УДК 681.326:519.613

В.И. ХАХАНОВ, Е.И. ЛИТВИНОВА, И.А. ПОБЕЖЕНКО, TIECOURA YVES

Харьковский национальный университет радиоэлектроники, Украина

ПРИМЕР ОПРЕДЕЛЕНИЯ ТЕСТОПРИГОДНОСТИ ЦИФРОВОГО ПРОЕКТА

Предлагается алгебрологическая модель для вычисления критериев тестопригодности системных HDL-моделей, ориентированная на существенное повышение качества проектируемых компонентов цифровых систем на кристаллах (yield) и уменьшение времени разработки (time-to-market). Разработанные критерии управляемости и наблюдаемости применены для оценки качества графа управления в целях его улучшения и эффективного диагностирования семантических ошибок. Практическая значимость предложенных методик и моделей заключается в рыночной привлекательности и высокой заинтересованности технологических компаний в инновационных решениях проблемы эффективного тестирования и верификации программно-аппаратных изделий.

Ключевые слова: *тестирование, тестопригодность, верификация, ассерция, HDL-модель.*

Используется среда моделирования, тестопригодного анализа логической структуры HDL-программы для квазиоптимального размещения механизма ассерций [1, 2], применяемые в hardware design and test. Разработанные критерии управляемости и наблюдаемости [3, 4] применены для оценки качества графа управления в целях его улучшения и эффективного диагностирования семантических ошибок. Представлены примеры вычисления функций и критериев тестопригодности, а также поиска ошибок в программном HDL-коде реального цифрового изделия [3, 4].

1. Анализ тестопригодности графа управления

Учитывая, что автоматная модель программного продукта представлена взаимодействием операционного и управляющего автомата [4], рис. 1, то наряду с моделированием транзакционного графа, необходимо иметь возможность анализировать тестопригодность граф-схемы алгоритма управления (ГСА).

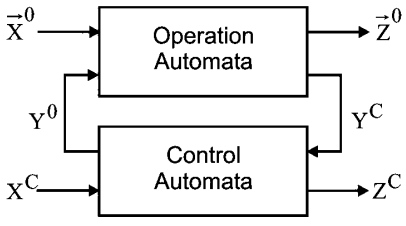


Рис. 1. Автоматная модель HDL-программы

Предлагается ГСА представить в виде содержательного графа управления (СГУ), который является подобным транзакционному графу. Здесь вершины есть операции программного кода, а дуги представляют условия перехода из одной вершины в другую для выполнения команды, обозначенной вершиной-стоком. Следовательно, для СГУ можно использовать процедуры, ранее разработанные для подсчета критериев тестопригодности транзакционного графа в части управляемости и наблюдаемости. Примером содержательного графа может служить рис. 2, имеющий 6 вершин и 9 дуг.

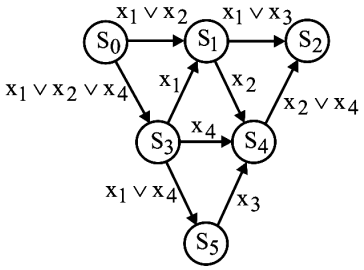


Рис. 2. Содержательный граф HDL-программы

Подсчет управляемостей графа [3, формула 1, 4, с.239], представленного на рис. 2, имеет следующий вид:

$$S_1 = T_3^3 T_4^1 \vee T_1^2;\ S_3 = T_3^3;$$

$$S_4 = T_3^3 T_4^1 T_6^1 \vee T_3^3 T_5^1 \vee T_3^3 T_8^2 T_9^1;$$

$$S_2 = T_3^3 T_4^1 T_6^1 T_7^2 \vee T_1^2 T_2^2 \vee$$

$$\vee T_3^3 T_5^1 T_7^2 \vee T_3^3 T_4^1 T_7^2 \vee T_3^3 T_8^2 T_9^1 T_7^2;$$

$$S_5 = T_3^3 T_8^2.$$

Подсчет наблюдаемостей графа [3, формула 1, 4, с.239], представленного на рис. 2, содержит следующие выражения:

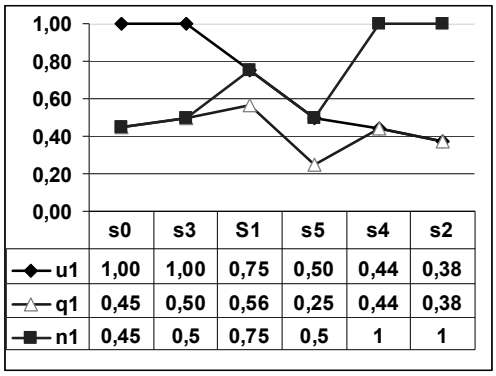
$$S_1 = T_2^2 \vee T_7^2 T_6^1;$$

$$S_3 = T_7^2 T_5^1 \vee T_7^2 T_9^1 T_8^2 \vee T_7^2 T_6^1 T_4^1 \vee T_2^2 T_4^1;$$

$$S_0 = T_2^2 T_1^2 \vee T_7^2 T_6^1 T_4^1 T_3^3 \vee T_7^2 T_5^1 T_3^3 \vee$$
$$\vee T_7^2 T_5^1 T_3^3 \vee T_7^2 T_9^1 T_8^2 T_3^3 \vee T_2^2 T_4^1 T_3^3;$$
$$S_4 = T_7^2;\ S_5 = T_7^2 T_9^1.$$

Для использования тестопригодности выполняется построение управляемости и наблюдаемости всех компонентов HDL-модели (рис. 3). Затем вычисляется обобщенная характеристика – тестопригодность каждого компонента как произведение управляемости и наблюдаемости:

$$Q_i = U_i \times N_i. \qquad (1)$$



	s0	s3	S1	s5	s4	s2
u1	1,00	1,00	0,75	0,50	0,44	0,38
q1	0,45	0,50	0,56	0,25	0,44	0,38
n1	0,45	0,5	0,75	0,5	1	1

Рис. 3. Графики тестопригодности для графа управления

Далее интерес представляет создание таблицы тестопригодности, управляемости и наблюдаемости [2, 4], а также соответствующий им график для визуального контроля «плохих» компонентов. Фиксация определенной планки тестопригодности, ниже которой значения будут считаться неприемлемыми, позволит разработчику создавать ассерции и другие дополнительные средства повышения тестопригодности для проблемных функциональных блоков. Кроме того, средства повышения тестопригодности должны обеспечивать глубину диагностирования до функционального компонента и привязанных к нему операций в целях быстрого восстановления работоспособности программной HDL-модели.

В целях построения алгоритмов поиска ошибок в программном коде можно использовать таблицу неисправностей, по аналогии с технологией тестирования hardware. Любопытное решение в процессе проверки функциональных блоков связано с сигнатурным анализом, где обобщенная сигнатура отождествляется с исправным поведением всего кода, а также с каждым компонентом. Любое несовпадение эталонной сигнатуры с фактической приводит к выполнению процедуры диагностирования и восстановления работоспособности HDL-модели путем исправления семантики кода.

Предложенная модель верификации HDL-проекта использует testbench, функциональное покрытие, механизм ассерций, описанную выше метрику оценки тестопригодности, таблицу неисправностей и вектор экспериментальной проверки (ВЭП), формируемый по заданным контрольным точкам путем сравнения сигнатур. Функциональное ограничение testbench связано с неразличимостью компонентов программного кода, в которых могут быть ошибки. Его основное назначение – проверка исправности HDL-модели. Поэтому в качестве дополнения к процедуре проверки придается механизм ассерций [4-6], основная цель которого с заданной глубиной – до программного компонента – определить место и вид ошибки на стадии выполнения диагностирования, после того, как testbench зафиксировал неправильное функционирование программного проекта. Две стадии верификации [4,6]: тестирование и диагностирование представлены ниже в виде двух векторно-матричных операций:

$$\begin{pmatrix} T_1^B \\ T_2^B \\ T_i^B \\ T_n^B \end{pmatrix} \oplus \begin{pmatrix} S_1^1 & S_1^2 & S_1^t & S_1^m \\ S_2^1 & S_2^2 & S_2^t & S_2^m \\ S_1^1 & S_i^2 & S_i^t & S_i^m \\ S_n^1 & S_n^2 & S_n^t & S_n^m \end{pmatrix} = \begin{pmatrix} L_1^B \\ L_2^B \\ L_i^B \\ L_n^B \end{pmatrix}$$

$$\cup$$

$$\begin{pmatrix} A_1^1 & A_1^2 & A_1^t & A_1^m \\ A_2^1 & A_2^2 & A_2^t & A_2^m \\ A_1^1 & A_i^2 & A_i^t & A_i^m \\ A_n^1 & A_n^2 & A_n^t & A_n^m \end{pmatrix} \oplus \begin{pmatrix} S_1^1 & S_1^2 & S_1^t & S_1^m \\ S_2^1 & S_2^2 & S_2^t & S_2^m \\ S_1^1 & S_i^2 & S_i^t & S_i^m \\ S_n^1 & S_n^2 & S_n^t & S_n^m \end{pmatrix} = \begin{pmatrix} L_1^1 & L_1^2 & L_1^t & L_1^m \\ L_2^1 & L_2^2 & L_2^t & L_2^m \\ L_1^1 & L_i^2 & L_i^t & L_i^m \\ L_n^1 & L_n^2 & L_n^t & L_n^m \end{pmatrix}.$$

Для первой стадии используется двоичный вектор экспериментальной проверки Z^B, формируемый на основе процедуры тестирования. На второй стадии используется уже матрица Z^A экспериментальной проверки, которая с наперед заданной глубиной определяет диагноз проекта на основе срав-

нения технических состояний HDL-модели и механизма ассерций:

$$Z^B = (Z_1, Z_2, .., Z_i, ..., Z_n);$$

$$Z^A = \begin{array}{|c|c|c|c|} \hline Z_1^1 & Z_1^2 & Z_1^t & Z_1^m \\ \hline Z_2^1 & Z_2^2 & Z_2^t & Z_2^m \\ \hline Z_1^1 & Z_i^2 & Z_i^t & Z_i^m \\ \hline Z_n^1 & Z_n^2 & Z_n^t & Z_n^m \\ \hline \end{array}$$

В процессе выполнения процедуры верификации выполняется сравнение фактического и эталонного (специфицированного) технического состояния компонента путем применения операции Xor: $\{Z_i, Z_i^t\} = S_i^t \oplus S_i^t(r) = \{0,1\}$.

Практически, если выполнены условия тестопригодности и правильно расставлены ассерции в критических точках программного кода для диагностирования всех компонентов, то ВЭП может однозначно идентифицировать адрес (место) и тип ошибки на основе построенной ранее таблицы неисправностей – механизма ассерций.

2. Верификация дискретного косинусного преобразования IP-core Xilinx

Представленные модели верификации программного HDL-кода проверены на реальном проекте DCT (Discrete Cosine Transform – дискретное косинусное преобразование) IP-core Xilinx для определения наличия в нем ошибок. При этом удалось определить неверную семантику работы программы для последующего исправления кода. Фрагмент модуля дискретного косинусного преобразования представлен листингом 1 [Open Source Xilinx.com]. Вся HDL-модель насчитывает 900 строк кода System Verilog.

Листинг 1

```
module Xilinx
`timescale 1ns/10ps
module dct ( CLK, RST, xin,dct_2d,rdy_out);
output [11:0] dct_2d;
input CLK, RST;
input[7:0] xin; /* input */
output rdy_out;
wire[11:0] dct_2d;
……
/* The first 1D-DCT output becomes valid after 14 +64 clk cycles. For the first 2D-DCT output to be valid it takes 78 + 1clk to write into the ram + 1clk to write out of the ram + 8 clks to shift in the 1D-DCT values + 1clk to register the 1D-DCT values + 1clk to add/sub + 1clk to take compliment + 1 clk for multiplying + 2clks to add product. So the 2D-DCT output will be valid at the 94th clk. rdy_out goes high at 93rd clk so that the first data is valid for the next block*/
Endmodule
```

В соответствии с правилами тестопригодного анализа, приведенными выше, спроектирован транзакционный граф как развитие графа регистровых передач [7,8], представленный на рис. 4, который для module Xilinx имеет 28 вершин-компонентов (входная и выходная шины, логические и регистровые переменные, векторы и память).

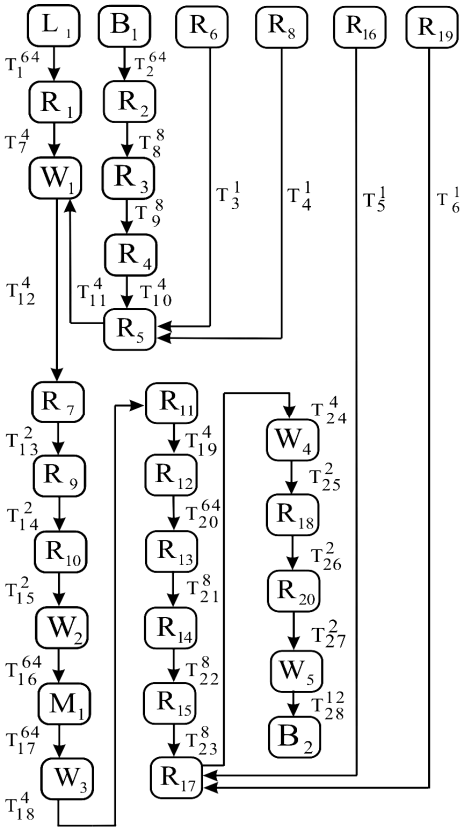


Рис. 4. Транзакционный граф Xilinx модели

Идентификатор дуги имеет верхний индекс, обозначающий число транзакций в программе между исходящей и входящей вершинами. Для каждой вершины строятся логические функции управляемости и наблюдаемости. Пример логической функции управляемости для вершины B_2 имеет следующий вид:

$$B_2 = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}(T_5^1 \vee T_6^1 \vee$$
$$\vee T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}(T_1^{64}T_7^4 \vee$$
$$\vee T_{11}^{4}T_2^{64}T_8^8T_9^8T_{10}^4 \vee T_{11}^{4}T_3^1 \vee T_{11}^{4}T_4^1)) =$$
$$= T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_5^1 \vee T_6^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4} \vee$$
$$\vee T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4} \wedge$$
$$\wedge T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}T_1^{64}T_7^4 \vee$$
$$\vee T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4} \wedge$$
$$\wedge T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}T_{11}^{4}T_2^{64}T_8^8T_9^8T_{10}^4 \vee$$
$$\vee T_{11}^{4}T_3^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64} \wedge$$
$$\wedge T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8} \vee$$
$$\vee T_{11}^{4}T_4^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64} \wedge$$
$$\wedge T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}.$$

Для остальных вершин аналогично выполняется вычисление ДНФ функций управляемостей.

Примеры вычисления функций наблюдаемостей для отдельных вершин имеют следующий вид:

$$R_2 = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{23}^{8}T_{22}^{8}T_{21}^{8}T_{20}^{64}T_{19}^{4}T_{18}^{4} \wedge$$
$$\wedge T_{17}^{64}T_{16}^{64}T_{15}^{2}T_{14}^{2}T_{13}^{2}T_{12}^{4}T_{11}^{4}T_{10}^{4}T_{9}^{8}T_{8}^{8};$$
$$B_1 = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{23}^{8}T_{22}^{8}T_{21}^{8}T_{20}^{64}T_{19}^{4}T_{18}^{4} \wedge$$
$$\wedge T_{17}^{64}T_{16}^{64}T_{15}^{2}T_{14}^{2}T_{13}^{2}T_{12}^{4}T_{11}^{4}T_{10}^{4}T_{9}^{8}T_{8}^{8}T_{2}^{64}.$$
$$L_1 = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{23}^{8}T_{22}^{8}T_{21}^{8}T_{20}^{64} \wedge$$
$$\wedge T_{19}^{4}T_{18}^{4}T_{17}^{64}T_{16}^{64}T_{15}^{2}T_{14}^{2}T_{7}^{4}T_{1}^{64}.$$

Синтезированные логические функции задают все возможные пути управления, как во времени, так и в пространстве, что можно считать новой аналитической формой описания тестопригодности проекта. По ДНФ, следуя выражениям для подсчета тестопригодности [1], можно определить критерии управляемости (наблюдаемости) для всех компонентов HDL-модели. Здесь следует рассмотреть для варианта (сценария) обсчета программной модели. 1) Учитывается только графовая структура, где вес каждой дуги равен 1, независимо от числа транзакций в программном коде. 2). Все дуги графа отмечаются реальным количеством транзакций, имеющих место быть между двумя рассматриваемыми вершинами-компонентами транзакционного графа. Оценки тестопригодности описанных процедур могут существенно различаться друг от друга. Пользователь должен определиться, что важнее только структура программного кода – применить первый сценарий, или иметь более сложную и точную модель транзакций, распределенных во времени, на множестве графовых компонентов. В качестве примера ниже приводится процедура вычисления управляемости для вершины B_2 :

$$U(B_2) = \frac{1}{22 \times 6} \times (6 + 6 + 19 + 22 + 19 + 19) = 0{,}54 .$$

Применение аналогичных вычислений управляемостей (наблюдаемостей) для других вершин графа дает результат в виде графика, представленного на рис. 5, которые позволяют определить критические точки для установки необходимых ассерций.

Такой вершиной может быть компонент R_{15}, если транзакционный граф представлен одиночными дугами. Для случая, когда дуги отмечены реальным количеством транзакций, критические вершины принадлежат компонентам, находящихся ближе к выходной шине B_2 . Здесь существенным представляется не структура графа, а вес дуги входящей, который в большей степени оказывает негативное влияние, если структурная глубина рассматриваемого компонента достаточно высока. Используется формула (1) вычисления тестопригодности с мультипликативными членами $U_i \times N_i$, что дает оценку ниже, чем любой из сомножителей (управляемость, наблюдаемость).

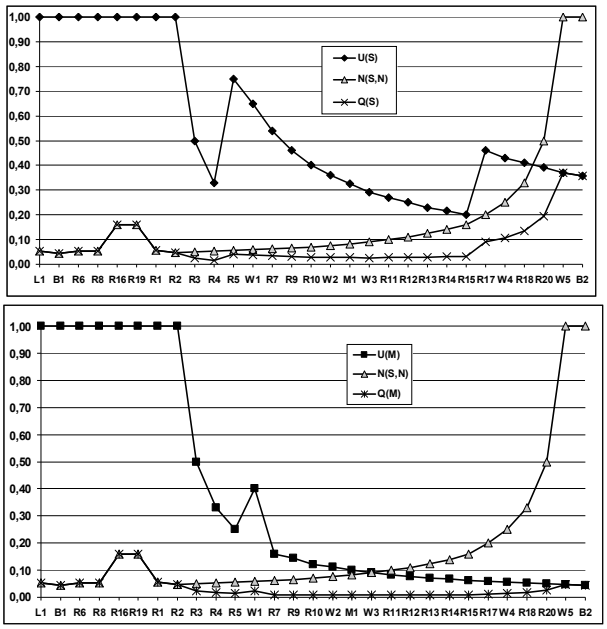


Рис. 5. Графики М-тестопригодности Xilinx модели

Если модифицировать формулу (1) исчисления тестопригодности для компонентов к виду:

$$Q_i = U_i + N_i ,$$

то кривая тестопригодности существенно поднимется вверх по оси ординат, чем обеспечивается меньший разброс параметров для каждой вершины. Данное обстоятельство фиксирует несколько отличные таблицы и графики, представленные ниже (рис. 6).

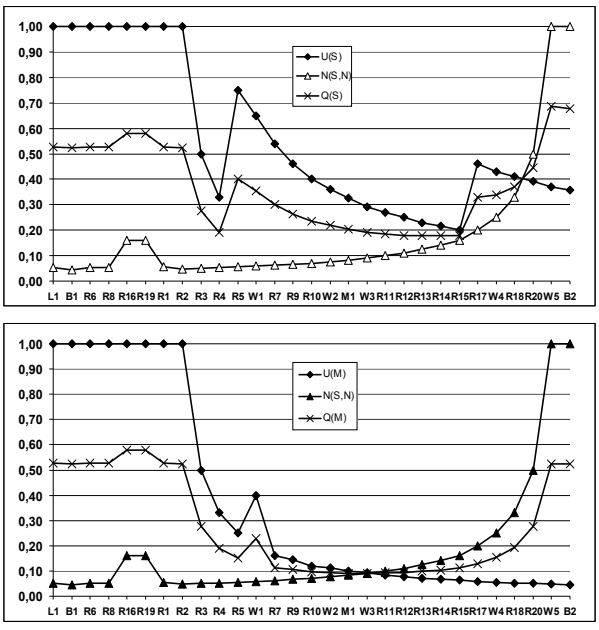


Рис. 6. Графики А-тестопригодности Xilinx модели

Интересным представляется поведение отдельных вершин. Например, управляемость вершины R_{17} в мультипликативном транзакционном графе HDL-кода неожиданно «упала» вниз по сравнению с графом единичных дуг. Это связано с высоким весом транзакций, поступающих на рассматриваемую вершину со стороны входных компонентов L_1, B_1, которые практически превращают в ноль значимость единичных транзакций от вершин R_{16}, R_{19}.

После определения управляемостей и наблюдаемостей вершин транзакционного графа выполняется подсчет обобщенного критерия тестопригодностей каждого компонента программного кода в соответствии с выражением (5). Затем определяется интегральная оценка тестопригодности проекта по формуле:

$$Q = \frac{1}{n}\sum_{i=1}^{n}(U_i \times N_i),$$

которая определяет качество проектного варианта, что представляется весьма существенным при сравнении нескольких альтернативных решений. В качестве примера позитивного использования разработанных моделей и методов предлагается анализ тестопригодности программного кода дискретного косинусного преобразования из Xilinx библиотеки. Было выполнено построение транзакционной модели, подсчет характеристик тестопригодности ($Q = \{0,382; 0,287\}$), определение критических точек. Затем в соответствии с числом и типами компонентов было разработано функциональное покрытие, фрагмент которого представлен листингом 2.

Листинг 2

```
c0: coverpoint xin
   {
     bins minus_big={[128:235]};
     bins minus_sm={[236:255]};
     bins plus_big={[21:127]};
     bins plus_sm={[1:20]};
     bins zero={0};
     }
  c1: coverpoint dct_2d
     {
     bins minus_big={[128:235]};
     bins minus_sm={[236:255]};
     bins plus_big={[21:127]};
     bins plus_sm={[1:20]};
     bins zero={0};
     bins zero2=(0=>0);
     }
endgroup
```

Для критических точек, определенных в результате анализа тестопригодности транзакционного графа разработана ассерционная модель проверки основных характеристик дискретного косинусного преобразования. Существенный фрагмент кода механизма ассерций представлен листингом 3.

Листинг 3

```
sequence first( reg[7:0] a, reg[7:0]b);
    reg[7:0] d;
    (!RST,d=a)
     ##7 (b==d);
  endsequence
 property f(a,b);
    @(posedge CLK)
      // disable iff(RST||$isunknown(a)) first(a,b);
       !RST |=> first(a,b);
 endproperty
   odin:assert property (f(xin,xa7_in))
    // $display("Very good");
   else $error("The end, xin =%b,xa7_in=%b", $past(xin,
7),xa7_in);
```

В результате проведенной верификации дискретного косинусного преобразования в среде Questa, Mentor Graphics были найдены неточности в восьми строках исходного кода HDL-модели:

```
//  add_sub1a <= xa7_reg + xa0_reg;//
```

Последующее исправление ошибок привело к появлению исправленного фрагмента кода, который показан в листинге 4.

Листинг 4

```
add_sub1a <= ({xa7_reg[8],xa7_reg} + {xa0_reg[8],xa0_reg});
add_sub2a <= ({xa6_reg[8],xa6_reg} +{xa1_reg[8],xa1_reg});
add_sub3a <= ({xa5_reg[8],xa5_reg} +{xa2_reg[8],xa2_reg});
add_sub4a <= ({xa4_reg[8],xa4_reg} + {xa3_reg[8],xa3_reg});
end
else if (toggleA == 1'b0)
begin
add_sub1a <= ({xa7_reg[8],xa7_reg} - {xa0_reg[8],xa0_reg});
add_sub2a <= ({xa6_reg[8],xa6_reg} - {xa1_reg[8],xa1_reg});
add_sub3a <= ({xa5_reg[8],xa5_reg} - {xa2_reg[8],xa2_reg});
add_sub4a <= ({xa4_reg[8],xa4_reg} - {xa3_reg[8],xa3_reg});
```

Заключение

1. Предложен комплекс технологических мероприятий и рекомендаций, ориентированных на тестопригодный анализ и последующий синтез программных продуктов, пригодных для тестирования и верификации.

2. Показаны примеры анализа тестопригодности путем подсчета управляемости и наблюдаемости транзакционного и управляющего графов в целях определения критических точек с последующим решением практической проблемы поиска и устранения ошибок в реальном DSP-проекте от компании Xilinx.

3. Построены логические функции управляемости, наблюдаемости для двух реальных примеров и графики (таблицы), соответствующие им, которые дают возможность определить критические точки для последующего улучшения кода проекта путем установки ассерций.

4. Практическая значимость предложенных методик и моделей заключается в рыночной привлекательности и высокой заинтересованности технологических компаний в инновационных решениях проблемы эф-

фективного тестирования и верификации программно-аппаратных изделий на системном уровне проектирования в целях уменьшения time-to market и повышения выхода годной продукции – yield.

5. Дальнейшие исследования будут направлены на разработку стандартных интерфейсов в целях последующей интеграции моделей, методов и программных средств в технологические маршруты проектирования цифровых систем на кристаллах.

Литература

1. Foster H. Assertion-based design – Second edition / H. Foster, A. Krolnik, D. Lacey. – Kluwer Academic Publishers, – Springer, 2005.– 392 p.

2. Abramovici M. Digital System Testing and Testable Design. Computer Science Press / M. Abramovici, M. A. Breuer, A.D. Friedman, 1998. – 652 p.

3. Тестирование и верификация HDL-моделей компонентов SOC. I. / В.И. Хаханов, Е.И. Литвинова, С.В. Чумаченко, И.А. Побеженко, C. U. Ngene // Радиоэлектроника и информатика. – 2009. – № 3.– С. 45-52.

4. Хаханов В.И. Проектирование и тестирование цифровых систем на кристаллах / В.И. Хаханов, Е.И. Литвинова, О.А. Гузь. – Х.: ХНУРЭ, 2009. – 484 с.

5. Verification Methodology. Manual for SystemVerilog / J. Bergeron, E. Cerny, A. Hunter, A. Nightingale. – Springer, 2005. – 528 p.

6. Bergeron J. Writing testbenches: functional verification of HDL models / J. Bergeron.– Boston: Kluwer Academic Publishers.– 2001.– 354 p.

7. Шаршунов С.Г. Построение тестов микропроцессоров. 1. Общая модель. Проверка обработки данных / С.Г. Шаршунов // Автоматика и телемеханика. – 1985.– № 11.– С. 145-155.

8. Jerraya A.A. System Level Synthesis SLS. TIMA Laboratory. Annual Report / A.A. Jerraya. – 2002.– P. 65-75.

Поступила в редакцию 12.02.2010

Рецензент: д-р техн. наук, проф., проф. кафедры Ф.В. Новиков, Харьковский национальный экономический университет, Харьков.

ПРИКЛАД ВИЗНАЧЕННЯ ТЕСТОПРИДАТНОСТІ ЦИФРОВОГО ПРОЕКТУ

В.І. Хаханов, Є.І. Литвинова, І.О. Побіженко, Tiecoura Yves

Запропоновано комплекс технологічних заходів і рекомендацій, орієнтованих на тестопридатний аналіз і послідовний синтез програмних продуктів і придатних для тестування й верифікації. Наведено приклади аналізу тестопридатності шляхом підрахунку керованості та спостережуваності транзакційного та керувального графів в цілях визначення критичних точок з наступним вирішенням практичної проблеми пошуку і усунення помилок в реальному DSP-проекті від компанії Xilinx.

Ключові слова: тестування, тестопридатність, верифікація, асерція, HDL-модель.

CASE-STUDY OF TESTABILITY DETERMINATION FOR DIGITAL PROJECT

V.I. Hahanov, E.I. Litvinova, I.O. Pobezhenko, Tiecoura Yves

The technological tools, focused to testable analysis and subsequent synthesis of software is proposed. It is applicable for testing and verification. The examples of testability analysis by determination the controllability and observability of the transaction and control graph to detect the critical points with subsequent solving of the problem detection and removal of faults in a real DSP project of Xilinx.

Key words: testing, testability, verification, assertion, HDL-model.

Хаханов Владимир Иванович – декан факультета КИУ ХНУРЭ, д-р техн. наук, профессор кафедры АПВТ ХНУРЭ, e-mail: hahanov@kture.kharkov.ua.

Литвинова Евгения Ивановна – канд. техн. наук, доцент, доцент кафедры технологии и автоматизации производства РЭС и ЭВС ХНУР, e-mail: kiu@kture.kharkov.ua.

Побеженко Ирина Александровна – аспирантка кафедры АПВТ ХНУРЭ, e-mail: hahanov@kture.kharkov.ua.

Tiecoura Yves – аспирант кафедры АПВТ ХНУРЭ, e-mail: hahanov@kture.kharkov.ua.