материализованных представлений или прогнозов путем повторного воспроизведения событий в любое время и оказания помощи в тестировании и отладке системы. Список событий также может использоваться для анализа производительности приложений и выявления тенденций поведения пользователей или для получения другой полезной бизнес-информации.

Данный подход обычно используется тогда, когда необходимо минимизировать или полностью избежать конфликтов при обновлении данных, а также когда особую важность играет возможность воспроизводить происходящие в системе события, чтобы восстановить состояние системы, отменить изменения или сохранить журнал истории и аудита.

Очень часто EventSourcing используется вместе с шаблоном CQRS (Command-QueryResponsibilitySegregation– разделение ответственности на команды и запросы), что позволяет разделить процессы записи и чтения данных, увеличивая производительность и надежность системы [3].

Список использованных источников:

1. Martin Fowler, Development of Further Patterns of Enterprise Application Architecture

2. Event Sourcing Pattern [Электронный ресурс] / Christopher Bennag. – 23 июня 2017. – Режим доступа: https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing.

3. Betts, D. Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure. / D. Betts. – Microsoft, 2013. – 213 p.

ПРОГРАММНОЕ СРЕДСТВО ОБФУСКАЦИИ VHDL-КОДА

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Крагель Н.А.

Иванюк А.А.. – профессор, д.т.н., доцент

Одной из важных проблем, возникающих при проектировании цифровых устройств является защита интеллектуальной собственности. Существуют различные способы ее обеспечения и доказательства авторских прав: лексические и функциональные обфускации проектных описаний, внедрение водяных знаков и др.

Обфускация - приведение исходного текста проектных описаний к виду, сохраняющему функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию.

Цели обфускации:

Затруднение изучения и модификации проектных описаний;

Усложнение реверс инжиниринга проектных описаний.

Целью создания программы была автоматизация обфускации VHDL-кода с поддержкой основных конструкций языка.

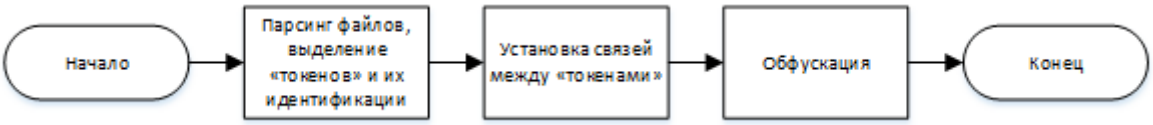

Рис. 1. Основные этапы работы программы.

Процесс работы программы представлен на рисунке 1. На первом этапе исходные файлы разбиваются на «токены». На втором этапе происходит инициализация «токенов» и установление связей между ними. Таким образом после этого этапа имеется полное программное представление загруженных исходных файлов. Рассмотрим подробнее шаг обфускации с внедрением водяного знака. Для обфускации генерируются новые имена для переменных, содержащие только знаки «1», «0», «O», «I» и «L». Можно выделить 2 группы элементов:

Элементы визуально похожие на «1»: «L», «I»;

Элементы визуально похожие на «0»: «O».

Распределение данных групп в рамках генерируемых названий нормальное. Однако можно управлять распределение элементов в рамках группы, увеличивая частоту использования одного символа и уменьшая частоту другого.

В рамках реализованной программы обфускации подлежат все переменные, сигналы, сущности и порты. Перед процессом обфускации были выделены все связи, что теперь позволяет пройтись по всем элементам, объявленным в загруженных файлах. Такой подход позволяет избежать проблем, возникающих, когда были загружены не все исходные файлы и некоторая часть структурных элементов была объявлена в отсутствующих файлах.

Ниже представлен пример работы программы.

Исходный код:	Обфусцированный код:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity D_trigger is
Port(
mD, mC, mR, mS : in std_logic;
mQ, mnQ : out std_logic);
end D_trigger;
architecture Behavioral of D_methoda is
begin
process(mC, mR, mS)
begin
if(mr = '0') then
if(ms = '0') then
mq <= 'X';
mnq <= 'X';
else
mq <= '0';
mnq <= '1';
end if;
elsif(ms = '0') then
mq <= '1';
mnq <= '0';
elsif rising_edge(mc) then
mq <= md;
mnq <= not(md);
end if;
end process;
end Behavioral;
	library ieee; use ieee.std_logic_1164.all; entity OOl0llOllOlO10ll0Ol01100lOOl01 is port(llOl01001l0lO10l1Ol0010l0l0llOl,llOl01001l0lOl0l1Ol0010l0l0llOl,llOl01001l0lO10l1Ol0010l0l0llOl,llOl01001l0lOl0l1Ol0010l0l0llOl:in std_logic; ll0l01001l0lOl0l1Ol0010l0l0llOl,ll0l01001l0lOl0l10l0010l0l0llOl:out std_logic); end OOl0llOllOlO10ll0Ol01100lOOl01; architecture behavioral of OOl0llOllOlO10ll0Ol01100lOOl01 is begin process(llOl01001l0lOl0l1Ol0010l0l0llOl,llOl01001l0lO10l1Ol0010l0l0llOl,llOl01001l0lOl0l1Ol0010l0l0llOl) begin if(llOl01001l0lO10l1Ol0010l0l0llOl='0')then if(llOl01001l0lOl0l1Ol0010l0l0llOl='0')then ll0l01001l0lOl0l1Ol0010l0l0llOl<='X'; ll0l01001l0lOl0l10l0010l0l0llOl<='X'; else ll0l01001l0lOl0l1Ol0010l0l0llOl<='0'; ll0l01001l0lOl0l10l0010l0l0llOl<='1'; end if; elsif(llOl01001l0lOl0l1Ol0010l0l0llOl='0')then ll0l01001l0lOl0l1Ol0010l0l0llOl<='1'; ll0l01001l0lOl0l10l0010l0l0llOl<='0'; elsif rising_edge(llOl01001l0lOl0l1Ol0010l0l0llOl)then ll0l01001l0lOl0l1Ol0010l0l0llOl<=llOl01001l0lO10l1Ol0010l0l0llOl; ll0l01001l0lOl0l10l0010l0l0llOl<=not(llOl01001l0lO10l1Ol0010l0l0llOl); end if; end process; end behavioral;

Рис. 2. Упрощенное представление архитектуры программы

На рисунке 2 представлена упрощенная архитектура программы. Ключевой особенностью данной архитектуры является модульность обфускаторов, каждый из которых производит свою, независимую от других обфускаторов, операцию. Например, один производит лексическую обфускацию сигналов, второй – лексическую обфускацию переменных и т.д. Основным недостатком лексической обфускации является то, что процесс синтеза нивелирует ее. Другими словами, схема, полученная синтезом описания с внедрением лексической обфускации будет идентична схеме, полученной синтезом описания, которое не было подвержено лексической обфускации. В свою очередь это требует добавления схемотехнической обфускации. Однако архитектура разработанного приложения позволяет добавлять любое количество новых независимых обфускаторов, тем самым позволяя решить это проблему.

Список использованных источников:

1. Иванюк, А.А. Проектирование встраиваемых цифровых устройств и систем: монография / А. А. Иванюк – Минск: Бестпринт, 2012. – 337 с.

2. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection: Obfuscation, Watermarking, and Tamperproofing for Software Protection. By Christian Collberg, Jasvir Nagra. Published Jul 24, 2009 by Addison-Wesley Professional.