УДК 681.326

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

МОДЕЛИ HDL–ОПИСАНИЙ ЦИФРОВЫХ УСТРОЙСТВ ДЛЯ ДИАГНОСТИРОВАНИЯ

Введение. Полный цикл диагностирования модели цифрового устройства (ЦУ), начиная от его описания на языках описания аппаратуры (Hardware Description Language, HDL) и заканчивая поиском места возникновения ошибки проектирования, включает в себя этапы: описание модели на языках описания аппаратуры, компилирование внутренней модели, генерацию тестов, построение списка ошибок проектирования, получение эталонов, алгоритмы обработки диагностической информации и, собственно, методы поиска дефектов. Таким образом, процесс верификации HDL-моделей ЦУ основывается на следующих составляющих: модель ЦУ, позволяющая генерировать детерминированные тесты; модель ошибки проектирования, характерная для HDL-описания; алгоритм генерации тестов; критерии оценки качества верификации [1]. При этом важную роль играет способ преобразования HDL-описаний ЦУ в модели для целей диагностирования.

Постановка задачи. Целью данной работы является разработка эффективных методов диагностирования моделей цифровых устройств, представленных на языках описания аппаратуры, для уменьшения глубины диагноза и ускорения локализации места, причины и виды дефекта. Для достижения поставленной цели необходимо решить следующие задачи.

1. Разработать теоретические основы диагностирования HDL-моделей ЦУ
2. Определить класс ошибок проектирования.
3. Усовершенствовать графовую модель HDL–кода для целей поиска ошибок проектирования.

Двойственность HDL-модели. Аппаратные системы сложнее «чистых» программистских, т.к. это hard-realtime системы. На каждую функцию есть жесткое ограничение по времени, за которое она должна отработать при любых входных данных. На задержки влияет геометрия и взаимное положение элементов — еще один фактор, в принципе отсутствующий в разработке программного обеспечения (ПО). Это не слишком сильно влияет на встраиваемые системы на этапе проектирования, но это принимается во внимание, потому что может влиять на архитектуру. Каждый лишний миллиметр площади кристалла – это порядка 6 центов для техпроцесса при 0,13мкм. Одна итерация с прототипом — это минимум 4 месяца. Перевыпуск фотошаблона – это от сотен тысяч долларов до порядка миллиона (или выше), если речь идет о современных тонких техпроцессах. Языки описания аппаратуры служат для формального описания дискретных устройств вычислительной техники и могут быть использованы на всех этапах разработки цифровых электронных систем. VHDL может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры так же, как и для передачи данных о проекте, модификации и сопровождения. Первоначально HDL предназначался для моделирования (что и объясняет его большую универсальность), но позднее из него было выделено синтезируемое подмножество. Написание алгоритмической модели на синтезируемом подмножестве гарантирует автоматический синтез по этой модели алгоритмической схемы. В HDL были введены понятия модельного времени, сигнала, события, компонента и другие. Использование HDL позволяет не привязывать проект заранее к конкретному физическому способу реализации, одна и та же логическая HDL-реализация является источником генерации различных физических реализаций. В основе языка лежат следующие принципы построения: поддержка функциональной декомпозиции (функциональная иерархия и рекурсия); поддержка структурной декомпозиции (структурная иерархия); представление системы в виде параллельно функционирующих взаимодействующих процессов; использование абстрактных типов данных; использование событийного моделирования; поддержка различных уровней абстракции и детализации представления проекта. То есть главная особенность HDL – параллелизм и наличие сигналов в описании. В программной модели все операторы выполняются друг за другом (не учитывая переходы и вызовы функций). В некоторый момент времени выполняется один оператор. В аппаратной модели на ЯОА все назначения сигналов выполняются одновременно в один и тот же момент времени. Рассмотрим простой фрагмент кода: *a=b; c=a*. Если это фрагмент программной модели (где операторы выполняются последовательно), то ясно, что ‘c’ и ‘a’ будут содержать значение ‘b’. Если это фрагмент аппаратной модели (где операторы выполнятся параллельно), то ‘c’ будет содержать старое значение ‘a’, а в ‘a’ будет помещено значение ‘b’. Таким образом, модели на языках описания аппаратуры, с одной стороны, выглядят и ведут себя как код на языках программирования (например, при использовании подпрограмм или переменных); с другой стороны, обладают рядом кардинальных отличий (сигналы, параллелизм, синтезируемость). Отсюда невозможно тестировать и диагностировать такие модели методами исключительно верификации ПО или методами диагностирования аппаратуры.

Определение. Система синтеза – это комплекс программных средств, выполняющий преобразование кодов, составленных на языках описания аппаратуры, в некоторую схемную реализацию по некоторым правилам.

Определение. Логически связанные операторы – это операторы, описывающие некоторую законченную аппаратную реализацию различной степени абстракции.

Пример:

```
if (reset='1') then OUT_P<=(others=>'0');
        elsif clk='1' then
        if (IN_choice ='0') then   A<=in_P; else  B<=in_P;
end if;   end if;    OUT_P<=("0"&A)+("0"&B);
```

Логически связанные операторные конструкции данного примера результируются в АЛУ–подобную структуру.

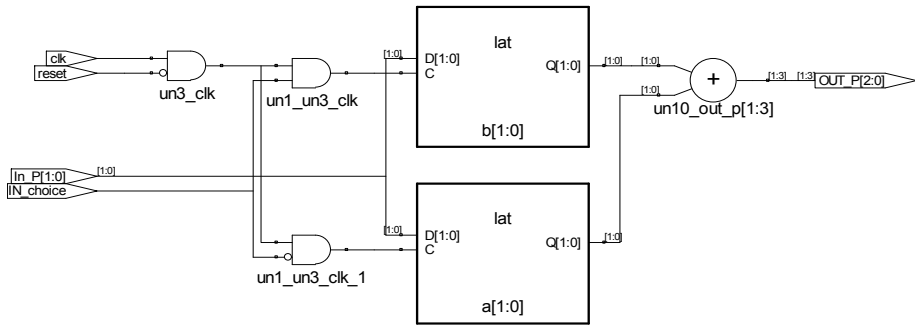

Рис. 1. Результат синтеза связанных операторов

Определение. Уровень регистровых передач в системе синтеза – это иерархический уровень цифровых устройств, на котором используются модели, отражающие обмен сигналами между блоками такими, как вентили, регистры, сумматоры, дешифраторы

Утверждение 1. Система синтеза однозначно преобразует исходное VHDL описание в схемную реализацию уровня регистровых передач в независимости от выбранной технологии. Уровень регистровых передач основывается на библиотечных элементах, которые не зависят от выбранной реализации. Правила преобразования операторных конструкций стандартизированы (примером может служить стандарт 1076.6-1999 IEEE Standard for VHDL RTL Synthesis и 1364.1-2002 IEEE Standard for Verilog RTL Synthesis и их более поздние версии). Возможные настройки систем синтеза не влияют на реализацию исходного стандарта. Значит, можно предположить, что в рамках стандарта каждый оператор языка преобразуется в единственно возможную схемную реализацию.

Утверждение 1 и определение системы синтеза позволяют утверждать об однозначном соответствии операторов языка описания и схемных эквивалентов в выбранной системе синтеза.

Условие 1. Постсинтезное моделирование списка соединение при проведении двух и более экспериментов над одним и тем же исходным языковым описанием результируется в идентичные временные диаграммы. Если это не так, то в процесс синтеза была внесена ошибка.

Утверждение 2. Каждый оператор или группа логически связанных операторов соответствует некоторой схемной реализации. Предположим обратное: пусть операторы языка при каждой итерации преобразования результируются в различные схемные структуры. Различные схемные реализации одного и того же алгоритма будут содержать в таком случае разные временные параметры (из-за вносимых системой синтеза задержек на каждом элементе) и результироваться в отличающиеся друг от друга временные диаграммы. В случае соблюдения условия 1 схемные реализации совпадают.

Модель ошибки проектирования. Ошибка проектирования как дефект. Стоит задача, исходя из двойственности HDL – модели, определить характер возможных ошибок, их влияние на конечную реализацию (устройство после синтеза) и методы поиска. Вместо термина «дефект» и «неисправность» в дальнейшем будет использоваться понятие «ошибка проектирования». Для HDL-моделей вводится модель ошибки проектирования, соответствующая ошибке в любом операторном выражении и не относящиеся к синтаксическим ошибкам. Суть его состоит в том, что любое из значений, поступающее на любую переменную в поведенческом или структурном описании объекта может быть ошибочным. Необходимо установить, приведет ли ошибочное значение на переменной к ошибочному значению на выходе. Воздействуемое VHDL выражение – это выражение, соответствующее ошибке. Различают ошибки одиночные и постоянные: только одна ошибка вводиться в один момент времени и эффект от данной ошибки сохраняется во время всего моделирования. Другие операторы считаются исправными, т.к. константные ошибки на том же сигнале, но в разных операторах считаются разными.

Модель ошибки разделена на: ошибки на данных, ошибки на выражениях и ошибки на операторах (1).

$$F = \{F^D, F^{OP}, F^{EX}\}, \quad (1)$$

где F^D – ошибки на данных; F^{OP} – ошибки в операторах; F^{EX} – ошибки в выражениях. Ошибки на данных основываются на ошибках типа ОКН. Ошибки на выражениях делятся на 2 класса: ошибки в управляющих конструкциях и в левой части выражений назначения (2).

$$F^D = \{F^{Dest}, F^{Source}\}, \quad (2)$$

где F^{Dest} – ошибки на сигналах – приёмниках; F^{Source} – ошибки на сигналах – источниках. Ошибки в операторах также могут встречаться в управляющих конструкциях при невыполнении любой из альтернатив (3).

$$F^{OP} = \{F^{log}, F^{arith}, F^{shift}, \ldots\}, \quad (3)$$

где F^{OP} – ошибки на каждом типе операторов (логических, арифметических, сдвига и т.д.). Внесение неисправностей в описание вводится путём изменения VHDL-кода. Результат моделирования сравнивается с результатом исправного моделирования (4).

$$\text{Ошибка любого типа определяется: } \begin{cases} F^{general} = 1, \text{если } R^{expert} \neq R^{etalon} \\ 0, \text{в противном случае} \end{cases}, \quad (4)$$

где R^{expert} – результаты моделирования ОД на заданном тесте; R^{etalon} – эталонные реакции на тех же входных воздействиях. При чём в один момент времени F определяется только, как $F=\{F^D\}$, либо $F=\{F^{OP}\}$, либо $F=\{F^{EX}\}$. В работе рассматривается только $F=\{F^{OP}\}$! Модель одиночной ошибки для VHDL-описания полностью ориентирована на модели аппаратных неисправностей, и поэтому используется только в пределах синтезируемых описаний. С точки зрения верификации аппаратной модели операторы HDL не содержат внутри себя ошибок и работают исправно. Если бы это было не так, то верификация HDL-модели включала бы в себя проверку системы моделирования. А эти две задачи должны выполнять разные специалисты — программист и проектировщик. Поэтому не имеет смысла выбирать в качестве ошибок проектирования неисправности работы оператора (так как верифицировать планируется поведенческое описание объекта). Определение перечня ошибок проектирования необходимо для разработки стратегии верификации описания устройства. Пример ошибочных назначений приведен ниже в таблице 1.

Таблица 1

Пример ошибки проектирования

Требуемый оператор fj	Ошибочный оператор fi
+ (add)	– (sub),*(mul), / (div)
And	or, xor, xnor, nor, nand
Прямое назначение	Not (инверсное)

Утверждение 3. Каждая ошибка проектирования типа «замена оператора» результируется в ошибку аппаратной реализации одного или группы логически связанных операторов.

Следствие 1. Ошибка проектирования является существенной, если в синтезированной схеме на специфицируемом подмножестве данных есть воздействия, на которых результаты работы (моделирования) отличаются от спецификации.

Пример исправного (предполагаемого) кода.

```
process (CLK, D)
begin
        if CLK = '1' and CLK'event then Q <= D;
end if; end process;
```

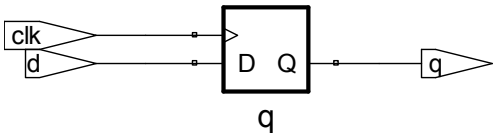

Рис. 2. Результат синтеза исправного кода

Пусть была внесена ошибка проектирования T_i/T_j и вместо *CLK = '1'* употребили *CLK = '1' and CLK'event.*

Пример неисправного кода.

```
process (CLK, D)
begin
        if CLK = '1' then Q <= D;
end if; end process;
```

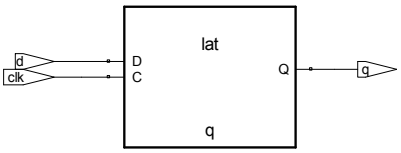

Рис. 3. Результат синтеза кода с внесенной ошибкой

Это приводит к ошибочному синтезу синхронного триггера вместо триггера-защелки.

Следствие 2. Ошибка проектирования несущественна, если после синтеза схема на основе кода с данной ошибкой и схема без нее будут давать одинаковые результаты на всех специфицируемых воздействиях.

Пример исправного (предполагаемого) кода.

```
architecture beh of replace is
begin
c<=A and B;
end;
```

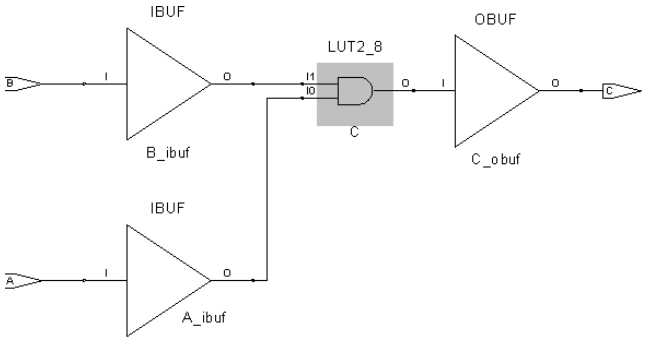

Рис. 4. Результат синтеза исправного кода

Пусть была внесена ошибка проектирования T_i/T_j и вместо *c<=A and B;* употребили *c<=not (not(A) or not (B));.*

Пример неисправного кода с несущественной ошибкой.

```
architecture beh of replace is
begin
c<=not (not(A) or not (B));
 end;
```

При анализе компонентов схема, полученной после синтеза видно, что схемы идентичны (значит, и результаты моделирования совпадают), то есть ошибка проектирования оказалась несущественной.

Следствие 3. Тест, проверяющий все ошибки проектирования в поведенческой модели на языке описания аппаратуры, проверит все существенные ошибки замены на уровне регистровых передач и списка соединений синтезированной модели (очевидна аналогия теоремы Рота для дискретных устройств). Таким тестом может являться тест, построенный на основании предложенного в [2] подхода к верификации моделей цифровых устройств на языках проектирования аппаратуры, подразумевающий построение различающих последовательностей и обнаружение существенных ошибок в HDL-модели. Таким образом, задача диагностирования HDL-модели сводится в различении ошибочных состояний среди всех возможных в неизбыточных (содержащих только существенные ошибки) моделях.

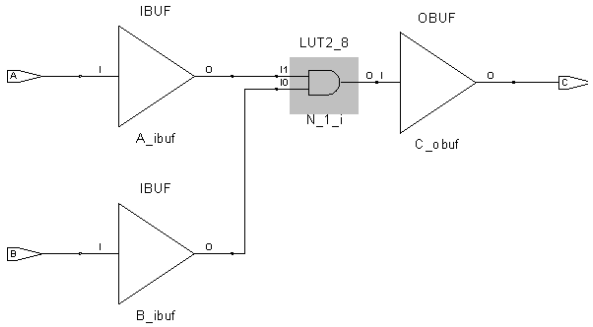

Рис.5. Результат синтеза кода с внесенной несущественной ошибкой

HDL-модель как объект диагностирования. Если рассматривать языки HDL с точки зрения создания моделей ЦУ, то большинству его конструкций и структур можно поставить в соответствие структурные, функциональные и физические характеристики цифровых систем. Принимая во внимания особенности HDL, в качестве объекта диагностирования можно эффективно использовать графовое представление описания устройства на HDL [3]. Описание на HDL представляется в виде двух графов. Первый граф задается выражением $G_i = (V_i, E_i, L_c\ T_i)$, где V_i определяет набор операндовых вершин, E_i – набор функциональных вершин и T_i – набор связей между операндовыми и функциональными вершинами. V_i состоит из двух типов компонентов: V_{sig} и V_{var}. V_{var} – операндовые вершины типа «переменная», перезаписывающее свое значение при обновлении (подобно типичным переменным в языках программирования) и V_{sig} – операндовые вершины регистрового типа «сигнал» (например, типы **signal** и **port** в VHDL). E_i состоит из двух типов вершин. $E_{transfer}$ – набор вершин, соответствующих простой передаче данных (например, оператор назначения сигнала в VHDL), а $E_{function}$ – набор вершин, соответствующих двух или многоместным функциям-операторам. L_i – набор меток. Причем для T_i справедливо следующее выражение: T_i принимает либо 0, либо L_i. Второй граф задается выражением $G_c = (C_c, L_c, T_c)$, где V_c – набор вершин-условий L_c – набор вершин-меток, T_c – набор связей между метками и условиями. Набор меток L_i и набор вершин-меток L_c связывают два графа меду собой.

Информационный I-граф описывает поток данных и их преобразование (подобно операционному автомату в классической композиционной модели с микропрограммным управлением) без учета условных ветвей. Управляющий граф соответствует цепочке условий, при выполнении которых выполняется тот или иной оператор.

Как было отмечено, I-граф содержит два типа вершин: операнды и функции. Типы операндов: целые числа и беззнаковые вектора. Типы функций ограничены синтезируемым подмножеством HDL (то есть теми конструкциями, которые имеют физические эквиваленты в системах синтеза и имплементации при подготовке проекта к реализации на ПЛИС). Список синтезируемых конструкций может изменяться в зависимости от версии подсистемы синтеза. Дуги соединяют вершины следующим образом: вершина, соответствующая входному операнду и ставшая источником, соединяется с функциональной вершиной, затем дуга выходит из функциональной вершины и входит в вершину, соответствующую выходному операнду и ставшую таким образом приемником. Дуги, исходящие из функциональных вершин и входящие в вершины-приемники, могут быть условные и безусловные. Условные дуги соответствуют операторам, которые находятся внутри условных конструкций HDL. Условные дуги содержат метки, кодирующие условие срабатывания перехода по дуге.

В свою очередь, второй граф (управляющий C-граф) содержит условные конструкции (например, **case**, **if**...**then**..., **with** … **select**) из исходного описания ЦУ. C-граф – это граф с 2-мя типами вершин: условия и метки. Вершины условий содержат вычисляемые условия. Вершины меток – конечные, не имеющие входной дуги. Эти вершины содержат имя метки. Результат моделирования C-графа – набор меток (метка), по которым необходимо сделать переходы в I-графе. Например, выражение **if** reset=1 **then**… представляется следующим образом. Имя выражения – *E1*, имя метки – *L1*.

Заметим, что если полученная модель имеет обратные связи (ОС), то это ухудшает качество диагностирования, так как локализация ошибки осуществляется, как правило, до группы операторов, охваченных циклом ОС. Поэтому для достижения необходимой глубины диагностирования (до оператора HDL–кода) необходимо иметь «плоскую» модель, не содержащую обратных связей. Определим принципы разрыва обратных связей. Для этого учтем свойство параллельности операндов, имеющих тип «сигнал», определяющее, что новые значения записываются в них после окончания всего цикла моделирования. Таким образом, на каждом временном интервале (до изменения значений сигналов) каждый фрагмент HDL-кода можно рассматривать как «плоский» граф. Каждый операнд, задаваемый ключевыми словами **signal** и **port**, будет представляться своей операндовой вершиной, но с таким же именем.

Рассмотри модель JK триггера на языке VHDL.

```
entity JK is port (QN, RN, J, K, CLK: in bit;   Q: out bit; SN: out bit); end JK;
architecture JK of JK is begin
signal QN: bit;
 process (SN, RN, CLK) begin
        if RN = '0' then Q<= '1';  elsif SN = '0' then Q<= '0';
         elsif CLK = '0' and CLK'event then Q <= (J and not Q) or (not K and Q);
end if; end process ;
QN <= not Q;
end JK;
```

Управляющий граф при разрыве ОС не будет подвержен изменениям (рис. 6). Каждая i-ая вершина E соответствует предикатному выражению, найденному в условных конструкциях.

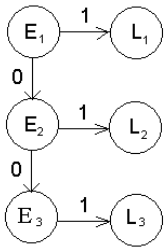

Рис. 6. Управляющий граф на основе VHDL-модели JK триггера

До преобразования информационный граф содержит обратные связи. Вершины с 0 и 1 соответствуют константным значениям, обнаруженным в коде.

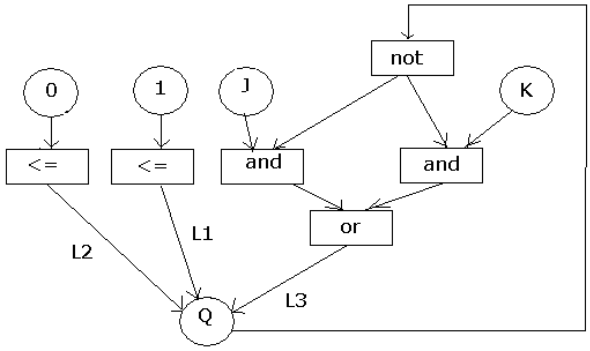

Рис. 7. Информационный граф на основе VHDL-модели JK триггера

Преобразованный граф без обратных связей выглядит так (рис. 8). Все элементы в правой части операторных выражений являются входными вершинами, не имеющими предшественников.

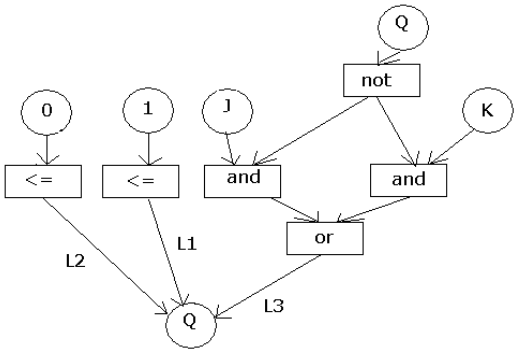

Рис. 8. Преобразованный информационный граф

Теперь рассмотрим присутствие «обычных» переменных в коде. От сигналов они отличаются тем, что информация об прошедших изменениях значения не записывается, а новое значение моментально «затирает» старое и обновляет содержимое области памяти, выделенное под переменную в системе моделирования. Зачастую переменные не имеют физического эквивалента за исключением тех случаев, когда созданы условия для генерации триггерных структур. Однако эти отличия существенны только на этапе исправного моделирования и синтеза в соответствующих системах и не влияют на визуальное отображение процесса обработки данных. Поэтому различия между переменными и сигналами будут только при программной реализации системы диагностирования. Приведем пример преобразования кода, содержащего переменную, в граф.

```
entity shema2v is port( x1, x2, x3, x4 : in STD_LOGIC;  q : out STD_LOGIC); end shema2v;
architecture shema2v3 of shema2v is begin
        process  (x1,x2,x3,x4) is
        variable s5,s6,s7,sq: std_logic;      begin
                s5:= (x2 nor s6);             s6:= (x1 nand s5);
                s7:= (x3 nand x4);            sq:= (s6 or s7);
                q<=sq;   end process; end shema2v3;
```

Явно видна линия обратной связи с выхода элемента И с инверсией по одному входу на вход элемента ИЛИ с инверсией по одному входу. Соответствующая схема после синтеза:

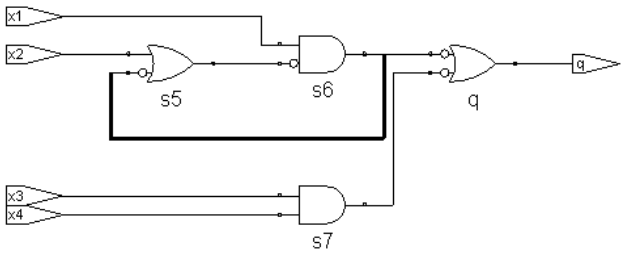

Рис. 9. Результат синтеза

Преобразование в информационный граф без обратных связей приведет к следующей визуальной модели.

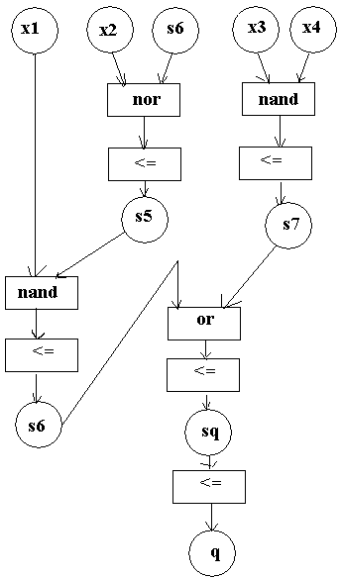

Рис.10. Информационный граф без обратных связей

Разрыв обратных связей позволяет применить структурные методы диагностирования на основе многозначных таблиц неисправностей и матриц достижимостей. «Плоская» модель позволяет легко выделить одновыходовые подграфы для обеспечения глубины анализа равной одному объекту графовой

модели. Предложенная модель решает проблему построения тестов, т.к. обеспечивает проведение импликации как вперед, так и назад, что невозможно выполнить по исходной языковой модели.

Выводы. Предложенная модель позволяет использовать HDL-описание в качестве объекта диагностирования, т.к. уровень декомпозиции позволяет получить глубину анализа равную одному оператору, а в отдельных случаях до одного операнда – источника или приемника, – что на 100% решает задачу обеспечения наблюдаемости внутренних точек модели. Кроме того, предложенная модель решает проблему, собственно, построения тестов, т.к. обеспечивает проведение импликации как вперед, так и назад, что невозможно было выполнить по исходной языковой модели. Модель неисправностей механизма обработки данных соответствует ошибкам проектирования в I-графе. Ошибка проектирования VHDL-кода типа «замена» вызвана ошибкою в употреблении операторов назначения или присваивания сигналов и переменных. То есть f_i вместо f_j (или f_i / f_j). Под f понимается оператор языка проектирования из синтезируемого подмножества. Модель неисправностей механизма активизации операции соответствует ошибкам проектирования в C-графе. Предложенная модель ошибки проектирования обобщает экспериментальные сведения, полученные от производителей, касающиеся типов ошибок и их влияния на модель в целом. Таки образом, ошибки, связанные с «человеческим» фактором и результирующиеся в неверном (несвоевременном или ошибочном) употреблении того или иного оператора или группы операторов занимают второе место по частоте.

ЛИТЕРАТУРА:

1. Таранов В.Б., Сыревич Е.Е., Чегликов Д.И., Зинченко Д.Ю., Карасев А.Л. Применение методов поиска дефектов при верификации моделей цифровых устройств на vVHDL // Весник ХНТУ. – 2008. – № 1(30). – С. 335-341.
2. Шкиль А.С., Сыревич Е.Е., Карасев А.Л., Чегликов Д.И. Тестовая верификация поведенческих языковых моделей цифровых устройств // Автоматизированные системы управления и приборы автоматики. – 2006. – Вып. 134.– С. 4-12.
3. Чегликов Д.И., Шкиль А.С., Зинченко Д.Е. Реализация процедур импликации на графовой структуре // Радиоэлектронные компьютерные системы. –2006. – Вып. 6. – С.153-157.

АЛЬМАДХОУН Самер – аспирант кафедры автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники (ХНУРЭ).

Научные интересы: диагностирование и поиск дефектов ЦУ, верификация HDL-моделей.

СЫРЕВИЧ Евгения Ефимовна – к.т.н., доцент кафедры автоматизации проектирования вычислительной техники ХНУРЭ.

Научные интересы: верификация HDL-моделей, проектирование на ПЛИС. Увлечения: иностранные языки.

ШКИЛЬ Александр Сергеевич – к.т.н., доцент кафедры автоматизации проектирования вычислительной техники ХНУРЭ.

Научные интересы: верификация HDL-моделей, системы тестирования. Увлечения: технологии дистанционного образования.