ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Maciej Brzozowski[1], Vyacheslav Yarmolik[1]

OBFUSCATION QUALITY IN HARDWARE DESIGNS

Abstract: Software is more and more frequently distributed in form of source code. Unprotected code is easy to alter and build in others projects. The answer for such attacks is obfuscation. Transformation of software source code which preserves its semantical functionality but analizability is made very dificult. We focus in our research on Very High Speed Integrated Circuits Hardware Description Language (VHDL). In previous articles we presented transformats assimilated from other high level languages for needs of hardware designs and we showed a set of new transformants which do not appear in different languages than VHDL. The next step we have to do is to establish a set of criterias through which we can rate quality of analyzed transformats. In article we present current bacground of software quality measure and we rate their usage for protecting intellectual property on digital designs. After that we propose a new way of obfuscation quality measure taking into account requirements of hardware designs.

Keywords: Obfuscation, Very High Speed Integrated Circuits Hardware Description Language (VHDL), Intelectual Property Protection

1. Introduction

Hackers' activity shows that it should secure not only software before stealing but also a hardware projects. Source code is stolen and built in other projects. Many of the security tools that can be used to protect the software were created like watermarks, fingerprints and obfuscation (Figure 1).

The purpose of watermarking [9,14,10] is to insert a mark (owner copyright notice) to the program that is invisible and difficult to remove. With the watermark, it is possible to prove the owner rights to the code or project.

Next technique for protecting projects is fingerprinting [7,6]. Fingerprinting is similar to watermarking, except a different (unique) secret message is embedded in every distributed cover message. A typical fingerprint includes a seller, product, and customer identification numbers. This allow to detect when and where theft has occurred and to trace the copyright offender.

[1] Faculty of Computer Science, Bialystok Technical University, Białystok

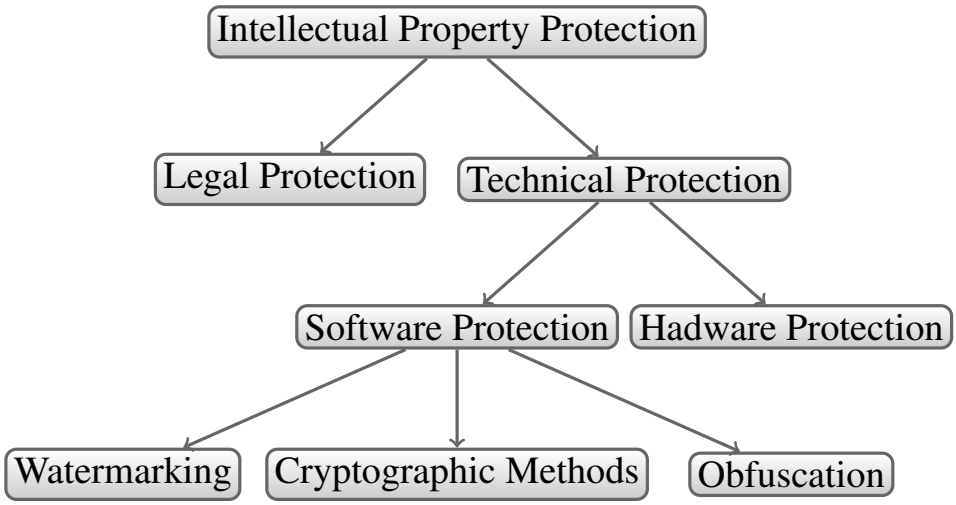

Fig. 1. Ways of intelectual property protection.

The last one technique is obfuscation [5] - one of the most modern and effective software protection technique. Obfuscation alters the code so that it is very difficult to read and understand. This type of protection does not protect against illegal copying and execution. The aim of obfuscation is prevention against stealing parts of source code (analysing and rebuilding in) and on the reverse engineering.

Subject of obfuscation is so young so it is hard to tell about its history. The first articles date from the second half of nineties. The precursor of subject was Christian Collberg. He wrote the first publication connected with obfuscation [5]. In several articles, which coauthor he is, Collberg presents techniques of code obfuscation and divide them into four groups by the target application - layout, data, control and preventive transformations. A lot of articles were written but no one concerned obfuscation of VHDL code. It is very easy to change code of other language like JAVA or C# because size and execute time of transformed program is not so significant. In VHDL these metrics are the most important for a software engineer.

The purpose of our work is to study obfuscation techniques and choose most suitable of them to protecting intellectual property rights on VHDL source code. We should remember that this language in comparison with other most frequently used high level languages is, in his building, unique. VHDL language makes uses of parallel processing and a lot of obfuscation techniques which can not be used in such languages like C# or Java are very useful and relatively cheep.

2. Definition of obfuscation

For people who are not familiar with the subject of obfuscation, it is intentional action conducting to modify the software code in such way that it becomes difficult to understand. Gaël Hachez in [9] changed definition of obfuscation created by Collberg in [5] and also used in [8] to:

Definition 1 (Obfuscating Transformation) $P \xrightarrow{\tau} P'$ be a transformation of a source program P into a target program P'. $P \xrightarrow{\tau} P'$ is an obfuscating transformation, if P and P' have the same observable behavior. More precisely, in order for $P \xrightarrow{\tau} P'$ to be a legal obfuscating transformation the following conditions must hold:

a) If P fails to terminate or terminates with an error condition, then P' fails to terminate or terminates with an error.
b) Otherwise, P' must terminate and produce the same output as P.

We should remember, that every program P' will be possible to reconstruct to P'', which will have very similar structure to P. Obviously, such operation will absorb much time and costs, but always it will be possible [3]. Therefore problem of obfuscation is not a protection before decompilation of program, but makes it very difficult.

Listing 1.1. Winner of 1st International Obfuscated C Code Contest in category Dishonorable mention - 1984

```
int i;main(){for(;i["]<i;++i){--i;}"];read('-'-'-',i+++"hell\
o,␣world!\n",'/'/'/'));}read(j,i,p){write(j/p+p,i---j,i/i);}
```

In Listing 1.1 is showed program source code anonimous.c - winner of 1st International Obfuscated C Code Contest in category Dishonorable mention. Obfuscated example shows how complicated analysis of protected code may be. For people who would like to take analysis of code and check its correctness we place base code (Listing 1.2).

Listing 1.2. Hello word. Code before obfuscation.

```
#include<stdio.h>
int main(void){
    printf("Hello␣World!\n");
    return 0;}
```

3. Quality characteristics - theoretical background

Before we attempt to describe obfuscation transforms, we need to be able to evaluate their quality. In this section we describe theoretical background of quality code measurement. We will also examine different new approaches to this problem.

3.1 International Standard ISO/IEC-9126

In 1991 the International Standards Organization introduced standard ISO/IEC-9126 [1], whose task was to define the standards and requirements for the description of the software. In subsequent years, the standard has been extended by four characteristics:

- ISO/IEC 9126-1:2001 Quality model
- ISO/IEC 9126-2:2003 External metrics
- ISO/IEC 9126-3:2003 Internal metrics
- ISO/IEC 9126-4:2004 Quality in use metrics

It is currently the most widely known and wide-spread quality standard of software. The standard ISO/IEC-9126 is often confused with the characteristics ISO/IEC-9126-1:2001 [2] introduced in 2001. The model defines the following six main quality characteristics:

- Functionality - a collection of attributes that describe a set of software functions and their designated properties.
- Reliability - a collection of attributes describing the capabilities of software to maintain the specified requirements as to its stability under strictly defined conditions of work.
- Usability - a collection of attributes that describe the level of work needed to learn to use the software and the subjective assessment of the attractiveness of the software expressed by group members.
- Efficiency - a measurement of the use of system resources compared to the performance of software subject to certain conditions.
- Maintainability - a collection of attributes that describe the level of work required to bring about changes in the software.
- Portability - a collection of attributes describing the product the ability to transfer programming between other systems.

They are supported by less significant characteristics in order to increase the precision of individual metrics.

3.2 Collberg's quality measures

In 1997 in the technical report [5] Christian Collberg with Clark Thomborson and Douglas Low defined the concept of obfuscation and attempted to assess its quality.

They defined quality of obfuscation as combination of:

- Potency - degree of difficulty understanding obfuscated code for the analysing person. Influence on the power of transformation have: introduce new class and methods, increase predicates and nestling level of conditional, increase the highest of the inheritance tree, increase long-range variables dependencies.
- Resilience - how much time a developer has to spend to implement automatic deobfuscator, which can effectively lower the power of transform.
- Cost - metric applies to both increase the cost of the implementation of the program as well as increased demand for resources like memory or CPU time.

In the lectures [16] Collberg previously defined range of metrics expends by a descriptive metric stealth. Its estimate is dependent on the context in which it was used.

3.3 Petryk's software metrics

In [15] Siarhei Petryk with Ireneusz Mrozek and Vyacheslav N. Yarmolik proposed metric composed from two submetrics. First group based on code complexity like the number of conditional operators compared to all words in the program or average depth of branching operators. The second based on program execution time and its length. Values of metrics from first class should be maximized whereas from the second class should be minimized. They assumed that the best obfuscator optimise the program code simultaneously makes source code more confusing and harder to interpret.

4. Obfuscation quality - new approach

Presented above metrics have been developed for non hardware languages. For example size of the code does not have such signification in designing hardware systems in opposite to the software projects. We are not bound by the amount of RAM or CPU speed. Above metrics do not take into account the signal delay propagation time on critical path which is one of most significant metric in digital designs and not occur in software projects. Therefore, there is a need to develop metrics which are intended solely to evaluate the quality obfuscation transforms used in hardware projects.

First step in design quality metric is divide metrics in two groups:

- critical metrics - most important metrics like the path delay of the system and the amount of used area on a chip.
- less important metric but not meaningless like complexity and code size.

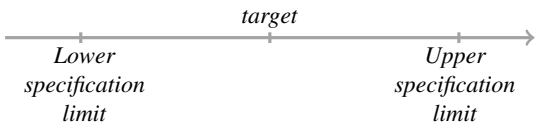

Fig. 2. Target and tolerance for metrics.

ISO/IEC-9126 [1] does not define how to measure metrics described in it. Some of quality attributes are subjective and difficult to measure. The standard describe only how the metrics should work and measure of them leaves for experts. We can not rely on metrics which estimate will be so costly. By expensive we understand the time effort and resources needed for estimate. The better expert that his time is more and more expensive.
Effective metrics satisfy the following conditions:

- performance is clearly defined in a measurable entity - the metric is quantifiable.
- exists a capable system to measure metric in the entity.

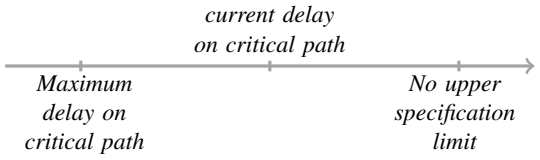

Fig. 3. Target and tolerance metric for critical path delay in obfuscated design.

The second criteria that metrics must be estimated taking into account is the parameters imposed by the user. In Figure 2 is presented target value of metric witch allowable deviation where a characteristic will be still acceptable. Lower specification limit (LSL) and upper specification limit (USL) are break points where make

worse performance will make design unable to work properly and improve make design extremely useful.

It is mean that user has to know metric or estimated value expected ranges for obfuscated design. Figure 3 present maximal delay on critical path propagation of protected design source code. The propagation delay reflects how fast system can operate and is usually considered as speed of the system. Minimal delay is not defined because delay lower than maximal or in special cases lower than nominal delay will be always acceptable if we consider only combinational circuits. In other cases we have to remember about triggers time specification - setup time and hold time. We should note that reach of low value of propagation delay will lead to high costs of code transformation.

The second most important metric is area on the target chip used by obfuscated design. As in the case of propagation delay user does not define the upper value of the used area because it is not necessary and any value in range below maximal would be acceptable. Design area greater than maximal would not fit in target chip. Circuit area combined with propagation delay is most important design criteria in digital systems.

The third most significant metric is cost of apply obfuscation transformation. We count the cost as effort of resources such as time of execute transformation, implementation effort and human resources needed.

$$E_1 = a_1 \frac{t_d}{t'_d} + a_2 \frac{area}{area'} + \frac{a_3}{E_c}$$

$$LSL_{t_d} = \frac{t_d}{t_{d_{max}}} \Rightarrow \frac{t_d}{t_{d_{max}}} \leqslant \frac{t_d}{t'_d} \Rightarrow t_{d_{max}} \geqslant t'_d$$

$$LSL_{area} = \frac{area}{area_{max}} \Rightarrow \frac{area}{area_{max}} \leqslant \frac{area}{area'} \Rightarrow area_{max} \geqslant area'$$

$$E_c = \{free \mapsto 1,\ cheap \mapsto 2,\ expensive \mapsto 2^2,\ very\ expensive \mapsto 2^3\}$$

t_d	- nominal value of propagation delay on critical path
$t_{d_{max}}$	- maximal value of propagation delay on critical path
t'_d	- value of propagation delay on critical path after obfuscation transformation applying
$area$	- nominal value of used area
$area_{max}$	- maximal value of used area
$area'$	- value of used area after obfuscation transformation applying
E_c	- cost of apply obfuscation transformation
a_1, a_2, a_3	- weight of the importance of metrics

The second group describes how good obfuscated code is. It is very easy to describe how correctly written code should look but it is very difficult to reflect it in metrics. The first easiest to estimate submetric is number of lines of code (LOC). Counting lines of code is not reliable in high level languages therefore quality metric should based on instruction statements to (IS). More instruction statements to line of code increase submetric value and makes analysis difficult.

Other useful metric is cyclomatic complexity defined by Thomas McCabe in [13]. The measure describe the number of linearly independent (distinct) paths through a program's flow graph. To have poor testability and maintainability we recommends that program module should exceed a cyclomatic complexity of 10. This high level of metric will fuzzy logical pattern of design.

Our quality metric lets user to choose more suitable metrics for him. User might chose for example:

- one or more from Halstead metrics [11] - metric currently less used
- Henry and Kafura's structure complexity metric [12] - increasing fan-in and fan-out will decrease readability and analyzability of project

$$E_2 = a_4 \frac{E'_{CC}}{E_{CC}} + a_5 \frac{E'_{IS/LOC}}{E_{IS/LOC}} \underbrace{+a_6 \frac{V'_p}{V_p} + a_7 \frac{C'_P}{C_P} + \cdots}_{optional}$$

$$E_{IS/LOC} = \frac{IS}{LOC}$$

CC - Cyclomatic Complexity
IS - Instruction Statements
LOC - Lines Of Code
V - Halstead Volume
C_P - Henry and Kafura's structure complexity
$a_4, \ldots$ - weight of the importance of metrics

On the basis of previous parts of metrics we achieved our goal:

$$E = E_1 + E_2$$

E_1 is a group of the most important metrics for digital designs and E_2 represents complexity of obfuscated code.

5. Estimation quality of obfuscating transformation

Layout obfuscation [5] relies on changing (removing) information in the source code that does not effect operation of program. It is often called free because does not influence on size of program neither on its speed of operation after transformation. There is one-way transformation because once deleted information can not be restored. Amount of helpful information decreases for analysing person. This technique contains removing comments, descriptions and changing names of variables which suggest what is it for. For scrambling identifiers does not change propagation delay on critical path nor project area on the target chip.

$$E_{sc} = a_1 + a_2 + a_3 + a_4 + a_5$$

$$\frac{t_d}{t'_d} = \frac{area}{area'} = \frac{E'_{CC}}{E_{CC}} = \frac{E'_{IS/LOC}}{E_{IS/LOC}} = 1$$

$$E_c = free = 1$$

It is easy to see that layout obfuscation is one of the cheapest techniques allowed to protect intellectual property.

There is transformation which does not appear in other programming languages. In view of its specification VHDL is based on parallel processing instruction. Almost every code written in VHDL may be converted from or to sequential processing.

Listing 1.3. Conversion of sequential processing to parallel

```
process (x)
        variable tmp : std_logic;
begin
        tmp := '0';
        for i in x'range loop
                tmp := tmp xor x(i);
        end loop;
        y <= tmp;
end process;
                    ⇓
tmp <= tmp(N-2 downto 0)&'0' xor x;
y <= tmp(N-1);
```

Mentioned above transform is not difficult to realize. Listing 1.3 presets two obfuscation techniques from group of control transformation. We used technique called unrolling loops connected with conversion of sequential processing to parallel [4]. Transformation does not influence on critical part of metric.

6. Conclusion

We have developed the first hardware design obfuscation quality metering scheme. The metric takes into account the signal delay propagation time on critical path and the amount of used area by design on a target chip. Moreover with this quality tool a user can choose obfuscation transforms answering his requirements of created project.

References

[1] ISO/IEC-9126, International Standard ISO/IEC. In *Information technology: Software product evaluation: Quality characteristics and guidelines for their use*. International Standards Organisation, 1991.

[2] ISO/IEC-9126-1:2001, International Standard ISO/IEC 9126. In *Information Technology – Product Quality – Part1: Quality Model*. International Standard Organization, June 2001.

[3] Impagliazzo R. Rudich S. Sahai A. Vadhan S. Yang K. Barak B., Goldreich O.: On the (im)possibility of obfuscating programs, Lecture Notes in Computer Science, 2139:1–14, 2001.

[4] Yarmolik V. N. Brzozowski M.: Vhdl obfuscation techniques for protecting intellectual property rights on design, 5th IEEE East-West Design and Test Symposium, pages 371–375, 2007.

[5] Low D. Collberg C., Thomborson C.: A taxonomy of obfuscating transformations, Technical report, July 1997.

[6] Thomborson C. Collberg C.: The limits of software watermarking, 1998.

[7] Thomborson C. Collberg C.: Watermarking, tamper-proofing, and obfuscation – tools for software protecti on, Technical Report TR00-03, Thursday, 10 2000.

[8] Low D. Collberg C. S., Thomborson C. D.: Breaking abstractions and unstructuring data structures, In International Conference on Computer Languages, pages 28–38, 1998.

[9] Hachez G.: A comparative study of software protection tools suited for e-commerce with contributions to software watermarking and smart cards, Universite Catholique de Louvain, March 2003.

[10] Wroblewski G.: General Method of Program Code Obfuscation, PhD thesis, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002.

[11] Halstead M. H.: Elements of Software Science, North-Holland, Amsterdam, The Netherlands, 1977.

[12] Kafura D. Henry S.: Software structure metrics based on information flow, 7(5):510–518, September 1981.

[13] McCabe T. J.: A complexity measure, IEEE Trans. Software Eng., 2(4):308–320, 1976.
[14] Petitcolas F. A. P. Katzenbeisser S.: Information hiding - techniques for steganography and digital watermarking, Artech House, Norwood, 2000.
[15] Yarmolik V. N. Petryk S., Mrozek I.: Efficiency of obfuscation method based on control transformation, In ACS, pages 351–359, 2006.
[16] www.cs.arizona.edu/∼collberg/Research/Publications/index.html.

JAKOŚĆ OBFUSKACJI PROJEKTÓW SPRZĘTOWYCH

Streszczenie Obfuskacja jest techniką przekształcania kodu źródłowego oprogramowania, który zachowuje swoje działanie semantyczne, ale znacząco zostaje utrudniona jego analiza oraz zrozumienie. Swoje badania skupiliśmy na sprzętowym języku Very High Speed Integrated Circuits Hardware Description Language (VHDL). W poprzednich pracach przestawiliśmy szereg transformat zaasymiowanych z języków wysokiego poziomu na potrzeby projektów sprzętowych oraz zaproponowaliśmy nowe nie występujące w innych językach niż VHDL. Kolejnym krokiem jaki należy wykonać jest ustalenie kryteriów dzięki którym będzie można ocenić jakość analizowanych transformat. W artykule przedstawimy dotychczas używane metody oceny jakości oprogramowania oraz przeprowadzimy ich ocenę na potrzeby ochrony własności intelektualnej projektów sprzętowych. Następnym krokiem będzie przedstawienie nowego sposobu oceny jakości obfuskacji z uwzględnieniem wymagań jakie stawiają przed programistą projekty sprzętowe.

Słowa kluczowe: Obfuskacja, VHDL, ochrona własności intelektualnej