Автоматика и телемеханика, № 1, 2013

© 2013 г. Л.А. ЗОЛОТОРЕВИЧ, канд. техн. наук,
(Белорусский государственный университет, Минск)

ВЕРИФИКАЦИЯ ПРОЕКТОВ И ПОСТРОЕНИЕ ТЕСТОВ КОНТРОЛЯ СБИС НА УРОВНЕ RTL

Предлагается метод верификации проектов и направленного построения тестов контроля СБИС, представленных на уровне RTL на языке VHDL. Задача построения тестов и верификации проектов решается на основе КНФ – выполнимости некоторой системы булевых функций.

1. Состояние проблемы верификации проектов и построения тестов

Проблемы верификации проектов и построения тестов контроля сложнофункциональных электронных систем остаются наиболее наукоемкими во всем спектре проблем проектирования СБИС [1]. Современные интегральные схемы содержат порядка миллиардов транзисторов на кристалле, и разработка тестов для объектов такого размера на уровне структурного представления оказалась практически неразрешимой задачей. В то же время острая потребность в тестах имеет место на всех этапах жизненного цикла, начиная с самого начала процесса проектирования, так как формальные методы верификации развиты недостаточно, и верификация проектов на практике осуществляется на основе моделирования. Тесты контроля необходимы также на этапе производства для отбраковки готовых изделий и при эксплуатации для оценки работоспособности объектов.

Следует заметить, что быстро развивающаяся электронная отрасль выставляет все новые требования и условия к задаче построения тестов. Если в конце прошлого века рассматривалась задача эффективного построения теста контроля на уровне заданной структуры объекта, то в настоящее время кроме данной задачи дана формулировка и ведется поиск решения задачи построения тестов для систем, представленных в разных системах идентификации.

Подойти к решению задачи построения тестов для контроля СБИС оказалось возможным на основе идентификации объекта на начальных этапах проектирования, когда имеется некоторое поведенческое описание или описание объекта на уровне межрегистровых передач, которое содержит существенно меньшее число базовых примитивов, чем на структурном уровне. Вопросы разработки тестов контроля СБИС, когда отсутствуют сведения относительно структурной реализации объекта, решаются преимущественно на основе моделирования неисправностей и случайного построения и рассматриваются в [2–4] . В [5] предлагается методика построения функциональных неисправностей, аргументированно соответствующих неисправностям структурной реализации соответствующего механизма. Известны работы, рассматривающие

задачи направленного построения тестов на верхних уровнях проектирования [3, 4].

В [5] приведен общий подход к иерархической генерации тестов СБИС на RTL-уровне. Он основан на том, что каждая операция программного кода, описывающего объект на уровне RTL, реализуется на структурном уровне некоторым набором аппаратных средств. Тест контроля этих средств может быть построен известными методами и средствами на основе структурного представления устройства. Тест вносится в описание объекта, устанавливаются ограничения на функционирование объекта. Задача построения теста контроля всего объекта сводится к решению системы арифметических уравнений с внесенными ограничениями.

В настоящей работе описывается механизм решения задачи направленного построения тестов, основанный на построении системы арифметических уравнений и итерационном решении задачи КНФ-выполнимости некоторой системы булевых функций.

В разделе 2 приводится структура представления объекта на языке VHDL. В разделе 3 рассматривается задача построения тестов на основе решения системы арифметических уравнений. В разделе 4 предложенный метод построения тестов распространяется на решение задачи верификации. Проблема построения тестов непосредственно связана с проблемой верификации проектов сложно-функциональных цифровых систем. На сегодняшний день проблема верификации проектов на разных этапах проектирования решается, в основном, на основе моделирования объекта, так как применяемые методы формальной верификации ориентированы на решение некоторых частных задач. Полнота верификации проектов обеспечивается полнотой применяемых при моделировании тестов. Предлагаемый в данной статье метод направлен как на решение задачи направленного построения тестов контроля, так и на верификацию проекта путем моделирования на заданном тесте.

2. Структура представления объекта на языке VHDL

С учетом сложившейся традиции цифровая система рассматривается на уровне RTL как две подсистемы – операционная, выполняющая преобразование данных в соответствии с заданными алгоритмами, и управляющая, реализующая управление операционной частью. Поэтому в качестве математической платформы для описания цифровой системы будем использовать описание в виде графов потоков данных и потока управления. Граф потока управления – это ориентированный граф с узлами, соответствующими операторам программного кода, и ребрами, указывающими порядок выполнения операторов. Граф потока данных – это ориентированный граф с узлами, соответствующими выполняемым операциям и ребрами, указывающими последовательность операций по преобразованию некоторой переменной или сигнала.

Граф потока управления и графы потоков данных разрабатываются при статическом анализе программного кода. Отображение описания объекта в виде графов потоков данных и потока управления применяется обычно при создании компиляторов. По существу графы потоков данных и потока управ-

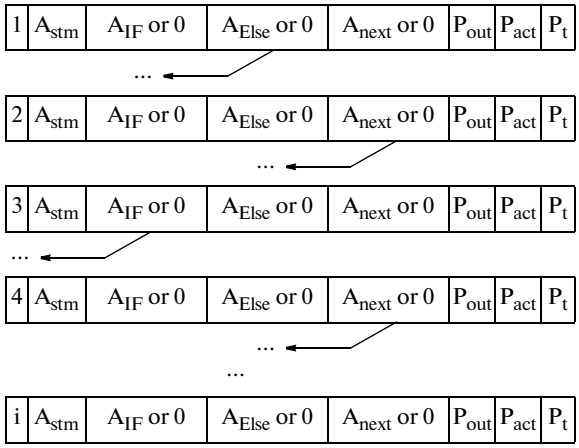

Рис. 1. Структура DD потока управления.

ления представляют собой внутреннюю структуру данных компилятора, в которой заложен результат анализа текста программы. К сожалению, внутреннее представление объекта в существующих фирменных компиляторах VHDL не доступно для использования извне. Большая часть работ по построению графов потоков данных и потока управления базируется на применении моделей, построенных вручную. В [6] предложена реализация идеи формального построения графа применительно к С-спецификации. Ниже приводится структура внутреннего представления RTL-описания для реализации метода направленного построения теста.

На рис. 1 приведена общая структура DD потока управления. Это есть ориентированный ациклический граф, в котором вершинами являются операторы языка VHDL. Ориентированные ребра соответствуют передаче управления от одной вершины к другой. Каждый оператор программного кода представлен некоторым узлом DD потока управления. Узел имеет маркер, который представляется семеркой вида $< A_{stm}, A_{if}, A_{else}, A_{next}, P_{out}, P_{act}, P_t >$.

Первое значение A_{stm} является адресом выполняемой логической или арифметической операции, последующие значения A_{if}, A_{else}, A_{next} представляют собой ссылки на соответствующие узлы графа DD потока управления. Если узел № 1 соответствует оператору if и условие ложно, то $A_{if} = 0$, а A_{else} содержит указатель на узел DD, соответствующий следующему выполняемому оператору. Если узел соответствует не оператору if, а некоторому оператору с логической или арифметической операцией, $A_{if} = 0$, $A_{else} = 0$, а A_{next} содержит указатель на некоторый узел DD, соответствующий следующему выполняемому оператору. P_{out} – признак наблюдаемости, P_{act} – признак активизации соответствующего оператора, P_t – время активизации.

На рис. 2 приведена структура DD потоков данных: Stm – адрес начала описания узла; ID – идентификатор левого операнда соответствующего оператора; A_{prvs} – ссылка на предшествующий оператор с аналогичным идентификатором левого операнда (или нуль); A_{next} – ссылка на последую-

Stm	ID	A_{prvs} or “0”	A_{next} or “0”	Expression
Stm_1				
Stm_2			A_{next} or “0”	
...				
Stm_n				

Рис. 2. Структура DD потоков данных.

1	A^1_{stm}	A_{IF}	A_{Else}	“0”	P^1_{out}	P^1_{act}	P^1_t
2	A^2_{stm}	“0”	“0”	A_{next}	P^2_{out}	P^2_{act}	P^2_t
3	A^3_{stm}	“0”	“0”	“0”	P^3_{out}	P^3_{act}	P^3_t
4	A^4_{stm}	A_{IF}	“0”	“0”	P^4_{out}	P^4_{act}	P^4_t
5	A^5_{stm}	“0”	“0”	A_{next}	P^5_{out}	P^5_{act}	P^5_t
6	A^6_{stm}	“0”	“0”	A_{next}	P^6_{out}	P^6_{act}	P^6_t
7	A^7_{stm}	“0”	“0”	A_{next}	P^7_{out}	P^7_{act}	P^7_t
8	A^8_{stm}	“0”	“0”	A_{next}	P^8_{out}	P^8_{act}	P^8_t
9	A^9_{stm}	“0”	“0”	A_{next}	P^9_{out}	P^9_{act}	P^9_t
10	A^{10}_{stm}	“0”	“0”	“0”	P^{10}_{out}	P^{10}_{act}	P^{10}_t

Рис. 3. DD потока управления.

щий оператор с аналогичным идентификатором левого операнда (или нуль); Expression – вычисляемое арифметическое или логическое выражение.

Ниже рассмотрен пример построения DD потока управления (рис. 3) и потоков данных (рис. 4) для фрагмента программного кода, приведенного на рис. 5.

В теории тестового диагностирования известны методы направленного и случайного построения тестов. Если рассматривать возможность их реализации в указанной выше постановке, то следует отметить весьма существенную особенность. Если методы построения теста случайным образом можно пытаться реализовать на основе имеющихся фирменных компиляторов языка

Stm	DD	A_{prvs}	A_{next}	Expression
Stm_1	RST	0	0	rst='1'
Stm_2	RAB1	0	A^5_{stm}	RAB1:=0;
Stm_3	RAB2	0	A^6_{stm}	RAB2:=0;
Stm_4		0	0	clk' event and clk=1
		...		
Stm_5	RAB1	A^2_{stm}	0	RAB1:=X1;
Stm_6	RAB2	A^3_{stm}	0	RAB2:=X2;
Stm_7	RAB3	0	0	RAB3:=RAB1+RAB2;
Stm_8	RAB4	0	0	RAB4:=RAB5;
Stm_9	Z1	0	0	Z1:=RAB4;
Stm_{10}	Z2	0	0	Z2:=RAB3;

Рис. 4. DD потоков данных.

```
entity Vonescnt is
      port (X1, X2, rst, clk: in bit;
            Zl, Z2: out bit);
end Vonescnt;
architecture Vonescnt_arch of Vonescnt is
process (rst, elk)
variable Rabl, Rab2, Rab3, Rab4, Rab5:bit
   begin
1       if (rst='l')
2           RAB1:=0;
3           RAB2:=0;
4       Else if (clk' event and clk=l)

5                              RAB1:=X1;
6                              RAB2:=X2;
7           ...                RAB3:=RAB1+RAB2;
8           ...                RAB4:=RAB5;
9           ...                Z1:=RAB4;
10           else              Z2:=RAB3;
            ...
     end if;
end process
end Vonescnt;
```

Рис. 5. Фрагмент программного кода.

VHDL, то для направленного построения тестов необходимо знание внутреннего представления программного кода для построения структур графов, которое, к сожалению, не раскрывается разработчиками компиляторов и не может быть использовано при необходимости. Для построения графов необходим синтаксический анализатор программного кода, который используется при построении одного графа потока управления и столько графов потоков данных, сколько переменных описывает полное состояние моделируемой системы.

Ответственным моментом является выбор моделей неисправностей рассматриваемых операторов программного кода. В настоящее время нет доказательной базы для выбора некоторой определенной модели используемой неисправности, поэтому целесообразно работать в более широком диапазоне моделей, известных в литературе. В данной работе, кроме известных моделей, таких как модель выпавшего оператора, замены условного перехода безусловным, одной операции некоторой другой, неисправности константного типа некоторой переменной, сигнала, некоторого разряда переменной, – будет использоваться так же и модель функциональной неисправности, которая аргументированно соответствует неисправности константного типа реального объекта [5].

Научная гипотеза, используемая при разработке метода направленного построения тестов на RTL-уровне, формулируется так: для того, чтобы найти входные данные, которые позволят определить по выходным данным наличие или отсутствие некоторой неисправности в системе, представленной в рамках любой системы идентификации, необходимо активизировать неисправность, т.е. заставить ее проявиться на выходе некоторого элемента системы, затем заставить ее проявиться хотя бы на одном из выходов системы, после чего необходимо вычислить все входные данные, которые позволят сохранить условия, полученные при решении данной задачи. Данная гипотеза сформулирована на основе идеи Рота, реализованной при построении тестов контроля объекта на структурном уровне.

3. Построение и решение системы арифметических уравнений

Общая идея метода направленного построения тестов контроля цифровых систем, описанных на уровне RTL на языке VHDL заключается

• в переходе от системы арифметических и логических уравнений, описывающих поведение объекта с некоторой внесенной неисправностью, к системе КНФ булевых функций разрешения;

• конъюнктивном объединении функций разрешения;

• решении задачи выполнимости результирующей КНФ разрешения объекта.

Предположим, что все входные переменные являются целочисленными размерностью n бит (mod 2^n). Для генерации тестов необходимо:

1) на основе программного кода объекта составить систему арифметических уравнений, описывающих функционирование объекта;

2) выполнить корректировку системы с учетом внесения неисправностей соответствующего оператора;

3) поставить в соответствие каждой целочисленной переменной размерностью n бит (mod 2^n) логический вектор длины n;

4) итеративно найти решение системы. Для получения i-го бита результата арифметические выражения транслируются в КНФ булевых функций разрешения (правила перехода приведены в разделе 4);

5) все полученные КНФ-функции разрешения объединяются по правилу И; решается задача КНФ-выполнимости полученной системы булевых

```
if S = '0' then
  D <= A;
else D <= B;
  G <= B + C;
  E <= D + C;
  F <= E * G;
  L <= D < G;
endif;
```

Рис. 6. Фрагмент программного кода.

$$D = \overline{S} * A + S * B;$$
$$\boxed{G = B + C;}$$
$$E = D + C;$$
$$F = E * G;$$
$$L = D < G;$$

Рис. 7. Система арифметических уравнений (mod 2^n).

функций. Если система выполнима, то получен i-й бит разрабатываемого теста контроля объекта с внесенной неисправностью. В противном случае тест не может быть построен, так как внесенные ограничения не могут быть удовлетворены. В таком случае для проверки рассматриваемой неисправности необходимо изменить систему управления объекта с целью повышения его управляемости и наблюдаемости.

Пример. Рассмотрим фрагмент некоторого программного кода, приведенный на рис. 6. Здесь A, B, C, S – входные данные, L – переменная выхода. Положим, что переменные A, B, C являются целочисленными по модулю 2^n, а S – однобитовая переменная. На рис. 7 приведена система арифметических уравнений, описывающих функционирование объекта, представленного программным кодом, приведенным на рис. 6. В объекте имеется мультиплексор, два сумматора, умножитель и схема сравнения. Построим тест, проверяющий правильность выполнения оператора целочисленного сложения $G = B + C$.

С данным оператором связаны аппаратные средства (некоторый механизм реализации), обеспечивающие сложение целых чисел. Положим, что известен тест (последовательность входных наборов) для контроля сумматора по модулю 2^n и выбран один набор теста, который задает $B = 15$, $C = 1$, $G' = 17$ по модулю 2^n , где G' – неисправное значение переменной G. Для построения теста контроля рассматриваемого объекта система уравнений (рис. 7) должна быть скорректирована, как показано в табл. 1, чтобы обеспечить распространение неисправности к выходам объекта и определение входных переменных. Здесь уравнения 1–5 описывают исправный исходный объект, операторы 6–8 описывают процесс внесения неисправности, операторы 9–13 задают ограничения, обеспечивающие распространение эффекта неисправности к выходу объекта.

Идея решения подобных систем арифметических уравнений основана на поразрядном подходе к вычислению значений и состоит в том, что каждой целочисленной переменной $m <= 2n$ ставится в соответствие определенный логический вектор размерности n. Для описания алгоритма вычисления си-

Таблица 1. Система арифметических уравнений (mod 2^n) с внесенной неисправностью

№ п/п	Арифметические уравнения	№ п/п	Арифметические уравнения
1	$D = \overline{S} * A + S * B;$ (mod 2^n)	9	$G' \neq G;$ (mod 2^n)
2	$G = B + C;$ (mod 2^n)	10	$F' = E * G';$ (mod 2^n)
3	$E = D + C;$ (mod 2^n)	11	$F \neq F'$ (mod 2^n)
4	$F = E * G;$ (mod 2^n)	12	$L' = D < G';$ (mod 2^n)
5	$L = D < G;$ (mod 2^n)	13	$L \neq L';$ (mod 2^n)
6	$B = 15;$ (mod 2^n)		
7	$C = 1;$ (mod 2^n)		
8	$G' = 17;$ (mod 2^n)		

Таблица 2. Функции разрешения и запрета

a	b	f	F^f	$\overline{F^f}$
0	0	0	1	0
0	1	0	1	0
1	0	0	1	0
1	1	1	1	0
0	0	1	0	1
0	1	1	0	1
1	0	1	0	1
1	1	0	0	1

стемы арифметических уравнений используем логическую функцию разрешения, которая задает соотношения между исправными логическими состояниями выводов физических элементов, реализующих определенную логическую функцию.

Функция F^f, называемая функцией разрешения для логической функции f, вводится в [7]. Функция F^f зависит не только от аргументов функции f, но и от самой f и принимает значение логической 1 при всех допустимых состояниях входных и выходной переменных. Функция F^f принимает значение 0 при всех недопустимых состояниях входных и выходной переменных. Для полноты изложения рассмотрим получение функции разрешения F^f для функции конъюнкции $f = a * b$ (табл. 2).

Получим СКНФ функции F^f по табл. 2, выбирая конституэнты 0. Для этого

1) выбираются наборы аргументов, на которых функция обращается в нуль;

2) выписываются дизъюнкции, соответствующие этим наборам, причем если аргумент x_i входит в набор как нуль, то в дизъюнкцию он вписывается без изменения. Если же аргумент x_i входит в данный набор как единица, то в соответствующую дизъюнкцию вписывается его отрицание;

Таблица 3. Функции разрешения

№ п/п	Однобитовые арифметические и логические уравнения	КНФ функций разрешения
1	$f = b \vee c$	$(\overline{b} \vee f)(\overline{c} \vee f)(b \vee c \vee \overline{f})$
2	$f = b * c$	$(b \vee \overline{f})(c \vee \overline{f})(\overline{b} \vee \overline{c} \vee f)$
3	$f = a \oplus b$	$(a \vee b \vee \overline{f})(a \vee \overline{b} \vee f)(\overline{a} \vee b \vee f)(\overline{a} \vee \overline{b} \vee \overline{f})$
4	$f = a \sim b$	$(a \vee b \vee f)(a \vee \overline{b} \vee \overline{f})(\overline{a} \vee b \vee \overline{f})(\overline{a} \vee \overline{b} \vee f)$
5	$f = \overline{s} * a + s * b$	$(s \vee a \vee \overline{f})(a \vee b \vee \overline{f})(\overline{s} \vee b \vee \overline{f}) *$ $*(s \vee \overline{a} \vee f)(\overline{s} \vee \overline{b} \vee f)(\overline{a} \vee \overline{b} \vee f)$
6	$f = a \geqslant b$	$(a \vee f)(\overline{a} \vee \overline{f})$
7	$f = b < c$	$(\overline{b} \vee \overline{f})(c \vee \overline{f})(b \vee \overline{c} \vee f)$
8	$f = 1$	f
9	$f = 0$	$\overline{f}$
10	$f \neq f'$	$(f \vee f' \vee \overline{f''})(f \vee \overline{f'} \vee f'')(\overline{f} \vee f' \vee f'') *$ $*(\overline{f} \vee \overline{f'} \vee \overline{f''})$

Таблица 4. Функции разрешения для вычисления 1-го бита результата (f_0''-результат предыдущей итерации)

№ п/п	Арифметические уравнения	Эквивалентные функции разрешения
1	$D_0 = \overline{S_0} * A_0 + S_0 * B_0 \pmod 2$	$(S_0 \vee A_0 \vee \overline{D_0})(A_0 \vee B_0 \vee \overline{D_0})(\overline{S_0} \vee B_0 \vee \overline{D_0}) *$ $*(S_0 \vee \overline{A_0} \vee D_0)(\overline{S_0} \vee \overline{B_0} \vee D_0)(\overline{A_0} \vee \overline{B_0} \vee D_0)$
2	$G_0 = B_0 + C_0 \pmod 2$	$(C_0 \vee B_0 \vee \overline{G_0})(C_0 \vee \overline{B_0} \vee G_0)(\overline{C_0} \vee B_0 \vee G_0) *$ $*(\overline{C_0} \vee \overline{B_0} \vee \overline{G_0})$
3	$E_0 = D_0 + C_0 \pmod 2$	$(C_0 \vee D_0 \vee \overline{E_0})(C_0 \vee \overline{D_0} \vee E_0)(\overline{C_0} \vee D_0 \vee E_0) *$ $*(\overline{C_0} \vee \overline{D_0} \vee \overline{E_0})$
4	$F_0 = E_0 * G_0 \pmod 2$	$(E_0 \vee \overline{F_0})(G_0 \vee \overline{F_0})(\overline{E_0} \vee \overline{G_0} \vee F_0)$
5	$L_0 = D_0 < G_0 \pmod 2$	$(\overline{D_0} \vee \overline{L_0})(G_0 \vee \overline{L_0})(D_0 \vee \overline{G_0} \vee L_0)$
6	$B_0 = 1 \pmod 2$	B_0
7	$C_0 = 1 \pmod 2$	C_0
8	$G_0' = 1 \pmod 2$	G_0'
9	$G_0' \neq G_0 \pmod 2$	$(G_0 \vee G_0' \vee \overline{f_0''})(G_0 \vee \overline{G_0'} \vee f_0'')(\overline{G_0} \vee G_0' \vee f_0'') *$ $*(\overline{G_0} \vee \overline{G_0'} \vee \overline{f_0''})$
10	$F_0' = E_0 * G_0' \pmod 2$	$(E_0 \vee \overline{F_0'})(G_0 \vee \overline{F_0'})(\overline{E_0} \vee \overline{G_0} \vee F_0')$
11	$F_0 \neq F_0' \pmod 2$	$(F_0 \vee F_0' \vee \overline{f_0''})(F_0 \vee \overline{F_0'} \vee f_0'')(\overline{F_0} \vee F_0' \vee f_0'') *$ $*(\overline{F_0} \vee \overline{F_0'} \vee \overline{f_0''})$
12	$L_0' = D_0 < G_0' \pmod 2$	$(\overline{D_0} \vee \overline{L_0'})(G_0' \vee \overline{L_0'})(D_0 \vee \overline{G_0'} \vee L_0')$
13	$L_0 \neq L_0' \pmod 2$	$(L_0 \vee L_0' \vee \overline{f_0''})(L_0 \vee \overline{L_0'} \vee f_0'')(\overline{L_0} \vee L_0' \vee f_0'') *$ $*(\overline{L_0} \vee \overline{L_0'} \vee \overline{f_0''})$

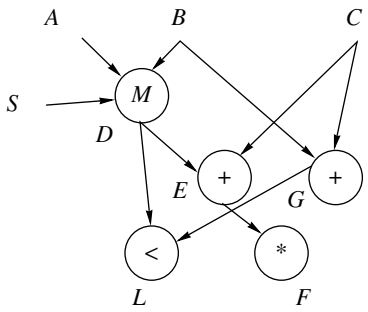

Рис. 8. Ярусно-параллельная форма представления кода.

3) все выписанные дизъюнкции соединяют знаком конъюнкции:

$$F^f = (a \vee b \vee \overline{f})(a \vee \overline{b} \vee \overline{f})(\overline{a} \vee b \vee \overline{f})(\overline{a} \vee \overline{b} \vee f) = \\ = (a \vee \overline{f})(b \vee \overline{f})\left(\overline{a} \vee \overline{b} \vee f\right).$$

Заметим, что функция $\overline{F^f}$, называемая функцией запрета, принимает значения, инверсные функции разрешения F^f. Выражение функции разрешения для операции поразрядного сложения $f = a \oplus b$ выглядит так: $(a \vee b \vee \overline{f})(a \vee \overline{b} \vee f)(\overline{a} \vee b \vee f)(\overline{a} \vee \overline{b} \vee \overline{f})$. В табл. 3 приведены однобитовые операции, используемые в рассматриваемом в табл. 1 примере и соответствующие функции разрешения в виде КНФ.

Решение системы уравнений, приведенной в табл. 1, выполняется итеративно. Чтобы одновременно решить равенства по mod 2, объединим все КНФ разрешения логических функций вместе, используя логическую функцию конъюнкции. Чтобы найти второй бит решения этого уравнения, надо найти рекурсивное уравнение по mod 2 аналогично, используя результаты предыдущей итерации.

Для вычисления очередного бита результата формируется система функций разрешения, соответствующих каждой арифметической функции, затем решается задача выполнимости конъюнкции всех функций разрешения. К примеру, для уравнения $E = D + C$ вначале вычисляется значение младшего бита результата $E_0 = D_0 + C_0$. После вычисления E_0 его результат используется при формировании следующих битов результата. Для определения порядка вычисления битов высшего порядка из битов более низкого порядка необходимо рассмотреть различные формы уравнений. Для операции суммирования начиная со 2-й итерации очередной бит результата вычисляется следующим образом: $E_i = D_i + C_i + P_i$, где P_i – значение переноса из $i + 1$-го разряда.

В табл. 4 приведены функции разрешения для первого бита системы арифметических уравнений, приведенных в табл. 1.

Заметим, что для рекурсивного вычисления неравенства необходимо учитывать, что неравенство может быть определено только в старшем i-м бите (i от 0 до $n-1$). Все функции разрешения объединяются знаком конъюнкции,

решается задача выполнимости полученной системы булевых функций:

$$\Big((S_0 \vee A_0 \vee \overline{D_0})(A_0 \vee B_0 \vee \overline{D_0})(\overline{S_0} \vee B_0 \vee \overline{D_0})*$$
$$*(S_0 \vee \overline{A_0} \vee D_0)(\overline{S_0} \vee \overline{B_0} \vee D_0)(\overline{A_0} \vee \overline{B_0} \vee D_0)\Big)*$$
$$*\Big((C_0 \vee B_0 \vee \overline{G_0})(C_0 \vee \overline{B_0} \vee G_0)(\overline{C_0} \vee B_0 \vee G_0) * (\overline{C_0} \vee \overline{B_0} \vee \overline{G_0})\Big)*$$
$$*\Big((C_0 \vee D_0 \vee \overline{E_0})(C_0 \vee \overline{D_0} \vee E_0)(\overline{C_0} \vee D_0 \vee E_0) * (\overline{C_0} \vee \overline{D_0} \vee \overline{E_0})\Big)*$$
$$*\Big((E_0 \vee \overline{F_0})(G_0 \vee \overline{F_0})(\overline{E_0} \vee \overline{G_0} \vee F_0)\Big) * \Big((\overline{D_0} \vee \overline{L_0})(G_0 \vee \overline{L}_0)(D_0 \vee \overline{G_0} \vee L_0)\Big)*$$
$$B_0 * C_0 * G'_0 *$$
$$*\Big((G_0 \vee G'_0 \vee \overline{f''_0})(G_0 \vee \overline{G'}_0 \vee f''_0)(\overline{G_0} \vee G'_0 \vee f''_0) * (\overline{G_0} \vee \overline{G'}_0 \vee \overline{f''_0})\Big)*$$
$$*\Big((E_0 \vee \overline{F'_0})(G_0 \vee \overline{F'_0})(\overline{E}_0 \vee \overline{G_0} \vee F'_0)\Big)*$$
$$*\Big((F_0 \vee F'_0 \vee \overline{f''}_0)(F_0 \vee \overline{F'_0} \vee f''_0)(\overline{F_0} \vee F'_0 \vee f''_0) * (\overline{F_0} \vee \overline{F'_0} \vee \overline{f''_0})\Big)*$$
$$*\Big((\overline{D}_0 \vee \overline{L'_0})(G'_0 \vee \overline{L'}_0)(D_0 \vee \overline{G'_0} \vee L'_0)\Big)*$$
$$*\Big((L_0 \vee L'_0 \vee \overline{f''}_0)(L_0 \vee \overline{L'}_0 \vee f''_0)(\overline{L_0} \vee L'_0 \vee f''_0) * (\overline{L}_0 \vee \overline{L'_0} \vee \overline{f''_0})\Big) = 1.$$

Более детальное рассмотрение примера до момента получения конечного результата является громоздким, так как требует описания процедур, связанных с итерационным вычислением выполнимости функции разрешения. Рассмотрение примера в статье направлено на то, чтобы показать основные этапы построения теста. Дадим интерпретацию ожидаемого результата.

На рис. 8 приведена ярусно-параллельная форма рассматриваемого на рис. 6 программного кода. В объекте имеется мультиплексор, два сумматора, умножитель и схема сравнения.

Рассматривается неисправность, которая приводит к появлению ошибочного результата $G' = B + C = 17$ ($B = 15$, $C = 1$) вместо $G = B + C = 16$ (табл. 1). Очевидно, что тестом должны быть такие значения входных для рассматриваемого кода переменных, т. е. A, B, C и S, при которых будет выполняться условие $L \neq L'$ или $F \neq F'$. При этом переменные B и C определены на этапе внесения неисправности. Поэтому будут вычислены значения переменных A и S. Для получения теста контроля объекта по выходу L необходимо определить такие значения параметров A и S, которые обеспечат выполнение условия $L \neq L'$.

Если $S = 1$, то для построения теста необходимо, чтобы выполнялись условия $(D = 15) < (G = 16)$ и $(D = 15) \geqslant (G' = 17)$, или $(D = 15) \geqslant (G = 16)$ и $(D = 15) < (G' = 17)$, которые являются конфликтными, что говорит о том, что неисправность по выходу L не определяется. Если $S = 0$, то для построения теста необходимо, чтобы выполнялись условия $(D = A) < (G = 16)$ и $(D = A) \geqslant (G' = 17)$ или $(D = A) \geqslant (G = 16)$ и $(D = A) < (G' = 17)$. Решением будет $A = 16$. Тестом явится $S = 0$, $A = 16$, $B = 15$, $C = 1$.

4. Верификация проектов на RTL-уровне

Верификация проектов, как известно, занимает большую часть времени проектирования сложно-функциональной СБИС. На сегодняшний день основными методами практической верификации на всех этапах проектирования является моделирование. Такой подход, основанный на моделировании, требует наличия тестов. Для моделирования объекта, описанного на языке VHDL на уровне RTL, предлагается метод, который заключается в следующем.

1. Представить программный код в виде системы арифметических уравнений.

2. Добавить в полученную систему уравнений означивание входо-выходных переменных.

3. Итеративно решить систему. Для этого перейти от системы арифметических уравнений к системе логических функций разрешения, учитывая особенность соответствующей итерации.

4. Получить КНФ-функцию разрешения и вычислить ее выполнимость. По результатам перейти к новому входо-выходному экземпляру теста или закончить моделирование и перейти к анализу результатов.

Рассмотрим моделирование объекта, описанного программным кодом, приведенным на рис. 6. Предположим, что известен тест, одним из шаблонов которого является $S = 0$, $A = 34$, $B = 45$, $C = 7$, $L = 1$, $F = 2132$. Система арифметических уравнений дополняется тестовыми значениями входно-выходных переменных и переводится в систему функций разрешения для проведения первого этапа вычислений, затем формируется КНФ – разрешения конъюнктивным объединением полученных функций и решается задача выполнимости. В табл. 5 приведены система арифметических уравнений и система логических функций разрешения для выполнения первой итерации вычислений.

Таблица 5. Функции разрешения для получения 1-го бита результата верификации

$D_0 = \overline{S_0} * A_0 + S_0 * B_0 \pmod 2$	$(S_0 \vee A_0 \vee \overline{D_0})(A_0 \vee B_0 \vee \overline{D_0})(\overline{S_0} \vee B_0 \vee \overline{D_0}) * (S_0 \vee \overline{A_0} \vee D_0)(\overline{S_0} \vee \overline{B_0} \vee D_0)(\overline{A_0} \vee \overline{B_0} \vee D_0)$
$G_0 = B_0 + C_0 \pmod 2$	$(C_0 \vee B_0 \vee \overline{G_0})(C_0 \vee \overline{B_0} \vee G_0)(\overline{C_0} \vee B_0 \vee G_0) * (\overline{C_0} \vee \overline{B_0} \vee \overline{G_0})$
$E_0 = D_0 + C_0 \pmod 2$	$(C_0 \vee D_0 \vee \overline{E_0})(C_0 \vee \overline{D_0} \vee E_0)(\overline{C_0} \vee D_0 \vee E_0) * (\overline{C_0} \vee \overline{D_0} \vee \overline{E_0})$
$F_0 = E_0 * G_0 \pmod 2$	$(E_0 \vee \overline{F_0})(G_0 \vee \overline{F_0})(\overline{E_0} \vee \overline{G_0} \vee F_0)$
$L_0 = D_0 < G_0 \pmod 2$	$(\overline{D_0} \vee \overline{L_0})(G_0 \vee \overline{L_0})(D_0 \vee \overline{G_0} \vee L_0)$
$A_0 = 0 \pmod 2$	$\overline{A_0}$
$B_0 = 1 \pmod 2$	B_0
$C_0 = 1 \pmod 2$	C_0
$L_0 = 1 \pmod 2$	L_0
$F_0 = 0 \pmod 2$	$\overline{F_0}$

5. Выводы

Предложен единый подход к решению задач направленного построения тестов и верификации проектов сложно-функциональных СБИС, представленных на уровне межрегистровых передач на языке VHDL, который является стандартом для описания электронных систем.

СПИСОК ЛИТЕРАТУРЫ

1. Electronic Design Automation: Synthesis, Verification, and Test / Edited by L.-T. Wang, Y.-W. Chang, K.-T. Cheng. Elsevier. 2009.
2. *Gharebaghi A.M., Navabi Z.* High-Level Test Generation from VHDL Behavioral Descriptions // Proc. VHDL Int. Users Forum Fall Workshop. Orlando, Florida. 18–20 October 2000. P. 123–126.
3. *Murray B.T., Hayes J.P.* Hierarchical Test Generation Using Precomputed Tests for Modules // Int. Test Conf. Washington. September 1988. P. 221–229.
4. *Goloubeva O., Sonza Reorda M., Violante M.* High-level test generation for hardware testing and software validation // Workshop of High-Level Design Validation and Test. San Francisco, California. 12–14 November 2003. P. 143–148.
5. *Золоторевич Л.А., Ильинкова А.В.* Разработка тестов для анализа контролепригодности СБИС на верхних уровнях проектирования // АиТ. 2010. № 9. С. 162–174.

   Zolotorevich L.A., Il'inkova A.V. Development of Tests for VLSI Circuit Testability at the Upper Design Levels // Autom. Remote Control. 2010. V. 71. No. 9. P. 1888–1898.
6. *Vallerio K.S., Jha N.K.* Task graph extraction for embedded system synthesis // Proc. IEEE Conf. VLSI Design. Portland, Oregon. 2003.
7. *Larrabee T.* Test pattern generation using Boolean satisfiability // IEEE Trans. Computer-Aided Design. 1992. V. 11. No. 1. P. 4–15.

Статья представлена к публикации членом редколлегии П.П. Пархоменко.

Поступила в редакцию 24.01.2012