

Атрибуты языка VHDL

Атрибуты — это элементы языка VHDL, соотнесенные с определенными объектами языка VHDL и отражающие их некоторые свойства.

Атрибутами могут выступать следующие элементы языка VHDL :

- значение ;
- диапазон ;
- функция ;
- сигнал ;
- тип ;
- константа .

Синтаксис доступа к атрибуту :

`имя_атрибутируемого_объекта 'имя_атрибута`

Пример :

`clk 'event ;`

Атрибуты языка VHDL (категории атрибутов)

Атрибуты подразделяются на следующие категории :

- предопределенные атрибуты ;
- пользовательские атрибуты .

Предопределенные атрибуты предоставляют дополнительную информацию о указанном объекте. Сигналам, представляющим атрибуты, нельзя присваивать новые значения.

Пользовательские атрибуты представляют собой константы произвольного типа. Чаще всего пользовательские атрибуты используются для передачи инструментам моделирования или синтеза дополнительной информации об определенных объектах проекта на языке VHDL.

Атрибуты языка VHDL

(пользовательские атрибуты)

Синтаксис объявления пользовательского атрибута :

```
attribute имя_атрибута : тип_атрибута ;
```

Имя атрибута впоследствии используется для ссылки на него.

Синтаксис определения атрибута заданному объекту :

```
attribute имя_атрибута of спецификация_объекта  
is выражение ;
```

Спецификация объекта задается в виде :

```
имя_объекта : класс_объекта
```

Имя объекта ссылается на атрибутируемый объект. Класс объекта может принимать следующие значения : **entity**, **architecture**, **signal**, **variable** и т.д.

Атрибуты языка VHDL (примеры определения пользовательских атрибутов)

Для сигнала **clk** определяем атрибут с именем **KEEP** и со значением типа **string** – **"TRUE"** :

```
attribute KEEP : string ;  
attribute KEEP of clk : signal is "TRUE" ;
```

Для архитектуры **sdram_controller** определяем атрибут с именем **KEEP_HIERARCHY** и со значением типа **string** – **"TRUE"** :

```
attribute KEEP_HIERARCHY : string ;  
attribute KEEP_HIERARCHY of  
    csdram_controller : architecture is "TRUE" ;
```

Предопределенные атрибуты языка VHDL (атрибуты типов)

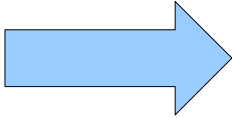
T'left	Левая граница значений T
T'right	Правая граница значений T
T'low	Нижняя граница значений T
T'high	Верхняя граница значений T
T'image (X)	Строка символов, представляющая значение X
T'pos (X)	Позиция значения X в наборе значений T
T'value (N)	Значение элемента в позиции N в наборе значений T
T'leftof (X)	Значение в наборе значений T, в позиции слева от X
T'rightof (X)	Значение в наборе значений T, в позиции справа от X
T'pred (X)	Значение в наборе значений T, на одну позицию меньше от X
T'succ (X)	Значение в наборе значений T, на одну позицию больше от X

Атрибуты языка VHDL

(использование атрибутов типа)

```
type example is (one,two,three) ;  
type positive is range -2147483648 to 2147483648;  
type positive is integer range to integer'high ;
```

example'left	three
example'right	one
positive'low	1
positive'high	2147483647
integer'image(256)	"256"
example'pos	2
example'value(2)	three
example'leftof(two)	one
example'rightof(two)	three
integer'pred(2)	1
integer'succ(2)	3



Предопределенные атрибуты языка VHDL (атрибуты агрегатов)


A'left (N)	Левая граница диапазона индексов N-й координаты массива A
A'right (N)	Правая граница диапазона индексов N-й координаты массива A
A'low (N)	Нижняя граница диапазона индексов N-й координаты массива A
A'high (N)	Верхняя граница диапазона индексов N-й координаты массива A
A'range (N)	Диапазон индексов N-й координаты массива A
A'reverse_range (N)	Обратный диапазон индексов N-й координаты массива A
A'length (N)	Длина интервала индексов N-й координаты массива A
A'ascending (N)	TRUE, если интервал индекса задан от меньшего к большому

Атрибуты языка VHDL

(использование атрибутов типа)

```
type example is array (2 downto 1, 0 to 3) of integer ;
```

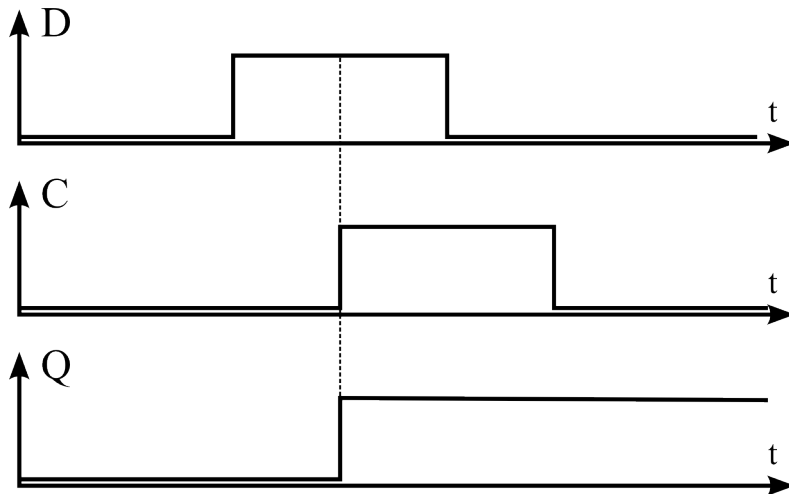
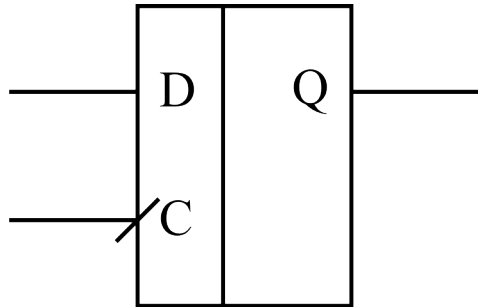
example'left(1)	2
example'right(2)	3
example'low(2)	0
example'high(1)	2
integer'range(2)	0 to 3
example'reverse_range(1)	1 to 2
example'length(2)	4
example'ascending(two)	one



Предопределенные атрибуты языка VHDL (атрибуты сигналов)

S'delayed(T)	Значение S, существовавшее на время T перед вычислением атрибута
S'event	Сигнализирует об изменении сигнала
S'stable	not S'event
S'active	TRUE, если присвоение сигналу выполнено, но значение еще не изменено (не окончен временной интервал, заданный выражением after)
S'quite	not S'active
S'last_event	Время от момента вычисления атрибута до последнего изменения сигнала
S'last_active	Время от момента вычисления атрибута до последнего присвоения сигналу (не совпадает с last_event при наличии слова after в определяющем выражении)
S'last_value	Значение S, непосредственно перед последним изменением

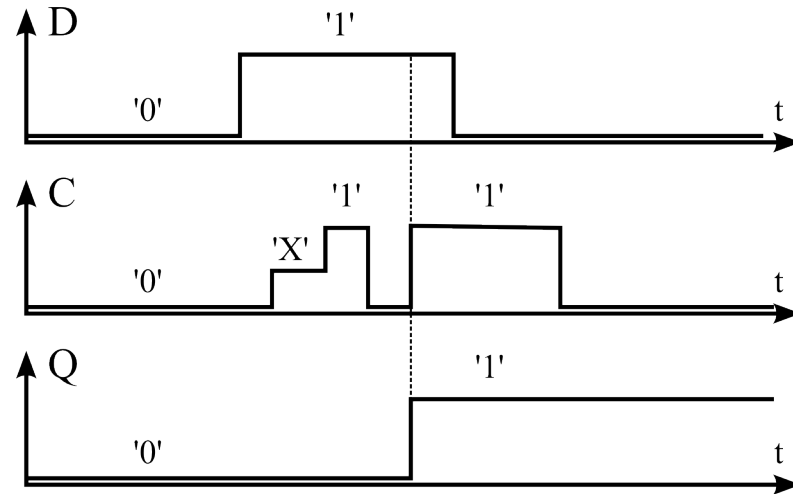
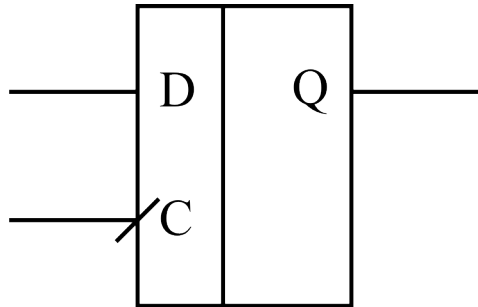
Пример использования атрибутов в проекте на VHDL



```
entity ff is
    port (
        C,D : in    std_logic;
        Q   : out   std_logic
    );
end ff;

architecture behavioural of ff is
begin
    if C'event and C = '1' then
        Q <= D;
    end if;
end ff;
```

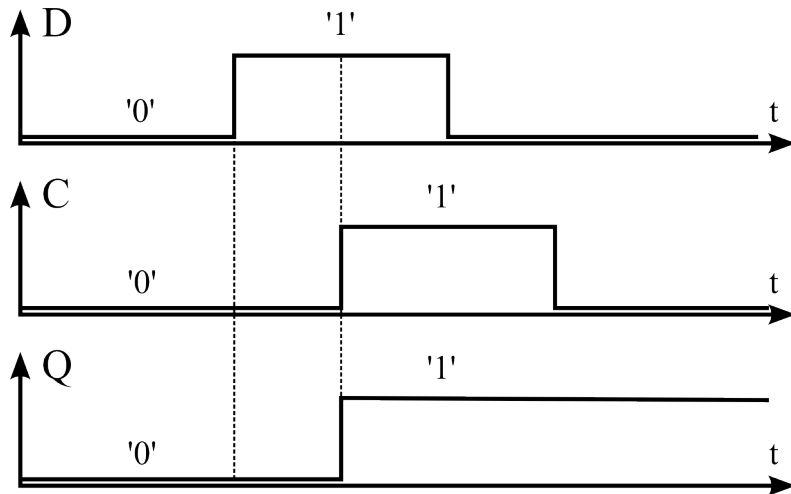
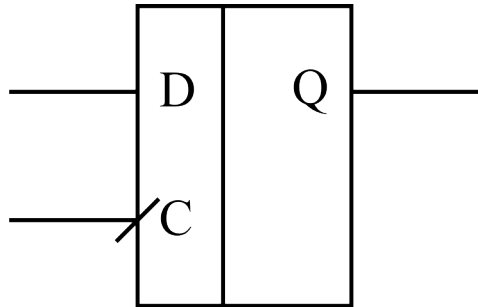
Пример использования атрибутов в проекте на VHDL



```
entity ff is
    port (
        C,D : in    std_logic;
        Q   : out   std_logic
    );
end ff;

architecture behavioural of ff is
begin
    if C'event and C = '1' and C'last_value = '0' then
        Q <= D;
    end if;
end ff;
```

Пример использования атрибутов в проекте на VHDL



```
entity ff is
    port (
        C,D : in    std_logic;
        Q    : out   std_logic
    );
end ff;

architecture behavioural of ff is
begin
    if C'event and C = '1' then
        assert D'last_event >= 5 ns
            report "Setup violation"
            severity error;
        Q <= D;
    end if;
end ff;
```

Операторы языка VHDL

Все операторы языка VHDL подразделяются на три основных типа:

- операторы объявления (декларации) ;
- параллельные операторы ;
- последовательные операторы .

Операторы объявления используются для определения **констант** (цифровых или строковых), **типов** (скалярных, агрегатных и т.д.), **объектов** (таких как *сигналы*, *переменные*, *компоненты*) и **подпрограмм** (*процедур* и *функций*) используемых в проекте.

Операторы объявления могут использоваться в самых различных частях VHDL проекта в зависимости от области действия элементов, которые они определяют.

Операторы объявления языка VHDL (объявление компоненты)

Синтаксис объявления компоненты :

```
component имя_компоненты [is]  
    [настроечные константы]  
    [порты объекта]  
end component [имя_компоненты] ;
```

Оператор объявления компоненты обычно используется в декларативной части проектного модуля **architecture**.

Имя компоненты должно совпадать с именем проектного модуля **entity**, который определяет объект, используемый в качестве компоненты.

Операторы объявления языка VHDL (объявление компоненты)

Пример использования :

```
component register
```

```
generic (
```

```
    rise,fall    : time
```

```
);
```

```
port (
```

```
    input  : in std_logic_vector(7 downto 0);
```

```
    clk    : in std_logic;
```

```
    output : out std_logic_vector(7 downto 0)
```

```
);
```

```
end component;
```

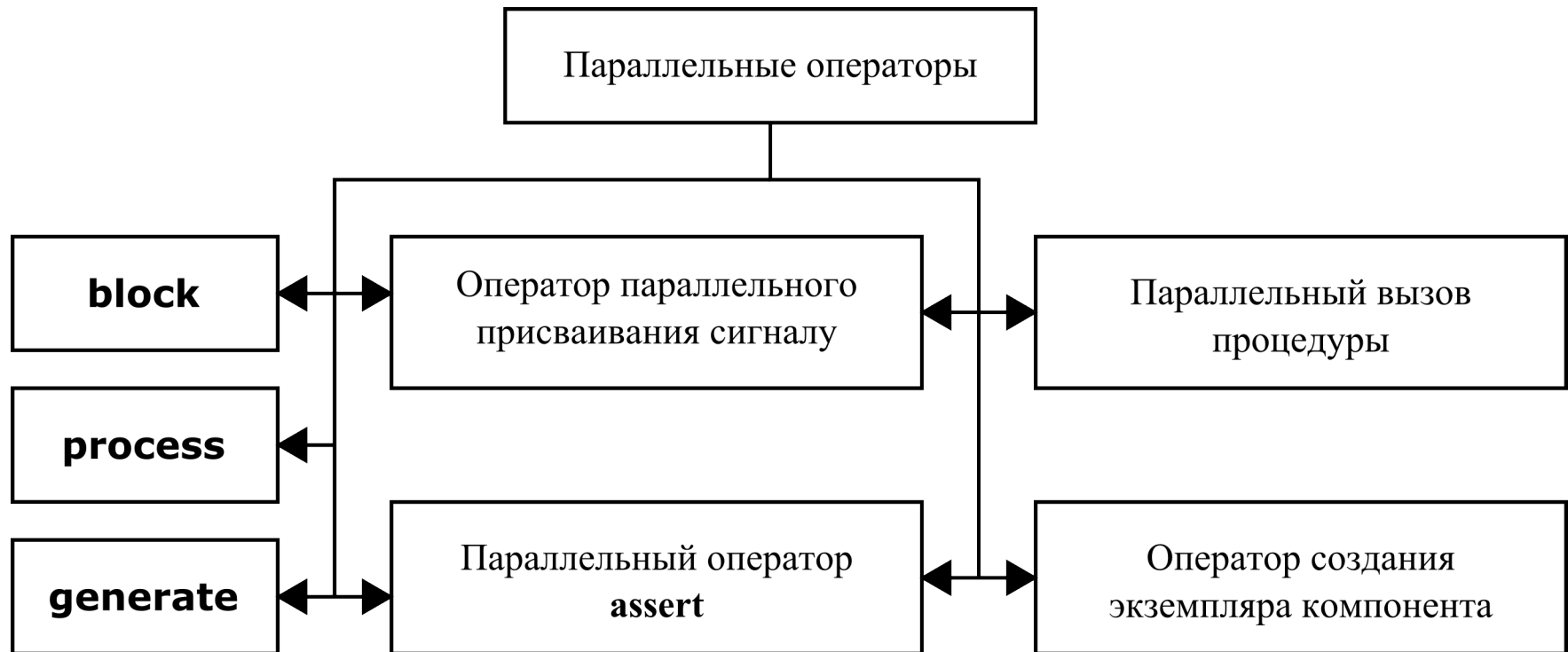
Параллельные операторы языка VHDL

Параллельные операторы выполняются параллельно, независимо от порядка их следования в VHDL модуле.

Свойства параллельных операторов:

- параллельные операторы выполняются при любом изменении сигналов, входящих в его список чувствительности ;
- результаты выполнения параллельных операторов (значения изменяемых сигналов) доступны только после завершения выполнения всех параллельных операторов, инициированных общим событием .

Параллельные операторы языка VHDL



Параллельный оператор присваивания значения сигналу

Различают следующие типы параллельного оператора присваивания значения сигналу :

- параллельный оператор безусловного присваивания ;
- параллельный оператор условного присваивания ;
- параллельный оператор селективного присваивания .

Параллельный оператор безусловного присваивания значения сигналу

Синтаксис параллельного оператора безусловного присваивания :

```
[метка:] имя_изменяемого_сигнала <= [модель_задержки]  
элемент1 [, элемент2 ...] ;
```

Элемент **модель_задержки** может принимать следующие значения :

```
transport или [reject выражение времени] inertial;
```

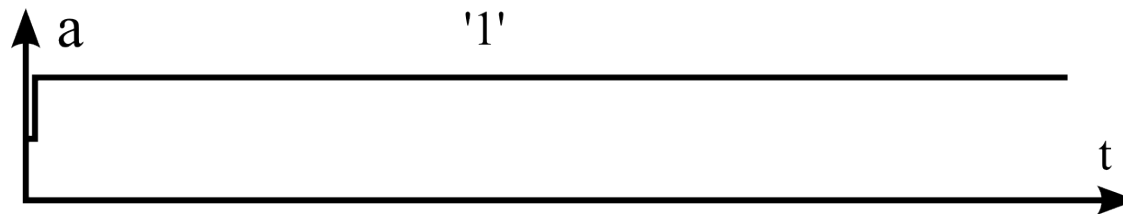
Элемент **элемент** имеет следующий вид :

```
значение [after выражение времени] ;
```

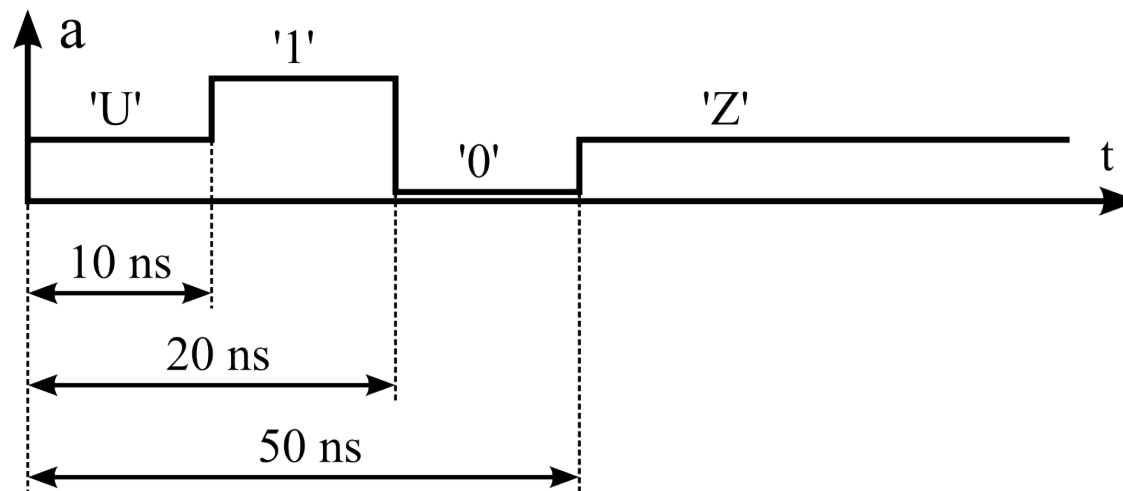
Параллельный оператор безусловного присваивания значения сигналу (примеры)

```
signal a : std_logic;
```

```
a <= '1';
```



```
a <= '0' after 10 ns, '1' after 20 ns, 'Z' after 50 ns;
```



Параллельный оператор условного присваивания значению сигналу

Синтаксис параллельного оператора безусловного присваивания :

```
[метка:] имя_изменяемого_сигнала <= [модель_задержки]  
элемент1 when условие else  
...  
[элементN when условие] ;
```

Пример :

```
q <= "0001" after 10 ns when sel = 0 else  
"0010" after 10 ns when sel = 1 else  
"0100" after 10 ns when sel = 2 else  
"1000" after 10 ns when others;
```

Параллельный оператор селективного присваивания значения сигналу

Синтаксис параллельного оператора селективного присваивания :

with выражение select

имя_изменяемого_сигнала <= [модель_задержки]

элемент1 when условие1 ,

. . .

[элементN when условиеN] ;

Пример :

with sel select

q <= "0001" after 10 ns when = 0 ,

"0010" after 10 ns when sel = 1 ,

"0100" after 10 ns when sel = 2 ,

"1000" after 10 ns when others;

Параллельный оператор создания экземпляра компонента

Оператор создает экземпляр компонента, представленного проектным модулем **entity**. Этот оператор выполняет подключение портов ввода/вывода и при необходимости установку значений настроечных констант.

Синтаксис :

метка_компоненты : имя_компоненты

[generic map (

[имя_параметра =>] значение_параметра ,

[имя_параметра =>] значение_параметра)]

[port map (

[имя_параметра =>] имя_сигнала ,

[имя_параметра =>] имя_сигнала)] ;

Параллельный оператор создания экземпляра компонента (пример)

Entity `and_logic` is

```
    port (  X1_in, X2_in      : in    std_logic;  
           Y_out, notY_out   : out   std_logic);  
end and_logic;
```

architecture `and_logic_arch` of `and_logic` is

```
    component and_gate  
        generic (  delay      : time);  
        port      (  X1,X2    : in   std_logic;  
                    Y         : out  std_logic);  
    end component;  
    signal Y_int      : std_logic;  
begin  
    and_0 : and_gate  
        generic map (  delay => 10 ns)  
        port map (X1=>X1_in ,X2=>X2_in,Y=>Y_int);  
    Y_out <= Y_int;  
    notY_out <= not Y_int;  
end and_logic_arch;
```


Параллельный оператор создания экземпляра компонента (пример)

```
Entity and_logic is
    port ( X1_in, X2_in : in    std_logic;
           Y_out       : out   std_logic);
end and_logic;

architecture and_logic_arch of and_logic is
    component and_gate
        generic ( delay : time);
        port ( X1,X2 : in  std_logic;
               Y,notY : out std_logic);
    end component;
    signal Y_int : std_logic;
begin
    and_0 : and_gate
        generic map (10 ns)
        port map (X1_in , X2_in, Y_out, open);

end and_logic_arch;
```

Параллельный оператор создания экземпляра компонента

Синтаксис с использованием ключевого слова *entity*:

```
метка_компоненты : entity имя_entity[(имя_архитектуры)]  
  [generic map (  
    [имя_параметра =>] значение_параметра , ....  
    [имя_параметра =>] значение_параметра )]  
  [port map (  
    [имя_параметра =>] имя_сигнала , ....  
    [имя_параметра =>] имя_сигнала)] ;
```

Параллельный оператор создания экземпляра компонента (пример)

```
Entity and_logic is
    port ( X1_in, X2_in      : in    std_logic;
           Y_out, notY_out   : out   std_logic);
end and_logic;

architecture and_logic_arch of and_logic is
    signal Y_int : std_logic;
begin
    and_0 : entity work.and_gate (and_gate_arch)
        generic map ( delay => 10 ns )
        port map ( X1=>X1_in ,X2=>X2_in,Y=>Y_int );
    Y_out <= Y_int;
    notY_out <= not Y_int;
end and_logic_arch;
```

Параллельный оператор assert

Оператор assert используется для выявления ошибок моделирования и сообщения о них пользователю через консоль программы моделирования.

Синтаксис :

```
[метка] : assert условие  
          [report строка_сообщения]  
          [severity выражение] ;
```

Пример :

```
assert a = '0'  
  report "Error"  
  severity failure ;
```

Параллельный оператор process

Параллельный оператор process предназначен для реализации одного независимого параллельного процесса, описываемого последовательными операторами.

Синтаксис :

```
[метка] : process [(список_чувствительности)] [is]
```

декларативная часть

```
begin
```

операторная часть

```
end process [метка];
```

Декларативная часть параллельного оператора process используется для объявления *констант, типов, подпрограмм, переменных* и т.д.

Операторная часть может состоять исключительно из последовательных операторов.

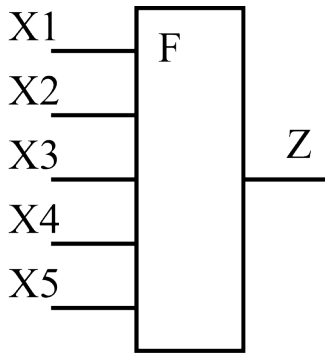
Параллельный оператор **process**

Свойства параллельного оператора **process** :

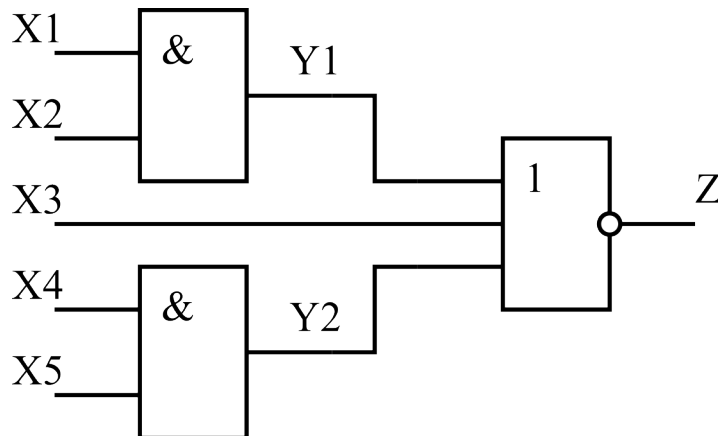
- Оператор **process** запускается при изменении любого сигнала, перечисленного в списке чувствительности. Если список чувствительности пуст, либо отсутствует, то процесс безусловно выполняется при начальном запуске, а также после завершения выполнения последнего оператора.
- Все последовательные операторы, входящие в **process**, выполняются друг за другом от начала до конца, за исключением случаев приостановки выполнения оператором **wait**.

Параллельный оператор **process**

Пример использования :



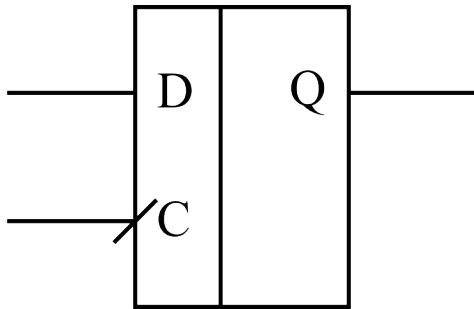
$$Z = \overline{X1 \cdot X2 + X3 + X4 \cdot X5}$$



```
and_logic : process (X1,X2,X3,X4,X5)
    variable Y1,Y2 : std_logic;
begin
    Y1 := X1 and X2;
    Y2 := X4 and X5;
    Z <= not (Y1 or Y2 or X3);
end process;
```

Параллельный оператор **process**

Пример использования :



```
and_logic : process (C)
begin
    if C'event and C = '1' then
        Q <= D;
    end if;
end process;
```


Параллельный оператор **block**

Параллельный оператор **block** предназначен для объединения нескольких параллельных операторов в один блок.

Возможности, предоставляемые оператором **block** :

- Структуризация текста описания, т.е. возможность ясного и наглядного выделения совокупности операторов, описывающих законченный функциональный узел.
- Возможность объявления в блоке локальных типов, сигналов, подпрограмм и т.д. и ограничение их области видимости.

Параллельный оператор **block**

Синтаксис :

[метка] : block [is]

заголовок блока

декларативная часть

begin

операторная часть

end block [метка];

[generic (...);

[generic map (...);]]

[port (...);

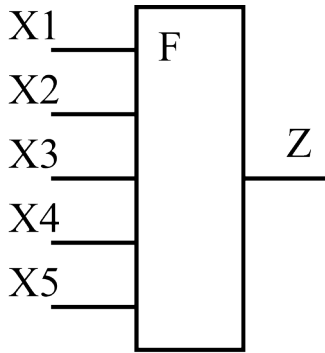
[port map (...);]]

Декларативная часть параллельного оператора **block** используется для объявления *констант, типов, подпрограмм, сигналов* и т.д.

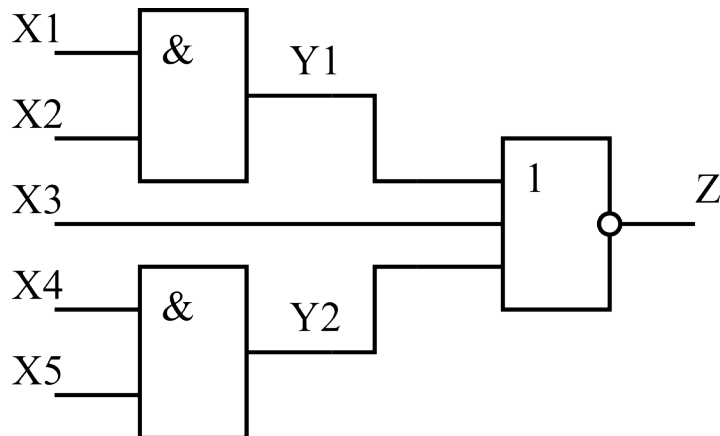
Операторная часть может включать параллельные операторы.

Параллельный оператор **block**

Пример использования :



$$Z = \overline{X1 \cdot X2 + X3 + X4 \cdot X5}$$



```
logic : block
```

```
    signal Y1, Y2 : std_logic ;
```

```
begin
```

```
    Y1 <= X1 and X2;
```

```
    Y2 <= X4 and X5;
```

```
    Z <= not (Y1 or Y2 or X3);
```

```
end block ;
```

Параллельный оператор **generate**

Параллельный оператор **generate** предназначен для детализации определенной части VHDL.

Он позволяет разработчику решать следующие задачи:

- Компактно описывать повторяющиеся части кода, содержащие параллельные операторы. При этом создается несколько блоков этих операторов и предоставляется возможность их настройки.
- Выбирать определенную реализацию на параллельных операторах из множества возможных. При этом используются условия, по результатам вычисления которых выбирается та или иная реализация части VHDL кода.

Операторы **generate** могут быть вложенными.

Параллельный оператор **generate**

Синтаксис :

```
метка : схема generate  
    [декларативная часть  
begin]  
    операторная часть  
end generate [метка];
```

Поле **схема** может иметь следующий формат :

for идентификатор **in** диапазон — схема цикла

Или

if условие — схема условия

Параллельный оператор **generate**

Поле *метка* является обязательной частью оператора *generate*. Она используется для формирования меток параллельных операторов находящихся в операторной части оператора *generate*.

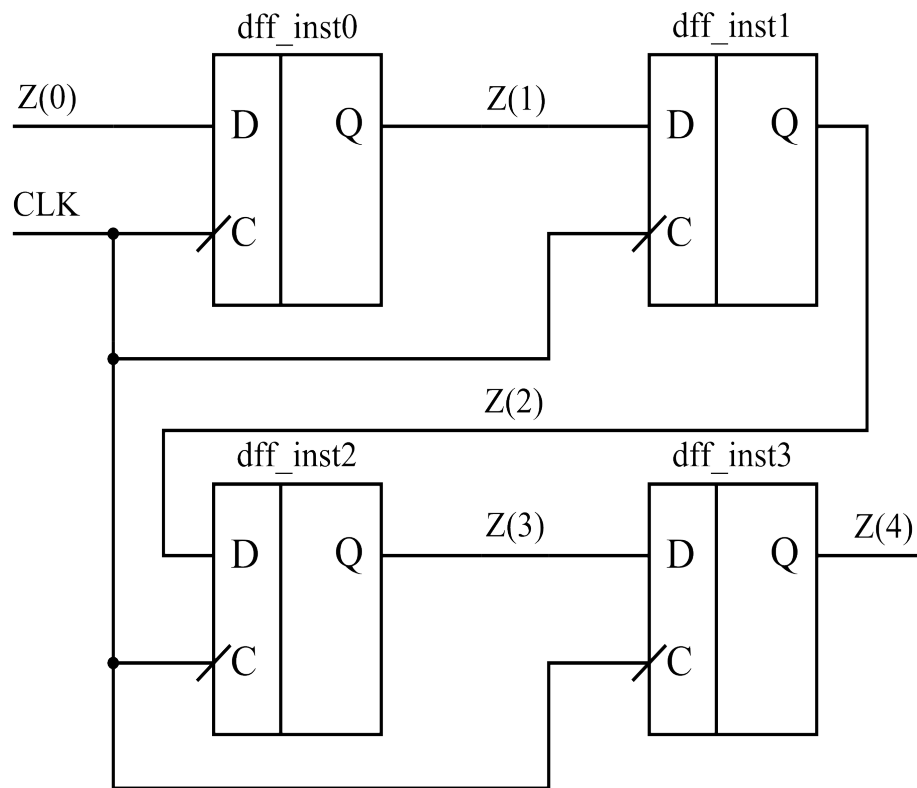
Операторная часть может содержать только параллельные операторы.

Соответствующие схемы оператора *generate* используются в следующих случаях:

- Схема цикла используется для описания повторяющихся участков VHDL кода.
- Схема условия позволяет выбирать один из вариантов реализации из множества возможных.

Параллельный оператор **generate**

Пример :

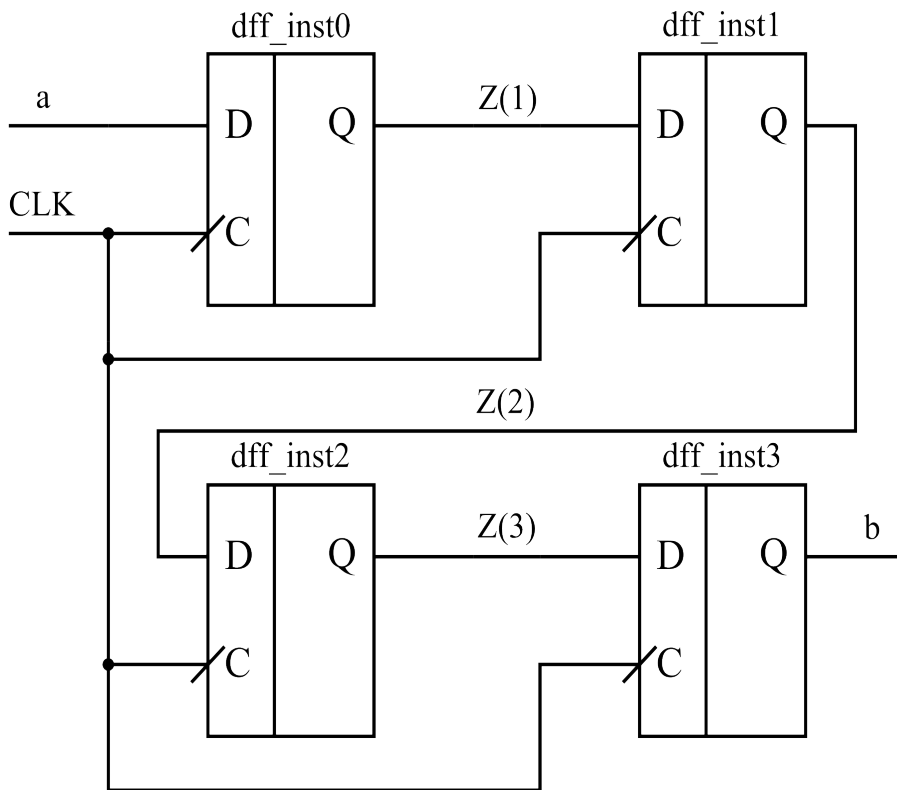


```
shift_reg : for i in 0 to 3 generate  
    dff_inst : dff port map (z(i), clk, z(i+1));  
end generate;
```

```
dff_inst0 : dff port map (z(0), clk, z(1));  
dff_inst1 : dff port map (z(1), clk, z(2));  
dff_inst2 : dff port map (z(2), clk, z(3));  
dff_inst3 : dff port map (z(3), clk, z(4));
```

Параллельный оператор **generate**

Пример :



```
shift_reg : for i in 0 to 3 generate
  if i = 0 generate
    dff_inst : dff port map (a, clk, z(0));
  end generate;
  if i = 3 generate
    dff_inst : dff port map (z(3), clk, b);
  end generate;
  if (i > 0) and (i < 3) generate
    dff_inst : dff port map (z(i), clk, z(i+1));
  end generate;
end generate;
```