*electronics*

*Article*

# Approaches to Extend FPGA Reverse-Engineering Technology from ISE to Vivado

**Soyeon Choi and Hoyoung Yoo ***

Department of Electronics Engineering, Chungnam National University, Daejeon 34134, Republic of Korea; soyeonchoi@cnu.ac.kr
* Correspondence: hyyoo@cnu.ac.kr

**Abstract:** SRAM-based FPGA(Field Programmable Logic Arrays) requires external memory since its internal memory gets erased when power is cut off. The process of transmitting the circuit netlist in bitstream from external memory during power-up in FPGA is vulnerable to malicious attacks such as bitstream theft and tampering. Previous FPGA reverse-engineering methods focus on FPGAs, supported by ISE (ISE Design Suite). This is because ISE provides XDLRC (Xilinx Design Language Routing Configurable logic) and XDL (Xilinx Design language) files, which are essential for reverse engineering. However, Vivado Design Suite (Vivado) does not offer those files, making it impossible to extend the coverage of reverse engineering to the FPGAs supported by Vivado. In this paper, we propose a method to generate XDLRC and XDL through Vivado. According to experimental results, the XDLRC and XDL generated through Vivado, respectively, match 99% and 75% with those generated in ISE for Artix-7 100T. As a result, this paper has expanded the scope of reverse engineering from being mainly focused on ISE to now also include Vivado. It is important to note that this paper does not encourage bitstream attacks through reverse engineering but rather highlights the risk associated with malicious attacks and emphasizes the importance of security.

**Keywords:** reverse engineering; Xilinx; Vivado design suite; ISE design suite; Xilinx design language file

**Citation:** Choi, S.; Yoo, H. Approaches to Extend FPGA Reverse-Engineering Technology from ISE to Vivado. *Electronics* **2024**, *13*, 1100. https://doi.org/10.3390/electronics13061100

Academic Editors: Raffaele Giordano and Alexander Barkalov

Received: 6 February 2024
Revised: 4 March 2024
Accepted: 14 March 2024
Published: 16 March 2024

## 1. Introduction

SRAM-based FPGAs, along with the Flash-based FPGA (Field Programmable Logic Array) and the Antifuse-based FPGA, play a significant role in various applications, such as signal processing, communication systems, image processing, control systems, encryption, and security [1]. Notably, AMD Xilinx and Intel Altera FPGAs demonstrate the highest market share in the SRAM-based FPGA industry [2]. FPGA chip manufacturers typically provide EDA tools to support circuit synthesis and implementation on FPGA chips, with Xilinx offering two such tools: ISE (ISE Design Suite) for low-power, low-end FPGAs [3], and Vivado (Vivado Design Suite) for the latest high-end FPGAs [4]. Specifically, ISE supports older FPGA series up to 7-series and some low-end 7-series FPGAs [3], while Vivado supports all 7-series and subsequent state-of-the-art FPGAs [4]. Table 1 summarizes the FPGA-specific support Design Suite for each Xilinx FPGA, where Virtex and Kintex series FPGAs generally have larger chip sizes and are fabricated using more advanced processes compared to Spartan and Artix series FPGAs [5].

Through the diverse FPGA chip portfolio provided by FPGA manufacturers and the stability of EDA tools, FPGA applications have expanded significantly. However, SRAM-based FPGAs have a critical drawback, which is that they require external memory since their internal memory gets erased when power is cut off [6]. The process of transmitting the netlist in bitstream format from external memory during power-up in FPGA systems makes it vulnerable to malicious attacks like bitstream theft and tampering, as shown in Figure 1 [7–9]. The complete extraction of the bitstream enables the potential for a cloning attack, and tampering with the hardware results in malfunctions, leading to potential damage. Additionally, intelligent reverse engineering [10–21] can be utilized to analyze the design of the netlist. When a circuit is attacked, the circuit does not behave as intended, causing serious problems. In the case of reverse engineering, all circuit information is exposed to the attacker, causing serious problems with circuit security.

**Table 1.** Design Suites available for each FPGA series.

| Family | Series | ISE | Vivado |
|---|---|---|---|
| Spartan | 3 | O | X |
| | 6 | O | X |
| | 7 | X | O |
| Virtex | 5 | O | X |
| | 6 | O | X |
| | 7 | Δ | O |
| | UltraScale | X | O |
| | UltraScale+ | X | O |
| Artix | 7 | O | O |
| | UltraScale+ | X | O |
| Kintex | 7 | O | O |
| | UltraScale | X | O |
| | UltraScale+ | X | O |

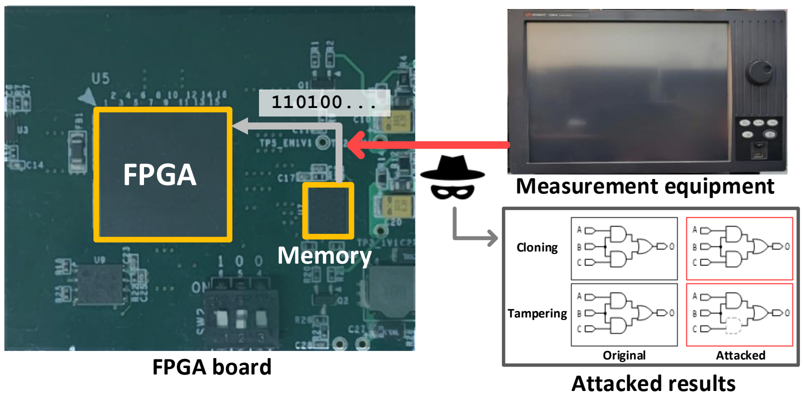


**Figure 1.** How to attack the FPGA system.

The previous FPGA reverse-engineering tools primarily focus on FPGAs supported by ISE, as shown in Table 1. The FPGAs that have been targeted by reverse engineering are mostly devices in the Spartan-3 series and Virtex-5 series in previous research [10–14]. This is because ISE provides of readable XDLRC (Xilinx Design Language Routing Configurable logic) and XDL (Xilinx Design language) files [22]. Note that the XDLRC file is a hardware structure file, and the XDL file is the netlist file provided by ISE. Based on XDLRC, the previous FPGA reverse-engineering techniques [10–14] identify the association between XDL and bitstream. The association is converted into a database that is used for reverse engineering. All hardware elements in the FPGA described in XDLRC appear in the bitstream, and the value is 1 in the bitstream only for the hardware elements used in XDL. The database is generated by modifying the XDL files that applying information on XDLRC and comparing bitstreams. Therefore, XDL and XDLRC files are essential for securing a database for reverse engineering. Although some reverse-engineering techniques using Vivado have been announced [15–21], the techniques are very limited and still at a rudimentary level. The reason the latest FPGA reverse-engineering techniques in Vivado are restricted and disturbed is the absence of textual netlist files such as XDLRC and XDL. Consequently, this paper proposes a method to generate XDLRC and XDL files in Vivado, similar to those in ISE, to extend the coverage of the previous reverse-engineering techniques. It is important to note that this paper does not encourage bitstream attacks through reverse engineering but rather highlights the risk associated with malicious attacks and emphasizes the importance of security. The remaining sections of this paper are organized as follows: Section 2 explains the structure of Xilinx FPGAs and the design flow of ISE and Vivado. In Section 3, we verify XDL and XDLRC files generated using ISE and propose a method to generate them using Vivado. Section 4 analyzes the differences between the files generated by the two EDA tools. Finally, Section 5 presents the conclusions of this paper.

## 2. Background

In Xilinx FPGA devices, the XDLRC file represents a textual description of all available hardware resources within the FPGA device, while the XDL file specifically details the activated resources among the entire hardware [22]. To properly comprehend XDLRC and XDL files, it is fundamental to grasp the fundamental structure of Xilinx FPGA devices. In this chapter, we provide a comprehensive overview of the inherent hierarchical structure of Xilinx FPGA architectures. Following that, we outline the sequential procedures for circuit synthesis, implementation, and bitstream generation using both the ISE and Vivado.

### *2.1. Structure of Xilinx FPGA*

Xilinx FPGAs feature a hierarchical structure for their hardware resources, which can be simplified as depicted in Figure 2 [23–30]. The interior of the FPGA is composed of tiles. The positions of these tiles are defined using Cartesian coordinates ($X$, $Y$). The interconnections between tiles are fixed, and these static connections within the FPGA are referred to as conn. The input and output ports of the tiles are defined as wires, and the internal connections between wires are denoted as PIPs (Programmable Interconnect Points). Each wire can be connected to one or more PIPs, with the appropriate PIPs selected depending on the circuit implementation.

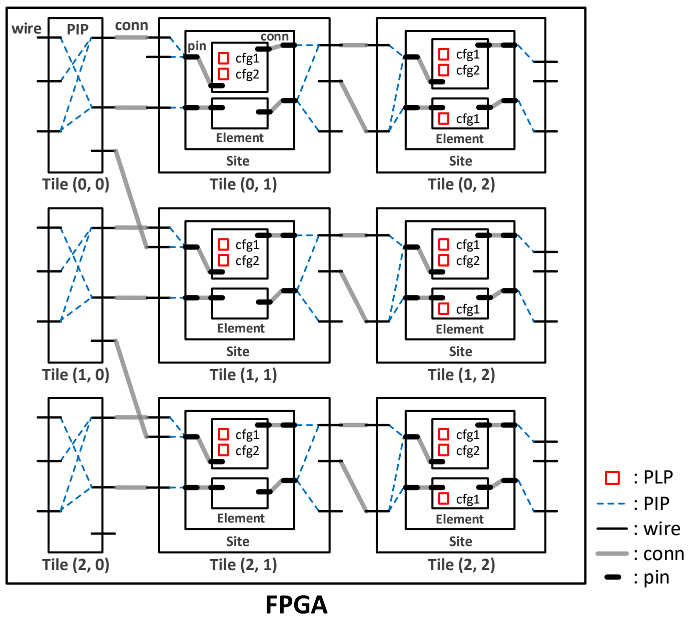


**Figure 2.** Hierarchical structure of FPGA.

Within each tile, there is a sub-level block known as a *site*. The input and output ports of the site are defined as *pin*, and the connections between pins are also represented as *conn*. Inside the site, there is another sub-level block called an *element*. Like the site, the input and output ports of the element are composed of *pins*. Elements can either include or exclude a configurable logical option referred to as *cfg*. Among elements that include *cfg*, those responsible for constructing logic elements such as MUXs (Multiplexers) are called PLPs

(Programmable Logic Points), while those supporting data storage functions like LUTs (Look-Up Tables) are termed PDPs (Programmable Data Points).

### 2.2. *ISE Design Flow*

The design flow in ISE, from circuit synthesis to implementation, is depicted in Figure 3. As shown in Figure 3, the ISE design flow comprises a total of five stages, each executed using Tcl commands. The process begins with synthesis, which requires HDL source files, including the design, and a UCF (User Constraint File) file specifying constraints. Once these two files are prepared, synthesis is initiated using the '*xst*' command. After synthesis, an NGC (Native Generic Constraint) file is generated, combining the design and constraints.

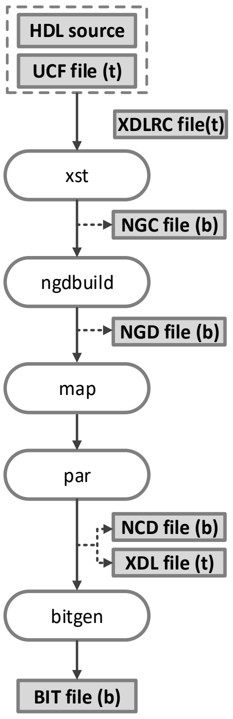


**Figure 3.** ISE design flow.

Following that, the '*ngdbuild*' stage is executed, which assigns the information described in the NGC file to the physical resources of the FPGA, generating the NGD (Native Generic Database) file, which includes routing and timing details in the form of a netlist. After this, during the '*map*' stage, the physical resources outlined in the NGD file are mapped to real hardware components such as LUTs, MUXs, and BRAMs (Block RAMs). Finally, when the '*par*' stage, responsible for P&R (Placement and Routing), is executed, it results in the generation of the netlist files, the NCD (Native Circuit Description) file, and the XDL file. While netlist files such as NGC and NGD, including the NCD file, are all in a binary format, the XDL file is in textual format, making it user-readable. Once the P&R process is completed, the netlist files can be transformed into a BIT file, which can be programmed onto the FPGA, via the '*bitgen*' stage. It is noteworthy that, within the ISE, the XDLRC file, describing all available hardware resources within the FPGA, can be generated at any stage of the design flow after project creation.

### 2.3. *Vivado Design Flow*

Vivado, on the other hand, follows a design flow consisting of a total of seven stages, as depicted in Figure 4. In the Vivado design flow, it is essential to have both the HDL source, comprising the design, and the XDC file, detailing the constraints. Once these two files are prepared, circuit synthesis is performed using the '*synth_design*' command.

After synthesis, the implementation proceeds through five stages. As the initial step of implementation, '*opt_design*' is performed to optimize the synthesis results, followed by '*power_opt_design*' to further optimize from a power perspective.

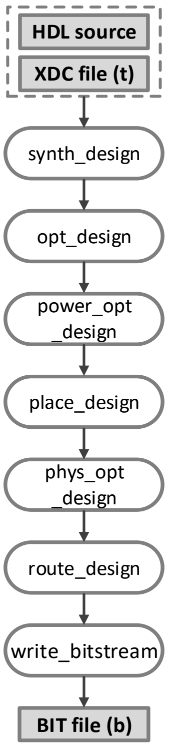


**Figure 4.** Vivado design flow.

Following that, '*place_design*' is responsible for positioning the optimized design onto the FPGA hardware resources, while '*phys_opt_design*' focuses on optimization with regard to physical placement. Afterward, '*route_design*' manages routing, and the '*write_bitstream*' stage generates the BIT file that can be programmed onto the FPGA. It is important to note that no netlist files, including XDL files, are generated during the synthesis and implementation stages in Vivado design flow, and XDLRC files are also not provided.

## 3. Netlist File Generation

The XDLRC is essential for understanding the hardware structure, and the XDL shows that the hardware elements appear in the bitstream as a readable netlist. These two files are extracted from ISE with a single Tcl command in ISE, but in Vivado, they are extracted with multiple Vivado Tcl commands. This section describes how to extract XDLRC and XDL files from ISE and Vivado.

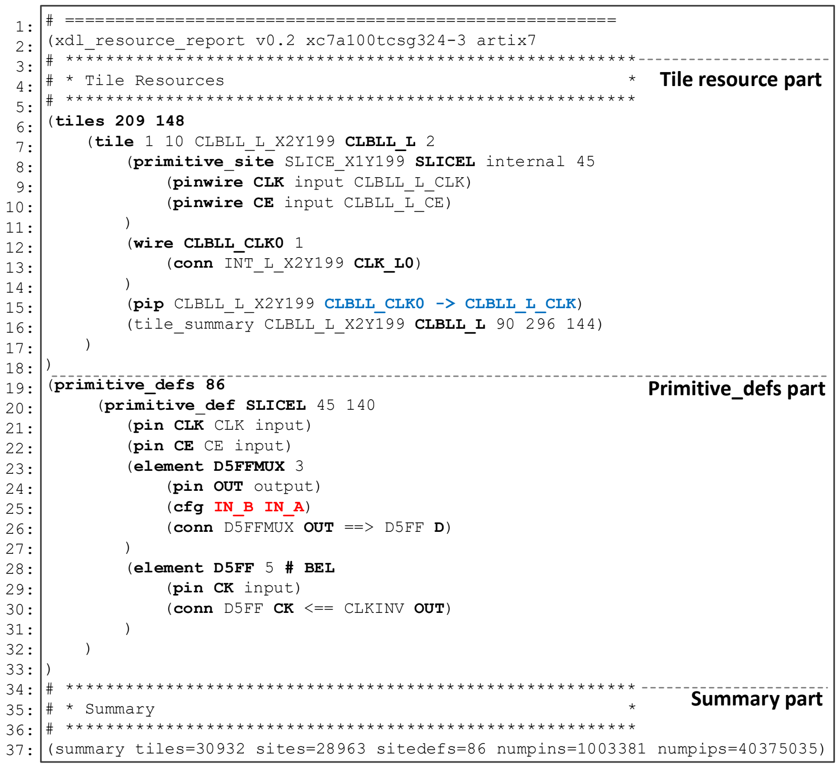
### *3.1. XDLRC File Generated by ISE*

An XDLRC file can be generated using a Tcl command in ISE, and the Tcl command to create an XDLRC file is as follows:

*xdl -report [-pips] [-all_conns] <part> [<outfile name>]*

The options [-*pips*] and [-*all_conns*] are used with the '*xdl -report*' command. [-*pips*] generates a report containing pip routing information, while [-*all_conns*] displays all connections to a tile wire, regardless of the containing adjacent tile. The <part> parameter represents the FPGA device, and it should include the speed grade and package details. [<*outfile name*>] is an optional parameter; if not specified, the XDLRC file will be created with a default name following the pattern '<*part*>.*xdlrc*'.

As an example in this paper, the Artix-7 100t device, with a speed grade of −1 and 324 external I/O pads in the csg324 package, is used. If you want to generate an XDLRC file to list the tiles and sites included for this device, '*xdl -report xc7a100t-1csg324*' should be executed to generate the XDLRC file. However, if you wish to obtain comprehensive information, including all pips, wires, and conns for the same device, '*xdl-report -pips -all_conns xc7a100t-1csg324*' should be executed to generate the XDLRC file. Figure 5 provides an example of an XDLRC file generated using the options [-pips] and [-all_conns]. The XDLRC file, as seen in Figure 5, consists of three parts: the tile resource part, the primitive_defs part, and the summary part.


Tile resource part
Primitive_defs part
Summary part

```
 1: # =======================================================
 2: (xdl_resource_report v0.2 xc7a100tcsg324-3 artix7
 3: # **********************************************************
 4: # * Tile Resources                                         *
 5: # **********************************************************
 6: (tiles 209 148
 7:     (tile 1 10 CLBLL_L_X2Y199 CLBLL_L 2
 8:         (primitive_site SLICE_X1Y199 SLICEL internal 45
 9:             (pinwire CLK input CLBLL_L_CLK)
10:             (pinwire CE input CLBLL_L_CE)
11:         )
12:         (wire CLBLL_CLK0 1
13:             (conn INT_L_X2Y199 CLK_L0)
14:         )
15:         (pip CLBLL_L_X2Y199 CLBLL_CLK0 -> CLBLL_L_CLK)
16:         (tile_summary CLBLL_L_X2Y199 CLBLL_L 90 296 144)
17:     )
18: )
19: (primitive_defs 86
20:      (primitive_def SLICEL 45 140
21:         (pin CLK CLK input)
22:         (pin CE CE input)
23:         (element D5FFMUX 3
24:             (pin OUT output)
25:             (cfg IN_B IN_A)
26:             (conn D5FFMUX OUT ==> D5FF D)
27:         )
28:         (element D5FF 5 # BEL
29:             (pin CK input)
30:             (conn D5FF CK <== CLKINV OUT)
31:         )
32:     )
33: )
34: # **********************************************************
35: # * Summary                                                *
36: # **********************************************************
37: (summary tiles=30932 sites=28963 sitedefs=86 numpins=1003381 numpips=40375035)
```

**Figure 5.** XDLRC generated by ISE.

The Tile resource part represents all of the FPGA hardware resources at the tile level, starting from the tile at position (0, 0) and describing each tile position, tile name, and tile type. In Figure 5, line 7 begins to depict the internal structure of the tile located at (1, 10). The numbers *1* and *10* represent the *X* and *Y* coordinates of the tile, respectively. The subsequent *CLBLL_L_X2Y199* indicates the tile name, with *CLBLL_L* denoting the tile type. The number, 2, signifies the number of sites contained within this tile. Line 8 provides information about the sites configured within the tile. The sites are declared as primitive_site. The *SLICE_X1Y199* indicates the site name, *SLICEL* represents the site type, and *internal 45* specifies that the site contains 45 pins. Lines 9 and 10 detail the site pins and wires. In the tile resource part, the internal structure of the tile is described by only the site pins. Following primitive_site, the wires associated with the tile are listed. At a lower level of wire, the conn describes how each wire is linked, specifying which tile wire it connects to. Therefore, through lines 12 and 13, we can discern that the *CLBLL_CLK0* wire is connected to the *CLK_L0* wire of *INT_L_X2Y199* via the conn. The last line representing the tile includes the tile name, type, the number of pinwires, wires, and pips present in the tile.

In the primitive_defs part, the complete internal structures of all sites included in this device are shown. Line 19 marks the beginning of revealing the internal structure of the *SLICEL* site, with the numbers *45* and *140* following *SLICEL* signifying the respective counts of input/output pins and elements within the site. Each pin in the section describes the pin name, the associated pinwire name from the tile resource part, and whether the pin functions as an input or output represented in line 21 and line 22. The details of the elements appear, as seen in line 23, where the element name and a number denoting the number of pins within the element are provided. If there is a configuration present representing *cfg* in Figure 6, all configurations of the element are listed, like line 25. The *conn* of the elements represent the connections between pins of the elements. For instance, line 26 signifies the connection between the *OUT* pin of the *D5FFMUX* element and the *D* pin of the *D5FF* element. When the line beginning with the element name in the primitive_defs part is suffixed with # *BEL*, it indicates that this element serves as a basic element as representing *bel* in Figure 6, such as an LUT or an FF(Flip-Flop).

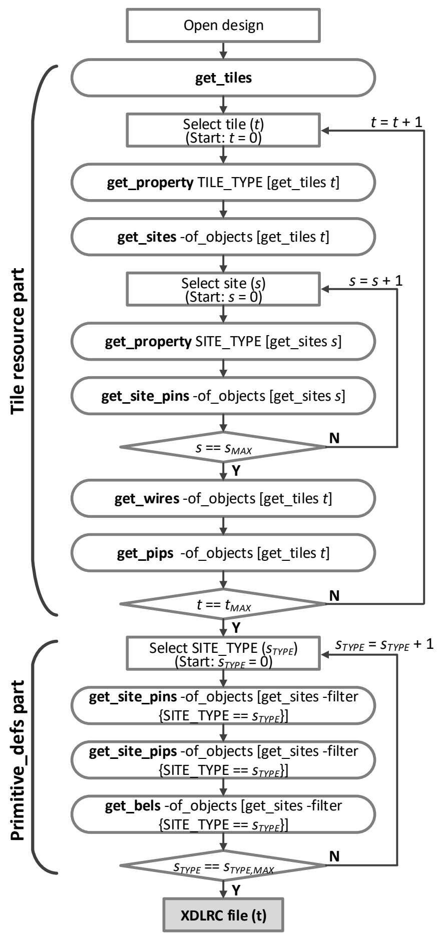

Open design
get_tiles
Select tile (t) (Start: t = 0)
t = t + 1
get_property TILE_TYPE [get_tiles t]
get_sites -of_objects [get_tiles t]
Select site (s) (Start: s = 0)
s = s + 1
get_property SITE_TYPE [get_sites s]
get_site_pins -of_objects [get_sites s]
s == s_MAX
N
Y
get_wires -of_objects [get_tiles t]
get_pips -of_objects [get_tiles t]
t == t_MAX
N
Y
Select SITE_TYPE (s_TYPE) (Start: s_TYPE = 0)
s_TYPE = s_TYPE + 1
get_site_pins -of_objects [get_sites -filter {SITE_TYPE == s_TYPE}]
get_site_pips -of_objects [get_sites -filter {SITE_TYPE == s_TYPE}]
get_bels -of_objects [get_sites -filter {SITE_TYPE == s_TYPE}]
s_TYPE == s_TYPE,MAX
N
Y
XDLRC file (t)
Tile resource part
Primitive_defs part

**Figure 6.** Flow chart for generating XDLRC with Vivado.

In the summary part, details regarding the number of tiles, sites, site types, site input/output pin counts, and the number of PIPs encompassed by this device are presented.

## *3.2. XDLRC File Generated by Vivado*

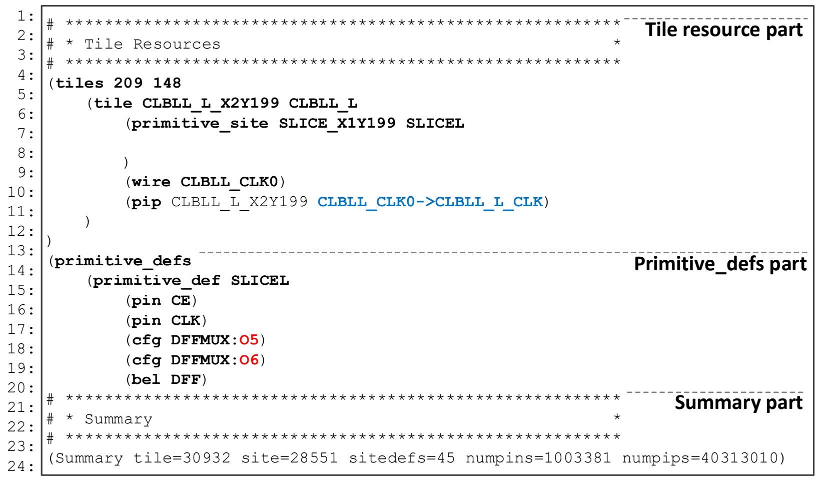
In Vivado, there is no Tcl command like '*xdl-report*' to generate an XDLRC file. However, through Vivado Tcl commands supported by Vivado, it is possible to obtain information about FPGA hardware resources and create an XDLRC file.

To construct the tile resource part of the XDLRC file, information about tile names, tile types, sites within the tile, wires and conns contained within the tile, and pips are required. To access this information, commands such as '*get_tiles*', '*get_sites*', '*get_wires*', and '*get_pips*' are demanded. These commands alone provide only the names of each component.

Identifying tile types and site types corresponds to particular attributes referred to as TILE_TYPE and SITE_TYPE. To access these properties, the '*get_property*' command should be employed. It is important to note that extracting information about fixed connections conn with Tcl commands in Vivado is not feasible.

The primitive_defs part, which describes the internal structure of each site type, requires information about the pins within the site, the *cfg* of elements, *conn*, and whether an element is a *bel*. Information regarding the pins within a site can be obtained through the use of the '*get_site_pins*' command. The configuration of an element, which represents the programmable points within the site, can be obtained using the '*get_site_pips*' command. Elements that are *bels* can be distinguished using '*get_bels*'. However, it is important to note that extracting conn, similar to the tile resource part, is not feasible through available commands.

Figure 6 illustrates the process of generating an XDLRC file in Vivado as a flowchart. To create the tile resource part, the process of extracting the required information for each tile is repeated for all tiles. For the primitive_defs part, the process of discovering internal site details is repeated for each site type. The summary part determines the count of each component using the Tcl command '*llength*'. Through these steps, the XDLRC file was created in Vivado, as shown in Figure 7.



```
# *********************************************************
# * Tile Resources                                        *
# *********************************************************
(tiles 209 148
    (tile CLBLL_L_X2Y199 CLBLL_L
        (primitive_site SLICE_X1Y199 SLICEL

        )
        (wire CLBLL_CLK0)
        (pip CLBLL_L_X2Y199 CLBLL_CLK0->CLBLL_L_CLK)
    )
)
(primitive_defs
    (primitive_def SLICEL
        (pin CE)
        (pin CLK)
        (cfg DFFMUX:O5)
        (cfg DFFMUX:O6)
        (bel DFF)
# *********************************************************
# * Summary                                               *
# *********************************************************
(Summary tile=30932 site=28551 sitedefs=45 numpins=1003381 numpips=40313010)
```

Tile resource part
Primitive_defs part
Summary part

**Figure 7.** XDLRC generated by Vivado.

### 3.3. XDL File Generated by ISE

The TCL command for creating an XDL file using ISE is as follows:

*xdl -ncd2xdl [-nopips] <ncdfile name> [<xdlfile name>]*

[-*nopips*] is an option for the '*xdl-ncd2xdl*' command that suppresses the reporting of PIPs, and <*ncdfile name*> is the name of the netlist file generated after performing P&R, typically matching the top module name. [<*xdlfile name*>] is an optional parameter to specify the output file name. If the output file name is not specified, an XDL file with the same name as the NCD file is generated. For instance, if the top module name is 'TestDesign' and you want to generate an XDL file for this module with the name 'TestDesign_v1.xdl', the Tcl command should be written as follows: '*xdl -ncdtoxdl TestDesign.ncd TestDesign_v1.xdl*'. An example of an XDL file generated using the '*xdl-ncd2xdl*' command in ISE is illustrated in Figure 8 and is structured in four parts: the design part, the instance part, the net part, and the summary part. In the design part, information such as the design name, device type, and the version of ISE are presented.

The instance part displays hardware resources used in the design implementation at the cell. A cell can be either a primitive or a hierarchical instance within a netlist. Examples of cells include FFs, LUTs, I/O buffers, and hierarchical instances. The first line representing information about a single cell, as seen in line 11 of Figure 8, and includes the cell name, the site type and site name where the cell is implemented, and the name of the tile where it is located. Following the keyword *cfg* on the next line are all PLPs contained within the site, used for the circuit implementation, listed along with their respective configurations. Unused PLPs are marked with *#OFF* in the configuration string, while PDPs display data stored as Boolean functions.

```
 1: # =======================================================
 2: # design <design_name> <part> <ncd version>;                       Design part
 3: # =======================================================
 4: design "lut_6input_0" xc7a100tcsg324-1 v3.2 ,
 5:   cfg "
 6:        _DESIGN_PROP:P3_PLACE_OPTIONS:EFFORT_LEVEL:high
 7:        _DESIGN_PROP::PK_NGMTIMESTAMP:1618248090";
 8: #  =======================================================
 9: #      instance <name> <sitedef>, placed <tile> <site>, cfg <string> ;  Instance part
10: #  =======================================================
11: inst "O_OBUF" "SLICEL",placed CLBLL_L_X2Y145 SLICE_X0Y145  ,
12:   cfg " A5FFINIT::#OFF A5FFMUX::#OFF A5FFSR::#OFF A5LUT::#OFF
13:        A6LUT:LUT6_inst:#LUT:O6=(~A6*(~A1*(~A2*(~A3*(~A4*~A5)))))
14:        ACY0::#OFF AFF::#OFF AFFINIT::#OFF AFFMUX::#OFF AFFSR::#OFF AOUTMUX::#OFF
15:        AUSED::0 CLKINV::#OFF COUTMUX::#OFF COUTUSED::#OFF CUSED::#OFF
16:        PRECYINIT::#OFF SRUSEDMUX::#OFF SYNC_ATTR::#OFF "
17:   ;
18: inst "XDL_DUMMY_OLOGIC_X0Y149" "OLOGICE3",placed LIOI3_SING_X0Y149 OLOGIC_X0Y149  ,
19:   cfg "_NO_USER_LOGIC::  _ROUTETHROUGH:D1:OQ "
20:   ;
21: #  ================================================
22: #     net <name> <type>,                                            Net part
23: #       outpin <inst_name> <inst_pin>,
24: #       inpin <inst_name> <inst_pin>,
25: #       pip <tile> <wire0> <dir> <wire1> , # [<rt>]
26: #       ;
27: #  ================================================
28: net "O" , cfg " _BELSIG:PAD,PAD,O:O",
29:   ;
30: net "O_OBUF" ,
31:   outpin "O_OBUF" A ,
32:   inpin "O" O ,
33:   pip CLBLL_L_X2Y145 CLBLL_LL_A -> CLBLL_LOGIC_OUTS12 ,
34:   pip INT_L_X2Y145 LOGIC_OUTS_L12 -> NW6BEG0 ,
35:   pip LIOI3_SING_X0Y149 IOI_IMUX34_0 -> IOI_OLOGIC0_D1 ,
36:   pip LIOI3_SING_X0Y149 IOI_OLOGIC0_D1 -> LIOI_OLOGIC0_OQ ,  #  _ROUTETHROUGH:D1:OQ
37:   pip LIOI3_SING_X0Y149 LIOI_OLOGIC0_OQ -> LIOI_O0 ,
38:   ;
39: # =======================================================
40: # SUMMARY                                                      Summary part
41: # Number of Module Defs: 0
42: # Number of Module Insts: 0
43: # Number of Primitive Insts: 8
44: # Number of Nets: 14
45: # =======================================================
```

**Figure 8.** XDL generated by ISE.

The net part encompasses all the necessary nets for circuit implementation. Most of these nets, as seen in the green lines of Figure 9, represent a collection of all the PIPs required to connect an outpin from site A to an inpin in site *C*. In this context, the connections along this path, conn, are not explicitly detailed in the net part since their routing is predetermined. Therefore, when these nets are represented in XDL, each green line is described with one outpin, one inpin, and three PIPs. For instance, let us consider the *O_OBUF* net from Figure 8 when examining Figure 9. In this case, pin *A* in line 31 of Figure 8 corresponds to the outpin of site *A* in Figure 9, and pin *O* in line 32 of Figure 8 corresponds to the inpin of site *C* in Figure 9. These two pins are interconnected by the five PIPs detailed from line 33 to line 37 in Figure 8. However, nets connected to input and output pads, as illustrated in line 28 and line 29 of Figure 8, provide information solely about the pads themselves.

## *3.4. XDL File Generated by Vivado*

To generate an XDL file in Vivado, several Vivado Tcl commands that allow obtaining the necessary information, like with XDLRC, should be used. To obtain the necessary information for the design part in Vivado, the following commands are required: '*find_top*' for retrieving the top design name, '*get_parts*' for obtaining the FPGA device name, and version to fetch the tool version. For the instance part, the cell names can be identified using the '*get_cells*' command. Information about the site type, site name, and tile name containing the cell can be acquired, similar to XDLRC, by utilizing the '*get_property*', '*get_sites*', and '*get_tiles*' commands. The '*get_site_pips*' command should be used to obtain information about the PLPs within a site, ensuring the inclusion of the '*-filter {IS_USED}*' option to focus on the activated logic necessary for circuit implementation. The data stored in PDP can be revealed in hexadecimal format using the '*get_property*' INIT command.

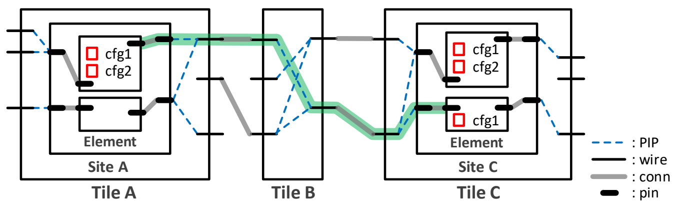


**Figure 9.** Example of a net.

The information needed for the net part can be acquired using the following commands: '*get_nets*' for net names, '*get_pins*' for pins, and '*get_pips*' for PIPs. When obtaining information about pins and PIPs, like with PLP, it is essential to include *the* '*-filter {IS_USED}*' option to extract only the active pins and PIPs, similar to in the instance part.

Figure 10 represents the flowchart of the XDL file generation process in Vivado. It involves obtaining information about the design part, followed by iterating through the process of obtaining all the information about cells and nets required for the instance part and the net part. Through this process, the XDL file generated in Vivado appears, as shown in Figure 11.

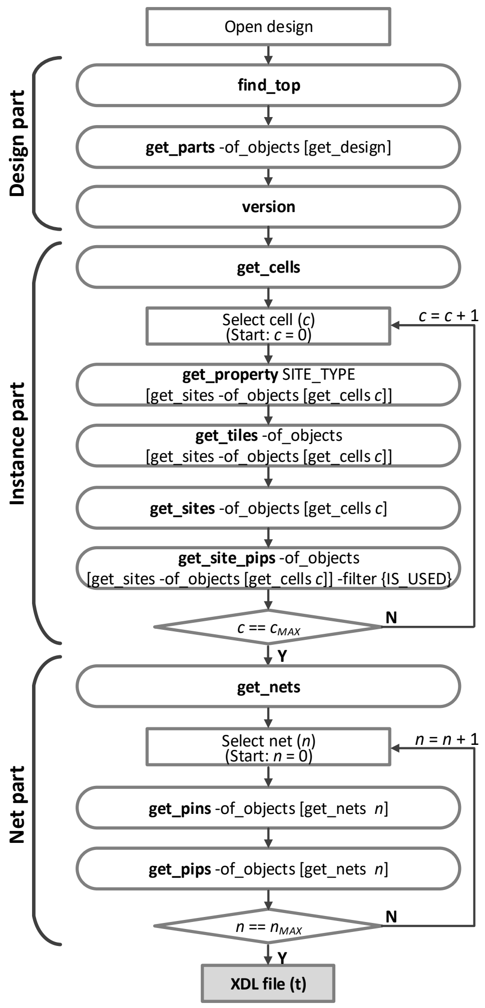
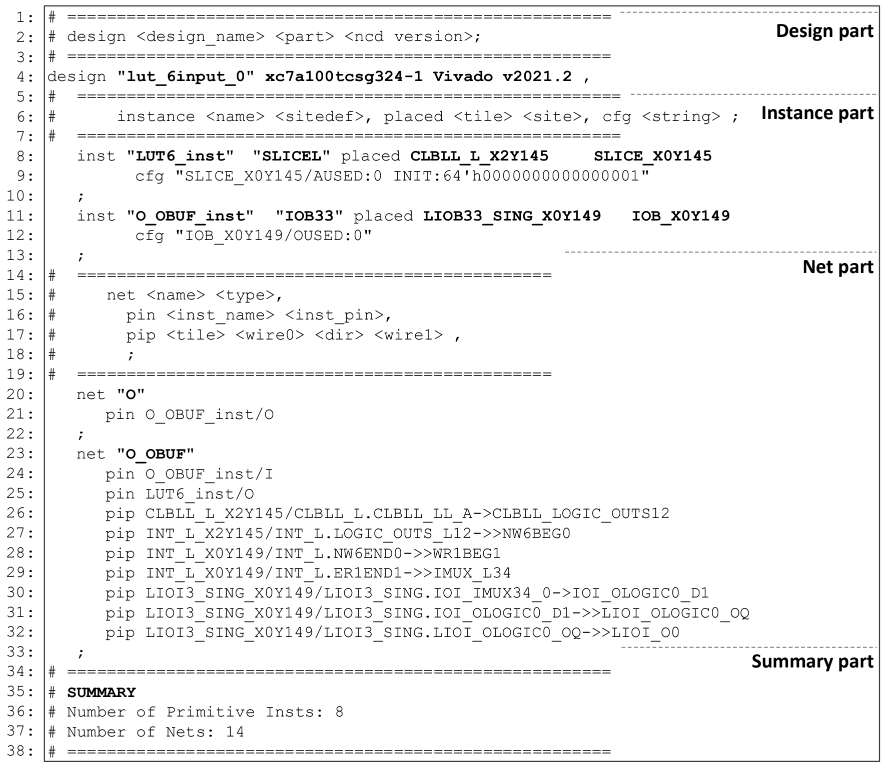

Open design
Design part
find_top
get_parts -of_objects [get_design]
version
Instance part
get_cells
Select cell (c) (Start: c = 0)
c = c + 1
get_property SITE_TYPE [get_sites -of_objects [get_cells c]]
get_tiles -of_objects [get_sites -of_objects [get_cells c]]
get_sites -of_objects [get_cells c]
get_site_pips -of_objects [get_sites -of_objects [get_cells c]] -filter {IS_USED}
c == c_MAX
N
Y
Net part
get_nets
Select net (n) (Start: n = 0)
n = n + 1
get_pins -of_objects [get_nets n]
get_pips -of_objects [get_nets n]
n == n_MAX
N
Y
XDL file (t)

**Figure 10.** Flow chart for generating XDL with Vivado.


Design part
Instance part
Net part
Summary part

```
# ======================================================
# design <design_name> <part> <ncd version>;
# ======================================================
design "lut_6input_0" xc7a100tcsg324-1 Vivado v2021.2 ,
#   ======================================================
#       instance <name> <sitedef>, placed <tile> <site>, cfg <string> ;
#   ======================================================
   inst "LUT6_inst"  "SLICEL" placed CLBLL_L_X2Y145     SLICE_X0Y145
         cfg "SLICE_X0Y145/AUSED:0 INIT:64'h0000000000000001"
   ;
   inst "O_OBUF_inst"  "IOB33" placed LIOB33_SING_X0Y149   IOB_X0Y149
         cfg "IOB_X0Y149/OUSED:0"
   ;
#   ================================================
#      net <name> <type>,
#        pin <inst_name> <inst_pin>,
#        pip <tile> <wire0> <dir> <wire1> ,
#        ;
#   ================================================
   net "O"
      pin O_OBUF_inst/O
   ;
   net "O_OBUF"
      pin O_OBUF_inst/I
      pin LUT6_inst/O
      pip CLBLL_L_X2Y145/CLBLL_L.CLBLL_LL_A->CLBLL_LOGIC_OUTS12
      pip INT_L_X2Y145/INT_L.LOGIC_OUTS_L12->>NW6BEG0
      pip INT_L_X0Y149/INT_L.NW6END0->>WR1BEG1
      pip INT_L_X0Y149/INT_L.ER1END1->>IMUX_L34
      pip LIOI3_SING_X0Y149/LIOI3_SING.IOI_IMUX34_0->IOI_OLOGIC0_D1
      pip LIOI3_SING_X0Y149/LIOI3_SING.IOI_OLOGIC0_D1->>LIOI_OLOGIC0_OQ
      pip LIOI3_SING_X0Y149/LIOI3_SING.LIOI_OLOGIC0_OQ->>LIOI_O0
   ;
# ======================================================
# SUMMARY
# Number of Primitive Insts: 8
# Number of Nets: 14
# ======================================================
```

**Figure 11.** XDL generated by Vivado.

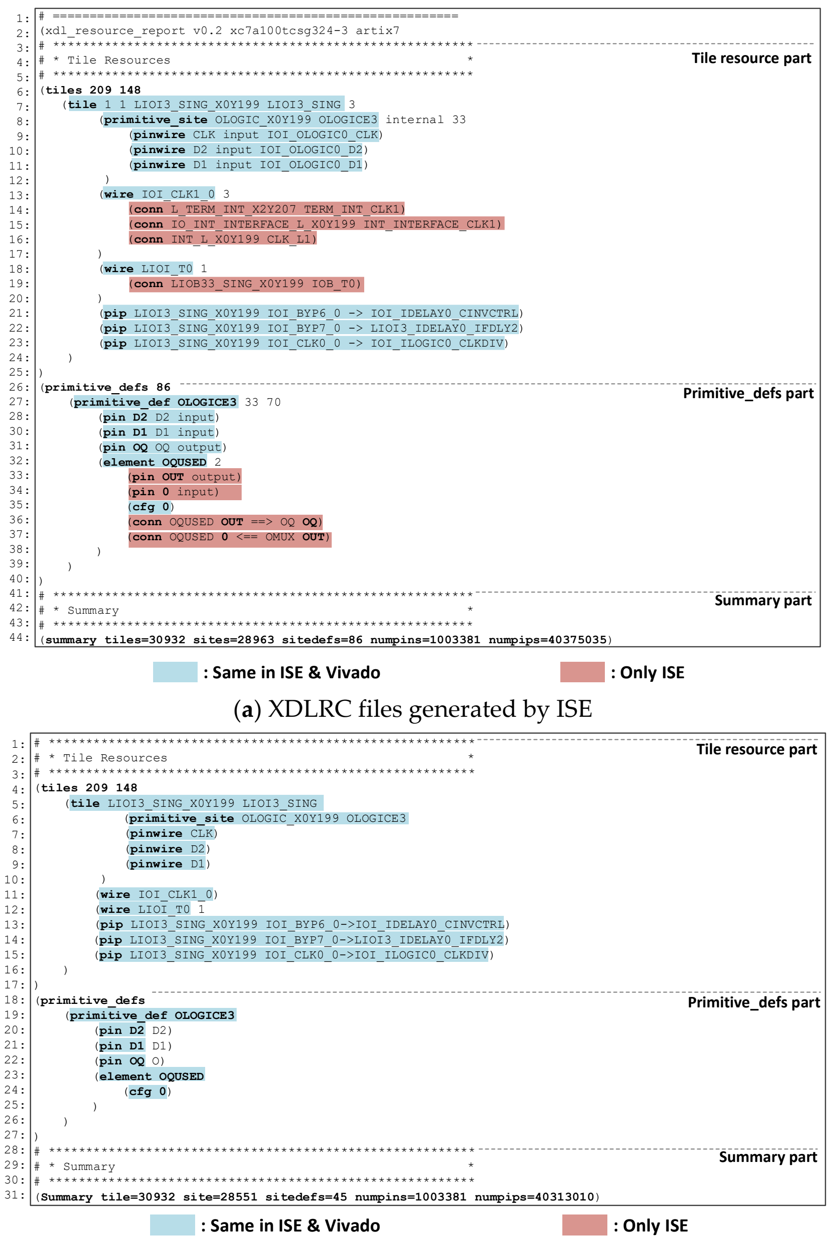
## 4. Analysis

In this paper, experiments were conducted using the Artix-7 100T device with a speed grade of −1 and the csg324 package. ISE version 14.7 and Vivado version 2020.2 were employed for the generation of both XDL and XDLRC files.

### *4.1. Comparison of XDLRC*

If you create XDLRC with two EDA tools for the same FPGA, as shown in Figure 12, Figure 12a shows the XDLRC file generated by ISE, and Figure 12b shows part of the XDLRC file generated by Vivado. The components shown highlighted in blue in Figure 12 mean that they appear exactly the same in both files. In the tile resource part, the tile name, the tile type, the site type that constitutes the tile, the *wire* of the tile, and the PIP inside the tile are the same. In the primary_defs part, the site type, the site pin, and the elements including PLP or PDP inside the site are the same. On the other hand, in Figure 12, the component highlighted in red, the *conn* of the element, and the *pin* of the element can be checked in the XDLRC file generated by ISE with fixed connections, but not in the XDLRC file generated by Vivado, which only shows information related to programmable points. For example, it was figured out that the *LIOI3_SING_X0Y199* tile contains one primitive_site, *OLOGIC_X0Y199*, with three inputs, including *CLK*, *D2*, and *D1* and two wires and three PIPs through Figure 12a,b, while three *conns* of the wire *IOI_CLK1_0* and one conn of the wire *LIOI_T0* are only seen in XDLRC extracted from ISE, as shown in Figure 12a. The components contained in the primitive_site OLOGIC_X0Y199 appear the same in Figure 12a,b except for *conn* and *pin* of the element.

For most tiles, the basic structure appears the same in the XDLRC files generated by both EDA tools, as shown in Figure 12. However, sites that do not contain PLP or PDP cannot be verified in the XDLRC file generated by Vivado. In addition, if the connection between start wire and end wire is determined to be only one of the PIPs, the PIPs cannot also be confirmed in the XDLRC generated by Vivado because the connection operates as fixed. In summary, only when programmable points among the components are identified in the XDLRC file generated by ISE can the XDLRC generated by Vivado be verified.


Tile resource part
Primitive_defs part
Summary part

```
# ======================================================
(xdl_resource_report v0.2 xc7a100tcsg324-3 artix7
# **********************************************************
# * Tile Resources                                         *
# **********************************************************
(tiles 209 148
   (tile 1 1 LIOI3_SING_X0Y199 LIOI3_SING 3
        (primitive_site OLOGIC_X0Y199 OLOGICE3 internal 33
            (pinwire CLK input IOI_OLOGIC0_CLK)
            (pinwire D2 input IOI_OLOGIC0_D2)
            (pinwire D1 input IOI_OLOGIC0_D1)
         )
        (wire IOI_CLK1_0 3
            (conn L_TERM_INT_X2Y207 TERM_INT_CLK1)
            (conn IO_INT_INTERFACE_L_X0Y199 INT_INTERFACE_CLK1)
            (conn INT_L_X0Y199 CLK_L1)
        )
        (wire LIOI_T0 1
            (conn LIOB33_SING_X0Y199 IOB_T0)
        )
        (pip LIOI3_SING_X0Y199 IOI_BYP6_0 -> IOI_IDELAY0_CINVCTRL)
        (pip LIOI3_SING_X0Y199 IOI_BYP7_0 -> LIOI3_IDELAY0_IFDLY2)
        (pip LIOI3_SING_X0Y199 IOI_CLK0_0 -> IOI_ILOGIC0_CLKDIV)
    )
)
(primitive_defs 86
    (primitive_def OLOGICE3 33 70
        (pin D2 D2 input)
        (pin D1 D1 input)
        (pin OQ OQ output)
        (element OQUSED 2
            (pin OUT output)
            (pin 0 input)
            (cfg 0)
            (conn OQUSED OUT ==> OQ OQ)
            (conn OQUSED 0 <== OMUX OUT)
        )
    )
)
# **********************************************************
# * Summary                                                *
# **********************************************************
(summary tiles=30932 sites=28963 sitedefs=86 numpins=1003381 numpips=40375035)
```

🟦 : Same in ISE & Vivado 🟥 : Only ISE

(**a**) XDLRC files generated by ISE

```
# **********************************************************
# * Tile Resources                                         *
# **********************************************************
(tiles 209 148
    (tile LIOI3_SING_X0Y199 LIOI3_SING
            (primitive_site OLOGIC_X0Y199 OLOGICE3
            (pinwire CLK)
            (pinwire D2)
            (pinwire D1)
         )
        (wire IOI_CLK1_0)
        (wire LIOI_T0 1
        (pip LIOI3_SING_X0Y199 IOI_BYP6_0->IOI_IDELAY0_CINVCTRL)
        (pip LIOI3_SING_X0Y199 IOI_BYP7_0->LIOI3_IDELAY0_IFDLY2)
        (pip LIOI3_SING_X0Y199 IOI_CLK0_0->IOI_ILOGIC0_CLKDIV)
    )
)
(primitive_defs
    (primitive_def OLOGICE3
        (pin D2 D2)
        (pin D1 D1)
        (pin OQ O)
        (element OQUSED
            (cfg 0)
        )
    )
)
# **********************************************************
# * Summary                                                *
# **********************************************************
(Summary tile=30932 site=28551 sitedefs=45 numpins=1003381 numpips=40313010)
```

🟦 : Same in ISE & Vivado 🟥 : Only ISE

(**b**) XDLRC files generated by Vivado

**Figure 12.** XDLRC files generated by (**a**) ISE and (**b**) Vivado.

## 4.2. Comparison of XDL

To generate XDL files, an RTL design is essential. In this paper, a design instantiated with one six-input LUT primitive, as shown in Figure 13, is used as an example design to generate XDL files. The six-input LUT primitive, which employs six inputs and one output and utilizes a single LUT, is configured with an INIT value of 64’h0000_0000_0000_0001.

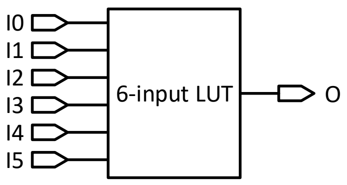
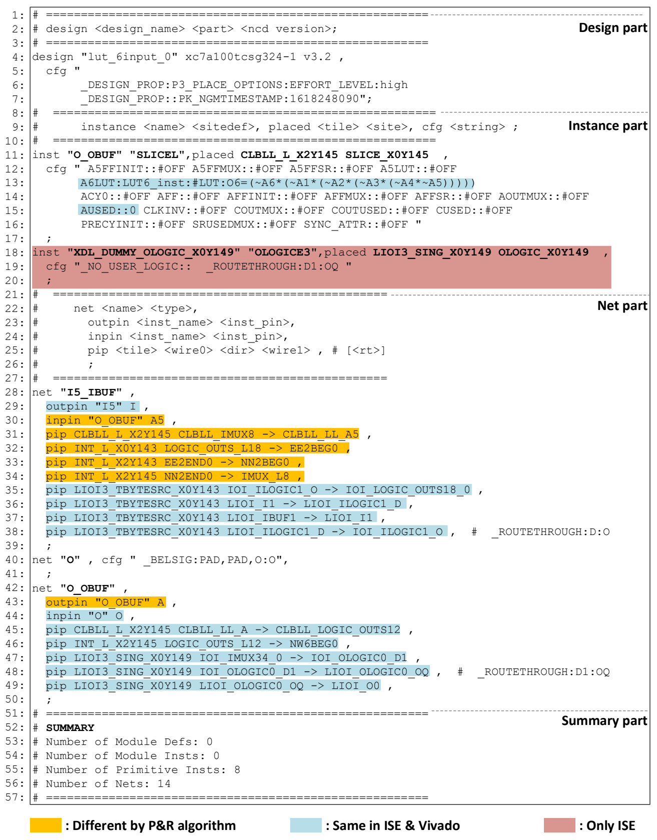


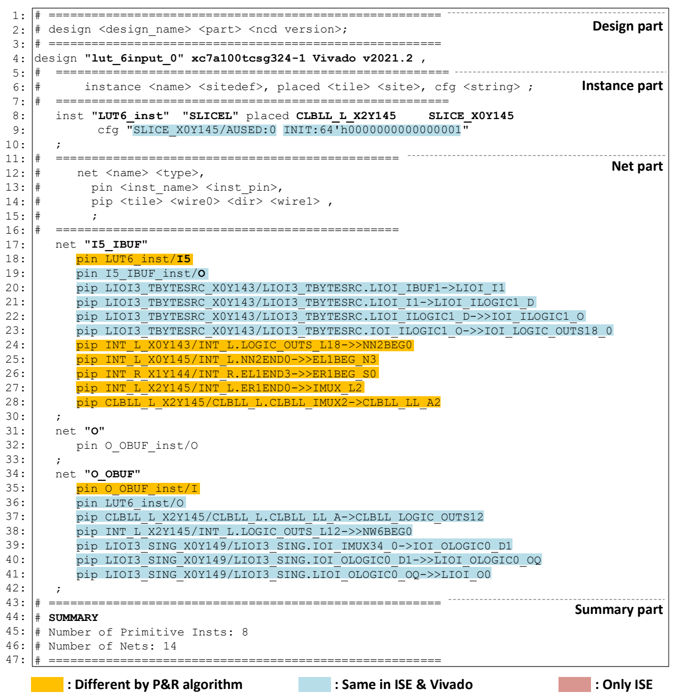
**Figure 13.** Example design for XDL file generation.

The XDL files generated by the two EDA tools are compared in Figure 14. In the comparison, information that is identical in both files is highlighted in blue, and information unique to ISE is emphasized in red. Additionally, the content highlighted in yellow represents information present in both files but with differing names.



```
# =======================================================
# design <design_name> <part> <ncd version>;
# =======================================================
design "lut_6input_0" xc7a100tcsg324-1 v3.2 ,
  cfg "
       _DESIGN_PROP:P3_PLACE_OPTIONS:EFFORT_LEVEL:high
       _DESIGN_PROP::PK_NGMTIMESTAMP:1618248090";
#  =======================================================
#      instance <name> <sitedef>, placed <tile> <site>, cfg <string> ;
#  =======================================================
inst "O_OBUF" "SLICEL",placed CLBLL_L_X2Y145 SLICE_X0Y145  ,
  cfg " A5FFINIT::#OFF A5FFMUX::#OFF A5FFSR::#OFF A5LUT::#OFF
       A6LUT:LUT6_inst:#LUT:O6=(~A6*(~A1*(~A2*(~A3*(~A4*~A5)))))
       ACY0::#OFF AFF::#OFF AFFINIT::#OFF AFFMUX::#OFF AFFSR::#OFF AOUTMUX::#OFF
       AUSED::0 CLKINV::#OFF COUTMUX::#OFF COUTUSED::#OFF CUSED::#OFF
       PRECYINIT::#OFF SRUSEDMUX::#OFF SYNC_ATTR::#OFF "
  ;
inst "XDL_DUMMY_OLOGIC_X0Y149" "OLOGICE3",placed LIOI3_SING_X0Y149 OLOGIC_X0Y149  ,
  cfg "_NO_USER_LOGIC::  _ROUTETHROUGH:D1:OQ "
  ;
#  ===============================================
#     net <name> <type>,
#       outpin <inst_name> <inst_pin>,
#       inpin <inst_name> <inst_pin>,
#       pip <tile> <wire0> <dir> <wire1> , # [<rt>]
#       ;
#  ===============================================
net "I5_IBUF" ,
  outpin "I5" I ,
  inpin "O_OBUF" A5 ,
  pip CLBLL_L_X2Y145 CLBLL_IMUX8 -> CLBLL_LL_A5 ,
  pip INT_L_X0Y143 LOGIC_OUTS_L18 -> EE2BEG0 ,
  pip INT_L_X2Y143 EE2END0 -> NN2BEG0 ,
  pip INT_L_X2Y145 NN2END0 -> IMUX_L8 ,
  pip LIOI3_TBYTESRC_X0Y143 IOI_ILOGIC1_O -> IOI_LOGIC_OUTS18_0 ,
  pip LIOI3_TBYTESRC_X0Y143 LIOI_I1 -> LIOI_ILOGIC1_D ,
  pip LIOI3_TBYTESRC_X0Y143 LIOI_IBUF1 -> LIOI_I1 ,
  pip LIOI3_TBYTESRC_X0Y143 LIOI_ILOGIC1_D -> IOI_ILOGIC1_O ,  #  _ROUTETHROUGH:D:O
  ;
net "O" , cfg " _BELSIG:PAD,PAD,O:O",
  ;
net "O_OBUF" ,
  outpin "O_OBUF" A ,
  inpin "O" O ,
  pip CLBLL_L_X2Y145 CLBLL_LL_A -> CLBLL_LOGIC_OUTS12 ,
  pip INT_L_X2Y145 LOGIC_OUTS_L12 -> NW6BEG0 ,
  pip LIOI3_SING_X0Y149 IOI_IMUX34_0 -> IOI_OLOGIC0_D1 ,
  pip LIOI3_SING_X0Y149 IOI_OLOGIC0_D1 -> LIOI_OLOGIC0_OQ ,  #  _ROUTETHROUGH:D1:OQ
  pip LIOI3_SING_X0Y149 LIOI_OLOGIC0_OQ -> LIOI_O0 ,
  ;
# =======================================================
# SUMMARY
# Number of Module Defs: 0
# Number of Module Insts: 0
# Number of Primitive Insts: 8
# Number of Nets: 14
# =======================================================
```

(**a**) XDL files generated by ISE



```
# =======================================================
# design <design_name> <part> <ncd version>;
# =======================================================
design "lut_6input_0" xc7a100tcsg324-1 Vivado v2021.2 ,
#  =======================================================
#      instance <name> <sitedef>, placed <tile> <site>, cfg <string> ;
#  =======================================================
   inst "LUT6_inst"  "SLICEL" placed CLBLL_L_X2Y145    SLICE_X0Y145
        cfg "SLICE_X0Y145/AUSED:0 INIT:64'h0000000000000001"
   ;
#  ===============================================
#     net <name> <type>,
#       pin <inst_name> <inst_pin>,
#       pip <tile> <wire0> <dir> <wire1> ,
#       ;
#  ===============================================
   net "I5_IBUF"
      pin LUT6_inst/I5
      pin I5_IBUF_inst/O
      pip LIOI3_TBYTESRC_X0Y143/LIOI3_TBYTESRC.LIOI_IBUF1->LIOI_I1
      pip LIOI3_TBYTESRC_X0Y143/LIOI3_TBYTESRC.LIOI_I1->LIOI_ILOGIC1_D
      pip LIOI3_TBYTESRC_X0Y143/LIOI3_TBYTESRC.LIOI_ILOGIC1_D->>IOI_ILOGIC1_O
      pip LIOI3_TBYTESRC_X0Y143/LIOI3_TBYTESRC.IOI_ILOGIC1_O->>IOI_LOGIC_OUTS18_0
      pip INT_L_X0Y143/INT_L.LOGIC_OUTS_L18->>NN2BEG0
      pip INT_L_X0Y145/INT_L.NN2END0->>EL1BEG_N3
      pip INT_R_X1Y144/INT_R.EL1END3->>ER1BEG_S0
      pip INT_L_X2Y145/INT_L.ER1END0->>IMUX_L2
      pip CLBLL_L_X2Y145/CLBLL_L.CLBLL_IMUX2->CLBLL_LL_A2
   ;
   net "O"
      pin O_OBUF_inst/O
   ;
   net "O_OBUF"
      pin O_OBUF_inst/I
      pin LUT6_inst/O
      pip CLBLL_L_X2Y145/CLBLL_L.CLBLL_LL_A->CLBLL_LOGIC_OUTS12
      pip INT_L_X2Y145/INT_L.LOGIC_OUTS_L12->>NW6BEG0
      pip LIOI3_SING_X0Y149/LIOI3_SING.IOI_IMUX34_0->IOI_OLOGIC0_D1
      pip LIOI3_SING_X0Y149/LIOI3_SING.IOI_OLOGIC0_D1->>LIOI_OLOGIC0_OQ
      pip LIOI3_SING_X0Y149/LIOI3_SING.LIOI_OLOGIC0_OQ->>LIOI_O0
   ;
# =======================================================
# SUMMARY
# Number of Primitive Insts: 8
# Number of Nets: 14
# =======================================================
```

(**b**) XDL files generated by Vivado

**Figure 14.** XDL files generated by (**a**) ISE and (**b**) Vivado.

In the XDL file generated by ISE, the PLP and PDP used in the SLICE_X0Y145 site of CLBLL_L_X2Y145 tile, implementing the O_OBUF logic, are identical. Note that, in the case of PDP, it is all expressed as a Boolean function in ISE, but in Vivado, it is expressed as hexadecimal. However, dummy cells that do not use logic are confirmed in XDL generated by ISE by adding the keyword DUMMY in line 18 to line 20 of Figure 14a, but not in XDL generated by Vivado because logic is not used. For the net part, the two files are identified as the same number of nets and the same name. However, some pins for the input/output of the net, such as the A5 pin in line 30 of XDL generated by ISE, as shown in Figure 14a, are represented differently in the XDL file produced by Vivado, appearing as I5 in the line 18 of Figure 14b. Regarding the PIPs, which consist of the net, while the PIPs connected to input and output pins are the same in both files, as shown in Figure 14a,b, some PIPs that constitute the internal connection of the net differ due to variations in the P&R algorithms of the ISE and Vivado. For example, the *I5_IBUF* net represents the net for the *I5* input of the six inputs in Figure 11, from the IOI (Input/Output Interface) tile where *I5* is connected to the CLB tile where the six-input LUT is implemented. The IOI tile and the CLB tile are not connected directly but through an interconnect tile, the INT tile. As shown in the two files in Figure 14, the wires where the *I5_IBUF* net starts are the same as the *LIOI_IBUF1* wire in the *ILIOI3_TBYTESRC_X0Y143* tile. However, the net ends at tile *CLBLL_L_X2Y145*, but the wires are different: *CLBLL_LL_A5* in Figure 14a and *CLBLL_LL_A2* in Figure 14b. In other words, the input pins of a six-input LUT described as *I5* in the HDL are connected differently to the input wires of the CLB where the LUT is implemented, even though they are placed in the same IOI tile in ISE and Vivado. In addition, some of the PIPs from the *LIOI_IBUF1* wire to the destination wire are different due to the different termination points. The reason for the different connections is due to the different routing algorithms mentioned earlier, which only affect the configuration of the network.

In summary, while there are differences in dummy cells and PIPs due to variations from the P&R algorithms, both XDL files generated by ISE and Vivado present all the necessary information for logic implementation in an identical manner. Consequently, the circuits represented in both XDL files are logically perfect and the same.

## 5. Conclusions

In this paper, a method for generating textual netlists in both ISE and Vivado is proposed. When comparing XDL and XDLRC files generated from ISE and Vivado using the proposed method, it is found that XDLRC files match by 99% for programmable points such as PLP, PIP, and PDP. In the case of XDL files, there is approximately a 75% match between files generated by the two EDA tools. The remaining 25% mismatch is attributed to differences in dummy cells and routing algorithms. However, these differences in XDL files do not impact the functionality of the circuit when using each respective XDL file for circuit reconfiguration.

This paper demonstrates that through the proposed method, textual netlists generated from Vivado contain the same programmable point information as those from ISE. It is possible to extend the application scope of the previous reverse-engineering tools to cover devices supported by Vivado, as essential textual netlists can be obtained from both ISE and Vivado. This indicates that the scope of previous reverse-engineering techniques can be extended. Therefore, we believe that the XDLRC and XDL files generated by the proposed method can be used to develop reverse-engineering techniques for FPGAs supported by ISE and Vivado simultaneously. It is also expected that a basis will be provided for developing reverse-engineering techniques for FPGAs supported only by Vivado.

Finally, this paper does not encourage malicious bitstream attacks through reverse engineering but rather highlights the risk associated with malicious attacks and emphasizes the importance of security measures. In addition, in terms of reverse engineering the design implemented in the FPGA, it can help researchers understand the feasibility of such attacks from the white hacker side and develop countermeasures against such methods.

**Author Contributions:** Conceptualization, S.C. and H.Y.; methodology, S.C.; software, S.C.; validation, S.C.; formal analysis, S.C.; investigation, H.Y.; resources, H.Y.; data curation, S.C.; writing—original draft preparation, S.C.; writing—review and editing, H.Y.; visualization, S.C.; supervision, H.Y.; project administration, H.Y.; funding acquisition, H.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no funding.

**Data Availability Statement:** The raw data supporting the conclusions of this article will be made available by the authors upon request.

**Acknowledgments:** This work was supported by BK21 FOUR Program by Chungnam National University Research Grant, 2022.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Drimer, S. *Volatile FPGA Design Security—A Survey*; Computer Laboratory, University of Cambridge: Cambridge, UK, 2008.
2. Wallat, S.; Fyrbiak, M.; Schlögel, M.; Paar, C. A look at the dark side of hardware reverse engineering—A case study. In Proceedings of the 2017 IEEE 2nd International Verification and Security Workshop (IVSW), Thessaloniki, Greece, 3–5 July 2017; pp. 95–100.
3. Swierczynski, P.; Fyrbiak, M.; Koppe, P.; Paar, C. FPGA Trojans Through Detecting and Weakening of Cryptographic Primitives. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1236–1249. [CrossRef]
4. Swierczynski, P.; Fyrbiak, M.; Paar, C.; Huriaux, C.; Tessier, R. Protecting against Cryptographic Trojans in FPGAs. In Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, Canada, 2–6 May 2015; pp. 151–154.
5. FPGA Leadership Across Multiple Process Nodes. Available online: https://www.xilinx.com/products/silicon-devices/fpga.html (accessed on 6 February 2024).
6. De Mulder, E.; Buysschaert, P.; Ors, S.; Delmotte, P.; Preneel, B.; Vandenbosch, G.; Verbauwhede, I. Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem. In Proceedings of the EUROCON 2005—The International Conference on Computer as a Tool, Belgrade, Serbia, 21–24 November 2005; pp. 1879–1882.
7. Choi, S.; Im, N.; Yoo, H. FPGA Design Duplication based on the Bitstream Extraction. In Proceedings of the 2021 18th International SoC Design Conference (ISOCC), Jeju Island, Republic of Korea, 6–9 October 2021; pp. 373–374.
8. Lee, D.; Lee, S.; Cho, M.; Lee, H.-M.; Kim, Y. Data extraction from flash memory and reverse engineering using Xilinx 7 series FPGA boards. In Proceedings of the 2022 19th International SoC Design Conference (ISOCC), Gangneung-si, Republic of Korea, 19–22 October 2022; pp. 330–331.
9. Swierczynski, P.; Becker, G.T.; Moradi, A.; Paar, C. Bitstream Fault Injections (BiFI)–Automated Fault Attacks Against SRAM-Based FPGAs. *IEEE Trans. Comput.* **2018**, *67*, 348–360. [CrossRef]
10. Note, J.-B.; Rannaud, E. From the bitstream to the netlist. In Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 24–26 February 2008; p. 264.
11. Benz, F.; Seffrin, A.; Huss, S.A. Bil: A tool-chain for bitstream reverse-engineering. In Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, 29–31 August 2012; pp. 735–738.
12. 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]
13. Lavin, C.; Padilla, M.; Lundrigan, P.; Nelson, B.; Hutchings, B. Rapid prototyping tools for FPGA designs: RapidSmith. In Proceedings of the 2010 International Conference on Field-Programmable Technology, Beijing, China, 8–10 December 2010; pp. 353–356.
14. Zhang, T.; Wang, J.; Guo, S.; Chen, Z. A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL Code. *IEEE Access* **2019**, *7*, 38379–38389. [CrossRef]
15. Available online: https://prjxray.readthedocs.io/en/latest/index.html (accessed on 6 February 2024).
16. Yu, H.; Lee, H.-M.; Shin, Y.; Kim, Y. FPGA reverse engineering in Vivado design suite based on X-ray project. In Proceedings of the 2019 International SoC Design Conference (ISOCC), Jeju, Republic of Korea, 6–9 October 2019; pp. 239–240.
17. Yu, H.; Cho, M.; Lee, S.; Lee, H.M.; Kim, Y. Multi Look-up Table FPGA Reverse Engineering with Bitstream Extraction and Multiple PIP/PLP Matching. *J. Semicond. Technol. Sci.* **2021**, *21*, 49–61. [CrossRef]
18. Kashani, S.; Emami, M.; Larus, J.R. Bitfiltrator: A general approach for reverse-engineering Xilinx bitstream formats. In Proceedings of the 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL), Belfast, UK, 29 August–2 September 2022; pp. 1–8.
19. Danesh, W.; Joshua, B.; Mostafizur, R. Turning the table: Using bitstream reverse engineering to detect FPGA trojans. *J. Hardw. Syst. Secur.* **2021**, *5*, 237–246. [CrossRef]

20. Zhang, T.; Tehranipoor, M.; Farahmandi, F. BitFREE: On Significant Speedup and Security Applications of FPGA Bitstream Format Reverse Engineering. In Proceedings of the 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 22–26 May 2023; pp. 1–6.
21. Gongye, C.; Luo, Y.; Xu, X.; Fei, Y. Side-Channel-Assisted Reverse-Engineering of Encrypted DNN Hardware Accelerator IP and Attack Surface Exploration. In Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2024; pp. 1–18.
22. Yu, H.; Lee, H.; Lee, S.; Kim, Y.; Lee, H.-M. Recent advances in FPGA reverse engineering. *Electronics* **2018**, *7*, 246. [CrossRef]
23. Xilinx. *Spartan-3 Generation FPGA User Guide (UG331)*; Xilinx: San Jose, CA, USA, 2011.
24. Xilinx. *Virtex-5 FPGA User Guide*; Xilinx: San Jose, CA, USA, 2012.
25. Xilinx. *7 Series FPGAs Configurable Logic Block (UG474)*; Xilinx: San Jose, CA, USA, 2016.
26. Xilinx. *UltraScale Architecture Configurable Logic Block (UG574)*; Xilinx: San Jose, CA, USA, 2017.
27. Xilinx. *Spartan-3 FPGA Family Advanced Configuration Architecture (XAPP 452)*; Xilinx: San Jose, CA, USA, 2008.
28. Xilinx. *Virtex-5 FPGA Configuration User Guide (UG191)*; Xilinx: San Jose, CA, USA, 2017.
29. Xilinx. *7 Series FPGAs Configuration (UG470)*; Xilinx: San Jose, CA, USA, 2018.
30. Xilinx. *UltraScale Architecture Configuration (UG570)*; Xilinx: San Jose, CA, USA, 2020.