Reverse Engineering for Xilinx FPGA Chips using ISE Design Tools

So Yeon Choi[1], Ji Woon Park and Ho Young Yoo[a]
Department of Electronics Engineering, Chungnam National University
E-mail : [1]soyeonchoi@cnu.ac.kr

Abstract - **SRAM-based Field Programmable Gate Arrays (FPGAs) are more widely used compared to Flash-based and anti-fuse based FPGAs in various industries. One disadvantage of the SRAM-based FPGA is that SRAM is natively volatile and thus it requires additional nonvolatile memory to store the netlist information of the circuit outside the FPGA. When the FPGA is powered on, the bitstream file is transferred from the external nonvolatile memory to the SRAM-based FPGA. The secure issues arise if the bitstream is modified or corrupted by attacker resulting in a fatal problem in the circuit. Therefore, reverse engineering that converting a bitstream into an internal netlist is necessary to find such harmful modification. In this paper, we describe the overall process of reverse engineering based on ISE design tools in details. According to the experimental results, the proposed reverse engineering tool can recover 88% internal circuit as for the example of 64-bit LFSR design.**

***Keywords*—ISE Design Tools, Programmable logic points, Programmable interconnect points, Reverse Engineering, Xilinx FPGA**

I. INTRODUCTION

Application-specific integrated circuit (ASIC) must be designed to satisfy the specifications of application systems. Thus, ASIC requires considerable time and cost to build the system, and once the circuit is produced as a chip, it cannot be modified. To mitigate this disadvantage of ASICs, field-programmable gate arrays (FPGAs) are used in several industries. FPGAs have the advantage of parallel and high-speed processing because the circuit can be added or changed freely even after the circuit has been configured and the digital circuit operates directly [1].

FPGAs are classified into SRAM-based, FLASH-based, and FUSE-based FPGAs according to the fabrication method, and the SRAM-based FPGA is most widely used owing to advantages in area, process, and speed [2]. However, the SRAM-based FPGA is volatile and requires additional nonvolatile memory to store the netlist information of the circuit outside the FPGA. When the FPGA is powered, the internal circuit information is transferred from the external nonvolatile memory to the FPGA as a bitstream, and the FPGA operates based on this transferred bitstream. The bitstream transferred from the external memory contains all the circuit configuration information of the FPGA. Thus, if the bitstream is corrupted, it will cause a fatal problem in the circuit. It is necessary to determine whether the transmitted bitstream contains the original circuit information to minimize the damage caused by an impaired circuit. The process of converting a bitstream into a file that contains the internal netlist information of the FPGA is called reverse engineering. Various studies on reverse engineering have been conducted recently to restore the programmable logic points (PLPs) and programmable interconnect points (PIPs) of FPGAs [3]-[10]. Several reverse engineering tools have been developed, including Debit [3], which was first developed, BIL [4], Bit2ncd [5], BRET [6], and Bit2RTL [7] to enhance the recovery range.

This paper introduces the reverse engineering process to reconstruct PIP and PLP into a circuit using Xilinx design language (XDL), Xilinx design language routing and configurable logic block (XDLRC), and bitstream files generated using the ISE design tool. In this process, a mapping table is created by comparing the options of PLP and PIP containing the configurable information of the FPGA with the bitstream. The bitstream is restored to a netlist file based on this mapping table.

II. BACKGROUND

To implement a circuit in Xilinx FPGAs, we can use the ISE design tool or Vivado developed by Xilinx. The ISE design tool is used for FPGAs before the 7 series that was recently developed by Xilinx, and Vivado is used for FPGAs after the 7 series. However, Vivado does not provide the internal netlist in a readable format for developers, making it difficult to obtain the information required to perform reverse engineering. Therefore, we performed reverse engineering with FPGAs using the ISE design tool, which provides an internal netlist as a form of XDLRC and XDL.

Figure 1 shows a flowchart of the Xilinx ISE design tool with files created in each step. The first native generic database (NGD) file created expresses the input register-transfer-level (RTL) design as an internal netlist file. When an NGD file, which is a netlist file, is mapped to an FPGA

[a] Corresponding author; hyyoo@cnu.ac.kr

Manuscript Received Dec. 02, 2019, Revised Dec. 23, 2019, Accepted Dec. 26, 2019

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/bync/3.0) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

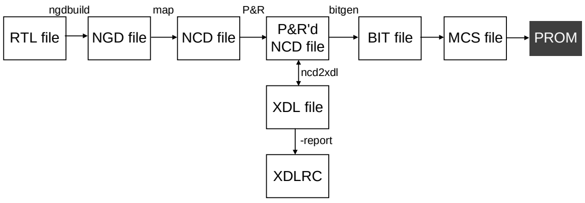
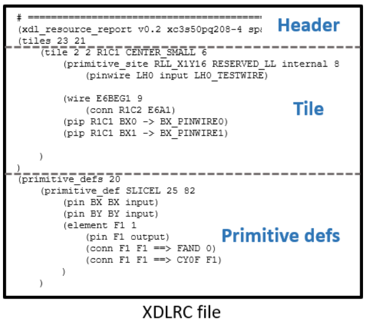


Fig. 1. Xilinx ISE Design Tools flow


Header
Tile
Primitive defs
XDLRC file

```
# ==========================================
(xdl_resource_report v0.2 xc3s50pq208-4 spa
(tiles 23 21
    (tile 2 2 R1C1 CENTER_SMALL 6
        (primitive_site RLL_X1Y16 RESERVED_LL internal 8
            (pinwire LH0 input LH0_TESTWIRE)

        (wire E6BEG1 9
            (conn R1C2 E6A1)
        (pip R1C1 BX0 -> BX_PINWIRE0)
        (pip R1C1 BX1 -> BX_PINWIRE1)

    )
)
(primitive_defs 20
    (primitive_def SLICEL 25 82
        (pin BX BX input)
        (pin BY BY input)
        (element F1 1
            (pin F1 output)
            (conn F1 F1 ==> FAND 0)
            (conn F1 F1 ==> CY0F F1)
        )
    )
```

Fig. 2. XDLRC file format.

circuit, a native circuit description (NCD) file is generated. The NCD file expresses the RTL design as a primitive of the circuit. When the NCD file undergoes the place & route process, a P&Red NCD file is created. This NCD file contains information mapped to the FPGA from the RTL input to the FPGA, but users cannot understand it because it is stored in binary format. An NCD file can be converted to a Xilinx design language (XDL) file that can be understood by users. The XDL file can be created using the ncd2xdl option of the xdl command in the ISE design tool. The XDL file can provide the PLP and PIP information of the FPGA because it has the same netlist information as the NCD file. However, the XDL file only shows the circuit information used in the FPGA, and the information of the entire chip is unknown. A Xilinx design language routing and configurable logic block (XDLRC) file is required to find out the information of the entire chip, and it can be generated using the -report option of the xdl command. The Xilinx FPGA is an SRAM-based FPGA, which stores internal netlist information in external memory as a bitstream. The bitstream file format converted from the FPGA is a BIT file and the information of the NCD file is stored as a bitstream. The BIT file can be created using the bitgen command with the -b option, which creates a raw bit file in ASCII code. The BIT file is stored in an external memory PROM as an MCS (configuration memory) file, and the stored bitstream is transferred to the FPGA when the FPGA is powered.

A. Analysis of XDLRC file

The XDLRC file describes the FPGA chip in a hierarchical top-down manner, and the overall configuration consists of a *header* section, *tile* section, *primitive_defs* section as shown in Figure 2. The *header* section at the top of the file shows the type and information of the FPGA chip currently represented in the XDLRC file. For example, if the target device is xc3s50vq100-5, xc3 indicates the Spartan-3 device, s50 indicates the size of the chip, vq100 indicates the type of package applied to the s50 chip, and -5 indicates the speed level of the chip. The *tile* section shows the total number of rows and columns of tiles when the FPGA is configured in a two-dimensional array, the name and type of each *tile*, and the number of declared *primitive*s in this order. In addition, the types of *primitive_site*, *wire*, *conn*, and *pip* declared according to each tile type are indicated in the *tile* section.

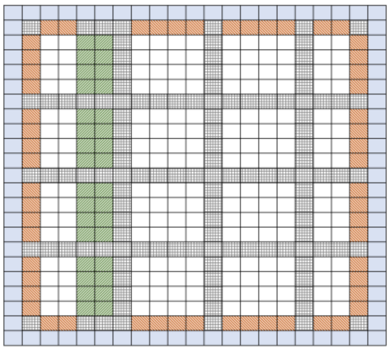


Fig. 3. Internal structure of the xc3s50 devices.

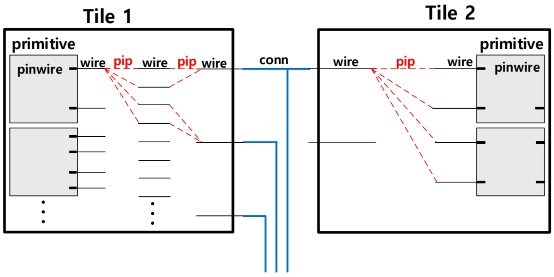


Fig. 4. Internal circuit structure of tile.

Figure 3 shows a schematic of the entire chip with each tile type categorized using the XDLRC file of xc3s50vq100-5. The types of tiles are configurable logic block (CLB), IOB, TERM, Block RAM, and GCLK. CLB consists of logical resources such as PLP and PIP for implementing sequential and combination circuits. TERM is located at the edge of the FPGA and consists of wires and pips that take charge of the signal connections between nearby tiles. Block RAM is a default storage space provided by FPGA, and there is a multiplier near each block RAM. GCLK is a set of tiles containing *primitive*s related to clock information such as clock buffer and digital clock manager (DCM). IOB provides unidirectional or bidirectional interfaces between the pins in package of the chips and the internal logic of FPGA. Figure 4 shows a schematic internal structure of the tiles in an FPGA chip. *Tile*s are the largest unit of an FPGA chip and are arranged in a grid form in the FPGA. *Primitive*s in a *tile* represent circuits for performing a specific operation, and the available *primitive*s differ depending on the type of *tile*. Table I lists the types of *primitive*s that can be used in each tile. To express the circuit configuration of tiles, we need not only primitives, but also *wire*s which indicate the paths of signals declared in a tile, *conn*s which indicate fixed connections between different *tile*s, and *pip*s which indicate connections between freely configurable wires.

The *primitive_defs* section, which describes the configuration information of a primitive that is a lower layer of the *tile*, is described after the *tile* section in the XDLRC

TABLE I.

Types and numbers of primitives according to the category of tiles.

Tile	Type of primitive	Number of primitives
GCLK	BSCAN, DCIRESET, GLOBALSIG, ICAP, BUFGMUX, CAPTURE, DCI, PMV, STARTUP, DCM, VCC, RESERVED_LL	12
CLB	SLICEL, SLICEM, VCC, RESERVED_LL	4
IOB	DIFFM, DIFFS, IOB, RESERVED_ANDOR, VCC, RESERVED_LL	6
Block RAM	RAMB16, MULT18X18	2
TERM	-	-

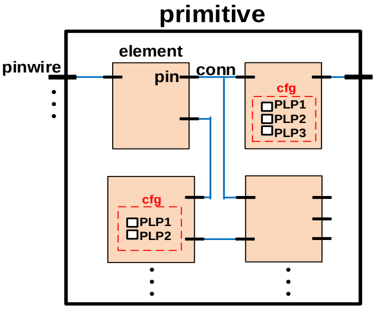


Fig. 5. Internal circuit structure of primitive.

file. This section shows the internal structure of all kinds of *primitive*s configured in the *primitive_site*, the physical location of the *primitive*s. The *primitive_defs* section consists of pinwires corresponding to the input and output and elements implementing the internal circuit. Figure 5 shows the configuration of the *primitive_site* based on the *primitive_defs* section and Table II shows the number of PLPs that appear in each *primitive*. An *element* in a *primitive* consists of a *pin* that represents the input and output of the *element*, a *conn* that represents a fixed connection to another *element*, and a *cfg* that can be used to configure freely the inside of the circuit. The XDLRC file representing the xc3s50vq100-5 chip in Figure 2 has a size of 46,143,454 bytes and consists of 1,824,280 lines.

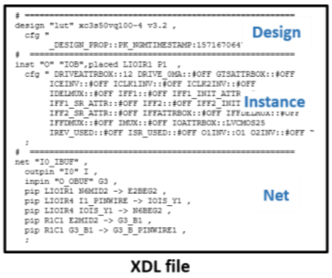
B. Analysis of XDL file

The internal structure of an XDL file is largely composed of three sections: *design*, *instance*, and *net* as shown in Figure 6. The *design* section, which is located at the top of the file, shows the name of the design and the FPGA device information. The *instance* section describes the primitive instance information of a *primitive_site* in a specific *tile*, which includes the instance name, the type of the instanced *primitive*, and the layout information in the chip, which shows the *primitive_site*s of specific *tile*s. Next, there is a *cfg* string that shows the internal element configuration according to the type of the instanced primitive. The *cfg* also shows the type of PLP used in the *primitive* and the currently used option of PLP. Finally, the *net* section at the bottom shows the *inpin* and *outpin* information and PIP information. The *inpin* and *outpin* are determined by the *pinwire* of the

TABLE II.

Number of PLPs according to the type of primitive.

Type of primitive	Number of PLPs	Type of primitive	Number of PLPs
BSCAN	0	SLICEL	31
DCIRESET	0	SLICEM	51
GLOBALSIG	1	RESERVED_ANDOR	1
ICAP	3	DIFFM	54
BUFGMUX	4	DIFFS	54
CAPTURE	3	IOB	54
DCI	1	DCM	31
PMV	0	RAMB16	12
STARTUP	3	MULT18X18	3
RESERV_LL	0	VCC	0



```
# ===================================================================
design "lut" xc3s50vq100-4 v3.2 ,
  cfg "
       _DESIGN_PROP::PK_NGMTIMESTAMP:157167064
#  ==================================================================
inst "O" "IOB",placed LIOIR1 P1  ,
  cfg " DRIVEATTRBOX::12 DRIVE_0MA::#OFF GTSATTRBOX::#OFF
       ICEINV::#OFF ICLK1INV::#OFF ICLK2INV::#OFF
       IDELMUX::#OFF IFF1::#OFF IFF1_INIT_ATTR
       IFF1_SR_ATTR::#OFF IFF2::#OFF IFF2_INIT
       IFF2_SR_ATTR::#OFF IFFATTRBOX::#OFF IFF
       IFFDMUX::#OFF IMUX::#OFF IOATTRBOX::LVCMOS25
       IREV_USED::#OFF ISR_USED::#OFF O1INV::O1 O2INV::#OFF "
  ;
#  ==================================================================
net "I0_IBUF" ,
  outpin "I0" I ,
  inpin "O_OBUF" G3 ,
  pip LIOIR1 N6MID2 -> E2BEG2 ,
  pip LIOIR4 I1_PINWIRE -> IOIS_Y1 ,
  pip LIOIR4 IOIS_Y1 -> N6BEG2 ,
  pip R1C1 E2MID2 -> G3_B1 ,
  pip R1C1 G3_B1 -> G3_B_PINWIRE1 ,
  ;
```

Fig. 6. XDL file format.

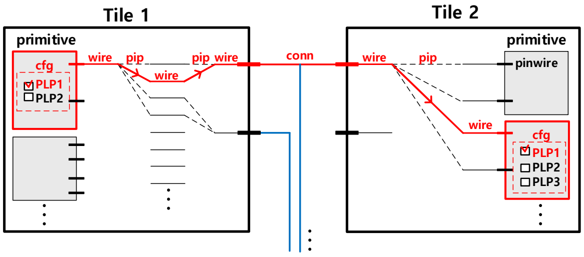


Fig. 7. Reconstruction of the internal circuit according to XDL.

instanced *primitive*. This section declares the PIPs required to connect from the first *outpin* to the final *inpin* using *conn*s, which are fixed connections with external *tile*s. Figure 7 shows the result of implementing a circuit by combining the PLPs and PIPs of the *instance* and the *net* sections of the XDL. Whether to use *primitive_site*s inside a tile is determined by the circuit to be implemented. The red line in Figure 7 indicates the currently used *primitive*s among the various *primitive*s in the *tile*, and the currently used options of PLP and PIP among the various options.

C. Analysis of BIT file

The internal netlist information is stored in external memory as a bitstream. The **bitgen** command of the ISE design tool is used to create a BIT file in binary format. To convert a binary file into a human-readable file using ASCII code, a raw bit file is created using the **-b** option and this file is used for reverse engineering. A raw bit file consists of 13,681 lines of 32 bits per line, and the entire file is divided into a *command* section, *configuration data* section, and *terminal* section as shown in Figure 8. The *configuration command* section shows commands to inform

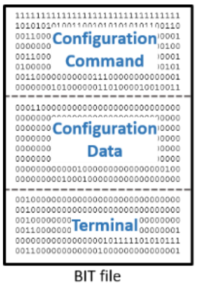


Fig. 8. Raw bit file format.

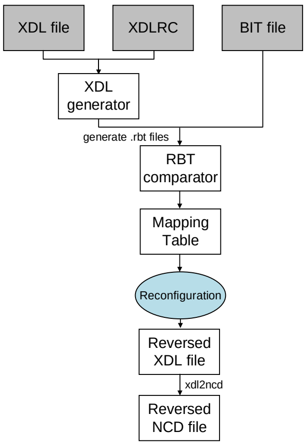


Fig. 9. Algorithm of reverse engineering.

synchronization, CRC check, and the amount of configuration data expressed in bitstreams. The *configuration data* section stores information about all the *tile*s of the FPGA, and analysis of this information is essential for reverse engineering. Bitstreams are arranged sequentially by the type of *tile*s in the same way as the chip configuration. One frame consists of 37 lines, and the number of frames represented as bitstreams is fixed for each tile category. The *terminal* section is located at the end of the raw bit file, and it consists of CRC check section and desync section, which terminates synchronization with the external memory.

III. REVERSE ENGINEERING

Reverse engineering must be performed using the information stored in the files analyzed in section II. Figure 9 shows the algorithm of reverse engineering. First, the mapping table of PLP and PIP is created by using BIT, XDL, and XDLRC files. Then, reconfiguration is performed to generate the XDL file by using the mapping table and bitstreams.

A. Mapping table generation

As the PLP and PIP have different bitstreams depending on the configuration options, the bitstreams for all configurable options must be generated and compared by modifying the XDL. Furthermore, the process of arranging the comparison result of raw bit files into a mapping table is essential for reverse engineering.

First, an XDL file applying the configurable options of all PLPs must be generated for each *primitive* to create a mapping table of PLPs. For example, SLICEL has 31 PLPs and the 31 PLPs have 1 to 7 options excluding *#OFF*. Hence, a total of 89 XDL files are required to create a mapping table for SLICEL. The number of XDL files to be generated according to each primitive is hundreds for PLPs and thousands for PIPs. Thus, an XDL generator can be created to generate XDL files automatically by applying the options of each PLP. Figure 10 shows a flowchart of the algorithm of the XDL generator. The inputs of the XDL generator are the original XDL file and the target *primitive* type, and the output is the XDL file with changed options of each PLP belonging to the target *primitive*. Figure 11 illustrates the behavior of the XDL generator with an example of FFX when the PLP has three options: *#OFF, #FF,* and *#LATCH*. Figure 11 (a) shows the *cfg* string representing SLICEL in the original XDL file, and the options of FFX are determined by the red squared part. Figure 11 (b) shows the cfg of XDL files with FFX options changed to *#OFF, #FF*, and *#LATCH*.

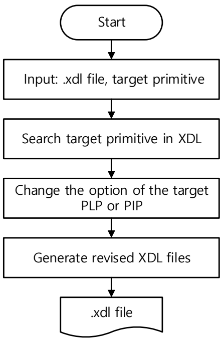
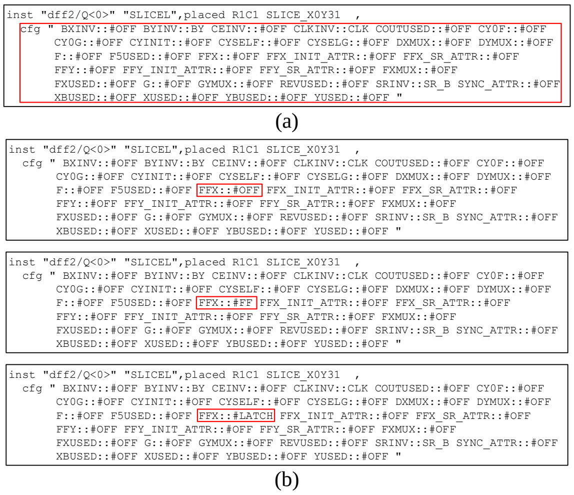

Start
Input: .xdl file, target primitive
Search target primitive in XDL
Change the option of the target PLP or PIP
Generate revised XDL files
.xdl file

Fig. 10. Algorithm of XDL generator.



```
inst "dff2/Q<0>" "SLICEL",placed R1C1 SLICE_X0Y31  ,
  cfg " BXINV::#OFF BYINV::BY CEINV::#OFF CLKINV::CLK COUTUSED::#OFF CY0F::#OFF
       CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::#OFF
       F::#OFF F5USED::#OFF FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
       FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF FXMUX::#OFF
       FXUSED::#OFF G::#OFF GYMUX::#OFF REVUSED::#OFF SRINV::SR_B SYNC_ATTR::#OFF
       XBUSED::#OFF XUSED::#OFF YBUSED::#OFF YUSED::#OFF "
```

(a)

```
inst "dff2/Q<0>" "SLICEL",placed R1C1 SLICE_X0Y31  ,
  cfg " BXINV::#OFF BYINV::BY CEINV::#OFF CLKINV::CLK COUTUSED::#OFF CY0F::#OFF
       CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::#OFF
       F::#OFF F5USED::#OFF FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
       FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF FXMUX::#OFF
       FXUSED::#OFF G::#OFF GYMUX::#OFF REVUSED::#OFF SRINV::SR_B SYNC_ATTR::#OFF
       XBUSED::#OFF XUSED::#OFF YBUSED::#OFF YUSED::#OFF "
```

```
inst "dff2/Q<0>" "SLICEL",placed R1C1 SLICE_X0Y31  ,
  cfg " BXINV::#OFF BYINV::BY CEINV::#OFF CLKINV::CLK COUTUSED::#OFF CY0F::#OFF
       CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::#OFF
       F::#OFF F5USED::#OFF FFX::#FF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
       FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF FXMUX::#OFF
       FXUSED::#OFF G::#OFF GYMUX::#OFF REVUSED::#OFF SRINV::SR_B SYNC_ATTR::#OFF
       XBUSED::#OFF XUSED::#OFF YBUSED::#OFF YUSED::#OFF "
```

```
inst "dff2/Q<0>" "SLICEL",placed R1C1 SLICE_X0Y31  ,
  cfg " BXINV::#OFF BYINV::BY CEINV::#OFF CLKINV::CLK COUTUSED::#OFF CY0F::#OFF
       CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::#OFF
       F::#OFF F5USED::#OFF FFX::#LATCH FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
       FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF FXMUX::#OFF
       FXUSED::#OFF G::#OFF GYMUX::#OFF REVUSED::#OFF SRINV::SR_B SYNC_ATTR::#OFF
       XBUSED::#OFF XUSED::#OFF YBUSED::#OFF YUSED::#OFF "
```

(b)

Fig. 11. Operation of XDL generator; (a)original XDL file (b)modified XDL files.

The XDL files with the PLP or PIP options changed one by one are first converted to NCD and then used to generate bitstream files. A mapping table for bitstream files is created by comparing bitstream files with the *#OFF* option indicating that PLP is not used and with other options indicating that PLP for each PLP and checking the bits that have changed. To compare between bitstream files, 13,681 lines of bitstreams with each line consisting of 32 bits must be compared for the number of times excluding the *#OFF* option of the target PLP or PIP. For this task, a bitstream

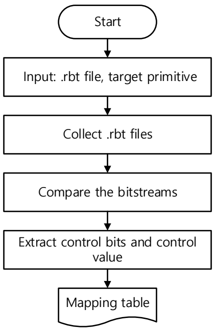
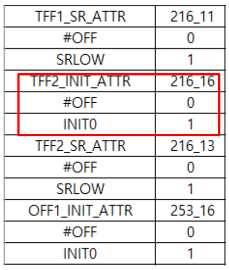

Start
Input: .rbt file, target primitive
Collect .rbt files
Compare the bitstreams
Extract control bits and control value
Mapping table

Fig. 12. Algorithm of BIT comparator.



TFF1_SR_ATTR	216_11
#OFF	0
SRLOW	1
TFF2_INIT_ATTR	216_16
#OFF	0
INIT0	1
TFF2_SR_ATTR	216_13
#OFF	0
SRLOW	1
OFF1_INIT_ATTR	253_16
#OFF	0
INIT0	1

Fig. 13. Mapping table of I/O tile.

comparator that performs this comparison automatically must be used. Figure 12 shows the algorithm for the overall operation of the bitstream (BIT) comparator. The inputs are the entire raw bit file and the target primitive. By comparing the bitstreams with the *#OFF* option applied for all PIPs and PLPs and the raw bit file with the other options, mapping table for PLPs and PIPs for each *primitive* can be created as shown in Figure 13. The PLP, the position of the bit, the bit value of the OFF option, and the bit value of the option appear in this order. For example, among the PLPs of the IOB tile in Figure 13, the position where the bit for TFF2_INIT_ATTR appears is 216_16, the bit value of the OFF option is 0, and the bit value of the INIT0 option is 1. It is worthwhile to notice that the proposed reverse engineering is more efficient compared to Bit2ncd [5]. Whereas Bit2ncd[5] constructs all the possible network to recover a single PIP, the proposed method construct a branch of an entire network, and thus it can highly save a time to build the PIP mapping table by reducing searching space.

B. Circuit reconstruction

The aim for circuit reconstruction is to convert a raw bit file into an XDL file based on the created mapping table. It is required to know which of the PLP and PIP options were used in order to perform this conversion. The bits that appear in the raw bit file to be converted are searched in the mapping table created earlier. In this searching process based on the position of the bit, the position of the *tile*, the type of *primitive*, the used PLP or PIP, and the used options can be observed. However, if the PLP option has the same bit representation as the PLP option of *#OFF*, the recovery becomes more complicated since no bit difference appears when it is compared with the basic XDL file. In this case, the option of the PLP to be used must be selected by comparing it with the options of other PLPs that are already represented as bits. For example, if an option other than *#OFF* is represented by the bit FFX_INIT_ATTR, FFX can be declared only if an option other than the *#OFF* option is selected. Therefore, it must be considered that the *#FF* option having the same bit representation as the *#OFF* option among the FFX options was used.

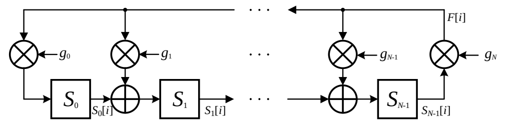
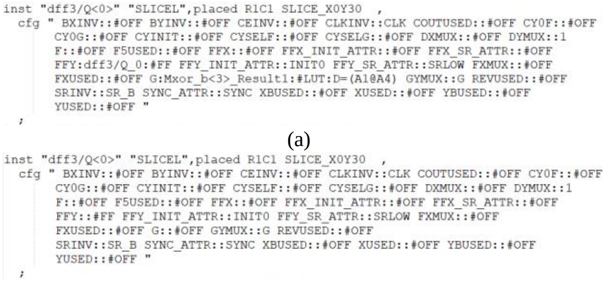


Fig. 14. 64-bit LFSR circuit when N = 64.



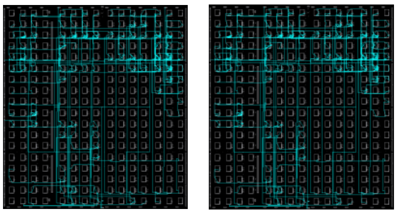
```
inst "dff3/Q<0>" "SLICEL",placed R1C1 SLICE_X0Y30  ,
  cfg " BXINV::#OFF BYINV::#OFF CEINV::#OFF CLKINV::CLK COUTUSED::#OFF CY0F::#OFF
       CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::1
       F::#OFF F5USED::#OFF FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
       FFY:dff3/Q_0:#FF FFY_INIT_ATTR::INIT0 FFY_SR_ATTR::SRLOW FXMUX::#OFF
       FXUSED::#OFF G:Mxor_b<3>_Result1:#LUT:D=(A1@A4) GYMUX::G REVUSED::#OFF
       SRINV::SR_B SYNC_ATTR::SYNC XBUSED::#OFF XUSED::#OFF YBUSED::#OFF
       YUSED::#OFF "
  ;
```

(a)

```
inst "dff3/Q<0>" "SLICEL",placed R1C1 SLICE_X0Y30  ,
  cfg " BXINV::#OFF BYINV::#OFF CEINV::#OFF CLKINV::CLK COUTUSED::#OFF CY0F::#OFF
       CY0G::#OFF CYINIT::#OFF CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::1
       F::#OFF F5USED::#OFF FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
       FFY::#FF FFY_INIT_ATTR::INIT0 FFY_SR_ATTR::SRLOW FXMUX::#OFF
       FXUSED::#OFF G::#OFF GYMUX::G REVUSED::#OFF
       SRINV::SR_B SYNC_ATTR::SYNC XBUSED::#OFF XUSED::#OFF YBUSED::#OFF
       YUSED::#OFF "
  ;
```

(b)

Fig. 15. Part of (a) original XDL file and (b) recovered XDL file for 64-bit LFSR.



(a) (b)

Fig. 16. (a) Original circuit and (b) recovered circuit based on recovered XDL file

IV. EXPERIMENTAL RESULTS

The bitstream of the 64-bit LFSR circuit in Figure 14 implemented in an xc3s50 device among the Spartan-3 FPGAs was restored to an XDL file through the XDL recovery process using a mapping table. The 64-bit LFSR in Figure 15 mapped on xc3s50 device using 16 CLB tiles, 34 IOB tiles, and 1 BUFGMUX tile for GCLK. When the bitstream file representing a 64-bit LFSR in Figure 14 is compared with a raw bit file with no declaration of PLP and PIP, the position of a bit with a different bit representation must first be determined. Figure 15 shows part of the XDL file restored by performing XDL recovery based on the mapping table with the entire basic raw bit file. However, in the state immediately after the restoration based on the mapping table, the options having the same bit representation as the *#OFF* option cannot be restored. These options must be determined by comparing them with the options of other PLPs that appear in the bit representations. Finally, after recovering the PLP options that did not appear when compared with the default raw bit file, the original and restored XDL files are 88% identical to each other. The recovery ratio is calculated based on whether the number of restored 1's among all 1's in a bitstream. Although it is hardly to say 88% LFSR can works as an original LFSR, the recovered XDL definitely help to estimate the target design as a LFSR. When the generated XDL file is converted into an NCD file using the xdl2ncd command of the ISE design tool, Fig. 16 shows the result of mapping the converted NCD file to the circuit. The recovered netlist by the proposed reverse engineering is nearly perfect recovered from a graphical view point.

V. CONCLUSIONS

A reverse engineering tool focused on PIP and PLP recovery was implemented, and the process of reverse engineering and the operation of the automation tools required in each step were described. When a 64-bit LFSR was restored using reverse engineering tools and Xilinx ISE design tools focused on PIP and PLP restoration, 88% of the total circuits could be restored. Reverse engineering tools have been actively developed using the ISE design tool, and their recovery rates are up to 80%. Therefore, even if the bitstreams of the external memory are attacked, whether the circuit information has been damaged can be determined. For a further study, a reverse engineering tool for 7-series FPGA chips using Vivado will be developed to enlarge a reverse engineering area.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF-2019M3F3A1A01074448), and EDA tools were supported by IDEC, Korea .

REFERENCES

[1] H. Yu, H. Lee, S. Lee, Y. Kim, and H.-M. Lee, "Recent Advances in FPGA Reverse Engineering," Electronics, vol. 7, no. 10, 2018.

[2] M. Wirthlin, "High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond," *in Proceedings of the IEEE*, vol. 103, no. 3, pp. 379-389, March 2015.

[3] J.-B. Note and É. Rannaud, "From the bitstream to the netlist," in Proc. *16th Int. ACM/SIGDA Symp. FPGA*, vol. 8, pp. 264-264, 2008.

[4] F. Benz, A. Seffrin, and S. A. Huss, "Bil: A tool-chain for bitstream reverse-engineering," in *22th International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 735-738.

[5] Z. Ding, Q. Wu, Y. Zhang, and L. Zhu, "Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation," *Microprocessors and Microsystems*, vol. 37, no. 3, pp. 299-312, 2013.

[6] J. Yoon et al., "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, 2018.

[7] T. Zhang, J. Wang, S. Guo, and Z. Chen, "A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL Code," *IEEE Access*, vol. 7, pp. 38379-38389, 2019.

[8] Lavin, Christopher, et al. "Rapid prototyping tools for FPGA designs: RapidSmith." *2010 International Conference on Field-Programmable Technology. IEEE,* 2010.

[9] Lavin, Christopher, et al. "Rapidsmith: Do-it-yourself cad tools for xilinx fpgas." *2011 21st International Conference on Field Programmable Logic and Applications. IEEE*, 2011.

[10] Malhotra, Shawn, et al. "The quartus university interface program: enabling advanced fpga research." In Proceedings of *2004 IEEE International Conference on Field-Programmable Technology (IEEE Cat. No. 04EX921)*, 2004.

[11] M. Jeong, J. Lee, E. Jung, Y. H. Kim and K. Cho, "Extract LUT Logics from a Downloaded Bitstream Data in FPGA," *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, 2018, pp. 1-5.

[12] Moradi, Amir, et al. "On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs." Proceedings of the *18th ACM conference on Computer and communications security. ACM*, 2011.M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

So Yeon Choi received the B.S. degree in electronics engineering from Chungnam National University, Daejeon, Korea, in 2018.

Her main interests are VLSI for error correction codes and FPGA reconfiguration.

Ji Woon Park is working toward the B.S. degree in electrical engineering from Chungnam National University, Daejeon, Korea, in 2020.

His main interests are VLSI for error correction codes and FPGA reconfiguration.

Ho Young Yoo received the B.S. degree in electrical & electronics engineering from Yonsei University, Seoul, Korea, in 2010. He received the M.S. and Ph.D. degree in electronic engineering from KAIST in 2012 and 2016. Since 2016, he has been with the department of Electronics Engineering, Chungnam National University, Daejeon, Korea, where he is now an Assistant Professor.

His research interests are VLSI for 5G communication systems and VLSI for Machine Learning Accelerators.