ИНФОРМАТИКА

2007 **январь-март** **№ 1**

УДК 519.718.4

А.А. Иванюк

МОДЕЛИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ НЕИСПРАВНОСТЕЙ ЦИФРОВЫХ УСТРОЙСТВ СРЕДСТВАМИ ЯЗЫКА VHDL

Описывается методика внедрения моделей функциональных неисправностей в проектные описания цифровых устройств на языке VHDL. Рассматриваются вопросы, связанные с распределением неисправностей в процессе моделирования. Предлагается использовать верификационные компоненты для проектирования средств самотестирования цифровых устройств. Обосновывается применение предлагаемой методики.

Введение

Стремительный рост рынка средств вычислительной техники требует от разработчиков ускорения процесса проектирования, верификации и изготовления цифровых компонент. Современные системы автоматизированного проектирования позволяют использовать различные подходы для создания цифровых проектов. Среди всего многообразия методов проектирования отчетливо выделяются языковые средства, которые обладают неоспоримыми преимуществами по сравнению с классическими методами. Наличие средств логического и физического синтеза, систем функционального и параметрического моделирования позволяет максимально автоматизировать процесс верификации цифровых проектов и тем самым изготавливать устройства с достоверным функционированием. Однако соблюдение всех норм и требований технических заданий, использование проверенных методов синтеза и технологий изготовления не позволяет судить о надежности реализованного устройства при его функционировании по назначению. Важнейшим фактором, который негативно влияет на достоверность работы цифровых устройств, является наличие физических дефектов в полупроводниковых кристаллах [1]. Тип дефекта зависит как от технологии изготовления устройства, так и от условий его эксплуатации. Существующее многообразие математических абстракций физических дефектов представляется обширным классом функциональных неисправностей, которые используются для разработки эффективных тестов для их обнаружения [1, 2].

В настоящей работе предлагается использовать языковые VHDL-средства для описания функциональных моделей неисправностей с последующим их внедрением в существующие проекты цифровых устройств. Показано, что представленная методика применения внедренных описаний неисправностей может быть успешно применена для анализа поведения моделей проектируемых устройств и для разработки встроенной аппаратуры самотестирования.

1. Языковые средства описания цифровых проектов

Современные технологии и системы проектирования позволяют создавать цифровые компоненты произвольной сложности: от простейших комбинационных схем до специализированных многоядерных микропроцессоров и систем на кристалле (СнК). При этом все чаще используется методология нисходящего проектирования, при которой применяется последовательная декомпозиция абстрактной модели уровня системного представления устройства [3]. Если классическое нисходящее проектирование доходит до уровня физической топологии устройства, то современные подходы, как правило, ограничиваются уровнем представления регистровых передач (RTL, Register Transfer Level) стандартных цифровых узлов [4]. Наличие высокоуровневых языковых средств позволяет проектировать цифровые устройства при помощи функциональных и структурных абстракций. Так, язык VHDL обладает возможностью описать структуру устройства на системном, RTL- и вентильном уровнях [4, 5]. В то же время он позволяет использовать поведенческое описание цифровой компоненты на вышеперечисленных уровнях. Результат логического синтеза языковых абстракций представляет собой схемотехни-

ческое описание будущего устройства в базисе RTL-примитивов или логических вентилей. При этом качественные и количественные характеристики результата синтеза зависят не только от программных компиляторов, но и от типа исходного языкового описания и от индивидуального стиля разработчика.

Языковые описания цифровых компонент являются технологически независимыми, переносимыми и повторно используемыми. Разработанная один раз VHDL-компонента может быть повторно применена в качестве библиотечного элемента в другом проекте. Такие компоненты называются IP-компонентами (Intellectual Property), которые принято классифицировать на стандартные и заказные [2, 6, 7]. Примером стандартной IP-компоненты может служить настраиваемое VHDL-описание двоичного синхронного счетчика, для которого задается его разрядность, вид счета и тип синхронизации. Подобная программная компонента описывается на синтезируемом подмножестве языка VHDL, является технологически независимой и может быть применена в различных цифровых проектах. В свою очередь, законченное проектное VHDL-описание нестандартного специализированного устройства представляет собой заказную IP-компоненту, которая после незначительной доработки может быть настраиваемой и повторно используемой.

Среди всего многообразия IP-компонент выделяют так называемые аппаратные IP-компоненты, представляющие собой синтезированные описания цифровых проектов для конкретной технологии их реализации, которые могут быть оптимизированы по таким критериям, как аппаратурные затраты, быстродействие либо потребляемая энергия. Разработчик имеет доступ к аппаратным IP-компонентам только на интерфейсном уровне и не имеет возможности видоизменять их внутренние описания.

В настоящее время мировой рынок цифровых компонент вырос настолько, что проектирование полнофункциональных систем на кристалле может осуществляться только при помощи имеющихся программных и/или аппаратных IP-компонент. Помимо высокоуровневых языковых описаний, они могут представлять собой синтезированные RTL-описания либо описания межсоединений стандартных библиотечных элементов. При этом потребитель IP-компонент сам решает задачи синтеза, размещений и трассировки при проектировании своего цифрового устройства.

Рассмотрим один из сценариев, по которому устройство типа СнК проектируется на основе IP-компонент. При использовании высокоуровневого языка описания цифровой аппаратуры VHDL законченный проект представляет собой компоненту высшего системного уровня иерархии. Подобная компонента имеет структурное описание, элементами которого являются сторонние IP-компоненты или собственные компоненты, разработанные проектировщиком, которые объединяются посредством карт межсоединений (**port map**) и сигналов (**signal**). В свою очередь, IP-компоненты могут быть спроектированы посредством поведенческого или структурного VHDL-описания, содержащего собственные либо сторонние компоненты в зависимости от степени сложности разрабатываемого устройства. Следует отметить, что многие разработчики используют смешанный стиль, при котором компонента описывается как на структурном, так и на поведенческом подмножестве языка VHDL.

При этом поведенческое описание может быть реализовано в качестве псевдоструктурного описания, в котором функционирование внутренних компонент определяется при помощи параллельных конструкций **process**, а их взаимосвязь – посредством сигналов и глобальных переменных (**shared variable**) (рис. 1).

Таким образом, основным синтезируемым языковым объектом, который участвует в межсоединениях IP-компонент на различных уровнях иерархии, является **signal**. Как в случае использования программных IP-компонент, так и в случае аппаратных IP-компонент **signal** является наиболее адекватным представлением межсоединительных сигнальных линий реализованного цифрового устройства.

Отсутствие каких-либо жестких критериев и ограничений оставляет право за разработчиком выбирать тот или иной стиль при описании цифровых устройств. Учитывая приведенные особенности использования языка VHDL, рассмотрим некоторые подходы для внедрения моделей функциональных неисправностей в проект цифрового устройства.

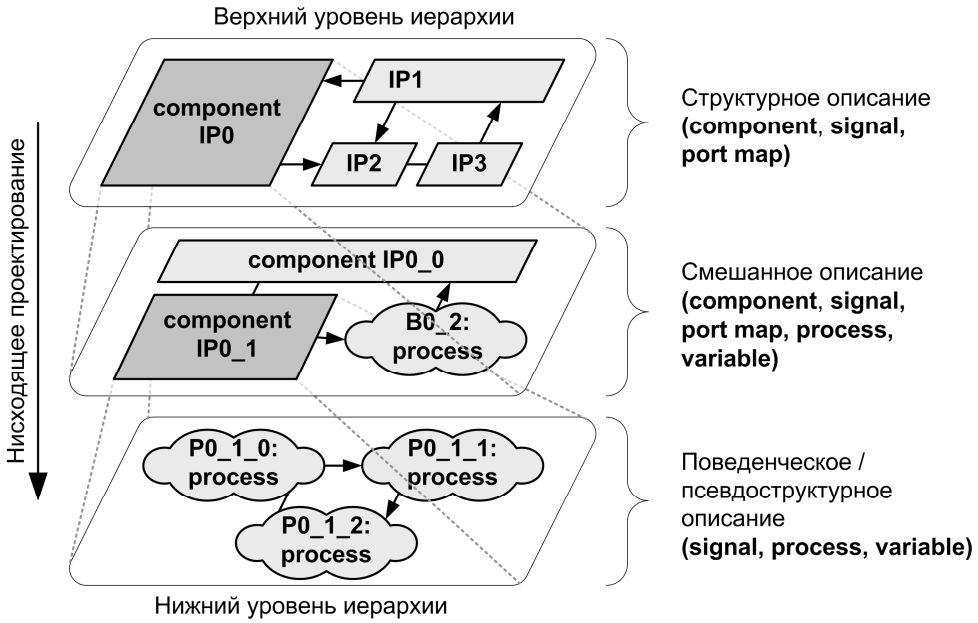


Рис. 1. Иерархическое представление VHDL-описания цифрового устройства

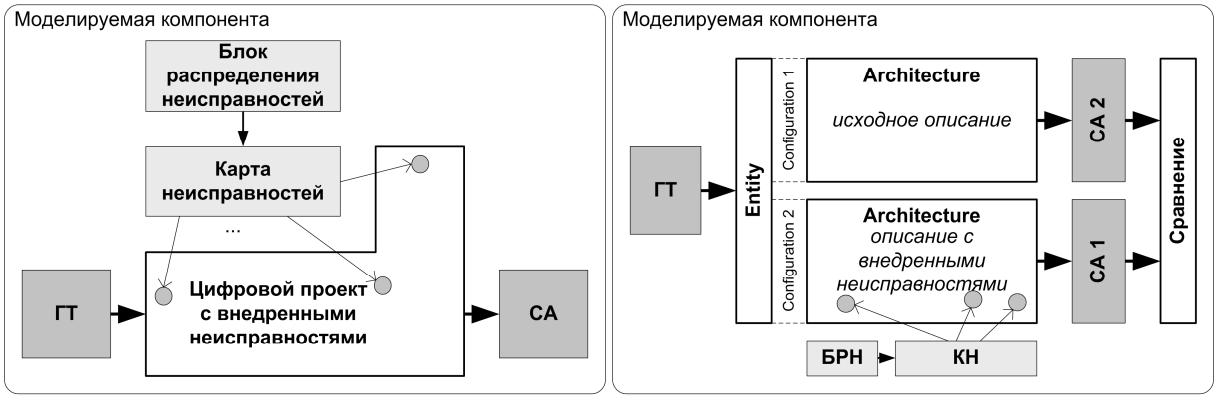
2. Методы описания и внедрения функциональных неисправностей

Для корректного VHDL-описания цифрового устройства внедрение моделей функциональных неисправностей означает изменение поведения проектной модели. Такие изменения могут быть осуществлены как в поведенческом, так и в структурном представлении устройства [8, 9] и заключаются в применении нескольких подходов. Один из подходов использует замену исходного описания на описание, которое представляет некорректное функционирование компоненты согласно применяемой модели неисправности [10]. Еще одним примером может служить метод, который управляет значениями сигналов и переменных в процессе моделирования цифровой компоненты, не изменяя исходного VHDL-описания [11]. В работе [12] описана программная среда, которая является надстройкой над языком VHDL и позволяет вносить модели неисправностей в цифровой проект на различных уровнях абстракции. В отличие от существующих подходов, в данной статье предлагается новая методика внедрения и моделирования функциональных неисправностей, которая использует языковые средства VHDL и не нуждается в дополнительных программных средствах. Основная идея предлагаемой методики заключается в дополнении исходного языкового проекта управляемыми моделями функциональных неисправностей, которые могут быть прозрачными для средств логического синтеза. Помимо внедряемых неисправностей методика подразумевает использование VHDL-описания таких блоков, как генератор тестов (ГТ), схема анализа выходных реакций (СА), блок распределения неисправностей (БРН) и карта неисправностей (КН) (рис. 2, *а*).

КН представляет собой структуру данных, содержащую такие сведения, как тип неисправности, ее местоположение и статус (активна/не активна). Компонента БРН может быть использована для распределения местоположения, кратности и типов неисправностей при моделировании. Компоненты ГТ и СА необходимы для автоматизации процесса подачи тестовых воздействий на цифровой проект с внесенными неисправностями и анализа его реакций. Если к моделируемой компоненте добавить исходное описание цифрового проекта, которое не содержит внедренные модели неисправностей, то подобная модификация может служить для проверки качества применяемых тестов и достоверности используемых алгоритмов анализа реакций (рис. 2, *б*). При успешно проведенном процессе моделирования, в результате которого был выбран необходимый тест, синтезируемые компоненты ГТ и СА могут быть применены для реализации встроенных средств самотестирования.

Согласно рассмотренным иерархическим описаниям проектируемых цифровых компонент внедряемые модели функциональных неисправностей могут быть описаны в качестве не-

корректных значений сигналов. Языковой объект **signal** синтезируемых типов описывает поведение реальных проводящих линий, которые связывают между собой различные аппаратные и программные IP-компоненты. К основным функциональным моделям неисправностей проводящих линий относят: константные неисправности SAF (Stuck-At Faults) и неисправности типа обрыв SOF (Stuck-Off Faults) [2].



а) *б)*

Рис. 2. Структура моделируемой компоненты

Представленный перечень моделей неисправностей является базовым и может быть применен для описания более сложных видов неисправностей, таких, например, как мостиковые неисправности (Bridging Faults) [2]. Неисправности типа SAF характеризуются тем, что логическое значение на входной или выходной линии цифровой компоненты остается все время постоянным. Различают неисправности типа константного нуля (SA0) и неисправность типа константной единицы (SA1). Неисправности типа обрыв обычно описываются наличием высокоомной нагрузки на входной или выходной линии компоненты, что эквивалентно отключению порта от сигнальной линии.

Внедрение моделей неисправностей в цифровой проект должно быть управляемым и настраиваемым под различные ситуации моделирования. Языковые VHDL-модели неисправностей должны отвечать таким требованиям, как прозрачность для средств логического и физического синтеза, возможность задания типа неисправности, ее местоположения и кратности, способность к автоматизации процесса моделирования цифрового проекта. Исходные данные для внедрения неисправностей могут быть описаны в структуре данных, которая называется картой неисправностей. Так, для приведенных моделей функциональных неисправностей предлагается следующий подход. Пусть карта неисправностей представляет собой строку, элементами которой являются четырехзначные символы. Размерность строки соответствует количеству потенциальных сигналов цифрового проекта, для которых могут быть внедрены модели неисправностей. Каждый символ строки принимает следующие значения: «-», «0», «1», «Z», которые соответственно определяют отсутствие неисправности, наличие неисправностей SA0, SA1 или SOF.

Рассмотрим пример создания КН для VHDL-описания цифровой компоненты, имеющей четыре сигнальных порта (рис. 3). Предположим, что неисправности равновероятно могут возникать на всех четырех портах компоненты, соответственно значение кратности неисправности одного типа может варьироваться от 1 до 4. Исходя из этого, определим КН FAULT_MAP в виде четырехкомпонентного символьного вектора, элементы которого могут принимать вышеописанные значения. Тогда вектор FAULT_MAP = «----» соответствует отсутствию неисправностей, в то время как значения FAULT_MAP = «---1» и FAULT_MAP = «0-0-» определяют наличие неисправности SA1 на первом порте и двукратную неисправность SA0 на втором и четвертом портах цифровой компоненты.

Языковое описание цифровой компоненты имеет следующие варианты:

– поведенческое описание при помощи операторных блоков **process**;

– поведенческое описание посредством параллельных операторов блока **architecture**;

– структурное описание с использованием декларативных блоков **component**.

Для всех перечисленных случаев корректное внедрение неисправности должно заключаться в перехвате всех входных и выходных сигналов, которые подвержены изменениям.

Модификации поведенческого описания являются не всегда корректными и могут обладать непредсказуемыми результатами моделирования и синтеза. Например, изменение VHDL-кода блока **process** с целью внедрения неисправности может быть ошибочным, а порой и невозможным ввиду индивидуальных стилей проектировщиков. Так, игнорирование второй ветки альтернативного условия **else** в конструкции **if-then-else** внутри блока **process** может привести к синтезу элемента памяти, либо изменение полной конструкции **if-then-else** может привести к синтезу буфера с тремя состояниями вместо комбинационной схемы мультиплексора [13]. В связи с этим предлагается модели неисправностей разрабатывать в виде отдельной компоненты, которая определенным образом изменяет поведение перехватываемого сигнала.

Моделируемая компонента

```
entity FComponent is
    port ( A : in  std_logic;
           B : in  std_logic;
           C : in  std_logic;
           F : out  std_logic);
end FComponent;

architecture Beh of FComponent is
```

Компонента перехвата

```
component FAULT
port( SIN  : in  std_logic;
     SOUT  : out std_logic;
     FType : in  character);
end component;
```

Карта неисправностей

```
constant FAULT_MAP: string (4 downto 1):="----";
```

Сигналы перехвата

```
signal Ax,Bx,Cx,Fx:std_logic;

begin
F1:FAULT port map (A,Ax,FAULT_MAP(1));
F2:FAULT port map (B,Bx,FAULT_MAP(2));
F3:FAULT port map (C,Cx,FAULT_MAP(3));
```

Исходное описание компоненты

```
MAIN:process(Ax,Bx,Cx)
begin
...
end process;

F4:FAULT port map (Fx,F,FAULT_MAP(4));
end Beh;
```

Компонента перехвата

```
entity FAULT is
    port ( SIN   : in  std_logic;
           SOUT  : out std_logic;
           FType : in  character);
end FAULT;

architecture Beh of FAULT is
begin
MAIN: process(SIN,FType)
begin
       FCASE: case FType is
       -- Неисправность отсутствует
       when '-' =>   SOUT<=SIN;
       -- SA0-неисправность
       when '0' =>   SOUT<='0';
       -- SA1-неисправность
       when '1' =>   SOUT<='1';
       -- SOF-неисправность
       when 'Z' =>   SOUT<='Z';
       -- Неисправность отсутствует
       when others => SOUT<=SIN;
       end case;
end process;
end Beh;
```

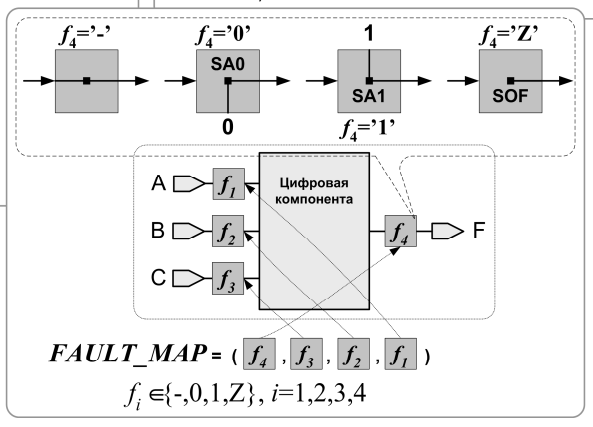


Рис. 3. Результат внедрения карты и моделей неисправностей

Следует отметить, что компонента FAULT спроектирована на синтезируемом подмножестве VHDL и представляет собой описание мультиплексора «четыре к одному». Подобное свойство может быть использовано для имитации неисправностей на реализованном прототипе устройства. В то же время компонента FAULT может быть прозрачной для син-

теза исходного описания цифрового устройства посредством задания входного аргумента FType = «-».

3. Средства моделирования VHDL-описаний с внедренными неисправностями

Для проверки поведения модели цифрового устройства с внедренными неисправностями можно использовать несколько подходов. Стандартный подход заключается в применении тестовых компонент TestBench, которые являются языковой надстройкой над моделируемой компонентой. Различают два основных типа TestBench-модулей [14]:

1) модуль, получающий выходные реакции моделируемой компоненты, которые затем анализируются посредством программных пакетов моделирования (редактора временных диаграмм, символьного отладчика и т. п.);

2) модуль, реализующий анализ выходных реакций моделируемой компоненты путем сравнения с ожидаемыми (эталонными) значениями.

Оба представленных типа могут автоматизировать процесс посредством использования системных сообщений и команд прерывания моделирования. Использование второго типа TestBench-модуля часто представляет собой реализацию предсказуемых выходных значений для контрольных состояний простейших цифровых компонент. Если проектируемая компонента является сложным устройством, то наилучшим способом будет использование второго типа TestBench-модуля, который генерирует детерминированный набор входных значений, при этом выходные наборы в целях экономии ресурсов моделирования сжимаются в короткие сигнатуры. Полученные значения сигнатур сравниваются с заранее вычисленными эталонными значениями.

Представленный вид TestBench-модуля может быть использован для моделирования цифровых компонент с внедренными неисправностями. Для этого из представленной на рис. 2, *б* структуры могут быть исключены компоненты, содержащие исходное VHDL-описание и схему анализа CA2, вместо которых используется описание константного значения эталонной сигнатуры устройства, при этом описание CA1 представляет собой VHDL-код алгоритма сжатия выходных наборов. Вычисление значения эталонной сигнатуры реализуется при моделировании цифрового проекта, для которого все внедренные неисправности являются пассивными. Если VHDL-описания компонент ГТ и CA1 являются синтезируемыми, они могут быть использованы для проектирования встроенных средств самотестирования будущего цифрового устройства (рис. 4). Например, схемы ГТ могут представлять собой генераторы псевдослучайных тестовых последовательностей, а схемы СА – многоканальные сигнатурные анализаторы либо иные схемы получения компактных оценок [15]. Представленный на рис. 4 исходный VHDL-код TestBench-модуля содержит синтезируемые описания таких цифровых устройств, как генератор тестов, одноканальный сигнатурный анализатор и схема сравнения сигнатур, которые без изменений могут быть использованы для проектирования встроенной аппаратуры самотестирования.

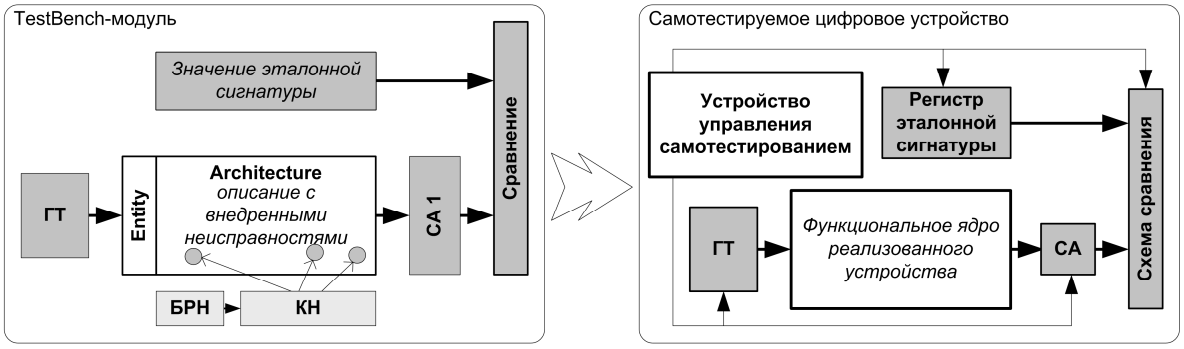
Определение достоверности выбранных генераторов тестов и схем анализа реакций подразумевает моделирование цифровой компоненты с использованием конкретного сценария распределения неисправностей. Для этого введем следующие обозначения: m – общее число сигналов, для которых внедряются модели неисправностей; n – число сигналов, для которых моделируются активные неисправности; r – количество различных типов неисправностей. Тогда можно выделить два сценария:

1) моделирование всех возможных r^n типов неисправностей для $n = m$ сигналов;

2) моделирование всех возможных $C_m^n \cdot r^n$ типов неисправностей для всех $n < m$ сигналов.

Первый сценарий приемлем для малого количества сигнальных линий и типов моделируемых неисправностей. Второй сценарий предполагает моделирование определенного множества всевозможных комбинаций из n неисправностей на m сигнальных линиях и может быть применен для проверки достоверности разрабатываемого метода при обнаружении всех типов неисправностей кратности, не превышающей значение n. Оба сценария являются приемлемыми для относительно небольшого количества проводящих линий, которые участвуют в межсоеди-

нениях аппаратных IP-компонент. Например, в цифровом проекте, реализующем микропроцессорную систему, значение m не превышает нескольких сотен. Для возможности осуществления всех приведенных сценариев предлагается следующая модель распределителя неисправностей, которая состоит из генератора псевдослучайных чисел (ГПСЧ), генератора векторов определенного веса (ГВОВ) и собственно генератора карты неисправностей (ГКН).



VHDL-описание TestBench-модуля

```
entity BIST_VHD is
end BIST_VHD;

architecture Beh of BIST_VHD is
--Компонента с внедренными неисправностями
component fcomponent
port(
         a,b,c : in std_logic;
         f     : out std_logic);
end component;
-- Сигнал синхронизации
signal   clk    : std_logic:='0';
-- Сигнал ошибки (сигнатуры не равны)
signal   err    : std_logic:='0';
-- Асинхронный сигнал инициализации
signal   rst    : std_logic;
-- Сигнал, подключаемый к выходному порту
-- тестируемой компоненты
signal   f      : std_logic;
-- Сигналы обратных связей
signal   fb1,fb2: std_logic;
-- Сигнал для описания генератора тестов (ГТ)
signal   lfsr   : std_logic_vector(2 downto 0);
-- Сигнал для описания сигнатурного анализатора (СА)
signal   sisr   : std_logic_vector(3 downto 0);
-- Значение эталонной сигнатуры
constant sig    : std_logic_vector(3 downto 0):="0100";
-- Значение половины периода сигнала синхронизации
constant hp     : time:=20 ns;
-- Сигнал для описания двоичного счетчика
signal   cnt    : std_logic_vector(2 downto 0):="000";

begin
-- Процесс выработки управляющих сигналов
ctrl:process
begin
         rst<='1';
         wait for hp;
         rst<='0';
         for i in 1 to 4 loop
                  wait for hp; clk<='1';
                  wait for hp; clk<='0';
                  cnt<=cnt+1;
         end loop;
         wait for 2*hp;
end process;
```

```
-- Подключение тестируемой компоненты
uut: FComponent port map
(lfsr(0),lfsr(1),lfsr(2),f);
-- Описание генератора тестов
fb1<=lfsr(0) xor lfsr(2);
tpg:process(clk,rst,fb1,lfsr)
begin
         if rst='1' then
                  lfsr<="001";
         elsif clk'event and clk='1' then
                  lfsr<=lfsr(1 downto 0) & fb1;
         end if;
end process;
-- Описание сигнатурного анализатора
fb2<=sisr(0) xor sisr(3) xor f;
sa: process(clk,rst,fb2,sisr)
begin
         if rst='1' then
                  sisr<="0001";
         elsif clk'event and clk='0' then
                  sisr<=sisr(2 downto 0) & fb2;
         end if;
end process;
-- Описание схемы сравнения сигнатур
cmp:process(cnt,sisr,rst)
begin
         if rst='1' then
                  err<='0';
         else
                  if cnt="100" then
                           if sisr=sig then
                                    err<='0';
                           else
                                    err<='1';
                           end if;
                  end if;
         end if;
end process;
end Beh;
```

Рис. 4. Повторное использование компонент TestBench-модуля

Рассмотрим реализацию блока распределения неисправностей для случая описанных выше типов неисправностей ($r = 3$) и примера на рис. 3 ($m = 4$). В качестве схемы ГПСЧ вы-

берем LFSR-структуру, построенную на основе четырехразрядного сдвигового регистра с возможностью определения его начального состояния и примитивного неприводимого полинома, определенного над конечным полем третьего порядка GF(3). По пришествии очередного входного такта синхронизации ГПСЧ вырабатывает последующее четырехразрядное выходное значение. В двоичном схемотехническом представлении каждый выход ГПСЧ состоит из двух линий, которые кодируют троичные значения следующим образом: 0 – «01», 1 – «10», 2 – «11». В свою очередь, представленные три значения будут использованы для задания трех типов неисправностей: SA0, SA1 и SOF. Так, первый троичный выход ГПСЧ кодирует модель неисправности для первого сигнала f_1 (см. рис. 3). Аналогичным образом используются остальные три выхода ГПСЧ для задания типов неисправностей для сигналов f_2, f_3 и f_4. Схема ГВОВ способна вырабатывать все возможные четырехразрядные двоичные значения определенного веса ($1 \le n \le 4$). Генератор карты неисправностей выбирает лишь те пары выходных значений ГПСЧ, которым соответствуют единичные выходные значения схемы ГВОВ (рис. 5). Каждая пара при этом кодируется одним из четырех символьных значений КН.

Генератор векторов определенного веса

```
entity GC42 is
       port ( CLK,RST : in  STD_LOGIC;
              OUTPUT  : out STD_LOGIC_VECTOR(3 downto 0));
end GC42;

architecture behavioral of GC42 is
begin
MAIN: process(CLK,RST)
variable x: std_logic_vector(3 downto 0):="0011";
variable a0: integer:=0;
variable a1: integer:=1;
begin
if RST='1' then
       x:="0011"; a0:=0; a1:=1;
elsif CLK'event and CLK='1' then
       if a0<2 then
              if a1<3 then
                     x(a1):='0'; a1:=a1+1; x(a1):='1';
              else
                     x(a0):='0'; x(a1):='0';
                     a0:=a0+1; a1:=a0+1;
                     x(a0):='1'; x(a1):='1';
              end if;
       end if;
end if;
OUTPUT<=x;
end process;
end behavioral;
```

	OUTPUT 3	2	1	0
...	x	x	x	x
1	0	0	1	1
2	0	1	0	1
3	1	0	0	1
4	0	1	1	0
5	1	0	1	0
6	1	1	0	0
7	1	1	0	0
...	.	.	.	.
P	0	0	1	1

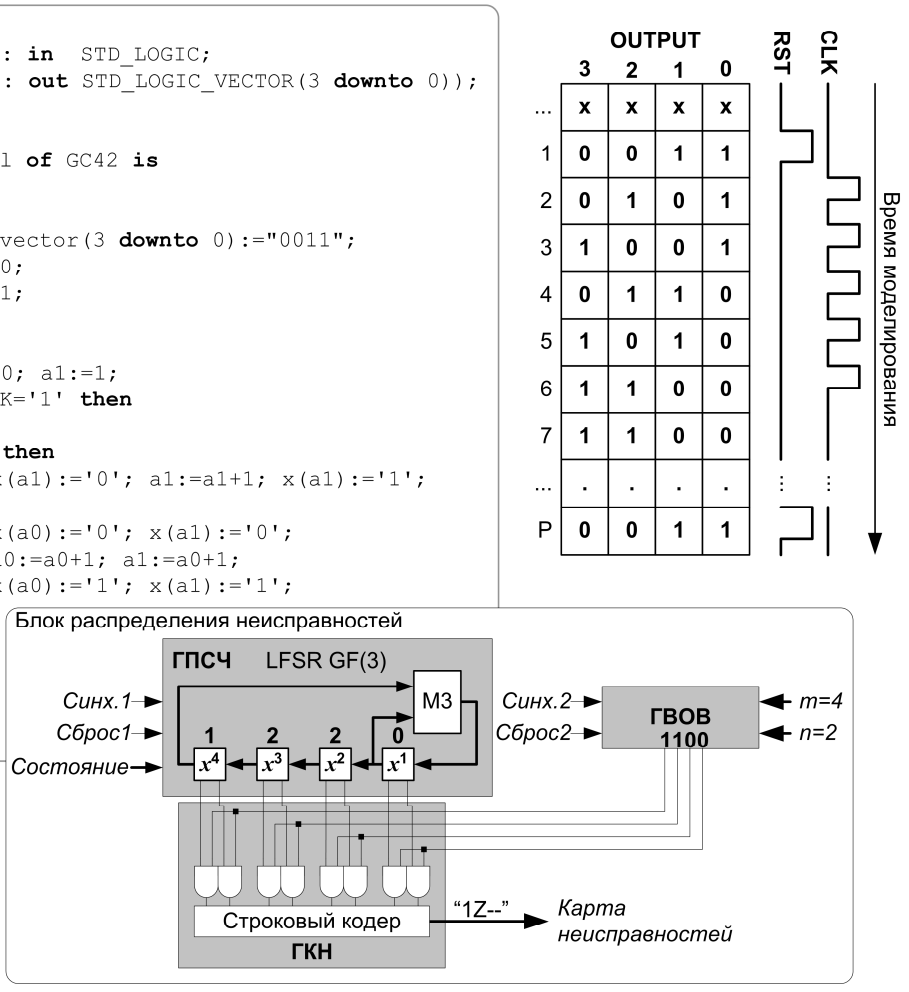


Рис. 5. Структура блока распределения неисправностей и VHDL-код схемы ГВОВ

Вырабатываемое ГКН строковое значение является текущей картой неисправностей для моделируемого цифрового устройства. Приведенная структура позволяет использовать три сценария моделирования неисправностей посредством подачи соответствующих последовательностей синхроимпульсов *Синх. 1*, *Синх. 2*, значений *n* и начального состояния ГПСЧ. Так,

первый сценарий для $r = 3$ может быть реализован установкой значения $n = 4$ и последовательной подачей $3^4 - 1 = 80$ синхроимпульсов на вход *Синх. 1*.

Второй сценарий заключается в генерировании всех C_4^n комбинаций 3^n неисправностей. При этом необходимо для каждой выходной комбинации схемы ГВОВ выработать количество наборов, не превышающее значение половины периода ГПСЧ. Экспериментально было установлено, что минимальное количество синхроимпульсов *Синх. 1*, необходимых для выработки всех возможных значений кодов неисправностей при выходном значении «0101» схемы ГВОВ, равно 31, а для значения «1001» – 25.

Разработанная структура БРН позволяет моделировать один выбранный тип неисправности заданной кратности. Для этого в ГПСЧ записывается начальное значение, соответствующее типу неисправности (например, «0000» для SA0). Затем для определенного значения n на схему ГВОВ подается необходимое количество синхронизирующих импульсов.

Представленный код может быть легко модифицирован для произвольной разрядности m и незначительно переделан для произвольных значений $n \le m$.

Заключение

В статье был рассмотрен один из подходов к моделированию функциональных неисправностей цифровых устройств. Показано, что для современных цифровых проектов наиболее приемлемым является внедрение неисправностей сигнальных линий. Предложен подход, который совмещает верификацию цифровых проектов с разработкой аппаратуры встроенных средств самотестирования. Приведена методика, воспроизводящая несколько сценариев распределения неисправностей при моделировании цифровых компонент.

Список литературы

1. Lala, P.K. Digital Circuits Testing and Testability / P.K. Lala. – New York: Academic Press, 1997. – 199 p.

2. Stroud Charles, E. A Designer's Guide to Built-In Self-Test / E. Stroud Charles. – Norwell: Kluwer Academic Publishers, 2002. – 319 p.

3. Rajsuman, R. System-on-a-Chip. Design and Test / R. Rajsuman. – Boston: Artech House Publishers, 2000. – 277 p.

4. Бибило, П.Н. Синтез логических схем с использованием языка VHDL / П.Н. Бибило. – М.: СОЛОН-Р, 2002. – 384 с.

5. Hwang, E.O. Digital Logic and Microprocessor Design with VHDL / E.O. Hwang. – Brooks/Cole, 2005. – 513 p.

6. Core design and system-on-a-chip integration / A.M. Rincon [et al.] // IEEE Design and Test of Computers. – 1997. – № 14. – P. 26–35.

7. Hunt, M. Blocking in a system on a chip / M. Hunt, J.A. Rowson // IEEE Spectrum. – 1996. – № 11. – P. 35–41.

8. Золоторевич, Л.А. Моделирование неисправностей в структурах СБИС на языке VHDL / Л.А. Золоторевич // Информатика. – 2005. – № 1. – С. 89–94.

9. Золоторевич, Л.А. Моделирование неисправностей СБИС на поведенческом уровне на языке VHDL / Л.А. Золоторевич // Информатика. – 2005. – № 3. – С. 135–144.

10. Youngmin, H. Design error simulation based on error modeling and sampling techniques / H. Youngmin, S.A. Szygenda // Mathematics and computers in simulation. – 1998. – Vol. 46, № 11. – P. 35–46.

11. Fault Injection into VHDL Models: The MEFISTO Tool / E. Jenn [et al.] // Proc. of the 24th International Symposium on Fault Tolerant Computing. – Austin, Texas, USA, 1994. – P. 66–75.

12. Sieh, V. VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions / V. Sieh, O. Tschache, F. Balbach // Proc. of 27th International Symposium on Fault-Tolerant Computing. – Seattle, Washington, USA, 1997. – P. 32–36.

13. Xilinx Integrated Software Environment 8.2i [Electronic resource]. – 2006. – Mode of access: http://www.xilinx.com.

14. Bergeron, J. Writing Testbenches. Functional Verification of HDL Models / J. Bergeron. – Boston: Kluwer Academic Publishers, 2000. – 354 p.

15. Ярмолик, В.Н. Контроль и диагностика цифровых узлов ЭВМ / В.Н. Ярмолик. – Минск: Наука и техника, 1988. – 240 с.

Поступила 27.10.06

Белорусский государственный университет
информатики и радиоэлектроники,
Минск, ул. П. Бровки, 6
e-mail: ivaniuk@bsuir.unibel.by

A.A. Ivaniuk

VHDL APPLICATION FOR DIGITAL DEVICE FUNCTIONAL FAULTS SIMULATION

The methodology of digital device functional faults model insertion into VHDL-projects is proposed. Problems of faults distribution in modeling, test generation and response analysis automation are considered. The validity of the proposed methodology is analyzed.