

# КОМП'ЮТЕРНІ ЗАСОБИ, МЕРЕЖІ ТА СИСТЕМИ

*В статті представлена автоматизована система, упрощаюча процес верифікації. Розглянуто процес перетворення VHDL програми в BDD структури.*

© Д.А. Закутайло, 2003

УДК 519.713.1

Д.А. ЗАКУТАЙЛО

## ТРАНСЛЯЦІЯ VHDL ПРОГРАММИ В BDD СТРУКТУРИ

В наші часи в Інституті кібернетики ім. В.М.Глушкова НАН України розробляється автоматизована система верифікації цифрових систем, що дозволяє тестувати істинність тимчасових тверджень. Як вхідна інформація використовується опис пристрою на VHDL. Особливості системи включаються в наступне:

- 1) розробляється система верифікації, орієнтована на взаємодію з існуючими САПР для проектування цифрових систем на ПЛИС;
- 2) запропоновано варіант тимчасової логіки, що спрощує запис тимчасових властивостей для розробників цифрових систем;
- 3) реалізовано можливість верифікації схем з затримками;
- 4) можлива генерація тестових послідовностей, що задовольняють вимогам користувача;
- 5) за бажанням користувача здійснюється перевірка досяжності стану в системі.

Розроблена система для верифікації тимчасових формул дозволяє тестувати істинність тимчасових тверджень, використовуючи як вхідну інформацію опис пристрою на VHDL. Цей мовою опису було обрано завдяки широкому використанню її серед мов опису в САПР на ПЛИС. Вибір такої вхідної мови забезпечує широке застосування розробленої системи. Вхідною інформацією системи може бути як поведінкове описання, так і структурне описання

на VHDL. В качестве языка задания временных свойств был выбран вариант CTL логики, обеспечивающий компромисс между выразительными возможностями и сложностью верификации. Отличия собственного варианта временной логики от CTL заключаются в следующем:

- можно ссылаться на предыдущее значение сигнала;
- введен ряд операторов для манипуляции с сигналами.

Очевидно, разработчик должен хорошо разбираться в CTL логике, чтобы задать требуемые свойства. Система, получив входные данные, проверяет истинность или ложность сформулированных утверждений на всех достижимых состояниях схемы, описанной на VHDL. При невыполнимости требуемого свойства генерируется контрпример. В случае выполнения свойства система (по желанию пользователя) генерирует пример, подтверждающий истинность утверждения. Упрощенно алгоритм работы системы изображен на рис. 1.

Разработанная система позволяет устанавливать эквивалентность двух описаний устройств на VHDL. Такая возможность может быть полезна в следующих случаях:

- 1) даны два описания на VHDL, требуется установить их семантическую идентичность;
- 2) с помощью САПР получено структурное описание устройства на VHDL, пользователь вносит изменения и необходимо узнать, не повлияют ли они на функционирование схемы;
- 3) поведенческое описание на VHDL преобразуется в структурное описание посредством синтезатора САПР. В этом процессе очень важно гарантировать эквивалентность двух описаний. Если обнаруживается несоответствие, то это свидетельствует о серьезных ошибках в работе синтезатора САПР. Таким образом, можно провести тестирование синтезатора САПР.

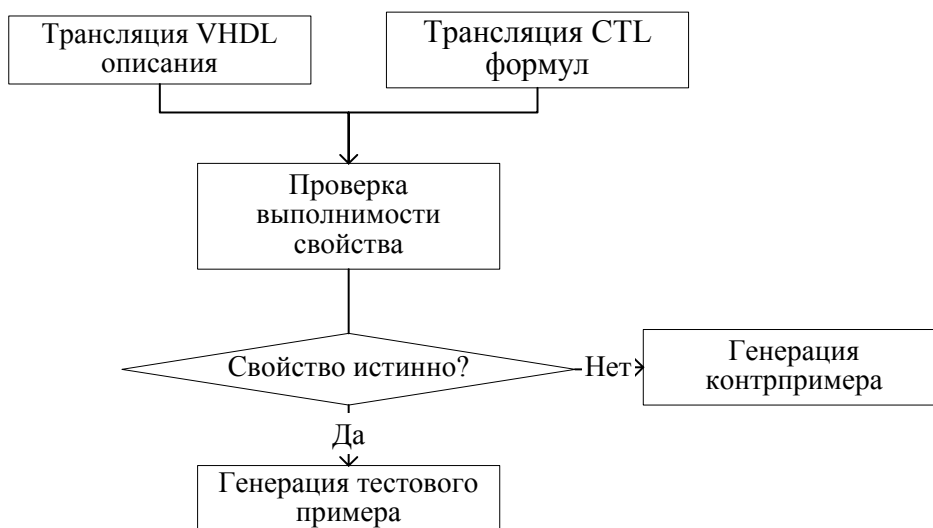


РИС. 1. Алгоритм работы системы

Для автоматизированных систем верификации важным является внутренний способ представления различных описаний в виде структур данных, а также методы для манипулирования такими структурами. В разрабатываемой системе в качестве внутренних структур данных, представляющих описание на VHDL, используются структуры данных типа BDD [1].

BDD расшифровывается как «binary decision diagram» (диаграмма двоичного решения). BDD представляет собой структуру данных для описания булевых функций. Современное представление BDD предложено Bryant [2], хотя общие идеи появились гораздо раньше, в виде так называемых «ветвящихся программ», в отечественной литературе – так называемые релейные деревья.

BDD обладает рядом полезных свойств:

1) многие функции имеют компактное представление в виде BDD. На рис.2 изображена BDD для функции, сумма по модулю 2. Функции такого типа требуют для своего представления  $2n-1$  узлов, где  $n$  – число переменных, тогда как стандартное представление занимает экспоненциальное количество узлов [2];

2) BDD удобно манипулировать. Эффективные алгоритмы существуют как для простейших булевских операций (AND, OR, NOT и т.д.), так и для более сложных;

3) если зафиксировать порядок, в котором появляются переменные, то BDD является каноническим представлением булевой функции, т.е. любая функция  $f: B^n \rightarrow B$  имеет в точности только одно представление в виде ROBDD. Таким образом, сравнение булевских функций сводится к простому сравнению указателей.

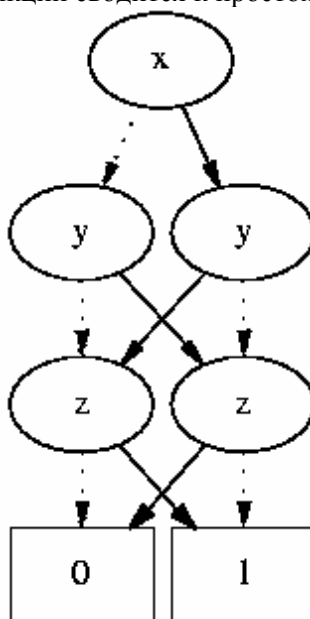


РИС. 2. ROBDD для функции  $x \oplus y \oplus z$

Рассмотрим преобразование VHDL описания в BDD структуры. На рис. 3 представлено описание трехбитного сбрасываемого счетчика. Если на входе счетчика сигнал `reset='0'`, то счетчик сбрасывается в ноль и устанавливает сигнал `ready='1'`. После этого ожидается подтверждение от внешнего устройства посредством сигнала `ack`, который должен изменить свое состояние с 0 в 1. Если `ack='1'`, то сигнал готовности сбрасывается и ожидается сброс подтверждения от внешнего устройства. Только после прихода заднего фронта этого сигнала счетчик устанавливает сигналы `b0 <= '1'`, `b1 <= '0'`, `b2 <= '0'`. Дальнейший счет будет осуществляться по переднему фронту сигнала `clk`. На рис. 4 представлена граф-схема (ГС) счетчика.

```
entity cnt is
  port ( start, reset, ack, clk: in BIT;
        ready: out BIT;
        b0, b1, b2 : inout BIT := '0');
end entity cnt;
architecture cnt_arch of cnt is
begin
  process
  begin
    if (reset='0') then
      b0 <= '0';
      b1 <= '0';
      b2 <= '0';
      ready='1';
      wait until ack='1'; --1
      ready='0';
      wait until ack='0'; --2
    else
      b0 <= not b0;
      b1 <= b1 xor b0;
      b2 <= b2 xor (b1 and b0);
      wait until clk='1'; --3
    end if;
  end process;
end architecture cnt_arch;
```

РИС. 3. Трехбитный сбрасываемый счетчик

На первом этапе необходимо найти участки исходного текста, где выполнение программы может приостановиться. Такими участками служат конструкции типа **wait on signal\_list until** cond. Если условие cond не выполняется или в списке сигналов signal\_list не произошло ни одного события, то выполнение программы приостанавливается.

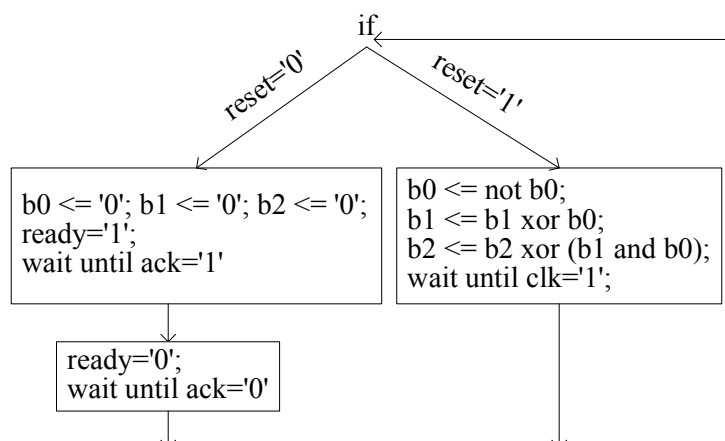


РИС. 4. ГС счетчика

В вышеприведенном примере существуют три **wait** конструкции. Каждой из этих конструкций необходимо присвоить некоторое уникальное числовое значение. Пометим их числами 1, 2 и 3 (указаны в комментариях). Введем понятия счетчика состояний исходной программы. Такой счетчик будет изменять свое значение при переходе из одного **wait** выражения в другое. Обозначим состояние этого счетчика переменной state. Для учета начального состояния программы зарезервировано значение state равное 0. Из **wait** выражения можно выделить условие, при котором программа возобновит свое выполнение. Например, выполнение будет продолжено для первого **wait** выражения, если выполнится ready='1' and ready'event = '1'. С учетом введения счетчика состояний условие, при котором программа возобновит свое выполнение, запишется следующим образом:

```

resumption = r0 or r1 or r2 or r3;
r0 = (state=0);
r1 = (state=1 and ack='1' and ack'event = '1');
r2 = (state=2 and ack='0' and ack'event = '1');
r3 = (state=3 and clk = '1' and clk'event = '1').
    
```

В более общем виде запишем таким образом:

$$(state = 0) \vee \bigvee_{i=2}^n (state = i) \wedge cond_i \wedge \left( \bigvee_{s \in sig_i} s'event \right)$$

Здесь индекс  $i$  служит для указания номера **wait** конструкции;  $n$  – количество таких конструкций;  $cond_i$  – условие  $i$  **wait** конструкции;  $sig_i$  – список сигналов из, которых образовано  $i$ -е условие и список сигналов перечисленных явно в списке чувствительности.

ГС программы преобразуется следующим образом: перед каждым **wait** выражением ставится конструкция вида state =  $i$ , где  $i$  – уникальный номер; **wait**

выражения удаляются; первым выражением программы становится условный оператор выбора участка программы, с которого начнется выполнение; последним выражением становится оператор **wait**, учитывающий все условия, при которых программа возобновит работу. Преобразованная ГС программы изображена на рис. 5.

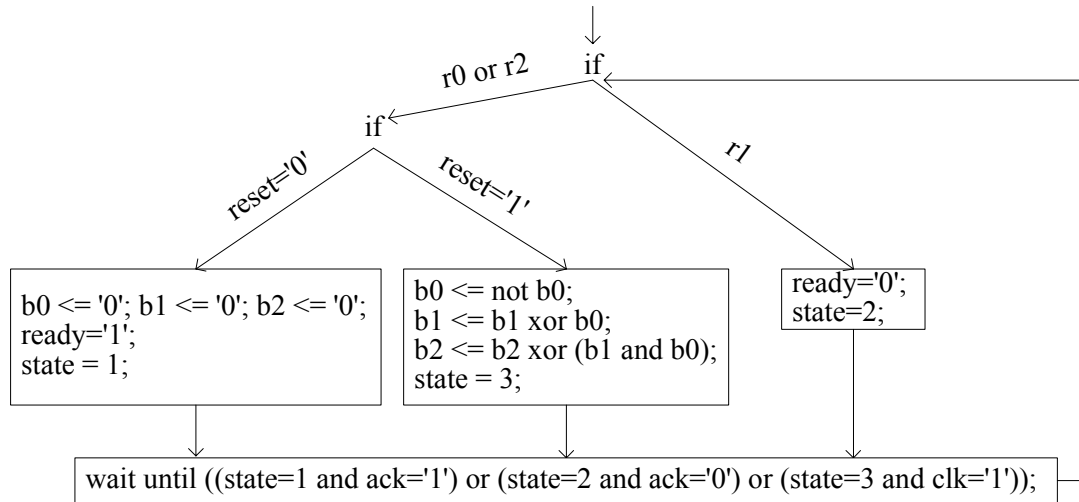


РИС. 5. Преобразованная ГС счетчика

Дальнейшие преобразования ГС связаны с применением нижеследующих трансформаций:

1) конструкция вида

```
seq_statM
if cond1 then seq_stat1
      else seq_statN
end if;
```

преобразуется в

```
if cond1 then seq_statM
      seq_stat1
      else seq_statM
      seq_statN
end if;
```

конструкция вида

```
if cond1 then seq_stat1
      else seq_statN
end if;
seq_statM
```

преобразуется в

```
if cond1 then seq_stat1
      seq_statM
      else seq_statN
      seq_statM
end if;
```

2) если на некотором участке осуществляется присвоение переменной, и эта переменная используется в последующих выражениях, то заменить все вхождения переменной ее значением.

Пример: конструкция вида

$v:=12; d\leq k; s\leq v+1;$  преобразуется в  $d\leq k; s\leq 13;$

1) и 2) преобразования осуществляются для того, чтобы переместить операторы if как можно ближе к вершине ГС и избавиться от тех переменных, значение которых можно легко вычислить. В примере со счетчиком эти преобразования применять нет необходимости, так как ГС уже находится в нужном состоянии.

На втором этапе необходимо из ГС выделить ГС для всех переменных и сигналов, оставшихся после преобразований. На рис. 6 изображены ГС для сигнала b0 и переменной state. Каждая из этих структур представляет собой дерево, узлами которого являются условные вершины, а листьями - выражения присвоения. Такие деревья легко преобразуются в BDD структуры.

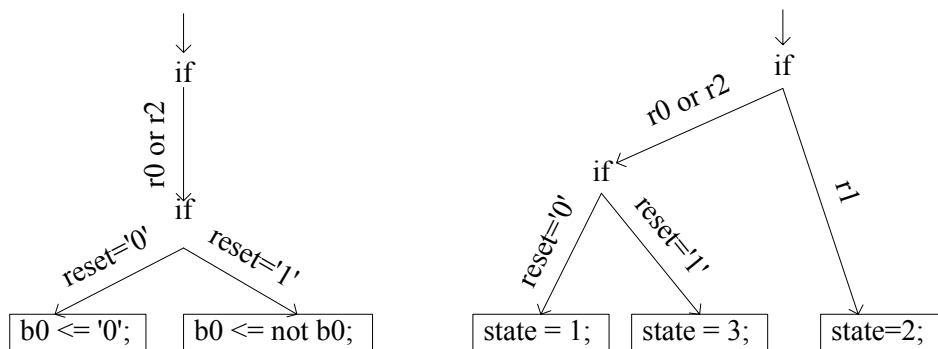


РИС. 6. ГС для сигнала b0 и переменной state

Автоматизированная система для верификации цифровых устройств, использующая BDD структуры, находится в завершающей стадии разработки. С ее помощью будет верифицирована многокластерная интеллектуальная система [3], разрабатываемая в Институте кибернетики им. В.М.Глушкова НАН Украины.

1. *Закутайло Д.А.* Использование структур данных типа BDD для формальной верификации аппаратуры //Нові комп'ютерні засоби, обчислювальні машини та мережі. – К.: Ін-т кібернетики ім. В.М. Глушкова НАН України, 2001. – Т. 1. – С.140–152.
2. *Randal E.Bryant.* Graph-based algorithms for Boolean function manipulation // IEEE Transactions on Computers. – 1986. – 8(C-35). – P. 677-691.
3. *Koval V., Bulavenko O., Rabinovich Z.* Parallel Architectures and Their Development on the Basis of Intelligent Solving Machines // Intern. Conf. On Parallel Computing in Electrical Eng. – Warsaw, Poland, 22-25 September 2002. – P. 21-26.

Получено 15. 06. 2003