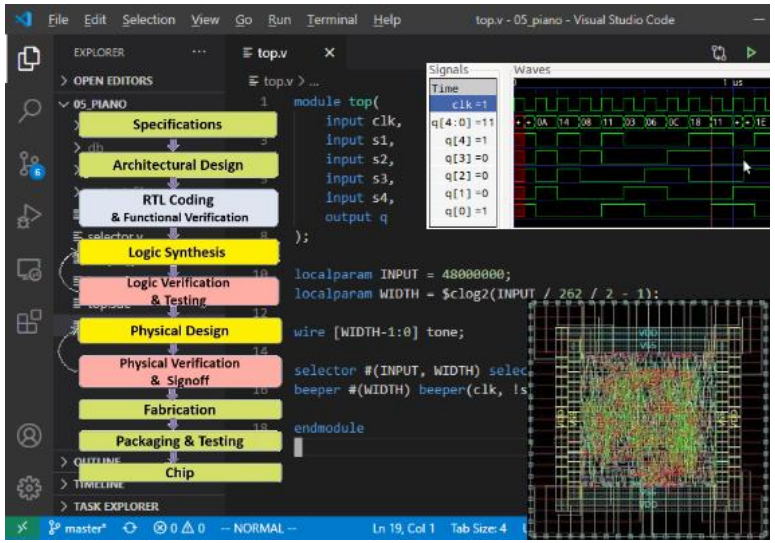
МИЭТ
Национальный исследовательский университет «МИЭТ»
Институт интегральной электроники

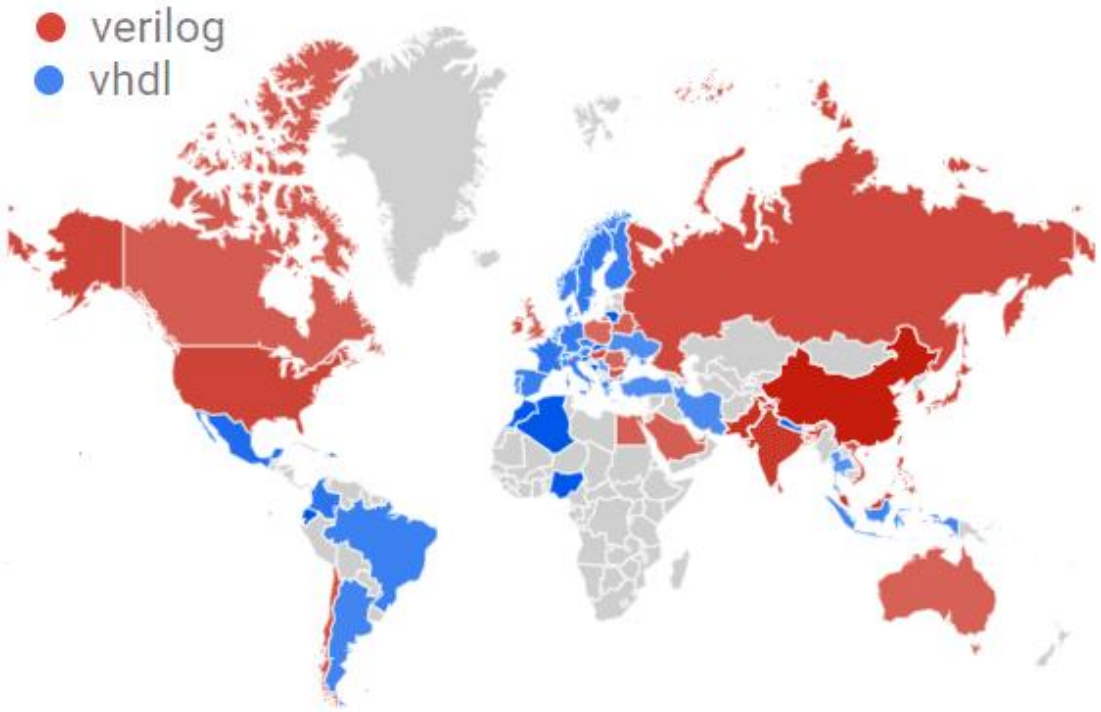
Лингвистические средства проектирования

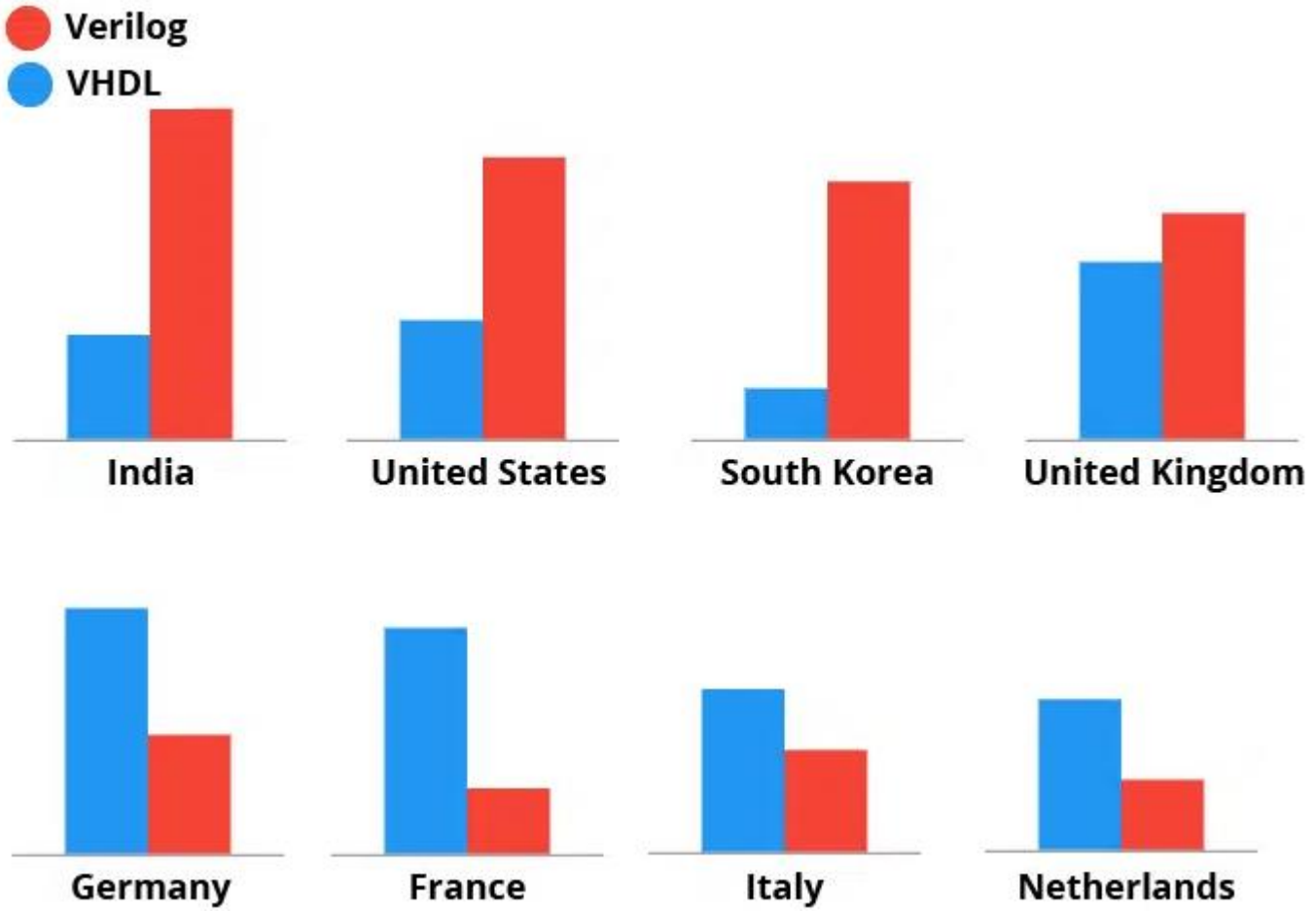


Лекция 6

Языки описания аппаратуры. Язык VHDL

VHDL - язык разработки в Европе

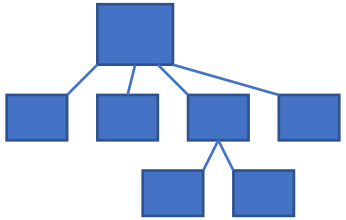


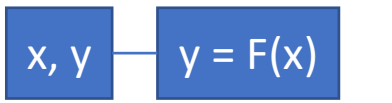


* https://vhdlwhiz.com/should-i-learn-vhdl-if-verilog-is-becoming-more-popular/

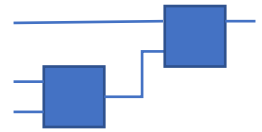
Черты языка VHDL

- Поддержка иерархии – проектируемые устройства разбиваются на иерархические блоки.
- Описание элемента разделено на интерфейс и архитектурное тело.
- Один объект проектирования может иметь одновременно несколько архитектурных тел.
- Функционирование может быть задано либо с помощью алгоритма, либо с помощью логических функций, либо с помощью указания перечня компонентов и их связей между ними.
- Возможность моделирования параллельно протекающих процессов.
- Возможность проведения временного и функционального моделирования.
- Поддержка пользовательских типов данных.

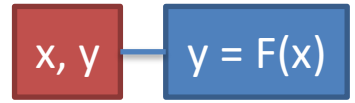


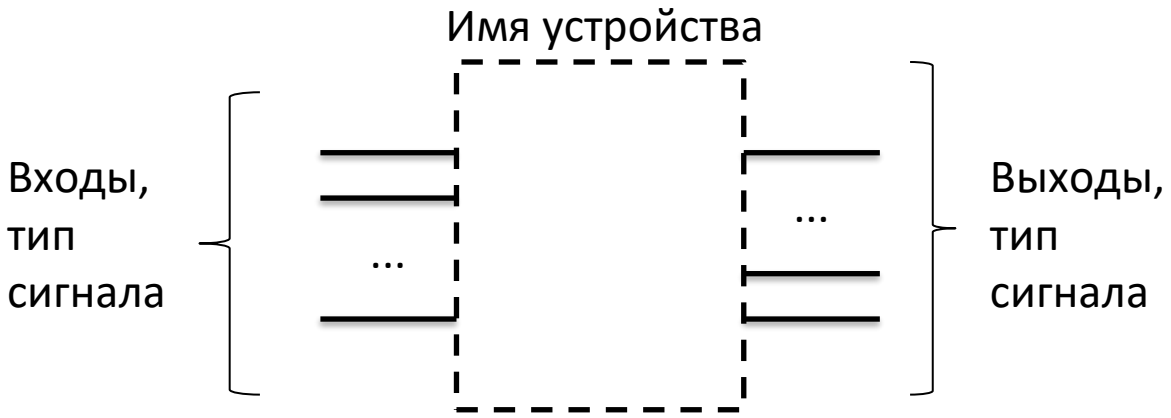



```
if(X = 1) then
  Y <= '0';
else
  Y <= '1';
end if;
```



Описание схем на языке VHDL: интерфейс



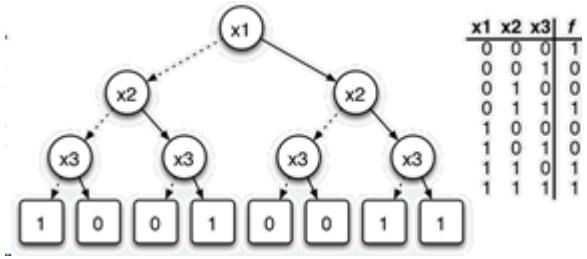


Имя устройства

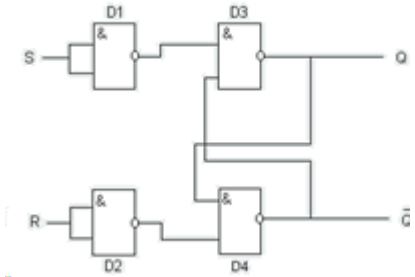
```
entity inv is
  port(x: in  bit;
       y: out bit);
end inv;
```

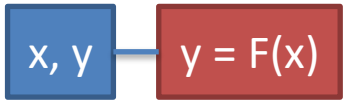
Порты устройства, с указанием направлений и передаваемых типов данных

Описание схем на языке VHDL: архитектурное тело



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1





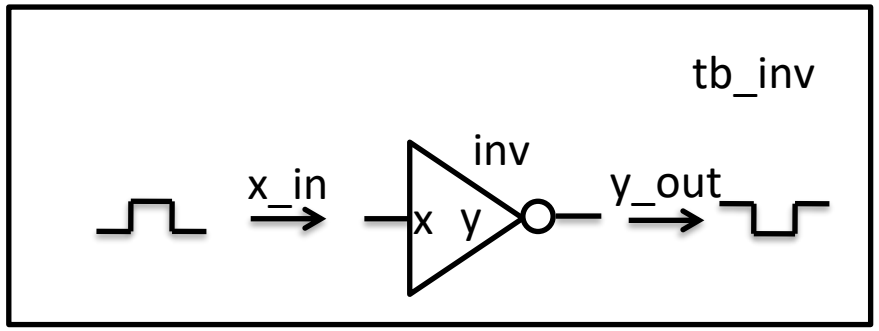
Имя архитектурного тела

Функционирование какого из элементов описывает

```
architecture RTL of inv is
begin
  y <= not x;
end RTL;
```

Как функционирует

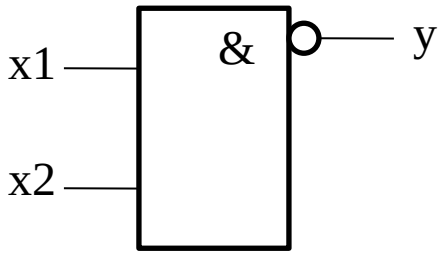
Написание тестового окружения




```
entity tb_inv is
end tb_inv;

architecture test of tb_inv is

  component inv
    port(x: in bit; y: out bit);
  end component;

  signal x_in, y_out: bit;

begin
  p1 : inv port map(x_in, y_out);

  p2 : x_in <= '0' after 0 ns, '1' after 10 ns, '0' after 20 ns;
end test;
```

Описание портов устройства: VHDL

```
entity <имя_интерфейса> is

  [ generic(<список_параметров>); ]
  [ port(<список_портов>); ]

[ begin

  <типы данных,
   значения констант>;

]

end [entity] [<имя_интерфейса>];
```

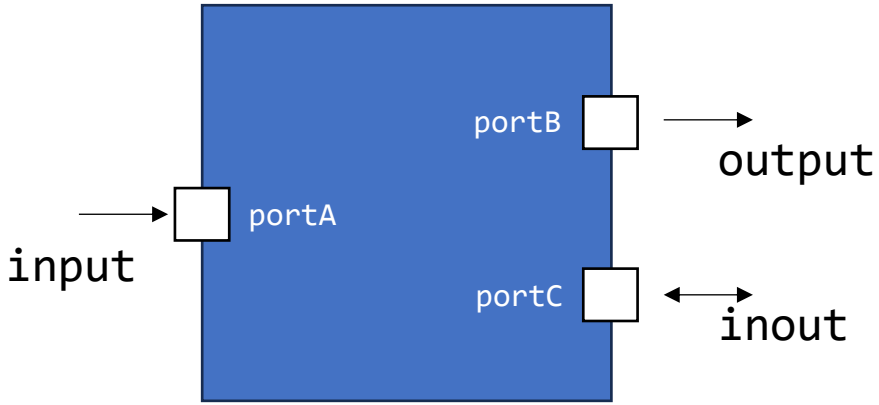



```
entity NAND2 is
  port(x1: in  bit;   -- первый вход
       x2: in  bit;   -- второй вход
        y: out bit    -- выход
  );
end NAND2;
```

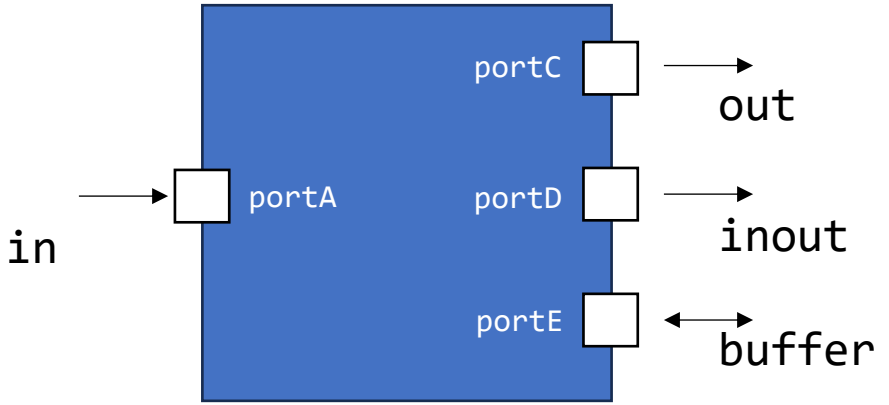
```
entity NAND2 is
  port(x1, x2: in  bit;
               y: out bit);
end NAND2;
```

Направления портов

Verilog



VHDL



Типы данных портов и сигналов

Verilog

```
logic : {
  0,  -- логический 0
  1,  -- логическая 1
  X,  -- неизвестное состояние
  Z,  -- высокий импеданс
}
```

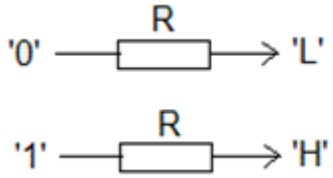
VHDL

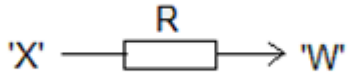
Тип данных bit

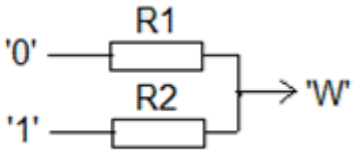
```
BIT : {'0', '1'};
```

Тип данных std_logic

```
STD_LOGIC : {
  'U',  -- неинициализированное значение
  'X',  -- неизвестное значение
  '0',  -- логический 0
  '1',  -- логическая 1
  'Z',  -- высокий импеданс
  'W',  -- слабый сигнал, непонятно, он 0 или 1
  'L',  -- слабый сигнал, подтянутый к 0
  'H',  -- слабый сигнал, подтянутый к 1
  '-'   -- безразличное состояние
}
```







Тип std_logic требует подключения библиотеки std_logic_1164

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

Описание схем с разными типами данных

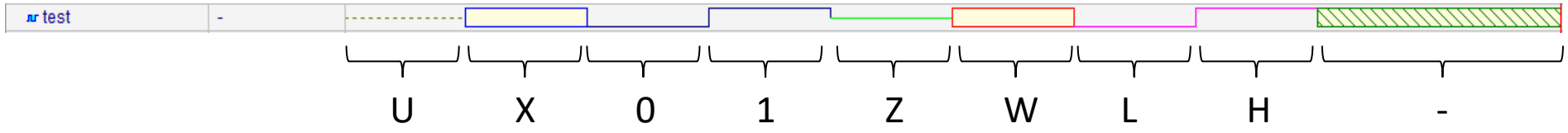
```
entity and2 is
  port(x1, x2: in  bit;
               y: out bit);
end and2;

architecture RTL of and2 is
begin
  y <= x1 and x2;
end RTL;
```

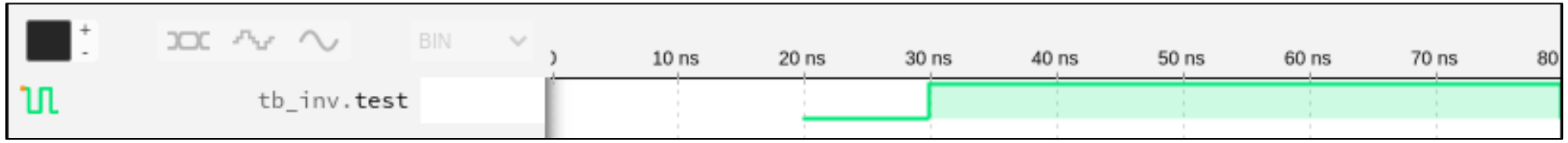
```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
  port(x1, x2: in  std_logic;
               y: out std_logic);
end and2;

architecture RTL of and2 is
begin
  y <= x1 and x2;
end RTL;
```

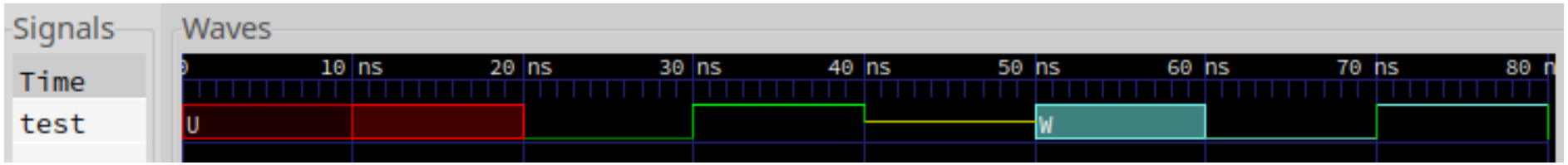
Отображение значений сигналов в постпроцессорах

```
test <= 'X' after 10 ns,
        '0' after 20 ns,
        '1' after 30 ns,
        'Z' after 40 ns,
        'W' after 50 ns,
        'L' after 60 ns,
        'H' after 70 ns,
        '-' after 80 ns;
```

```
test <='U' after 0 ns,
       'X' after 10 ns,
       '0' after 20 ns,
       '1' after 30 ns,
       'Z' after 40 ns,
       'W' after 50 ns,
       'L' after 60 ns,
       'H' after 70 ns,
       '-' after 80 ns;
```



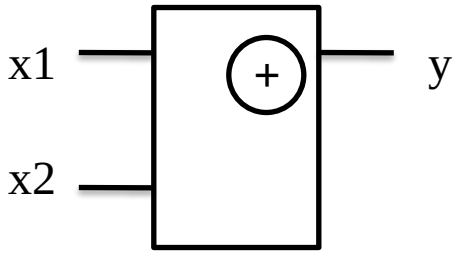


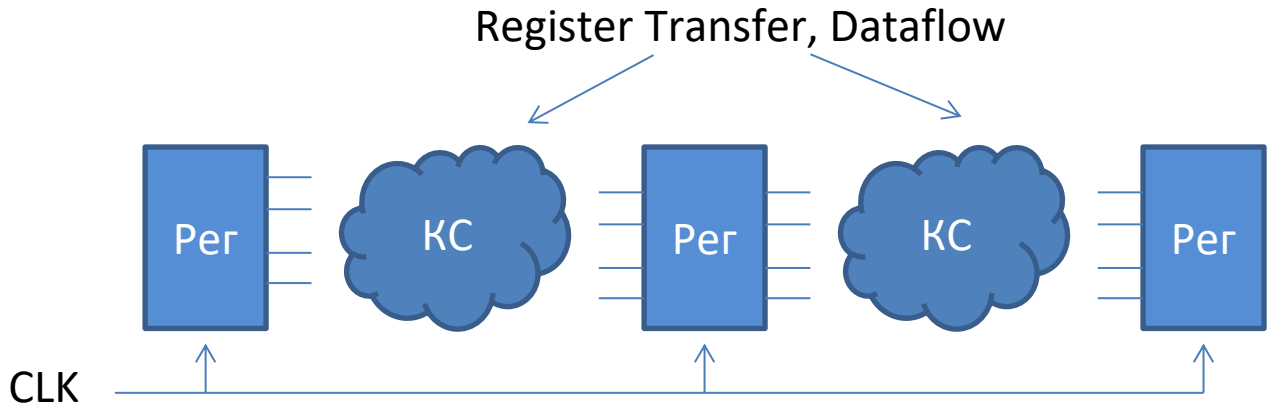


Синтаксис описания архитектуры

```
architecture <имя_архитектруры> of <имя_интерфейса> is
  [<список_объявлений>;]
begin
  <список_параллельных_операторов>;
end [architecture] [<имя_архитектуры>];
```

Описание архитектурного тела типа RTL





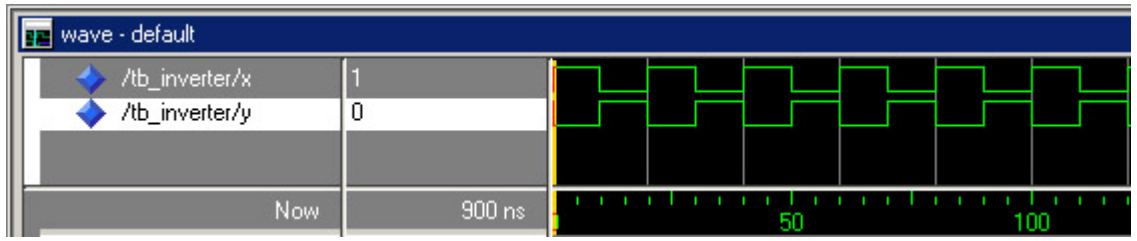

```
entity xor2 is
  port(x1, x2: in  std_logic;
               y: out std_logic);
end xor2;

architecture RTL of xor2 is
begin
  y <= x1 xor x2;
end RTL;
```

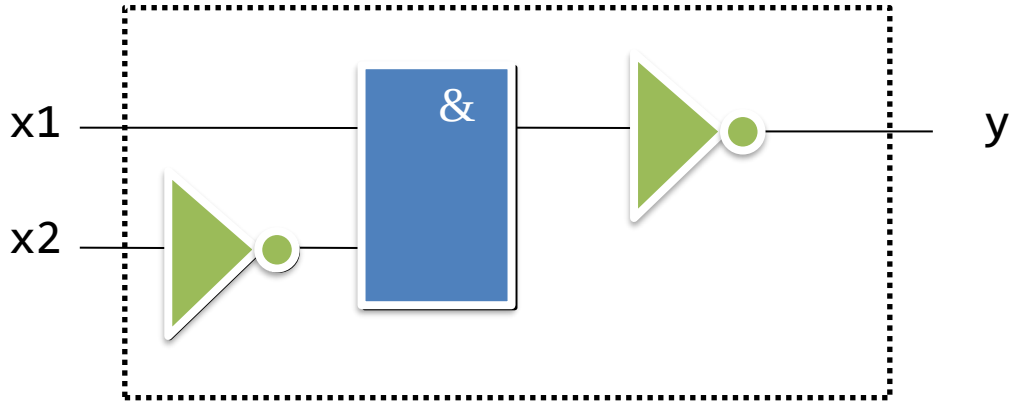
```
architecture RTL of xor2 is
begin
  y <= (x1 and (not x2)) or (not x1 and x2);
end RTL;
```

Этап 1. Проектирование библиотеки элементов

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

entity inv is
  port(x: in  std_logic;
       y: out std_logic);
end inv;

architecture RTL of inv is
begin
  y <= not x;
end RTL;
```

\+ тестовое окружение

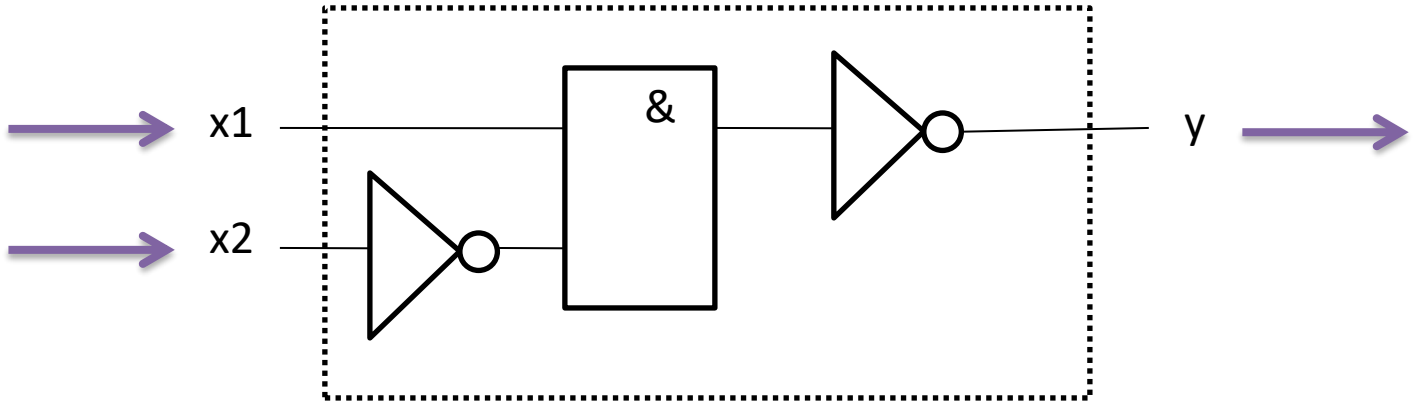


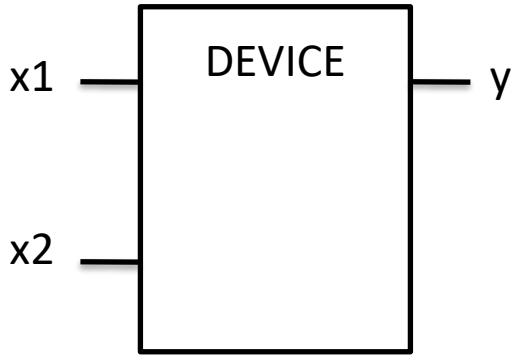



```
entity and2 is
  port(x1, x2: in  std_logic;
               y: out std_logic);
end and2;

architecture RTL of and2 is
begin
  y <= x1 and x2;
end RTL;
```

\+ тестовое окружение

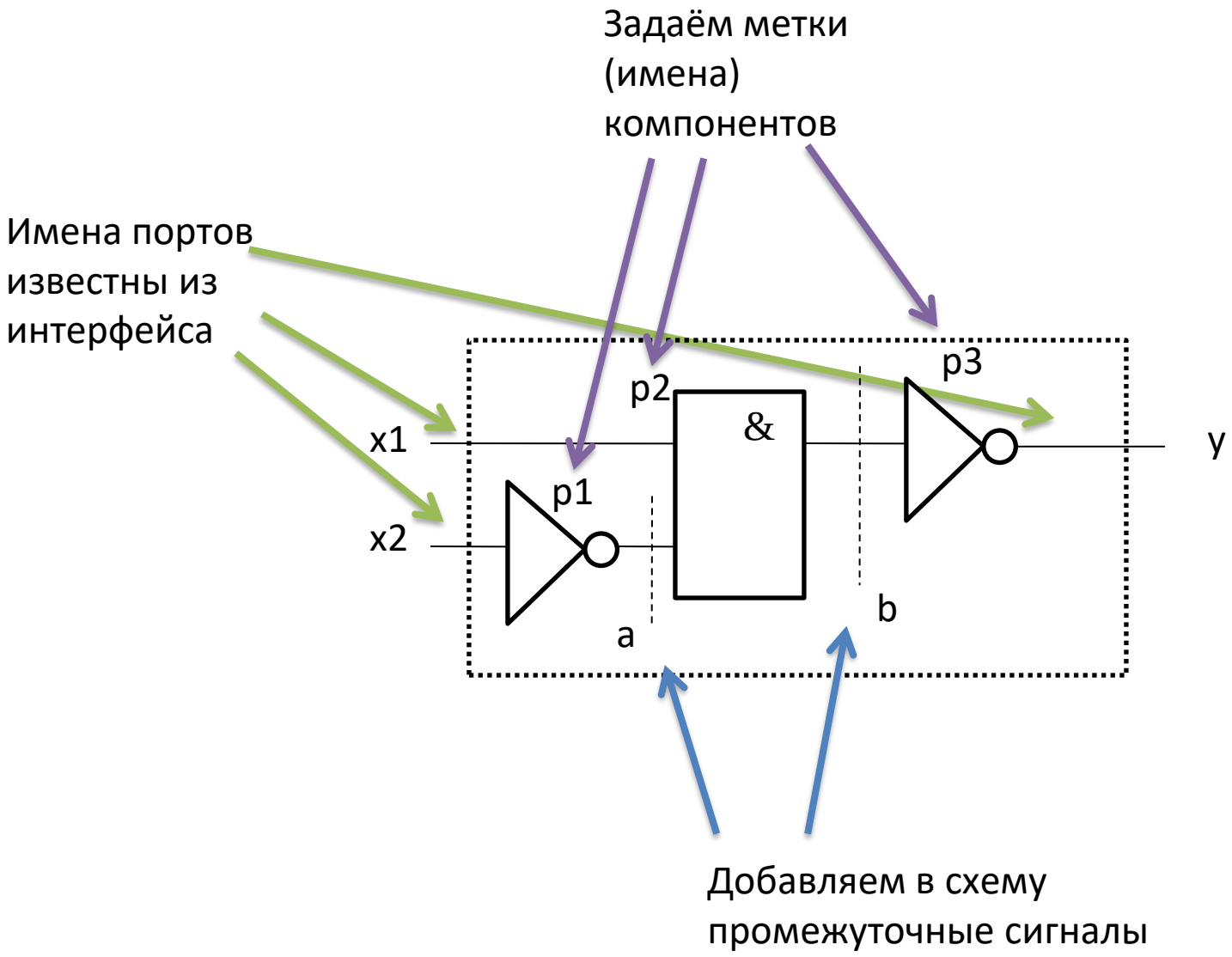
Этап 2. Описание интерфейса устройства






```
entity DEVICE is
  port(x1: in  std_logic;
       x2: in  std_logic;
        y: out std_logic);
end DEVICE;
```

Этап 3-а. Расстановка недостающих объектов



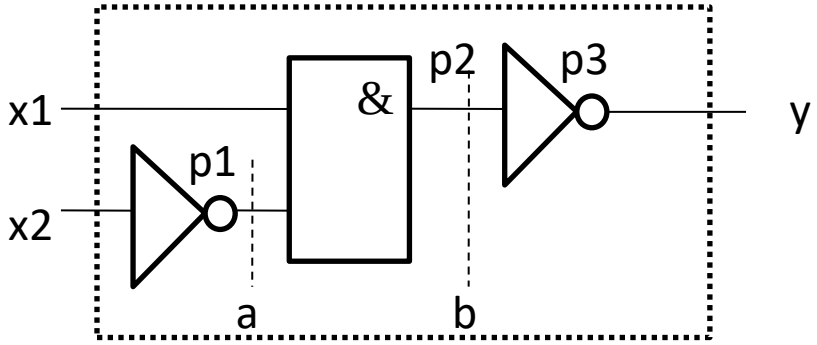
Этап 3-б. Описание взаимосвязи компонентов

```
architecture STR of DEVICE is
  component inv
    port(x: in  std_logic;
         y: out std_logic);
  end component;
  component and2
    port(x1, x2: in  std_logic;
                 y: out std_logic);
  end component;

  signal a, b: std_logic;

begin

  p1 : inv port map(x2, a);
  p2 : and2 port map(x1, a, b);
  p3 : inv port map(b, y);
end STR;
```

Какие из описанных ранее элементов будут использованы?

Перечень недостающих сигналов

Связь элементов посредством сигналов и портов



Позиционное и ассоциативное назначение портов

Язык Verilog

```
inv  i1(x2, a);
and2 i2(x1, a, b);
inv  i3(b, y);
```

```
inv  i1(.x(x2), .y(a));

and2 i2(.x1(x1), .x2(a), .y(b));

inv  i3(.x(b), .y(y));
```

Язык VHDL

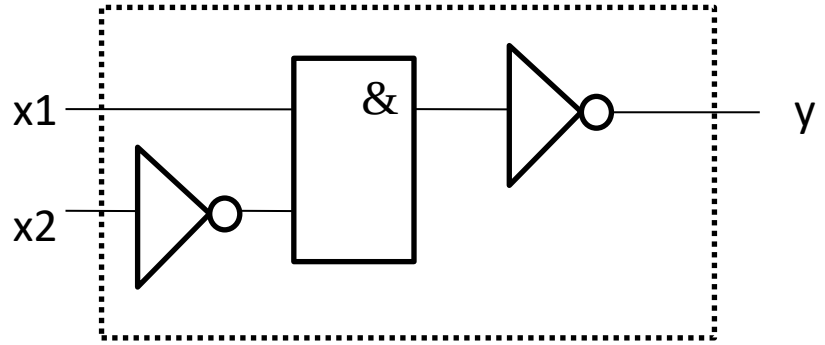
```
i1 : inv  port map(x2, a);
i2 : and2 port map(x1, a, b);
i3 : inv  port map(b, y);
```

```
i1 : inv port map(x=>x2, y=>a);

i2 : and2 port map(x1=>x1, x2=>a, y=>b);

i3 : inv port map(y=>y, x=>b);
```

Поведенческое описание схемы на VHDL

x1	x2	y
0	0	1
0	1	1
1	0	0
1	1	1

```
entity DEVICE is
  port(x1, x2: in  std_logic;
               y: out std_logic);
end DEVICE;

architecture BEH of DEVICE is
begin

  process(x1, x2)
  begin
    if (x1='1' and x2='0') then
      y <= '0';
    else
      y <= '1';
    end if;
  end process;

end BEH;
```



Требования к описанию процессов/процедурных блоков

В процессе обязана быть одна из двух конструкций:

1. список чувствительности;

   Без списка чувствительности процесс работает **бесконечно** переходя от одной итерации к другой.

2. оператор ожидания.

Если процесс не содержит ни списка чувствительности, ни оператора ожидания, он **входит в зацикливание, моделирование становится невозможным**.

Формы записи оператора wait

`wait` – оператор ожидания.

4 формы записи оператора:

1. ожидание в течение определённого времени;

```
wait for <время>;
```

2. ожидание изменения сигнала;

```
wait on <список_сигналов>;
```

3. ожидание выполнения логического условия,

```
wait until <логическое_условие>;
```

4. остановка процесса.

```
wait;
```

Формы записи оператора wait: wait for


```
p1: x <= '1' after 0 ns, '0' after 20 ns, '1' after 40 ns, ...
```

```
p1 : process
     begin
       x <= '1';
       wait for 20 ns;
       x <= '0';
       wait for 20 ns;
     end process;
```

```
p1 : process
     begin
       x <= not x;
       wait for 20 ns;
     end process;
```

Аналог в языке Verilog:

```
always
  #20 x = ~x;
```

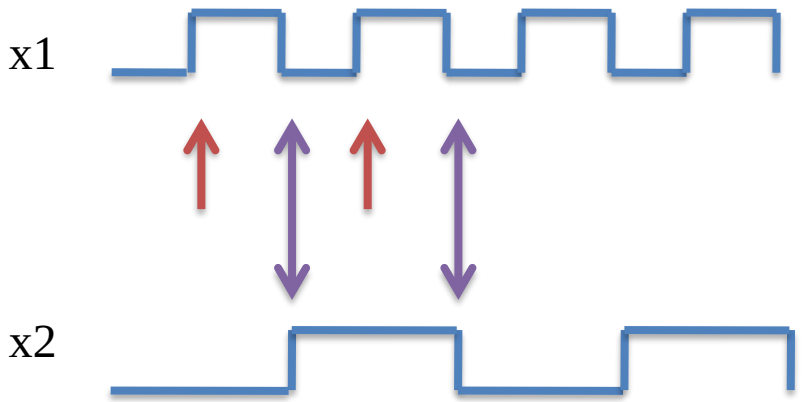
Пример на wait on

AND2

x2	x1	y
0	0	0
0	1	0
1	0	0
1	1	1

```
p1 : process
     begin
       x1 <= not x1;
       wait for 20 ns;
     end process;

p2 : process
     begin
       x2 <= not x2;
       wait for 40 ns;
     end process;
```

```
always
  #20 x = ~x;

always
  #40 x = ~x;
```



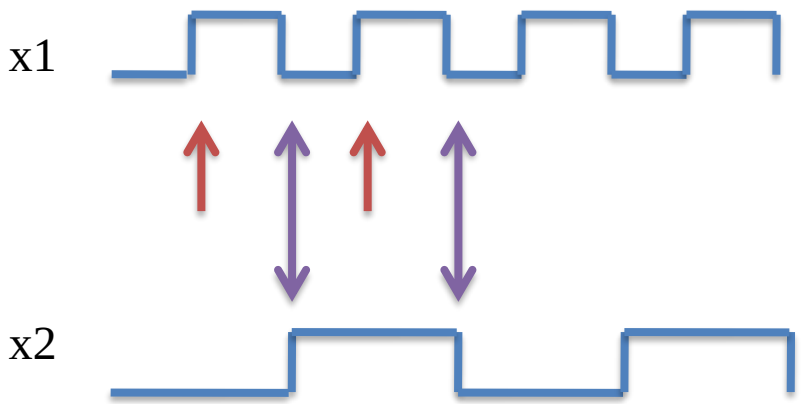

```
p1 : process
     begin
       x1 <= not x1;
       wait for 20 ns;
     end process;

p2 : process
     begin
       wait on x1;
       wait on x1;
       x2 <= not x2;
     end process;
```

Пример на wait until

AND2

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

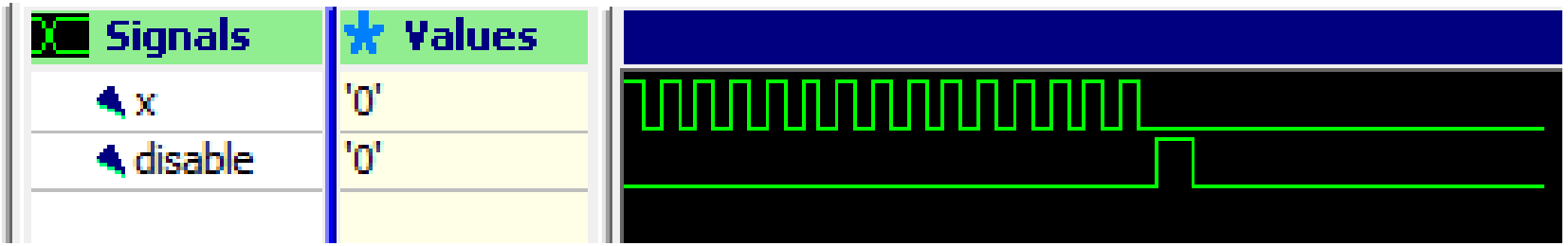
```
p1 : process
     begin
       x1 <= not x1;
       wait for 20 ns;
     end process;

p2 : process
     begin
       x2 <= not x2;
       wait for 40 ns;
     end process;
```




```
p1 : process
     begin
       x1 <= not x1;
       wait for 20 ns;
     end process;

p2 : process
     begin
       wait until (x1 = '0');
       x2 <= not x2;
     end process;
```

Остановка процесса: пример на wait;

```
p1 : disable <= '0', '1' after 150 ns, '0' after 160 ns;

p2 : process
     begin
       x <= not x;
       wait for 5 ns;
       if (disable = '1') then
         wait;
       end if;
     end process;
```

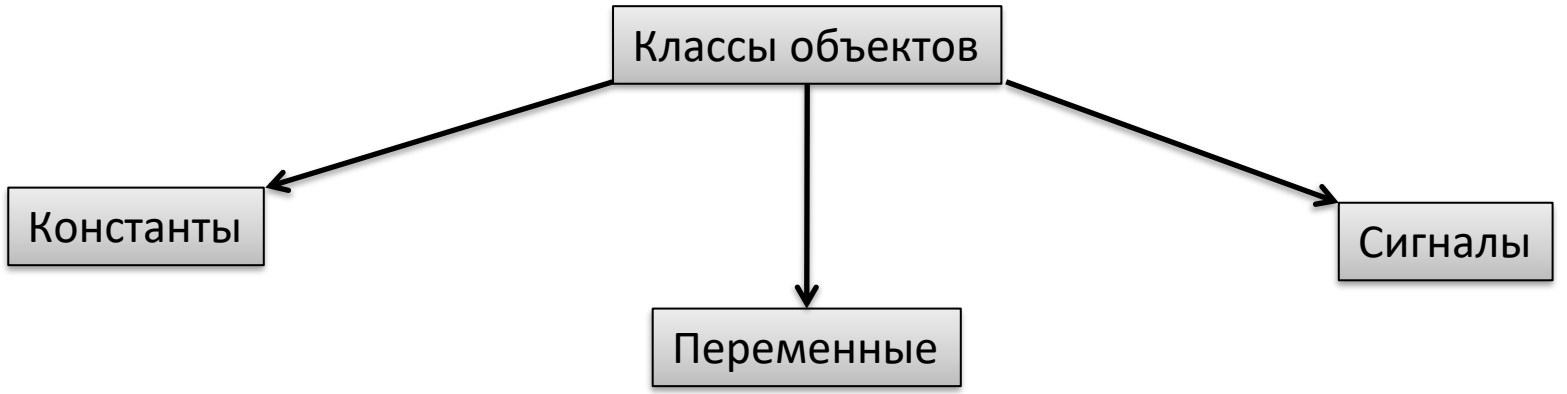


Атрибуты сигналов в VHDL

Имя атрибута	Назначение атрибута, что возвращает
`S'stable, S'stable(T)`	true, если сигнал не менялся (в течение времени T)
`S'event`	true, если происходит изменение сигнала
`S'active`	true, если была запись, но значение ещё не поменялось
`S'quiet(T)`	true, если не было обращений в течение времени T
`S'last_value`	предыдущее значение сигнала
`S'last_event`	время предыдущего изменения сигнала

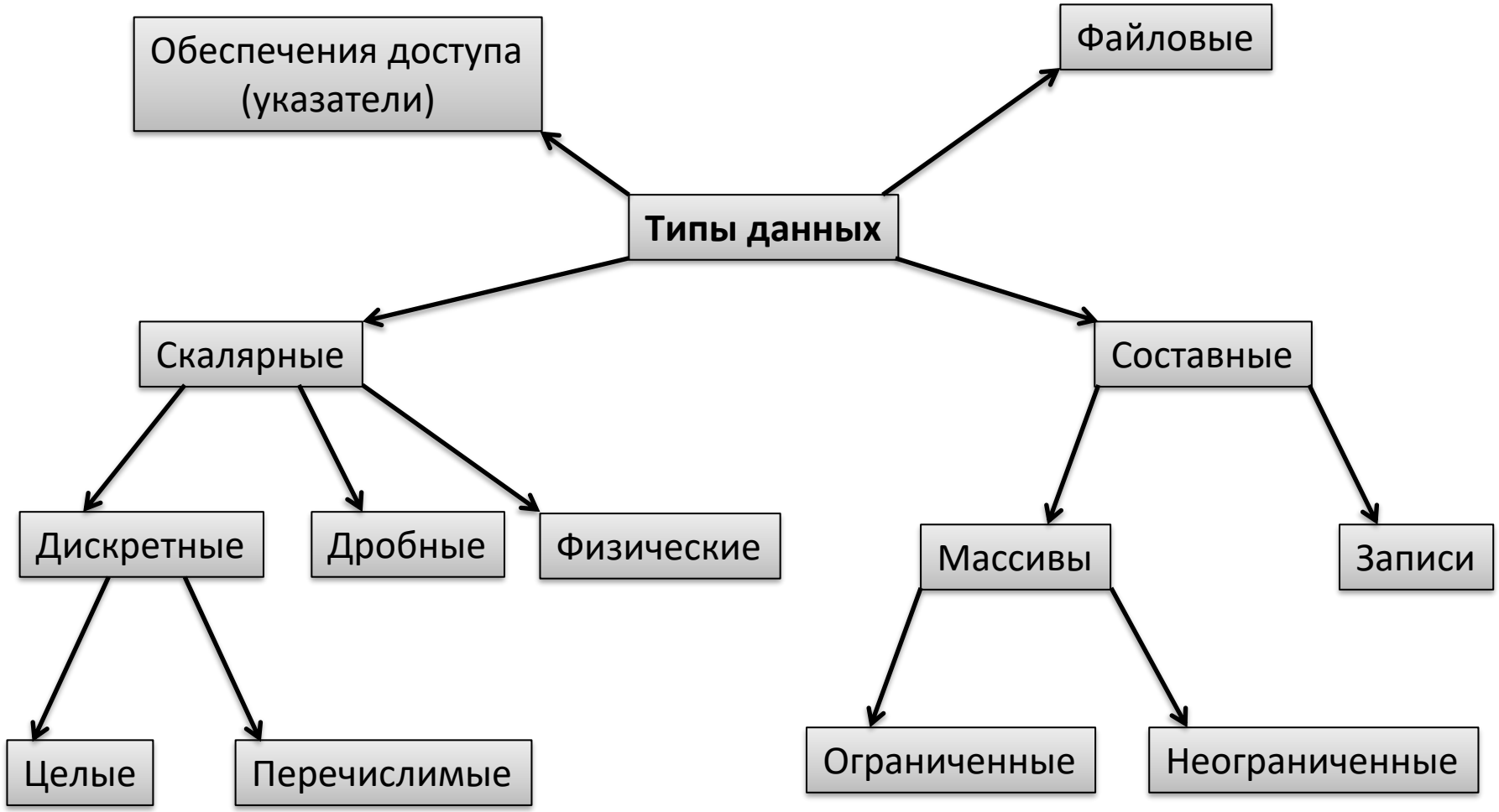
```
process
  variable a1, a2: Boolean;
begin
  Q <= '1' after 10 ns;

  a1 := Q'active;              -- true

  a2 := Q'event;               -- false
```

Классы объектов в VHDL




```
constant LOGIC_ONE : std_logic := '1';

variable TEMP_VAR  : std_logic := '1';

signal   CONNECT   : std_logic := '1';
```

Типы данных языка VHDL



Векторные типы данных (1)

Вектор с типом данных bit:

```
signal x: bit_vector(1 to 8);
```

Вектор с типом данных std_logic:

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

...

signal x: std_logic_vector(1 to 8);

signal x: std_logic_vector(8 downto 1);

signal x: std_logic_vector(10 downto 6);
```

Векторные типы данных (2)

```
signal x: std_logic_vector(1 to 8);
```

```
x    <= "10110101";                    x : "10110101";
x(1) <= '0';                           x : "00110101";
x    <= ('1', others => '0');          x : "10000000";
x    <= x(5 to 8) & x(1 to 4);         x : "00001000";
x    <= x(1) & "001100" & '1';         x : "00011001";
x    <= (others => '1');               x : "11111111";
```

Атрибуты для работы с массивами

Имя атрибута	Назначение атрибута
A'left	левая граница массива
A'right	правая граница массива
A'high	максимальное значение диапазона
A'low	минимальное значение диапазона
A'range	диапазон массива
A'length	длина массива

```
signal x: std_logic_vector(1 to 4)     := "0000";
signal y: std_logic_vector(4 downto 1) := "0000";
```

Имя атрибута	Что вернёт
x'left	1
x'right	4
x'high	4
x'low	1
x'range	1 to 4
x'length	4

Имя атрибута	Что вернёт
y'left	4
y'right	1
y'high	4
y'low	1
y'range	4 downto 1
y'length	4

Объявление своих типов и подтипов данных в VHDL

```
INTEGER    от -(2^31 - 1) до +(2^31 - 1)

NATURAL    от 0 до +(2^31 – 1)

POSITIVE   от 1 до +(2^31 – 1)
```

```
type small_int is range -1023 to 1024;
```

или

```
subtype small_int is Integer range -1023 to +1024;
```

или

```
variable small_int: Integer is range -1023 to +1024;
```

Объявление перечислимых данных в VHDL

Перечислимые типы:

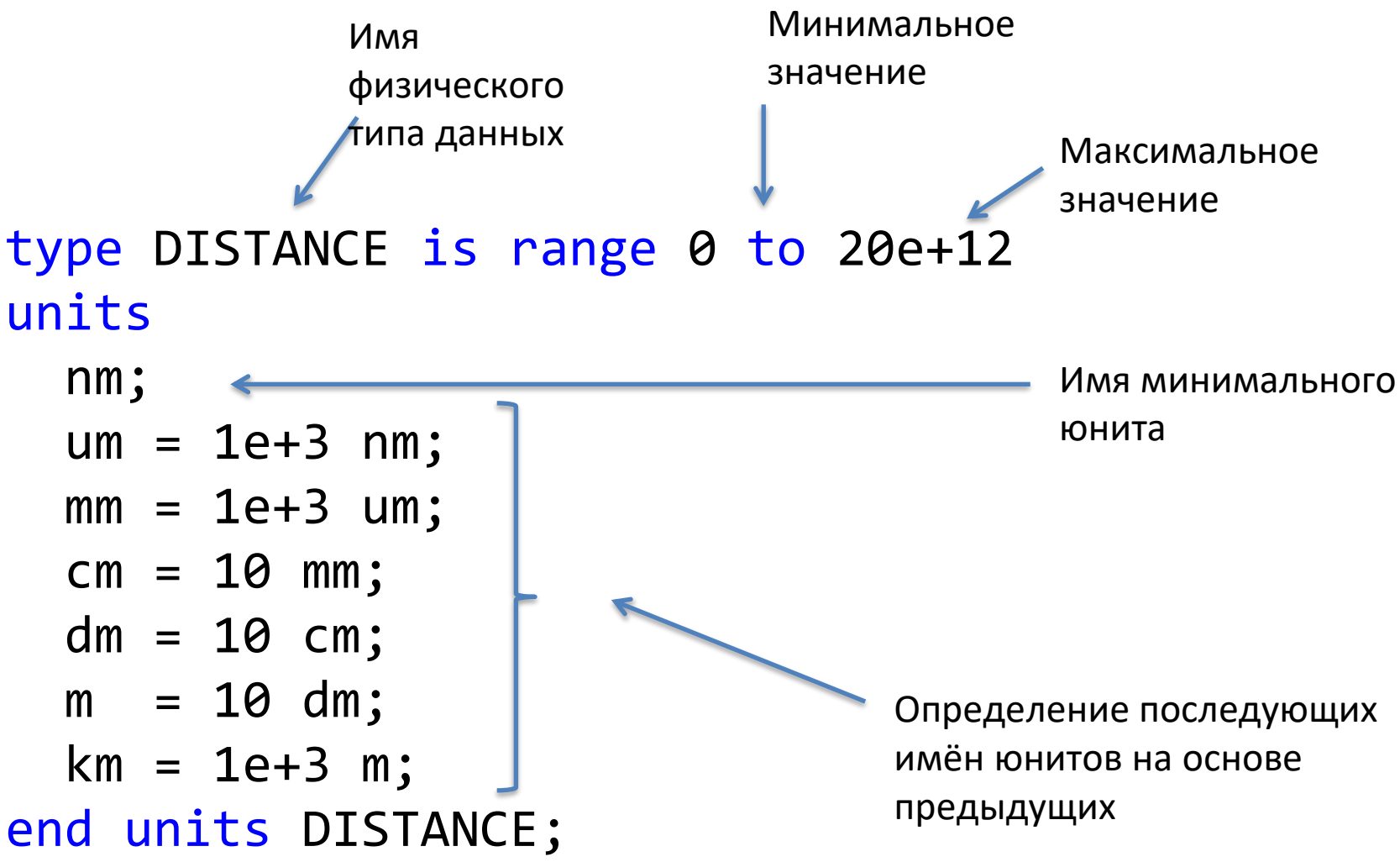
```
type BIT is ('0', '1');

type severity_level is (note, warning, error, failure);


type status is (unknown, idle, work);
```

Использование перечислимых типов:

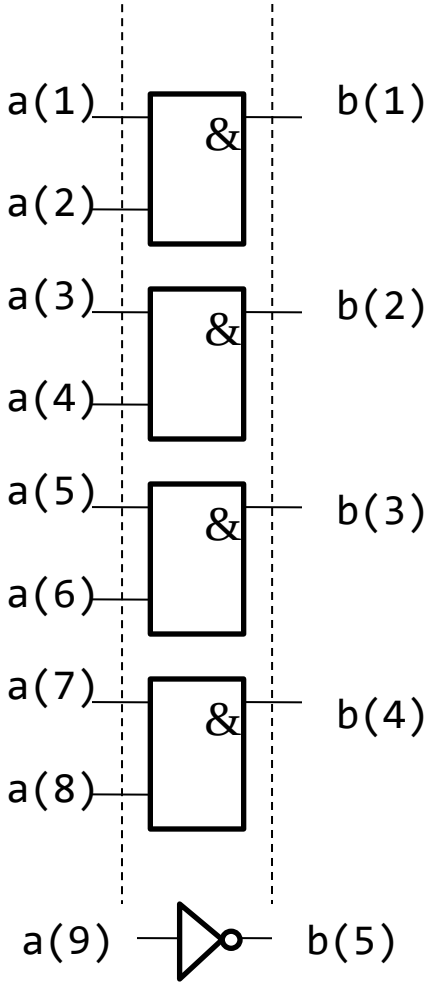
```
signal   A: bit;
variable B: bit;

...
A <= '1';
B := '0';
```

```
signal   S1: status;
variable S2: status;

...
A <= idle;
B := work;
```

Физические типы данных в VHDL



Оператор генерации подстановки компонента




```
architecture STR of DEVICE is
  component and2
    port(x1, x2: in  bit;
                 y: out bit);
  end component;
  component inv
    port(x: in  bit;
         y: out bit);
  end component;
  ...
  signal a: bit_vector(1 to 9);
  signal b: bit_vector(1 to 5);
begin

g1: for i in 1 to 4 generate

      p1: and2 port map(a(i*2-1), a(i*2), b(i));

    end generate;

p2: inv port map(a(9), b(5));

end STR;
```

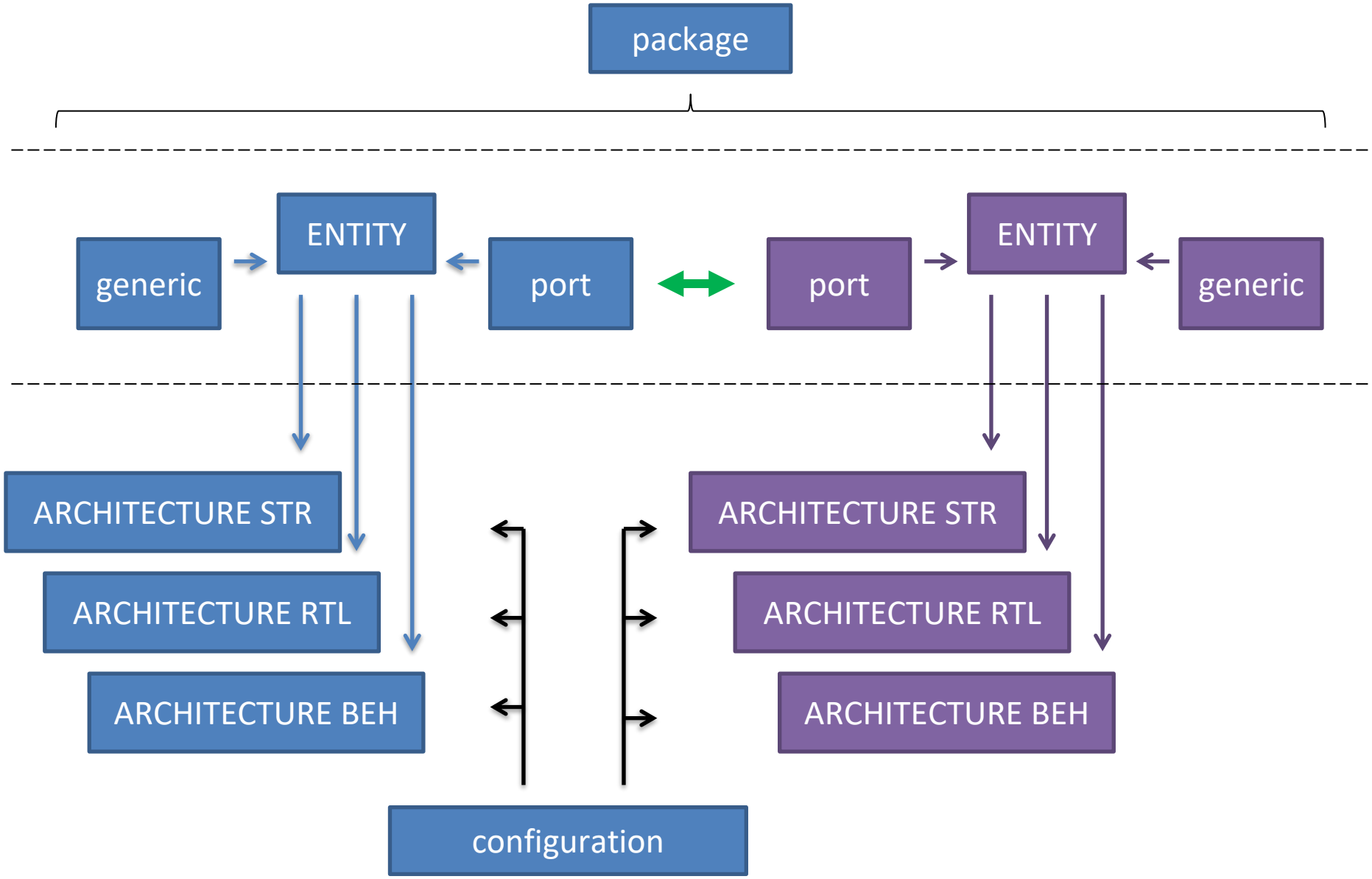
Конфигурации в языке VHDL

```
entity tb_device is
end tb_device;
```

```
architecture TEST of tb_device is
  component DEVICE
  ...
begin
  p1 : DEVICE port map ...
  p2 : DEVICE port map ...
  p3 : DEVICE port map ...
end TEST;
```

```
configuration config1 of tb_device is
  for TEST
    for p1 : DEVICE use entity work.DEVICE(STR); end for;
    for p2 : DEVICE use entity work.DEVICE(RTL); end for;
    for p3 : DEVICE use entity work.DEVICE(BEH); end for;
  end for;
end configuration;
```

Архитектура дизайна в языке VHDL



Запуск симулятора GHDL


```
ghdl -a adder.vhd

ghdl -a tb_adder.vhd



ghdl -r tb_adder --vcd=adder.vcd --stop-time=100ns
```

Библиотека SystemC для C++

```
entity inv is
  port(x: in  bit;
       y: out bit);
end inv;

architecture BEH of inv is
begin
  process(x)
  begin
    y <= not x;
  end process;
end RTL;
```

```
module inv(x, y);

  input  x;
  output y;

  always@(x)
    y = ~x;

endmodule
```

```
SC_MODULE(inv) {

  sc_in <bool>  x;
  sc_out<bool>  y;

  void operate() {
    y.write(!x.read());
  }

  SC_CTOR(inv) {
    SC_METHOD(operate);
    sensitive << x;
  }

};
```

Модуль MyHDL для Python

```
@block
def inv(x, y):

    @always(x)
    def logic():
        y.next = not x

    return logic


@block
def and2(x1, x2, y):

    @always(x1, x2)
    def logic():
        y.next = x1 & x2

    return logic
```

```
@block
def dff(q, d, clk):

    @always(clk.posedge)
    def logic():
        q.next = d

    return logic
```