УДК 519.873:519.718.7

Л.А. Золоторевич

ПОСТРОЕНИЕ ТЕСТОВ И ВЕРИФИКАЦИЯ ПОТОКОВЫХ МОДЕЛЕЙ ЦИФРОВЫХ УСТРОЙСТВ НА ЯЗЫКЕ VHDL

Предлагается единый подход к верификации проектов и направленному построению тестов контроля СБИС, представленных в потоковом виде на уровне RTL на языке VHDL с использованием арифметических, логических операторов и оператора If. Задача построения тестов и верификации проектов решается на основе КНФ-выполнимости некоторой системы булевых функций.

Введение

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 миллиарда 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 верхних уровней процесса проектирования, так как формальные методы верификации развиты недостаточно и верификация проектов на практике решается 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 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 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 работах [2–4]. В работе [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].

Общий подход к иерархической генерации тестов СБИС на RTL-уровне [5], основанный на том, что каждая операция текстового 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 направленной генерации тестов, основанный на построении системы арифметических уравнений и итерационном решении задачи КНФ-выполнимости некоторой системы булевых функций.

Проблема построения тестов непосредственно связана с проблемой верификации проектов сложнофункциональных 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. Метод исследован на примерах описаний с использованием логических операторов, операторов целочисленного сложения, умножения, назначения значений переменным и сигналам и оператора If.

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

Общая структура *DD* (decision diagrams) потока управления (рис. 1) 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 соответствует некоторому узлу *DD* потока управления. Узел имеет маркер, который представляется семеркой вида $<A_{stm}, A_{IF}, A_{Else}, A_{next}, P_{out}, P_{act}, P_t>$.

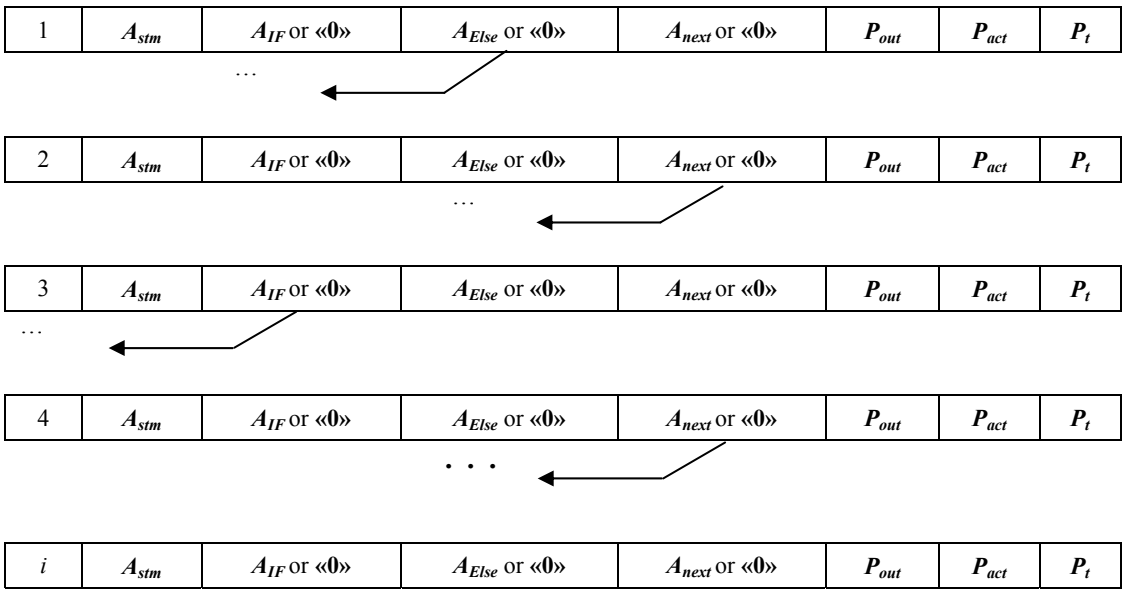

Рис. 1. Структура *DD* потока управления

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, то $A_{IF} = 0$, $A_{Else} = 0$, 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 изображена 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 последующий оператор с аналогичным идентификатором левого операнда (или нуль); Expression – вычисляемое арифметическое или логическое выражение.

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

На рис. 3 приведен некоторый набор операторов языка VHDL, который является модельным и не описывает реальный объект (после оператора 4, перед и после оператора 10 поставлены многоточия). Ниже рассмотрен пример построения *DD* потока управления (рис. 4) и потоков данных (рис. 5) для этого набора операторов.

```
entity Vonescnt is
     port  (X1, X2: in unsigned (7downto 0), rst, clk: in bit;
              Z1, Z2:  out  unsigned (7downto 0);
end Vonescnt;
architecture Vonescnt_arch of Vonescnt is
process (rst, clk)
variable Rab1, Rab2, Rab3, Rab4, Rab5: unsigned (7downto 0);
    begin
1       if (rst ='1') then
2           RAB1:=0;
3           RAB2:=0;
4       Elsif  (clk' event and clk=1) then
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. 3. Фрагмент VHDL-кода

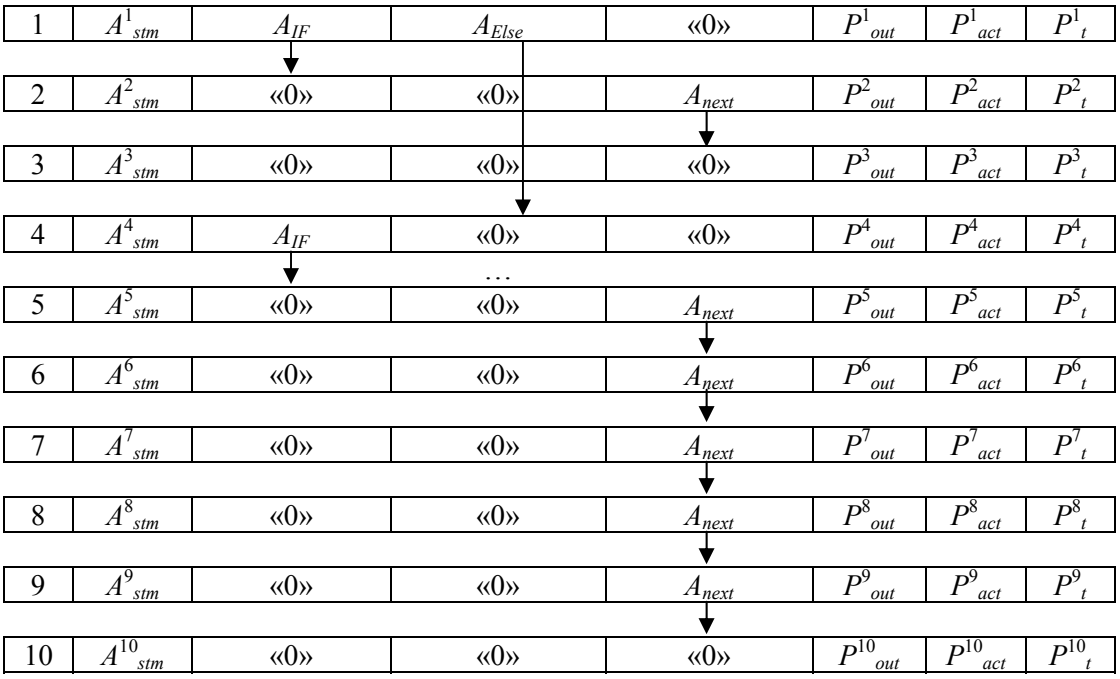

Рис. 4. Структура *DD* потока управления для примера на рис. 3

Stm	*DD*	A_{prvs}	A_{next}	Expression
Stm_1	*RST*	0	0	rst='1'
Stm_2	*RAB*1	0	A^5_{stm}	RAB1:=0;
Stm_3	*RAB*2	0	A^6_{stm}	RAB2:=0;
Stm_4		0	0	clk' event and clk=1
		…		
Stm_5	*RAB*1	A^2_{stm}	0	RAB1:=X1;
Stm_6	*RAB*2	A^3_{stm}	0	RAB2:=X2;
Stm_7	*RAB*3	0	0	RAB3:= RAB1+RAB2;
Stm_8	*RAB*4	0	0	RAB4:=RAB5;
Stm_9	*Z*1	0	0	Z1:=RAB4;
Stm_{10}	*Z*2	0	0	Z2:=RAB3;

Рис. 5. Структура *DD* потоков данных для примера на рис. 3

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 VHDL, то для направленного построения тестов необходимо знание внутреннего представления 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 кода описания объекта. 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; замена условного перехода безусловным, одной операции некоторой другой; неисправности константного типа некоторой переменной, сигнала, некоторого

разряда переменной, будет использоваться также 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. Эта гипотеза сформулирована на основе идеи Рота, реализованной при построении тестов контроля объекта на структурном уровне.

2. 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 построению системы 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:

– на основе VHDL-кода объекта составить систему арифметических уравнений, описывающих функционирование объекта;

– выполнить корректировку системы с учетом внесения неисправностей соответствующего оператора;

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

– итеративно выполнить решение системы. Для получения i-го бита результата арифметические выражения транслируются в КНФ булевых функций разрешения.

Все полученные КНФ-функции разрешения объединяются по правилу И, решается задача КНФ-выполнимости полученной системы булевых 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 VHDL-кода (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 показана система арифметических уравнений, которые описывают функционирование объекта, представленного VHDL-кодом (см. рис. 6). В объекте имеется мультиплексор, два сумматора, умножитель и схема сравнения. Построим тест, проверяющий правильность выполнения оператора целочисленного сложения $G = B + C$.

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

Рис. 6. Фрагмент VHDL-кода

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

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

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 на рис. 8, чтобы обеспечить распространение неисправности к выходам объекта и определение входных переменных. Здесь операторы 1–5 описывают исправный исходный объект, операторы 6–8 – процесс внесения неисправности, операторы 9–13 задают ограничения, обеспечивающие распространение эффекта неисправности к выходу объекта.

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

Рис. 8. Система арифметических уравнений (mod 2^n) с внесенной неисправностью

Идея решения подобных систем арифметических уравнений основана на поразрядном подходе к вычислению значений и состоит в том, что каждой целочисленной переменной $m <= 2^n$ ставится в соответствие определенный логический вектор размерности n. Для описания алгоритма вычисления системы 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 работе [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. 1).

Таблица 1

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

Получим СКНФ функции F^f по табл. 1, выбирая конституэнты 0. Для этого необходимо произвести следующие действия:

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

2) выписать 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;

3) все выписанные дизъюнкции соединить знаком конъюнкции:

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

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: $(a \vee b \vee \overline{f})(a \vee \overline{b} \vee f)(\overline{a} \vee b \vee f)(\overline{a} \vee \overline{b} \vee \overline{f})$.

Отметим, что описание объектов в виде КНФ функций разрешения и решение задачи SAT (выполнимости булевой функции) в последнее время достаточно широко используются в литературе при решении задач логического проектирования [8–12].

В табл. 2 приведены однобитовые операции, используемые в рассматриваемом на рис. 8 примере, и соответствующие функции разрешения в виде КНФ.

Таблица 2

Функции разрешения

Однобитовые арифметические и логические уравнения	КНФ функций разрешения
$f = b \vee c$	$(\overline{b} \vee f)(\overline{c} \vee f)(b \vee c \vee \overline{f})$
$f = b \times c$	$(b \vee \overline{f})(c \vee \overline{f})(\overline{b} \vee \overline{c} \vee f)$
$f = a \oplus b$	$(a \vee b \vee \overline{f})(a \vee \overline{b} \vee f)(\overline{a} \vee b \vee f)(\overline{a} \vee \overline{b} \vee \overline{f})$
$f = a \sim b$	$(a \vee b \vee f)(a \vee \overline{b} \vee \overline{f})(\overline{a} \vee b \vee \overline{f})(\overline{a} \vee \overline{b} \vee f)$
$f = \overline{s} \times a + s \times b$	$(s \vee a \vee \overline{f})(a \vee b \vee \overline{f})(\overline{s} \vee b \vee \overline{f}) \times$ $\times(s \vee \overline{a} \vee f)(\overline{s} \vee \overline{b} \vee f)(\overline{a} \vee \overline{b} \vee f)$
$f = a \geq b$	$(a \vee f)(\overline{a} \vee \overline{f})$
$f = b < c$	$(\overline{b} \vee \overline{f})(c \vee \overline{f})(b \vee \overline{c} \vee f)$
$f = 1$	f
$f = 0$	$\overline{f}$
$f \neq f'$	$(f \vee f' \vee \overline{f''})(f \vee \overline{f'} \vee f'') \times$ $\times(\overline{f} \vee f' \vee f'')(\overline{f} \vee \overline{f'} \vee \overline{f''})$

Решение системы уравнений, приведенной на рис. 8, выполняется итеративно. Чтобы одновременно решить равенства по модулю 2, объединяем все КНФ разрешения логических функций вместе, используя логическую функцию конъюнкции. Чтобы найти второй бит решения этого уравнения, необходимо найти рекурсивное уравнение по модулю 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. 3 приведены функции разрешения для первого бита системы арифметических уравнений (см. рис. 8).

Таблица 3

Функции разрешения для вычисления первого бита результата (f_0'' – результат предыдущей итерации)

Арифметические уравнения	Эквивалентные функции разрешения
$D_0 = \overline{S_0} * A_0 + S_0 * B_0 \pmod 2$	$(S_0 \vee A_0 \vee \overline{D_0})(A_0 \vee B_0 \vee \overline{D_0})(\overline{S_0} \vee B_0 \vee \overline{D_0}) \times$ $\times (S_0 \vee \overline{A_0} \vee D_0)(\overline{S_0} \vee \overline{B_0} \vee D_0)(\overline{A_0} \vee \overline{B_0} \vee D_0)$
$G_0 = B_0 + C_0 \pmod 2$	$(C_0 \vee B_0 \vee \overline{G_0})(C_0 \vee \overline{B_0} \vee G_0) \times$ $\times (\overline{C_0} \vee B_0 \vee G_0)(\overline{C_0} \vee \overline{B_0} \vee \overline{G_0})$
$E_0 = D_0 + C_0 \pmod 2$	$(C_0 \vee D_0 \vee \overline{E_0})(C_0 \vee \overline{D_0} \vee E_0) \times$ $\times (\overline{C_0} \vee D_0 \vee E_0)(\overline{C_0} \vee \overline{D_0} \vee \overline{E_0})$
$F_0 = E_0 * G_0 \pmod 2$	$(E_0 \vee \overline{F_0})(G_0 \vee \overline{F_0})(\overline{E_0} \vee \overline{G_0} \vee F_0)$
$L_0 = D_0 < G_0 \pmod 2$	$(\overline{D_0} \vee \overline{L_0})(G_0 \vee \overline{L}0)(D_0 \vee \overline{G_0} \vee L_0)$
$B_0 = 1 \pmod 2$	B_0
$C_0 = 1 \pmod 2$	C_0
$G_0^{'} = 1 \pmod 2$	$G_0^{'}$
$G_0^{'} \neq G_0 \pmod 2$	$(G_0 \vee G^{'}_0 \vee \overline{f_0''})(G_0 \vee \overline{G^{'}}_0 \vee f_0'') \times$ $\times (\overline{G_0} \vee G^{'}_0 \vee f_0'')(\overline{G_0} \vee \overline{G^{'}}_0 \vee \overline{f_0''})$
$F^{'}0 = E_0 * G^{'}0 \pmod 2$	$(E_0 \vee \overline{F^{'}0})(G_0 \vee \overline{F'0})(\overline{E}0 \vee \overline{G_0} \vee F^{'}0)$
$F_0 \neq F_0^{'} \pmod 2$	$(F_0 \vee F^{'}_0 \vee \overline{f''}_0)(F_0 \vee \overline{F^{'}_0} \vee f_0'') \times$ $\times (\overline{F_0} \vee F^{'}_0 \vee f_0'')(\overline{F_0} \vee \overline{F^{'}_0} \vee \overline{f_0''})$
$L^{'}0 = D_0 < G^{'}0 \pmod 2$	$(\overline{D}_0 \vee \overline{L^{'}_0})(G_0^{'} \vee \overline{L^{'}}_0)(D_0 \vee \overline{G_0^{'}} \vee L^{'}_0)$
$L_0 \neq L_0^{'} \pmod 2$	$(L_0 \vee L^{'}_0 \vee \overline{f_0''})(L_0 \vee \overline{L^{'}}0 \vee f_0'') \times$ $\times (\overline{L_0} \vee L^{'}_0 \vee f_0'')(\overline{L}_0 \vee \overline{L^{'}_0} \vee \overline{f_0''})$

Для рекурсивного вычисления неравенства необходимо учитывать, что оно 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:

$$((S_0 \vee A_0 \vee \overline{D_0})(A_0 \vee B_0 \vee \overline{D_0})(\overline{S_0} \vee B_0 \vee \overline{D_0}) * (S_0 \vee \overline{A_0} \vee D_0)(\overline{S_0} \vee \overline{B_0} \vee D_0)(\overline{A_0} \vee \overline{B_0} \vee D_0)) \times$$
$$\times ((C_0 \vee B_0 \vee \overline{G_0})(C_0 \vee \overline{B_0} \vee G_0)(\overline{C_0} \vee B_0 \vee G_0) * (\overline{C_0} \vee \overline{B_0} \vee \overline{G_0})) \times$$
$$\times ((C_0 \vee D_0 \vee \overline{E_0})(C_0 \vee \overline{D_0} \vee E_0)(\overline{C_0} \vee D_0 \vee E_0) * (\overline{C_0} \vee \overline{D_0} \vee \overline{E_0})) \times$$
$$\times ((E_0 \vee \overline{F_0})(G_0 \vee \overline{F_0})(\overline{E_0} \vee \overline{G_0} \vee F_0)) * ((\overline{D_0} \vee \overline{L_0})(G_0 \vee \overline{L}0)(D_0 \vee \overline{G_0} \vee L_0)) \times \mathrm{B}_0 \times \mathrm{C}_0 \times G_0^{'} \times$$
$$\times ((G_0 \vee G^{'}0 \vee \overline{f_0''})(G_0 \vee \overline{G^{'}}0 \vee f_0'')(\overline{G_0} \vee G^{'}0 \vee f_0'') * (\overline{G_0} \vee \overline{G^{'}}0 \vee \overline{f_0''})) \times$$
$$\times ((E_0 \vee \overline{F^{'}0})(G_0 \vee \overline{F'0})(\overline{E}0 \vee \overline{G_0} \vee F^{'}0)) \times$$
$$\times ((F_0 \vee F^{'}0 \vee \overline{f''}_0)(F_0 \vee \overline{F^{'}0} \vee f_0'')(\overline{F_0} \vee F^{'}0 \vee f_0'') * (\overline{F_0} \vee \overline{F^{'}0} \vee \overline{f_0''})) \times$$
$$\times ((\overline{D}_0 \vee \overline{L^{'}_0})(G_0^{'} \vee \overline{L^{'}}_0)(D_0 \vee \overline{G_0^{'}} \vee L^{'}_0)) \times$$
$$\times ((L_0 \vee L^{'}0 \vee \overline{f_0''})(L_0 \vee \overline{L^{'}}0 \vee f_0'')(\overline{L_0} \vee L^{'}_0 \vee f_0'') * (\overline{L}_0 \vee \overline{L^{'}_0} \vee \overline{f_0''})) = 1.$$

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.

Ярусно-параллельная форма показанного на рис. 6 VHDL-кода включает мультиплексор, два сумматора, умножитель и схему сравнения (рис. 9).

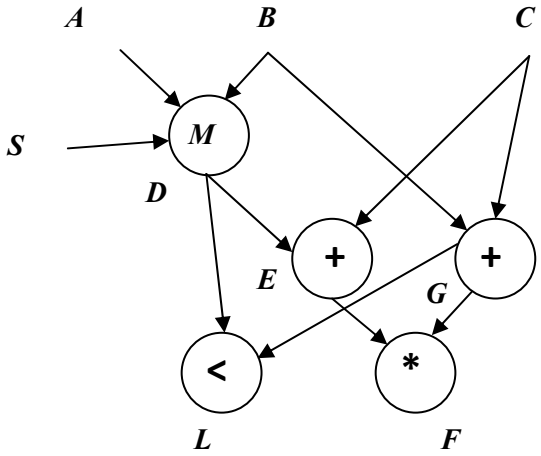

Рис. 9. Ярусно-параллельная форма представления кода

Рассматривается неисправность, которая приводит к появлению ошибочного результата $G' = B + C = 17$ ($B = 15$, $C = 1$) вместо $G = B + C = 16$ (см. рис 8). Очевидно, что тестом должны быть такие значения входных переменных для рассматриваемого кода, 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 $(D = 15) \geq (G' = 17)$ или $(D = 15) \geq (G = 16)$ и $(D = 15) < (G' = 17)$, которые являются конфликтными. Это 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 $(D = A) \geq (G' = 17)$ или $(D = A) \geq (G = 16)$ и $(D = A) < (G' = 17)$. Решением будет $A = 16$. Тестом явится $S = 0$, $A = 16$, $B = 15$, $C = 1$.

3. Верификация проектов на RTL-уровне

Верификация проектов, как известно, занимает большую часть времени проектирования сложнофункциональной СБИС. На сегодняшний день основным методом 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 VHDL-код в виде системы арифметических уравнений.
2. Добавить в полученную систему уравнений означивание входных и 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 входно-выходному экземпляру теста или закончить моделирование и перейти к анализу результатов.

Рассмотрим моделирование объекта, описанного VHDL-кодом, который приведен 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 входных и выходных переменных и переводится в систему функций разрешения для проведения первого этапа вычислений, затем формируется КНФ функции разрешения конъюнктивным объединением полученных функций и решается задача выполнимости. В табл. 4 приведены система арифметических уравнений и система логических функций разрешения для выполнения первой итерации вычислений.

Таблица 4

Функции разрешения для получения первого бита результата верификации

Арифметические уравнения	Эквивалентные функции разрешения
$D_0 = \overline{S_0} * A_0 + S_0 * B_0 \pmod 2$	$(S_0 \vee A_0 \vee \overline{D_0})(A_0 \vee B_0 \vee \overline{D_0})(\overline{S_0} \vee B_0 \vee \overline{D_0}) \times$ $\times(S_0 \vee \overline{A_0} \vee D_0)(\overline{S_0} \vee \overline{B_0} \vee D_0)(\overline{A_0} \vee \overline{B_0} \vee D_0)$
$G_0 = B_0 + C_0 \pmod 2$	$(C_0 \vee B_0 \vee \overline{G_0})(C_0 \vee \overline{B_0} \vee G_0) \times$ $\times(\overline{C_0} \vee B_0 \vee G_0)(\overline{C_0} \vee \overline{B_0} \vee \overline{G_0})$
$E_0 = D_0 + C_0 \pmod 2$	$(C_0 \vee D_0 \vee \overline{E_0})(C_0 \vee \overline{D_0} \vee E_0) \times$ $\times(\overline{C_0} \vee D_0 \vee E_0)(\overline{C_0} \vee \overline{D_0} \vee \overline{E_0})$
$F_0 = E_0 * G_0 \pmod 2$	$(E_0 \vee \overline{F_0})(G_0 \vee \overline{F_0})(\overline{E_0} \vee \overline{G_0} \vee F_0)$
$L_0 = D_0 < G_0 \pmod 2$	$(\overline{D_0} \vee \overline{L_0})(G_0 \vee \overline{L}_0)(D_0 \vee \overline{G_0} \vee L_0)$
$A_0 = 0 \pmod 2$	$\overline{A_0}$
$B_0 = 1 \pmod 2$	B_0
$C_0 = 1 \pmod 2$	C_0
$L_0 = 1 \pmod 2$	L_0
$F_0 = 0 \pmod 2$	$\overline{F_0}$

Заключение

Предложенный подход к направленному построению тестов и верификации ориентирован на проекты цифровых объектов, представленных потоковой моделью на уровне RTL на языке VHDL с использованием арифметических, логических операторов и оператора if-then-else. При переходе от RTL-уровня к структурному представлению объекта, когда объект рассматривается как сеть из функциональных блоков комбинационного и последовательностного типов, для построения тестов применяются другие подходы.

Список литературы

1. Electronic Design Automation: Synthesis, Verification, and Test / ed. L.-T. Wang, Y.-W. Chang, K.-T. Cheng. – Elsevier, 2009.

2. Gharebaghi, A.M. High-Level Test Generation from VHDL Behavioral Descriptions / A.M. Gharebaghi, Z. Navabi // Proc. of VHDL Intern. Users Forum Fall Workshop. – Orlando, Florida, 2000. – P. 123–126.

3. Murray, B.T. Hierarchical Test Generation Using Precomputed Tests for Modules / B.T. Murray, J.P. Hayes // Intern. Test Conf. – Washington, 1988. – P. 221–229.

4. Goloubeva O. High-level test generation for hardware testing and software validation / O. Goloubeva, M. Sonza Reorda, M. Violante // Workshop of High-Level Design Validation and Test. – San Francisco, California, 2003. – P. 143–148.

5. Zolotorevich, L.A. Development of tests for VLSI circuit testability at the upper design levels / L.A. Zolotorevich, A.V. Il'inkova // Automation and Remote Control. – 2010. – Vol. 71, iss. 9. – P. 1888–1898.

6. Vallerio, K.S. Task graph extraction for embedded system synthesis / K.S. Vallerio, N.K. Jha // Proc. of IEEE Conf. on VLSI Design. – Portland, Oregon, 2003.

7. Larrabee, T. Test pattern generation using Boolean satisfiability / T. Larrabee // IEEE Trans. Computer-Aided Design. – 1992. – Vol. 11, № 1. – P. 4–15.

8. Новиков, Д.Я. Верификация функциональных описаний с неопределенностью на основе парафазного представления булевых функций / Д.Я. Новиков, Л.Д. Черемисинова // Информатика. – 2010. – № 3. – С. 54–62.

9. Test Pattern Generation using Boolean Proof Engines / R. Drechsler [et al.]. – Springer, Dordrecht, Heidelberg, London, New York, 2009.

10. Automatic Constraint Based Test Generation for Behavioral HDL Models / S.K. Hari [et al.] // IEEE Trans. on VLSI systems. – 2008. – Vol. 16, № 4. – P. 408–421.

11. Alizadeh, B. High level test generation without ILP and SAT Solvers / B. Alizadeh, M. Fujita // Int. Workshop on High Level Design Validation and Testing (HLDVT07). – Irvin, Ca, 2007. – P. 298–304.

12. Koo, H.-M. Functional Test Generation Using Design and Property Decomposition Techniques / H.-M. Koo, P. Mishra // ACM Transactions on Embedded Computing Systems. – 2009. – Vol. 8, № 4. – Article 32. – P. 1–32.

Поступила 16.03.12

Белорусский государственный университет,
Минск, пр. Независимости, 4
e-mail: zolotorevichLA@bsu.by

L.A. Zolotorevich

CONSTRUCTION OF TESTS AND VERIFICATION OF DIGITAL DEVICES FLOW MODELS ON VHDL LANGUAGE

A unified approach to verification of projects and directed construction of VLSI tests control presented at the RTL level in language VHDL is offered. The problem of tests construction and verifications of projects is solved via CNF-satisfiability of some system of Boolean functions.