УДК 681.326

С. Альмадхоун, Е.Е. Сыревич, А.С. Шкиль

ПОИСК ОШИБОК ПРОЕКТИРОВАНИЯ В HDL-МОДЕЛЯХ ЦИФРОВЫХ АВТОМАТОВ

Введение. В современной САПР РЭА основным способом описания устройств являются языки описания аппаратуры (Hardware Description Language, HDL), например VHDL или Verilog, которые позволяют ускорить процесс разработки систем–на–кристалле в десятки раз. В основе разрабатываемых структурно-функциональных подходов к проектированию и верификации HDL-моделей лежит дуализм HDL-кода. С одной стороны HDL-модель – это описание на алгоритмическом языке с формализацией языковых конструкций и наличием специализированной среды разработки. С другой стороны HDL-модель – это описание цифрового устройства для САПР РЭА. Данное описание по формальным правилам преобразуется (синтезируется) в структурно-функциональные схемные модели разного уровня иерархии. Стандартизованные (шаблонные) фрагменты HDL-кода преобразуются в стандартные схемные реализации. Синтезируемые модели строго формально преобразовываются в схемные реализации устройства (ПЛИС, заказные СБИС, микропроцессорные структуры). Таким образом, HDL-модель – это фактически схема и к ней применимы методы схемного анализа, синтеза тестов, построения алгоритмов поиска дефектов, которые достаточно хорошо разработаны для цифровых схем.

Классификация цифровых устройств по типам языковых описаний. Тесты поиска дефектов позволяют определить (локализовать) ошибку (неисправность) в HDL-модели после применения тестов верификации. Вместо термина «дефект» и «неисправность» в дальнейшем используется термин «ошибка проектирования». Для HDL-моделей вводится модель ошибки проектирования, соответствующая ошибке в любом операторном выражении и не относящаяся к синтаксическим ошибкам. [1].

Языки описания аппаратуры позволяют описывать устройства различной сложности и назначения. В зависимости от типа описания и от наличия спецификации на верифицируемую модель используются разные методы построения тестов поиска ошибок проектирования.

Условно можно выделить следующие типы описаний устройств.

1. Простые:

– описания логических и арифметических (комбинационных) устройств – логические уравнения, преобразователи кодов, мультиплексоры, шифраторы, сумматоры, умножители, и т.д.;

– описания последовательностных устройств – триггеры, счетчики, регистры;

– описания моделей цифровых автоматов – «чистые» управляющие автоматы (Мили, Мура) в случае, если используется «автоматный» шаблон.

2. Сложные:

– описания устройств с микропрограммным управлением (АЛУ, микропроцессоры);

– описания алгоритмических устройств, реализующих алгебраические, тригонометрические или другие математические преобразования;

– аппаратно-ориентированные описания – описания с использованием библиотек, специфичных для выбранной аппаратной реализации;

– описания интерфейсов (UART, устройства сопряжения и приема/передачи данных);

– описания композиционных устройств (цифровой автомат с неотделимой и значительной операционной частью);

– нешаблонные описания – описания устройств, которые невозможно отнести их ни к одной группе, или они содержат части описаний разных типов.

Поиск ошибок проектирования в декомпозированных модулях. Повышение глубины локализации ошибки проектирования в HDL-модели достигается путем декомпозиции (явной – с выделением структурно и функционально законченного модуля на этапе реализации системы, неявной – с помощью средств анализа в специализированных САПР) и доискивания в декомпозированных модулях (применение ad hoc процедур для каждого типа модуля). Доискивание в простых описаниях, совпадающих с аппаратной реализацией, позволяет применить процедуры функционального или структурного методов поиска дефектов или их производные. Доискивание в описаниях функций дает возможность применить методы тестирования ПО и методы математического анализа для модулей (или фрагментов кода), реализующих непрерывные функций.

Выделение фрагментов HDL-кода, описывающих поведение автоматов стилем «автоматный шаблон», позволяет определить ошибки проектирования типа «неправильный переход в графе переходов автомата». Для их обнаружения и локализации можно провести стандартный диагностический эксперимент (ДЭ) над автоматами (проверка достижимости вершин с использованием линий сброса, обход всех дуг, обход всех

вершин).

В простых описаниях, совпадающих с аппаратной реализацией, легко восстановить соответствие между спецификацией (словесным, графовым, табличным представлением, представлением на основе графиков, временных зависимостей и т.д.) и HDL-моделью примитива, поэтому диагностирование неисправности в примитиве будет соответствовать локализации ошибки проектирования.

Локализация ошибок проектирования подразумевает наличие списка контрольных точек (КТ), тестовых значений и эталонов в указанных КТ. Существуют два типа КТ, используемых при поиске места ошибки проектирования HDL-модели. КТ первого рода – сигнал (переменная) модели, эталонные значения которых известны из спецификации. КТ второго рода – сигнал (переменная) модели, значения которых наблюдаемы, но до начала диагностического эксперимента неизвестны [2]. Все вышесказанное подразумевает доступность спецификации и ее однозначность.

Таким образом, предметом рассмотрения в данной работе является метод поиска ошибок проектирования в HDL–моделях цифровых конечных автоматов. Исходным описанием для построения алгоритма поиска ошибок проектирования и проведения диагностического эксперимента является спецификация в КТ первого рода, представленная содержательным графом переходов управляющего автомата. HDL-код модели автоматного модуля представляется в форме двухпроцессного автоматного шаблона на языке VHDL. Необходимо разработать порядок проведения и способ анализа результатов проведения диагностического эксперимента над фрагментом HDL-кода, ограниченного КТ первого рода.

Требования к тестопригодности HDL-модели управляющего автомата. Рассмотрим более подробно структуру HDL-моделей цифровых автоматов. Как было указано ранее, автоматы могут содержать значительную часть операционных вычислений, либо же реализовывать только функцию управления. Остановимся на втором типе автоматных моделей. САПР РЭА однозначно интерпретируют примитив как автомат, если при составлении HDL-модели был использован так называемый автоматный шаблон. Перечислим основные требования к тестопригоности HDL-модели управляющего автомата стиля «автоматный шаблон».

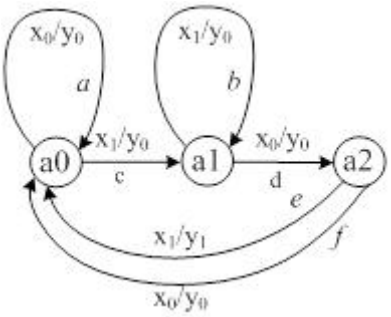
1. Общепринятая модель управляющего (структурного) автомата (модель Хаффмена) состоит из комбинационного и последовательностного компонентов. Последовательностный компонент содержит элементы памяти, такие как синхронные триггеры, которые запоминают состояние и позволяют изменять его синхронно, например, по переднему фронту синхроимпульса. Комбинационный компонент состоит из логических элементов, которые реализуют две логические функции: функцию выходов и функцию переходов. *Функция выходов* вычисляет значения выходных сигналов. *Функция переходов* вычисляет новые значения элементов памяти (т.е. значение следующего состояния). Для описания поведения конечного автомата обычно используется граф переходов (в англоязычной литературе – *state diagram* или диаграмма состояний).

2. Обязательных компонентов в автоматном шаблоне два: анализ синхросигнала для активизации перехода или сохранения состояния (блок синхронных D-триггеров) и анализ текущего состояния (и входных значений) для выбора следующего состояния. Таким образом, во-первых, необходимо реализовать описание блока D-триггеров с общей синхронизацией, что требует условного оператора типа `if-then-else` с асинхронным сбросом. Во-вторых, необходимо анализировать несколько альтернатив (текущих состояний) для реализации функции переходов, что требует оператора выбора типа `case` и задания альтернатив перечислимым типом.

3. Для повышения тестопригодности HDL-модели автомата наложим на HDL-код следующие ограничения: использование двух раздельных процессов, условный оператор, вычисляющий функцию перехода, всегда должен иметь ветку `else` , сброс в начальное состояние – асинхронный.

Как более общий случай, рассмотрим граф переходов автомата Мили, вершины которого могут иметь более двух исходящих дуг и который в общем случае может представляться мультиграфом, т.е. для перехода $a_i \Rightarrow a_j$ может существовать более, чем одна дуга с разными функциями выходов. Указанные особенности реализуются в автоматном шаблоне с использованием условных операторов `if()-elsif()-else`. Каждой дуге графа переходов автомата Мили соответствует альтернатива в условном операторе : `if (Cond) then A1; Out1; else A2; Out2; end if;`, где `(Cond)-` условие выполнения перехода, `A1` – следующее состояние и вычисление выходного значения `Out1` на переходе по выполнению условия; `A2` – следующее состояние, `Out2` – вычисление выходного значения на переходе по невыполнению условия. Если необходимо реализовать несколько исходящих дуг, то вместо ветки `else` используется ветка `elsif` с новым условием.

На рис.1 представлен граф переходов автомата Мили, который осуществляет выделение группы из трех символов (101) из двоичной последовательности произвольной длины и фрагмент его VHDL-модели в форме двухпроцессного автоматного шаблона.



а)

```
  begin
p1 : process (state, data) is
begin
case state is
    when a0 =>
       if  (data='0')  then
              nextstate <= a0;
              y <= '0';
       else nextstate <= a1;
              y <= '0';
       end if;
```

б)

Рис. 1. Граф перехода автомата Мили а) и фрагмент его VHDL-модели б)

На рис.2. приведена временная диаграмма моделирования работы рассматриваемого автомата при подаче на него входной последовательности 0–0–1–1–0–0–1–0–1. Синхронизация во втором процессе выполняется по переднему фронту. При этом реализуется следующий маршрут обхода графа:

a0–a(y0)–a0–c(y0)–a1–b(y0)–a1–d(y0)–a2–f(y0)–a0–c(y0)–a1–d(y0)–a2–e(y1)–a0.

Отметим, что в скобках рядом с именем дуги указывается выходной сигнал автомата Мили, соответствующий данной дуге.

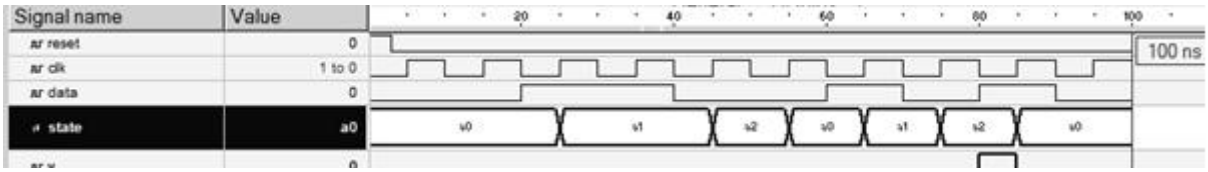


Рис. 2. Результат моделирования роботы автомата Мили в среде Active-HDL

Рассматриваемый класс автоматов и его описания должны удовлетворять следующим условиям.

1) Для исправного автомата A = < X, Y, Z, f, g, a0 >, где f – функция переходов, g – функция выходов. Автомат имеет начальное состояние a0 и установочную последовательность для перехода в это состояние конечной длины (в простейшем случае длина установочной последовательности не превышает 1 за счет наличия цепей сброса), причем модуль, реализующий сброс в a0 всегда исправен, автомат может быть установлен в a0 при любых условиях или неисправностях.

2) Неисправности автомата не изменяют числа его состояний. Если это было бы не так, то на этапе проверки синтаксиса компилятор указал бы соответствующую ошибку. Исправный и все множество неисправных автоматов составляют исключительный класс, т.е. имеют различные функции выходов для каждого состояния. Применительно к HDL-моделям автоматов это обеспечивается способом представления и отображения результатов моделирования работы автомата в виде временной диаграммы (*Waveform*), например, в среде проектирования, в частности, Active–HDL.

3) Веса всех дуг графа переходов автомата одинаковы, а приоритет задан последовательностью альтернатив в условном операторе.

4) Выполняются только безусловные эксперименты над автоматом и в процессе проведения эксперимента дополнительных неисправностей в автомате не возникает.

5) Для описания HDL-модели используется двухпроцессный шаблон. Последовательностная часть отделена от комбинационной и оттестирована заранее. Т.к. последовательностная часть при использовании автоматного шаблона представляется синхронным D-триггером с асинхронным сбросом, то его тестирование проводится стандартным тестом «пара нулей-пара единиц» (00110). Аналогичный прием был использован в методе ScanPath, описанном R.G. Bennetts [3].

Рассмотрим возможные ошибки в HDL-модели комбинационного блока, которые вытекают из автоматного шаблона (рис.1б).

1) Оператор выбора `case` содержит не все альтернативы, заданные в перечислимом типе `type statetype is (a0, a1, a2);`.
2) В условном операторе `if` нет альтернативы `else`.
3) Ошибочное назначение следующего состояния `nextstate`, формирующее новый переход.
4) Ошибочное условие, повлекшее переход не в то состояние и, соответственно, вычисление ошибочного выходного значения `y`.
5) Ошибочное назначение выходного значения `y`.

С точки зрения теории тестирования программного обеспечения любой код должен быть предварительно проинспектирован так называемым экспертом. В роли эксперта обычно выступает

проектировщик более высокого уровня квалификации или специально выделенный для этого специалист. В классической литературе [4] указано, что более 60% ошибок в программном коде можно обнаружить с помощью неформального исследования (инспектирования). Инспектирование использует знания о предметной области и языке программирования/проектирования. Эксперт должен знать типы ошибок, присущие конкретным языкам и программам определенного типа. Для начала процесса инспектирования программы необходимы следующие условия.

1. Наличие точной спецификации программного кода, предназначенного для инспектирования. Без полной спецификации невозможно обнаружить ошибки в проверяемом компоненте.
2. Эксперт должен хорошо знать стандарты разработки.
3. У эксперта должна быть синтаксически корректная последняя версия программного кода.

Таблица 1.

Ошибки, выявляемые при инспектировании HDL-кода

Класс ошибок	Вопросы, помогающие выявлять ошибки
Ошибки данных	Все ли переменные/сигналы в HDL–коде инициализированы до начала использования их значений? Все ли константы именованы? Равна ли верхняя граница массива его размеру или на единицу меньше этого размера?
Ошибки управления	Выполняются ли условия для каждого условного оператора? Все ли циклы завершаются? Правильно ли в составных операторах расставлены скобки? Все ли выборы выполняются в операторах выбора ? Все ли альтернативы, заданные в перечислимом типе type statetype is (a0, a1, a2); содержатся в операторе выбора case? Все ли условные операторы if содержат альтернативу else.
Ошибки ввода-вывода	Используются ли в HDL-коде входные сигналы? Всем ли выходным сигналам перед выводом назначаются значения?

Таким образом, ошибки проектирования типа «Оператор выбора `case` содержит не все альтернативы, заданные в перечислимом типе данных `type statetype is (a0, a1, a2)`» или «В условном операторе `if` нет альтернативы `else`» в комбинационном блоке могут быть выявлены и устранены путем инспектирования.

Диагностический эксперимент. Выделение в HDL-коде шаблонов описания автоматов позволяет провести ДЭ над автоматами путем проверки достижимости всех вершин с использованием линий сброса.. Для построения теста реализуется стратегия обхода всех дуг графа переходов конечного автомата начиная с начальной вершины при условии допустимости наличия более, чем одной дуги $a_i \Rightarrow a_j$ (смешанная стратегия). При этом проверяются все одиночные неисправности переходов, а также исправности функций автомата, обеспечивающих эти переходы [5]. Построение входных воздействий (тестов) осуществляется с использованием методики, которая реализована в программе ASFTEST [6].

ДЭ над HDL-моделью конечного автомата состоит в подаче на нее входных воздействий в соответствии с выбранной стратегией обхода содержательного графа переходов, получении выходных реакций в виде списка состояний автомата на Waveform или списка обхода графа в файле и сравнения полученных реакций с эталоном (спецификацией в КТ первого рода) визуальным или программным путем. На основании этого делается вывод о соответствии HDL-модели спецификации. ДЭ проводится с использованием системы верификации HDL-моделей (TestBench) в среде проектирования Active–HDL.

Для описания графа переходов предложено использовать модифицированную матрицу смежности. Данная матрица характеризуется тем, что для каждой вершины графа (строки матрицы) в ячейках матрицы указываются имена дуг, соединяющих данную вершину с преемниками (столбцами). При этом в одной ячейке могут быть более одной дуги (мультиграф), что представлено на рис.3.

	a0	a1	a2
a0	*a*	*c*	
a1		*b*	*d*
a2	*e, f*		

Рис. 3. Матрица смежности для графа, представленного на рис. 1а)

Для реализации стратегии обхода всех дуг графа строится дерево решений (дерево обхода графа). Алгоритм построения дерева решений следующий.

1. Построение дерева обхода графа начинается с начальной вершины a0. Анализируется первая строка матрицы смежности и составляется список вершин-преемников.

2. Из списка преемников выбирается первая вершина и включается в дерево. Соответствующая ячейка в матрице (дуга) помечается.

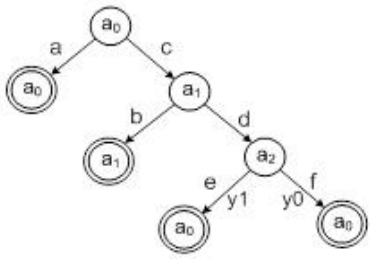
3. Для строки, соответствующей этому номеру, составляется список непомеченных преемников.

4. Построение маршрута в графе завершается терминальной (конечной) вершиной. Вершина в дереве становится терминальной, если :

– входящая дуга является петлей (элемент главной диагонали матрицы смежности);
– очередная вершина-преемник является начальной вершиной a0 (завершен цикл работы автомата);
– все ячейки в очередной строке (при выполнении п.3) окажутся помеченными.

5. П.п. 2, 3 и 4 повторяются до тех пор, пока все дуги (ячейки матрицы смежности) не окажутся помеченными.

На рис.4 показано дерево решений для обхода дуг графа переходов и маршруты обхода. При этом следует отметить, что число маршрутов обхода графа переходов равно числу терминальных вершин (вершин, не имеющих преемников) дерева решений. На рис. 4а) терминальные вершины отмечены двойной окружностью.



а)

A	
1	a0-a-a0
2	a0-b-a1-c a1
3	a0-c-a1-d-a2-f-a0
4	a0-c-a1-d-a2-e-a0

б)

Рис. 4. Стратегия обхода всех дуг графа переходов: а) – дерево решений; б) – маршруты обхода графа

Проведение диагностического эксперимента.

Пусть в HDL-коде есть ошибка «альтернатива `else` в операторе `if` комбинационного блока автоматного шаблона (анализ состояния a1) выполняется неверно». Это ведет к формированию новой дуги (`a1-a0`) и обрыву дуги d (на переходе `a1-a2`). На рис.5 приведен фрагмент VHDL-модели рассматриваемого автомата с ошибочным оператором назначения нового состояния (серый цвет). Входной сигнал автомата в приведенной модели обозначен `data = {'0','1'}`.

```
when a1 =>
  if (data='1') then nextstate <= a1;
        y <= '0';
  else nextstate <= a0; (вместо nextstate <= a2)
        y <= '0';
end if;
```

Рис. 5. Фрагмент его VHDL-модели автомата Мили с ошибочным оператором назначения

В качестве элементарной проверки P_i при проведении ДЭ используется реализация определенного маршрута обхода графа, при этом номер маршрута (матрица А на рис.4б) соответствует номеру элементарной проверки. Результат элементарной проверки v_i считается отрицательным, если терминальная вершина на этом маршруте достигнута, в противном случае результат элементарной проверки считается положительной.

$$v_i = \begin{cases} 0 \rightarrow \textit{если по этому маршруту достигнута терминальная вершина} \ (\textit{тест прошел}); \\ 1 \rightarrow \textit{если по этому маршруту не достигнута терминальная вершина} \ (\textit{тест не прошел}), \end{cases}$$

Результатом проведения ДЭ по обходу графа является вектор экспериментальных проверок (ВЭП) $V = (v_1, v_2, .. v_m)$, где m – число терминальных вершин дерева решений.

Для проведения диагностического эксперимента использована встроенная система верификации HDL-моделей (TestBench) в среде проектирования Active–HDL. На рис.5 приведен фрагмент HDL-кода TestBench, реализующего подачу входных воздействий на HDL-модель, которые обеспечивают реализацию маршрутов обхода графа из матрицы А (рис. 4б).

```
architecture TB_ARCHITECTURE of catcher_tb_my is
    ...
begin
    UUT : catcher port map (reset => reset,
                            clk => clk,
                            data => data,
                            y => y );
process
begin
    clk<='0', '1' after 5 ns; wait for 10 ns;
end process;

process
begin
    reset<='1','0' after 3 ns; wait for 200 ns;
end process;

process
begin
    data <= '0'; wait for 20 ns; data <= '1'; wait for 20 ns;
    data <= '0'; wait for 20 ns; data <= '1'; wait for 10 ns;
    data <= '0'; wait for 10 ns; data <= '1'; wait for 10 ns;
end process;
 end;
```

Рис. 6. TestBench для подачи входных воздействий, реализующих маршруты обхода графа, в ходе проведения диагностического эксперимента

Результат проведения диагностического эксперимента над HDL-моделью с ошибкой проектирования (`else nextstate <= a0;` вместо `nextstate <= a2`) в виде временной диаграммы (Waveform) приведен на рис.7.

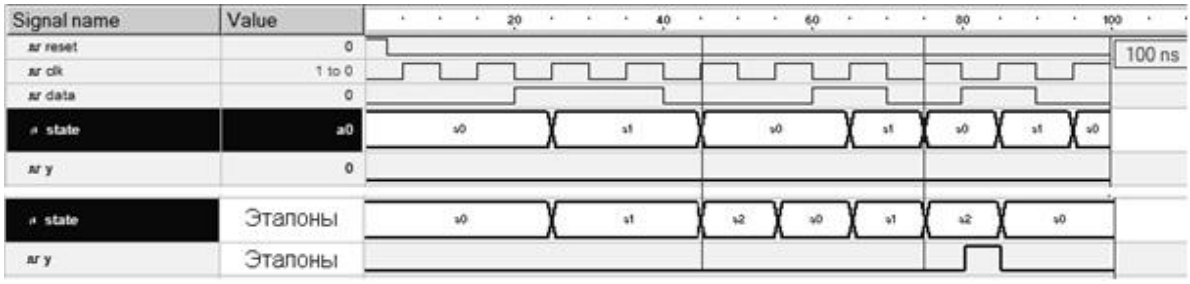
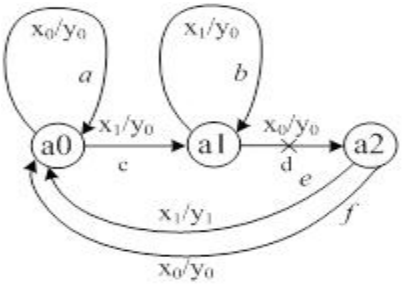


Рис. 7. Временная диаграмма результатов проведения диагностического эксперимента

На приведенной Waveform элементарные проверки реализованы в следующие промежутки времени: P_1 – {0-15 ns}, P_2 – {15-35 ns}, P_3 – {35-55 ns}, P_4 – {55-75 ns}. Сравнение с эталоном произведено визуально, эталоны взяты из Waveform на рис.2. Элементарные проверки P_3 и P_4 положительны, т.к. состояние a2 в моменты времени 45 ns и 75ns, соответственно, не достигнуто.

Матрица обхода графа (B) и ВЭП (как результат проведения ДЭ в графе с внесенной ошибкой проектирования)) представлены на рис.8. Следует отметить, что матрица A (списки маршрутов, рис.4б) и матрица B (рис.7б) эквивалентны, но матрица B более удобна для машинной обработки результатов ДЭ.



а)

B	a	b	c	d	e	f	ВЭП (V)
P1	1						0
P2		1	1				0
P3			1	1		1	1
P4			1	1	1		1

б)

Рис. 8. Результат ДЭ а) – граф с ошибочной дугой *d*; б) – маршруты обхода графа переходов и ВЭП

Место возникновения ошибки в маршруте обхода графа переходов находится по формуле (1) [7]; где M_j^B - j-я строка матрицы B.

$$D = \cap|_{v_j=1} M_j^B - \cup|_{v_j=0} M_j^B \quad (1)$$

D = {c, d, e} ∩ {c, d, f} – {a} ∪ {b, c} = {c, d} – {a, b, c} = { d }, т.е. ошибочная дуга d в графе переходов найдена. Для нахождения места возникновения ошибки проектирования необходимо возвратиться к HDL-коду автоматного шаблона и выполнить визуальное инспектирование участка кода, реализующего ошибочный фрагмент маршрута обхода графа (a1-d-a2). После нахождения ошибочного оператора или его фрагмента (`else nextstate <= a0;`) он корректируется (`else nextstate <= a2;`) и диагностический эксперимент по проверки корректности HDL-кода повторяется.

Выводы. В работе рассмотрен подход к поиску ошибок проектирования в структурированном HDL-коде на этапе его функциональной верификации, если HDL-модель представлена в виде автоматного шаблона. Для построения теста реализуется стратегия обхода всех дуг графа переходов конечного автомата начиная с начальной вершины при условии допустимости наличия более, чем одной дуги $a_i \Rightarrow a_j$. Предложенный метод позволяет находить ошибки проектирования типа «не выполняется переход из состояния a_i в состояние a_j » в фрагментах HDL-кода, описанных двухпроцессным автоматным шаблоном при условии, что спецификацией на этот фрагмент кода является содержательный граф переходов конечного автомата Мили. В качестве примера рассмотрена модель конечного автомата Мили на языке VHDL, в качестве инструментального средства проведения диагностического эксперимента использована среда проектирования Active–HDL.

ЛИТЕРАТУРА

1. Шкиль А.С. Модели описаний цифровых устройств для диагностирования / А.С. Шкиль, Е.Е. Сыревич, С. Альмадхоун // Вестник Херсонского государственного технического университета. – 2010. – №2 (38). – С. 258-265.
2. Альмадхоун С. Методы поиска ошибок проектирования в HDL-моделях цифровых устройств в условиях неполной спецификации устройств / С. Альмадхоун, Е.Е. Сыревич, А.С. Шкиль // Інформаційно-керуючі системи на залізничному транспорті. – 2010. – № 4. – С. 30-32.
3. Беннеттс Р.Дж. Проектирование тестопригодных логических схем: пер. с англ. / Р. Дж Беннеттс. – М.: Радио и связь; 1990. – 176 с.
4. Соммервилл И. Инженерия программного обеспечения: пер. с англ. / И. Соммервилл. – М.: Изд. дом «Вильямс», 2002. – 624 с.
5. Хаханов В.И. Система генерации тестов для проектирования цифровых автоматов в среде ACTIVE-HDL / В.И. Хаханов, Е.В. Ковалев, В.В. Ханько, Масуд М.Д. Мехеди // АСУ и приборы автоматики. – Харьков. – 2000. – Вып.111. – С. 15-22.
6. Хаханов В.И. Модели цифровых автоматов для проектирования тестов в среде Active-HDL / В.И. Хаханов, А.С. Шкиль, Е.В. Ковалев // Радиоэлектроника и информатика. – 2000. – № 2 (11). – С. 86–92.
7. Шкиль А.С. Методы поиска ошибок проектирования в HDL-коде / А.С. Шкиль, Е.Е. Сыревич, Д.Е. Кучеренко, Г.П. Фастовец // Радиоэлектроника и информатика. – 2008 – №. 3. – С. 47–53.

АЛЬМАДХОУН Самер – аспірант кафедри Автоматизації проектування обчислювальної техніки Харківського національного університету радіоелектроніки (ХНУРЕ).

СИРЕВИЧ Євгенія Юхимівна – кандидат технічних наук, доцент кафедри Електронних обчислювальних машин Харківського національного університету радіоелектроніки (ХНУРЕ).

ШКІЛЬ Олександр Сергійович – кандидат технічних наук, доцент кафедри Автоматизації проектування обчислювальної техніки Харківського національного університету радіоелектроніки (ХНУРЕ)