

Analysis of the Extensibility of FPGA Reverse Engineering

Soyeon Choi, Yunjin Noh, Dohun Kim, and Hoyoung Yoo
Dept. of Electronics Engineering, Chungnam National University
{sychoi.cas, yjnoh.cas, dhkim21.cas}@gmail.com, hyyoo@cnu.ac.kr

Abstract

The process of transmitting the bitstream from external memory in SRAM-based FPGA is vulnerable to attacks such as reverse engineering. The previous FPGA reverse engineering primarily focus on low-end FPGAs, supported by Xilinx ISE. This is because ISE provides readable netlists, which are essential data in reverse engineering. However, Vivado does not offer textual netlists, making it is difficult to reverse engineer the FPGAs supported by Vivado. In this paper, we propose a method to generate textual netlists in Vivado. According to experimental results, the XDLRC and XDL generated in Vivado match 99% and 75% with those generated in ISE, respectively.

Keywords: Reverse Engineering, Xilinx, VIVADO, ISE, XDL, XDLRC

1. Introduction

AMD Xilinx and Intel Altera FPGAs demonstrate the highest market share in the SRAM-based FPGA industry [1]. FPGA chip manufacturers typically provide EDA tools to support circuit synthesis and implementation on FPGA chips, with Xilinx offering two such tools: ISE Design Suite (ISE) for low-end FPGAs, and VIVADO Design Suite (Vivado) for the latest high-end FPGAs.

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 [2]. The process of transmitting the circuit's netlist in bitstream format from external memory during power-up in FPGA systems makes it vulnerable to malicious attacks like reverse engineering [3-4].

Presently, existing FPGA reverse engineering tools primarily focus on Xilinx's low-end FPGAs, supported by Xilinx ISE. This preference stems from Xilinx ISE provision of readable text-based XDLRC (Xilinx Design Language Routing Configurable logic) and XDL (Xilinx Design language) [5], which are essential for reverse engineering. Note that, XDLRC is a file describing all available hardware resources within the FPGA and XDL is the text-

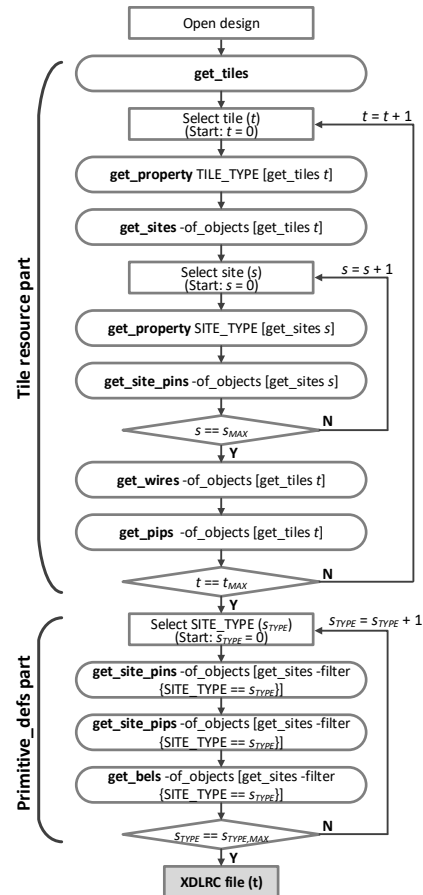


Figure 1. Flow chart to generate XDLRC in Vivado

based netlist on the FPGA. When using ISE, XDLRC and XDL are generated by only one Tcl command 'xdl'. However, since Vivado does not support XDL and XDLRC, reverse engineering is at a fundamental level. Consequently, this paper proposes a method to generate XDLRC and XDL in Vivado, similar to those in ISE, to extend reverse engineering technology.

2. Netlist generation in Vivado

2.1. XDLRC generation

XDLRC contains the information for all tiles in the FPGA including PLP (programmable logic point),

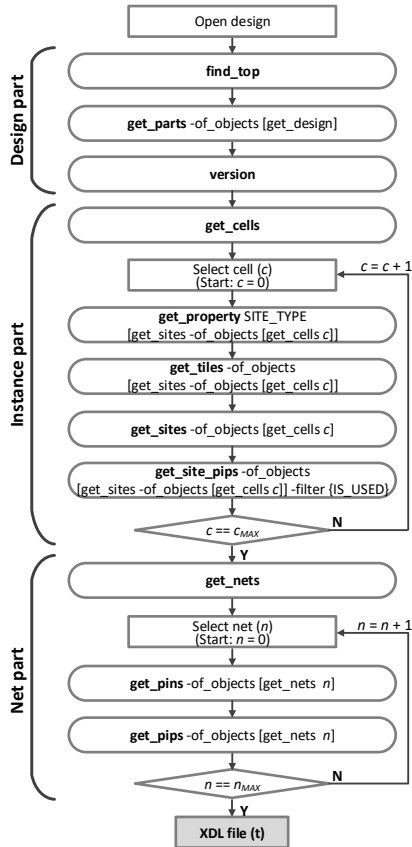


Figure 2. Flow chart to generate XDL in Vivado.

PIP (programmable interconnect point), and PDP (programmable data point).

In Vivado, there is no Tcl command like ‘*xdl*’ to generate an XDLRC. However, by utilizing the Tcl commands supported by Vivado, it is possible to obtain information about FPGA hardware resources and create an XDLRC. Figure 1 illustrates the process of generating an XDLRC file in Vivado as a flowchart with Tcl commands. To extract the information of tiles, the process is repeated for all tiles. To obtain the internal structure, the process of discovering internal site details is repeated for each site type.

2.2. XDL generation

XDL represents the netlist of the target circuit which implemented on FPGA. To generate an XDL file in Vivado, several Tcl commands that allow obtaining the necessary information, like with XDLRC, should be used. To obtain the necessary information for the XDL, Tcl commands in Fig. 2 are required. Unlike XDLRC, only used components must be extracted, so each command is required to use the `-filter {IS_USED}` option. Figure 2 represents the flowchart of the XDL generation process in Vivado. It involves obtaining information about the design part, followed by iterating through the process of obtaining all the necessary information

```

1: .....
2: #hdl_resource_report v0.2 xc7a100tcs324-3 artix7
3: # .....
4: # * Tile Resources
5: # .....
6: (tiles 209 148)
7: (tile 1 : I1013_S1MG_X0V199_I1013_S1MG_3
8: (primitive_site OLOGIC_X0V199_OLOGIC3 internal 33
9: (pinwire O02 input: IOI_OLOGIC0_O02
10: (pinwire O02 input: IOI_OLOGIC0_O02
11: (pinwire O02 input: IOI_OLOGIC0_O02
12: )
13: )
14: (conn IO_I0V7_INTERFACE_I_X0V199_INT_INTERFACE_CLK1
15: (conn I0V7_I_X0V199_CLK_I13
16: )
17: )
18: (wire IOI_CLK1_0 3
19: )
20: (wire I1003_S1MG_X0V199_I003_T01
21: )
22: (pip I1013_S1MG_X0V199_I01_BVPE_0->IOI_I0B1AY0_C1W0CTR0
23: (pip I1013_S1MG_X0V199_I01_BVFP_0->I1013_I0B1AY0_I0B1AY0
24: (pip I1013_S1MG_X0V199_I01_CLK0_0->IOI_I0LOGIC0_C1R03V)
25: )
26: )
27: )
28: )
29: )
30: )
31: )
32: )
33: )
34: )
35: )
36: )
37: )
38: )
39: )
40: )
41: )
42: # * Summary
43: # .....
44: #summary tiles=30932 sites=28963 sitesdefs=6 mmpins=100381 mmpips=40375035

```

(a) XDLRC generated in ISE

```

1: .....
2: # * Tile Resources
3: # .....
4: (tiles 209 148)
5: (tile I1013_S1MG_X0V199_I1013_S1MG_3
6: (primitive_site OLOGIC_X0V199_OLOGIC3
7: (pinwire O02
8: (pinwire O02
9: (pinwire O02
10: )
11: )
12: (wire IOI_CLK1_0 1
13: )
14: (pip I1013_S1MG_X0V199_I01_BVPE_0->IOI_I0B1AY0_C1W0CTR0
15: (pip I1013_S1MG_X0V199_I01_BVFP_0->I1013_I0B1AY0_I0B1AY0
16: (pip I1013_S1MG_X0V199_I01_CLK0_0->IOI_I0LOGIC0_C1R03V)
17: )
18: )
19: )
20: )
21: )
22: )
23: )
24: )
25: )
26: )
27: )
28: )
29: )
30: )
31: )
32: )
33: )
34: )
35: )
36: )
37: )
38: )
39: )
40: )
41: )
42: # * Summary
43: # .....
44: #summary tiles=30932 sites=28551 sitesdefs=45 mmpins=100381 mmpips=40313010

```

(b) XDLRC generated in Vivado

Figure 3. XDLRC generated by (a) ISE and (b) Vivado.

about cells and nets required for design implementation for instance and net part.

3. Analysis of files

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

3.1. Comparison of XDLRC

When comparing XDLRCs generated for the same FPGA using two EDA tools, the number and names of tiles always remain consistent as long as the FPGA device is the same. However, the ‘*conn*’s which representing the hard-wired connection between tiles or elements are not extracted by Vivado. However, for sites that include as PLPs or PDPs, the structures are same in both ISE and Vivado. In terms of PIPs, the number of PIPs confirmed in ISE and Vivado are 40,375,035 and 40,313,010, respectively. In Vivado, 62,025 PIPs are not recognized. Each of them has only one connection, linking a start wire to an end wire.

Figure 3 illustrates the internal structure of a input/output interface (IOI) tile with results obtained from both ISE and Vivado. In Fig. 3, the components

```

1: # =====
2: # design <design_name> <part> <end version>
3: # =====
4: design "lut_6input_0" xc7a100tcsq324-1 v3.2 ,
5: cfg "
6:   _DESIGN_PROP:P3_PLACE_OPTIONS:EFFORT_LEVEL:high
7:   _DESIGN_PROP:PK_NGNTIMESTAMP:1618248909"
8: # =====
9: instance <name> <siteDef>, placed <tile> <site>, cfg <string> ;
10: # =====
11: inst "O_OBUF" "SLICEL" placed CLBLL_L_X2V145 SLICE_X0V145
12: ;
13: cfg "
14:   ASFFINIT:#OFF ASFFMUX:#OFF ASFFSR:#OFF ASLUP:#OFF
15:   AELUP:L0U6_inst:#L0U6_OG*(-A6*(-A1*(-A2*(-A3*(-A4*(-A5))))))
16:   ACYD:#OFF AFF:#OFF AFFINIT:#OFF AFFMUX:#OFF AFFSR:#OFF AOUTMUX:#OFF
17:   AUSERCSO:CLKTW:#OFF COUTMUX:#OFF COUTSRD:#OFF CUSED:#OFF
18:   PRECYINIT:#OFF PRUSERMUX:#OFF SYNC_ATTR:#OFF "
19: ;
20: # =====
21: #
22: # net <name> <type>,
23: #   outpin <inst_name> <inst_pin>,
24: #   inpin <inst_name> <inst_pin>,
25: #   pip <tile> <wire0> <dir> <wire1>, # [ctrl]
26: #
27: #
28: net "I5_IBUF" ,
29:   outpin "I5" 5,
30:   inpin "O_OBUF" A5 ,
31:   @pip CLBLL_L_X2V145 CLBLL_IMUX8 -> CLBLL_LL_A5 ,
32:   @pip INT_L_X0V143 LOGIC_OUTPS_L18 -> N2ZB800 ,
33:   @pip INT_L_X2V143 REZEND0 -> NNZB800 ,
34:   @pip INT_L_X0V145 NNZEND0 -> IMUX_18 ,
35:   @pip LIO13_TBVTESRC_X0V143 IOI_ILOGIC1_0 -> IOI_LOGIC_OUTS18_0 ,
36:   @pip LIO13_TBVTESRC_X0V143 LIOI_I1 -> LIOI_ILOGIC1_D ,
37:   @pip LIO13_TBVTESRC_X0V143 LIOI_IBUF1 -> LIOI_I1 ,
38:   @pip LIO13_TBVTESRC_X0V143 LIOI_ILOGIC1_D -> IOI_ILOGIC1_0 , # _ROUTETHROUGH:D1:OQ
39: ;
40: net "O" , cfg " _BELSIG:PAD,PAD,0:0" ,
41: ;
42: net "O_OBUF" ,
43:   outpin "O_OBUF" 8 ,
44:   inpin "O" 8 ,
45:   @pip CLBLL_L_X2V145 CLBLL_LL_A -> CLBLL_LOGIC_OUTS12 ,
46:   @pip INT_L_X2V145 LOGIC_OUTPS_L12 -> NN6B800 ,
47:   @pip LIO13_SING_X0V149 IOI_IMUX8_0 -> IOI_OLOGIC_D1 ,
48:   @pip LIO13_SING_X0V149 IOI_OLOGIC_D1 -> LIOI_OLOGIC_OQ , # _ROUTETHROUGH:D1:OQ
49:   @pip LIO13_SING_X0V149 LIOI_OLOGIC_OQ -> LIOI_O0 ,
50: ;
51: # =====
52: # SUMMARY
53: # Number of Module Defs: 0
54: # Number of Module Insts: 0
55: # Number of Primitive Insts: 8
56: # Number of Nets: 14
57: # =====

```

(a) XDL files generated in ISE

```

1: # =====
2: # design <design_name> <part> <end version>
3: # =====
4: design "lut_6input_0" xc7a100tcsq324-1 Vivado v2021.2 ,
5: cfg "
6:   instance <name> <siteDef>, placed <tile> <site>, cfg <string> ;
7: # =====
8: inst "L0U6_inst" "SLICEL" placed CLBLL_L_X2V145 SLICE_X0V145
9: ;
10: cfg "
11:   SLICE_X0V145/AUSERCSO INT:64#0000000000000001"
12: ;
13: # =====
14: # net <name> <type>,
15: #   pin <inst_name> <inst_pin>,
16: #   pip <tile> <wire0> <dir> <wire1>,
17: #
18: #
19: net "I5_IBUF"
20:   pin L0U6_inst/I5
21:   pin I5_IBUF_inst/O
22:   @pip LIO13_TBVTESRC_X0V143/LIO13_TBVTESRC.LIOI_IBUF1->LIOI_I1
23:   @pip LIO13_TBVTESRC_X0V143/LIO13_TBVTESRC.LIOI_I1->LIOI_ILOGIC1_D
24:   @pip LIO13_TBVTESRC_X0V143/LIO13_TBVTESRC.LIOI_ILOGIC1_D->LIOI_ILOGIC1_0
25:   @pip LIO13_TBVTESRC_X0V143/LIO13_TBVTESRC.FOI_ILOGIC1_0->LIOI_LogiC_OUTS18_0
26:   @pip INT_L_X0V143/INT_L_LOGIC_OUTPS_L18->NNZB800
27:   @pip INT_L_X2V145/INT_L_NNZEND0->EL18EG_N3
28:   @pip INT_R_X1V144/INT_R_ELEND3->SR18EG_S0
29:   @pip INT_L_X2V145/INT_L_REZEND0->SMUX_L2
30:   @pip CLBLL_L_X2V145/CLBLL_L_CLBLL_IMUX8->CLBLL_LL_A2
31: ;
32: net "O"
33:   pin O_OBUF_inst/O
34: ;
35: net "O_OBUF"
36:   pin O_OBUF_inst/O
37:   pin L0U6_inst/O
38:   @pip CLBLL_L_X2V145/CLBLL_L_CLBLL_LL_A->CLBLL_LOGIC_OUTS12
39:   @pip INT_L_X2V145/INT_L_LOGIC_OUTPS_L12->NN6B800
40:   @pip LIO13_SING_X0V149/INT_L_IMUX8_0->LIOI_OLOGIC_D1
41:   @pip LIO13_SING_X0V149/LIO13_SING.IOI_OLOGIC_D1->LIOI_OLOGIC_OQ
42:   @pip LIO13_SING_X0V149/LIO13_SING.LIOI_OLOGIC_OQ->LIOI_O0
43: ;
44: # =====
45: # SUMMARY
46: # Number of Primitive Insts: 8
47: # Number of Nets: 14
48: # =====

```

(b) XDL files generated in Vivado

Figure 4. XDL generated by (a) ISE and (b) Vivado.

highlighted in red boxes are showed only in the XDLRC generated in ISE. The blue boxes in Fig. 3 are extracted components with any EDA tools.

3.2. Comparison of XDL

To generate XDL, an RTL design which describes the target circuit design is required. In this paper, the RTL instantiated a 6-input LUT primitive with 6 inputs and 1 output is used as an example design.

When comparing the XDL, as depicted in Fig. 4, the PLP and PDP used for circuit implementation are equally extracted from both EDA tools. However, in

the case of PDP, it is expressed as a Boolean function in ISE and Hexadecimal in Vivado. The difference between XDL lies in the presence or absence of dummy cells. In XDL generated in ISE, additional dummy cells highlighted by red box in Fig. 4 with names are included. In XDL files generated by Vivado, information about the dummy cells used for route-through is not provided.

In net part, the pin part of the net is extracted the same, but some pin names are different, like the yellow boxes in Fig. 4. In addition, the composition of the PIP that constitutes the net vary due to the difference between P&R algorithm of ISE and Vivado, and in Fig. 4, PIPs expressed in yellow boxes mean the difference between the net generated by the two EDA tools.

4. Conclusion

In this paper, we proposed a method for generating textual netlists in both ISE and Vivado. By comparing the XDL and XDLRC files using the proposed method, it is found that 99% and 75 % match are achieved, respectively. Therefore, it becomes feasible to extend the applicability of existing reverse engineering tools cover devices supported by Vivado through the acquisition of essential textual netlists.

Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2022R1A5A8026986), Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R111A3055806), and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (2022-0-01170)

References

- [1] S. Wallat, M. Fyrbiak, M. Schlögel and C. Paar, "A look at the dark side of hardware reverse engineering - a case study," *IEEE 2nd International Verification and Security Workshop (IVSW)*, 2017, pp. 95-100
- [2] E. De Mulder et al., "Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem," *EUROCON 2005*, pp. 1879-1882.
- [3] Z. Ding, Q. Wu, Y. Zhang, and L. Zhu, "Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation," *Microprocessors Microsystems-Embedded Hardware Des.*, vol. 37, no. 3, pp. 299-312, 2013.
- [4] T. Zhang, J. Wang, S. Guo and Z. Chen, "A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL Code," in *IEEE Access*, vol. 7, pp. 38379-38389, 2019.
- [5] Hoyoung Yu, et al. "Recent advances in FPGA reverse engineering." *Electronics*, vol. 7, no. 10, 2018.