Maciej BRZOZOWSKI, Vyacheslav N. YARMOLIK
POLITECHNIKA BIAŁOSTOCKA, WYDZIAŁ INFORMATYKI

Obfuskacja w zabezpieczaniu praw autorskich do projektów sprzętowych

mgr inż. Maciej BRZOZOWSKI

Ukończył studia na Wydziale Informatyki Politechniki Białostockiej w 2005. Od 2005 jest asystentem na wydziale Informatyki Politechniki Białostockiej. Specjalizuje się w zabezpieczaniu praw autorskich do projektów sprzętowych. Prowadzi badania nad wykorzystaniem znaków wodnych (*watermarking*) oraz obfuskacji w ochronie własności intelektualnej.

e-mail: brzozowski@ii.pb.bialystok.pl

prof. dr hab. inż. Vyacheslav N. YARMOLIK

Ukończył studia na Białoruskim Państwowym Uniwersytecie Informatyki i Radioelektroniki w Mińsku w 1973. W 1979 obronił pracę doktorską, a w 1990 uzyskał tytuł doktora habilitowanego. Od 34 lat pracuje w dziedzinie kryptografii oraz testowania pamięci.

e-mail: yarmolik@ii.pb.bialystok.pl

Streszczenie

Oprogramowanie jest coraz częściej rozpowszechniane w postaci kodu źródłowego. W niezabezpieczonym kodzie łatwo jest dokonać modyfikacji a następnie włączyć w „nowej" formie w inny projekt (produkt). Powstało wiele technik i narzędzi zabezpieczających przed kradzieżą oprogramowania takich jak znaki wodne lub odciski palca. Jedną z metod ochrony praw autorskich, zabezpieczającą przeciwko nieautoryzowanemu wykorzystaniu kodu, jest obfuskacja. Proces ten sprawia, że kod źródłowy staje się nieczytelny i trudny w analizie aczkolwiek nadal posiada wszystkie poprzednie właściwości funkcjonalne. W artykule przedstawimy techniki obfuskacji dla sprzętowego języka VHDL (Very High Speed Integrated Circuit Hardware Description Language) oraz przykłady ich zastosowania w ochronie praw autorskich.

Abstract

Software is more and more frequently distributed in form of source code. Unprotected code is easy to alter and build in others projects. One of the method for protection against attacks on intellectual rights is obfuscation, a process that makes software unintelligible but still functional. In this paper we review several generic techniques of obfuscation VHDL (Very High Speed Integrated Circuit Hardware Description Language) code and present a set of them for projects protection. We are going to describe some related experimental work.

Słowa kluczowe: VHDL, obfuskacja, ochrona własności intelektualnej.

Keywords: VHDL, obfuscation, intellectual property protection.

1. Obfuskacja

Zaciemnianie kodu (obfuskacja, z ang. *obfuscation*) jest techniką przekształcania kodu źródłowego oprogramowania, która zachowuje jego działanie semantyczne, ale znacząco utrudnia analizę i zrozumienie. Obfuskacja jest wiec zamierzonym działaniem, mającym na celu ochronę własności intelektualnej przy zachowaniu pełnej funkcjonalności oprogramowania. Ujmując rzecz bardziej ogólnie obfuskacja jest próbą ukrycia sposobu działania zabezpieczanego programu.

Jako pierwszy definicję obfuskacji podał Christian Collberg w „A taxonomy of obfuscating transformations" [1] używaną i rozwijaną w późniejszych publikacjach, których był współautorem np. [2].

$P \xrightarrow{\tau} P'$ jest transformatą obfuskacyjną jeżeli:

a) program P przerywa swoją pracę lub kończy ją z błędami P' może też przerywać swoje działanie lub skończyć je z błędami.
b) w przeciwnym przypadku P' musi zakończyć swoje działanie i wygenerować identyczne wyniki co P.

Gaël Hachez w zbiorczej pracy „A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards" [3] zauważa, że definicja przyjęta przez Collberga [1] daje możliwość wprowadzenia w błąd użytkownika oprogramowania. Na podstawie warunku (a) „...może też przerywać swoje działanie lub skończyć je z błędami" istnieje możliwość, że oryginalny program zostanie zakończony z błędami natomiast zmodyfikowany program (zabezpieczony) zakończy się nie generując (zgłaszając użytkownikowi) żadnych błędów. Tego typu niepożądane działanie zostało zmienione przez zmodyfikowanie definicji obfuskacji na:

$P \xrightarrow{\tau} P'$ jest transformatą obfuskacyjną jeżeli:

a) program P przerywa swoją pracę lub kończy ją z błędami program P' musi też przerywać swoje działanie lub skończyć je z błędami.
b) w przeciwnym przypadku P' musi zakończyć swoje działanie i wygenerować identyczne wyniki co P

Program P' może posiadać inne zachowanie w stosunku do programu P takie jak łączenie z internetem, wysyłanie poczty elektronicznej czy tworzenie plików jednak działania te nie będą miały żadnego wpływu na komunikację z użytkownikiem. Od zabezpieczonego programu P' nie jest wymagane by jego wydajność była równa z wydajnością programu P.

W tym miejscu należy zauważyć ograniczenia jakim podlega obfuskacja [4]. Należy pamiętać, że każdy program P' będzie można odtworzyć do P'', który swoją budową będzie bardzo zbliżony do programu P. Działanie tego typu pochłonie dużą ilość czasu oraz ogromny nakład środków, lecz zawsze będzie możliwe do wykonania. Dlatego zadaniem obfuskacji nie jest zabezpieczenie programu przed jego dekompilacją, lecz w znaczącym stopniu wydłużenie czasu potrzebnego na analizę i zrozumienie jego działania.

```
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);}
```

Listing. 1. Zwycięzca 1st International Obfuscated C Code Contest - w kategorii Dishonorable mention – 1984
Listing. 1. Winner of 1st International Obfuscated C Code Contest – in category Dishonorable mention – 1984

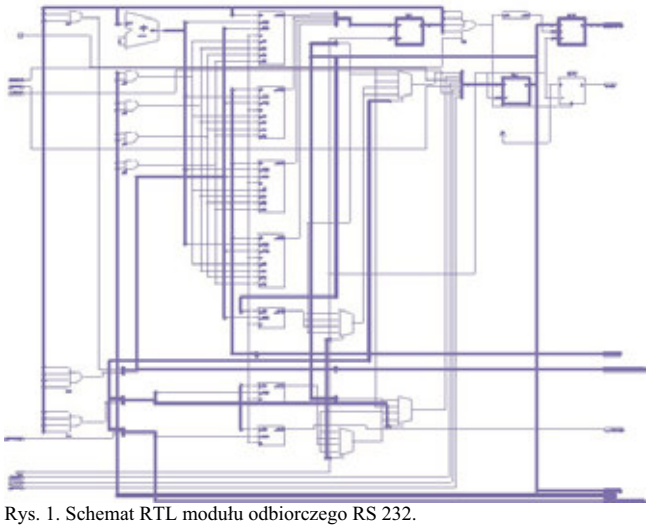
Na Listingu 1 przedstawiono kod źródłowy programu *anonimous.c* zwycięzcy 1st International Obfuscated C Code Contest 1984 w kategori Dishonorable mention (autor nieznany). „Zaciemniony" przykład pokazuje w jakim stopniu można utrudnić analizę krótkiego kodu. Dla osób które chciałyby podjąć się analizy kodu i sprawdzić jej poprawność zamieszczam pierwotny kod (Listing 2).

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

Listing. 2. Hello World!
Listing. 2. Hello World!

Obecnie znana jest szeroka gama technik obfuskacyjnych jednak ich zastosowanie nie zostało przeniesione na pole języków programowania sprzętowego.

W następnych paragrafach zaprezentujemy różne techniki obfuskacyjne oraz ich zastosowanie w odniesieniu do języka VHDL (Very High Speed Integrated Circuit Hardware Description Language) na przykładzie komponentu odbiornika RS232a.



Rys. 1. Schemat RTL modułu odbiorczego RS 232.
Fig. 1. RTL Scheme of RS 232 reciver unit (Register Transfer Level)

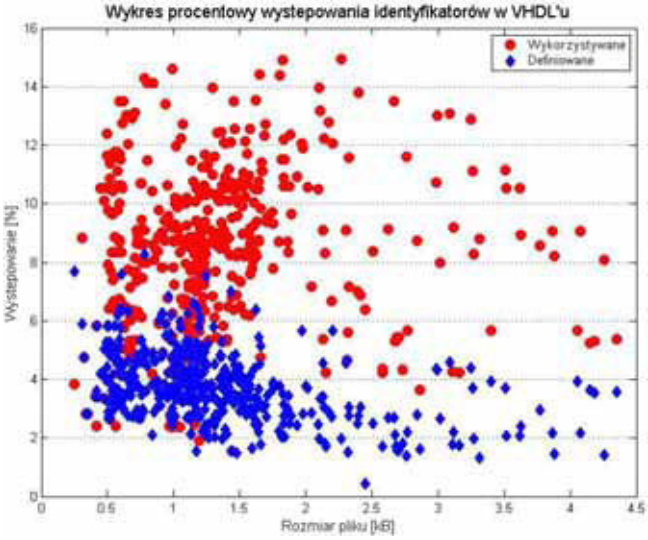
W analizie uwzględniliśmy takie parametry jak: liczbę zajętych komórek, liczbę komórek z przerzutnikami, wykorzystane cztero wejściowe komórki LUT (*lookup table*) oraz maksymalną częstotliwość taktowania układu. Projekt syntezowały był dla układu Xilinx Spartan2 xc2s100 przy jednakowych ustawieniach kompilatora.

Number of Slices:	16	out of	1200	1%
Number of Slice Flip Flops:	21	out of	2400	0%
Number of 4 input LUTs:	13	out of	2400	0%

Minimum period: 7.804ns (Maximum Frequency: 128.139MHz)

2. Zaciemnianie układu strony

Pierwszą grupą technik zaciemniania kodu jest zbiór transformat odpowiedzialnych za układ i przejrzystość kodu źródłowego. Zaciemnianie układu strony (*layout obfuscation*) polega na zmianie lub usunięciu informacji, które nie mają wpływu na wykonanie programu. Transformaty tego typu są najłatwiejsze do implementacji i zastosowania. Grupa często nazywana darmową (*free*) ponieważ jej zastosowanie w żadnym stopniu nie wpływa na zwiększenie objętości programu ani na prędkość działania po transformacji.



Rys. 2. Wykres deklaracji oraz wykorzystania identyfikatorów przez studentów w kodzie źródłowym języka VHDL
Fig. 2. Figure of declaracion and use identifiers in source code of VHDL by students

Rysunek 2 przedstawia występowanie identyfikatorów w kodzie źródłowym języka VHDL napisanym przez studentów. Należy pamiętać, że kod źródłowy napisany przez studentów różni się w znacznym stopniu od kodu napisanego przez profesjonalnych programistów. Występowanie identyfikatorów zostało podzielone na dwie grupy ze względu na ich deklaracje i wykorzystanie. Identyfikatory oraz komentarze upraszczają i skracają czas potrzebny na przeprowadzenie analizy kodu źródłowego. Komentarze można łatwo usunąć a nazwy identyfikatorów zmienić na inne – nie sugerujące wykorzystania w projekcie.

Transformaty zaciemniania układu strony noszą też nazwę - jednokierunkowe (*one-way*) - raz usunięta informacja lub zmieniona nazwa zmiennej nie jest możliwa do odtworzenia. Swoim działaniem zmniejsza ilość informacji ułatwiających pracę osobie analizującej. Zaciemnianie układu strony opiera się na:

- usuwaniu komentarzy opisujących działanie programu oraz informacji dla *debugger'a* - często programiści wprowadzają dużą ilość danych opisując kod (działanie funkcji i procedur, opis parametrów itp.).
- zamianie identyfikatorów (*Scrambling identifiers*) - wszystkie klasy, metody oraz pola są identyfikowane poprzez unikalną nazwę. Nazwy w dużym stopniu pomagają atakującemu poznać jaką rolę pełni w programie. Ogólnie przyjętą zasadą programowania jest nadawanie nazw elementom odpowiednich ich przeznaczeniu lub zastosowaniu.
- usuwaniu formatowania (*change formating*) - usuwanie różnego rodzaju wcięć - tabulacji, spacji, znaków białych, sklejanie wierszy - w kodzie źródłowym.

```
signal data_in : std_logic_vector(M-1 downto 0);
                          ⇓
signal O010100101:std_logic_vector(l01010101-1 downto 0);
```

Listing. 3. Zmiana identyfikatorów (VHDL)
Listing. 3. Scrambling identifiers in VHDL code

Na Listingu 3 zmianie uległy identyfikatory sygnałów data_in na O010100101 oraz M na l01010101. Symbol $\Rightarrow$ oznacza transformatę kodu źródłowego. Czas propagacji oraz obszar zajmowany przez projekt zawsze pozostają bez zmian przy wykorzystaniu transformat z grupy zaciemniania układu strony.

Transformaty z rodziny zaciemniania układu strony, ze względu na niewielkie potrzeby nakładu pracy i analizy, są najłatwiejszymi do zastosowania w ochronie własności intelektualnej dla języka VHDL i innych języków sprzętowych. Swoim działaniem nie wpływają na żadną z metryk analizowanych w artykule.

3. Zmiana kontroli

Celem tego typu obfuskacji (*control obfuscation*) jest zmiana kontroli w programie (modyfikacja grafu przepływu – *control flow*) nie zmieniając przy tym części wyliczeniowych programu. Wiele technik z tej grupy wywodzi się bezpośrednio z teorii optymalizacji wykorzystywanej w kompilatorach. W dalszej części przeanalizujemy tylko kilka technik z rodziny zmiany kontroli.

Pierwszą z przeanalizowanych technik jest zmiana kolejności wyrażeń - programiści wykazują tendencję do umieszczania razem powiązanego ze sobą kodu. Te lokalne relacje można zmienić używając kilku technik zamazujących.

```
□ <= □ + □;          □ <= □ + □;
□ <= □ + □;    ⇒     □ <= □ + □;
```

Listing 4. Zmiana kolejności wyrażeń w języku VHDL
Listing 4. Ordering change in VHDL code

Ze względu na budowę języka VHDL tego typu transformaty nie mają żadnego wpływu na przetwarzanie instrukcji. Nie ma znaczenia czy w kodzie projektu najpierw wystąpi $b+c$ czy $a+z$ (Listing 4) ponieważ obie te instrukcje wykonują się równolegle. Użytkownik może dowolnie zmieniać kolejność instrukcji (dotyczy to bloków przetwarzania równoległego) a działania te nie będą miały wpływu na funkcjonowanie projektu.

Kolejną techniką obfuskacyjną, z grupy zmiany kontroli, jest wyłączanie metod – wyłanianie części kodu i tworzenie na ich podstawie nowych funkcji, procedur lub komponentów.

```
y<=a+b;   ⇒   change: new_component port map (a, b, y);
                                or
              y <= function_add (a, b);
```

Listing. 5. Wyłączanie metod w języku VHDL
Listing. 5. Outline statements in VHDL code

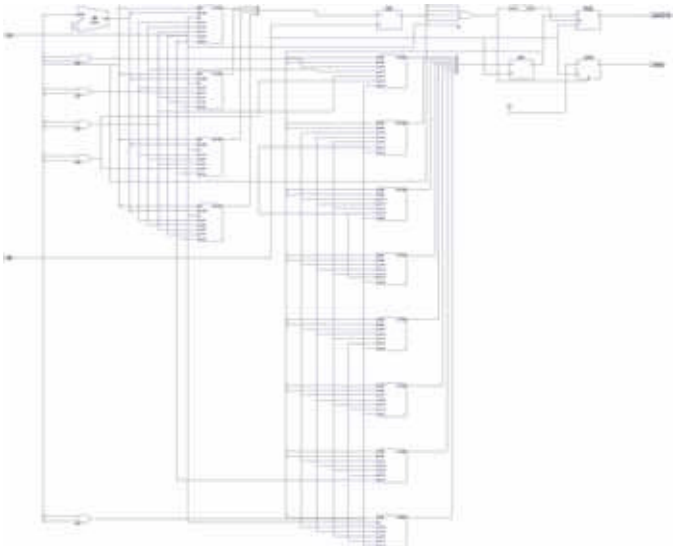
Number of Slices:	19 out of	1200	1%
Number of Slice Flip Flops:	21 out of	2400	0%
Number of 4 input LUTs:	29 out of	2400	1%

Minimum period: 7.804ns (Maximum Frequency: 128.139MHz)

Jak widać powyżej po zastosowaniu techniki wyłączania metod liczba zajmowanych komórek zwiększyła się jednak maksymalna prędkość taktowania dla danego projektu nie zmieniła się. Na Listingu 4 przedstawiono wydzielenie sumowania w komponent oraz w funkcję. Po technikach z grupy zaciemniania układu strony oraz zmiany kolejności wyrażeń jest to najprostsza technika do zastosowania w ochronie praw autorskich.

4. Wnioski

Techniki zaciemniania kodu można zastosować w odniesieniu do języków wysokiego poziomu takich jak C#, Java [5] jak i do języków niskiego poziomu takich jak assembler [6]. W większości wypadków działania zabezpieczające przed nieautoryzowana modyfikacją pociągają za sobą wzrost objętości kodu wykonywalnego lub zwiększenie czasu potrzebnego na jego poprawne wykonanie. Jednym z największych ograniczeń obfuskacji dla języków sprzętowych jest czas reakcji układu na zmieniające się sygnały wejściowe (Maximum combinational path delay, Maximum Frequency).



Rys. 3. Schemat RTL modułu odbiorczego RS 232 - zobfuskowany.
Fig. 3. RTL Scheme of RS 232 reciver unit - obfuscated

Number of Slices:	16 out of	1200	1%
Number of Slice Flip Flops:	21 out of	2400	0%
Number of 4 input LUTs	13 out of	2400	0%

Minimum period: 6.859ns (Maximum Frequency: 145.794MHz)

Na Rysunku 3 przedstawiono zabezpieczony projekt technikami z grupy zaciemniania układu strony oraz odpowiednio dobranymi technikami z grupy zmiany kontroli dobranymi pod względem wpływu na maksymalną częstotliwość taktowania układu. Jak można odczytać z wyników, transformaty nie wpłynęły na objętość projektu po zsyntezowaniu. Natomiast wydajność układu zwiększyła się o 13,7%. Zaprezentowane powyżej metody mogą zostać z powodzeniem zastosowane w ochronie własności intelektualnej do projektów sprzętowych.

5 Literatura

[1] C. Collberg, C. Thomborson, and D. Low. “A Taxonomy of Obfuscating Transformations”, Technical Report 148, Department of Computer Science. The University of Auckland, July 1997

[2] Christian S. Collberg, Clark D. Thomborson, and Douglas Low. Breakingabstractions and unstructuring data structures. In *International Conferenceon Computer Languages (ICCL)*, pages 28–38, 1998.

[3] Gaël Hachez. “A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards”, PhD thesis, Universite Catholique de Louvain, March 2003

[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan and K. Yang. “On the (Im)possibility of Obfuscating Programs” In J. Kilian, editor, Advances in Cryptology - CRYPTO '01, volume 2139 of Lectures Notes in Computer Science (LNCS), pages 1-18. Springer-Verlag, 2001.

[5] D. Low, “Java Control Flow Obfuscation. Master's thesis”, Department of Computer Science, University of Auckland, June 1998

[6] Gregory Wroblewski, “General Method of Program Code Obfuscation”, PhD Dissertation, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002

[7] Maciej Brzozowski, Vyacheslav N. Yarmolik "VHDL obfuscation techniques for protecting intellectual property rights on design", 5th IEEE East-West Design and Test Symposium, Yerevan, 2007, p.371-375

[8] Maciej Brzozowski, Vyacheslav N. Yarmolik "Obfuscation as Intellectual Rights Protection in VHDL Language", 6th Computer Information Systems and Industrial Management Applications, Los Alamitos, 2007, p.337-340

[9] Maciej Brzozowski, Vyacheslav N. Yarmolik "System wbudowujący znak wodny w kod źródłowy oprogramowania" Modelowanie i symulacja komputerowa w technice : IV Sympozjum, Łódź, Wyższa Szkoła Informatyki (Łódź, Polska),Łódź, 2005

[10] Maciej Brzozowski, Vyacheslav N. Yarmolik "Analiza statystycznych właściwości kodu źródłowego", XI Warsztaty Naukowe PTSK : Symulacja w badaniach i rozwoju : zbiór referatów, PTSK 2004

[11] Maciej Brzozowski, Vyacheslav N. Yarmolik "Metody i implementacja ochrony praw autorskich w oprogramowaniu komercyjnym", Wiadomości Elektrotechniczne, nr 12 (2004), s.28-29

[12] Thomas J. McCabe. A complexity measure. *IEEE Transaction on Software Engineering*, 2(4):308–320, 1976.

[13] C. Collberg and C. Thomborson, “Watermarking, TamperProofing, and Obfuscation --- Tools for Software Protection”, IEEE Transactions on Software Engineering, Vol. 28, No. 8, August 2002.

[14] C. Collberg and C. Thomborson, “On the limits of Software Watermarking”, Technical Report #164, Department of Computer Science, The University of Auckland, August 1998

[15] Jasvir Nagra and Clark Thomborson and Christian Collberg, “A Functional Taxonomy for Software Watermarking”, Twenty-Fifth Australasian Computer Science Conference (ACSC2002),ACS, Melbourne, Australia, 2002

[16] Johnson N. F., Duric Z., Jajodia S.: Information hiding – steganography and watermarking – attacks and countermeasures, Kluwer Academic Publishers, Norwell 2001

[17] Katzenbeisser S., Petitcolas F. A. P.,: Information hiding – techniques for steganography and digital watermarking, Artech House, Norwood 2000

[18] Tapas Sahoo and Christian Collberg. “Software watermarking in the frequency domain: Implementation, analysis, and attacks”, Technical Report TR04-07, Department of Computer Science, University of Arizona, March 2004

[19] Chenxi Wang, A Security Architecture for Survivability Mechanisms, PhD Dissertation, University of Virginia, Department of Computer Science, October 2000

[20] Christian Collberg, Clark Thomborson, Douglas Low, “Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs”, SIGPLAN-SIGACT POPL'98, ACM Press, San Diego, CA, January 1998

Artykuł powstał w ramach Pracy Statutowej S/WI/6/2008
Politechniki Białostockiej.

Title: Obfuscation as intelectual protection protection tool for digital design.

Artykuł recenzowany