Проектирование цифровых устройств на языке VHDL

Введение в VHDL

Краткая история VHDL

VHDL

- VHDL – язык для описания цифровых устройств
  - **VHDL** = **V**HSIC (Very High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage
- Разработан по заказу DoD для описания ASIC
- Промышленный стандарт IEEE:
  - IEEE 1076-1987
  - IEEE 1076-1993 ← используется наиболее широко
  - IEEE 1076-2000 (незначительные изменения)
  - IEEE 1076-2002 (незначительные изменения)

Verilog (vs. VHDL)

- Частная разработка (1983 г.) для моделирования аппаратуры
- По функциональности в целом соответствует VHDL
  - Отсутствует аналог generate
- Отличия в синтаксисе
  - C-подобный (VHDL – Ada-подобный)
- Стандарт IEEE с 1995 г.:
  - IEEE 1364-1995
  - IEEE 1364-2001
  - IEEE 1364-2005 (незначительные изменения)

Пример

VHDL

```
process (clk, rstn)
begin
   if (rstn = '0') then
      q <= '0';
   elsif (clk'event and clk = '1') then
      q <= a + b;
   end if;
end process;
```

Verilog

```
always@(posedge clk or negedge rstn)
begin
   if (! rstn)
      q <= 1'b0;
   else
      q <= a + b;
end;
```

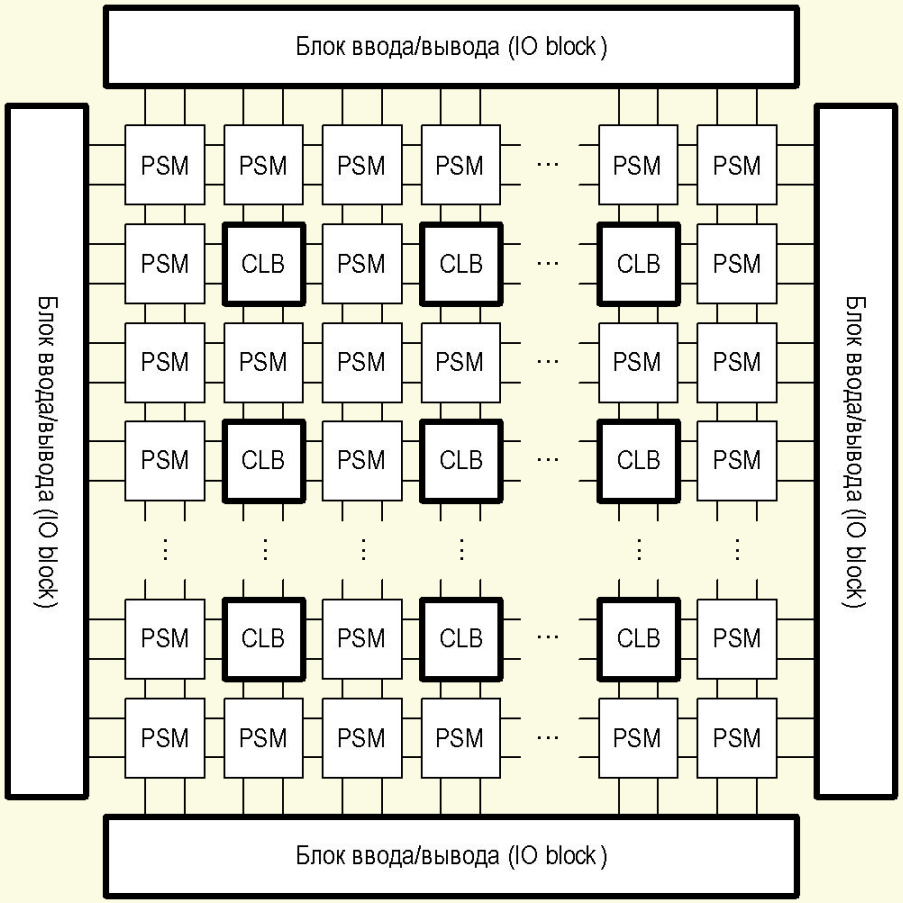
Преимущества VHDL / Verilog

- Промышленный стандарт IEEE
- Не зависит от конкретной технологии или производителя
- Код может использоваться многократно

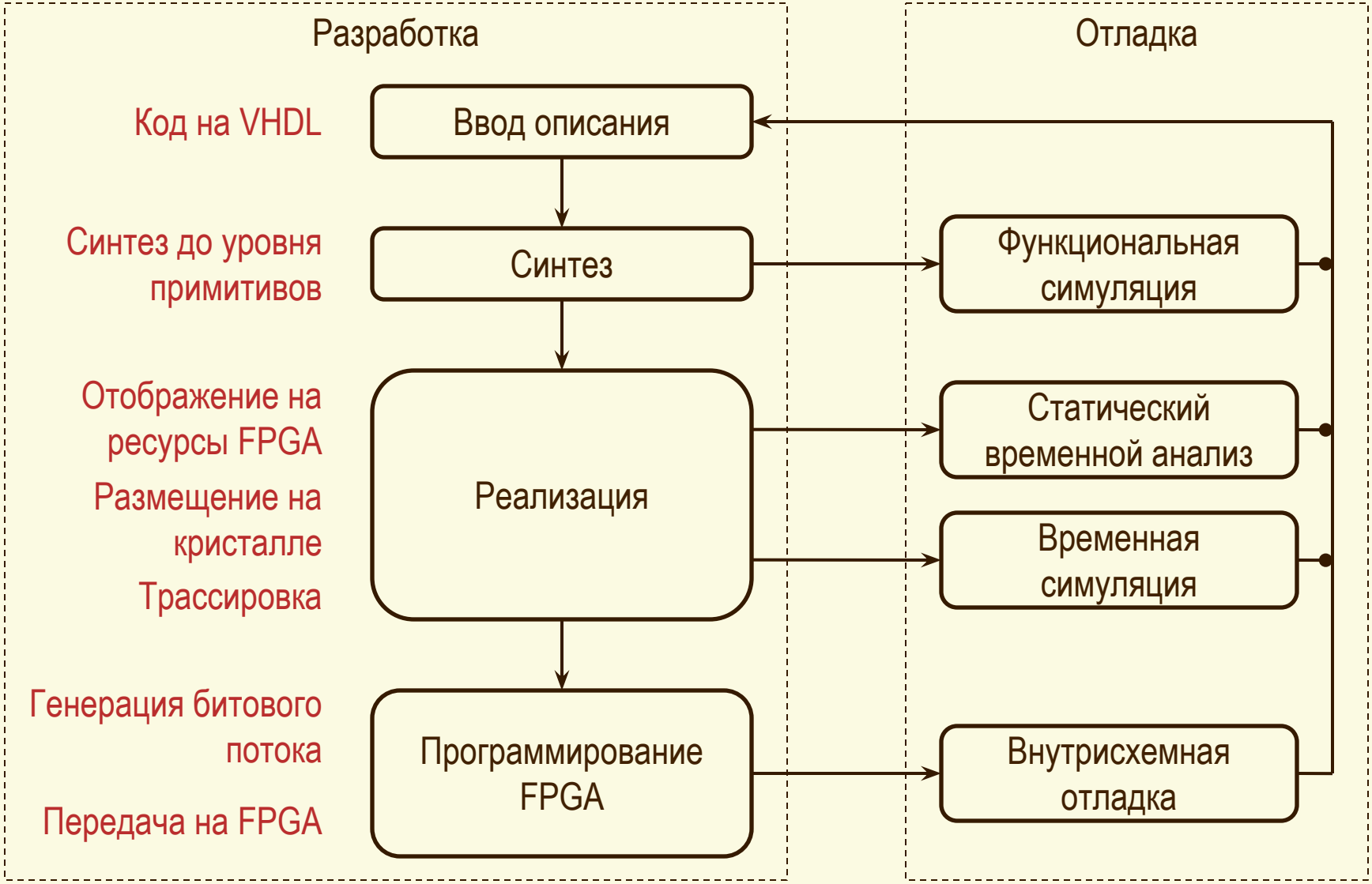
Методология проектирования на VHDL

Что такое FPGA

- **FPGA** = **F**ield **P**rogrammable **G**ate **A**rray
- Микросхема, конфигурируемая пользователем:
  - Элементарные функции – **C**onfigurable **L**ogic **B**locks (**CLB**)
  - Внутренние соединения – **P**rogrammable **S**witch **M**atrices (**PSM**)

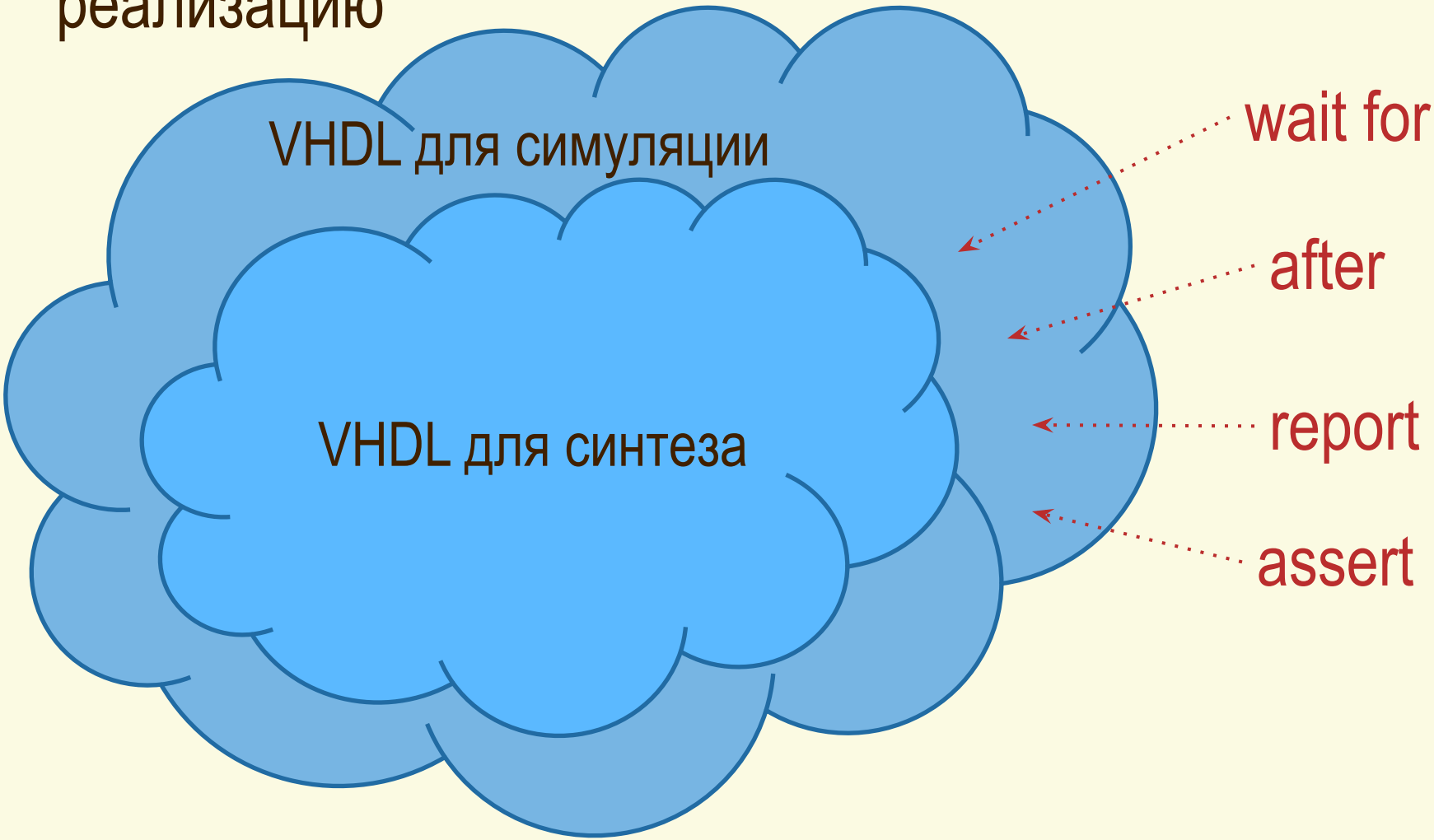


Процесс проектирования

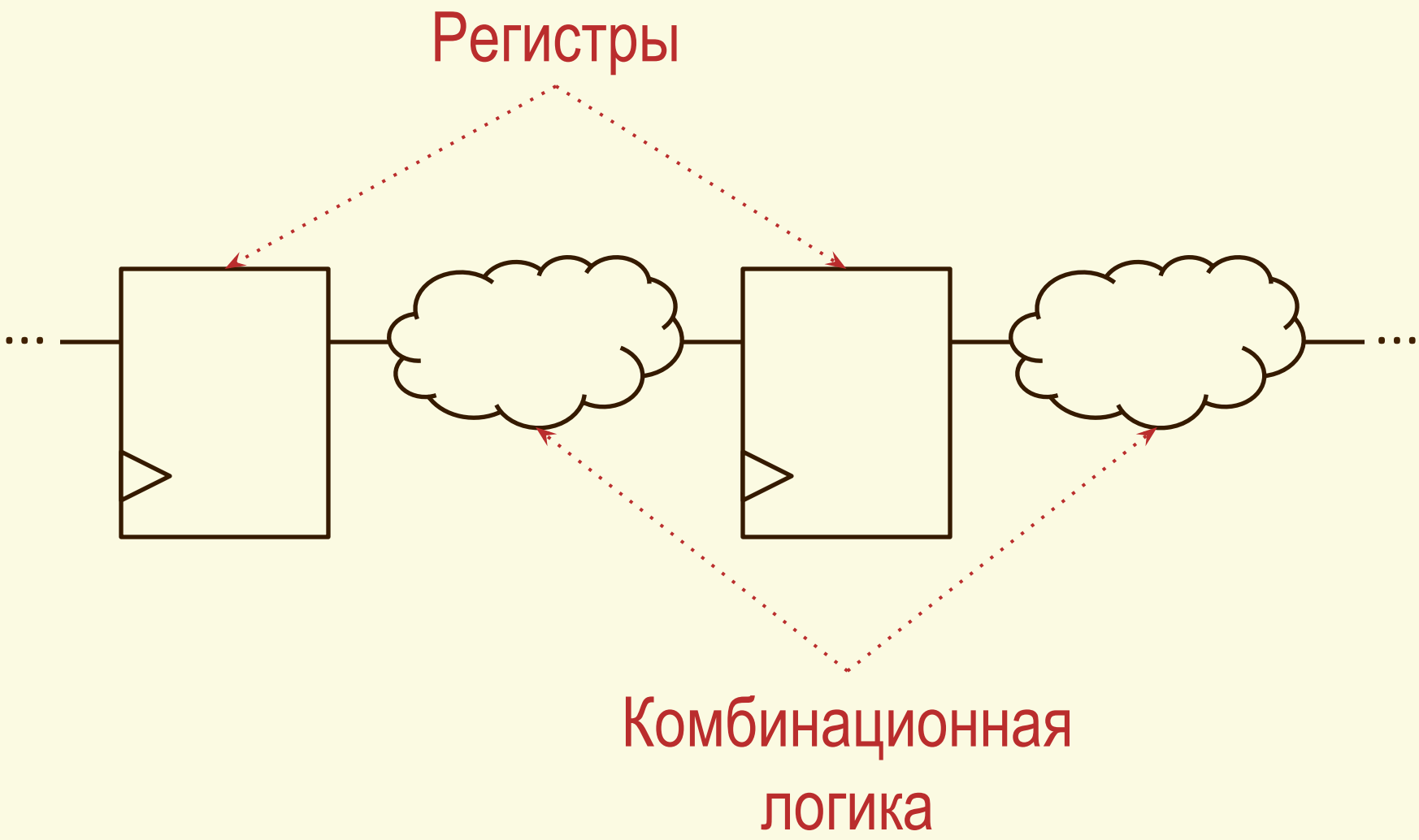


Подмножества VHDL

- Не все конструкции VHDL имеют аппаратную реализацию



Описание на уровне регистровых передач (RTL)

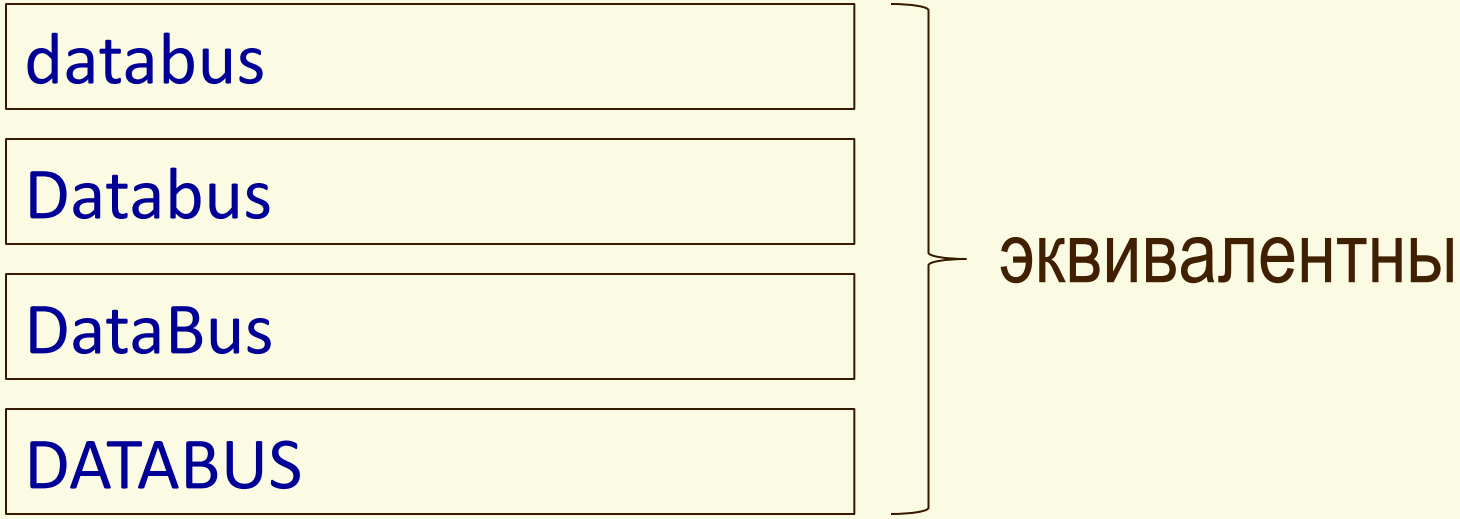



PHY
TPO+
TPO-
TPI+
TPI-
LNKB
LBK
CNTRLB
RBIAS
LEDA#
RJ45INTLED
VCC3_3
VCC3_3

Основы VHDL

Чувствительность к регистру

- VHDL не чувствителен к регистру



Имена и метки

- Должны быть уникальны в пределах области видимости
- Начинаются с буквы (a-z, A-Z)
- Содержат буквы (a-z, A-Z), цифры (0-9), подчеркивания (_)
- Не могут содержать знаков препинания и специальных символов (!, ?, ., &, # и т.д.)
- Два и более подчеркивания подряд запрещены

Формат кода

- VHDL не налагает ограничений на форматирование
- Пробелы и переводы строки эквивалентны

```
if (a = b) then
```

```
if        (a = b)      then
```

```
if (a =
b) then
```

эквивалентны

Комментарии

- Начинается с двойного дефиса (--) в любом месте строки
- Весь текст до конца той же строки считается комментарием (перевод строки заканчивает комментарий)
- Блочные комментарии в VHDL отсутствуют

```
-- обработка сброса
if (rst = ‘1’) then
    counter <= 0; -- обнуление
счетчика
end if;
```

Правила хорошего тона

ESA VHDL Modeling Guidelines

European Space Research and Technology Center

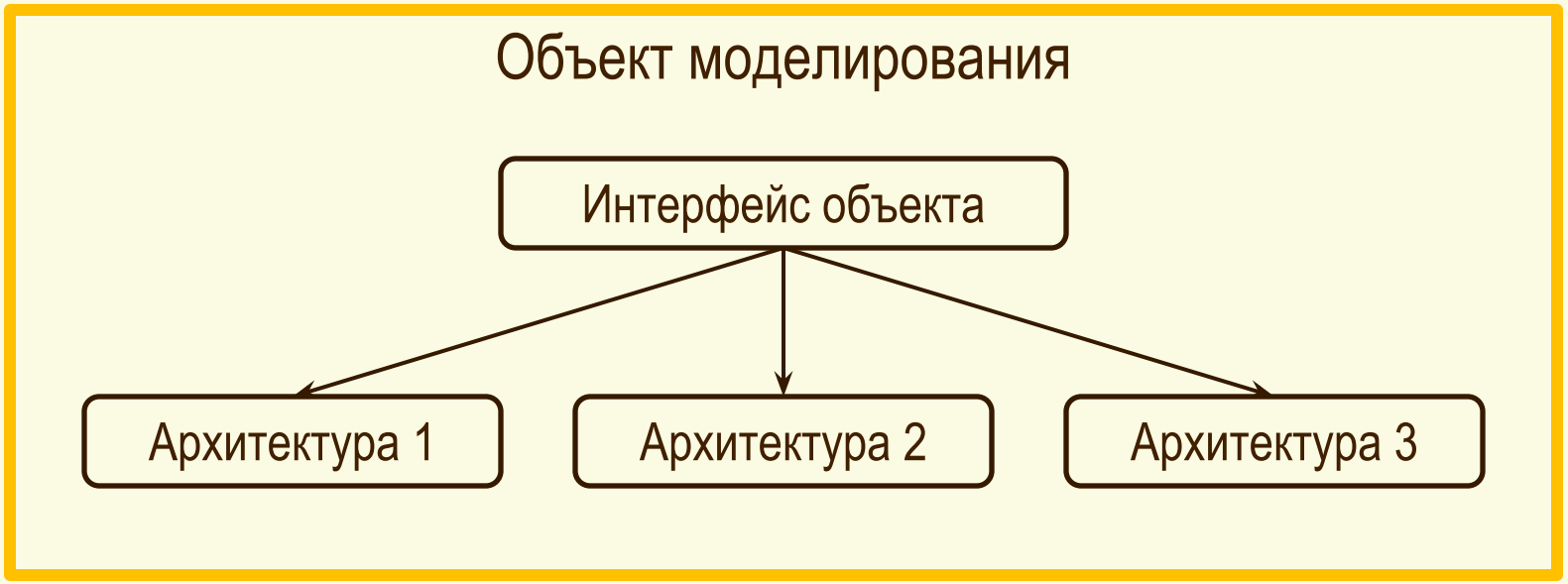
Сентябрь 1994

http://www.vhdl.org/rassp/vhdl/guidelines/ModelGuide.pdf

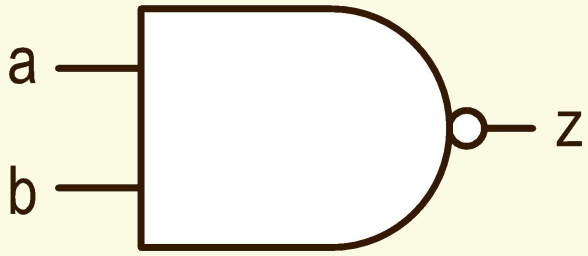
Объект моделирования (design entity)

Объект моделирования

- Базовый блок проекта
- Состоит из интерфейса и архитектур (одной или более)



Пример: элемент NAND (И-НЕ)



a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

Пример: реализация на VHDL

- Расширение файла .vhd
- Один файл – один объект
- Имя файла обычно совпадает с именем объекта

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
        port (
                a       : in std_logic;
                b       : in std_logic;
                z       : out std_logic);
end nand_gate;

architecture model of nand_gate is
begin
        z <= a nand b;
end model;
```

секция импорта

интерфейс объекта моделирования

архитектура объекта моделирования

Секция импорта

- Определяет используемые библиотеки
- Библиотека – набор участков кода, сгруппированных для повторного использования

```
library library_name;
use library_name.package_name.package_parts;
```

Секция импорта – пример

подключение библиотеки

```
library ieee;
use ieee.std_logic_1164.all;
```

использовать все объекты из пакета std_logic_1164 библиотеки ieee

Структура библиотеки

Основные библиотеки

- **ieee**
  - Определяет многозначную систему логики (типы std_logic, std_logic_vector)
- **std**
  - Определяет стандартные типы данных и связанные с ними операции
- **work**
  - Объекты, создаваемые пользователем (после синтеза)

необходимо явное объявление в секции импорта

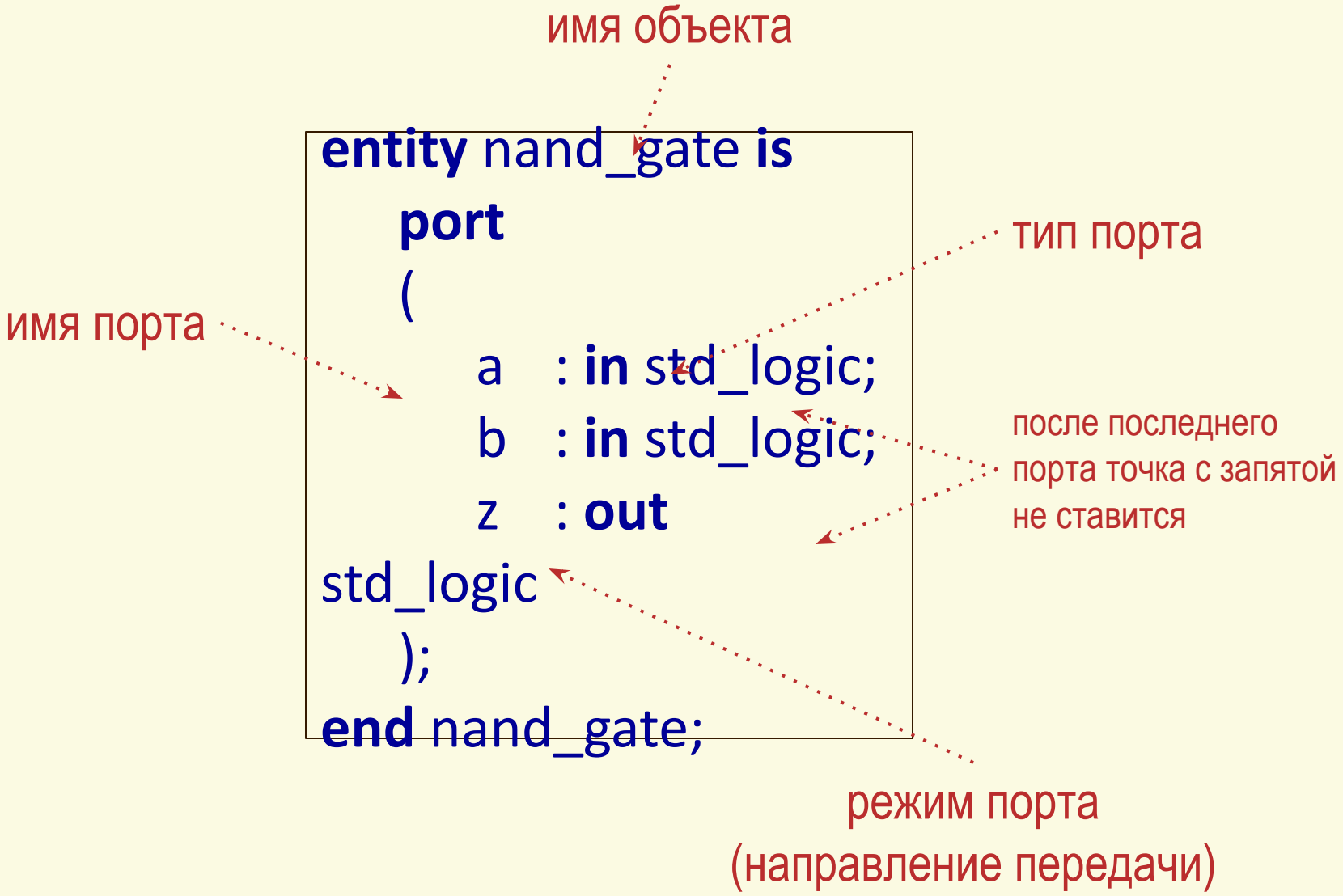
доступны по умолчанию

Интерфейс объекта

- Определяет вид объекта для внешнего мира («черный ящик») – входные и выходные сигналы

```
entity entity_name is
    port (
        port_name: port_mode signal_type;
        port_name: port_mode signal_type;
        . . . . . .
        port_name: port_mode signal_type
    );
end entity_name;
```

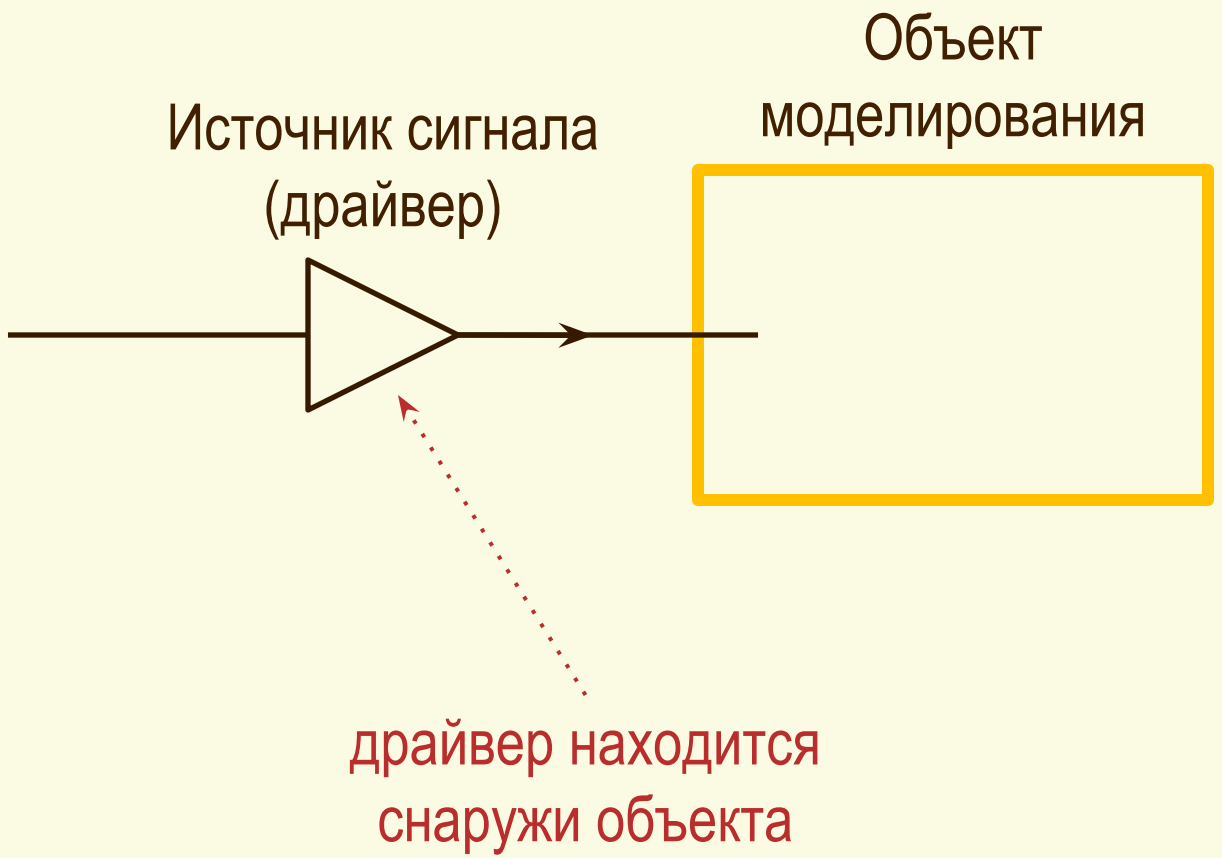
Интерфейс объекта – пример



Типы портов

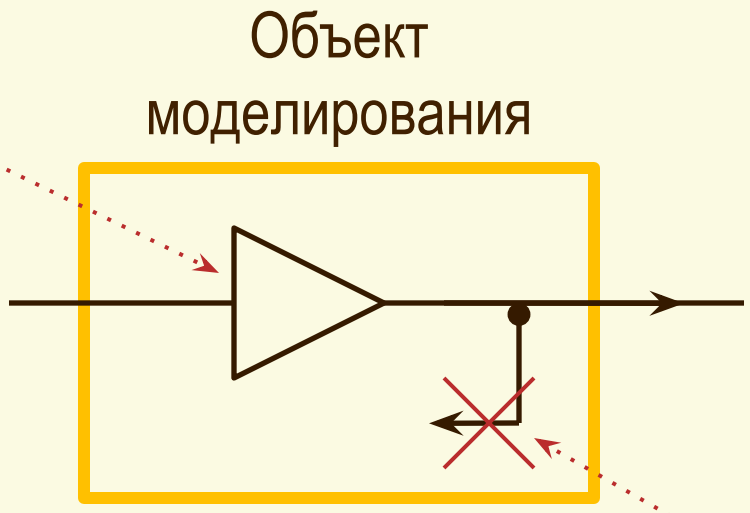
- Тип порта определяет направление передачи данных с точки зрения объекта моделирования
  - Тип **IN**: входной порт, внутри объекта разрешено только чтение (правая часть оператора присваивания)
  - Тип **OUT**: выходной порт, внутри объекта разрешена только запись (левая часть оператора присваивания)
  - Другие типы: **INOUT** (двунаправленный), **BUFFER** (буферный) – использовать не будем

Тип порта IN


Объект
моделирования
Источник сигнала
(драйвер)
драйвер находится
снаружи объекта

Тип порта OUT

драйвер находится внутри объекта



чтение порта внутри объекта запрещено

Архитектура объекта

- Описывает реализацию объекта
- Имя архитектуры должно быть уникально в пределах объекта

```
architecture architecture_name of entity_name is
    declarations
begin
    code
end architecture_name;
```

Архитектура объекта – пример

имя архитектуры

имя объекта

```
architecture model of nand_gate is
begin
    z <= a nand b;
end model;
```

описание функционала
(реализация)

Пример

nand_gate.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
	port (
		a	: in std_logic;
		b	: in std_logic;
		z	: out std_logic);
end nand_gate;

architecture model of nand_gate is
begin
	z <= a nand b;
end model;
```

Тип STD_LOGIC

STD_LOGIC

nand_gate.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
        port (
                a       : in std_logic;
                b       : in std_logic;
                z       : out std_logic);
end nand_gate;

architecture model of nand_gate is
begin
        z <= a nand b;
end model;
```

???

используется как логический тип

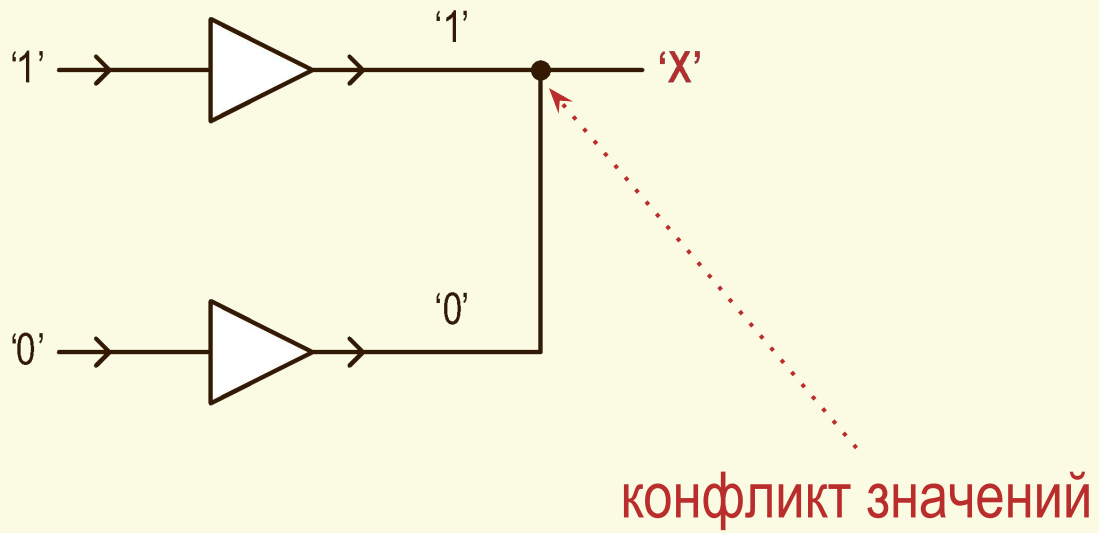
BIT vs. STD_LOGIC

- BIT – встроенный логический тип
  - Значения: ‘0’, ‘1’
- STD_LOGIC – девятизначная логика
  - Значения: ‘U’, ‘0’, ‘1’, ‘X’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘-’
  - Синтезируемые: ‘0’, ‘1’, ‘Z’
  - Остальные используются при симуляции

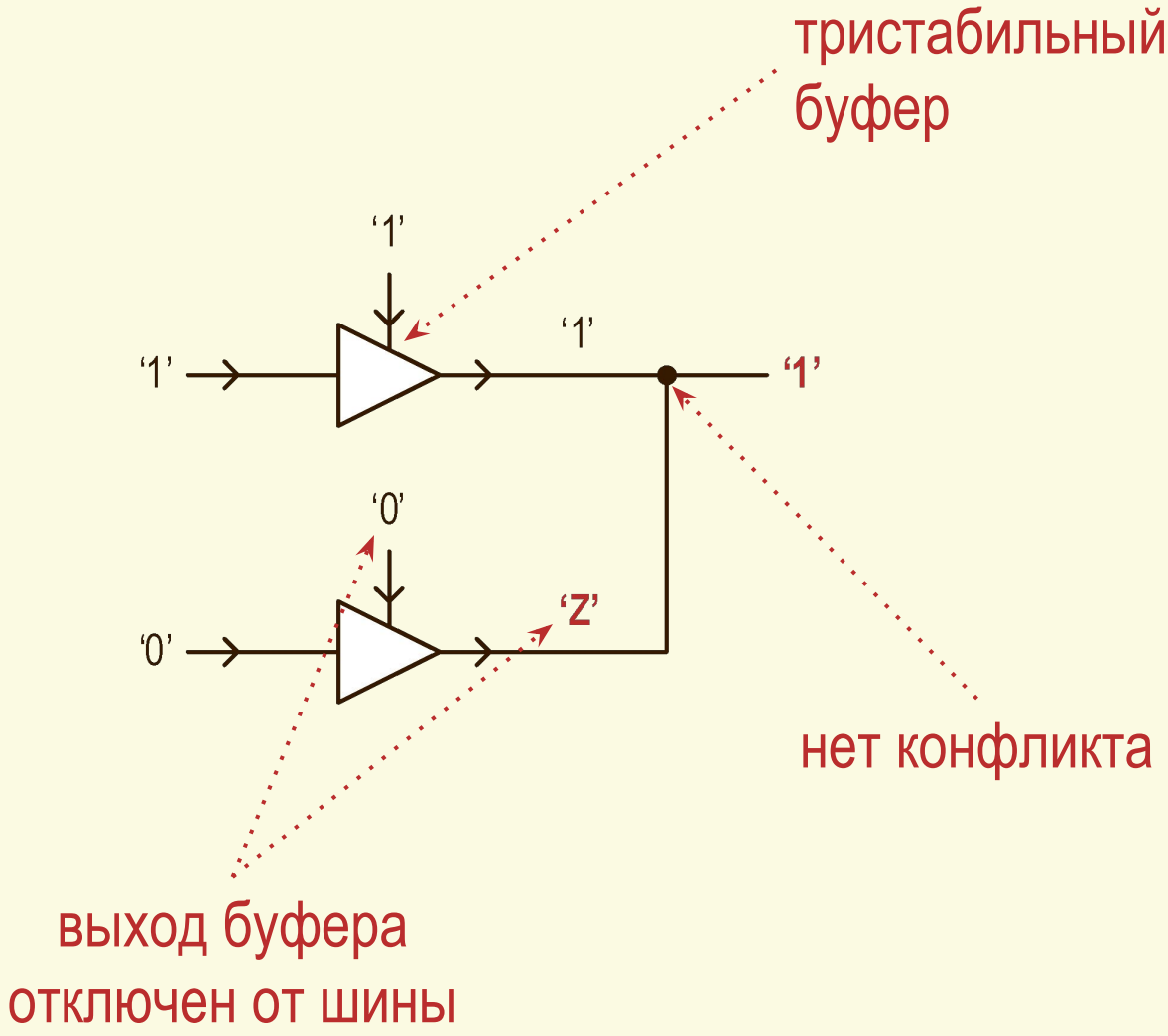
Значения STD_LOGIC

Значение	Пояснения
‘U’	Не инициализированный (значение не присваивалось)
‘X’	Неизвестный (чаще всего в результате конфликта)
‘0’	Низкий логический уровень
‘1’	Высокий логический уровень
‘Z’	Высокоимпедансное состояние («отключен»)
‘W’	Слабый неизвестный
‘L’	Слабый низкий логический уровень
‘H’	Слабый высокий логический уровень
‘-’	Значение не важно

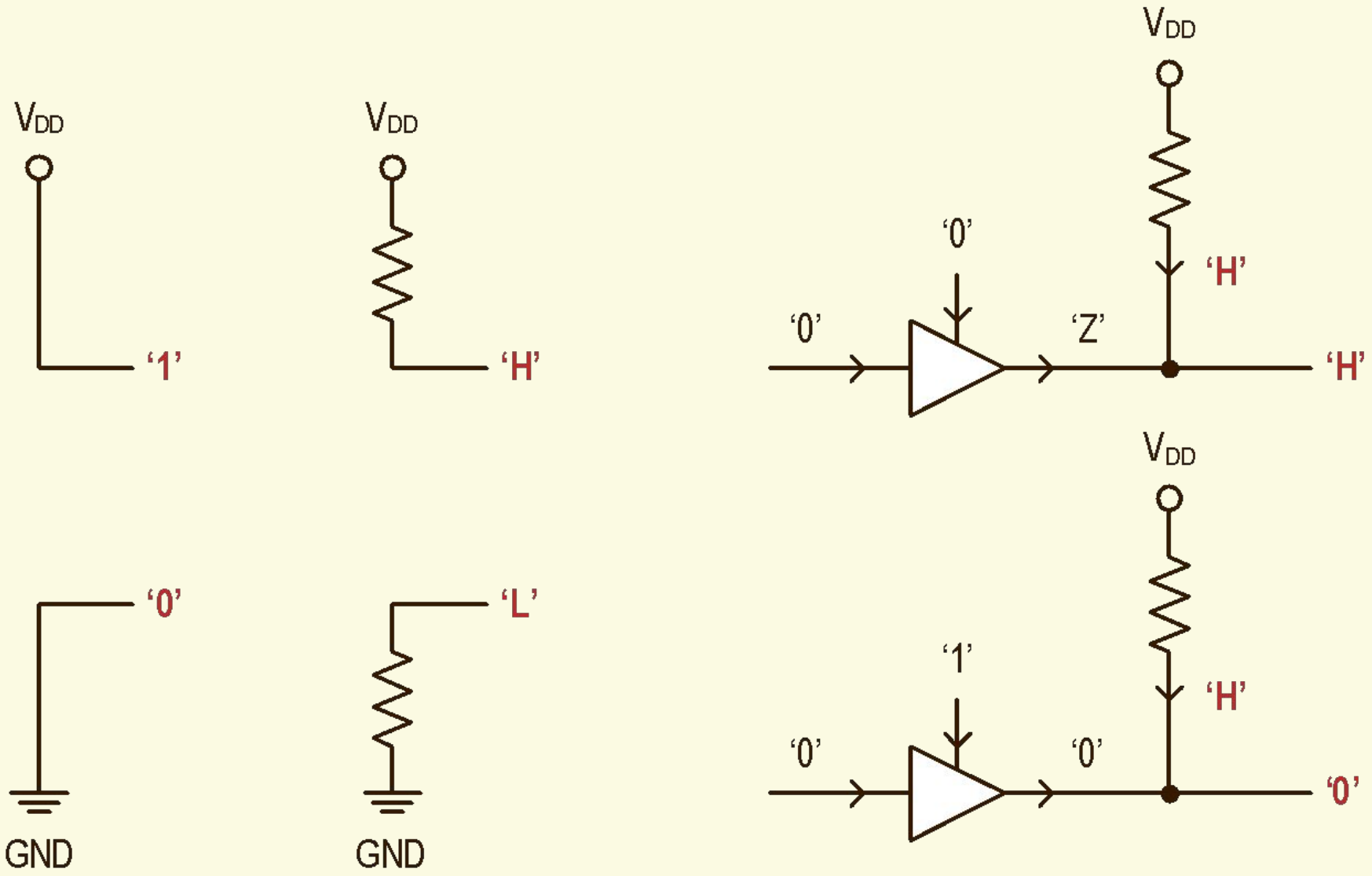
Значения STD_LOGIC – ‘X’



Значения STD_LOGIC – ‘Z’



Значения STD_LOGIC – ‘L’, ‘H’



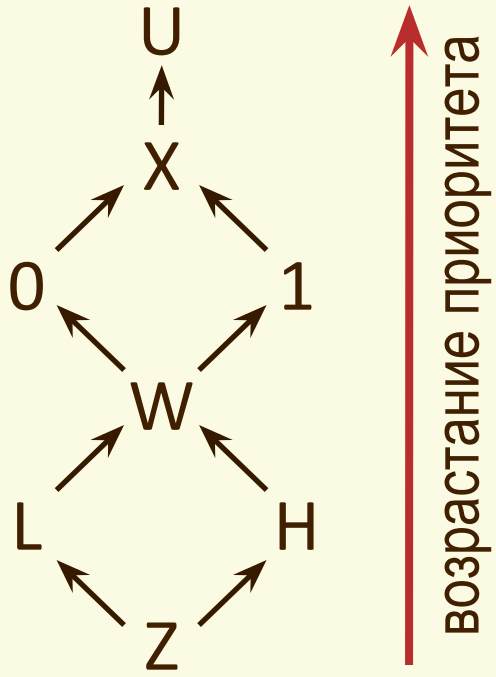
Значения STD_LOGIC – ‘-’

- Применяется, когда значение сигнала не важно
- Может присваиваться выходам (реальные значения – на усмотрение синтезатора) ⟶ оптимизация
- Не используйте в сравнениях

Разрешение конфликтов

- Функция разрешения определяет значение сигнала в случае нескольких драйверов

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X



Использование STD_LOGIC

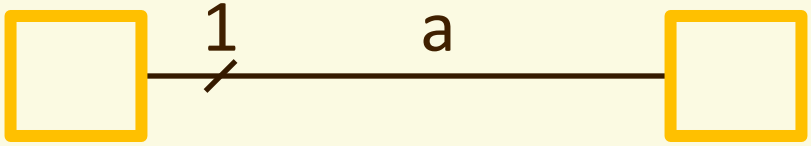
- Всегда используйте STD_LOGIC_VECTOR или STD_LOGIC для **всех портов объекта**
  - Другие типы могут использоваться внутри архитектуры
- Для преобразования типов используйте функции преобразования
  - Также можно оперировать типами STD_LOGIC_VECTOR или STD_LOGIC непосредственно

Одиночные проводники и шины

Скаляры и векторы

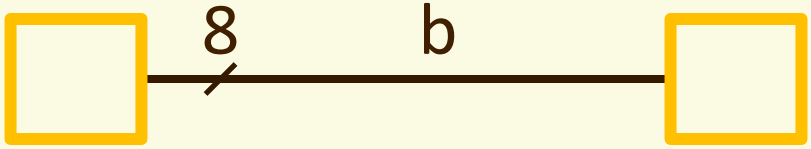
- Одиночный проводник = скаляр

```
signal a:  std_logic;
```



- Шина = вектор

```
signal b:  std_logic_vector(7 downto 0);
```



Использование кавычек

- Скалярные значения указываются в одинарных кавычках

```
signal a, b:   std_logic;

a <= ‘0’;
b <= ‘Z’;
```

- Векторные значения указываются в двойных кавычках

```
signal a, b:   std_logic_vector (1 downto 0);

a <= “00”;
b <= “1Z”;
```

Векторы STD_LOGIC_VECTOR

```
signal a: std_logic_vector(3 downto 0);
signal b: std_logic_vector(3 downto 0);
signal c: std_logic_vector(7 downto 0);
signal d: std_logic_vector(15 downto 0);
signal e: std_logic_vector(8 downto 0);

a <= “0000”;                    -- двоичный формат по умолчанию
b <= B“0000”;                   -- явное указание двоичного формата
c <= “0110_0111”;      -- подчеркивание для улучшения восприятия
кода
d <= X“AF67”;                   -- шестнадцатиричный формат
e <= O“723”;                    -- восьмиричный формат
```

- Старший бит **слева**

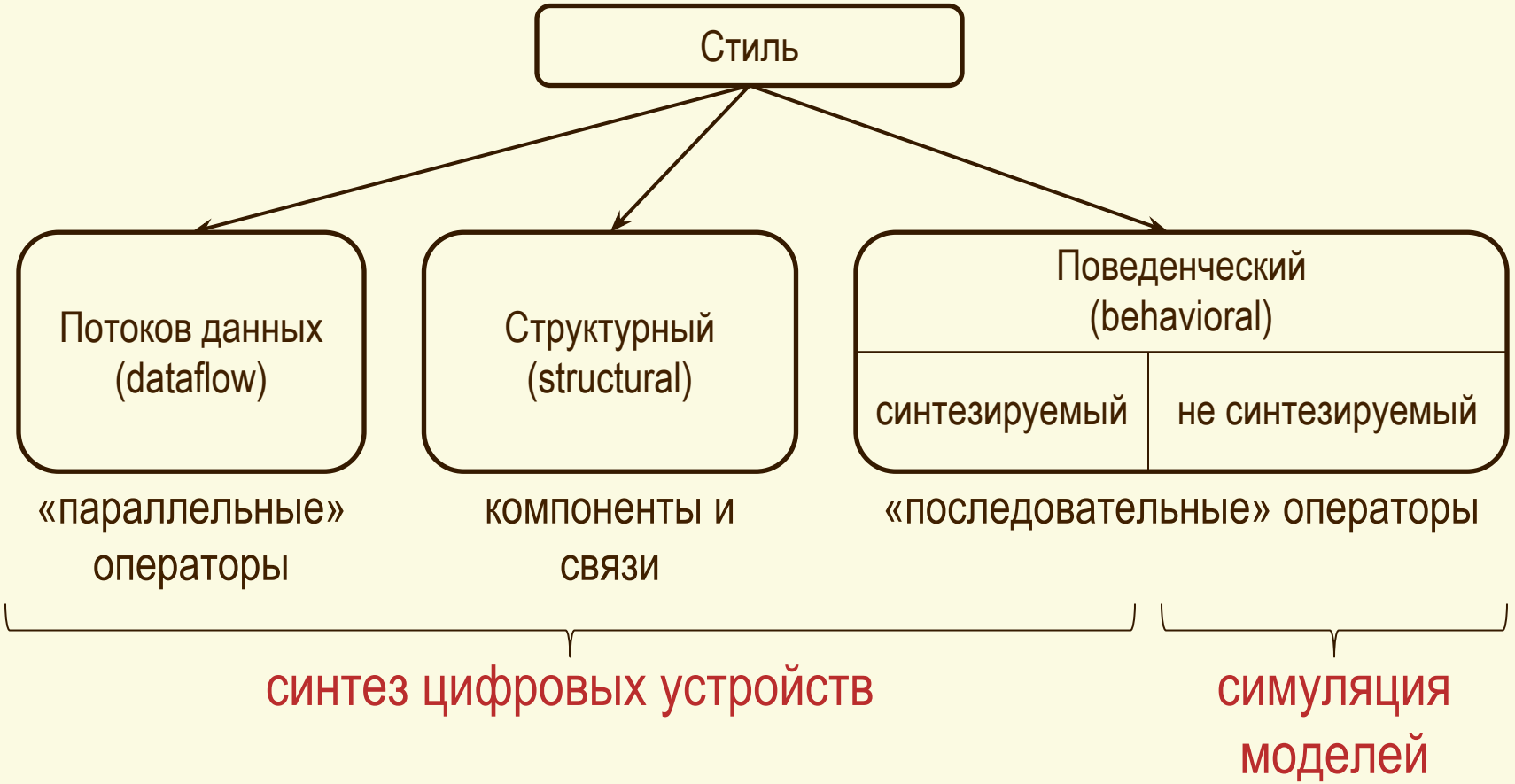
Конкатенация векторов

- Для конкатенации используется оператор **&**
- Для обращения к компоненту вектора указывается **индекс в скобках**
- Для обращения к части вектора указывается **диапазон в скобках**

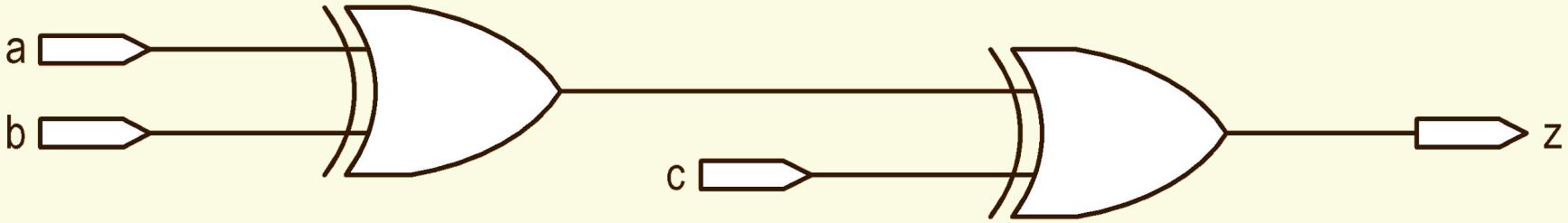
```
signal a, b:        std_logic_vector(3 downto 0);
signal c:           std_logic_vector(7 downto 0);
signal d:           std_logic_vector(5 downto 0);

a <= “1010”;
b <= ‘0’ & ‘0’ & ‘1’ & ‘1’;                    -- b = “0011”
c <= a & b;                                              -- c = “10100011”
d <= a(0) & b(0) & c(3 downto 0);        -- d = “010011”
```

Стили описания на VHDL

Различие стилей



Пример: элемент XOR3



Интерфейс

- Одинаковый для всех стилей

```
library ieee;
use ieee.std_logic_1164.all;

entity xor3 is
	port
	(
		a, b, c:	in std_logic;
		z:			out std_logic
	);
end xor3;
```

Стиль потоков данных

- Описывает перемещения данных по устройству и операции на данными
  - Используется для описания простых булевых функций
  - Все операторы выполняются параллельно → порядок операторов не важен

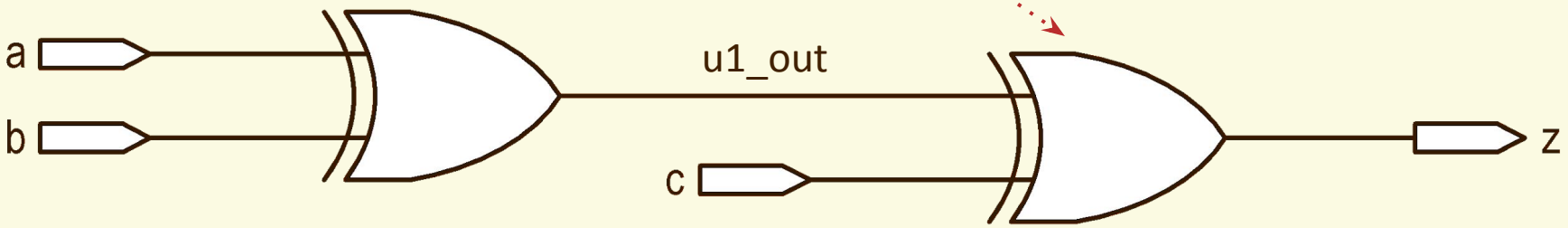
```
u1_out  <= a xor b;
z       <= u1_out
xor c;
```

```
z       <= u1_out
xor c; u1_out<= a xor b;
```

эквивалентны

Стиль потоков данных – пример

```
architecture dataflow of xor3 is
    signal u1_out:    std_logic;          -- промежуточный сигнал
begin
    u1_out  <= a xor b;
    z       <= u1_out xor c;
end dataflow;
```

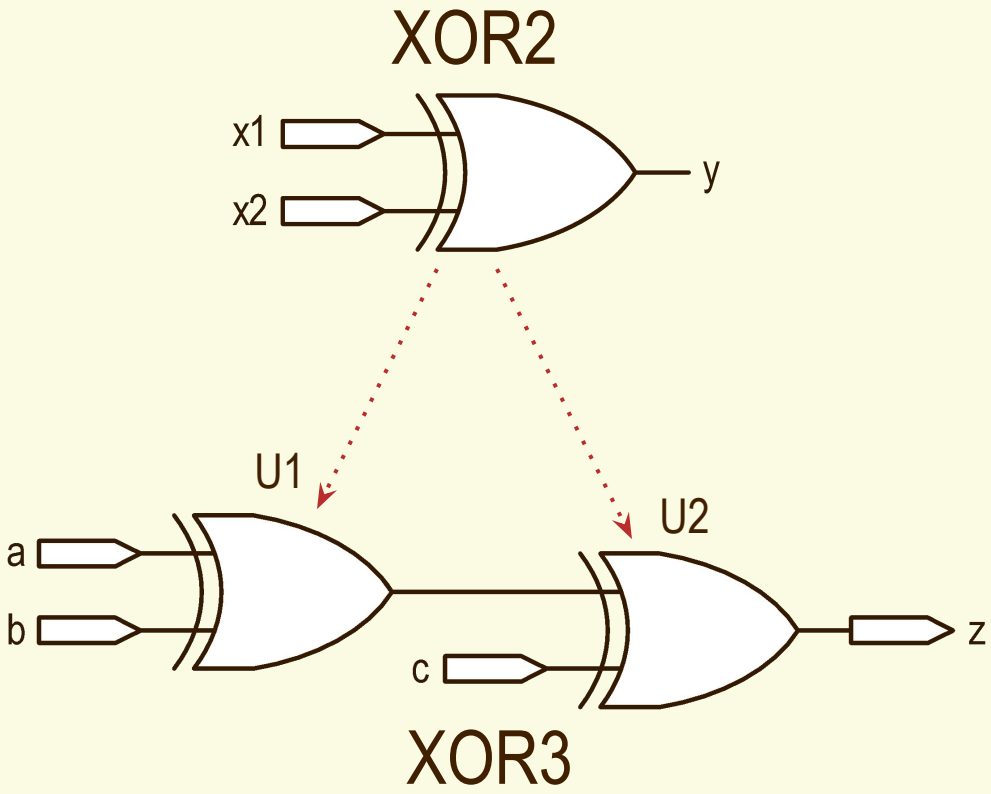


Структурный стиль

- Иерархическое соединение компонентов любой сложности
- Аналог функциональной электрической схемы
- Удобен, когда устройство разбивается на блоки естественным образом
  - Используется при проектировании сложных устройств

Структурный стиль – пример

```
architecture structural of xor3 is
    signal u1_out: std_logic;
    component xor2 is
        port (
            x1, x2:   in
std_logic;
            y:        out
std_logic);
    end component;
begin
    u1:  xor2port map (
                      x1
=> a,
                      x2
=> b,
                      y
=> u1_out);
    u2:  xor2port map (
                      x1
=> u1_out,
                      x2
=> c,
                      y
```



Объявление компонента

- «Черный ящик»
  - Аналог объявления интерфейса объекта
  - Описывает объект, реализованный в библиотеке
  - Позволяет использовать объекты, не имеющие VHDL-реализации (например, на Verilog)

```
component xor2 is
        port
        (
                x1, x2:    in std_logic;
                y:                 out
std_logic
        );
end component;
```

наименование компонента

описание портов

Создание объекта из компонента

- Именованный перечень соединений **(рекомендуется)**
  - Связи указываются явно

- Неименованный перечень соединений
  - Связи зависят от порядка перечисления

```
component xor2 is
      port (
            x1:  in std_logic;
            x2:  in std_logic;
            y:          out std_logic);
end component
```

```
u1:  xor2port map (
                        a,
                        b,
                        u1_out);
```

Поведенческий стиль

- Наиболее далек от особенностей аппаратной реализации
  - Реализация может оказаться неоптимальной
- Описывает алгоритм формирования выходов в зависимости от входов
- Для описания используются блоки PROCESS
  - Операторы внутри блока выполняются последовательно → порядок операторов имеет значение

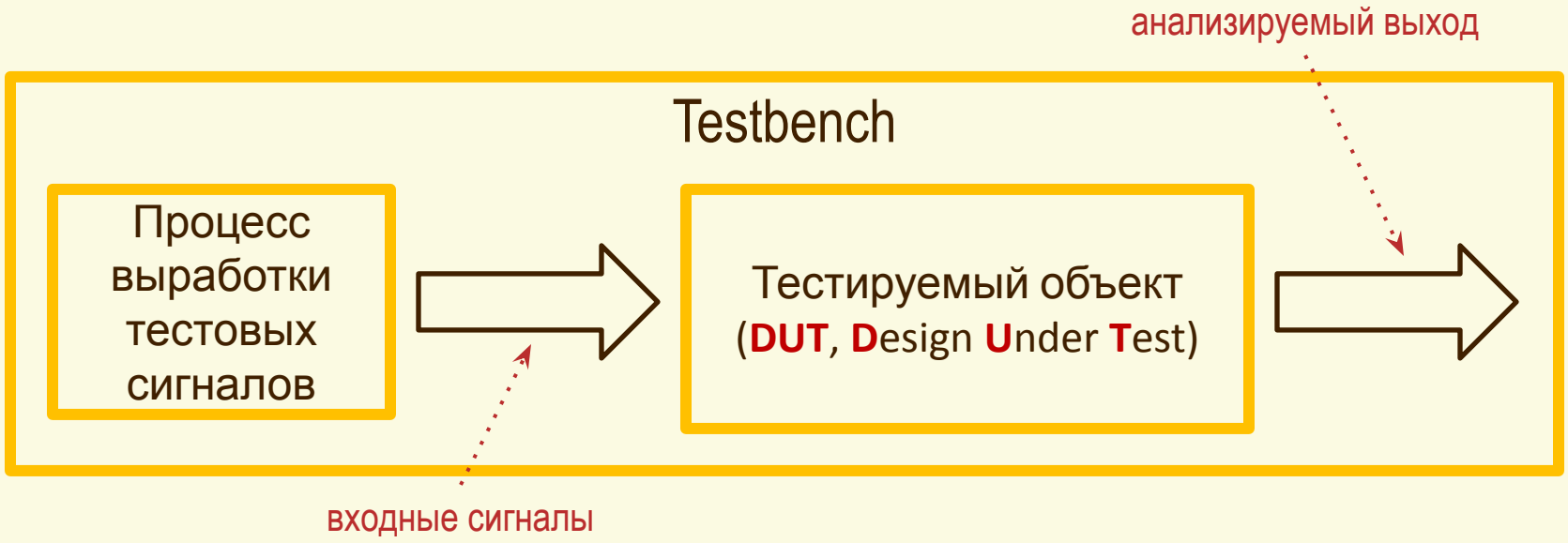
Поведенческий стиль – пример

```
architecture behavioral of xor3 is
begin
        process (a, b, c)
        begin
                if (a xor b xor c = '1') then
                        z <= '1';
                else
                        z <= '0';
                end if;
        end process;
end behavioral ;
```

Testbenches

Testbench

- Предназначен для автоматизированного тестирования объектов
- Подает значения на входы DUT
  - Может также анализировать значения на выходе



Testbench

- Составляется на VHDL
  - Не требует изучения специальных языков
  - Не привязан к конкретной среде
- Является объектом моделирования
  - **Не синтезируемый**
  - Не имеет портов
  - Выполняется в среде симулятора
- Рассматривает DUT как черный ящик
  - Легко адаптируется к различным архитектурам
- Выходы DUT отображаются в виде временных диаграмм или записываются в файл

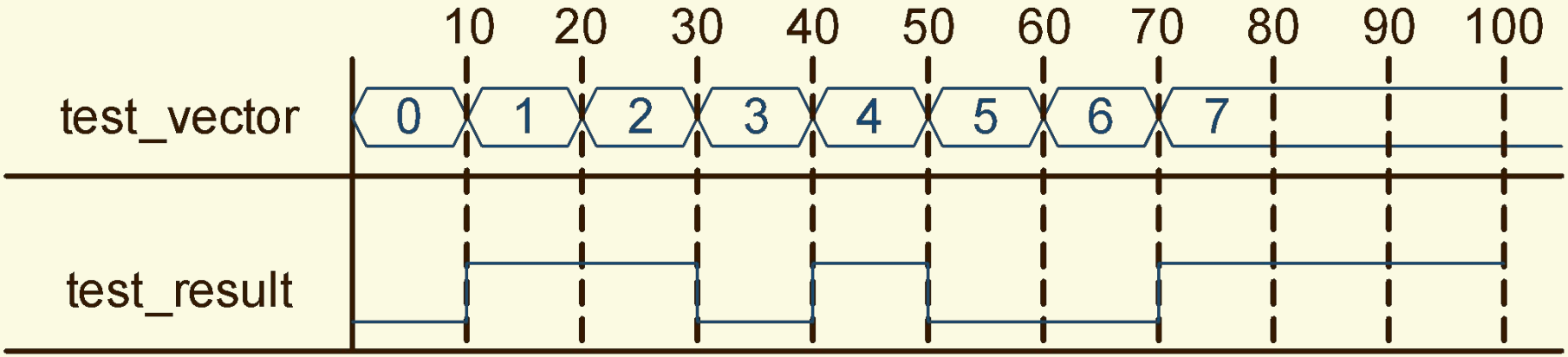
Testbench – пример для XOR3

```
library ieee;
use ieee.std_logic_1164.all;

entity xor3_tb is
end xor3;                                                    -- нет
портов

architecture testbench of xor3_tb is
        component xor3 is                                    --
тестируемый объект (DUT)
                port (
                        a, b, c:        in std_logic;
                        z:              out std_logic);
        end component;
        signal test_vector:     std_logic_vector(2 downto 0);   -- входные значения
        signal test_result:     std_logic;                      -- выходное
значение

begin
        dut: xor3 port map (                                    -- создание
объекта
                                a => test_vector(0),
                                b => test_vector(1),
                                c => test_vector(2),
                                z => test_result);
```

Testbench – пример для XOR3 (продолжение)

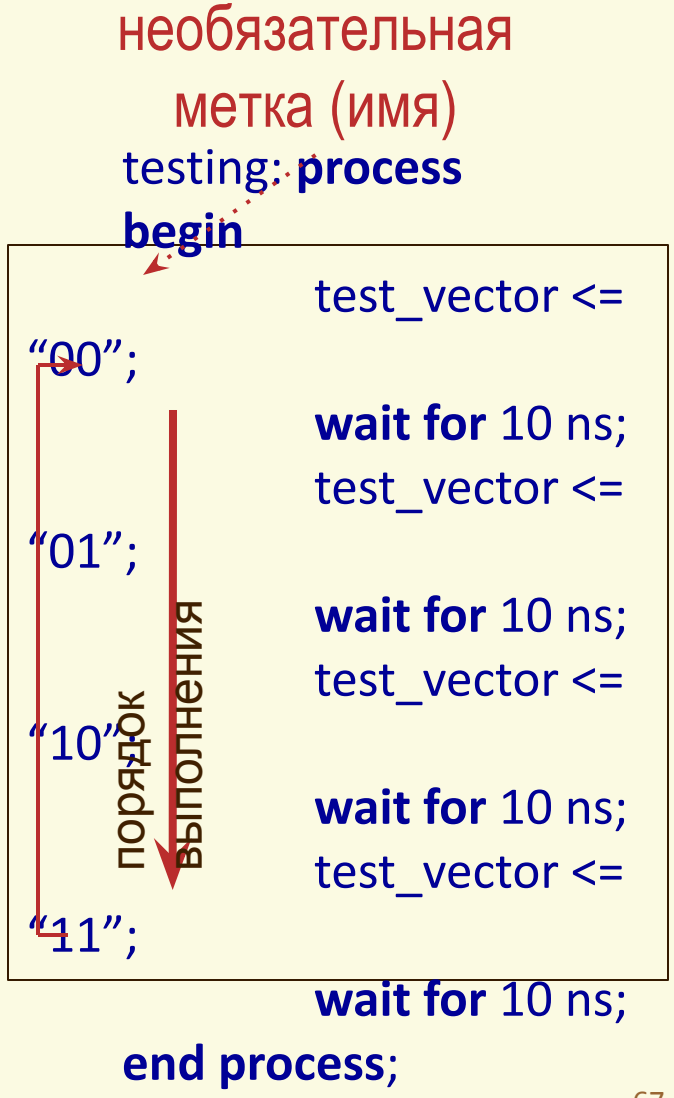
```
        process
        begin
                test_vector <= “000”;
                wait for 10 ns;                          -- задержка 10 нс
                test_vector <= “001”;
                wait for 10 ns;
                test_vector <= “010”;
                wait for 10 ns;
                test_vector <= “011”;
                wait for 10 ns;
                test_vector <= “100”;
                wait for 10 ns;
                test_vector <= “101”;
                wait for 10 ns;
                test_vector <= “110”;
                wait for 10 ns;
                test_vector <= “111”;
                wait;                                    -- остановка
        end process;
end testbench;
```

Testbench – пример для XOR3 (врем. диаграммы)



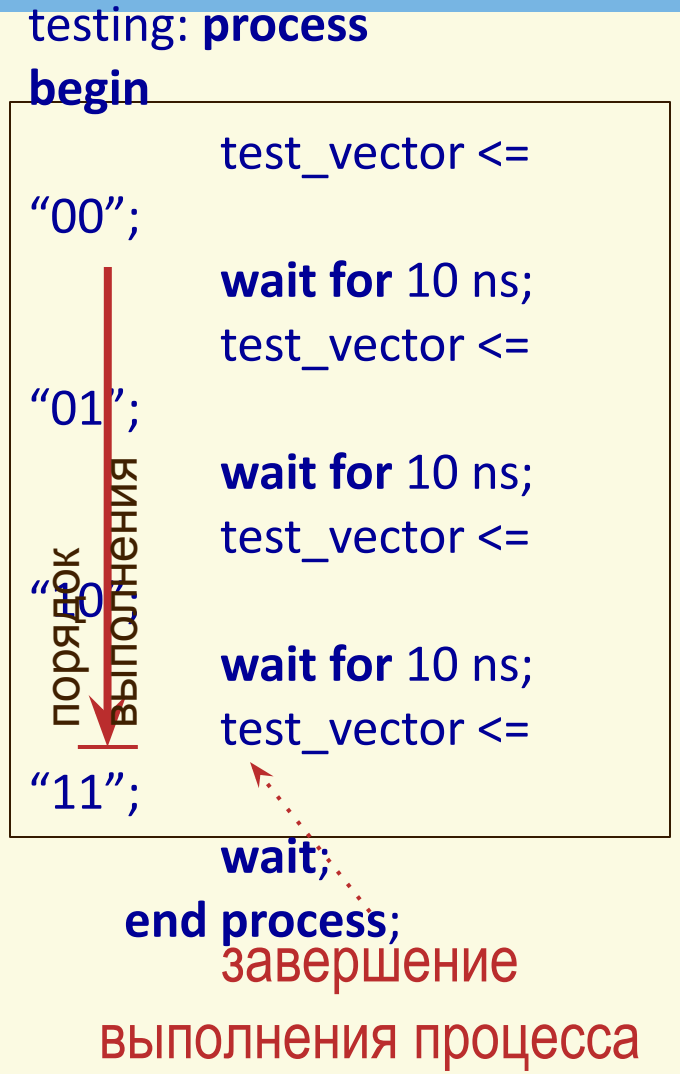
Процессы

- Процесс – набор инструкций
  - Размещаются между словами BEGIN и END PROCESS
  - Выполняются по очереди
  - После последней инструкции осуществляется возврат к первой



Процесс и оператор WAIT

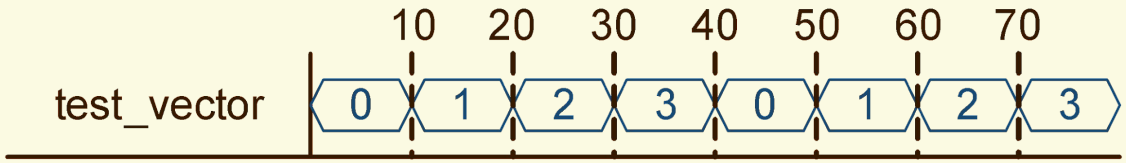
- Используется для приостановки процесса
- **wait for** time
  - Приостановка на заданное время (пауза)
  - Используется для моделирования задержки
- **wait until** condition
  - Приостановка до выполнения условия
- **wait on** signals
  - Приостановка до изменения любого из сигналов
- **wait**
  - Окончательная остановка
  - Используется в testbench после генерации всех сигналов



WAIT FOR vs. WAIT

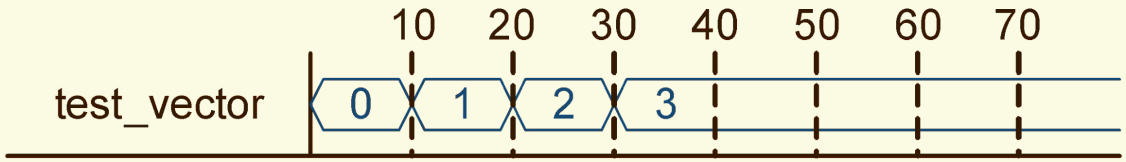
```
testing: process
begin
        test_vector <= “00”;
        wait for 10 ns;
        test_vector <= “01”;
        wait for 10 ns;
        test_vector <= “10”;
        wait for 10 ns;
        test_vector <= “11”;
        wait for 10 ns;
end process;
```

WAIT FOR: значения повторяются циклически




```
testing: process
begin
        test_vector <= “00”;
        wait for 10 ns;
        test_vector <= “01”;
        wait for 10 ns;
        test_vector <= “10”;
        wait for 10 ns;
        test_vector <= “11”;
        wait;
end process;
```

WAIT: значение остается постоянным после инструкции wait



Цикл FOR (кратко)

- Циклически повторяет набор инструкций

```
for i in range loop
    statements
end loop;
```

- Пример:

```
process
begin
    test_vector <= “000”;
    for i in 0 to 6 loop
        wait for 10 ns;
        test_vector <= test_vector +
“001”;
    end loop;
    wait for 10 ns;
end process;
```


```
process
begin
        test_vector <= “000”;
        wait for 10 ns;
        test_vector <= “001”;
        wait for 10 ns;
        test_vector <= “010”;
        wait for 10 ns;
        test_vector <= “011”;
        wait for 10 ns;
        test_vector <= “100”;
        wait for 10 ns;
        test_vector <= “101”;
        wait for 10 ns;
        test_vector <= “110”;
        wait for 10 ns;
        test_vector <= “111”;
        wait for 10 ns;
end process;
```