УДК 681.326:519.613

В.И. ХАХАНОВ, Е.И. ЛИТВИНОВА, И.А. ПОБЕЖЕНКО, TIECOURA YVES (ТИЕКУРА ИВ), NGENE CHRISTOPHER UMERAH (НГЕНЕ КРИСТОФЕР УМЕРАХ)

ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ HDL-МОДЕЛЕЙ КОМПОНЕНТОВ SOC. II

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).

1. Введение

Модели верификации программного HDL-кода получены с использованием среды моделирования, тестопригодного анализа логической структуры HDL-программы для квазиоптимального размещения механизма ассерций [1], применяемой в hardware design and test. Разработанные критерии управляемости и наблюдаемости [2] использованы 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].

2. 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).

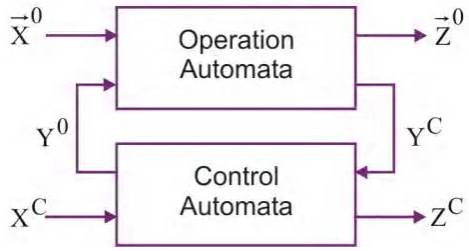


Рис. 1. Автоматная модель HDL-программы

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.

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

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

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

$$S_2 = T_3^3 T_4^1 T_6^1 T_7^2 \vee T_1^2 T_2^2 \vee T_3^3 T_5^1 T_7^2 \vee T_3^3 T_4^1 T_7^2 \vee T_3^3 T_8^2 T_9^1 T_7^2,$$

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

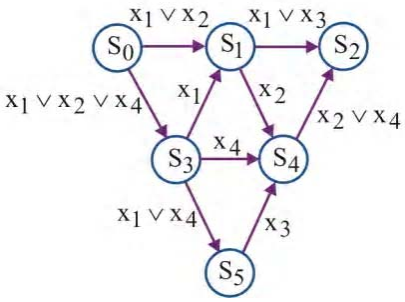


Рис. 2. Содержательный граф HDL-программы

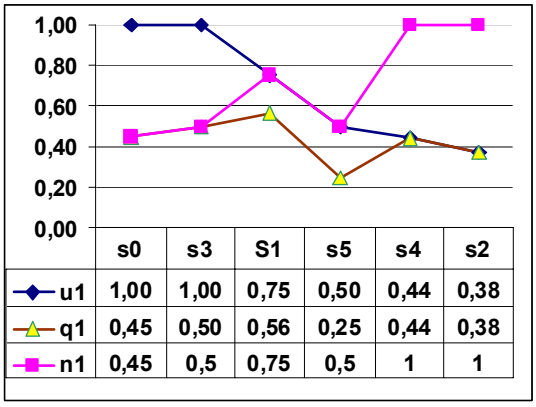
Подсчет наблюдаемостей графа [3, с. 239, формулы (1), (4)], представленного на рис. 2, содержит следующие выражения:

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

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

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

$$S_4 = T_7^2;\ S_5 = T_7^2 T_9^1.$$



	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

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:

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

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. Векторная модель среды верификации [4, 6] имеет вид:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_i \\ T_n \end{bmatrix} \oplus \begin{bmatrix} S_1 \\ S_2 \\ S_i \\ S_n \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_i \\ A_n \end{bmatrix} \xrightarrow{d} \begin{bmatrix} L_1 \\ L_2 \\ L_i \\ L_n \end{bmatrix} \tag{2}$$

В процессе моделирования выполняется сравнение реакций testbench и HDL-модели, что формирует состояния координат ассерционного вектора:

$$A_i = f(T_i, S_i) = T_i \oplus S_i,\ A_i = \{0,1,X\}.$$

Затем существенные $\{0,1\}$-координаты вектора ассерций маскируют матрицу достижимостей для получения списка программных блоков с ошибками путем выполнения одной из процедур, определенных выражениями:

$$\begin{cases} L_s(A) = \left(\bigcap_{\forall i(A_i=1)} A_i\right) \setminus \left(\bigcup_{\forall i(A_i=0)} A_i\right); \\ L_m(A) = \left(\bigcup_{\forall i(A_i=1)} A_i\right) \setminus \left(\bigcup_{\forall i(A_i=0)} A_i\right). \end{cases} \tag{3}$$

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.

3. Диагностирование по матрице достижимостей графа

Демонстрация технологии диагностирования неисправных блоков представлена следующим примером. Пусть имеется функционально-логический граф HDL-модели, изображенный на рис. 4.

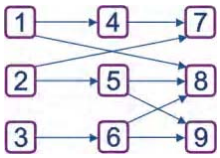


Рис. 4. Функционально-логический граф HDL-модели

Структура взаимосвязей графа представлена матрицей достижимостей:

M	1	2	3	4	5	6	7	8	9
1	1								
2		1							
3			1						
4	1			1					
5		1			1				
6			1			1			
7	1	1		1			1		
8	1	1	1		1	1		1	
9		1	1		1	1			1

(4)

Векторная модель диагностирования, заданная выражением (2), содержит списки блоков-предшественников для каждой вершины графа, которые получены из матрицы достижимостей. Каждой вершине графа ставится в соответствие ассерция, которая в процессе моделирования может быть доопределена значением {0,1}. В данном случае векторная модель поиска ошибочных программных блоков (2) для графа, представленного на рис. 4, трансформируется к виду:

T_1		S_1		$A_1 = X$		$L_1 = S_1$
T_2		S_2		$A_2 = X$		$L_2 = S_2$
T_3		S_3		$A_3 = X$		$L_3 = S_3$
T_4		S_4		$A_4 = 0$	d	$L_4 = S_1, S_4$
T_5	$\oplus$	S_5	=	$A_5 = X$	$\rightarrow$	$L_5 = S_2, S_5$
T_6		S_6		$A_6 = X$		$L_6 = S_3, S_6$
T_7		S_7		$A_7 = 1$		$L_7 = S_1, S_2, S_4, S_7$
T_8		S_8		$A_8 = 1$		$L_8 = S_1, S_2, S_3, S_5, S_6, S_8$
T_9		S_9		$A_9 = X$		$L_9 = S_2, S_3, S_5, S_6, S_9$

В результате выполнения диагностирования по системе уравнений (3), заключающейся в пересечении всех неисправных компонентов, которые соответствуют единичным координатам вектора ассерций, с последующим вычитанием объединения всех неисправных модулей, соответствующих нулевым координатам вектора A, получается список дефектных программных блоков (при условии существования только одного ошибочного модуля):

$$L_s(A_4 = 0, A_7 = 1, A_8 = 1) = L_7 \cap L_8 \setminus L_4 =$$

$$= S_1, S_2, S_4, S_7 \cap S_1, S_2, S_3, S_5, S_6, S_8 \setminus S_1, S_4 = S_2.$$

При использовании второго уравнения из выражения (3) можно получить список всех программных блоков, которые могут иметь ошибки, при условии существования нескольких дефектных компонентов:

$$L_m(A_4 = 0, A_7 = 1, A_8 = 1) = L_7 \cup L_8 \setminus L_4 =$$

$$= S_1, S_2, S_4, S_7 \cup S_1, S_2, S_3, S_5, S_6, S_8 \setminus S_1, S_4 =$$

$$= S_2, S_3, S_5, S_6, S_7, S_8.$$

Для иллюстрации следующего примера диагностического эксперимента можно убрать из рассмотрения первые два вектора, которые не являются существенными для процесса обработки списков неисправных блоков на основе анализа координат ассерционного вектора $L = d(A, S)$. Процедура вычисления списка одиночных дефектов имеет вид:

$A_1 = X$		$L_1 = S_1$		
$A_2 = X$		$L_2 = S_2$		
$A_3 = X$		$L_3 = S_3$		
$A_4 = X$	d	$L_4 = S_1, S_4$	$(L_8 \cap L_9) -$	
$A_5 = 0$	$\rightarrow$	$L_5 = S_2, S_5$	$-(L_5 \cup L_7)$	S_3, S_6
$A_6 = X$		$L_6 = S_3, S_6$		
$A_7 = 0$		$L_7 = S_1, S_2, S_4, S_7$		
$A_8 = 1$		$L_8 = S_1, S_2, S_3, S_5, S_6, S_8$		
$A_9 = 1$		$L_9 = S_2, S_3, S_5, S_6, S_9$		

Для множественных неисправных блоков на одном и том же векторе ассерций результат имеет большее число дефектных модулей:

$A_1 = X$		$L_1 = S_1$		
$A_2 = X$		$L_2 = S_2$		
$A_3 = X$		$L_3 = S_3$		
$A_4 = X$	d	$L_4 = S_1, S_4$	$(L_8 \cup L_9) -$	$S_3, S_6,$
$A_5 = 0$	$\rightarrow$	$L_5 = S_2, S_5$	$-(L_5 \cup L_7)$	S_8, S_9
$A_6 = X$		$L_6 = S_3, S_6$		
$A_7 = 0$		$L_7 = S_1, S_2, S_4, S_7$		
$A_8 = 1$		$L_8 = S_1, S_2, S_3, S_5, S_6, S_8$		
$A_9 = 1$		$L_9 = S_2, S_3, S_5, S_6, S_9$		

4. Диагностирование по векторно-логической форме графа

Интересна векторная форма модели проведения диагностического эксперимента. Здесь представлена матрица достижимостей, встроенная в модель диагностирования, где ассерции изображены в виде троичного вектора:

A	L	1	2	3	4	5	6	7	8	9
X	L_1	1								
X	L_2		1							
X	L_3			1						
X	L_4	1			1					
0	L_5		1			1				
X	L_6			1			1			
0	L_7	1	1		1			1		
1	L_8	1	1	1		1	1		1	
1	L_9		1	1		1	1			1
	L_s			1			1			

$$L_s = (L_8 \wedge L_9) \wedge$$
$$\wedge (\overline{L_5 \vee L_7}) =$$
$$= (001001000) =$$
$$= S_3, S_6$$

Результат поиска программных блоков выполнен при условии существования в проекте одного неисправного модуля, который записан в последней строке. Процедура диагностирования на основе ассерционного вектора A путем модификации (3) к выполнению векторных операций конъюнкции, дизъюнкции и отрицания

$$\begin{cases} L_s(A) = \left(\underset{\forall i(A_i = 1)}{\wedge} A_i\right) \wedge \left(\overline{\underset{\forall i(A_i = 0)}{\vee} A_i}\right); \\ L_m(A) = \left(\underset{\forall i(A_i = 1)}{\vee} A_i\right) \wedge \left(\overline{\underset{\forall i(A_i = 0)}{\vee} A_i}\right). \end{cases}$$

определена в правой части матричной модели диагностирования. Аналогичные вычисления для случая существования кратных дефектов в программных модулях проекта дают следующий результат:

A	L	1	2	3	4	5	6	7	8	9
X	L_1	1								
X	L_2		1							
X	L_3			1						
X	L_4	1			1					
0	L_5		1			1				
X	L_6			1			1			
0	L_7	1	1		1			1		
1	L_8	1	1	1		1	1		1	
1	L_9		1	1		1	1			1
	L_m			1			1		1	1

$$L_m = (L_8 \vee L_9) \wedge$$
$$\wedge (\overline{L_5 \vee L_7}) =$$
$$= (001001011) =$$
$$= S_3, S_6, S_8, S_9$$

В реальности результат диагностирования гарантирует наличие хотя бы одного дефектного блока из списка компонентов, подозреваемых в наличии неисправностей, определенных в [4].

5. Диагностирование по алгебрологической форме графа

Предлагается процедура диагностирования HDL-модели по структурно-логическому (транзакционному) графу, представленному алгебраической формой описания графовых структур [4]. Ее преимущества заключаются в компактности задания матрицы достижимостей, которая к тому же обладает свойством структуризации графа в форме задания всех путей, нагруженных на каждую вершину. Кроме того, сочетания дефектных компонентов, определяемые конъюнктивными термами, дают более точный результат диагностирования, по сравнению с заданием неисправных модулей в виде неупорядоченного множества элементов.

Для структуры, представленной на рис. 4, алгебраическая форма графа и вычисление списка одиночных неисправностей имеют следующий вид:

A	L	
X	$L_1 = S_1$	$L_s = (L_8 \wedge L_9) - (L_5 \vee L_7) =$
X	$L_2 = S_2$	$= (S_2S_5 \vee S_3S_6) - S_2S_5 =$
X	$L_3 = S_3$	$= S_3S_6$
X	$L_4 = S_1S_4$	
0	$L_5 = S_2S_5$	
X	$L_6 = S_3S_6$	
0	$L_7 = S_1S_4S_7 \vee S_2S_7$	
1	$L_8 = S_1S_8 \vee S_2S_5S_8 \vee S_3S_6S_8$	
1	$L_9 = S_2S_5S_9 \vee S_3S_6S_9$	
	$L_s = S_3, S_6$	

Процедура анализа логических функций графовой структуры содержит три пункта:

1. Все термы и переменные в ДНФ, соответствующей нулевому значению ассерционной координаты, равны нулю.

2. В других ДНФ, соответствующих единичному значению ассерционной координаты, левая часть конъюнктивного терма, включая переменную, ранее определенную нулем, удаляется.

3. Для единичных функций выполняется пересечение (объединение) оставшихся термов, если выполняется диагностирование одиночных (кратных) неисправных блоков.

Такие преобразования ДНФ путем специального моделирования нулевых сигналов переменных в единичных, относительно ассерций, логических функциях представления графа в целях получения списка неисправных программных модулей приведены в правой части алгебрологической модели диагностирования.

Следующий пример также подтверждает состоятельность анализа логических функций для вычисления множественных неисправных блоков:

A	L	
X	$L_1 = S_1$	$L_m = (L_8 \vee L_9) - (L_5 \vee L_7) =$
X	$L_2 = S_2$	$(S_1S_8 \vee S_2S_5S_8 \vee S_3S_6S_8$
X	$L_3 = S_3$	$S_2S_5S_9 \vee S_3S_6S_9) -$
X	$L_4 = S_1S_4$	$-(S_2S_5 \vee S_1S_4S_7 \vee S_2S_7) =$
0	$L_5 = S_2S_5$	$= S_3S_6S_8 \vee S_3S_6S_9$
X	$L_6 = S_3S_6$	
0	$L_7 = S_1S_4S_7 \vee S_2S_7$	
1	$L_8 = S_1S_8 \vee S_2S_5S_8 \vee S_3S_6S_8$	
1	$L_9 = S_2S_5S_9 \vee S_3S_6S_9$	
	$L_m = S_3, S_6, S_8, S_9$	

Для полученной функции L_m применение дополнительной ассерционной точки $A_6 = S_3S_6$ позволяет повысить глубину диагностирования до двух неисправных блоков:

$$L_m = S_3S_6S_8 \vee S_3S_6S_9 = \begin{cases} S_3S_6 \leftarrow A_6 = 1; \\ S_8 \vee S_9 \leftarrow A_6 = 0. \end{cases}$$

Проверка ассерционной точки $A_6 = 1$ не исключает блок S_8 из списка неисправных, но гарантирует факт наличия ошибки в компонентах S_3S_6. Поэтому после процедуры исправления некорректностей в блоках S_3S_6 необходимо повторять диагностический эксперимент. Что касается проверки ассерционной точки $A_6 = 0$, то она исключает наличие ошибок в блоках S_3S_6. Поэтому нулевая проверка всегда доставляет более ценную информацию для диагностического эксперимента. В целях уменьшения подозреваемого множества неисправных блоков следует еще одна проверка $(A_3 \vee A_9)$, которая устанавливает точный диагноз:

$$L_m = \begin{cases} = S_3 S_6 = \begin{cases} S_3 \leftarrow A_3 = 1; \\ S_6 \leftarrow A_3 = 0; \end{cases} \\ = S_8 \vee S_9 = \begin{cases} S_8 \leftarrow A_9 = 0\ ; \\ S_9 \leftarrow A_9 = 1. \end{cases} \end{cases}$$

После устранения ошибок диагностический эксперимент повторяется для поиска других составляющих кратных неисправностей HDL-блоков, которые могут присутствовать в программном коде [4].

Предложенная технология верификации является достаточно простой, ориентирована на обработку HDL-моделей большой размерности, включающей тысячи строк кода, который описывает системные структуры на языках VHDL, Verilog.

Технологический маршрут верификации программного кода представлен матрицей выполнения последовательно-параллельных процедур:

		P	
1	$F = f_2(P,S)$	$S = f_1(P)$	$T = f_3(P,S,F)$
2		$G = f_4(S,F)$	
3		$U = f_5(G,S,F)$	
4		$A = f_6(G,U,S,F,T)$	
5		$L_S = d(T,F,S,A)$	

,

которые имеют следующее содержание в виде 5 пунктов:

1) Создание HDL-модели, testbench, функционального покрытия по спецификации проекта в параллельном режиме.

2) Синтез транзакционного графа (вершины – элементы хранения информации и дуги – HDL-операторы, выполняющие транзакции между вершинами), представляющего структуру программного кода в компонентах и операторах HDL-языка. Операторы циклов размещаются в одном блоке. Граф не должен содержать глобальных обратных связей.

3) Определение тестопригодности графа путем подсчета наблюдаемостей всех вершин для планирования верификационного диагностического эксперимента.

4) Размещение ассерций в n% (25%) вершин, имеющих минимальные оценки наблюдаемостей. Их число должно обеспечивать заданную глубину диагностирования. Ассерции должны быть управляемыми, что обеспечивает их включение или отключение в процессе итеративной верификации в зависимости от результата предыдущей проверки. Данное свойство может существенно сократить время отладки программного кода. Такой подход условного (ассерционного) зондового диагностирования широко используется при тестировании сложных аппаратных цифровых изделий.

5) Диагностирование и исправление ошибок путем совместного моделирования HDL-кода, ассерций, на тестовых последовательностях testbench при условиях полноты, определенных функциональным покрытием. После устранения ошибок в дефектном блоке процедура моделирования и диагностирования повторяется.

6. Верификация DCT IP-core, Xilinx

Представленные модели верификации программного HDL-кода проверены на реальном проекте Xilinx IP-core в целях определения наличия в нем ошибок. При этом удалось получить положительный результат относительно неверной семантики работы программы для последующего исправления кода. Фрагмент модуля дискретного косинусного преобразования представлен листингом 1 [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, представленный на рис. 5, который для DCT-module Xilinx имеет 28 вершин-компонентов (входная и выходная шины, логические и регистровые переменные, векторы и память).

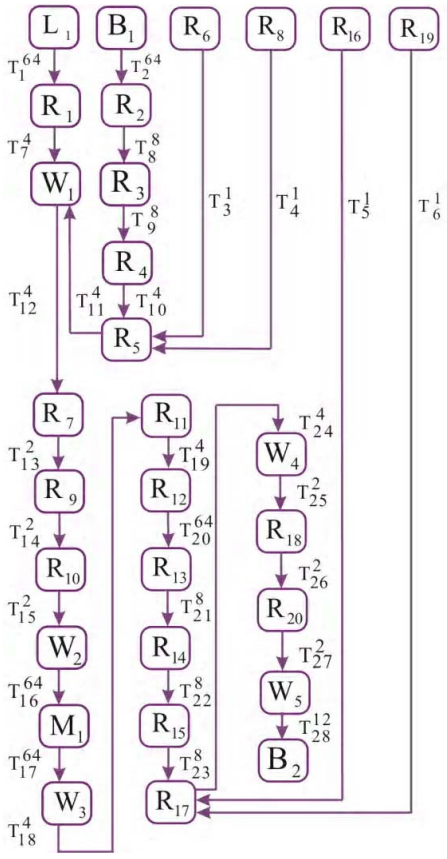


Рис. 5. Транзакционный граф Xilinx модели

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:

$$U^f(B_2) = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}(T_5^1 \vee T_6^1 \vee T_{12}^4T_{13}^2T_{14}^2T_{15}^2T_{15}^{64}T_{17}^{64}T_{18}^4T_{19}^4T_{20}^{64}T_{21}^8T_{22}^8T_{23}^8(T_1^{64}T_7^4 \vee$$

$$\vee T_{11}^4T_2^{64}T_8^8T_9^8T_{10}^4 \vee T_{11}^4T_3^1 \vee T_{11}^4T_4^1)) =$$

$$= T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_5^1 \vee T_6^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4} \vee$$

$$\vee T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^4T_{13}^2T_{14}^2T_{15}^2T_{15}^{64}T_{17}^{64}T_{18}^4T_{19}^4T_{20}^{64}T_{21}^8T_{22}^8T_{23}^8T_1^{64}T_7^4 \vee$$

$$\vee T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}T_{11}^{4}T_{2}^{64}T_{8}^{8}T_{9}^{8}T_{10}^{4} \vee$$

$$\vee T_{11}^{4}T_{3}^{1}T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8} \vee$$

$$\vee T_{11}^{4}T_{4}^{1}T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}.$$

Для остальных вершин аналогично выполняется вычисление ДНФ управляемостей.

Для вершины L_1 ДНФ наблюдаемости имеет вид:

$$N^{f}(L_1) = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{23}^{8}T_{22}^{8}T_{21}^{8}T_{20}^{64}T_{19}^{4}T_{18}^{4}T_{17}^{64}T_{16}^{64}T_{15}^{2}T_{14}^{2}T_{7}^{4}T_{1}^{64}.$$

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 можно рассматривать два 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 между двумя вершинами транзакционного графа. Оценки тестопригодности описанных процедур могут существенно отличаться 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 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. 6, 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, находящимся ближе к выходной шине B_2. Здесь существенным представляется не структура графа, а вес входящей дуги, 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).

Если модифицировать формулу (1) исчисления тестопригодности для компонентов к следующему 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. 7).

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 тестопригодности программного кода (формула (5), [3]). Для Xilinx DCT-модели такая оценка равна 0,382. Она характеризует качество проектного варианта, что представляется весьма существенным при сравнении нескольких альтернативных решений. В качестве примера позитивного использования разработанных моделей и методов был выполнен анализ тестопригодности программного кода дискретного косинусного преобразования (DCT) из Xilinx библиотеки. Построена транзакционная модель, вычислены характеристики тестопригодности и определены критические точки (R_1, R_5, R_9, R_{15}). В соответствии с числом и типами компонентов было разработано функциональное покрытие, фрагмент которого представлен листингом 2.

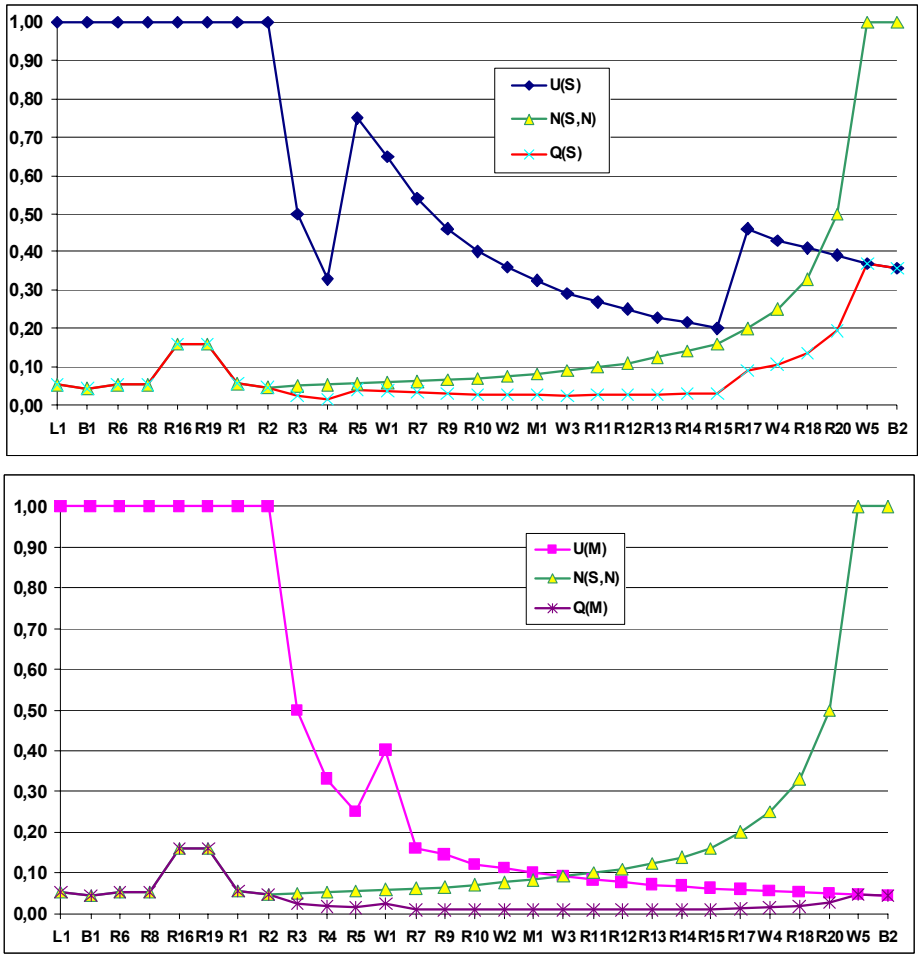


Рис. 6. Графики М-тестопригодности Xilinx модели

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 верификации программной HDL-модели дискретного косинусного преобразования в среде моделирования Active-HDL были найдены неточности в восьми строках исходного кода HDL-модели:

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

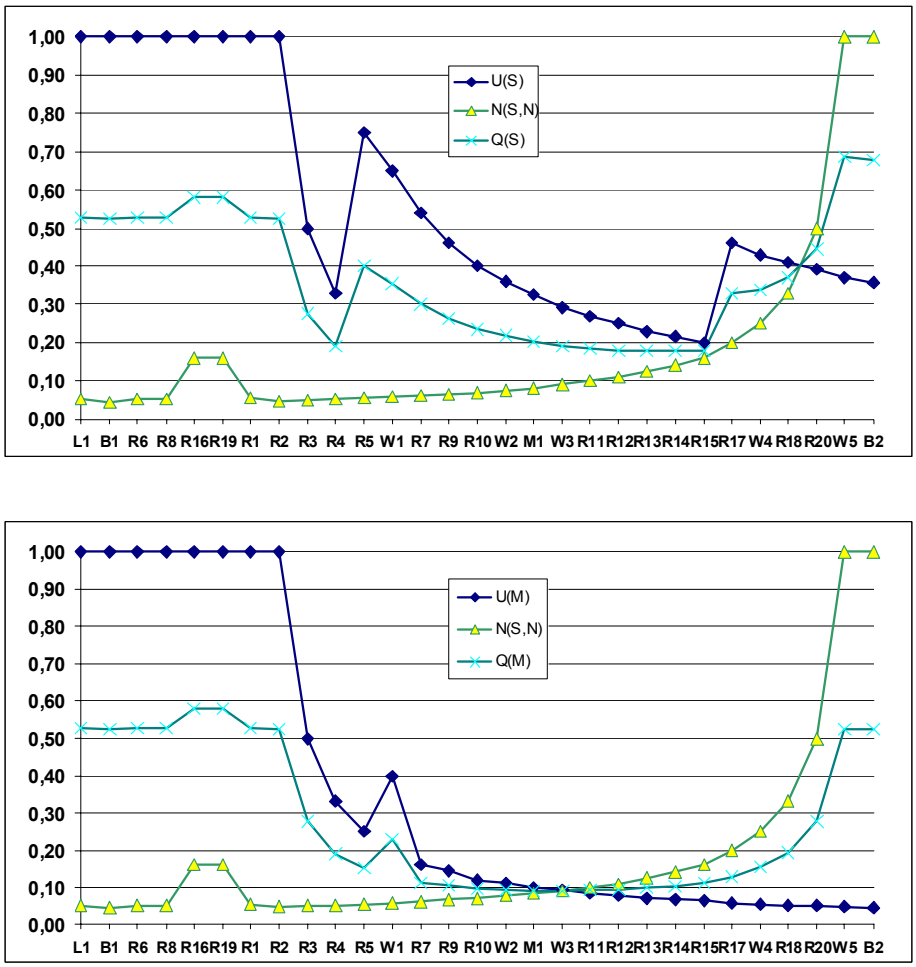


Рис. 7. Графики А-тестопригодности Xilinx модели

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});
```

7. Выводы

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 эффективного 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.

Список литературы: 1. *Harry Foster, Adam Krolnik, David Lacey.* Assertion-based design.– Second edition. Kluwer Academic Publishers. Springer. 2005. 392 p. **2.** *Abramovici M., Breuer M.A. and Friedman A.D.* Digital System Testing and Testable Design. Computer Science Press. 1998. 652 p. **3.** *Хаханов В.І., Литвинова Є.І., Побеженко І.О., Ngene Christopher Umerah, Чумаченко С.В.* Тестирование и верификация HDL-моделей компонентов SOC. I // Радиоэлектроника и информатика. 2009. № 3. С. 38-45. **4.** *Хаханов В.И., Литвинова Е.И., Гузь О.А.* Проектирование и тестирование цифровых систем на кристаллах. Харьков: ХНУРЭ. 2009. 484 с. **5.** *Janick Bergeron, Eduard Cerny, Alan Hunter, Andrew Nightingale.* Verification Methodology. Manual for SystemVerilog. Springer. 2005. 528 p. **6.** *Bergeron, Janick.* Writing testbenches: functional verification of HDL models. Boston: Kluwer Academic Publishers. 2001. 354 p. **7.** *Шаршунов С.Г.* Построение тестов микропроцессоров. 1. Общая модель. Проверка обработки данных // Автоматика и телемеханика. 1985. №11. С. 145-155. **8.** *Jerraya A.A.* System Level Synthesis SLS. TIMA Laboratory. Annual Report. 2002. P. 65-75.

Поступила в редколлегию 11.07.2009

Хаханов Владимир Иванович, декан факультета КИУ ХНУРЭ, д-р техн. наук, профессор кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем, сетей и программных продуктов. Увлечения: баскетбол, футбол, горные лыжи. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326. E-mail: hahanov@kture.kharkov.ua.

Литвинова Евгения Ивановна, канд. техн. наук, доцент кафедры технологии и автоматизации производства РЭС и ЭВС ХНУРЭ. Научные интересы: автоматизация диагностирования и встроенный ремонт компонентов цифровых систем в пакете кристаллов. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-421. E-mail: kiu@kture.kharkov.ua.

Побеженко Ирина Александровна, аспирантка кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем и сетей. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326. E-mail: hahanov@kture.kharkov.ua.

Tiecoura Yves, аспирант кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем и сетей. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326. E-mail: hahanov@kture.kharkov.ua.

Ngene Christopher Umerah, аспирант кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем и сетей. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326. E-mail: hahanov@kture.kharkov.ua.