Yu. I. Petrov, PhD, Educational, Programs Manager, e-mail: mailtoyuripetrov@gmail.com
Kaspersky Lab,
Yu. A. Karnaukhov, Head of Production Department, e-mail: fishmanship@gmail.com,
IN-Media AG

"FK-Monitoring" Software as a Tool for a Business Process Monitoring

The last decades are characterized by the sharp growth of informatization of economic, administrative and manufacturing processes both in state and commercial structures. Mainly it is explained by aspiration of the organizations to increase efficiency of their activity and, respectively, to strengthen their competitive positions; first of all nowadays it is possible by using of various information systems (IS). Among other types information analytical systems (IAS) are of great importance and allow providing adoption of highly effective administrative decisions and support of elements of strategic planning. At the same time systems of this type are usually developed on demand (custom-made) since the requirements to in and out information, to processing of internal documentation and to reports are strongly depend on company specifics.

Article considers issues of design and implementation of software (analytical information system) for production monitoring at IN-Media AG Company (Switzerland). The software suite includes three components: FK-Server (both console and windowed versions, fetches and parses data from production lines), FK-Client (visualize and analyze data from FK Server, including wide-range statistics and planning ability) and FK Web Client (shows actual production information on non-desktop computer devices). FK Client is a core multi-user component and has bilingual user interface language (can be switched between Russian and German).

The architecture of the software suite is suggested, illustrations of its functioning are presented.

Keywords: *information system, analytical information system, software, business process, monitoring, publishing holding, OLAP, Delphi, PostgreSQL*

References

1. **IN-Media** — Home. URL: http://www.in-media.ch.
2. in-online.ch — *Das online Magazin des in. Werbe-Folders.* URL: http://www.in-online.ch/.
3. **Common** *Format and MIME Type for Comma-Separated Values (CSV) Files.* URL: http://www.ietf.org/rfc/rfc4180.txt
4. **[MS-XLSX]:** *Excel (.xlsx) Extensions to the Office Open XML SpreadsheetML File Format.* URL: http://msdn.microsoft.com/en-us library/dd922181 (v=office. 12).aspx
5. **Petrov Yu. I.** Arhitektura i algoritmicheskoe obespechenie informacionnoj sistemy normokontrolja vypusknyh kvalifikacionnyh rabot. *Programmnaja inzhenerija*. 2013. № 3. P. 26—32.
6. **Spinellis D., Gusios G.** *Ideal'naja arhitektura. Vedushhie specialisty o krasote programmnyh arhitektur.* Spb.: Simvol-Pljus, 2010. 528 p.
7. **Ford N., Najgard M.** 97 jetjudov dlja arhitektorov programmnyh sistem. Spb.: Simvol-Pljus, 2010. 224 p.
8. **PostgreSQL:** *The world's most advanced open source database.* URL: http://www.postgresql.org/
9. **Delphi** XE2 // *Embarcadero Technologies.* URL: http://edn. embarcadero.eom/article/41593
10. **Introducing** JSON. URL: http://www.json.org/
11. **HTML5.** URL: http://www.w3.org/TR/htm15/
12. **CSS3** Introduction. URL: http://www.w3schools.com/css/ css3_intro.asp
13. **Obzor** JavaScript. URL: http://developer.mozilla.org/ru/docs/ Web/JavaScript/ru_JavaScript_Overview.
14. **Petrov Yu. I.** *Razvitie biznes processov sozdanija konkurentosposobnoj tehnicheski slozhnoj produkcii mashinostroitel'nyh predprijatij:* Dis. ... kand. jekon. nauk. Moscow, 2012. 148 p.

УДК 681.5:004.414.2

Н. А. Авдеев, канд. техн. наук, ст. науч. сотр.,
П. Н. Бибило, д-р техн. наук, проф., зав. лаб., e-mail: bibilo@newman.bas-net.by
Объединенный институт проблем информатики Национальной академии наук Беларуси, г. Минск

Расширение возможностей автоматизированного проектирования цифровых систем при использовании стандарта VHDL'2008

Кратко анализируются и иллюстрируются примерами расширения стандарта высокоуровневого языка VHDL, являющегося одним из основных языков автоматизированного проектирования цифровых систем на базе СБИС. Расширение множества конструкций в стандарте VHDL'2008 предназначено для удобства написания сложных тестирующих программ и верификации.

Ключевые слова: *автоматизированное проектирование, цифровая система, VHDL, тестирование, функциональная верификация*

Введение

Одним из базовых языков проектирования цифровых систем на основе сверхбольших интегральных схем (СБИС), выполняемых как в виде ПЛИС (программируемых логических интегральных схем), так и в виде заказных СБИС, является в настоящее время VHDL (**V**ery high speed integrated circuits **H**ardware **D**escription **L**anguage). Первый стандарт VHDL'1987 появился в 1987 г., следующий — VHDL'1993 стал, пожалуй, основным. Он был полностью реализован для целей моделирования описаний цифровых систем в промышленных САПР СБИС ведущих фирм, таких как Synopsys, Mentor Graphics и многих других. Для автоматического синтеза логических схем из VHDL'1993 было выделено синтезируемое подмножество (стандарт IEEE Std 1076.6—1999), называемое часто RTL-подмножеством. Синтез схем по описаниям на RTL-подмножестве был реализован в автоматических системах синтеза — синтезаторах, например, в системах синтеза структур ПЛИС типа FPGA (синтезатор XST фирмы Xilinx) и системах синтеза логических схем для заказных СБИС (синтезатор Leonardo-Spectrum фирмы Mentor Graphics).

Дальнейшее развитие язык получил в стандартах VHDL'2002 и VHDL'2008. В этих последних стандартах усовершенствование старых и появление новых конструкций языка в значительной мере было обусловлено необходимостью написания сложных тестирующих программ и проведения различных видов моделирования в целях верификации исходных спецификаций проекта цифровой системы. Тенденция развития средств автоматизированного проектирования такова, что выполнение верификации потребовало применения более совершенных языковых средств и VHDL-пакетов по сравнению со средствами, использующимися для автоматического синтеза схем по синтезируемым VHDL-описаниям. Проблемы верификации, по сути, выходят на передний план в проектировании, а автоматический синтез логических схем отходит на второй план.

В данной статье перечисляются и описываются новые конструкции, появившиеся в стандарте VHDL'2008 [1]. Примеры расширения языка иллюстрируются фрагментами VHDL-кодов. Там, где это уместно, проводится сравнение с конструкциями стандартов VHDL'1993 либо VHDL'2002. Полные тексты основных примеров VHDL-кодов из данной статьи можно найти на сайте [2]. Достаточно подробное описание VHDL'2008 содержится в книге [3]. Изменения в стандарте VHDL'2008, касающиеся синтезируемого подмножества языка, в данной статье не изучаются, так как эта тема заслуживает отдельного рассмотрения.

Добавление конструкций языка PSL для проверки свойств модели

Язык PSL (Property Specification Language) [4] был разработан специально для целей верификации. В русскоязычной литературе данный язык часто называют "язык ассертов". В текстах VHDL-программ, написанных по стандарту VHDL'2008, можно использовать конструкции языка PSL. При этом операторы PSL оформляются как комментарии и при синтезе (но не при моделировании!) пропускаются. Средства PSL используются для проверки различных поведенческих свойств модели цифровой системы и ориентированы на верификацию [5]. Заметим, что комментарий в VHDL начинается с двух дефисов и продолжается до конца строки.

В примере ниже имеются комментарии, начинающиеся "--psl", которые в VHDL'2008 при моделировании воспринимаются как команды на языке PSL:

```
type T_state is (a1, a2, a3, a4, a5, a6);
signal state: T_state;
   -- ps1 default clock is rising_edge(c1k);
   -- ps1 property prop1 is
            never {state=a2; state=a5};
   -- ps1 as1: assert prop1;
   --  ps1 as2: assert always {state=a2; state=a4} |=>
   {state=a6};
   -- ps1 as3: cover {state=a1; state=a4};
```

Использование типа protected для функциональной верификации

Тип protected (защищенный тип), появившийся в стандарте VHDL'2002, позволяет облегчить написание тестирующих программ, используется при функциональной верификации [6, 7] и базируется на концепции, похожей на классы в объектно-ориентированном программировании. Тип protected реализует инкапсуляцию в VHDL-коде, так как для такого типа осуществляются:

- объединение элементов данных и операций, которые могут выполняться над ними, в один объект;
- сокрытие деталей реализации типов данных от пользователей.

Пользователи VHDL, которые никогда не работали с защищенными типами, могут легко представить их как специальную версию типа запись (record), которая позволяет включать процедуры и функции дополнительно к обычным полям данных. Полное определение типа protected состоит из двух частей:

- декларации типа protected;
- тела (body) типа protected.

Объявления в декларативной части типа могут включать декларации подпрограмм (процедур и функций), спецификации атрибутов и конструкции подключения, использующие ключевое слово use. Тела подпрограмм объявляются в теле типа protected.

Подпрограммы, описанные при декларации типа protected, названы в стандарте VHDL'2008 [1] методами. Примеры использования типа protected и примеры VHDL-программ для выполнения функциональной верификации даны в работах [8, 9].

Параметризация типов и список параметров для пакетов

В VHDL'2008 появилась возможность декларировать типы как параметры, которые в дальнейшем получают вполне конкретные значения. Ниже в примере пакета func_pkg в качестве параметров объявлен тип element_type и функция add. Данный пакет содержит список параметров generic, что было недопустимо в предыдущих стандартах. Настройка параметров этого пакета осуществляется в другом пакете func_int_pkg, где формальному параметру — типу element_type — ставится в соответствие тип integer, а формально определенной функции add ставится в соответствие функция ass:

```
package func_pkg is
    generic (type element_type); -- VHDL'2008
    function add (a : element_type; b : element_type) return
    element_type;
end package func_pkg;

package func_int_pkg is new work.func_pkg
    generic map (element_type => integer,
        add => ass);
```

Массивы с неограниченными диапазонами

В стандарт VHDL'2008 добавлена возможность декларации и использования *двумерных* массивов, имеющих неограниченные диапазоны. Неограниченный диапазон массива описывается в виде (natural range <>), "неограниченность" понимается как отсутствие при декларации массива конкретных границ возрастающего (to) либо убывающего (downto) диапазона значений, которые может принять индекс (если массив одномерный), либо несколько индексов (в случае многомерного массива). Приведем пример декларации типа std_logic_matrix — двумерного массива с неограниченным диапазоном:

```
type std_logic_matrix is array (natural range <>) of
std_logic_vector;
```

После такой декларации можно продекларировать сигналы A, B, C — массивы с ограниченными диапазонами, т. е. диапазонами, имеющими конкретные значения границ изменения индексов по первой и второй размерности:

```
signal A : std_logic_matrix(5 downto 0)(7 downto 0);
signal B : std_logic_matrix(0 to 6)(7 downto 0);
signal C : std_logic_matrix(5 downto 0)(2 to 4);
```

Для сигнала A: (5 downto 0) — это диапазон изменения индекса массива по первой размерности — по строкам. Строки имеют номера 5, 4, 3, 2, 1,0 — всего шесть векторов длиной восемь битов. Диапазон (7 downto 0) — это диапазон изменения индекса массива по второй размерности — по столбцам (номера столбцов — 7, ..., 0). Ранее при использовании стандарта VHDL'1993 нужно было бы каждый раз (для сигналов с различными диапазонами) декларировать свой тип, т. е. записывать декларации:

```
-- VHDL'1993
type std_logic_matrix_A is array (5 downto 0) of
std_logic_vector (7 downto 0);
type std_logic_matrix_B is array (0 to 6) of std_logic_vector
(7 downto 0);
type std_logic_matrix_C is array (5 downto 0) of
std_logic_vector (2 to 4);
```

Заметим, что в VHDL'1993 можно декларировать одномерные массивы с неограниченным диапазоном.

Записи с массивами неограниченных диапазонов

Ниже в примере фрагмента VHDL-кода показывается использование записей, полями которых являются неограниченные массивы векторов (типа std_logic_vector), декларированных в пакете STD_LOGIC_1164, векторов типа знаковый (signed), беззнаковый (unsigned) из пакета NUMERIC_STD и других векторных типов. В примере запись типа complex содержит три таких поля. При декларировании сигналов такого типа поля записи ограничиваются конкретными диапазонами:

```
type complex is record
    a: std_logic_vector;
    b: signed;
    c: unsigned;
end record;
. . .
signal a1 : std_logic_vector (3 downto 0);
signal b1 : signed (1 to 5);
signal c1 : unsigned (6 downto 3);
signal x1 : complex (a (3 downto 0), b (1 to 5),
c (6 downto 3));
. . .
x1<= ("1111", "10000", "0100");
. . .
x1.a <= "1111";
x1.b <= "10000";
x1.c <= "1010";
. . .
a1<= x1.a;
b1<= x1.b;
c1<= x1.c;
```

Сигнал x1

```
signal x1: complex (a(3 downto 0), b(1 to 5), c(6 downto 3));
```

типа complex позволяет ограничить неограниченные массивы в записи (record) типа complex. Можно присваивать значение всей записи сразу:

```
x1<= ("1111", "10000", "0100");
```

Использование чисел с фиксированной точкой

Для работы с числами (без знака) с фиксированной точкой используется VHDL-пакет FIXED_PKG, в котором декларирован тип ufixed:

```
type ufixed is array (integer range <>) of std_logic;
```

Естественно, ссылка на этот пакет обязательна, если этот тип используется. Приведем пример (листинг 1) операций с числами типа ufixed.

Листинг 1. Операций над числами с фиксированной точкой

```
library IEEE;
use IEEE.STD_LOGIC_1164.a11;
use IEEE.FIXED_PKG.a11;
entity prov_fix_type is
end;
architecture beh of prov_fix_type is
    signal a: ufixed (3 downto –3) :- B"0110_100"; -- 6.5
    signal b: ufixed (4 downto –2) := B"0110_001"; -- 12,25
    signal c: ufixed (5 downto –3);
    signal d: ufixed (5 downto –3);
    signal e: ufixed (8 downto –5);
    signal k: ufixed (5 downto –8);
begin
    process
    begin
        c <= a + b;
        d <= b – a;
        e <= a * b;
        k <= a/b;
    end process;
end beh;
```

Число разрядов дробной части определяется числом, стоящим в правой части диапазона, например, декларация сигнала k предполагает, что целая часть числа k имеет шесть разрядов (номера этих разрядов с пятого по нулевой), а дробная часть имеет восемь двоичных разрядов — это разряды с номерами −1, ..., −8. При моделировании и просмотре временных диаграмм выдаются только значения компонент векторов (точка не ставится!), пользователь должен сам следить за числом разрядов в целой и дробной части числа.

Обозначим в общем случае диапазоны для операндов a, b:

a (l_a downto r_a), b (l_b downto r_b)

для того чтобы записать размерности операндов c, d, e, k типа ufixed, представляющих собой результаты арифметических операций (см. листинг 1).

Результат сложения: c = 010010.110 (18,75 в десятичной системе счисления).

Размерность результата сложения, т. е. числа c = a + b — это диапазон

(maximum (l_a, l_b) + 1 downto minimum (r_a, r_b)).

Результат вычитания: d = 0101.110 (5,75), резервируемая размерность результирующего вектора d = b − a — такая же, как при сложении.

Результат умножения: e = 001001111.10100 (79,625).

Размерность результата умножения, т. е. числа e = a × b — это диапазон

(l_a + l_b + 1 downto r_a + r_b).

Результат деления: k = 000000.10001000, что соответствует $k = 2^{-1} + 2^{-5} = 17:32 = 0{,}53125$, и это есть лучшее приближение к числу k = 6,5:12,25 = 0,53061. Если же рассмотреть двоичное число 000000.10000111_2, то в десятичной системе счисления оно будет равно $2^{-1} + 2^{-6} + 2^{-7} + 2^{-8} = 0{,}52734$ (что является худшим приближением к числу 0,53061 по сравнению с числом 0,53125).

Размерность результата деления, т. е. числа k = a/b — это диапазон

($l_a – r_b$ downto $r_a – l_b – 1$).

В рассматриваемом примере $l_a = 3$, $r_a = -3$, $l_b = 4$, $r_b = -2$. Для результата операции деления диапазон следующий (5 downto −8), так как

$l_a – r_b$ = (+3) – (–2) = 5;
$r_a – l_b – 1$ = (–3) – (+4) – 1 = –8.

Для работы с числами с фиксированной точкой, имеющими знак, используется тип sfixed, который определен в VHDL-пакете fixed_pkg:

```
type sfixed is array (integer range <>) of std_logic;
```

В этом пакете приводятся и соответствующие функции, поддерживающие арифметические операции с данным типом.

Использование чисел с плавающей точкой

Для работы с числами с плавающей точкой используется VHDL-пакет float_pkg, в котором декларирован тип float:

```
type float is array (integer range <>) of std_logic;
```

При использовании типа float важным вопросом является следующий: какое вещественное число представляет конкретный двоичный вектор, интерпретируемый как float.

Пусть сигнал a типа float имеет следующую декларацию:

```
signal a : float (N downto –M);
```

Таблица 1

Назначение	Знак (S)	Экспонента (E)			Дробная часть (F)				
Номер разряда диапазона (N downto-M)	N	N – 1	N – 2...1	0	–1	–2	–3...–(M – 2)	–(M – 1)	–M

В табл. 1 представлено разбиение диапазона (N downto $-M$) из $N + M + 1$ разрядов на три группы: знак числа, значение экспоненты, значение дробной части. Самый левый N-й бит — знак (S) числа (значение 1 этого бита соответствует отрицательному числу), следующие N бит — экспонента (E), следующие M бит — дробная часть (F).

Для перевода двоичного вектора типа float в вещественное число R (тип real) используется следующая формула:

$$R = S \times 2^{(E - E_{base})} \times (1{,}0 + (F/F_{base})),$$

где

S — знак числа (плюс (+1) либо минус (−1));

$E_{base} = 2^{N-1} - 1$ (база экспоненты);

E — целое число (экспонента), заданное в двоичном коде разрядами с номерами ($N - 1$, ..., 0), левый ($N - 1$)-й разряд является старшим;

$F_{base} = 2^M$ (база дробной части);

F — целое число (дробная часть), заданное в двоичном коде разрядами с номерами (-1, ..., $-M$), левый разряд является старшим.

В примере

```
signal A : float (5 downto -6):= b"1_01111_010000";
-- число --1.25
```

для сигнала A типа float имеем следующие значения величин: $S = 1$ (знак минус, т. е. двоичный вектор представляет отрицательное число), $N = 5$, $M = 6$, $E_{base} = 15$, $E = 15$, $F_{base} = 64$, $F = 16$. Следовательно,

$$R = (-1) \times 2^{(15-15)} \times (1{,}0 + (16/64)) = \\ = -1(1{,}0 + 0{,}25) = -1{,}25.$$

Для выполнения операций с векторами типа float требуется подключение пакета FLOAT_PKG. Ниже в листинге 2 дан пример представления чисел с плавающей точкой. Функция to_real(F) является функцией преобразования типа — эта функция преобразует операнд F типа float в операнд типа real, который легко воспринимается на временной диаграмме при моделировании. Конечно, в пакете FLOAT_PKG есть и много других функций преобразования типов, в том числе функция преобразования типа real в тип float.

Листинг 2. Операции над числами с плавающей точкой

```
library IEEE;
use IEEE.STD_LOGIC_1164.a11;
use IEEE.FLOAT_PKG.a11;

entity prov_float is
end;

architecture beh of prov_float is
    signal a, b : float (5 downto –6):= b"1_01111_010000";
    -- число –1.25
    signal c, d, e, k : float (5 downto –6);
    signal a_real, b_real, c_real, d_real, e_real, k_real : real;
begin
    c <= a + b;
    d <= b – a;
    e <= a * b;
    k <= a/b;
    a_real <= to_real(a);
    b_real <= to_real(b);
    c_real <= to_real(c);
    d_real <- to_real(d);
    e_real <= to_real(e);
    k_real <= to_real(k);
end beh;
```

Результат выполнения программы (листинг 2) представлен в табл. 2.

Таблица 2

Число	Операция	Тип float	Тип real
a	—	1_01111_010000	–1,25
b	—	1_01111_010000	–1,25
c	a + b	1_10000_010000	–2,5
d	b – a	0_00000_000000	0,0
e	a*b	0_01111_100100	1,5625
k	a/b	0_01111_000000	1,0

Если экспонента представляется нулевым вектором, то для перевода двоичного вектора типа float в вещественное число R используется следующая формула:

$$R = S \times 2^{(1 - E_{base})} \times F.$$

Например,

```
signal A : float (5 downto –6):= b"0_00000_100000";
```

представляет число

$$R = (+1) \times 2^{(1-15)} \times (1/2) = 2^{-15} = 0{,}0000305176.$$

Представление вещественного нуля является исключением — нуль представляется нулевым вектором типа float.

Иерархические ссылки на сигналы

При тестировании программ стандарт VHDL'2008 позволяет осуществлять доступ из тестирующей программы к любому сигналу на любом иерархическом уровне описания проекта цифровой системы.

Рассмотрим пример доступа к внутреннему сигналу state типа T_state компонента Mealy в тестирующей программе; предполагается, что тестируется компонент Mealy. Чтобы получить значение сигнала state, нужно провести в тестирующей программе декларацию типа T_state, затем декларировать сигнал State_top (имя выбирает пользователь), а затем присвоить значение этому сигналу, осуществив иерархическую ссылку. В правой части оператора

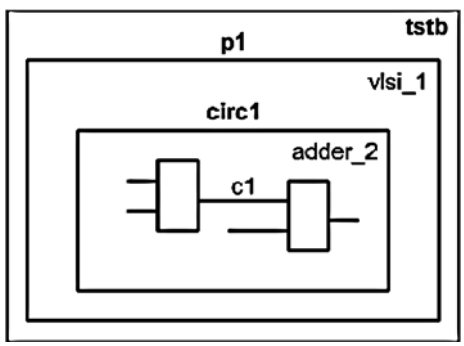
```
State_top <= <<signal. tstb.p0. state : T_state>>;
```

назначения сигнала стоит иерархическое имя, оно берется в двойные угловые скобки. Ссылка идет на имя головного модуля .tstb, затем указывается метка p0, по которой находится модуль Mealy, через точку указывается имя (state) сигнала и в конце указывается тип T_state сигнала:

```
entity tstb is
end;
architecture beh of tstb is
type T_state is (a1, a2, a3, a4, a5, a6);
signal state_top: T_state;
. . .
begin
p0: Mealy port map (x1, x2, x3, x4, clk, rst, y1, y2, y3, y4, y5, y6);
State_top <= <<signal p0. state : T_state>>;
-- вид 1 ссылки
-- State_top <= <<signal . tstb.p0. state : T_state>>;
-- вид 2 ссылки
```

При выполнении моделирования сигнал State_top можно наблюдать на временной диаграмме вместе с сигналами головного модуля tstb.

Приведем еще один пример. Пусть в тестирующей программе tstb (имя entity) меткой p1 помечен оператор port map создания экземпляра компонента vlsi_1. Компонент vlsi_1 описан на структурном уровне и в архитектурном теле содержит оператор port map с меткой circ3 создания экземпляра компонента adder_2. Пусть компонент adder_2 содержит внутренний сигнал c1 (см. рисунок). Доступ через два уровня иерархии к этому сигналу c1 из тестирующей программы tstb осуществляется указанием меток для операторов port map в иерархически вложенных модулях (entity):



Формирование имени сигнала c1 в тестирующей программе

```
<<signal . tstb.p1. circ3 . cl : std__logic>>;
```

Приведенное иерархическое имя сигнала c1 использует путь, начинающийся именем головного модуля tstb, затем идут имена меток p1 и circ1 в иерархии подчиненных модулей, заканчивается путь именем c1 сигнала в модуле adder_2. Отметим тот факт, что в иерархическом имени используются имена меток и не используются имена иерархически вложенных entity.

Фраза "all" для списка чувствительности процесса

В табл. 3 приведены два примера описания процесса: для стандарта VHDL'2008 можно в списке чувствительности процесса написать ключевое слово all и не перечислять все сигналы, как это требовалось в стандарте VHDL'1993.

Таблица 3

VHDL'1993	VHDL'2008

process (x1, x2, x3, x4, selection)
begin
case selection is
when "00" => F <= x1;
when "01" => F <= x2;
when "10" => F <= x3;
when others => F <= x4;
end case;
end process;
	process (all)
	begin
	case selection is
	when "00" => F <= x1;
	when "01" => F <= x2;
	when "10" => F <= x3;
	when others => F <= x4;
	end case;
	end process;

Упрощенный оператор case (выбора)

В стандарте VHDL'2008 оператор case был упрощен для анализа выражений, использующих векторы типа std_logic_vector. Случаи, анализируемые case, могут быть записаны в виде выражений, в которых используются операторы и функции из пакетов STD_LOGIC_1164, NUMERIC_STD, NUMERIC_ UNSIGNED:

```
process (x, y)
constant z : std_logic_vector (1 downto 0) := "10";
constant w1 : std_logic_vector (1 downto 0) := "10";
constant w2 : std_logic_vector (1 downto 0) := "11";

begin
   case ((x xor y) & z) is
      " when "1010" => F <= "1000" ;
      when w1 & w2 => F <= "0100" ;
      when "001-" => F <= "0010" ;
      when "1101" => F <= "0001" ;
      when "01--" => F <= "1111" ;
      when others => F <= "0110" ;
   end case ;
end process ;
```

Условия в операторе case должны быть локально статичными. Например, нельзя записать

```
when (x and y)& w2 => F <= "0110" ;
```

так как выражение (x and y) & w2 не является константным.

Безразличные условия в операторах case

Для типа std_logic_vector в стандарте VHDL'2008 введен новый оператор case?, в котором компоненты записываемых случаев (компоненты векторов) могут принимать три значения: 0, 1, — (don't care). Случаи должны быть записаны в виде ортогональных троичных векторов. Ортогональной считается пара значений (0, 1) соответствующего разряда вектора, неортогональными являются пары (0,0), (1,1), (−,-), (−,1), (−,0). В приведенном ниже примере имеется пара "01--", "0--1" неортогональных четырехразрядных векторов:

```
x : std_logic_vector (3 downto 0);
F : unsigned (3 downto 0)) ;
. . .
-- ошибочный оператор case
case? x is
    when "1---" => F <= "1000" ;
    when "01--" => F <= "0100" ;
    when "001-" => F <= "0010" ;
    when "0001" => F <= "0001" ;
    when "0--1" => F <= "1111" ;
    when others => F <= "0000" ;
end case? ;
```

Это является недопустимым — при компиляции сообщается об ошибке и указывается эта неортогональная пара векторов.

Каждая пара векторов должна быть ортогональна хотя бы по одному разряду вектора, как это показано ниже:

```
case? x is
    when "10--" => F <= "1000" ;
    when "01--" => F <= "0100" ;
    when "001-" => F <= "0010" ;
    when "0001" => F <= "0001" ;
    when "11--" => F <= "1111" ;
    when others => F <= "0000" ;
end case? ;
```

Упрощенные условные выражения

В предыдущих стандартах VHDL оператор if позволял проверять значение true, false выражения типа boolean. В стандарте VHDL'2008 оператор if может проверять значения типа std_logic, при этом значения '1', 'H' интерпретируются как true, остальные значения типа std_logic — как false. Оператором if могут проверяться также значения типа bit, в этом случае значение '1' интерпретируется как true, значение '0' — как false.

В пакет STD_LOGIC_1164 добавлены пять новых операторов сравнения ?=, ?/=, ?>=, ?<, ?<=, которые возвращают значения '0', '1' типа std_ulogic либо значения '0', '1' типа bit в зависимости от типа проверяемого значения. Однако, как и ранее, при записи выражений нельзя смешивать типы. В примере ниже операторы if проверяют выражения типа bit либо типа std_logic:

```
signal a, b, c, x : std_logic;
signal d_b: bit := '1';
signal e_b: bit := '0';
. . .
if (a or b) then x <= '1';
else x <= '0';
end if;
. . .
if (d_b and e_b) then x <= '0';
else x <= ' 1' ;
end if;
. . .
if ((a or b) and ((a or b) ?= '1')) then
    x <= ((a or b) ?= '1') ;
end if;
```

Проверяемое выражение (a or b ?= '1') может быть присвоено сигналу x типа std_logic.

Выражения в картах портов

В VHDL'2008 при указании сигналов, подаваемых на входные порты при конкретизации компонента (т. е. в операторе port map), можно записывать выражения с логическими операторами. Выражение от сигналов вносит дельта-задержку при моделировании. Если же выражение есть простой сигнал, константа либо функция преобразования типа, то дельта-задержка не вносится. В примере показан оператор port map с меткой circ1, использующий позиционное соответствие, при этом на входной порт t3 подается выражение ((a3 and a2) or a1), в котором участвуют сигналы, подаваемые на другие входные порты:

```
component MUX
port (t4, t3, t2, t1 : in std_logic;
    d4, d3, d2, d1 : out std_logic);
end component;
. . .
circl: MUX port map (a3, ((a3 and a2) or a1), a2, c,
    y4, y3, y2, y1) ;
```

Чтение выходных портов

В VHDL'2002 (и более ранних стандартах) сигналы, являющиеся выходными портами, могли только получать значения в архитектурном теле, а "отдавать" свои значения в этом же архитектурном теле они не могли. В примере ниже в модуле

out_port используется выходной порт y1 для вычисления значения выражения, присваиваемого другому выходному порту y2. Это называется "читать выходной порт y1". В VHDL'1993 (при моделировании и при синтезе) это недопустимо, а для VHDL'2008 при моделировании это разрешается:

```
entity out_port is
port (
    x1, x2, x3 : in bit;
    y1, y2 : out bit);
end out_port;
architecture beh of out_port is
begin
    y1 <= (x1 and x2);
    -- чтение выходного порта y1
    y2 <= (x1 xor x3) or y1;
end beh;
```

Ранее приходилось вводить дополнительный сигнал z, чтобы подать его в архитектуру

```
-- VHDL'1993
architecture beh of out_port is
-- дополнительный внутренний сигнал z
    signal z : bit;
begin
    z <= (x1 and x2);
    y1 <= z;
    y2 <= (x1 xor x3) or z;
end beh;
```

либо требовалось объявлять выходной (out) порт y1 как двунаправленный (inout).

Условные и выборочные присваивания в участках последовательного кода

К оператору <= назначения сигнала, интерпретируемого в зависимости от контекста то как параллельный, то как последовательный, в стандарте VHDL'2008 добавились операторы условного и выборочного назначения сигнала. Оператор *условного назначения сигнала*

```
z <= x when x > y else y;
```

в стандарте VHDL'2008 интерпретируется как последовательный среди последовательных операторов в процессах, функциях, процедурах, например,

```
process(clock)
begin
    if rising_edge(clock) then
        q <= '0' when reset else d;
    end if;
end process;
```

Ниже приведен пример (для стандарта VHDL'2008) применения другого параллельного оператора *выборочного назначения сигнала* (with ... select ...) в процессе, т. е. в участке последовательного кода. Ранее этот оператор допустимо было писать в участках для написания параллельных операторов, например в архитектурном теле.

```
process(clock)
begin
    if rising_edge(clock) then
        with s select
            q <=  a when "00",
                  b when "01",
                  c when "10",
                  d when "11";
    end if;
end process;
```

Оператор select в стандарте VHDL'2008 может использоваться для присваивания значений не только сигналам, но и переменным:

```
variable q : natural := 0;
. . .
with k select
    q := w(1) when "00",
         w(2) when "01",
         w(3) when "10",
         w(4) when "11";
```

Упрощенная запись строк символов

В стандарте VHDL'2008 можно сокращенно записывать строки символов, пользуясь 16-, 10-, 8-, и 2-ичной системами счисления, при этом, как и в предыдущих стандартах, для обозначения системы счисления используются обозначения:

- символ X соответствует 16-ричной;
- символ D соответствует 10-ичной;
- символ O соответствует 8-ричной;
- символ B соответствует 2-ичной системе счисления.

В табл. 4 представлены примеры [3] сокращенной записи символьных литералов в различных системах счисления.

В примере 1: в сокращенной записи символ O означает 8-ричную систему (три разряда для представления символа), цифра 3 представляется как 011, символ X повторяется три раза (это не есть цифра),

Таблица 4

Номер примера	Полная запись VHDL'1993	Сокращенная запись VHDL'2008
1	"011XXXZZZ100"	O"3XZ4"
2	"10100011--------"	X"A3--"
3	"010UU"	5B"10UU"
4	"011XXX"	O"3_X"
5	"11111111####????1111"	20SX"F#?F"
6	"0111100"	7X"3C"
7	"00000101"	8O"5"
8	"000000000X"	10B"X"
9	"11_1100"	6X"FC" ошибка

символ Z тоже не цифра — повторяется три раза, 4 — это цифра (представляется как 100).

В примере 2: символ X означает 16-ричную систему (4 разряда для представления символа), символ A является цифрой в 16-ричной системе и представляется как 1010, цифра 3 представляется как 0011.

В примере 6: цифра 7 в символьной записи 7X"3C" означает число разрядов в векторе, символ X свидетельствует о 16-ричной системе счисления, символ C представляется как 1100, цифра 3 представляется как 011, так как при указании длины (семь разрядов) последовательности отбрасываются крайние левые символы, причем эти отбрасываемые символы обязательно должны быть нулями. В данном примере из представления 0011 цифры 3 отброшен крайний левый нуль.

В примере 7: цифра 8 означает число символов (длину последовательности), цифра 5 представляется как 101, впереди комбинации 101 добавляется пять нулей, чтобы всех символов было восемь.

Пример 9 сокращенной записи является ошибочным, так как отбрасываемые в начале последовательности символы 11 не являются нулями. Ниже в примере даны декларации сигналов, упоминаемых в табл. 4:

```
signal s1 : std_logic_vector (1 to 12):= O"3XZ4";
signal s2 : std_logic_vector (1 to 16):- X"A3--";
signal s3 : std_logic_vector (1 to 5):= 5B"10UU";
signal s4 : std_logic_vector (1 to 6):= O"3_X";
signal s5 : string(1 to 20) := 20SX"F#?F";
signal s6 : std_logic_vector(1 to 7):= 7X"3C";
signal s7 : std_logic_vector (1 to 8):= 8O"5";
signal s8 : std_logic_vector(1 to 10):= 10B"X";
signal s9 : std_logic_vector (1 to 6):= 6X"FC"; -- ошибка
```

В VHDL'1993 и VHDL'2002 для задания битовых строк были допустимы только 8-ричная, 10-ичная и 16-ричная системы счисления, при этом длина вектора была кратна трем либо четырем битам, длина вектора не указывалась, допустимыми были только символы '0', '1' и символ подчеркивания.

Унарная редукция логических операторов

В стандарте VHDL'2008 к векторам типа bit_vector, std_logic_vector, signed, unsigned можно применять унарную операцию логических операторов and, or, nand, nor, xor, xnor, понимаемую как последовательное выполнение одного и того же логического оператора над компонентами вектора. Ниже в примерах комментариями записаны последовательности соответствующих операций для VHDL' 1993:

```
signal a: bit_vector (1 to 4):= "0111";
signal b, c, d, e, f, g: bit;
. . .
b <= and (a); -- b <= a(1) and a(2) and a(3) and a (4);
c <- nand (a); -- c <= not(a(1) and a(2) and a(3) and a(4));
d <= or (a); -- d <= a(1) or a(2) or a(3) or a(4);
e <= nor (a); -- e <= not(a(1) or a(2) or a(3) or a(4));
f <= xor (a); -- f <= a(1) xor a(2) xor a(3) xor a(4);
g <= xnor (a); -- g <= a(1) xnor a(2) xnor a(3) xnor a(4);
```

Обратим внимание на выполнение унарной редукции оператора nand — сначала над компонентами вектора выполняется оператор and, затем к полученному результату применяется унарный оператор not. Аналогично и для оператора nor — сначала выполняется оператор or для компонент вектора, а затем результат инвертируется с помощью оператора not.

Для векторов с большим числом компонент при использовании стандарта VHDL'1993 для получения значений сигналов b, c, d пришлось бы, возможно, написать соответствующие циклы.

Логические операторы вектора со скаляром

Можно выполнять логические операции одного бита (скаляра) с массивом битов (вектором): логический оператор применяется отдельно для скаляра и каждого разряда вектора — так формируется разряд результирующего вектора. В качестве векторов могут быть std_logic_vector (массивы значений типа std_logic), signed, unsigned, bit_vector:

```
signal a, b : std_logic_vector (1 to 4);
signal x : std_logic;
. . .
b <= a or x;
        -- b(1) <= a(1) or x;
        -- b(2) <= a(2) or x;
        -- b(3) <= a(3) or x;
        -- b(4) <- a(4) or x;
```

Установка константных значений force, release

В тестирующей программе, написанной в стандарте VHDL'2008, можно использовать операторы force, release для установки константных значений сигналов:

```
A <= force '1';
```

Можно написать A <= force in '1'; для входных сигналов (портов) и A <= force out '1'; для выходных сигналов, по умолчанию in, out можно пропускать. Отмена force осуществляется путем команды

```
A <= release;
```

В одном и том же процессе в тестирующей программе можно установить значение сигнала с помощью force, а затем отменить:

```
if (stop_force < 4) then
    DATA(0) <= force in '1' ;
    D(4) <= force out '1';
elsif (stop_force >= 4) then
    D(4) <= release;
end if;
```

Декларации контекстов

Под декларацией контекстов понимается возможность оформления деклараций библиотек (и пакетов из библиотек) в виде отдельных модулей проекта, называемых *контекстами*. Если имеются ссылки на используемые библиотеки и пакеты, то данную часть VHDL-кода можно представить в виде контекста, дать контексту имя. Рассмотрим пример контекста с именем Project1

```
Context Project1 is
    library IEEE;
    use IEEE.STD_LOGIC_1164.a11;
    use STD.TEXTIO.a11;
    use IEEE.STD_LOGIC_TEXTIO.a11;
    use IEEE.NUMERIC_STD.a11;
end;
```

Теперь имеется возможность ссылки на контекст Project1 в других VHDL-кодах, а именно, в тех местах, где декларируются пакеты, можно написать ссылку

```
library work;
context work.Project1;
```

Такая возможность сокращает код и позволяет унифицировать ссылки на пакеты в большом проекте.

Добавление нового пакета ENV в библиотеку STD

В стандарте VHDL-2008 добавлен новый пакет ENV в стандартную библиотеку STD. В пакете ENV вводятся несколько новых процедур и функций:

- STOP — процедура остановки моделирования;
- FINISH — процедура остановки моделирования с выходом из системы моделирования;
- resolution_limit — функция возвращает значения точности моделирования (тип DELAY_ LENGTH).

Соответствующие декларации в пакете ENV имеют вид

```
procedure stop (STATUS : INTEGER := 0);
procedure finish (STATUS : INTEGER : = 0);
function resolution_limit return DELAY_LENGTH;
```

В зависимости от значения параметра STATUS, передаваемого процедурам STOP и FINISH, предусмотрен разный вывод в консоль:

- 0 — нет вывода;
- 1 — выводит время моделирования и путь к entity, в котором выполнена процедура;
- 2 — выводит время, расположение и статистику о задействованной памяти и ресурсу процессора при моделировании;
- другие значения STATUS интерпретируются как 0.

Изменения в пакете STANDARD библиотеки STD

Пакет STANDARD является основополагающим — это единственный пакет, на который не является обязательной ссылка, в нем декларируются базовые типы. Список базовых типов расширен, в пакет STANDARD добавлены типы real_vector, time_vector, integer_vector, boolean_vector, а также функции maximum и minimum для всех базовых типов. В пакет также добавлена функция to_string, которая конвертирует базовые типы (но не векторные базовые типы) в строку символов, а также функции to_hstring, to_ostring преобразования битвектора в строку в 16-ричном либо 8-ричном формате. Добавлены функции rising_edge и falling_edge для определения переднего и заднего фронтов сигнала, которые ранее были определены в пакете NUMERIC_BIT. Использование функций to_string улучшает читаемость кода и сокращает запись для вывода строки (табл. 5). Ввод-вывод строк текстовых строк осуществляется с использованием средств пакета TEXTIO.

Таблица 5

VHDL'1993	VHDL'2008

write (1, string'("Real = "));
write (1, r);
swrite (1, " ""%6.3f"" = ");
write (1, r,"%6.3f");
write (1, LF);
	write (1, "Real = " & to_string(r) &
	" ""%6.3f"" = " & to_string(r,"%6.3f") & LF);

Изменения в пакете TEXTIO библиотеки STD

Пакет текстового ввода/вывода TEXTIO (Textual Input and Output) включает в себя объявления типов и подпрограмм, которые поддерживают чтение из текстового файла и запись в текстовый файл. В данный пакет добавлены процедуры:

- TEE — печать строки в файл и в стандартный выходной поток (на экран/терминал);
- SREAD, SWRITE — чтение и запись строки, например, вместо write (L, string' ("Hello!")); можно использовать swrite (L, "Hello!").

Добавлены также процедуры чтения и записи строк бит-векторов в 16-ричном и 8-ричном форматах, функция центрирования строки по левому либо правому краю другой строки и др.

Стандарт VHDL'2008 привел к появлению новых пакетов и модификации уже существующих пакетов библиотеки IEEE. Эти вопросы достаточно обширны и заслуживают самостоятельного рассмотрения.

Заключение

В статье перечислены практически все новые конструкции VHDL, введенные стандартом VHDL'2008, удобные при написании тестирующих программ и

облегчающие верификацию описаний цифровых систем на этапе алгоритмического проектирования. Использование стандарта VHDL'2008 и соответствующих систем моделирования и верификации сокращает время проверки правильности алгоритмических описаний цифровых систем, доля этого времени постоянно увеличивается в силу возрастания сложности проектируемых цифровых систем.

Список литературы

1. **IEEE** Standard VHDL Language Reference Manual, IEEE Std 1076-2008.
2. http://www.bsuir.by/vhdl/
3. **Ashenden P. J., Lewis J.** VHDL—2008. Just the New Stuff. Burlington, MA, USA: Morgan Kaufman Publishers, 2007. 244 p.
4. **Eisner C., Fismam D.** A Practical Introduction to PSL. N. Y., USA: Springer, 2006. 240 p.
5. **Грушвицкий Р., Михайлов М.** Проектирование в условиях временных ограничений: верификация проектов // Компоненты и технологии. 2008. № 5. С. 54—59.
6. **Using** Protected Types in VHDL Designs. URL: http://www.aldec.com/en/support/resources/documentation/articles/1179.
7. **Ashenden P. J., Lewis J.** The Designer's Guide to VHDL. Third Edition. Burlington, MA, USA: Morgan Kaufmann Publishers. 2008. 909 p.
8. **Авдеев Н. А., Бибило П. Н.** Средства VHDL для функциональной верификации цифровых систем // Современная электроника. 2013. № 3. С. 74—76.
9. **Авдеев Н. А., Бибило П. Н.** Средства VHDL для функциональной верификации цифровых систем. Методология OS-VVM // Современная электроника. 2013. № 5. С. 66—70.

N. A. Avdeev, Senior Researcher, e-mail: avdeev_n@newman.bas-net.by,
P. N. Bibilo, Head of Laboratory, e-mail: bibilo@newman.bas-net.by,
United Institute of Informatics Problems of National Academy of Sciences of Belarus

Enhanced Capabilities of Digital Systems Computer Aided Design Using VHDL'2008

In this paper the new and changed features of VHDL-2008 (standard IEEE Std 1076—2008) are introduced briefly. The emergence of new VHDL features is largely due to the need to write complex testing programs and to make various types of simulations in order to verify the original design specifications of a digital system. The possibility to use PSL (Property Specification Language) is added to standard VHDL'2008. PSL is a fully-oriented to verification of projects using assertions. For functional verification using the random test patterns generation and the functional coverage, the protected type is added, that is based on a concept similar to classes of object-oriented programming. The possibilities appear to use arrays with unconstrained range, fixed and floating-point numbers, hierarchical links to signals, new features for bit-string literals writing, new set of predefined matching relational operators and logical operators and other features. The standard packages are expanded: real_vector, time_vector, integer_vector, boolean_vector types, maximum and minimum functions for all basic types are added to the STANDARD package; new read and write procedures are added to the TEXIO package. New package ENV is added to the standard library STD. The readers are assumed to be familiar already with VHDL'1993.

Keywords: *computer aided design, digital system, VHDL, testing, functional verification*

References

1. **IEEE** *Standard VHDL Language Reference Manual*, IEEE Std 1076-2008.
2. http://www.bsuir.by/vhdl/
3. **Ashenden P. J., Lewis J.** VHDL-2008. *Just the New Stuff. Burlington*, MA, USA: Morgan Kaufman Publishers, 2007. 244 c
4. **Eisner C., Fismam P.** A *Practical Introduction to PSL*. N. Y., USA: Springer, 2006. 240 p.
5. **Grushvickii R., Mihailov M.** Proektirovanie v usloviach vremennih ogranichenii: verifikacia proektov. *Komponenti i technologii*. 2008. N. 5. P. 54—59.
6. **Using** *Protected Types in VHDL Designs*. URL: http://www.aldec.com/en/support/resources/documentation/articles/1179.
7. **Ashenden P. J., Lewis J.** *The Designer's Guide to VHDL*. Third Edition. Burlington, MA, USA: Morgan Kaufmann Publishers. 2008. 909 p.
8. **Avdeev N. A., Bibilo P. N.** Sredstva VHDL dlia funkcionalnoi verifikacii cifrovich sistem. *Sovremennaia electronika*. 2013. N. 3. P. 74—76.
9. **Avdeev N. A., Bibilo P. N.** Sredstva VHDL dlia funkcionalnoi verifikacii cifrovich sistem. Metodologia OS-VVM. *Sovremennaia electronika*. 2013. N. 5. P. 66—70.