УДК 519.681.3:004.41

БАЗОВЫЕ ПРИМИТИВЫ СХЕМНОЙ ОБФУСКАЦИИ ЦИФРОВЫХ УСТРОЙСТВ

В.В. СЕРГЕЙЧИК[1], А.А. ИВАНЮК[2]

Белорусский государственный университет информатики и радиоэлектроники
ул. П. Бровки, 6, г. Минск, 220013, Республика Беларусь
vovasq@mail.ru[1], ivaniuk@bsuir.by[2]

Рассматриваются особенности обфускации высокоуровневых описаний цифровых устройств. Даётся краткий обзор разновидностей обфускации, изучаются их достоинства и недостатки. Рассматриваются методы схемной обфускации. Приводятся базовые примитивы схемной обфускации, исследуется их применение.

Ключевые слова: обфускация VHDL, лексическая обфускация, схемная обфускация.

Введение

В настоящее время ущерб от несанкционированного производства и использования цифровых устройств составляет около 1 миллиарда долларов в день. Развиваются и другие виды угроз: атаки на аппаратные реализации криптографических алгоритмов (т.н. Side-Channel Attacks), вредоносные изменения схем (аппаратные трояны). В связи с этим решающее значение приобретает разработка методов защиты от подобных угроз. Обфускация – один из таких подходов. Она служит для запутывания понимания структуры и функционирования схемы с целью защиты от обратного проектирования [1]. Кроме того, обфускация применяется для сокрытия авторских водяных знаков и пользовательских отпечатков пальцев.

Схемная обфускация

В случае языка VHDL выделяют две разновидности методов обфускации: лексические и функциональные (схемные). Лексическая обфускация в случае языка VHDL затрагивает лишь уровень исходного описания. Отсюда происходит её главный недостаток: результат синтеза (схема) до и после обфускации остаётся неизменным. Именно поэтому простейшим способом атаки на лексические методы является логический (RTL) синтез. Суть схемной обфускации в получении более сложной для понимания схемы, имеющей эквивалентную функциональность:

$$V; V^* = O(V); DD(V) = S; DD(V^*) = S^*; S \neq S^*; Func(S) \equiv Func(S^*); \\ C(\{V^*, S^*\}) > C(\{V, S\}), \quad (1)$$

где V – исходное HDL-описание, V^* – обфусцированное описание, O – результат обфускации, DD – результат синтеза описания, S – схема, $Func$ – функциональность схемы, C – сложность.

Интересным способом схемной обфускации является внедрение генераторов констант (ГК). Они представляют собой разновидность непрозрачных предикатов, значения которых известны на этапе обфускации, но должны быть вычислены при анализе. Способ предполагает замену выводов «0» и «1» схемами (примитивами), генерирующими соответствующие логические значения постоянно:

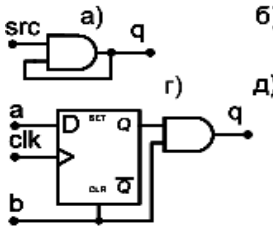
$$V_{\{0,1\}}; DD(V_{\{0,1\}}) = S_{0,1} \notin \{V_{DD}, GND\}; Func\{S_{0,1}\} \equiv Func\{V_{DD}, GND\}, \quad (2)$$

где $V_{\{0,1\}}$ – исходное описание примитива; V_{DD}, GND – сигнальные источники логической «1» и «0» соответственно.

Сложность непрозрачного предиката будет определяться сложностью анализа схемного примитива.

Важно добиться того, чтобы схема генератора не оказалась распознана и минимизирована средством синтеза. Различные схемы на основе только комбинационной логики (переключательных функций) не позволили достичь такого результата. Перспективными и эффективными с точки зрения числа используемых ресурсов выглядят последовательностные схемы и схемы, сочетающие последовательностную и комбинационную логику. В качестве дополнительных ухищрений может рассматриваться использование сигналов с хорошо определённым поведением (сигналы сброса, синхронизации и т.д.) на входах ГК.

Примеры ГК приведены на рис. 1. Они синтезированы с помощью XST I.24 Release 8.1i, синтезатор не смог распознать константы и минимизировать эти схемы. Схема *а* обладает двумя недостатками: синтезатор выдаёт предупреждение о комбинационном цикле, что даёт атакующему повод для анализа; если при включении на вход *src* подана «1», то до изменения значения входного сигнала на выходе *q* возможна «1», после изменения всегда будет «0». Устранить второй недостаток можно, подавая на вход сигнал системного сброса. Схема *г*, сочетающая последовательностную и комбинационную логику, лишена указанных недостатков, но требует больше аппаратных ресурсов.



б)
```
x <= x and src;
q <= x;
```

д)
```
process(a, clk, b)
begin
    if (b = '1') then
        val <= '0';
    elsif (rising_edge(clk))
    then val <= a; end if;
end process;
q <= val and b;
```

в)
```
oOooilool <=oOooilool
and ooOoilooi;ooOoilool
<= oOooilool ;
```

е)
```
process (oi1lOli1o,oi1lOli1o,oi1lOli1o
) begin if (oi1lOli1o= '1' )then oOOlOli1o
<='0'; elsif (rising_edge(oi1lOli1o)
) then oOOlOli1o <= oi1lOli1o;end if; end
process;ooilOli1o <= oOOlOli1o and oi1lOli1o;
```

Рис. 1. Генераторы констант: *а, г* – схемы примитивов; *б, д* – исходные VHDL-описания; в, е – результаты лексической обфускации

Достоинством схемной обфускации является усложнение для понимания не только высокоуровневого описания, но и результата синтеза. Приведённые примеры демонстрируют недостатки схемной обфускации: в некоторых случаях возможно увеличение аппаратных затрат и снижение скорости работы схемы.

Процесс внедрения примитивов осуществляется на уровне HDL-описаний.

Заключение

Использование схемной обфускации позволит скрыть константы – ценные источники информации для атакующего. Исследование зависимости входов примитивов от сигналов исходной схемы приведёт к дополнительному увеличению времени и сложности анализа обфусцированной схемы. Для увеличения защиты на всех уровнях абстракции имеет смысл использовать лексическую и функциональную обфускацию совместно.

Список литературы

1. *Chakraborty R.S.*, Hardware Security Through Design Obfuscation: Dis., Ph.D. Clevelend, 2010.