УДК 681.327.2.06

А.А. Иванюк, А.В. Степанов

ВНЕДРЕНИЕ ФУНКЦИОНАЛЬНЫХ НЕИСПРАВНОСТЕЙ ОЗУ В ОПИСАНИЯ ЦИФРОВЫХ УСТРОЙСТВ НА ЯЗЫКЕ VHDL

Рассматривается проблема описания моделей функциональных неисправностей оперативных запоминающих устройств при помощи языка VHDL. Предлагается методика внедрения моделей функциональных неисправностей ОЗУ в проектные описания цифровых устройств на языке VHDL. Показывается, что предложенная методика может быть применена для оценки поведения цифрового устройства при наличии в нем дефектов, а также для верификации алгоритмов тестирования и контроля ОЗУ.

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 [1, 2]. Так, язык VHDL обладает возможностью описать структуру устройства на трех уровнях: системном, уровне представления регистровых передач и вентильном [2, 3].

При разработке современных цифровых устройств особое внимание уделяется обеспечению высокой надежности готового устройства. Поэтому решения, обеспечивающие надежность функционирования, закладываются еще на начальных этапах проектирования цифровых устройств. Важнейшим фактором, который негативно влияет на надежность и достоверность работы реализованных цифровых устройств, является наличие физических дефектов в полупроводниковых кристаллах. Существующее многообразие математических абстракций физических дефектов представляется обширным классом функциональных неисправностей [4, 5]. Универсальность языка VHDL позволяет описывать и внедрять модели неисправностей в готовые проектные описания цифровых устройств. Внедряемые модели неисправностей позволяют не только оценить поведение устройства при наличии в нем дефектов, но и разработать достоверные методы тестирования и диагностики, которые могут являться основой для проектирования встроенных средств самотестирования и самодиагностики [6].

1. Постановка задачи

Проблеме внедрения функциональных неисправностей в описания цифровых устройств посвящены многие работы [6–10], однако они решают задачу внедрения только неисправностей цифровых вентилей. Кроме того, некоторые из них предполагают использование дополнительных программных средств. Для запоминающих устройств в настоящее время проблема решается посредством моделирования содержимого массива запоминающих ячеек, которое искажается случайным образом. Данный подход является неадекватным отображением реальных дефектов, поэтому настоящая работа представляет собой попытку описать на языке VHDL современную нотацию функциональных неисправностей ОЗУ, внедрить их в проектные описания цифровых устройств и произвести оценку эффективности предложенной методики.

Решение поставленной задачи (рис. 1) начнем с рассмотрения способов описания ОЗУ на языке VHDL.

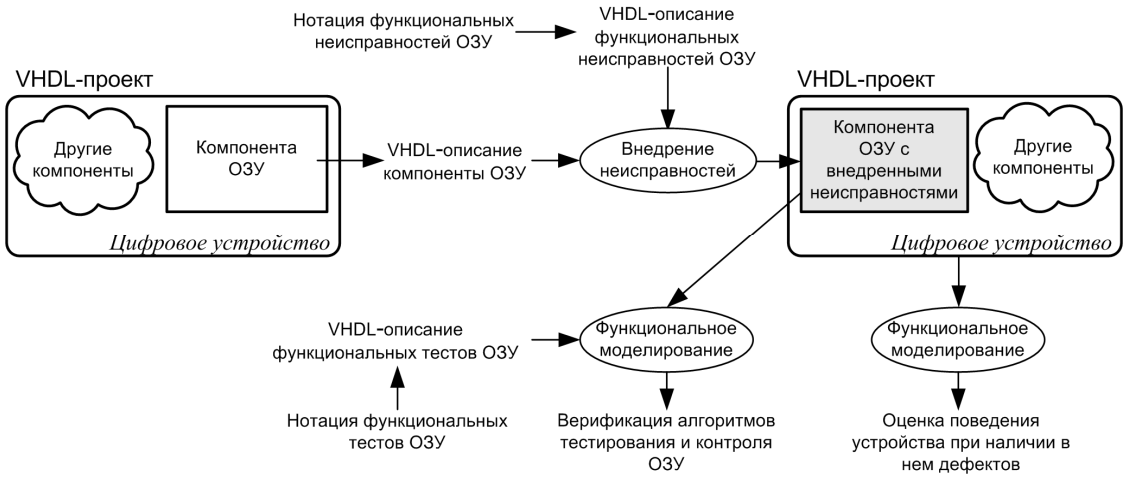

Рис. 1. Основные этапы решения поставленной задачи

2. Способы описания ОЗУ на языке VHDL

При проектировании сложных цифровых систем, таких как микроконтроллерные или микропроцессорные IP-ядра (Intellectual Property), перед проектировщиками встает задача разработки описаний ОЗУ. Средства языка VHDL не обладают стандартными лингвистическими конструкциям, которые однозначно бы воспринимались средствами синтеза в виде описания цифровых модулей ОЗУ. Тем не менее, функционирование подобных модулей достаточно подробно может быть описано на поведенческом подмножестве языка VHDL. Например, VHDL-описание модуля запоминающего устройства может включать в себя декларацию двухмерного сигнального объекта синтезируемого типа **bit** или **std_logic** и поведенческое описание процессов записи и чтения данных. Для большинства средств синтеза подобное описание будет транслироваться в RTL-модель, которая содержит набор базовых запоминающих элементов наподобие триггеров или защелок и набор комбинационных логических элементов, составляющих схемы адресации и управления запоминающего устройства. Рассмотрим примеры основных способов описания ОЗУ на языке VHDL.

1. *Использование шаблонных описаний модулей ОЗУ*. Данная методика заключается в применении типового VHDL-кода для описания в проекте используемого модуля ОЗУ. На стадии RTL-синтеза исходный VHDL-код распознается как типовое описание запоминающего устройства, для которого будут применены отличные от других цифровых схем методы технологического синтеза. Примером подобной методики может служить шаблонное VHDL-описание модулей ОЗУ, предлагаемое фирмой Synplicity в программном пакете проектирования Synplify Pro. В процессе синтеза распознанное типовое описание ОЗУ будет конструироваться на основе RTL-примитива двухпортового модуля ОЗУ (листинг 1, *а*, *б*). Недостатком такой методики является неадекватный результат синтеза в случае отклонения исходного VHDL-описания от типового шаблона.

2. *Использование IP-модулей ОЗУ*. Эта методика основана на включении в цифровой проект готовых библиотечных элементов модулей ОЗУ, которые являются уже синтезированными и оптимизированными для конкретной технологии изготовления (листинг 1, *в*). С точки зрения VHDL-описания подключение и использование таких модулей аналогично использованию сторонних компонент. При этом проектировщик не может изменить внутреннюю структуру и описание этих компонент. Например, программный пакет Core Generator фирмы Xilinx Inc., входящий в состав автоматизированной системы проектирования ISE, реализует приведенную методику.

3. *Использование специализированных средств синтеза модулей ОЗУ*. Такие средства часто называют компиляторами ОЗУ (Memory Compilers). Методика нацелена на проектирование внутренней топологии кристалла ОЗУ. При этом компонентами этой структуры могут быть не только логические вентили, но и транзисторные макроячейки, буферы-усилители и т. д. Данный подход к проектированию модулей ОЗУ отличается от предыдущего лишь тем, что все

функциональные блоки являются IP-компонентами, доступ к которым ограничен. Модификация таких IP-компонент требует наличия специализированных средств проектирования интегральных схем на транзисторном уровне, таких как Synopsys IC Station.

Листинг 1. **VHDL-описание исходной модели ОЗУ:** ***а)*** **VHDL-описание интерфейса объекта проекта;** ***б)*** **поведенческое VHDL-описание;** ***в)*** **использование IP-модуля ОЗУ**

```
entity RAM is
 generic (N,M: integer);
 port(
      CLK,WE,OE: in std_logic;
      AB: in  std_logic_vector(N-1 downto 0);
      DI: in  std_logic_vector(M-1 downto 0);
      DO: out std_logic_vector(M-1 downto 0));
end RAM;
```

а)

```
Architecture RTL of RAM is
 type mem_type is array (2**N-1 downto 0) of
      std_logic_vector (M-1 downto 0);
 signal mem: mem_type;
begin
 WR: process(CLK,WE,AB,DI)
 begin
  if CLK'event and CLK='1' then
   if WE='1' then
    mem(conv_integer(AB)) <= DI;
   end if;
  end if;
 end process;
 RD: process(AB,WE,OE,mem)
 begin
  if WE='0' and OE='1' then
   DO<=mem(conv_integer(AB));
  else
   DO<=(others=>'Z');
  end if;
 end process;
end RTL;
```

б)

```
architecture ipcore of ram is
component ip_ram
port (
      addr:in  std_logic_vector(1 downto 0);
      clk: in  std_logic;
      din: in  std_logic_vector(1 downto 0);
      dout:out std_logic_vector(1 downto 0);
      we:  in  std_logic);
end component;
signal d:std_logic_vector(1 downto 0);
begin
MEM:ip_ram port map (AB,CLK,DI,d,WE);
 OUTPUT:process(WE,OE,d)
 begin
  if WE='0' and OE='1' then
   DO<=d;
  else
   DO<=(others=>'Z');
  end if;
 end process;
end ipcore;
```

в)

В зависимости от того, какой из рассмотренных вариантов применяется для VHDL-описания модулей ОЗУ, возможны различные стратегии внедрения функциональных моделей неисправностей. В качестве следующего шага рассмотрим, каким образом функциональные модели неисправностей запоминающих устройств можно описать на языке VHDL. Главной задачей является достижение универсальности в описании моделей неисправностей цифровых запоминающих устройств, что позволит сделать процесс внедрения неисправностей расширяемым.

3. Универсальное описание моделей функциональных неисправностей ОЗУ

Применение достоверных средств проектирования и методик минимизации проектных ошибок не позволяет полностью исключить сбои в работе изготовленного цифрового устройства [11]. Причинами сбоев могут быть неисправности и влияние неблагоприятных факторов эксплуатации. Под неисправностью понимают физический дефект интегральной схемы, который приводит к появлению сбоя в функционировании цифрового устройства [11]. Природа физических дефектов и форма их проявления разнообразны и динамичны.

Различные физические дефекты могут быть описаны одной математической моделью [12]. Поэтому особое внимание уделяется формальному описанию физических дефектов в виде математических моделей неисправностей, которые адекватно описывают неисправные состояния запоминающих устройств.

Ввиду специфики внутренней организации цифровых запоминающих устройств существует многообразие моделей функциональных неисправностей массива запоминающих ячеек. В работе [13] предложена нотация для описания неисправностей цифровых запоминающих устройств, которая широко используется для описания простейших неисправностей и неис-

правностей взаимного влияния. Данная нотация позволяет не только описать существующие модели, но и расширить их множество путем ввода новых. Так, согласно предложенной нотации функциональная модель неисправностей ОЗУ, затрагивающих одну ячейку памяти, может быть представлена следующим образом:

$$< S / F / R >, \quad (1)$$

где S – последовательность операций, выполняемая над массивом запоминающих ячеек, которая спровоцирует искажение локальных данных отказавшего элемента памяти, т. е. функционирование запоминающего устройства не будет соответствовать ожидаемому поведению; F – хранимое значение неисправной ячейки памяти, $F \in \{0, 1\}$; R – значение, возвращаемое при выполнении операции чтения, $R \in \{0, 1, –\}$. При выполнении операции записи это значение не определено (R = '–').

В свою очередь, последовательность операций S содержит текущее значение ячейки памяти, тип выполняемой операции над ячейкой (чтение/запись) и значение, которое будет хранить ячейка памяти после выполнения данной операции.

Подобным образом описывается функциональная модель неисправностей взаимного влияния:

$$< Sa, Sv / F / R >, \quad (2)$$

где Sa – последовательность операций, выполняемая над ячейкой-агрессором; Sv – последовательность операций, выполняемая над ячейкой-жертвой. Под ячейкой-жертвой понимается ячейка ОЗУ, логическое состояние которой зависит от содержимого (0 или 1) или от логических переходов из 1 в 0 или из 0 в 1 ячейки, выступающей в роли агрессора [12].

Опишем данную нотацию, используя языковые средства языка VHDL. В листинге 2 приведен VHDL-код нотации, который позволит описать функциональные модели неисправностей запоминающих устройств, затрагивающих одну/две ячейки памяти.

***Листинг 2*. VHDL-описание нотаций < *S* / *F* / *R* > и < *Sa* / *Fv* / *R* >**

```
type TYPE_VALUE is ('0','1', '-');
type TYPE_OPERATION is ('w','r', '-');

-- тип SOS - sensitizing operation sequence
type TYPE_SOS is record
     stored_value: TYPE_VALUE;     -- хранимое значение ячейки.
     operation: TYPE_OPERATION;    -- 0 - чтение; 1 - запись
     value: TYPE_VALUE;            -- Если операция чтение, то ожидаемое
                                   -- значение, иначе записываемое.
end record TYPE_SOS;

-- тип Fault Primitive для Single-Cell. Нотация <S / F / R>.
type TYPE_FP_SCELL is record
     S: TYPE_SOS;
     F: bit;        -- хранимое значение неисправной ячейки памяти
     R: TYPE_VALUE; -- возвращаемое значение
end record TYPE_FP_SCELL;

-- тип Fault Primitive для Two-Cell. Нотация <Sa, Sv / F / R>.
type TYPE_FP_TCELL is record
     Sa: TYPE_SOS;  -- агрессор
     Sv: TYPE_SOS;  -- жертва
     F: bit;        -- хранимое значение неисправной ячейки памяти
     R: TYPE_VALUE; -- возвращаемое значение
end record TYPE_FP_TCELL;
```

Пример. В соответствии с данной нотацией переходная неисправность, которая характеризуется невозможностью логического перехода состояния ячейки запоминающего устройства из 0 в 1 при выполнении соответствующей операции записи, на языке VHDL может быть определена следующим образом:

```
constant fp_TF0   : TYPE_FP_SCELL := (('0', 'w', '1'), '0', '-');.
```
(3)

Приведенная ниже таблица наглядно демонстрирует применение выбранной нотации для описания функциональных неисправностей ОЗУ на языке VHDL.

Таблица

VHDL-описание функциональных неисправностей ОЗУ

Функциональная неисправность ОЗУ	VHDL-описание
Неисправность константного нуля	constant fp_SAF0 : type_fp_scell := (('-', '-', '-'), '0', '-');
Неисправность константной единицы	constant fp_SAF1 : type_fp_scell := (('-', '-', '-'), '1', '-');
Переходная неисправность	constant fp_TF0 : type_fp_scell := (('0', 'w', '1'), '0', '-');
Инверсная неисправность взаимного влияния	constant fp_CFin0 : type_fp_tcell := (('0', 'w', '1'), ('1', '-', '-'), '0', '-');
Неисправность прямого действия	constant fp_CFid0 : type_fp_tcell := (('0', 'w', '1'), ('-', '-', '-'), '0', '-');

Рассмотрим методику внедрения моделей функциональных неисправностей ОЗУ в проект цифрового устройства с учетом приведенного универсального описания.

4. Внедрение функциональных неисправностей ОЗУ

Для корректного VHDL-описания цифрового устройства внедрение моделей функциональных неисправностей означает изменение поведения проектной модели [14]. Такие изменения могут быть осуществлены как в поведенческом, так и в структурном представлении устройства [8, 15] и заключаются в применении нескольких подходов. Один из подходов использует замену исходного описания на описание, которое представляет некорректное функционирование компоненты согласно применяемой модели неисправности [7]. Еще одним примером может служить метод, который управляет значениями сигналов и переменных в процессе моделирования цифровой компоненты, не изменяя исходного VHDL-описания [9]. В работе [10] описана программная среда, которая является надстройкой над языком VHDL и позволяет вносить модели неисправностей в цифровой проект на различных уровнях абстракции.

Для моделирования сложных неисправностей запоминающих устройств часто прибегают к реализации поведенческой модели при помощи HDL-языков либо языков программирования высокого уровня. Обусловлено это прежде всего тем, что для проектирования цифровых запоминающих устройств используют транзисторный уровень абстракции, который, в свою очередь, затрудняет внедрение моделей функциональных неисправностей. Подавляющее большинство систем проектирования цифровых устройств не позволяет определять внутреннюю структуру встраиваемых ОЗУ, а использует модули памяти в качестве IP-компонент, для которых известными являются лишь интерфейсная часть и модель поведения. Один из подходов к внедрению и моделированию функциональных неисправностей запоминающих устройств показан на рис. 2.

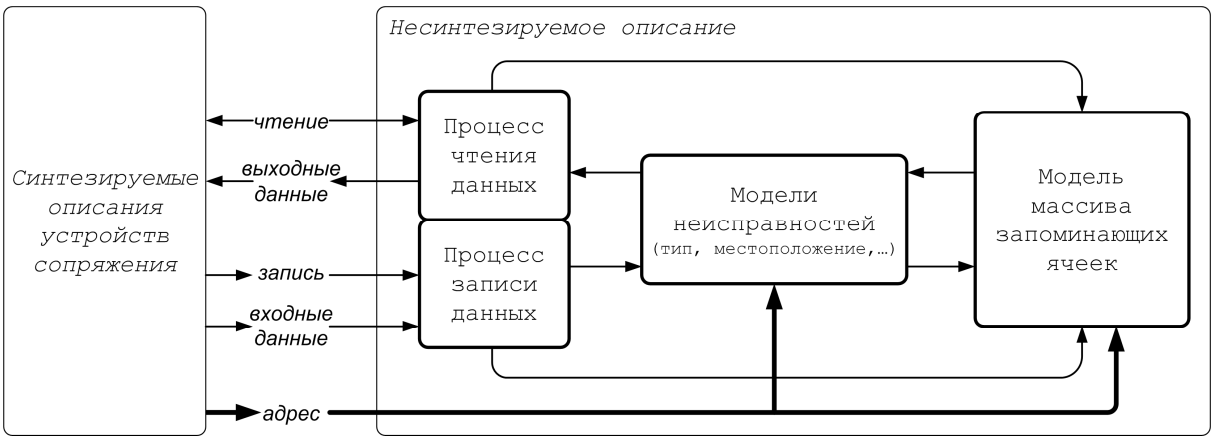

Рис. 2. Структура компоненты ОЗУ, реализованной на поведенческом уровне языка VHDL

Подход заключается в создании несинтезируемого HDL-описания поведенческой модели ОЗУ, в котором процесс записи и чтения данных дополняется внедряемыми моделями неисправностей. При этом модели неисправностей представляются HDL-кодом, искажающим данные в соответствии с негативным воздействием моделируемого дефекта.

Внедрение моделей неисправностей в цифровой проект должно быть управляемым и настраиваемым под различные ситуации моделирования. Языковые VHDL-модели неисправностей должны отвечать таким требованиям, как 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.

Рассмотрим применение предложенной методики для случая внедрения функциональных неисправностей запоминающих устройств в VHDL-описание бит-ориентированного синхронного ОЗУ. Вначале определим основные характеристики проектируемого ОЗУ:

– однопортовое бит-ориентированное ОЗУ с информационной организацией 2^N, где N – разрядность входной шины адреса ADDR, 2^N – количество информационных бит;

– ОЗУ имеет две одноразрядные шины данных: DATA_IN – входная шина данных, DATA_OUT – выходная;

– управление функционированием ОЗУ происходит посредством четырех специализированных входов: CLK – вход синхронизации, ReadWrite – выполнение операции чтения/записи, OutEnable – разрешение выходной шины данных, GEN_FAULT – управление процессом распределения неисправностей;

– операция записи данных в ОЗУ осуществляется по переднему фронту синхросигнала, поступающего на вход CLK с предварительной установкой значений на входах ADDR, DATA_IN и ReadWrite='1';

– чтение данных представляет собой асинхронную операцию, условием проведения которой является предварительная установка значений на входах ADDR, ReadWrite ='0' и OutEnable='1'. Если OutEnable='0', то выходная шина DATA_OUT будет находиться в отключенном (высокоомном) состоянии;

– операция формирования карты неисправностей осуществляется по переднему фронту сигнала, поступающего на вход GEN_FAULT.

Составим по приведенным характеристикам описание языкового блока интерфейса проектируемого модуля ОЗУ (листинг 3). Настраиваемыми параметрами выберем значения *ADDR_WIDTH* (разрядность входной шины адреса), *FAULTS_COUNT* (количество внедряемых неисправностей), *FAULT_TYPES_SCELL_PERMIT* (маска разрешения типов неисправностей, затрагивающих одну ячейку памяти) и *FAULT_TYPES_TCELL_PERMIT* (маска разрешения типов неисправностей, затрагивающих две ячейки памяти). Все входные и выходные линии модуля ОЗУ опишем в качестве портов типа **std_logic**.

***Листинг 3*. VHDL-описание интерфейса проектируемой модели ОЗУ**

```
entity BIT_SRAM is
      generic(
            -- разрядность входной шины адреса
            ADDR_WIDTH:   integer := 20;
            -- количество внедряемых неисправностей
            FAULTS_COUNT: integer := 1;

            -- последний бит = '1' - все типы неисправностей разрешены
            FAULT_TYPES_SCELL_PERMIT : std_logic_vector (0 to FP_SCELL_MAP'length) := "00000000000";
            FAULT_TYPES_TCELL_PERMIT : std_logic_vector (0 to FP_TCELL_MAP'length) := "000000000000");
      -- описание портов ввода/вывода
      port(
            CLK:              in  std_logic;
            GEN_FAULT:        in  std_logic;
            OutEnable:        in  std_logic;
            ReadWrite:        in  std_logic;
            ADDR:             in  std_logic_vector(ADDR_WIDTH-1 downto 0);
            DATA_IN:          in  std_logic;
            DATA_OUT:         out std_logic
            );
end BIT_SRAM;
```

Согласно исходной модели создадим поведенческое описание модуля ОЗУ, которое содержит три процесса: WRITE, описывающий операцию записи данных; READ, описывающий операцию чтения данных из ОЗУ, и GENERATE_FAULT, описывающий операцию формирования карты неисправностей. В декларации архитектурного блока объявим сигнал, который будет соответствовать массиву запоминающих ячеек ОЗУ, и глобальную переменную, которая будет соответствовать карте неисправностей. Карта неисправностей представляет собой структуру данных, содержащую такие сведения, как тип неисправности, ее местоположение и статус (активна/неактивна). Процесс GENERATE_FAULT выполняет распределение местоположения, кратности и типов неисправностей при моделировании. Процессы WRITE и READ реализованы так, что представляют собой функционирование компоненты памяти согласно внедренным функциональным неисправностям, описание которых соответствует предложенной универсальной нотации на языке VHDL (листинг 4).

***Листинг 4*. Пример поведенческого описания проектируемой модели ОЗУ на языке VHDL: *а)* процесс WRITE; *б)* процесс READ**

```
-- процесс записи данных
WRITE: process (CLK, ReadWrite, ADDR, DATA_IN, en_faults)
-- декларация переменных
begin
      ...
-- модифицировать функционирование в случае активизации
-- неисправности, затрагивающей одну ячейку ОЗУ
-- 1) не операция чтения
if (ReadWrite = '1' and fault.type_fault.S.operation /= 'r'
and -- 2) хранимое значение соответствует правилу
(fault.type_fault.S.stored_value = '-'
or (mem(address) = '1' and fault.type_fault.S.stored_value = '1')
or (mem(address) = '0' and fault.type_fault.S.stored_value = '0'))
and -- 3) записываемое значение соответствует правилу
(fault.type_fault.S.value = '-'
or (DATA_IN = '1' and fault.type_fault.S.value = '1')
or (DATA_IN = '0' and fault.type_fault.S.value = '0'))) then

-- записать некорректное значение в ячейку ОЗУ
      ...
end if;
      ...
-- неисправность необнаружена или неактивизирована
if ReadWrite = '1' and is_fault_activated = false then
      mem(address) <= DATA_IN;
end if;
      ...
end process;
```

а)

```
-- процесс чтения данных
READ: process (OutEnable, ReadWrite, ADDR, en_faults, mem)
-- декларация переменных
begin
if ReadWrite = '0' and OutEnable = '1' then
      ...
-- модифицировать функционирование в случае активизации
-- неисправности, затрагивающей одну ячейку ОЗУ
-- 1) операция чтения
if fault.type_fault.S.operation = 'r'
and -- 2) хранимое значение соответствует правилу
((mem(address) = '1' and fault.type_fault.S.stored_value = '1')
or (mem(address) = '0' and fault.type_fault.S.stored_value = '0'))
then
-- исказить выходные данные
      ...
end if;
      ...
-- неисправность необнаружена или неактивизирована
if is_fault_activated = false then
      DATA_OUT <= mem(address);
end if;

else
      DATA_OUT <= 'Z';
end if;

end process;
```

б)

Поддержка универсальной нотации моделей функциональных неисправностей позволяет внедрять их описания без какого-либо внесения изменений в исходный VHDL-код спроектированной модели ОЗУ. Для внедрения модели неисправности необходимо выполнить следующую последовательность действий:

– описать неисправность в соответствии с универсальной нотацией;

– добавить данную неисправность в общий список неисправностей;

– разрешить внедрение данной неисправности путем установки соответствующего бита маски разрешения типов неисправностей.

Для оценки эффективности предложенной методики внедрения неисправностей произведем оценку временных затрат на выполнение алгоритмов тестирования и контроля ОЗУ.

5. Верификация алгоритмов тестирования и контроля ОЗУ

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-модуля [16]:

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

2) 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. Применение 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.

В настоящее время среди большого разнообразия алгоритмов тестирования и контроля ОЗУ следует выделить маршевые тесты. Это высокоэффективные алгоритмы тестирования, которые позволяют с высокой вероятностью обнаруживать определенные типы неисправностей ОЗУ. В работе [17] был предложен универсальный язык описания тестов MTL, который широко используется для представления современных маршевых тестов.

Согласно MTL любой маршевый тест представляет собой последовательность маршевых элементов, разделяемых символом «;»:

$$< \textit{маршевый тест} > ::= \{< \textit{маршевый элемент} > \{; < \textit{маршевый элемент} >\}\}. \quad (4)$$

Используемые фигурные скобки служат для обозначения повторения выражения, заключенного в них. Выражение может повторяться k раз (k=0, 1, 2, ...). При k=0 выражение отсутствует. В свою очередь, каждый элемент содержит символ, определяющий порядок перебора адресов ЗУ: $\Uparrow$ – определяет линейный перебор адресного пространства ЗУ по возрастанию, $\Downarrow$ – определяет линейный перебор адресов по убыванию, $\Updownarrow$ – любой из способов линейного перебора (по убыванию либо возрастанию). Кроме этого, маршевый элемент содержит последовательность операций:

$$\begin{gathered} < \textit{маршевый элемент} > ::= < \textit{символ адресации} > (\ <\textit{операция}> \{, <\textit{операция}>\}) \\ < \textit{символ адресации} > ::= \Uparrow | \Downarrow | \Updownarrow . \end{gathered} \quad (5)$$

Вертикальная черта служит для обозначения альтернативных случаев. Каждая < *операция* > представляет собой элемент из следующего набора: «r0» – операция чтения ячейки памяти с ожидаемым значением 0, «r1» – операция чтения ячейки памяти с ожидаемым значением 1, «w0» – операция записи 0 в ячейку памяти, «w1» – операция записи 1 в ячейку:

$$< \textit{операция} > ::= r0 \mid r1 \mid w0 \mid w1. \quad (6)$$

Одна или несколько операций в маршевом элементе используются последовательно для адресуемой ячейки памяти. Переход к следующей ячейке будет осуществлен только после выполнения всех операций в текущем маршевом элементе.

Покажем, что данная нотация может быть реализована при помощи языка VHDL (листинг 5).

***Листинг 5*. Типы, необходимые для описания маршевых тестов на языке MTL**

```
-- порядок перебора адресов ЗУ
type TYPE_MODE_ADDRESSING is (UP, DOWN, ANY);
-- основные операции разрушающих маршевых тестов
type TYPE_TEST_OPERATION is (w0, w1, r0, r1);
-- набор операций маршевого элемента
type TYPE_MARCH_OPERATIONS is array (POSITIVE range <>) of TYPE_TEST_OPERATION;
type MARCH_OPERATIONS_ACCESS is access TYPE_MARCH_OPERATIONS;

-- описание элемента маршевого теста
type TYPE_MARCH_ELEMENT is record
      mode_addressing: TYPE_MODE_ADDRESSING;
      operations : MARCH_OPERATIONS_ACCESS;
end record TYPE_MARCH_ELEMENT;

-- описание маршевого теста
type TYPE_MARCH_TEST is array (POSITIVE range <>) of TYPE_MARCH_ELEMENT;
-- доступ к набору элементов маршевого теста
type TYPE_MARCH_TEST_ACCESS is access TYPE_MARCH_TEST
```

Пример. Согласно MTL маршевый тест MATS+ имеет следующую запись:

$$\text{MATS+: } \{\Updownarrow (w0);\ \Uparrow (r0, w1);\ \Downarrow (r1, w0)\}. \tag{7}$$

Приведем соответствующее описание данного маршевого теста на языке VHDL:

```
shared variable MARCH_TEST_MATS_P : TYPE_MARCH_TEST_ACCESS :=
new TYPE_MARCH_TEST'(
        (ANY, new TYPE_MARCH_OPERATIONS'(1 => w0))
        , (UP, new TYPE_MARCH_OPERATIONS'(r0, w1))
        , (DOWN, new TYPE_MARCH_OPERATIONS'(r1, w0))); .
```
(8)

Для автоматизации процесса верификации алгоритмов тестирования ОЗУ на основе предложенного описания маршевых тестов реализуем соответствующую процедуру на языке VHDL. Применение данного подхода позволит оценивать покрывающую способность произвольного маршевого теста, заданного в соответствии с введенной нотацией.

Используя предложенное средство верификации алгоритмов тестирования ОЗУ, оценим эффективность разработанной методики внедрения функциональных неисправностей запоминающих устройств. Для этого был поставлен эксперимент, который заключался в сравнении временных затрат на выполнение процедуры тестирования для поведенческой модели ОЗУ и предложенной модели ОЗУ с возможностью внедрения функциональных неисправностей запоминающих устройств. Обе модели ОЗУ описывают запоминающее устройство с информационной емкостью 1 Мбит. Для модели ОЗУ с возможностью внедрения функциональных неисправностей оценка временных затрат выполнялась для двух случаев: без внесения неисправностей и с внесением неисправностей произвольных типов одиночной кратности. Тип неисправности и ее местоположение выбирались посредством случайного равномерного распределения. В качестве алгоритма тестирования был выбран разрушающий маршевый тест MATS+. В ходе поставленного эксперимента для каждого выбранного случая маршевый тест выполнялся десять раз. Моделирование осуществлялось на базе пакета программных средств ModelSim PE 6.2f корпорации Model Technology, который в настоящее время является одной из самых распространенных систем HDL-моделирования [18].

Результаты эксперимента (рис. 3) свидетельствуют о том, что внедрение функциональных неисправностей в модель ОЗУ не повлекло за собой значительного снижения скорости выполнения алгоритма тестирования. Так, время выполнения процедуры тестирования возросло не более чем на 5 %.

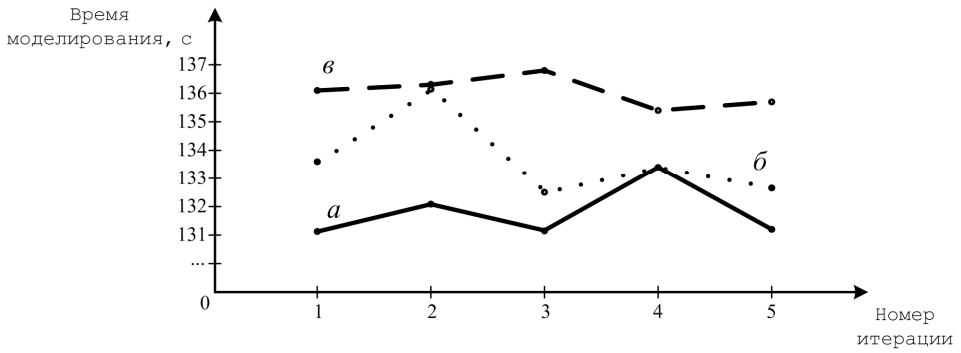

Рис. 3. Сравнительная оценка временных затрат: *а)* поведенческая модель ОЗУ; *б)* предложенная модель ОЗУ без внесения неисправностей; *в)* предложенная модель ОЗУ с внесением неисправностей произвольного типа одиночной кратности

Подобные результаты были получены при сравнительной оценке временных затрат на выполнение маршевых тестов различной сложности (рис. 4). Сложность теста L оценивается

как совокупность всех элементарных операций с тестируемым ОЗУ. Сложность теста MATS+ может быть оценена по следующей формуле:

$$L(\text{MATS+}) = N + 2N + 2N = 5N, \tag{9}$$

где N – информационная емкость ОЗУ в битах.

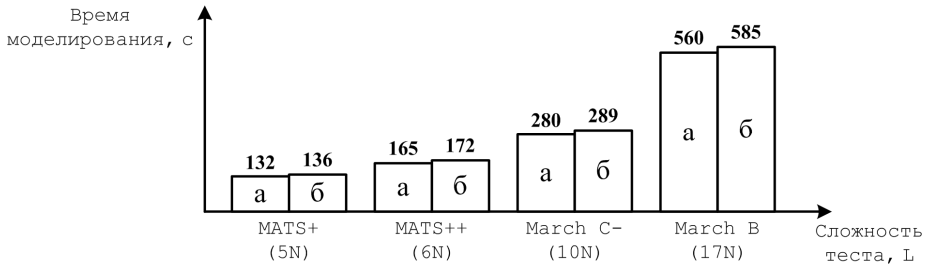

Рис. 4. Сравнительная оценка временных затрат на выполнение маршевых тестов различной сложности: *а)* поведенческая модель ОЗУ; *б)* предложенная модель ОЗУ с внесением неисправностей произвольного типа одиночной кратности

Представленные результаты проведенных исследований позволяют сделать вывод об эффективности разработанной методики внедрения функциональных неисправностей ОЗУ посредством языка VHDL.

Заключение

В статье предложена методика внедрения моделей функциональных неисправностей ОЗУ в проектные описания цифровых устройств на языке VHDL, которая может быть применена для верификации алгоритмов тестирования и контроля ОЗУ с учетом внедренных моделей неисправностей. Выполнена оценка эффективности разработанной методики. Разработан подход к универсальному описанию функциональных неисправностей и маршевых тестов средствами языка VHDL.

Список литературы

1. Hwang, E.O. Digital Logic and Microprocessor Design with VHDL / E.O. Hwang. – New-York: Brooks/Cole, 2005. – 513 p.

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

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

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

5. Stroud, C.E. A Designer’s Guide to Built-In Self-Test / C.E. Stroud. – Boston: Kluwer Academic Publishers, 2002. – 319 p.

6. Ярмолик, В.Н. Внедрение функциональных неисправностей в VHDL-описания цифровых устройств / В.Н. Ярмолик, А.А. Иванюк // Автоматика и вычислительная техника. – 2007. – № 3. – С. 3–12.

7. 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.

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

9. 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.

10. 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, 1997. – P. 32–36.

11. Иванюк, А.А. Проектирование контролепригодных цифровых устройств / А.А. Иванюк, В.Н. Ярмолик. – Минск: Бестпринт, 2006. – 296 с.

12. Неразрушающее тестирование запоминающих устройств / В.Н. Ярмолик [и др.]. – Минск: Бестпринт, 2005. – 230 с.

13. Goor, A.J. Functional Memory Faults: A Formal Notation and a Taxonomy / A.J. Goor, Z. Al-Ars // VTS 2000, 18th IEEE VLSI Test Symposium. – Dublin, Ireland, 2000. – P. 281–289.

14. Иванюк, А.А. Моделирование функциональных неисправностей цифровых устройств средствами языка VHDL / А.А. Иванюк // Информатика. – 2007. – № 1. – С. 30–40.

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

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

17. Goor, A.J. Towards a Uniform Notation for Memory Tests / A.J. Goor, A. Offerman // Proc. of European Design and Test Conference. – Paris, 1996. – P. 420.

18. ModelSim PE 6.2f [Electronic resource]. – 2007. – Mode of access: http://www.model.com. – Date of access: 01.02.2008.

Поступила 06.02.08

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

A.A. Ivaniuk, A.V. Stepanov

FUNCTIONAL RANDOM ACCESS MEMORY FAULTS INJECTION INTO VHDL-PROJECTS OF DIGITAL DEVICES

A methodology to inject functional RAM faults into the VHDL-projects of digital devices is presented. An approach to describe the fault models and memory test algorithms using VHDL language is proposed. It is shown that the proposed methodology allows to evaluate the system's behavior in case of the faults and to realize memory test algorithm verification.