УДК 004.3

П.Н. Бибило, В.И. Романов

Построение компактных тестов для функциональной верификации *VHDL*-описаний конечных автоматов

Разработана программа, позволяющая по полученной в результате моделирования последовательности состояний автомата строить ориентированный граф переходов автомата и находить покрытие всех дуг. Входные тестовые наборы, соответствующие дугам, вошедшим в покрытие, будут образовывать тест для функциональной верификации.

Розроблено програму, яка дозволяє за отриманою в результаті моделювання послідовністю станів автомата будувати орієнтовний граф переходів автомата і знаходити покриття всіх дуг. Вхідні тестові набори, відповідні дугам, які увійшли в покриття, створюватимуть тест для функціональної верифікації.

Введение. При проектировании цифровой аппаратуры широкое распространение получила модель цифрового устройства с памятью в виде конечного автомата, в зарубежной литературе называемая *FSM* (*Finite State Machine*). Конечные автоматы, как и другие модели цифровых устройств, представляются на языках *VHDL* [1], *Verilog* [2], предназначенных для проектирования цифровых схем на современной базе заказных СБИС (сверхбольших интегральных схем) либо программируемых пользователями логических интегральных схем типа *FPGA*. По *VHDL*-описаниям конечных автоматов синтезируются синхронные логические схемы в том или ином базисе логических элементов, называемом технологическим (целевым) базисом либо целевой библиотекой логических элементов. Сегодня процесс синтеза автоматизирован и важнейшей проблемой при создании проектов СБИС и систем-на-кристалле [3] – проблема верификации исходных спецификаций, представленных на *VHDL* либо других языках, используемых для алгоритмического описания проектируемых цифровых устройств и систем. Под *верификацией* понимается проверка правильности исходного *VHDL*-описания, т.е. проверка соответствия составленного синтезируемого *VHDL*-описания проектируемой цифровой системы спецификациям на проектирование [4].

Большое достоинство модели конечного автомата – данная модель может быть верифицирована [4]. Система моделирования *Questa* [5] имеет в своем составе средства для функциональной верификации *FSM*, если модель *FSM* написана соответствующим образом. Данные средства (опции) позволяют при моделировании распознавать конечный автомат, входящий в состав проекта, определить все пройденные (в конкретном сеансе моделирования) состояния конечного автомата и подсчитать число прохождений дуг в графе переходов автомата. Такие средства весьма полезны, однако конечные автоматы, как правило, входят в качестве управляющих блоков в состав более сложных проектов. Для проведения верификации всего проекта в целом требуется построение компактных функциональных тестов для управляющего блока – конечного автомата. Автоматизированное построение таких тестов по результатам моделирования *VHDL*-описаний конечных автоматов – цель данной статьи.

Постановка задачи и предлагаемый подход

Модель конечного автомата и его *VHDL*-описание назовем *корректными*, если из любого внутреннего состояния автомата имеется путь в начальное состояние автомата. Отметим, что для микропрограммных автоматов [6] данное условие – обязательно и всегда выполняется. Микропрограммные автоматы давно

используются в практике проектирования и описываются в настоящее время на современных языках [7, 8].

Под *тестом* понимается упорядоченная последовательность наборов значений входных сигналов, которые при моделировании подаются на входные порты *VHDL*-модели автомата. Для проведения моделирования готовится специальная тестирующая программа (*test-bench*) либо множество таких программ, ориентированных на различные цели тестирования и способы организации тестов. Тесты, позволяющие проверить правильность функционирования исследуемой модели автомата, называются *функциональными*.

Задача 1. Задано корректное *VHDL*-описание конечного автомата. Требуется построить тест $T_{\text{рез}}$, позволяющий при моделировании (данного автомата на наборах теста $T_{\text{рез}}$) выполнить *функциональную верификацию* – проверить выполнение всех имеющихся в модели автомата переходов между внутренними состояниями.

Например, в листинге 1 задано *VHDL*-описание конечного автомата *Mealy*. Отметим, что, кроме переходов $s_i \rightarrow s_j$ (смены внутреннего состояния) по переднему фронту синхросигнала *clk*, имеются также переходы в начальное состояние s_0 по единичному значению сигнала *rst*. Для краткости назовем первые из них *синхронными* и, соответственно, вторые – *асинхронными*. Под *состояниями* автомата всегда будут пониматься его внутренние состояния. Точнее, в качестве автоматов будут рассматриваться конечные автоматы с закодированными входными и выходными состояниями, такие конечные автоматы в научной литературе называются автоматами с *абстрактными* состояниями [9].

На рис. 1 показан граф $G^E(S^E, A^E)$ автомата *Mealy*, соответствующий *VHDL*-модели, представленной в листинге 1. Назовем этот граф *эталонным*. Эталонный граф $G^E(S^E, A^E)$ описывает только функцию переходов между состояниями автомата. На рис. 1 не показаны асинхронные переходы (рис. 2) из любого состояния s_i в начальное состояние s_0 при единичном значении сигнала *rst*. Дизъюнктивные нормальные формы (ДНФ) D_{ij}, описывающие условия переходов $s_i \rightarrow s_j$ между состояниями s_i, s_j, и значения выходных сигналов автомата при выполнении таких переходов, заданы в табл. 1.

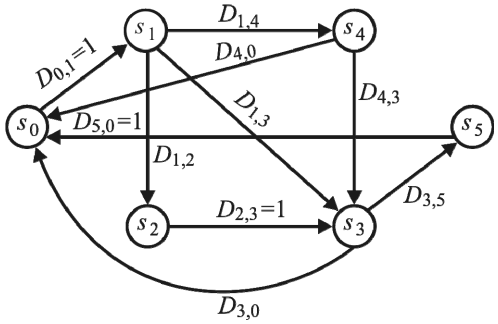

Рис. 1. Эталонный граф $G^E(S^E, A^E)$ переходов автомата *Mealy*

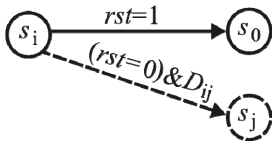

Рис. 2. Асинхронные переходы в начальное состояние s_0 в эталонном графе $G^E(S^E, A^E)$

Таблица 1. Таблица переходов автомата *Mealy*

s_i	s_j	Условия перехода	Выходные сигналы
s_0	s_1	$D_{0,1}=1$;	y_2
s_1	s_2	$D_{1,2}=x_1\bar{x}_2\bar{x}_3 \vee x_1x_2$	y_3
	s_3	$D_{1,3}=x_1\bar{x}_2x_3$	y_4
	s_4	$D_{1,4}=\bar{x}_1$	y_5
s_2	s_3	$D_{2,3}=1$	y_4
s_3	s_0	$D_{3,0}=\bar{x}_2$	y_1
	s_5	$D_{3,5}=x_2$	y_6
s_4	s_0	$D_{4,0}=\bar{x}_1x_4$	y_1
	s_3	$D_{4,3}=\bar{x}_4 \vee x_1x_4$	y_4
s_5	s_0	$D_{5,0}=1$	y_1

Далее будут рассматриваться примеры конечных автоматов на языке *VHDL*, однако, предлагаемый подход к построению тестов может быть применим для описаний автоматов на языке *Verilog*.

Листинг 1. *VHDL*-описание конечного автомата *Mealy*

```
library ieee;
use ieee.std_logic_1164.all;
entity Mealy is
port(        x1, x2, x3, x4 : in std_logic;
                    clk, rst : in std_logic;
     y1, y2, y3, y4, y5, y6 : out std_logic);
end Mealy;
architecture rtl of Mealy is
type T_state is (s0, s1, s2, s3, s4, s5);
signal NEXT_state, state: T_state;
signal w: std_logic_vector (1 to 6);
   begin
y1 <= w(1); y2 <= w(2);y3 <= w(3);y4 <= w(4);y5 <= w(5);
y6 <= w(6);
NS : process (state, x1, x2, x3, x4)
   begin
case state is
when s0 =>
           NEXT_state <= s1; w <="010000";
when s1 => if ( (x1 and not x2 and not x3) or (x1 and x2) ) = '1'
                 then NEXT_state <= s2; w <="001000";
           elsif ((x1 and not x2 and x3) = '1')
               then NEXT_state <= s3; w <="000100";
           elsif (not x1 = '1')
               then NEXT_state <= s4; w <="000010";
           end if;
when s2 => NEXT_state <= s3; w <="000100";
when s3 => if (not x2 = '1')
               then NEXT_state <= s0 ; w <="100000";
           elsif ( x2  = '1')
               then NEXT_state <= s5 ; w <="000001";
           end if;
when s4 => if ((not x1 and x4 ) = '1')
               then NEXT_state <= s0; w <="100000";
           elsif ( (not x4) or (x1 and x4) ) = '1'
                 then NEXT_state <= s3; w <="000100";
           end if;
when s5 => NEXT_state <= s0; w <="100000";
end case;
end process NS;
state_process: process (clk, rst)
   begin  -- process state_process
    if rst = '1' then      -- asynchronous reset (active low)
      state <= s0;
    elsif clk'event and clk = '1' then  -- rising clock edge
      state <= NEXT_state;
    end if;
  end process state_process;
end rtl;
```

Выделение из *VHDL*-описания автомата внутренних состояний автомата и построение графа $G^E(S^E, A^E)$ переходов – нетривиальная задача, так как сводится к анализу *VHDL*-кода, синтаксис которого сложный, а автомат может быть задан в другой форме, отличающейся от формы описания, представленной в листинге 1, и тогда система моделирования *Questa* не сможет его распознать. По сути, надо автоматизировать процесс построения математической модели графа переходов по *VHDL*-программе, задающей автомат, например, построить матрицу смежности ориентированного графа переходов. Чтобы избежать такого анализа, для решения задачи 1 предлагается подход, позволяющий получить приближенное решение и базирующийся на:

- моделировании *VHDL*-описания автомата на псевдослучайном тесте $T_{\text{исх}}$;
- выделении из теста $T_{\text{исх}}$ некоторых двоичных тестовых наборов, которые включаются в искомый тест $T_{\text{рез}}$.

Для того чтобы реализовать данный подход, надо правильно организовать моделирование, а именно: для каждого тестового набора (каждого такта моделирования) выдать внутреннее состояние автомата, в которое переходит автомат при подаче тестового набора на вход *VHDL*-модели автомата и в котором он будет находиться в следующем такте моделирования. Тестирующая программа для начального моделирования при фиксированном значении $rst = 0$ представлена в листинге 2.

Обратим внимание на то, что для доступа к внутренним сигналам проекта

```
state_top <= <<signal .tstb.p0.state :
                T_state>>;
```

требуется моделирование в системе *Questa* [5] с включенной опцией «Стандарт *VHDL*'2008» [10].

Проведем моделирование *VHDL*-описания автомата на 30 псевдослучайных наборах, задаваемых в файле *IN.TST* (табл. 2). Результатом моделирования будут потактовые выходные реакции автомата (текстовый файл *OUT.TST*) и текстовый файл *STATE.TST*, в котором задается последовательность пройденных состояний автомата. В табл. 2 приводится файл *IN.TST* тестовых наборов и файл *STATE.TST*. При моделировании осуществляются циклические попадания в начальное состояние автомата s_0, семь таких циклов приве-

Листинг 2. Тестирующая программа для моделирования *VHDL*-описания конечного автомата *Mealy* (*rst* = 0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use STD.TEXTIO.all;
use STD.ENV.all;
use IEEE.std_logic_TEXTIO.all;
use ieee.numeric_std.all;
entity tstb is    end;
architecture BEHAVIOR of tstb is
component Mealy is
port(        x1, x2, x3, x4 : in std_logic;
                  clk, rst : in std_logic;
          y1, y2, y3, y4, y5, y6 : out std_logic);
end component  ;
signal x1, x2, x3, x4 :  std_logic;
signal clk, rst :  std_logic;
signal y1, y2, y3, y4, y5, y6 :  std_logic;

signal W: std_logic_vector (4 downto 1);
signal DATA : std_logic_vector (4 downto 1);
signal YY : STD_LOGIC_VECTOR (6 downto 1);
type T_state is (s0, s1, s2, s3, s4, s5);
signal state_top : T_state;
begin
p0: Mealy
   port map (  x1, x2, x3, x4, clk, rst,  y1, y2,
                                  y3, y4, y5, y6);
state_top <= <<signal .tstb.p0.state : T_state>>;
W <=  DATA;
x1 <= W(4); x2 <= W(3); x3 <= W(2); x4 <= W(1);
YY <= (y1, y2, y3, y4, y5, y6);

Process
file INFILE,OUTFILE, OUTSTATE : text;
variable PTR,POKE, PRSTATE :line;
variable DATA_IN: std_logic_vector (4 downto 1);
variable DATA_OUT: std_logic_vector (6 downto 1);
variable STATE_OUT: line;
variable var_ST: T_state;
begin
file_open(INFILE,"IN.TST",read_mode);
file_open(OUTFILE,"OUT.TST",write_mode);
file_open(OUTSTATE,"STATE.TST",write_mode);
    rst <='1';
    clk <= '0';
    wait for   50.0 ns;
    clk <= '1';
    wait for   50.0 ns;
    clk <= '0';
    wait for   25.0 ns;
    rst <='0';
    while not (endfile(INFILE)) loop
     clk <= '0';
    wait for   25.0 ns;
         readline(INFILE,PTR);
         read(PTR,DATA_IN);
         DATA <= DATA_IN;
    wait for   25.0 ns;
    clk <= '1';
    wait for   50.0 ns;
         DATA_OUT := YY;
         write(POKE,DATA_OUT);
         writeline(OUTFILE,POKE);
    var_ST := state_top;
         write(PRSTATE, to_string(var_ST));
         writeline(OUTSTATE,PRSTATE);
    end loop;
file_close(INFILE);
file_close(OUTFILE);
file_close(OUTSTATE);
report "Done!" severity WARNING;
wait;
end process;
end;
```

дены в табл. 2, восьмой цикл не закончен. Будем считать, что в нулевом (начальном) такте моделирования автомат всегда находится в начальном состоянии, куда он попадает по *VHDL*-команде *rst* <='1' ; (выделенная жирным шрифтом строка в листинге 2).

Таблица 2. Результат моделирования *VHDL*-модели автомата *Mealy*

Такт	Тест $T_{исх}$ (*IN.TST*) $x_1x_2x_3x_4$	Состояния $S_{исх}$ (*STATE.TST*)	Циклы на графике *G* переходов							
			1	2	3	4	5	6	7	8
0		*s*0	*s*0							
1	0101	*s*1	*s*1							
2	1001	*s*2	*s*2							
3	0001	*s*3	*s*3							
4	1011	*s*0	*s*0	*s*0						
5	0100	*s*1		*s*1						
6	1000	*s*2		*s*2						
7	1000	*s*3		*s*3						
8	1110	*s*5		*s*5						
9	1000	*s*0		*s*0	*s*0					
10	1110	*s*1			*s*1					
11	0011	*s*4			*s*4					
12	1101	*s*3			*s*3					
13	0110	*s*5			*s*5					
14	1100	*s*0			*s*0	*s*0				
15	0111	*s*1				*s*1				
16	0011	*s*4				*s*4				
17	0011	*s*0				*s*0	*s*0			
18	0110	*s*1					*s*1			
19	0101	*s*4					*s*4			
20	1011	*s*3					*s*3			
21	0010	*s*0					*s*0	*s*0		
22	1100	*s*1						*s*1		
23	0111	*s*4						*s*4		
24	0001	*s*0						*s*0	*s*0	
25	0000	*s*1							*s*1	
26	0101	*s*4							*s*4	
27	0000	*s*3							*s*3	
28	1001	*s*0							*s*0	*s*0
29	0111	*s*1								*s*1
30	1110	*s*2								*s*2

По результатам моделирования автомата (файлам *IN.TST, STATE.TST*) строим граф $G(S, A)$, вершинами которого есть состояния автомата, а ребрами – переходы между состояниями. В файле *IN.TST* задаются тестирующие наборы – это последовательность $T_{исх}$, файл *STATE.TST* представляет собой последовательность $S_{исх}$ состояний автомата, в которые автомат переходит при подаче тестовых наборов из файла *IN.TST*. Обозначим через $x = (x_1, x_2, \ldots, x_n)$ входные информационные сигналы автомата (в этот список не включаются синхросигнал сигнал *clk* (тактирования) и сигнал *rst* (асинхронного сброса в начальное состояние).

Через V^x обозначим булево пространство над переменными вектора $x = (x_1, x_2, \ldots, x_n)$. Булево пространство V^x содержит 2^n двоичных наборов x_i^*. В рассматриваемом примере $n = 4$, $x = (x_1, x_3, x_4)$, а число всех возможных значений вектора x равно 16. Если при моделировании автомат из состояния s_i перешел в состояние s_j при подаче на его вход входного набора x_q^*, то в ориентированный граф $G(S, A)$ вносится дуга $s_i \to s_j$ и помечается набором x_q^*. В данном графе S – множество вершин графа (каждой вершине графа соответствует состояние автомата), A – множество дуг $s_i \to s_j$. Последовательность $x_{q1}^*, x_{q2}^*, \ldots, x_{qk}^*$ входных наборов – это наборы теста $T_{\text{исх}}$. Последовательность состояний $s_1, s_{i1}, s_{i2}, \ldots, s_{ik}$ обозначим через $S_{\text{исх}}$. Множество S состоит из символов *различных* состояний, имеющихся в последовательности $S_{\text{исх}}$. В рассматриваемом примере (табл. 2) переход автомата из состояния s_0 в состояние s_1 осуществляется при подаче на вход автомата двоичного набора $x_{q1}^* = (0101)$, переход автомата из состояния s_1 в состояние s_2 осуществляется при подаче на вход автомата двоичного набора $x_{q2}^* = (1001)$ и т.д.

Задача 2. По заданным последовательностям $T_{\text{исх}}$, $S_{\text{исх}}$ построить ориентированный граф $G = (S, A)$, вершинам которого соответствуют состояния s_i, а каждая дуга $s_i \to s_j$ которого помечена одним двоичным набором, вызывающим переход из состояния s_i в состояние s_j.

Решение задачи 2 не вызывает затруднения, при этом метка дуги устанавливается равной первому встреченному входному набору, вызвавшему соответствующий переход. В рассматриваемом примере результатом решения задачи 2 есть граф $G = (S, A)$, изображенный на рис. 3.

Отметим, что если в последовательности $S_{\text{исх}}$ нет соседних элементов s_i, s_j, то в графе $G = (S, A)$ будет отсутствовать дуга $s_i \to s_j$. В примере последовательность S не содержит соседних состояний s_1, s_3, поэтому в графе $G = (S, A)$ отсутствует дуга $s_1 \to s_3$. На рис. 3 отсутствующая дуга показана штриховой линией. Таким образом, требуется, чтобы начальный тест $T_{\text{исх}}$ был достаточно длинным и обеспечил наличие в графе $G = (S, A)$ всех переходов автомата, которые есть в *VHDL*-модели этого автомата. В примере тест $T_{\text{исх}}$ из 30 наборов короткий и не обеспечивает наличия в графе всех требуемых дуг. Требуется более длинный тест, как будет показано далее, для построения графа $G = (S, A)$, содержащего все требуемые дуги, достаточно псевдослучайного теста из 80 наборов. Цель начального моделирования будет достигнута, если граф $G(S, A)$ будет равен графу $G^E(S^E, A^E)$ (без учета пометок на ориентированных дугах данных графов).

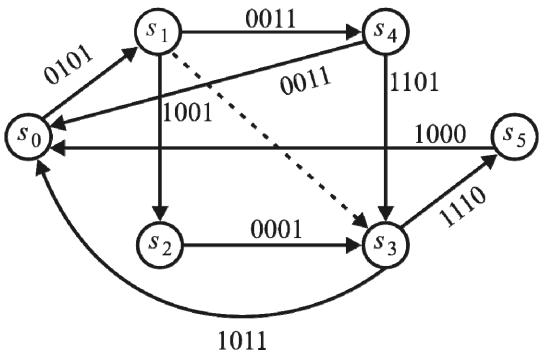

Рис. 3. Граф $G = (S, A)$, построенный по результатам начального моделирования на 30 наборах

С использованием графа $G = (S, A)$ задача 1 сводится к задаче 3.

Задача 3. Для заданного ориентированного графа $G = (S, A)$ найти минимальный по длине цикл, содержащий все дуги графа.

Данная задача – известный случай задачи о китайском почтальоне для ориентированных графов [11, 12].

Среди вершин S графа в качестве *целевой* выделим вершину s_0, соответствующую начальному (конечному) состоянию автомата. Искомый цикл может быть представлен набором контуров, содержащих указанную вершину s_0. Имея в распоряжении найденный цикл, нетрудно построить тест $T_{\text{рез}}$, обеспечивающий проверку *VHDL*-модели автомата, а после

синтеза и логической схемы, реализующей *VHDL*-модель.

Приближенный алгоритм решения задачи 3

Исходные данные – матрица смежности графа $G = (S, A)$, при этом каждая дуга имеет метку, представленную тестовым набором, породившим соответствующий переход между вершинами.

Ш а г 1. Реализуется вычислительная процедура, основанная на рекурсии и обеспечивающая нахождение кратчайших путей из каждой вершины графа G в целевую вершину.

Ш а г 2. Реализуется рекурсивная процедура нахождения в графе G циклов, начинающихся с исходной вершины и в совокупности содержащих все дуги графа. В качестве аргументов этой процедуры выступает номер рассматриваемой вершины, текущее состояние матрицы смежности и формируемый цикл, представленный последовательностью пройденных вершин графа. Кроме того, в процедуре используется глобально доступная матрица пока *непокрытых* дуг (в начальный момент – копия матрицы смежности). Непокрытой считается та дуга, которая не входит ни в один из уже построенных циклов.

Ш а г 2.1. Последовательно рассматриваются все дуги, инцидентные исследуемой вершине. Если такая дуга находится, то она удаляется из графа и матрицы непокрытых дуг. После этого реализуется очередной шаг рекурсии с определением концевой вершины дуги в качестве исследуемой.

Ш а г 2.2. По завершению перебора дуг, инцидентных исследуемой вершине, строится результирующий путь. Для этого его начальное состояние, переданное в качестве аргумента, пополняется кратчайшим путем из исследуемой вершины графа в целевую вершину.

Ш а г 2.3. Результирующий путь проверяется на *покрываемость* ранее сохраненными путями: если среди ранее найденных путей не находится такой, что содержит все дуги нового результирующего пути, то этот путь дополняет набор найденных. После пополнения проверяется наличие еще непокрытых дуг по глобальной матрице непокрытых дуг. Если еще остаются претенденты, то завершается текущий шаг рекурсии. При их отсутствии – завершается вся процедура поиска.

Ш а г 3. Найденное множество путей обычно не является минимальным по количеству содержащих в себе дуг и может быть подвергнуто следующей цепочке оптимизирующих преобразований, улучшающих найденное решение.

Ш а г 4. Множество путей упорядочивается в порядке убывания их длин.

Ш а г 5. Реализуется процедура разложения найденных путей на *простые* циклы – каждый такой путь содержит только одно вхождение целевой вершины – путь задается последовательностью вершин, которая при описании начинается и завершается номером целевой вершины. При реализации процедуры контролируется уникальность вносимых в множество простых путей – повторное внесение в результирующее множество ранее определенного пути не проводится. Например, цикл

$$0 \to 1 \to 0 \to 2 \to 3 \to 4 \to 0 \to 5 \to 0 \to 6 \to \\ \to 1 \to 6 \to 2 \to 6 \to 5 \to 4 \to 6 \to 1 \to 0$$

разложится на простые циклы:

$$0 \to 1 \to 0,$$
$$0 \to 2 \to 3 \to 4 \to 0,$$
$$0 \to 5 \to 0,$$
$$0 \to 6 \to 1 \to 6 \to 2 \to 6 \to 5 \to 4 \to 6 \to 1 \to 0.$$

В данном примере 0, 1, 2, 3, 4, 5, 6 – номера вершин, целевая (начальная) вершина имеет номер 0.

Оценка решения улучшается путем ликвидации повторов на множестве циклов.

Ш а г 6. Из путей временно исключаются присутствующие в ней петли.

Ш а г 7. Множество циклов упорядочивается в порядке убывания их длин.

Ш а г 8. Реализуется процедура исключения покрываемых циклов, при которой из множества циклов удаляются те, все дуги которых присутствуют в каком-то другом цикле из оставшегося множества. Например, при наличии во множестве пары путей $0 \to 6 \to 1 \to 5 \to 4 \to \\ \to 5 \to 0$ и $0 \to 6 \to 1 \to 5 \to 0$ после реализации процедуры останется только первый из них.

Ш а г 9. Реализуется процедура исключения повторения вложенных циклов. В рамках этой процедуры последовательно перебираются все циклы результирующего множества. В каждом цикле ищутся вложенные циклы и после обнаружения повторы таких же циклов ликвидируются для всего множества путей. Например, при наличии пары путей $0\to6\to2\to3\to4\to\to5\to4\to6\to5\to0$ и $0\to2\to3\to4\to5\to\to4\to6\to5\to0$ после реализации процедуры второй путь станет короче – $0\to2\to3\to4\to\to6\to5\to0$. В связи с возможностью наличия иерархической вложенности циклов процедура может оказаться полезной при многократном применении.

Ш а г 10. Множество циклов сортируется по значениям и из него исключаются прямые повторы, которые могут там появиться в результате ранее выполненных преобразований.

Ш а г 11. Множество циклов упорядочивается в порядке убывания их длин.

Ш а г 12. Реализуется процедура исключения контуров, покрываемых совокупностью других оставшихся путей исследуемого множества.

Ш а г 13. Реализуется восстановление ранее удаленных путей–петель.

Ш а г 14. Конец.

В рассмотренном примере в результате выполнения шага 2 находятся четыре цикла.

Цикл 1: $s_0 \to s_1 \to s_2 \to s_3 \to s_5 \to s_0$.

Цикл 2: $s_0 \to s_1 \to s_4 \to s_3 \to s_0$.

Цикл 3: $s_0 \to s_1 \to s_2 \to s_3 \to s_0$.

Цикл 4: $s_0 \to s_1 \to s_4 \to s_0$.

Ликвидация повторов дуг в данном простом примере приводит к удалению третьего цикла, так как все его дуги покрываются дугами циклов 1, 2: дуги $s_0\to s_1$, $s_1\to s_2$, $s_2\to s_3$ цикла 3 входят в цикл 1, дуга $s_3 \to s_0$ входит в цикл 2. По оставшимся циклам 1, 2, 4 легко строится функциональный тест (табл. 3) по всем синхронным переходам.

В данном случае при начальном моделировании на 30 наборах ни разу не был выполнен переход $s_1\to s_3$, в результирующем тесте его тоже нет. Если провести начальное моделирование на 80 наборах, то можно получить полный функциональный тест (табл. 4) для синхронных переходов. Всего в графе десять синхронных и шесть асинхронных переходов (из каждого состояния в состояние s_0). Тест из 16 наборов позволяет проверить все десять синхронных переходов. Таким образом, по длинным входным тестам начального моделирования моделируются все переходы автомата, алгоритм построения функционального теста может построить полный функциональный тест. Если же задать короткий для начального моделирования, то не все переходы между состояниями автомата будут промоделированы и полный функциональный тест (по всем переходам) не будет построен.

Таблица 3. Неполный функциональный тест, полученный по результатам моделирования на 30 наборах

Такт	Тест $T_{рез}$ $x_1\,x_2\,x_3\,x_4$	Состояния $S_{рез}$
0		$s0$
1	0101	$s1$
2	1001	$s2$
3	0001	$s3$
4	1110	$s5$
5	1000	$s0$
6	0101	$s1$
7	0011	$s4$
8	1101	$s3$
9	1011	$s0$
10	0101	$s1$
11	0011	$s4$
12	0011	$s0$

Таблица 4. Полный функциональный тест, полученный по результатам моделирования на 80 наборах

Такт	Тест $T_{рез}$ $x_1\,x_2\,x_3\,x_4$	Состояния $S_{рез}$
0		$s0$
1	0101	$s1$
2	1001	$s2$
3	0001	$s3$
4	1110	$s5$
5	1000	$s0$
6	0101	$s1$
7	0011	$s4$
8	1101	$s3$
9	1011	$s0$
10	0101	$s1$
11	1010	$s3$
12	1110	$s5$
13	1000	$s0$
14	0101	$s1$
15	0011	$s4$
16	0011	$s0$

Пример 2. На рис. 4 задан граф $G = (S, A)$ автомата *dk*14 – одного из тестовых примеров (*benchmarks*) конечных автоматов [13]. На данном примере можно проиллюстрировать те шаги алгоритма, которые пропущены при рассмотрении предыдущего примера.

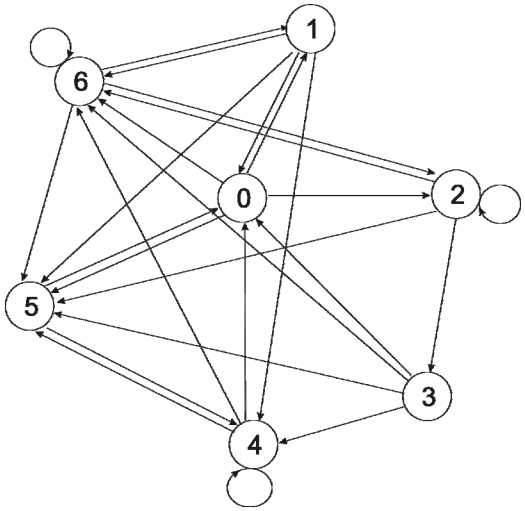

Рис. 4. Граф для синхронных переходов автомата *dk*14, начальная вершина 0

Проведем моделирование *VHDL*-описания автомата *dk*14 на 512 наборах значений входных сигналов. В процессе выполнения программы, решающей задачи 2 и 3, получим граф $G = (S, A)$, в котором будет 26 дуг. Граф $G = (S, A)$ содержит 71 цикл, в которые входит 1241 дуга. Простых циклов 37, в которые входят 249 дуг. В результате решения задачи 3 получим следующие девять циклов:

$0 \to 6 \to 6 \to 1 \to 4 \to 4 \to 5 \to 4 \to 6 \to 2 \to 2 \to$
$\to 6 \to 5 \to 0;$

$0 \to 6 \to 1 \to 6 \to 5 \to 4 \to 6 \to 1 \to 0;$

$0 \to 6 \to 1 \to 5 \to 4 \to 6 \to 1 \to 0;$

$0 \to 5 \to 4 \to 6 \to 2 \to 5 \to 0;$

$0 \to 2 \to 3 \to 6 \to 1 \to 4 \to 0;$

$0 \to 2 \to 3 \to 4 \to 5 \to 0;$

$0 \to 2 \to 3 \to 5 \to 0;$

$0 \to 2 \to 3 \to 0;$

$0 \to 1 \to 0.$

Результирующий тест $T_{\text{рез}}$ для функциональной верификации автомата *dk*14 содержит 54 тестовых набора, покрывает 26 дуг графа переходов для проверки синхронных переходов. Если провести моделирование на 40 наборах, то получим тест из 34 наборов и будет покрыто только 19 дуг.

Построение теста для выполнения асинхронных переходов

Проведенные эксперименты над конечными автоматами показали, что нахождение теста для проверки асинхронных переходов начальное моделирование со случайными значениями *rst* малоэффективно, так как требуется случайным образом сгенерировать входные наборы с единичными значениями сигнала *rst* для каждого состояния автомата. Более целесообразно построить тест для асинхронных переходов по тесту для синхронных переходов.

Задача 4. Задан функциональный тест $T_{\text{рез}}$ для проверки выполнения синхронных переходов. Требуется построить тест T^{ac} для проверки асинхронных переходов.

Для решения задачи 4 требуется провести моделирование автомата на тесте $T_{\text{рез}}$ и получить соответствующую последовательность $S_{\text{рез}}$ состояний автомата, затем в тест $T_{\text{рез}}$ добавить последний нулевой столбец, которому в формируемом тесте будет соответствовать сигнал *rst*. Для того чтобы построить тест для выполнения асинхронного перехода $s_i \to s_0$, надо найти в последовательности $S_{\text{рез}}$ кратчайшую подпоследовательность, начинающуюся с s_i и заканчивающуюся s_0, затем выделить соответствующие входные наборы из теста $T_{\text{рез}}$. После этого надо добавить еще один тестовый набор (повторить последний набор) и изменить в нем нулевое значение сигнала *rst* на единичное значение, что и позволит выполнить асинхронный переход из состояния s_i в состояние s_0.

Применим данный подход для теста, заданного в табл. 4. Построим тест для выполнения асинхронного перехода из состояния s_2 в состояние s_0.

В последовательности s_0, s_1, s_2, s_3, s_5, s_0, s_1, s_4, s_3, s_0, s_1, s_3, s_5, s_0, s_1, s_4, s_0 (табл. 4) найдем подпоследовательность s_0, s_1, s_2, которой соответствуют тестовые наборы

01010

10010.

Повторим последний вектор 10010 и последний нулевой разряд этого вектора заменим единичным значением. Получим тест

```
01010
10010
1001 1
```

для выполнения асинхронного перехода из состояния s_2 в состояние s_0 . Действуя аналогично, построим тесты для всех асинхронных переходов (табл. 5) автомата *Mealy*.

Таблица 5. Тест для проверки выполнения асинхронных переходов автомата *Mealy*

Такт	Тест T^{ac} $x_1\ x_2\ x_3\ x_4\ rst$	Состояния S^{ac}	Асинхронные переходы
0		$s0$	
1	0000**1**	$s0$	$s0 \rightarrow s0$
2	01010	$s1$	
3	0101**1**	$s0$	$s1 \rightarrow s0$
4	01010	$s1$	
5	10010	$s2$	
6	1001**1**	$s0$	$s2 \rightarrow s0$
7	01010	$s1$	
8	10100	$s3$	
9	1010**1**	$s0$	$s3 \rightarrow s0$
10	01010	$s1$	
11	00110	$s4$	
12	0011**1**	$s0$	$s4 \rightarrow s0$
13	01010	$s1$	
14	10100	$s3$	
15	11100	$s5$	
16	1110**1**	$s0$	$s5 \rightarrow s0$

Вычислительные эксперименты

Для автоматизированного построения функциональных тестов по результатам моделирования конечных автоматов была разработана программа *CoverGraph*, решающая задачи 2 и 3. С использованием программы *CoverGraph* были проведены вычислительные эксперименты на примерах *VHDL*-описаний конечных автоматов, имеющихся в библиотеке [11].

• Эксперименты показали, что *VHDL*-генерация длинных тестов, содержащих десятки тысяч тестирующих векторов, осуществляется за секунды работы персонального компьютера. *VHDL*-моделирование автоматов с десятками и сотнями состояний на таких тестах (для получения последовательностей состояний $S_{\text{исх}}$) занимает несколько минут работы персонального компьютера. Поэтому рекомендуется выполнить моделирование на возможно более длинных начальных тестах, чтобы обеспечить полноту построенного функционального теста. Отметим, что нетрудно написать тестирующую программу, в которой будут генерироваться входные наборы и записываться в текстовый файл *IN.TST*, а не считываться из файла.

• Выбор длины тестовой последовательности зависит от размерности автомата – числа входов и числа состояний автомата. Автоматы с большим числом состояний требуют более длинных тестовых последовательностей. Однако на получение графа $G(S, A)$, максимально приближенного к модельному графу $G^E(S^E, A^E)$ переходов конечного автомата, влияют также функции переходов. Если характеристическое множество булевой функции, представленной ДНФ D_{ij}, имеет небольшую мощность, то переход $s_i \rightarrow s_j$ может быть не выполнен при начальном моделировании, и граф $G(S, A)$ не будет содержать дугу $s_i \rightarrow s_j$, что приведет к неполному функциональному тесту, даже если программа *CoverGraph* найдет точное решение для графа $G(S, A)$. Точное решение задачи 3 гарантирует *минимальный* по длине тест $T_{\text{рез}}$, однако, на практике гораздо более важна не длина функционального теста, а его *полнота* – покрытие всех дуг.

• Программа *CoverGraph* позволяет быстро находить решение и гарантирует *полный функциональный тест*, соответствующий графу $G(S, A)$, например, для обработки текстовых файлов *IN.TST*, *STATE.TST*, содержащих 64 тысячи строк (тактов моделирования), требуется не более минуты. Во всех проведенных экспериментах программа построила полные функциональные тесты. Результаты эксперимента по нахождению тестов для функциональной верификации синхронных переходов конечных автоматов приведены в табл. 6.

Заключение. Представленный подход к получению функциональных тестов практичен и позволяет быстро получать компактные тесты для *VHDL*-моделей конечных автоматов, описывающих управляющую логику цифровых устройств. Получение функциональных тестов для управляющих блоков цифровых устройств

позволяет, в свою очередь, конструировать функциональные тесты для устройства в целом, так как появляется возможность дополнять тест и выполнять для заданных состояний управляющего блока операции над требуемыми операндами в операционном блоке, когда цифровое устройство представляется в виде композиции управляющего и операционного блока.

Таблица 6. Результаты экспериментов

Имя автомата	Длина теста $T_{исх}$	Число состояний автомата	Число дуг графа $G(S,A)$	Длина теста $T_{рез}$ для функциональной верификации
*Dk*14	40	7	19	34
*Dk*14	512	7	26	54
*Dk*14	64000	7	27	64
*Dk*15	512	4	12	25
*Dk*17	8000	8	23	63
*Dk*17	64000	8	23	63
Mealy	80	6	10	16
Mealy	8000	6	10	16
Kirkman	16000	16	32	32

Функциональные тесты для компонентных автоматов позволяют также конструировать тесты для верификации *VHDL*-описаний иерархических автоматов, для которых в определенных состояниях головного автомата начинает функционировать подчиненный конечный автомат.

1. *Ashenden P.J.*, *Lewis J.* VHDL-2008. Just the New Stuff. – Burlington, MA, USA. – Morgan Kaufman Publishers, 2008. – 244 с.
2. *Поляков А.К.* Языки *VHDL* и *VERILOG* в проектировании цифровой аппаратуры. – М.: СОЛОН-Пресс, 2003. – 320 с.
3. *Немудров В.*, *Мартин Г.* Системы-на-кристалле. Проектирование и развитие. – М.: Техносфера, 2004. – 216 с.
4. *Валидация* на системном уровне. Высокоуровневое моделирование и управление тестированием / М. Чэнь, К. Цинь, Х.-М. Ку и др. – М.: Техносфера, 2014. – 296 с.
5. *Лохов А.*, *Рабоволюк А.* Комплексная функциональная верификация СБИС. Система *Questa* компании *Mentor Graphics* // Электроника: наука, технология, бизнес. – 2007. – № 3. – С. 102–109.
6. *Баранов С.И.*, *Скляров В.А.* Цифровые устройства на программируемых БИС с матричной структурой. – М.: Радио и связь, 1986. – 272 с.
7. *Skliarova I.*, *Sklyarov V.*, *Sudnison A.* Design of FPGA-based Circuits using Hierarchical Finite State Machines. – Tallinn: TUT Press, 2012. – 242 p.
8. *Иванюк А.А.* Проектирование встраиваемых цифровых устройств и систем. – Минск: Бестпринт, 2012. – 337 с.
9. *Закревский А.Д.*, *Поттосин Ю.В.*, *Черемисинова Л.Д.* Логические основы проектирования дискретных устройств. – М. : Физматлит, 2007. – 589 с.
10. *Авдеев Н.А.*, *Бибило П.Н.* Расширение возможностей автоматизированного проектирования цифровых систем при использовании стандарта *VHDL'*2008 // Информационные технологии. – 2015. – № 7. – С. 510–520.
11. *Thimbleby H.* The directed Chinese Postman Problem // Software Practice and Experience. – 2003. – **33** (11). – P. 1081–1096.
12. *Бурдонов И.Б.*, *Косачев А.С.*, *Кулямин В.В.* Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай // Программмирование. – 2003. – № 5. – С. 11–30.
13. *Yang S.* Logic Synthesis and Optimization Benchmarks User Guide. V. 3.0. Technical Report. North. – Carolina. Microelectronics Center of North Carolina. – 1991. – 44 p.

Поступила 09.04.2015

Тел. для справок: +375 017 284-2084, 284-2076 (Минск)
E-mail: *bibilo@newman.bas-net.by*, *rom@newman.bas-net.by*

UDC 004.3

Bibilo P.N., Romanov V.I.

Constructing Compact Tests for Functional Verification of VHDL Descriptions of the Finite State Machines

In designing digital devices, the finite state machine (*FSM*) model is widely used. FSM as well as other models of digital devices is represented in VHDL and Verilog languages intended to design the digital circuits in the basis of modern VLSI or user-programmable FPGA logic integrated circuits. Using VHDL-description of the finite state machines, synchronous logic circuits are synthesized based on the logic elements, called technological (target) basis or target library gates. Now, the synthesis process is automated, and the most important issue in creating the VLSI projects is the problem of verifying the original and target specifications of algorithmic descriptions of the designed digital devices and systems. Verification of the correctness of VHDL-description means checking correspondence between the composed synthesizable VHDL-description and the digital system specifications. The great advantage of FSM model is that it can be verified. *Questa* simulation system in-

corporates the tools for the functional verification of the FSM if the model is written in a certain pattern, namely, in the form of two VHDL-processes. In the first process, the functions (transition and output) are recorded, and the other process realizes changes of the states, attaching to changes of the clock signals and the signal setting the FSM to the initial state. The first process is implemented in a combinational circuit after the circuit implementation, the second one in a memory element register. In the process of simulation, *Questa* simulation system allows recognizing a FSM, which is part of the project. This system identifies all passed (in particular modeling session) states of the FSM and calculates the number of passes of arcs in the transition graph of the FSM. Such tools are very useful; however, FSMs are included usually as parts of control blocks of more complicated projects. The verification of the whole project requires constructing compact functional tests for the control unit, FSM. The article considers the problem of automated construction of such tests according to the results of simulation of VHDL-description of FSM.

The FSM model and its VHDL-description are called correct if, in the FSM transition graph, there is a path from any internal state to the initial state of the FSM. Note that this condition is obligatory and always holds for the microprogram automata. The microprogram automata have been applied for a long time in the practice of engineering and are described in the present time in modern design languages. A *test* is an ordered sequence of sets of values of input signals fed to input ports of the FSM VHDL-model in simulation. A special test program (*testbench*) or a set of programs aimed at testing with different goals and different ways of organizing the tests is prepared for the simulation. The tests that allow you to check the correct functioning of the FSM model are called *functional*.

Problem. A correct VHDL-description of a FSM is given. It is necessary to construct a test T_{src} for functional verification by simulation that will check the performance of all the available transitions between internal states of the FSM.

The selection of FSM internal states from VHDL-description and the construction of the transition graph is a nontrivial task. In fact, it is necessary to automate the process of constructing a mathematical model of FSM in form of graph of transitions by analysis VHDL-program of FSM. To solve the problem we suggest an approach for obtaining an approximate solution based on

simulation of VHDL-description of the FSM on the pseudo-random test T_{src};

selection from the test T_{src} some test kits, which will be included in the target test T_{res}.

A program that allows constructing a directed graph of FSM transitions and finding coverage of all arcs on the base of the simulation results is developed. The input test kits corresponding to the arcs, which are in the coverage, make a test for the functional verification. An experimental research of the method of constructing compact tests for the verification of VHDL-models of FSM on standard examples is performed.

1. Ashenden P. J., Lewis J. VHDL-2008. Just the New Stuff. Burlington, MA, USA, Morgan Kaufman Publishers, 2008, 244 p.

2. Polyakov A. K. Yazyiki VHDL i VERILOG v proektirovanii tsifrovoy apparaturyi, M.: SOLON-Press, 2003, 320 p (In Russian).

3. Nemudrov V., Martin G. Sistemyi-na-kristalle. Proektirovanie i razvitie, M.: Tehnosfera, 2004, 216 p. (In Russian).

4. Chen M., Tsin K., Ku H.-M., Mishra P. Validatsiya na sistemnom urovne. Vyisokourovnevoe modelirovanie i upravlenie testirovaniem, M.: Tehnosfera, 2014, 296 p (In Russian).

5. Lohov A., Rabovolyuk A. Kompleksnaya funktsionalnaya verifikatsiya SBIS. Sistema Questa kompanii Mentor Graphics, Elektronika: nauka, tehnologiya, biznes. 2007, N 3, P. 102–109 (In Russian).

6. Baranov S.I., Sklyarov V.A. Tsifrovyie ustroystva na programmiruemyih BIS s matrichnoy strukturoy, M.: Radio i svyaz, 1986, 272 p (In Russian).

7. Skliarova I., Sklyarov V., Sudnison A. Design of FPGA-based Circuits using Hierarchical Finite State Machines, Tallinn: TUT Press, 2012, 272 p.

8. Ivanyuk A. A. Proektirovanie vstraivaemyih tsifrovyih ustroystv i system, Minsk: Bestprint, 2012, 337 p (In Russian).

9. Zakrevskiy A.D., Pottosin Yu.V., Cheremisinova L.D. Logicheskie osnovyi proektirovaniya diskretnyih ustroystv, M.: Fizmatlit, 2007, 589 p (In Russian).

10. Avdeev N.A., Bibilo P.N. Rasshirenie vozmozhnostey avtomatizirovannogo proektirovaniya tsifrovyih sistem pri ispolzovanii standarta VHDL'2008. Informatsionnyie tehnologii. 2015, N 7, P. 510–520 (In Russian).

11. Thimbleby H. The directed Chinese Postman Problem. Software Practice and Experience, 2003, V. 33 (11), P. 1081–1096.

12. Burdonov I.B, Kosachev A.S., Kulyamin V.V. Neizbyitochnyie algoritmyi obhoda orientirovannyih grafov. Determinirovannyiy sluchay. Programmirovanie, 2003, N 5, P. 11–30 (In Russian).

13. Yang S. Logic Synthesis and Optimization Benchmarks User Guide, v 3.0, Technical Report, North Carolina, Microelectronics Center of North Carolina, 1991, 44 p.