

# Obfuscation and Watermarking of FPGA Designs Based on Constant Value Generators

Vladimir V. Sergeichik, Alexander A. Ivaniuk  
Computer Science Department, BSUIR  
Minsk, Belarus  
vovasq@mail.ru, ivaniuk@bsuir.by

Chip-Hong Chang  
School of Electrical and Electronic Engineering  
Nanyang Technological University, Singapore  
echchang@ntu.edu.sg

**Abstract**—Obfuscation is a technique which makes design less intelligible in order to prevent or increase reverse engineering effort. In this paper, a new approach to hardware obfuscation by inserting constant value generators (CVGs) is proposed. A CVG is a circuit that generates the same fixed logic value but will not be minimized by the synthesizer. CVGs can be used to create new logic primitives, embed watermarks and introduce fictive interdependencies in the circuit. They help to hide actual design performance information by tricking the synthesizer tools to generate deceiving delay reports through the false paths.

## I. INTRODUCTION

Hardware piracy and other attacks are dire threats to FPGA designs. Side-channel signals can be exploited to deduce secret key on hardware implementations of cryptographic algorithms. Malicious hardware Trojan can be implanted to leak secret information, affect functionality, deactivate and destroy circuits. It is important to increase the reverse engineering effort of adversaries because many attacks rely on knowledge about the circuit internal structure. One possible solution is obfuscation, which aims to make the circuit structure and functionality difficult to perceive and comprehend so as to substantially increase the time and cost of reverse engineering [1]. Obfuscation is also often used to hide author's watermarks and user's fingerprints in hardware intellectual property (IP) protection [1].

Approaches to hardware obfuscation are vastly different. Some approaches focus on hardware description language (HDL) level, some operate with common design structure such as State Transition Graph (STG) while others introduce specific constructs. A comprehensive taxonomy of obfuscation transformations, which are also suitable for the protection of HDL sources, is presented in [2]. There were only a few studies thereafter on the application of these obfuscation transformations for HDLs. Classical Collberg's approaches were implemented for VHDL and Verilog in [3]. Almost all of them have no effect on the synthesized circuit. The rare exceptions have very high hardware overheads. Several VHDL-specific approaches which impact circuit and their overheads are presented in [4]. There are several approaches that manipulate with the STG. One approach uses the expansion of Control Flow Graph (CFG) state space and code-word concept [5]. Code-word is created step-by-step during the initialization. Several states are made dependent from the code-word bits. When the code-word is incorrect, the states are traversed in the wrong sequence. In stuttering circuits'

approach [6], a sequence of four structural transformations, namely retiming, resynthesis, sweep and conditional stuttering is employed. The circuit is transformed into an equivalent key-based circuit. When the key is wrong, the circuit is slower than expected. Another approach uses Simple Logical Implications (SLIs). Spawned from logic synthesis, they exploit the relation of type  $(a = 0) > (b = 1)$  between two circuit nodes  $a$  and  $b$  [7]. SLIs are used to introduce fictive combinational loops to make circuit unintelligible without changing its functionality.

Watermarking and fingerprinting are often used hand-in-hand with obfuscation. One technique searches the correspondence between the outputs of STG and watermark bits [8]. If the correspondence is found, then the bits are reused as watermarked bits; otherwise new input variable will be added. An improved variant of this method is presented in [9], where the watermark bits are inserted into pseudo-randomly selected bits with a longer watermark detection input sequence to increase the reuse of existing output bits and the watermark stealthy. In [10], ways to modulate the power dissipation by power signature generators (e.g. shift registers) are proposed for watermarking. In [11], additional constraints are imposed on the scan-cell ordering problem for watermarking while minimizing the overall test power. Obfuscation of watermark bits is desirable except that it is not carried out at the HDL level in the abovementioned schemes.

## II. LEXICAL AND CIRCUIT OBFUSCATION

There are two types of obfuscation for HDL: lexical and functional [12]. Lexical obfuscation affects only source code level. Its main disadvantage is the synthesized circuits before and after obfuscation are almost identical [12]. As the circuit remains unchanged, it is vulnerable to the simplest attack of logic synthesis [12].

Lexical obfuscation may be described as shown on Fig. 1, where  $V$  is the HDL source of the design,  $V^*$  is the lexically obfuscated source,  $obf$  is the obfuscation procedure,  $O$  is the big O notation in computational complexity,  $DD$  is the synthesis procedure, and  $Sch$  is the schematic representation of the synthesized circuit.

$$\begin{aligned} &V; \\ &V^* = obf(V); \\ &O(V) < O(V^*); \\ &Sch = DD(V) = DD(V^*); \end{aligned}$$

Fig. 1. High-level description of lexical obfuscation.

Some lexical methods for VHDL obfuscation may result in different expansion of Look-Up Tables (LUTs) due to the

heuristic of the synthesis algorithms. Their computational complexities are not fully explored. Some apparently lexical methods may change the design considerably, e.g., the method in [4] removes many levels of registers.

The main idea of *circuit* obfuscation is to create a more sophisticated and unintelligible circuit that has the functionality equivalent to the original design based on their externally observed behavior [12], as depicted in Fig. 2, where *func* is the functionality of the circuit.

$$\begin{aligned}
 &V; \\
 &V^* = obf(V); \\
 &Sch = DD(V); \\
 &Sch^* = DD(V^*); \\
 &Sch \neq Sch^*; \\
 &func(Sch) \equiv func(Sch^*); \\
 &O(V^*) > O(V); \\
 &O(Sch^*) > O(Sch);
 \end{aligned}$$

Fig. 2. High-level description of circuit obfuscation.

### III. CONSTANT GENERATORS

#### A. Theory and Implementation

Constant value generator (CVG) is a form of *opaque predicate*. Its value is known at obfuscation time and may possibly be deduced by the adversary only through rigorous analysis [2]. The “0” and “1” pins of the circuits are substituted by generating the appropriate logical values permanently, as shown in Fig. 3, where  $V_{\{0,1\}}$  is a HDL description of the primitive, and  $V_{DD}$  and  $GND$  are the logical “1” and “0” sources, respectively.

$$\begin{aligned}
 &V_{\{0,1\}}; \\
 &DD(V_{\{0,1\}}) = Sch_{0,1} \in \{V_{DD}, GND\}; \\
 &func\{Sch_{0,1}\} \equiv func\{V_{DD}, GND\};
 \end{aligned}$$

Fig. 3. High-level description of constant generator.

To prevent the CVG circuit from being recognized and minimized by the synthesis tool, the sequential and combinational logic are mixed. Examples of such CVGs are shown in Fig. 4. Other techniques include the usage of signals with well-defined semantics (e.g., system reset and clock) as the inputs of CVGs.

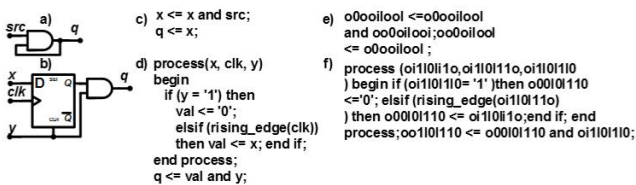


Fig. 4. (a), (b): generators of constant values; (c), (d): original HDL sources; (e), (f): lexically obfuscated sources.

The CVGs from Fig. 4 are synthesized using Xilinx xst 8.1i. The synthesizer is unable to recognize constants and minimize these CVGs. Circuit (a) has two flaws. First, the synthesizer issues warning about the combination loop, which gives vital clue to the adversary. Second, if the input *src* is “1” during power-on, then it is possible for the output to return “1” before it is changed. The second flaw can be circumvented by connecting the CVG’s input to the system reset. Circuit (b) is free from these flaws, but it incurs more hardware resources and may produce glitches.

#### B. Complexity

Software metrics are typically used to measure the complexity of obfuscation transformations. The metrics should take into account the VHDL specific attributes.

The complexity of ‘0’-signal source (ground) against CVG (Fig. 4(b)), measured in terms of TCASC metric [14], are compared in Table I. For circuit obfuscation, the logical circuit complexity needs to be considered, which can be estimated by the total number of gates or the gate count on the critical path. More accurate design entropy metric [15] can be used in addition to lexical metrics.

TABLE I. COMPLEXITY MEASUREMENTS

Metrics	Constant ‘0’	CVG
<b>Lexical Metrics</b>		
Fan-In	0	3
Fan-Out	1	1
Process Coupling	0	0.75
Process Number	0	1
Signal Number	1	5
Number of Operators	1	5
Number operators per process	0	4
Mean Sensitivity List Size	0	3
TCASC	1	2.9
<b>Total</b>	<b>4.0</b>	<b>25.65</b>
<b>Circuit Metrics</b>		
Design Entropy	0	7
Number of Gates	0	2
Number of Wires	1	6
<b>Total</b>	<b>1</b>	<b>15</b>

#### C. Insertion and use

A straightforward application of the primitives is the substitution of constants in the source code by the outputs of CVGs. For example, the statement “if (sig = ‘0’) then ...” can be replaced by the statement “if (sig = const\_out) then ...”, where *const\_out* is the output of zero CVG. This transformation forces the adversary to analyze the zero generator functions. Usual lexical obfuscation transformations (identifiers scrambling, format removal, etc.) can be employed in order to make the codes illegible to human reader.

With the help of CVGs, compound functions can be built. For instance, logical *AND* and *OR* functions can be implemented using a 2-input multiplexor (mux) with the basic primitives  $G_0$  and  $G_1$ , as shown in Fig. 5(a) and (b), respectively, where  $s_0$  and  $s_1$  are real signals. These compound primitives will not be recognized by the synthesizer.

The CVGs can also be used to introduce fictive connections and increase the dependencies between circuit nodes. This can be accomplished by the insertion of multiplexor (mux) with its data inputs connected to the outputs of original circuit nodes, and its selector input connected to the CVG. The mux will permanently pass the same input to the output. In order to make the circuit more difficult to analyze, the nodes are carefully chosen for CVG insertion. For such fictive dependencies to evade the removal by the synthesizer, both input nodes of the mux should have large fan-ins and fan-outs, and they should not be ports and should not lie on the critical path.



used with three optimization options: none, area and speed. Projects with pin constraints are examined. The results for 0, 1 and 5 CVGs are presented in Table III. The circuit names are appended with ‘\A’ and ‘\D’ to indicate that the percentage overheads are for area and delay, respectively. The acronyms OA, OS and *i*ON in the table heading represent area optimization, delay optimization and unoptimized options with *i* number of CVGs, respectively. All results (except 5ON[7]) were computed relative to 0ON; Negative value means improvement over 0ON.

TABLE III. PERCENTAGE OVERHEADS AFTER INSERTION OF 0, 1, 5 CVGS

Title	0ON	OA	OS	1ON	5ON	OA	OS	5ON[7]
c432\A	0	-57	-43	1	9	-30	-21	5.6
c432\D	0	-3	-2	20	29	47	20	-
c499\A	0	-53	-43	1	7	-42	-39	2.3
c499\D	0	12	12	28	105	103	124	-
c1355\A	0	-84	-82	0	2	-73	-73	1.9
c1355\D	0	5	-4	29	88	82	90	-
c1908\A	0	-79	-60	0	2	-77	-69	1.7
c1908\D	0	4	0	1	43	44	50	-
c3540\A	0	-69	-60	0	1	-58	-51	0.9
<b>c3540\D</b>	0	-3	-3	<b>32</b>	81	62	62	-
c6288\A	0	-62	-45	0	0	-55	-62	0.7
c6288\D	0	-19	-31	0	18	6	-16	-

The area overheads grow almost linearly with the number of CVGs for each optimization option, although they are relatively low. On the contrary, the delay overheads reported by the synthesizer are unacceptably high. The actual delay incurred by the CVG is computed manually to be 6.29 ns for c3540 with 1 CVG. It consists of the fixed propagation delay of LUT (mux) and the delays of two wires (4.5 ns). The maximum delay difference for different routing paths is found to be 1.79ns. The delay computed for the original worst critical path and the same path after 1 CVG insertion are 40.19 ns and 38.69 ns, respectively. The large discrepancy between the actual and synthesizer reported delay overheads can be explained as follows. The tool is deceived by the CVGs to consider false critical path through the nodes that will never be excited. The path may pass through the CVG inputs whose values are immaterial. Careful CVG placements can make the reported path delay much larger than the actual critical path delay. This shows that CVGs have successfully hidden the actual design performance to avoid it from being targeted by the adversaries. Comparison with the approach [7] for 5 dumping cycles was shown in the last column, where its percentage area overhead is measured based on the number of transistors in ASIC implementation.

## V. CONCLUSION

This paper presents an approach to circuit obfuscation by CVG insertion. It increases the complexity of HDL code of a design and its synthesized circuit. The proposed CVGs will not be minimized by different synthesizing tools. The approach incurs very low hardware and delay overheads but the synthesis tools will be fooled about the actual performance of the circuit. We have also explained how the CVGs can help to hide watermark or fingerprint in LUTs without incurring

additional overheads.

## REFERENCES

- [1] R. S. Chakraborty, *Hardware Security Through Design Obfuscation*. PhD Dissertation, Case Western Reserve University, 2010, <https://etd.ohiolink.edu/>.
- [2] C. Collberg, C. Thomborson, D. Low, “A taxonomy of obfuscating transformations,” *Tech. Report*, University of Auckland: Department of Computer Science, Auckland, 1997.
- [3] U. Meyer-Base, E. Castillo, G. Botello, “Intellectual property protection (IPP) using obfuscation in C, VHDL, and Verilog coding,” *SPIE IX*, 2011, vol. 8058.
- [4] M. Brzozowski, V. N. Yarmolik, “Obfuscation as intellectual rights protection in VHDL language,” in *Proc. 6th Int. Conf. Comput. Inform. Syst., Ind. Management App.*, Elk, Poland, Jun. 2007, pp. 337 – 340.
- [5] A. Desai, *Anti-Counterfeit and Anti-Tamper Implementation Using Hardware Obfuscation*, Master thesis, Blacksburg, 2013.
- [6] L. Li and H. Zhou, “Structural transformation for best-possible obfuscation of sequential circuits,” in *Proc. Hardware Oriented Security and Trust*, Texas, USA, June 2013, pp. 55 – 60.
- [7] V. A. Bespalov, A. L. Glebov and A. N. Kononov, “The obfuscation method of digital circuits based on the use of logical implications,” *Russian Microelectronics*, vol. 41, no. 7, pp. 415 – 418, Dec. 2012.
- [8] A. Abdel-Hamid, “Public-key watermarking technique for IP designs,” in *Proc. Des. Autom. Test Eur.*, vol. 1, Munich, Germany, Mar. 2005, pp. 330 – 335.
- [9] A. Cui, C. H. Chang, S. Tahar and A. Abdel-Hamid, “A robust FSM watermarking scheme for IP protection of sequential circuit design,” *IEEE Trans. CAD*, vol. 30, no. 5, pp. 678-690, May 2011.
- [10] D. Ziener and J. Teich, “FPGA core watermarking based on power signature analysis,” in *Proc. IEEE Int. Conf. on Field-Programmable Technology*, Thailand, Dec. 2006, pp. 205 – 212.
- [11] C. H. Chang and A. Cui, “Synthesis-for-testability watermarking for field authentication of VLSI intellectual property,” *IEEE Trans. Cir. Syst. I*, vol. 57, no. 7, pp. 1618 – 1630, July 2010.
- [12] A. A. Ivaniuk, *Design of Embedded Digital Systems and Devices*. Minsk: Bestprint, 2012 (in Russian).
- [13] J. Xu, J. Long, W. Huang “A DFA-based Distributed IP Watermarking Method Using Data Compression Technique” *J. Convergence Inform. Technol.*, 2011, No. 6, pp. 152-160.
- [14] J. R. R. Kumar and S. Dharavath, “Evaluation of spatial complexity metrics of object oriented software,” *Int. J. Emerging Tech. Advanced Engineering*, vol. 2, no. 4, p. 272 – 276, Apr. 2012.
- [15] B. Menhorn and F. Slomka, “Design entropy concept - a measurement for complexity,” in *Proc. 11th IEEE/ACM/IFIP Int. Conf. Hardware/software Codesign Syst. Synthesis*, Taipei, Taiwan, October 2011, p. 285 – 294.
- [16] M. Schmid, D. Ziener and J. Teich, “Netlist-level IP protection by watermarking for LUT-based FPGAs,” in *Proc. Int. Conf. Field-programmable Tech.*, Taipei, Taiwan, Dec. 2008, pp. 209 – 216.
- [17] Using look-up tables as shift registers (SRL16) in Spartan-3. [http://www.xilinx.com/support/documentation/application\\_notes/xapp465.pdf](http://www.xilinx.com/support/documentation/application_notes/xapp465.pdf).
- [18] OpenCores Mode of Access: <http://opencores.org/>.
- [19] ISCAS High-Level Models. University of Michigan. <http://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html>