社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－CSEC－42 (43)
2008 / 7 / 25

出力VHDLコードに透かしを埋め込むCADツールの不正コピー検知方式

福島　和英[†]　清本　晋作[†]　田中　俊昭[†]　櫻井　幸一[††]　安浦　寛人[††]

[†] 株式会社KDDI研究所　〒356-8502 埼玉県ふじみ野市大原 2-1-15
[††] 九州大学 大学院システム情報科学研究院　〒819-0395 福岡県福岡市西区元岡 744
E-mail: †{ka-fukushima,kiyomoto,toshi}@kddilabs.jp, ††sakurai@csce.kyushu-u.ac.jp,
†††yasuura@c.csce.kyushu-u.ac.jp

あらまし　本稿では，CADツールの不正コピーを検知する方法について検討する．我々は，考えられる複数の方式を比較し，それから，CADツールが出力するVHDLコードに埋め込んだ透かしにより，ツールの不正コピーを検知するフレームワークを提案する．はじめに，透かしとして埋め込む情報についての検討を行なう．次に，具体的な透かしの埋め込み方法として，暗号技術により透かしを変換する手法について検討する．最後に，VHDLコードに任意のビット列を埋め込む手法について検討する．これらの方法においては，安全性と効率性がトレードオフの関係にあるため，システムの制約条件を考慮して，用いる方法を選択する必要がある．提案するフレームワークは，コンパイラ・アセンブラ，ソフトウェア難読化ツール，マルチメディアオーサリングツールなど，透かし可能なデータを出力するプログラムの保護にも利用できる．

キーワード　CADツール，VHDL，不正コピー検知，電子透かし，ソフトウェア透かし，ソフトウェア難読化

Illegal Copy Detection Framework for CAD Tools based on Watermarks Embedded in VHDL Codes

Kazuhide FUKUSHIMA[†], Shinsaku KIYOMOTO[†], Toshiaki TANAKA[†], Kouichi SAKURAI[††], and Hiroto YASUURA[††]

[†] KDDI R&D Laboratories, Inc. 2-1-15 Ohara Fujimino, Saitama, 356-8502
[††] Department of Computer Science and Communication Engineering, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka, Fukuoka 819-0395,
E-mail: †{ka-fukushima,kiyomoto,toshi}@kddilabs.jp, ††sakurai@csce.kyushu-u.ac.jp,
†††yasuura@c.csce.kyushu-u.ac.jp

Abstract　In this paper, we discuss an illegal copy detection scheme for CAD tools. We compare possible solutions and propose an illegal copy detection framework for CAD tools based on watermarks embedded in VHDL codes. Then, we propose an illegal copy detection framework for CAD tools based on watermarks embedded in output VHDL codes. First, we study identification information to be embedded as a watermark. Next, we investigate a transformation mechanism for identification information using a cryptographic technique. We can prevent leaks of confidential information and alterations/removals of watermarks, and can detect alterations to VHDL codes by transforming the identification information to be watermarked. Finally, we propose watermark embedding mechanisms for VHDL codes. These embedding mechanisms involve a tradeoff between security and execution efficiency; thus, they should be selected giving due consideration to system requirements. Our framework can be applied to the protection of any software that outputs watermarkable data such as compilers/assemblers, software obfuscation tools, and multimedia authoring tools.

Key words　CAD Tool, VHDL, Illegal Copy Detection, Digital Watermarking, Software Watermarking, Software Obfuscation

1. Introduction

1.1 Background

Recently, system LSIs have been designed as source codes specified by hardware description language such as VHSIC Hardware Description Language (VHDL). In addition, the development costs of system LSIs have increased as they now have highly integrated large circuits. Design reuse and reuse-based design are widely considered as the most efficient way to reduce these costs. Existing circuits may be reused as subcircuits to construct a much higher performance LSI. However, we must guarantee provider's intellectual property rights and royalties in order to further promote reuse. Thus, many watermarking schemes for hardware description languages that embed copyright information have been proposed. Embedded watermarks are used to prove the ownership of the copyright of the code.

On the other hand, CAD tools are used to design system LSIs. They enable the circuits of system LSIs to be designed through a GUI interface and output source codes of a hardware description language. Generally, these CAD tools are expensive and illegal copying of them is another major issue. However, there is no perfect copy protection scheme, though many schemes have been proposed. Thus, we need not just techniques that prevent illegal copying but also techniques to detect the fact that illegal copying has taken place.

This paper proposes an illegal copy detection framework for CAD tools based on watermarks embedded in output VHDL codes. Now, many kinds of software are used to create or transform data such as image data, audio data, movies, and programs. Some of them are used as expensive enterprise software and illegal copying of them is a critical issue. We can apply the proposed framework to any software that outputs watermarkable data such as compiler/assembler, software obfuscator, and multimedia authoring tool.

1.2 Related Work

1.2.1 Software Watermark

Digital watermarking is a technique that embeds auxiliary information into content. This information is used to prove the ownership of copyright of the content or trace the user who illegally distributes the content.

Previously, digital watermarking schemes aimed mainly at copyrighted content such as image data, audio data, and text data. These kinds of content have a high level of redundancy; therefore, in many cases, a user cannot recognize that content is different if several bits are changed. Thus, we can embed a lot of information into these data.

On the other hand, software has less redundancy than image data, audio data, and text data. For example, if only one bit is changed, software may not run correctly. Therefore, it is more difficult to embed watermarks in software. However, watermarking schemes for software have also been proposed in order to prove that illegal copying of software has taken place.

Monden, et al. [1] proposed a scheme for Java byte codes that embeds a watermark into opcodes and numeric operands in dummy methods. Venkatesan et al. [2] proposed a scheme that represents a watermark as a control flow graph and merges it into the graph of the original program. These are static watermarking schemes; that is, a watermark can be detected without running a program. Collberg et al. [3] and Thomborson et al. [4] proposed a dynamic watermarking scheme, in which a watermark is output when a program is executed with specific input.

Watermarking schemes for hardware description languages have also been proposed. Horikawa et al. [5], Kubo et al. [6], and Yuan et al. [7] proposed schemes that embed watermarks into redundant hardware descriptions for *don't-care* conditions.

1.2.2 Software Obfuscation

An obfuscation scheme transforms an original source code or binary program into an obfuscated source code or program that is more complicated and difficult to analyze, while still preserving its functionality. That is, an obfuscated program gives the same output as that of the original program when both programs have an equivalent input.

Barak et al. [8] showed the existence of classes of functions that are not obfuscatable. However, some practical obfuscation schemes have been proposed. These schemes are intended to make the cost of analyzing a program higher than the value of this program. Monden et al. [9] and Gannod et al. [10] proposed an obfuscation scheme that obfuscates loops. Collberg et al. [11] proposed a scheme that inserts dummy instructions using conditional branching. Chan et al. [12] proposed a scheme that modifies identifiers contained in Java byte code in order to protect the code against decompilation. Sosonkin et al. [13] proposed a scheme that changes the structure of classes in a Java code by merging and dividing them. Obfuscation schemes with a theoretical basis have also been proposed. Wang et al. [14], Ogiso et al. [15] and Sakabe et al. [16] proposed obfuscation schemes based on NP-hard problems. Collberg et al. [18], Sato et al. [19], and Fukushima et al. [20] have both proposed obfuscation schemes that encode variables.

1.3 Our Contribution

In this paper, we discuss an illegal copy detection scheme for CAD tools. We compare the possible solutions: online license checking, offline license checking, and embedding a watermark in outputs. Then, we propose an illegal copy detection framework for CAD tools based on watermarks embedded in output VHDL codes. First, we study three candidates to be embedded as a watermark: 1) the execution history of the tool, 2) the tool ID associated with a user, and 3) the tool ID and PC information. Then, we propose a transformation mechanism for identification information using a cryptographic technique. We can prevent leaks of confidential information and alterations/removals of watermarks, and detect alterations made to VHDL codes by this transforming mechanism. Furthermore, we propose two watermark embedding mechanisms for VHDL codes: a mechanism using dummy instructions and a mechanism using an obfuscation technique in addition to an existing mechanism: a mechanism using redundant hardware description. These embedding mechanisms involve a tradeoff between security and execution efficiency; thus, they should be selected giving due consideration to system requirements. Our framework can be applied to the protection of any software that outputs watermarkable data such as compilers/assemblers, software obfuscation tools, and multimedia authoring tools.

2. Illegal Copy Detection Framework

2.1 Goal and Assumption

Our goal is to enable the manufacture of a CAD tool to check whether illegal copying has taken place or not by using watermarks embedded into output VHDL codes. We assume that a CAD tool is expensive and the number of users of this tool is limited, e.g., around 10. In this special situation, detecting the fact of illegal copying is important as well as tracing the illegal user. For example, one effective approach is to warn all users that the manufacturer will take legal action if it finds illegal distribution of this tool. An illegal user will think twice about distributing the CAD tool on the Internet if he/she fears that legal action will be taken. Instead, an illegal user may give a copy of the tool to a friend offline. Then, this friend may publish VHDL codes in order to provide or sell a circuit of his or her own design. Thus, we propose a framework that enables the illegal copying of a CAD tool to be detected based on watermarks embedded in output VHDL codes.

We assume that a secure software obfuscation scheme for the CAD tool is available. In our framework, the watermark embedding function added to the CAD tool is an essential function. If this function is removed or bypassed, our framework does not work. Thus, we must obfuscate the CAD tool in order to protect the watermark embedding function against analyses, alterations or removals.

2.2 Possible Solutions

We consider possible ways of detecting illegal copying of CAD tools. Generally, programs have a license checking mechanism to protect them against illegal use. There are two forms of program checking mechanisms: offline checking and online checking.

In online checking mechanisms, the program accesses a license management server to register ownership of a user when the program is installed in the user's PC. Or, it accesses the server to check whether the license is valid or not, whenever the program is started. However, this scheme has a drawback in that the program manufacturer has to build and manage an online license management server. Furthermore, network access between users and the license management server is needed. If the program cannot access the license management server, and an illegal user may not be detected.

The offline checking mechanism allows the license to be verified without network access. The general method of offline checking is to provide a tamper-proof device with the program. This tamper-proof device includes license information and/or a license checking mechanism. The tamper-proof device is difficult to copy and the program cannot be started without this device. However, this solution is not cost-effective because the program manufacturer has to provide an additional device for each user license.

Another solution is to embed license information in the output of the program as a watermark. CAD tools are used for generating VHDL codes; thus, a program manufacturer can discover if illegal copying of this tool has occurred by checking watermarks in the output VHDL codes. This solution may not detect illegal copying in real time; however, the detection process does not need online access or special hardware. Thus, we considered this approach as a way of detecting the illegal copying of CAD tools.

Table 1 shows a summary of the above discussion.

3. Our Protection Framework

We present the detailed description of our framework that enables detection of illegal copying of a CAD tool based on watermarks embedded in output VHDL codes. In our framework, a watermark embedding function is added to the CAD tool. This function embeds two kinds of watermarks into output VHDL codes. One is watermark $WM(M)$ that is used to detect illegal copying of the CAD tool. This watermark is automatically embedded by the watermark embedding function. The other is watermark $WM(U)$ that is used to detect illegal distribution or reuse of output VHDL codes. This watermark is selected by the user.

Figure 1 shows our framework. The manufacturer of the CAD tool and users of this tool delegate a distributor to detect copyright infringement of the CAD tool and output VHDL codes. The distributor can detect illegal copying of the CAD tool by extracting watermarks $WM(M_A)$ and $WM(M_B)$ from output VHDL codes. As discussed in Sect. 2.1, we do not always have to identify that Alice illegally gave a copy of the tool to Bob. Instead, it is only important to detect the fact of illegal copying itself. Furthermore, the distributor can also detect the illegal distribution of a VHDL code by extracting watermarks $WM(U_A)$ if Charlie steals the VHDL code Alice generated and publishes it as his own code.

There are many existing studies on watermarking schemes to prove the illegal reuse of VHDL code. Thus, we discuss only an embedding mechanism for $WM(M)$ in the following

Table 1 Summary of Possible Solutions

Solution	Advantages	Disadvantages
Online License Checking	Real-time detection	Network access
Offline License Checking	Local checking	Special hardware or logic needed
Watermark embedded in outputs	No special device or network needed	Impossible to detect in real time.

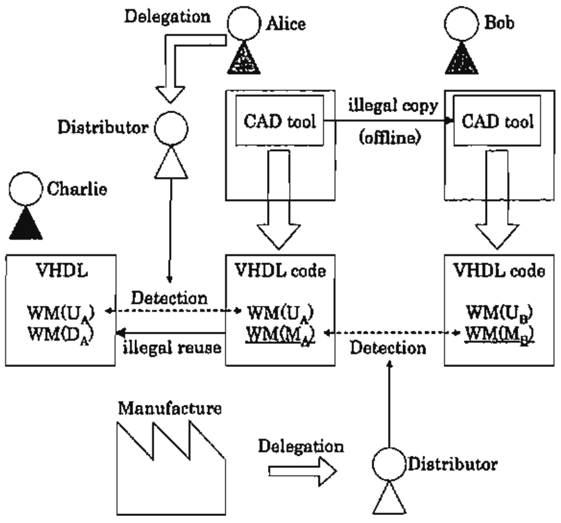

Figure 1 Our Framework

sections of this paper.

3.1 Identification Information to be Embedded

We consider three candidates to be embedded as watermark $WM(M)$: 1) the execution history of the tool, 2) the tool ID associated with a user, and 3) the tool ID and PC information.

3.1.1 Candidate 1: Execution History of the Tool

The CAD tool embeds the execution history into output VHDL codes. The execution history is a sequence of pairs of execution time and hardware information of the PC (PC information) where this tool is executed. This approach requires a trusted clock as well as a timestamp technique [21]. The PC information is dynamically obtained when the CAD tool is executed. Illegal copying of the CAD tool is detected by the watermark which indicates that this tool is executed on two or more PCs.

We assume that Alice executes the CAD tool at 9:00 on Jan. 1st and at 9:00 on Jan 2nd on her PC with PC information PC_A, and illegally gives a copy of this tool to Bob. Then, Bob executes the CAD tool at 9:00 on Jan 3rd on his PC with PC information PC_B, and obtains a VHDL code. The distributor can detect the fact of illegal coping from the watermark extracted from Bob's VHDL code since the execution history indicates that the tool has been executed on two PCs.

The CAD tool must store the execution history to embed it into output VHDL codes. For example, the tool itself may have the execution history not to be deleted by illegal users.

3.1.2 Candidate 2: Tool ID associated with a User

The CAD tool embeds the identification information of the tool (tool ID) into output VHDL codes. We assume that a distinct tool ID is assigned to each CAD tool and associated with a user when this tool is purchased. The pairs of a tool ID and user name must be registered by the distributor. Illegal copying of the CAD tool is detected by the watermark which indicates that the owner of an output VHDL code is different from the registered user of the tool. The distributor extracts a tool ID from the watermark in a VHDL code and seeks the user with whom this identification information is associated. Finally, the distributor checks whether the result is same as the user who publishes the VHDL code.

We assume a CAD tool that has the tool ID $Tool_A$ is associated with Alice. She illegally gives a copy of this tool to Bob. Then, Bob executes the CAD tool and obtains a VHDL code. The distributor can detect the fact of illegal copying from the watermark extracted from Bob's VHDL code since the extracted identification information indicates that this tool was purchased by Alice.

3.1.3 Candidate 3: Tool ID and PC Information

The CAD tool embeds the pair of the tool ID and PC information into output VHDL codes. Illegal copying of the CAD tool is detected by the watermarks which indicate that the same tool is executed on two or more PCs.

We assume Alice has the CAD tool with tool ID $Tool_A$. She executed this tool on her PC with PC information PC_A, and obtains a VHDL code. Then, she illegally gave a copy of the tool to Bob. Bob executed the CAD tool on his PC with PC information $Tool_B$, and obtains another VHDL code. The distributor can detect the fact of illegal copying from the watermarks $(Tool_A, PC_B)$ and $(Tool_A, PC_A)$ since the CAD tool with tool ID $Tool_A$ is executed on two PCs PC_A and PC_B.

3.1.4 Comparison

We compare the above three candidates.

Security: An illegal user can avoid detection by a backup attack when the execution history is embedded to VHDL codes. A user of the CAD tool can make a copy of the tool with no execution history by backing up the tool before he or she executes it.

Figure 2 shows an example. We assume that Alice buys the CAD tool and backs up this tool before executing it, say at 8:00 on Jan. 1st. Then, she executes the tool on her PC with PC_A at 9:00 on Jan. 1st and 9:00 on Jan. 2nd. She can give Bob the backed up tool as the CAD tool with

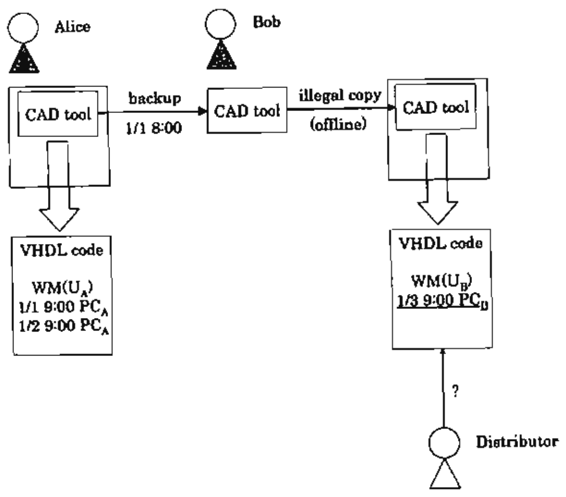

Figure 2 Backup Attack

no execution history. Bob executes the tool on his PC with PC_B at 9:00 on Jan 3rd. The distributor cannot detect the fact of illegal copying since both watermarks indicate that Alice's tool is only executed on her PC and Bob's tool is only executed on his PC.

This attack utilizes the fact that an execution history is dynamic i.e., it changes for every execution of this tool. On the other hand, it cannot be applied to other two candidates since this information does not contain dynamic data.

Operational Cost: The distributor must manage user information when the tool ID associated with a user is embedded into a VHDL code. Thus, this watermark entails a high operational cost. Furthermore, it does not work if the user of the CAD tool is different from the user who publishes the VHDL code. We assume Alice designs a system LSI using the CAD tool and Charlie, who is her colleague, publishes the VHDL code. In this case, the watermark indicates that the tool is illegally copied from Alice to Charlie even if she legally uses the tool.

Generate Unique PC Information: The distributor must generate unique PC information when the execution history or pair of the tool ID and PC information is embedded. We can consider the use of hardware information such as the identifier of a processor/hard disk drive/motherboard or the MAC address of a network interface card. However, the MAC addresses of some network interface cards can be changed. Furthermore, a user can replace a processor and/or hard disk drive. These identifiers may not uniquely specify a PC. Thus, for example, we may use the identifier of a motherboard to generate PC information since the motherboard is considered to be the most essential component of a PC.

Detection of Illegal Copying: Illegal copying can be detect from only one VHDL code when the execution history or tool ID associated with a user is embedded. However, we need two or more VHDL codes to detect illegal copying when the pair of the tool ID and PC information is embedded. That is, we have to find two VHDL codes that contain the same tool ID and different PC information.

Remark: We proposed three types of identification information to be embedded as a watermark. Table 2 compares these candidates based on the above discussion. The candidates have advantages and disadvantage; thus, we should select identification information after due consideration.

3.2 How to Embed Identification Information

We explain our technique for embedding identification information. The technique for embedding identification information consists of two steps: transformation of identification information and hiding the information in VHDL codes.

3.2.1 Transformation of Identification Information

In the first step of embedding identification information, the tool transforms the identification information into secure encrypted data. The transformation technique is described as Figure 3. First, the tool calculates a hash value of the identification information and concatenates the hash value with the identification information, and then the tool encrypts the data using the distributor's public key. A public-key based random encryption algorithm such as RSA-OAEP [22], [23] is used for the encryption. The public key is securely embedded in valid CAD tools by the distributor. The distributor has a private key for decrypting the data. Thus, the distributor obtains encrypted data from VHDL codes, decrypts it, and verifies the hash value of the identification information. The objective of the transformation is as follows;

- Alteration of the identification information can be detected. If the attacker changes a VHDL code including the watermark without the public key, the distributor can detect this since the distributor cannot correctly decrypt the identification information or fails to verify the identification information. Thus, the distributor can consider that the VHDL code is generated in an illegal way.
- Embedded information becomes pseudorandom data. The tool transforms the identification information into pseudorandom data using a random encryption algorithm. Different embedded data are generated for each transformation from the same identification information. By randomizing embedded data, the attacker will find it more difficult to break the hiding techniques discussed in Sect. 3.2.2.
- Embedded information becomes secret information. A valid distributor who has a private key can decrypt the iden-

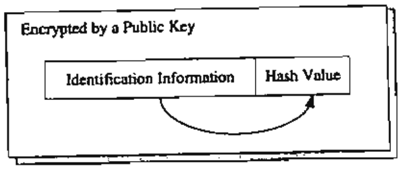

Figure 3 Transformation of Identification Information

Table 2 Comparison of Possible Candidates

Identification Information	Assumptions	Advantages	Disadvantages
Execution History	Trusted clock PC information	Detection using a code Low cost	Low resistance to backup attack
Tool ID associated with User	User registration	Detection using a code High resistance	High cost of managing user information
Tool ID and PC Information	PC Information	High resistance Low cost	Detection using multiple codes

tification information and detect illegal copying. The identification information may include sensitive information related to a user's privacy. Thus, the information should only be traceable by the distributor.

3.2.2 Hiding Identification Information

Transformed identification information is hidden in VHDL codes using the following embedding techniques. Then, we must obscure the position of the watermark to protect against coalition attacks.

a) Embedding Identification Information

We propose two watermark embedding mechanisms 2) and 3) for VHDL codes in addition to an existing mechanism 1).

Proposed mechanisms 2) and 3) use an obfuscation scheme, the xor-encoding scheme. This scheme encodes multiple variables are simultaneously encoded to multiple variables using a key, i.e., a Boolean matrix and an integer vector. The encoding process uses only exclusive-or operations.

1) Mechanism Using Redundant Hardware Descriptions: Horikawa et al. [5], Kubo et al. [6], and Yuan et al. [7] proposed schemes that embed watermarks in redundant hardware descriptions for *don't-care* conditions. In a VHDL code, the behavior of an operational unit is described by a conditional branch instruction `if` statement or `case` statement where each signal is used as a branch condition. We assume a unit whose behaviors are defined for 3-bit input signals "000", "001", "010", "011", "100", and "101". The behaviors of this unit for signals "110" and "111" are not defined and these signals are *don't-care*. Thus, we can consider that the hardware descriptions for these conditions are redundant.

A watermark can be embedded into numerical data and operations in these redundant hardware descriptions. Numerical data 00000001 of instruction `a <= "00000001";` can be replaced with any data. Thus, we can embed 8-bit information by replacing this data with arbitrary data. Additionally, operation `+` can be replaced with `-`, `*`, `/`, `%`, `and`, `or`, or `xor`. These eight operations are commutative. We can embed 3-bit information by replacing this information with any of these eight opcodes when we encounter one of them in a program. That is, we can express them in the data from "000" through "111" by assigning "000" to `+`, "001" to `-`, "010" to `*`, ..., and "111" to `ixor`.

For example, we can embed 6-bit information in the VHDL code in Fig. 4 by adding the redundant hardware description for signals "110" and "111".

```
case Signal is
  when "000" => a <= "000";
  when "001" => a <= "010";
  when "010" => a <= "100";
  when "011" => a <= "110";
  when "100" => a <= "001";
  when "101" => a <= "011";
  -- redundant hardware descriptions --
  when "110" => a <= "011";
  when "111" => a <= "111";
end case;
```

Figure 4 Sample of Redundant Hardware Descriptions

2) Mechanisms Using Dummy Instructions: Monden et al. [1] proposed a watermarking scheme for a Java program that embedded a watermark using a dummy method. We propose a mechanism based on the same technique that can be applied to VHDL codes. We can add dummy instructions that are never actually executed while keeping the functionality of a VHDL code. Then, a watermark can be embedded into numerical data and operations in these dummy instructions.

For example, we can embed 22-bit information in a VHDL code in Fig. 5 by replacing two operations and two 8-bit numerical data. We can extract the 22-bit sequence "0000010010011111011010" from this dummy method according to the above rules. `false_cond` in the `if` statement is a predicate that is always evaluated as false; thus, the instructions in the `if` statement are never executed.

In this mechanism, a sophisticated predicate is required to conceal the fact that the instructions in the `if` statement are dummy instructions. If we use a trivial predicate such as `i*i<0`, an illegal user may find that this predicate is evaluated as false and these instructions are dummy instructions to embed a watermark. Then, the illegal user may remove or alter these instructions to modify the watermark. Thus, we need a predicate that is difficult to analyze while it is always evaluated as false. This predicate is called an opaque predicate. We can construct an opaque predicate using the xor-encoding scheme. This obfuscation scheme encodes multiple variables using a key, i.e., a matrix and vector as follows:

$$\begin{pmatrix} \mathtt{A} \\ \mathtt{B} \\ \mathtt{C} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathtt{a} \\ \mathtt{b} \end{pmatrix} \oplus \begin{pmatrix} 10010110 \\ 01100101 \\ 10100101 \end{pmatrix}.$$

```
if (false_cond) then
  a <= a + "00100100";
  b <= b xor "11011010";
end if;
```

Figure 5 Sample of Dummy Instructions

In this situation, two variables a and b are encoded to A, B, and C. Encoded variables A, B, and C satisfy a non-trivial relation $A \oplus B \oplus C \oplus 01010110 = 0$. Thus, we can construct an opaque predicate using this relation. For example, `if (A xor B xor C xor 01010110 < 0)` is a false opaque predicate. On the other hand, normal predicate `if (a < 0)` that is sometimes evaluated as true and sometimes as false is transformed into `if (A xor B xor 11110011 < 0)`. Thus, it is difficult to distinguish an opaque predicate and a normal predicate since they have the same form.

3) Mechanism Using Obfuscation Technique:
We propose a mechanism based on the above obfuscation scheme. We can embed a watermark into a dummy variable by assigning arbitrary data to this variable. However, an illegal user can specify the dummy variable since the value of this variable is not used. To solve this issue, we use the xor-encoding scheme. This obfuscation scheme makes it difficult to specify the dummy variable since the value of a dummy variable is mixed with those of the existing valuables. Only the distributor who knows the key can decode the variables and obtain the watermark.

Figure 6 shows the VHDL code of an 8-bit counter. A watermark is embedded in variable dummy; however, the value of this variable is not used. Thus, an illegal user may know that this variable is a variable to embed a watermark. Figure 7 shows the obfuscated code where variables dummy and counter are encoded to D and C while keeping the functionality of the code in Fig. 6. These variables are encoded according to the rule

$$\begin{pmatrix} \mathtt{D} \\ \mathtt{C} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \mathtt{dummy} \\ \mathtt{counter} \end{pmatrix} \oplus \begin{pmatrix} 00101111 \\ 10110100 \end{pmatrix}.$$

The value of variable dummy is mixed with that of the existing variable counter. An illegal user cannot specify the dummy variable since both values of C and D are used to calculate the next value. The distributor who knows the encoding rule can decode variables D and C and obtain the watermark from variable dummy.

Remarks: We compare the security of the three watermark embedding mechanisms. The mechanism using redundant hardware description and the mechanisms using dummy instructions embed a watermark into redundant parts of a program; thus, these mechanisms do not affect the execution efficiency of the circuit described by a VHDL code. However, if an illegal user finds that they are redundant parts of this VHDL code, he or she may alter or remove the watermark by modifying these parts. On the other hand, the mechanism

```
counter:process (clock);
begin
  -- dummy variable --
  dummy <= "10010010";
  count <= "0";
  if (clock'event and clock = "1") then
  count <= count + 1;
  end if;
end process
```

Figure 6 Sample of Dummy Variable

```
counter:process (clock);
begin
  D <= "10111101";
  C <= "10110100";
  if (clock'event and clock = "1") then
    C <= D xor ((D xor C xor "10011011")
      + 1) xor "10011011";
    count <= D xor C xor "00101111";
  end if;
end process
```

Figure 7 Obfuscated VHDL Code

using the obfuscation technique decreases the execution efficiency of the circuit. However, this mechanism provides higher security since watermarked data is mixed with the existing data in the original VHDL code. It is difficult for illegal users to alter or remove only the watermarked data.

Therefore, these mechanisms involve a tradeoff between security and execution efficiency; thus, they should be selected giving due consideration to system requirements.

b) Obscure the Position of a Watermark

Existing watermarking schemes for VHDL codes are designed to embed common copyright information. Thus, these schemes cannot protect against coalition attacks when they are used to embed distinct identification information. Some parts of the identification information in a VHDL code are distinct for each user while the other parts are the same for all users. As a result, illegal users can collude to identify the position of a watermark by comparing their VHDL codes and finding the parts that are different. A comparison of VHDL codes can be carried out using the `diff` command in UNIX. They can easily alter or remove the watermarks by modifying parts of the watermark in VHDL codes.

We can apply an obfuscation scheme so that each user has a VHDL code with a distinct structure. A software obfuscation scheme transforms a code while keeping its functionality. Some obfuscation schemes for VHDL codes are available [24]~[26]. In this case, a desirable obfuscation scheme is a probabilistic scheme or a scheme with a parameter, e.g., key, which controls the transformation. These obfuscation schemes provide a distinct transformation for each user; thus, they make it difficult to specify the watermark by comparing VHDL codes.

4. Conclusion

In this paper, we presented an illegal copy detection scheme for CAD tools. We compared the possible solutions: online license checking, offline license checking, and watermark embedding in outputs. Then, we proposed an illegal copy detection framework for VHDL code based on watermarks embedded in output VHDL codes. First, we studied three candidates to be embedded as a watermark: 1) the execution history of the tool, 2) the tool ID associated with a user, and 3) the tool ID and PC information. Then, we proposed a transformation mechanism for identification information using a cryptographic technique. We can prevent leaks of confidential information and alterations/removals of watermarks, and detect alterations of VHDL codes by this transforming mechanism. Furthermore, we proposed two watermark embedding mechanisms for VHDL codes: a mechanism using dummy instructions and a mechanism using an obfuscation technique in addition to an existing mechanism: a mechanism using a redundant hardware description. These embedding mechanisms involve a tradeoff between security and execution efficiency; thus, they should be selected giving due consideration to system requirements. Our framework can be applied to the protection of any software that outputs watermarkable data such as compilers/assemblers, software obfuscation tools, and multimedia authoring tools.

References

[1] A. Monden, H. Iida, K. Matsumoto, K. Inoue, and K. Torii, "A practical method for watermarking java programs," Proc. 24th Computer Software and Applications Conference (COMPSAC2000), pp.191–197, 2000.

[2] R. Venkatesan, V. Vazirani, and S. Sinha, "A graph theoretic approach to software watermarking," Proc. 4th International Information Hiding Workshop (IHW2001), Lecture Notes in Computer Science 2137, pp.157–168, 2001.

[3] C. Collberg, and C. Thomborson, "Software watermarking: Models and dynamic embeddings," Proc. Principles of Programming Languages 1999 (POPL1999), pp.311–324, 1999.

[4] C. Thomborson, J. Nagra, R. Somaraju, and C. He, "Tamper-proofing software watermarks," Proc. 2nd Australasian Information Security Workshop (AISW2004), pp.27–36, 2004.

[5] T. Horikawa, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Soft ip protection algorithms (in japanese)," Proc. 14st Workshop on Circuits and Systems in Karuizawa, pp.591–596, 4 2001.

[6] Y. Kubo, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A watermarking technique for vhdl based on additional redundant descriptions (in japanese)," Proc. DA Symposium 2003, pp.37–42, 7 2003.

[7] L. Yuan, P.R. Pari, and G. Qu, "Soft ip protection: Watermarking hdl codes," Proc. 6th International Information Hiding Workshop (IHW2004), pp.224–238, 2004.

[8] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yangpages, "On the (im)possibility of software obfuscation," Proc. Advances in Cryptology (CRYPTO2001), Lecture Note in Computer Science, vol.2139, pp.1–18, 2001.

[9] A. Monden, Y. Takada, and K. Torii, "Methods for scrambling program containing loops," IEICE Transactions on Information and Systems, Part 1 (Japanese Edition), vol.J80-D-I, no.7, pp.644–652, 1997.

[10] G.C. Gannod, and B.H.C. Cheng, "Using informal and formal techniques for the reverse engineering of c programs," Proc. IEEE International Conference on Software Maintenance 1996 (ICSM'96), pp.265–275, 1996.

[11] C. Collberg, and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation — tools for software protection," IEEE Transactions on Software Maintenance, vol.28, no.8, pp.735–746, 2002.

[12] J.T. Chan, and W. Yang, "Advanced obfuscation techniques for java bytecode," Journal of Systems and Software, vol.71, no.1–2, pp.1–10, 2004.

[13] M. Sosonkin, G. Naumovich, and N.D. Memon, "Obfuscation of design intent in object-oriented applications," Proc. 3rd ACM workshop on Digital rights management (DRM2003), pp.142–153, 2003.

[14] C. Wang, J. Hill, J. Knight, and J. Davidson, "Software tamper resistance: obfuscating staticanalysis of programs," Tech. rep. SC-2000-12, Dept. of Computer Science, University of Virginia, 2000.

[15] T. Ogiso, Y. Sakabe, M. Soushi, and A. Miyaji, "Software obfuscation on a theoretical basis and its implementation," IEICE Transactions on Fundamentals, vol.E86-A, no.1, pp.176–186, 2003.

[16] Y. Sakabe, M. Soshi, and A. Miyaji, "Java obfuscation with a theoretical basis for building secure mobile agents," Proc. 7th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS2003), Lecture Note in Computer Science vol.2828, pp.89–103, 2003.

[17] D.E. Bakken, R. Parameswaran, D.M. Blough, A.A. Franz, and T.J. Palmer, "Data obfusacation: anonymity and desensitization of usable data sets," IEEE Security and Privacy, vol.2, pp.34–41, 2004.

[18] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Tech. rep. 148, Dept. of Computer Science, University of Auckland, 1997.

[19] H. Sato, A. Monden, and K. Matsumoto, "Program obfuscation by coding data and its operation," Tech. Rep. of IEICE, vol.102, no.743, pp.13–18, 2003.

[20] K. Fukushima, S. Kiyomoto, T. Tanaka, and K. Sakurai, "Analysis of program obfuscation schemes with variable encoding technique," IEICE Trans. on Fundamentals, vol.E91-A, no.1, pp.316–329, 2008.

[21] S. Haber, and W.S. Stornetta, "How to time-stamp a digital document," Journal of Cryptology, vol.3, no.2, pp.99–111, 1991.

[22] RSA Laboratories, "PKCS#1 v2.1: RSA Cryptography Standard," ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf, 2002.

[23] M. Bellare, and P. Rogaway, "Optimal asymmetric encryption," Proc. of EUROCRYPT'94, Lecture Note Computer Science 950, pp.92–111, 1995.

[24] M. Brzozowski, and V.N. Yarmolik, "Obfuscation as a intellectual protection in vhdl language," Proc. of 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM'07), pp.337–340, 2007.

[25] R. Stern, P. Maciaszek, A.G. Michael Hsia, and R. Karri, "Digital logic obfuscation techniques to thwart cloning of asics," http://isis.poly.edu/csaw/winners/research/Richard Stern.pdf.

[26] SEMANTIC DESIGNS, INC., "VHDL Source Code Obfuscator," http://www.semdesigns.com/Products/Obfuscators/VHDLObfuscator.html.