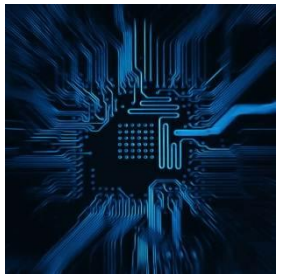
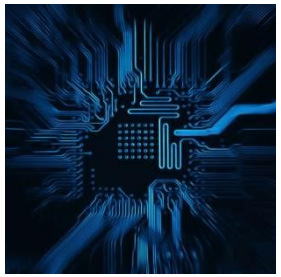


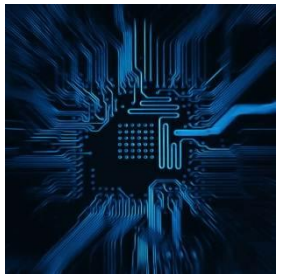
Основные конструкции языка VHDL

Программные блоки VHDL



Внешнее представление (entity):

Блок Entity



Entity – объявление модуля. Описывает внешний интерфейс модуля.

```
entity entity_name is
    port(
        port_names: mode data_type;
        port_names: mode data_type;
        ...
        port_names: mode data_type
    );
end entity_name;
```

mode – направление портов:

-in - входной (данные поступают в блок из вне)

-out – выходной (данные поступают из блока во вне)

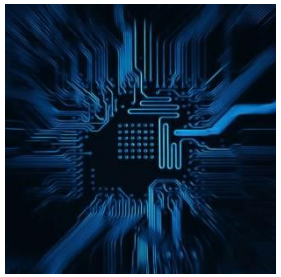
-Inout – двунаправленный порт (данные могут идти в обе стороны)

entity_name – имя блока, по которому он будет доступен из вне

port_names – имена портов

data_type – тип данных, передаваемых по порту

Блок Architecture



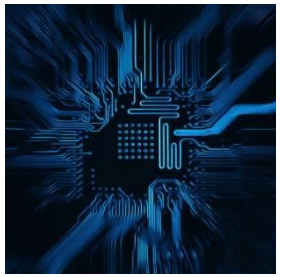
Architecture – архитектура модуля. Описывает внутреннюю реализацию. Возможно несколько реализаций одного и того же модуля.

```
architecture arch_name of entity_name is
   declarations;
begin
   concurrent statement;
   concurrent statement;
   concurrent statement;
   . . .
end arch_name;
```

arch_name – имя реализации модуля.
entity_name – имя модуля, который реализуется
declarations – объявление переменных и сигналов

concurrent statement – выражения, которые реализуют функциональность блока

Подключение внешних библиотек и пакетов



Подключение библиотеки:

LIBRARY library_name;

library_name – имя подключаемой библиотеки

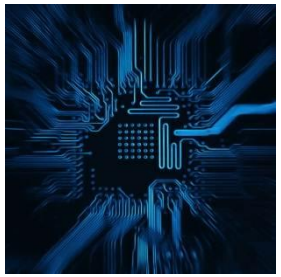
Подключение пакетов из библиотеки:

USE library_name.packet_name.(identifier | character_literal | operator_symbol | ALL);

packet_name – имя пакета из библиотеки library_name.

Последний идентификатор определяет что именно надо подключить из пакета (идентификатор ALL указывает, что всё).

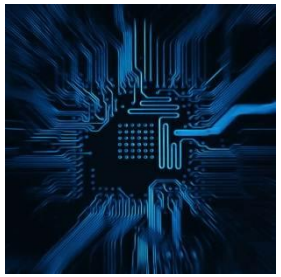
Элементы языка VHDL



- Комментарии . Начинаются с --
- Идентификаторы. my_signal, My_Signal – одно и тоже. Нету разницы между регистром букв. Можно использовать и цифры в назаниях: my_signal1, my_SIGnal03…
- Зарезервированные слова. Выделяются другим цветом
- Числа: присвоение значений сигналам типа integer
  - signal a: integer;
  - a <= 5, 0, 89E7;
  - Присвоение значений сигналам типа std_logic_vector:
  - signal vec: std_logic_vector (7 downto 0);
  - vec <= “11110111”; -- в двоичном формате
  - vec <= X”F7”; (либо 16#F7#)-- в 16-ом формате
- Объекты:
  - -- signal
  - -- variable
  - -- constant

```
constant BUS_WIDTH: integer := 32;
constant BUS_BYTES: integer := BUS_WIDTH / 8;
```

Использование signal в Architecture



ARCHITECTURE <Identifier> **OF**<Entity_identifier>**IS**

    SIGNAL clk**:** bit;

BEGIN

    Concurrent Statements

END ARCHITECTURE;

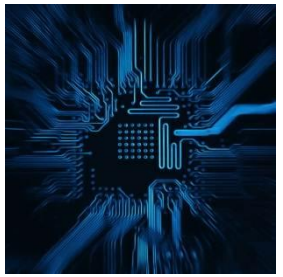
Структура описания сигнала: **Object_name: <Type> := <Value>;**

Object_name: имя

<Type>: тип

<Value>: инициализирующее значение

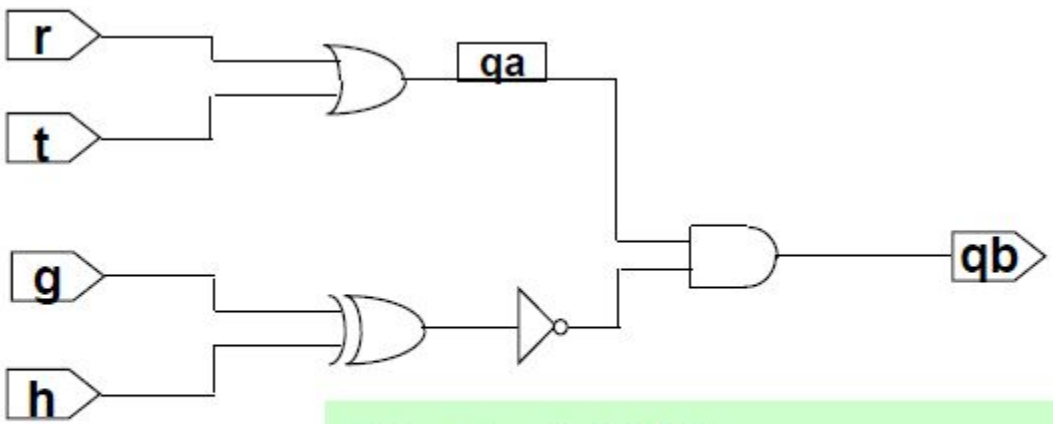
Использование signal в Architecture



Формат: Z <= A; (*A называют драйвером Z)*

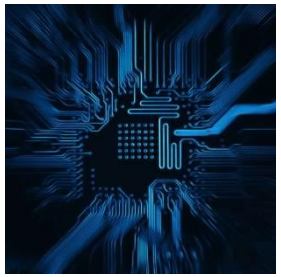
<signal_name> **<=** *<expression>***;**

Пример:

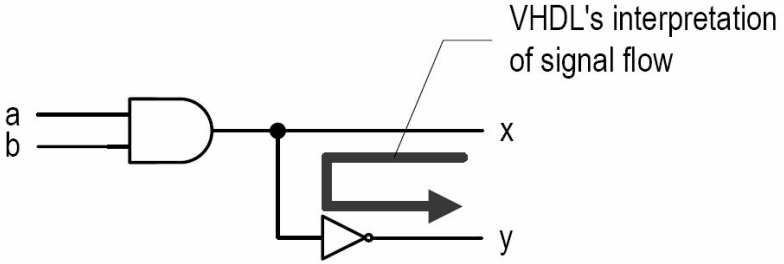



```
qa <=  r or t ;
qb <=  qa and not(g xor h);
```

Основные конструкции языка VHDL



Сигнал типа OUT не может быть присвоен другому сигналу

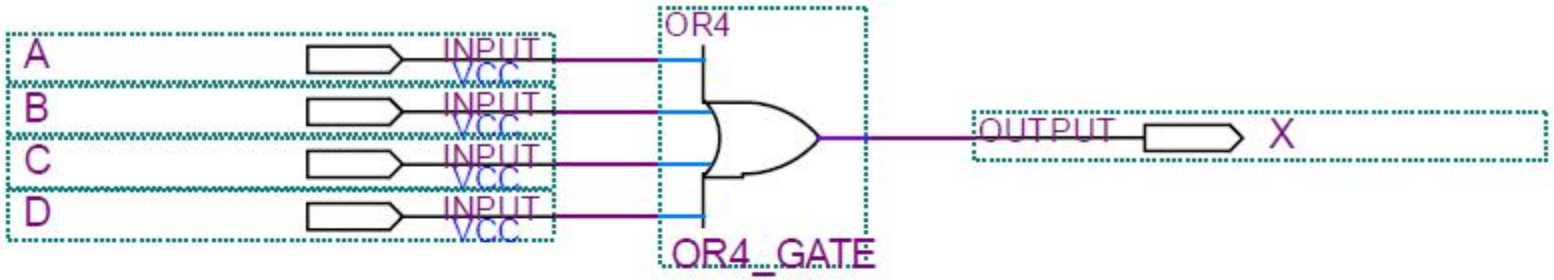



```
library ieee;
use ieee.std_logic_1164.all;
entity mode_demo is
  port(
    a, b: in std_logic;
    x, y: out std_logic);
end mode_demo;
architecture wrong_arch of mode_demo is
begin
    x <= a and b;
    y <= not x;
end wrong_arch;
```

Необходимо использовать промежуточный сигнал

```
architecture ok_arch of mode_demo is
    signal ab: std_logic;
begin
    ab <= a and b;
    x <= ab;
    y <= not ab;
end ok_arch ;
```

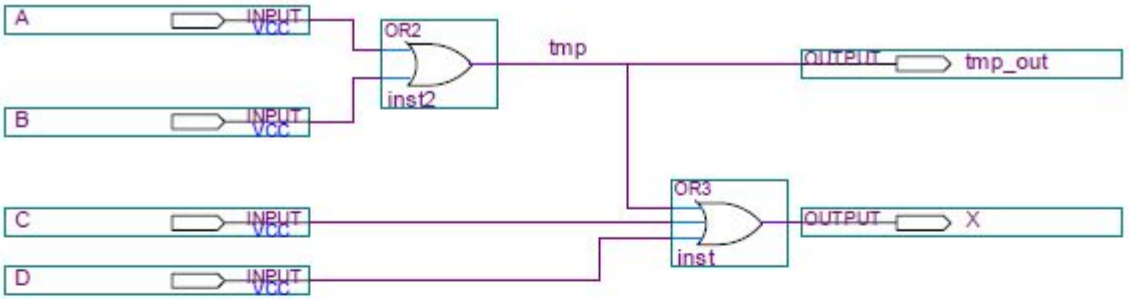
Пример задания архитектуры




```
ENTITY or4_gate IS
        PORT (
                A,B,C,D  : IN      BIT;
                X        : OUT     BIT
              );
END or4_gate;
```

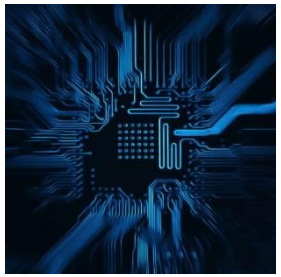
```
ARCHITECTURE arch_or4_gate OF or4_gate IS
BEGIN
        x<=a or b or c or d;
END arch_or4_gate;
```

Использование сигналов




```
ENTITY or4_gate IS
   PORT (
            A,B,C,D          : IN    BIT;
            X, tmp_out       : OUT  BIT
           );
END or4_gate;

ARCHITECTURE arch_or4_gate OF or4_gate IS

   SIGNAL tmp: BIT;

BEGIN
   tmp          <=        a or b;
   X            <=        tmp or c or d;
   tmp_out      <=        tmp;
END arch_or4_gate;
```

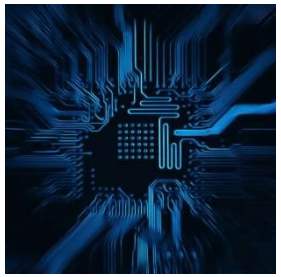
Типы данных VHDL



- Integer: от $(-2^{31}-1)$ до $(2^{31}-1)$. Представляется как 32-х битный массив
- Bollean : (true, false)
- Bit : (‘0’, ‘1’)
- Bit_vector – массив значений типа Bit

- В пакете std_logic_1164 определены дополнительные типы
- std_logic: (‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘-’)
- Значения для синтеза:
  - ‘0’, ‘1’ – тоже что и в типе bit – обыкновенные ‘0’ и ‘1’
  - ‘Z’ – состояние с высоким импедансом
- Значения для моделирования:
  - ‘L’, ‘H’ – слабый ‘0’, слабая ‘1’ (имеется в виду слабый ток)
  - ‘X’, ‘W’ – неизвестное, слабое неизвестное
  - ‘U’ – неинициализированное
  - ‘-’ – неважное значение
- std_logic_vector – массив значение типа std_logic

Задание диапазона



RANGE–задает диапазон изменения индексов в массиве

Диапазон может быть задан:

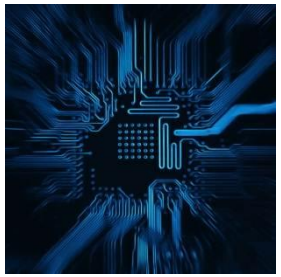
- –ввозрастающей последовательности (ascending)

  RANGE *left_bound* TO *right_bound*

- –в убывающей последовательности (descending).

  RANGE *left_bound* DOWNTO *right_bound*

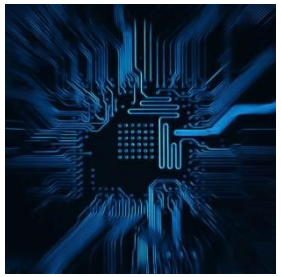
Одинаково или нет?



```
BEGIN
   tmp          <=        a or b;
   x            <=        tmp or c or d;
   tmp_out      <=        tmp;
END arch_or4_gate;
```

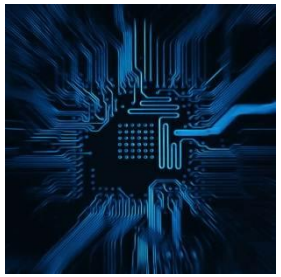
```
BEGIN
   x            <=        tmp or c or d;
   tmp_out      <=        tmp;
   tmp          <=        a or b;
END arch_or4_gate;
```

Операторы VHDL



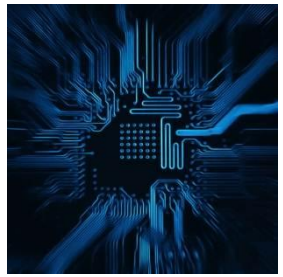
operator	description	data type of operand a	data type of operand b	data type of result
`a ** b`	exponentiation	`integer`	`integer`	`integer`
`abs` `a`	absolute value	`integer`		`integer`
`not` `a`	negation	`boolean, bit, bit_vector`		`boolean, bit, bit_vector`
`a * b`	multiplication	`integer`	`integer`	`integer`
`a / b`	division			
`a` **`mod`** `b`	modulo			
`a` **`rem`** `b`	remainder			
`+ a`	identity	`integer`		`integer`
`- a`	negation			
`a + b`	addition	`integer`	`integer`	`integer`
`a - b`	subtraction			
`a & b`	concatenation	1-D array, element	1-D array, element	1-D array

Операторы VHDL



a **sll** b	shift left logical	bit_vector	integer	bit_vector
a **srl** b	shift right logical			
a **sla** b	shift left arithmetic			
a **srl** b	shift right arithmetic			
a **rol** b	rotate left			
a **ror** b	rotate right			
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
a **and** b	and	boolean, bit,	same as a	boolean, bit,
a **or** b	or	bit_vector		bit_vector
a **xor** b	xor			
a **nand** b	nand			
a **nor** b	nor			
a **xnor** b	xnor			

Приоритеты операторов



Приоритет	Опрераторы
Выше	**, **abs**, **not**
	*, /, **mod**, **rem**
	+, - (унарные)
	&, +, - (бинарные)
	sll, srl, sla, sra, rol, ror
	=, /=, <, <=, >, >=
Ниже	**and, or, nand, nor, xor, xnor**

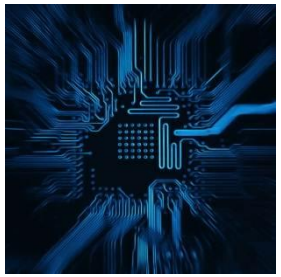
Пример: a + b > c **or** a < d

Скобки: a + b + c + d VS a + (b+(c+d))

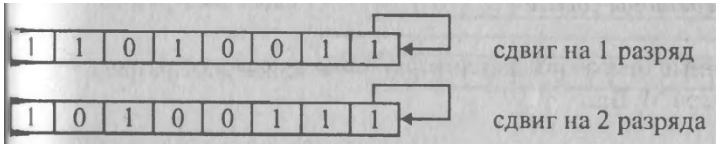
Для операторов **and** и **or** необходимо напрямую указывать порядок их выполнения, т.к. в VHDL

они имеют одинаковый приоритет (в отличие от Булевой алгебры).
(a **and** b) **or** (c **and** d)

Арифметический сдвиг **sla, sra**

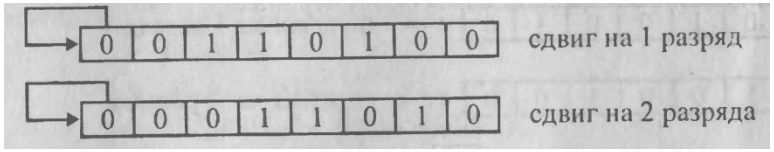


sla - сдвиг влево.



Размножается младший разряд

sra - сдвиг вправо.



Размножается старший разряд

Остальные сдвиги рассматриваем самостоятельно по книге Бибило

Оператор конкатенации &

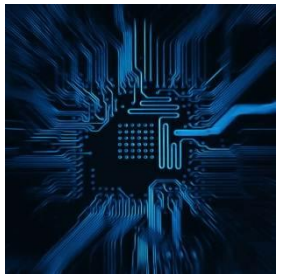
Объединяет отдельные биты в массив или шину

```
signal a,b: std_logic;
signal c: std_logic_vector(1 downto 0);
c <= a & b;
```

Реализация оп-ра арифметического сдвига с помощью оп-ра конкатенации

```
signal a, y: std_logic_vector (7 downto 0);
вправо:  y <= a(7) & a(7 downto 1);  -- 1 разряд
         y <= a(7)&a(7)&a(7 downto 2); --2
влево:   y <= a(6 downto 0) & a(0); -- 1 разряд
         y <= a(5 downto 0)&a(0)&a(0); -- 2
```

Присвоение значения массиву



signal a: std_logic_vector (7 downto 0);

a <= “10100000”;

Позиционная привязка:

a <= (‘1’,’0’,’1’,’0’, ’0’, ’0’, ’0’, ’0’);

Привязка по имени:

a <= (7 => ‘1’, 6 => ‘0’, 0 => ‘0’, 1 => ‘0’, 5 => ‘1’, 4 => ‘0’, 3 => ‘0’, 2 => ‘0’)

или: a <= (7|5 => ‘1’, 6|4|3|2|1|0 => ‘0’);

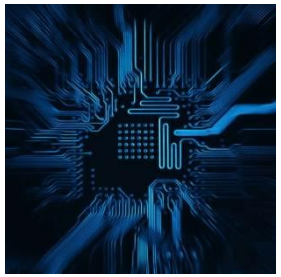
с использованием ключевого слова **others**

a <= (7|5 => ‘1’, **others** => ‘0’);

Присвоение нулевого значения:

a <= (**others** => ‘0’);

Преобразование типов



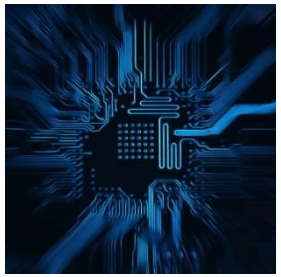
Пакет std_logic_1164

- to_bitvector – преобразование из std_logic_vector в bit_vector
- to_stdlogicvector – преобразование из bit_vector в std_logic_vector
- to_bit – преобразование из std_logic в bit.

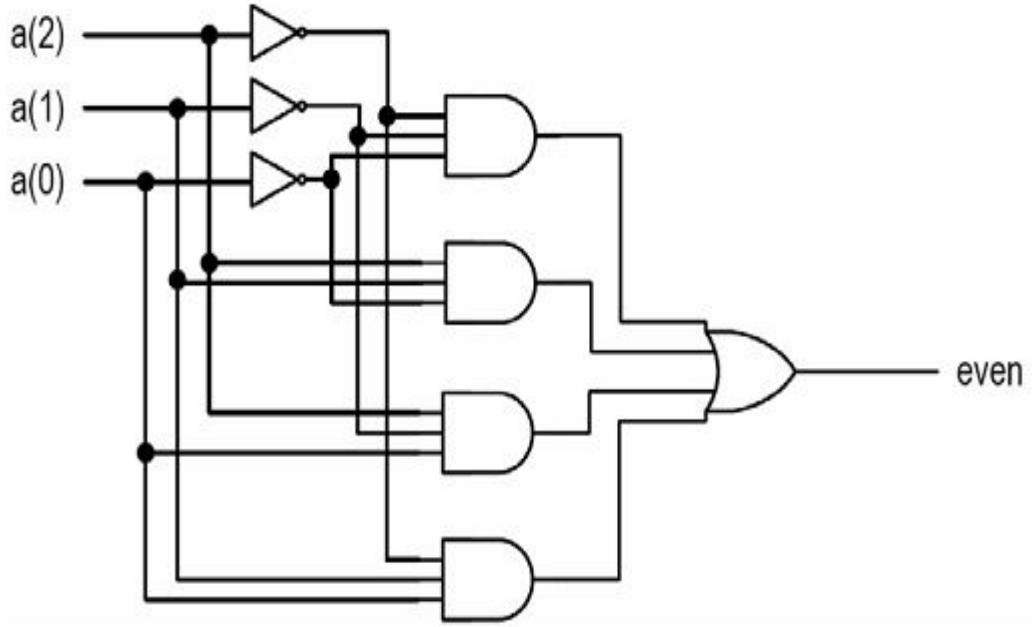
Пакеты std_logic_arith и std_logic_unsigned (либо std_logic_signed)

- conv_integer – преобразование из std_logic_vector в integer
- conv_std_logic_vector – преобразование из integer в std_logic_vector. Первый аргумент – само число, второй – ширина выходной шины.

Схема определения четности

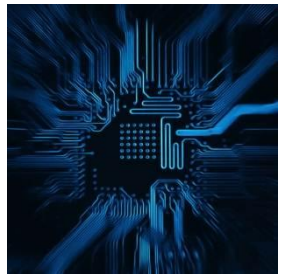


a(2)	a(1)	a(0)	even
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



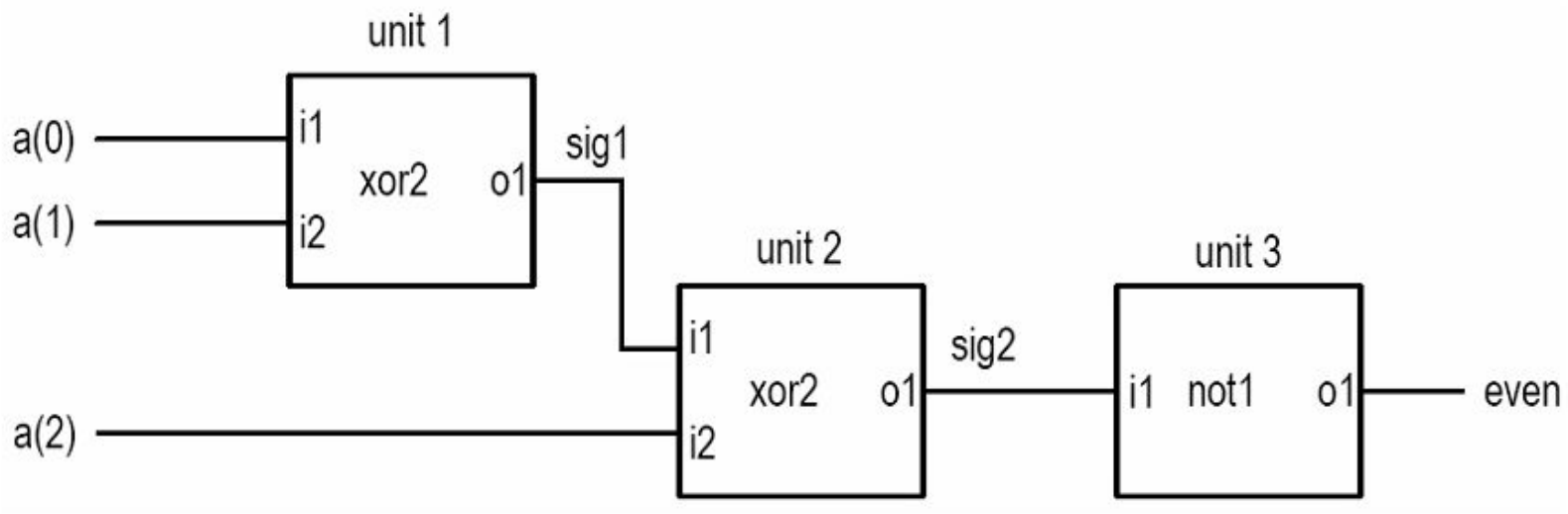
$$even = a0'a1'a2' + a0a1a2' + a0'a1a2' + a0'a1a2$$

Схема определения четности



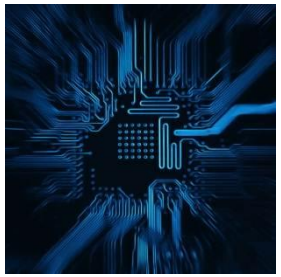
```
library ieee;
use ieee.std_logic_1164.all;
entity even_detector is
   port(
      a: in std_logic_vector(2 downto 0);
      even: out std_logic);
end even_detector;

architecture sop_arch of even_detector is
   signal p1, p2, p3, p4 : std_logic;
begin
   even <= (p1 or p2) or (p3 or p4)
   p1 <= (not a(2)) and (not a(1)) and (not a(0))
   p2 <= (not a(2)) and a(1) and a(0)
   p3 <= a(2) and (not a(1)) and a(0)
   p4 <= a(2) and a(1) and (not a(0))
end sop_arch ;
```

Схема определения четности. Структурное описание



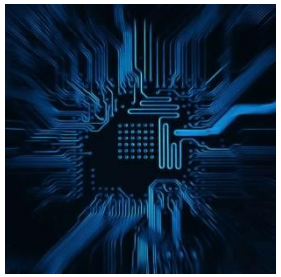
- $even = a0'a1'a2' + a0a1a2' + a0'a1a2' + a0'a1a2$

Схема определения четности. Структурное описание



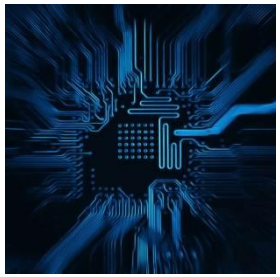
```
architecture str_arch of even_detector is
   component xor2
      port(
         i1, i2: in std_logic;
         o1: out std_logic);
   end component;
   component not1
      port(
         i1: in std_logic;
         o1: out std_logic);
   end component;
   signal sig1,sig2: std_logic;

begin
   unit1: xor2
      port map (i1 => a(0), i2 => a(1), o1 => sig1);
   unit2: xor2
      port map (i1 => a(2), i2 => sig1, o1 => sig2);
   unit3: not1
      port map (i1 => sig2, o1 => even);
end str_arch;
```

Схема определения четности. Структурное описание

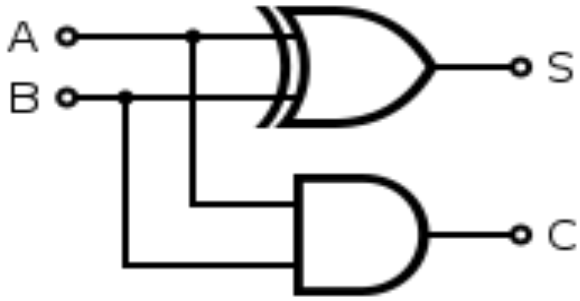


```
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
   port(
      i1, i2: in std_logic;
      o1: out std_logic);
end xor2;

architecture beh_arch of xor2 is
begin
   o1 <= i1 xor i2;
end beh_arch;

library ieee;
use ieee.std_logic_1164.all;
entity not1 is
   port(
      i1: in std_logic;
      o1: out std_logic);
end not1;
architecture beh_arch of not1 is
begin
   o1 <= not i1;
end beh_arch;
```

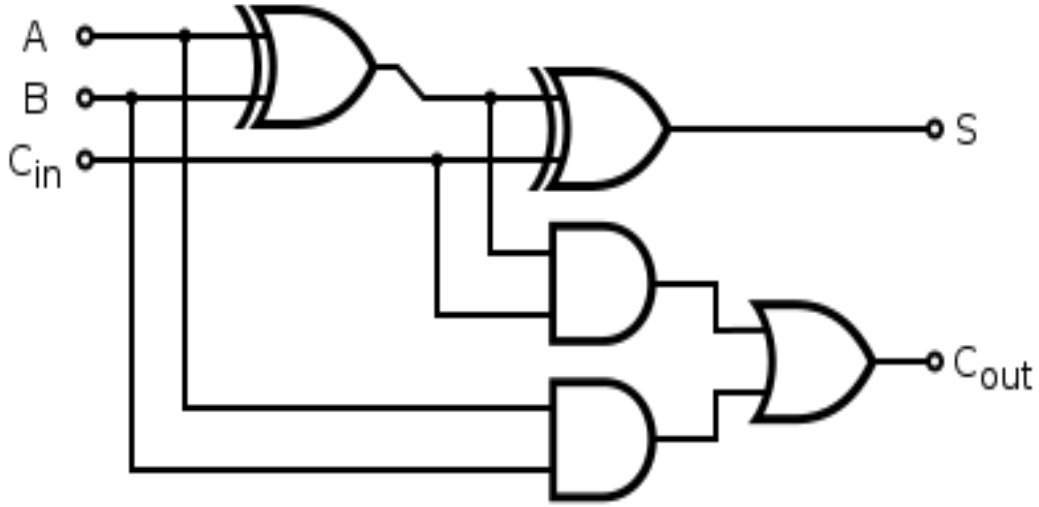
Практика



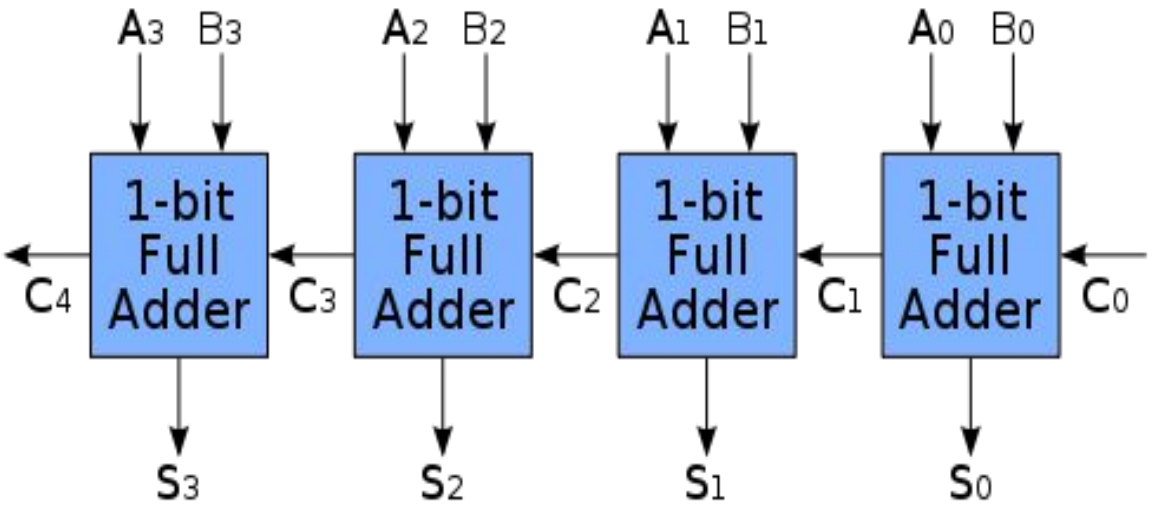
Полусумматор:



Элемент сумматора:



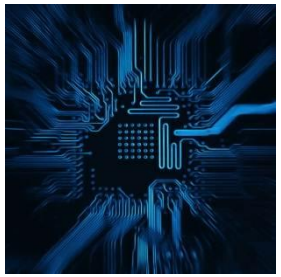
Сумматор



Задание: написать сумматор, изображённый на схеме. В итоге должно быть три файла: half_adder.vhd (полусумматор), full_adder_1.vhd (элемент сумматора, построенный на основе half_adder.vhd) и comb_02_adder.vhd (8-и битный сумматор). У компонента верхнего уровня должны быть следующие сигналы в интерфейсной части:

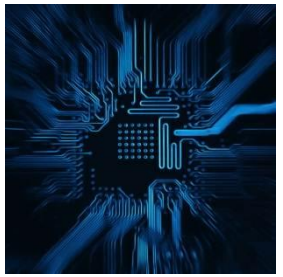
switch_in : in STD_LOGIC_VECTOR (7 downto 0);
leds_out : out STD_LOGIC_VECTOR (7 downto 0)

Моделирование. Добавление элемента



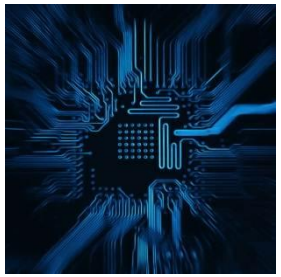
```
-- Instantiate the Unit Under Test (UUT)
   uut: comb_02_adder PORT MAP (
          switch_in => switch_in,
          leds_out => leds_out
        );
```

Моделирование процесс задания сигналов



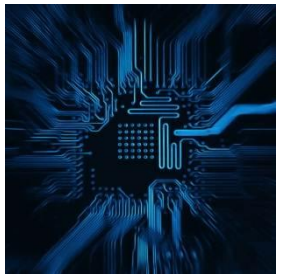
```
switch_in      <= B & A;
  -- Stimulus process
  stim_proc: process
  begin
            A <= "0000";
            B <= "0000";
    wait for 100 ns;
            A <= "0001";
            B <= "0100";
    wait for 100 ns;
wait;
end process;
```

Моделирование



Задание: написать testbench (adder_1_tb.vhd) для 8-ми битного сумматора

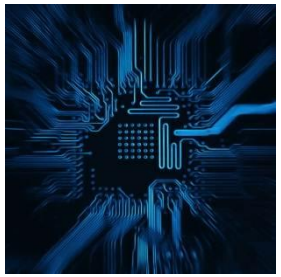
Реализация сумматора на плате Atlys



Создайте проект для реализации сумматора: входы к переключателям, а выходы – к диодам.

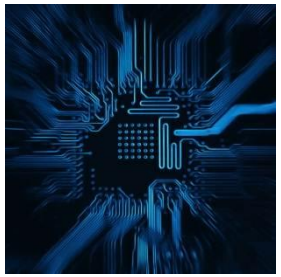
Для подключения потребуется ucf файл, в котором указано к каким ножкам необходимо подключить линии данных.

Входы



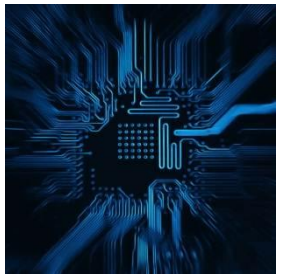
```
# onBoard switch
NET "switch_in<0>" LOC = "A10";
NET "switch_in<1>" LOC = "D14";
NET "switch_in<2>" LOC = "C14";
NET "switch_in<3>" LOC = "P15";
NET "switch_in<4>" LOC = "P12";
NET "switch_in<5>" LOC = "R5";
NET "switch_in<6>" LOC = "T5";
NET "switch_in<7>" LOC = "E4";
```

Выходы



```
# onBoard leds_outs
NET "leds_out<0>" LOC = "U18";
NET "leds_out<1>" LOC = "M14";
NET "leds_out<2>" LOC = "N14";
NET "leds_out<3>" LOC = "L14";
NET "leds_out<4>" LOC = "M13";
NET "leds_out<5>" LOC = "D4";
NET "leds_out<6>" LOC = "P16";
NET "leds_out<7>" LOC = "N12";
```

For generate



```
g1: for i in 0 to 3 generate
begin
        b: full_adder_1 port map (
                a           => A(i),
                b           => B(i),
                c_in            => C(i),
                s           => S(i),
                c_out           => C(i+1)
                );
end generate g1;
```

Задание: написать 4-х битный сумматор (comb_gen_adder.vhd) используя конструкцию *for generate* по аналогии с comb_02_adder.vhd.
Протестировать его с помощью testbench'а и опробовать на железе.