Input	Output	Supported Device	Pros	Cons
Debit	BIT file	XDL file	Virtex-2,3,4,5/Spartan-3	Simple	Low accuracy
BIL	XDLRC, XDL, BIT file	XDL file XML file	Virtex-5	Perfect reversing pip in INT tile	Low accuracy
Bit2ncd	BIT file	XDL file NCD file	Spartan-3,3E/Virtex-2,4,5	Perfect reversing all tile types	Long time
RapidSmith	XDL file, XDLRC file	XDL file	All devices	Flexible	Depending on other tools
Icestorm	BIT file	Verilog file	Ice40	High accuracy	Limited target devices

RapidSmith can manipulate the XDL to modify and customize the netlist file, which can be applicable to all devices. RapidSmith alone cannot be used for reverse engineering in FPGA, while it can provide the flexible function for manipulating XDL to assist the other reverse engineering tools.

Icestorm uses a custom-designed programming tool, as shown in Figure 16b, which also leads to the design of a custom reverse engineering tool. Unlike conventional reverse engineering tools, Icestorm can accurately extract the Verilog code from the bitstream, as shown in Figure 17. However, the Icestorm project is only applicable to Lattice FPGA products.

6. Conclusions

Debit is the first attempt to reverse engineer the FPGAs through bitstream extraction, and the BIL improves the bitstream correlation method by utilizing XDLRC files. As a further improvement, Bit2ncd reverse engineers all tiles by creating a mapping table while modifying both PLP and PIP in XDL files. However, the latter requires tremendous computational resources and is not an open source tool, so it cannot be verified by other users. Further research on FPGA reverse engineering is still needed, not only for improving the existing tools but also for reverse engineering the other FPGAs, such as recent Xilinx products using Vivado design flow.

Author Contributions: Conceptualization, Methodology, Data Curation, Validation, Writing Paper, H.Y., H.L., and S.L.; Funding Acquisition, H.-M.L.; Investigation, Supervision, Writing-Review & Editing, Project Administration, Y.K. and H.-M.L.

Funding: This work was supported as part of Military Crypto Research Center (UD170109ED) funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fong, R.J.; Harper, S.J.; Athanas, P.M. A versatile framework for FPGA field updates: an application of partial self-reconfiguration. In Proceedings of the IEEE International Workshop on Rapid Systems Prototyping, San Diego, CA, USA, 9–11 June 2003; pp. 117–123.
2. Virtex-5 FPGA Configuration User Guide, 2017. Available online: https://www.xilinx.com/support/documentation/user_guides/ug191.pdf (accessed on 18 September 2018).
3. Lee, J.-K.; Jhang, K.-S.; Cho, H.-J. Verilog functional model extraction from FPGA design data. *J. KIISE: Comput. Pract. Lett.* **2012**, *18*, 380–388.

4. Note, J.-B.; Rannaud, É. From the bitstream to the netlist. In Proceedings of the International ACM/SIGDA Symposium on Field Programmable Gate Arrays (FPGA), Monterey, CA, USA, 24–26 February 2008; Volume 18, p. 264.
5. Benz, F.; Seffrin, A.; Huss, S.A. Bil: A tool-chain for bitstream reverse-engineering. In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, 29–31 August 2012; pp. 735–738.
6. Ding, Z.; Wu, Q.; Zhang, Y.; Zhu, L. Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation. *Microprocess. Microsyst.* **2013**, *37*, 299–312. [[CrossRef](#)]
7. Project IceStorm. Available online: <http://www.clifford.at/icestorm> (accessed on 18 September 2018).
8. Wolf, C. A free and Open Source Verilog-to-Bitstream Flow for iCE40 FPGAs. Online Presentation, 2015. Available online: https://media.ccc.de/v/32c3-7139-a_free_and_open_source_verilog-to-bitstream_flow_for_ice40_fpgas (accessed on 18 September 2018).
9. Yosys Open Synthesis Suite, Online Tutorial. Available online: <http://www.clifford.at/yosys> (accessed on 18 September 2018).
10. Romanov, A.; Romanov, M.; Kharchenko, A. FPGA-based control system reconfiguration using open source software. In Proceedings of the IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg, Russia, 1–3 February 2017.
11. Yui, C.C.; Swift, G.M.; Carmichael, C.; Koga, R.; George, J.S. SEU mitigation testing of Xilinx Virtex II FPGAs. In Proceedings of the IEEE Radiation Effects Data Workshop, Monterey, CA, USA, 25–25 July 2003; pp. 92–97.
12. Moradi, A.; Barengi, A.; Kasper, T.; Paar, C. On the vulnerability of FPGA bitstream encryption against power analysis attacks. In Proceedings of the ACM Conference on Computer and Communications Security, Chicago, IL, USA, 17–21 October 2011; pp. 111–124.
13. Lavin, C.; Padilla, M.; Lundrigan, P.; Nelson, B.; Hutchings, B. Rapid prototyping tools for FPGA designs: Rapidsmith. In Proceedings of the International Conference on Field-Programmable Technology, Beijing, China, 8–10 December 2010.
14. Lavin, C.; Padilla, M.; Lamprecht, J.; Lundrigan, P.; Nelson, B.; Hutchings, B. Rapidsmith: Do-it-yourself CAD tools for Xilinx FPGAs. In Proceedings of the International Conference on Field Programmable Logic and Applications, Chania, Greece, 5–7 September 2011.
15. Malhotra, S.; Borer, T.; Singh, D.; Brown, S. The Quartus University Interface Program: Enabling advanced FPGA research. In Proceedings of the IEEE International Conference on Field-Programmable Technology, Brisbane, NSW, Australia, 6–8 December 2004; pp. 225–230.
16. Soni, R.K.; Steiner, N.; French, M. Open-source bitstream generation. In Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines, Seattle, WA, USA, 28–30 April 2013; pp. 105–112.
17. Tavaragiri, A.; Couch, J.; Athanas, P. Exploration of FGPA interconnect for the design of unconventional antennas. In Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 27 February–1 March 2011; pp. 219–226.
18. Söll, O.; Korak, T.; Muehlberghuber, M.; Hutter, M. EM-based detection of hardware trojans on FPGAs. In Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust, Arlington, VA, USA, 6–7 May 2014.
19. iCE40 LP/HX Family Data Sheet. Available online: <http://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40> (accessed on 18 September 2018).
20. iCEcube 201708 Tutorial. Available online: <http://www.latticesemi.com/Products/DesignSoftwareAndIP/FPGAandLDS/iCEcube2> (accessed on 18 September 2018).
21. Krieg, C.; Wolf, C.; Jantsch, A. Malicious LUT: A stealthy FPGA trojan injected and triggered by the design flow. In Proceedings of the International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 7–10 November 2016.

