Последовательные операторы языка VHDL

Последовательные операторы, в отличии от параллельных, выполняются последовательно друг за другом в порядке их записи в VHDL коде.

Последовательные операторы могут находится только внутри оператора *process* , внутри *функции* или *процедуры*.

Результат выполнения последовательных операторов недоступен прочим модулям до тех пор, пока не вызван оператор *wait* , либо пока не закончат выполнение все процессы, инициированные общим событием.

Все последовательные операторы, как и параллельные, могут иметь *метки*.

Последовательные операторы языка VHDL

if	Оператор условия
case	Оператор выбора
loop	Оператор цикла
<=	Последовательный оператор присваивания сигналу
:=	Последовательный оператор присваивания переменной
имя_подпрограммы	Оператор вызова подпрограммы
wait	Оператор ожидания
assert	Оператор проверки
report	Оператор отображения сообщения
next	Оператор перехода к следующей итерации цикла
exit	Оператор завершения цикла
return	Оператор выхода из подпрограммы
null	Пустой оператор

Последовательный оператор условия (IF)

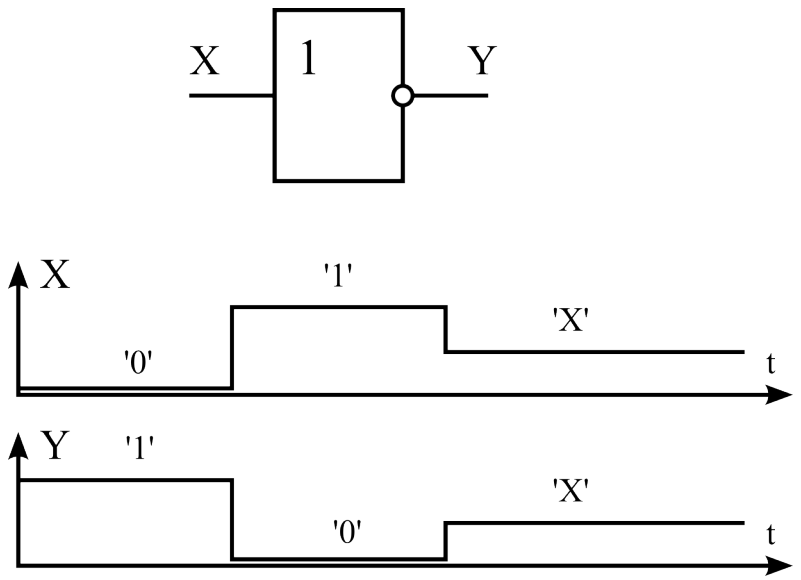
Оператор условия предоставляет возможность выполнять отличные наборы последовательных операторов, в зависимости от каких-либо условий, возникающих в описываемом устройстве в ходе его работы.

Синтаксис последовательного оператора условия :

```
[метка:]
   if условие then
      последовательные операторы
   [elsif условие then
      последовательные операторы]
   [else
      последовательные операторы]
   end if [метка];
```

Последовательный оператор условия (IF)

Пример использования :




```
not_logic :
    if X = '0' then
        Y <= '1';
    elsif X = '1'then
        Y <= '0';
    else
        Y <= 'X';
    end if not_logic;
```

Последовательный оператор выбора (CASE)

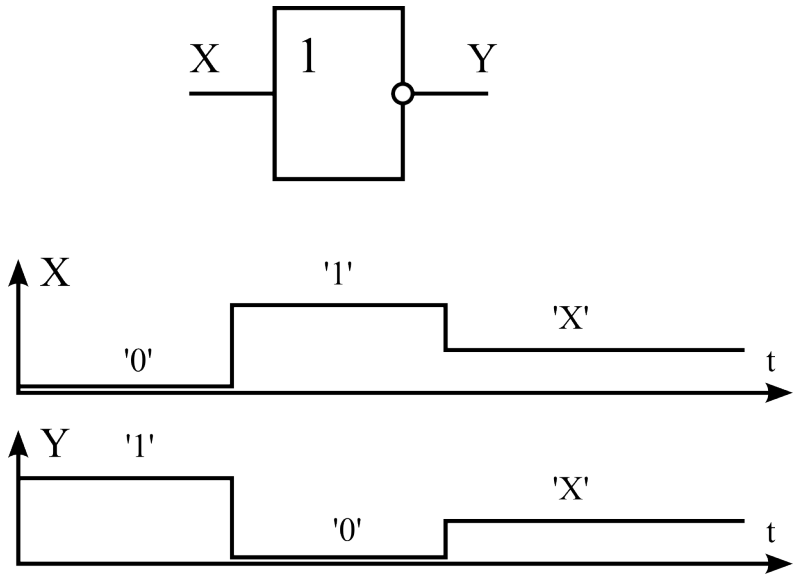
Оператор выбора, подобно оператору условия, позволяет выполнить разные наборы последовательных операторов по определенному условию.

Синтаксис последовательного оператора выбора :

```
[метка:]
   case выражение is
      case альтернатива =>
         последовательные операторы
      [case альтернатива =>
         последовательные операторы]
   end case [метка];
```

Последовательный оператор выбора (CASE)

Пример использования :




```
not_logic :
    case X is
        when '0' =>
            Y <= '1';
        when '1' =>
            Y <= '0';
        when others =>
            Y <= 'X';
    end case not_logic;
```

Последовательный оператор цикла (LOOP)

Оператор цикла позволяет выполнять один и тот же набор последовательных операторов необходимое количество раз.

Синтаксис оператора цикла :

```
[метка:]
    [итерационная_схема] loop
        последовательные операторы
    end loop [метка];
```

Поле итерационная_схема может быть двух типов :

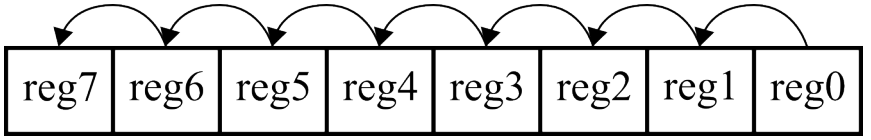
```
while условие
```

или

```
for имя_переменной in диапазон
```

Последовательный оператор цикла (LOOP)

Пример использования :




```
while_loop :
    while i < 7 loop
        reg(i+1) <= reg(i);
        i := i + 1;
    end loop while_loop;
```

```
for_loop :
    for i in 0 to 6 loop
        reg(i+1) <= reg(i);
    end loop for_loop;
```

Оператор изменения последовательности выполнения цикла (**next**)

Оператор **next** позволяет досрочно перейти к выполнению следующей итерации цикла.

Синтаксис оператора **next** :

```
[метка:] next [метка_цикла] [when условие];
```

Оператор **next** может применяться только внутри цикла, на который указывает поле **метка_цикла**. Если это поле пустое – оператор относится к циклу, внутри которого он непосредственно находится.

В случае наличия **условия**, оператор **next** выполняется, если условие возвращает **TRUE**.

Оператор изменения последовательности выполнения цикла (**exit**)

Оператор **exit** позволяет досрочно завершить цикл

Синтаксис оператора **exit** :

```
[метка:] exit [метка_цикла] [when условие];
```

Оператор **next** может применяться только внутри цикла, на который указывает поле **метка_цикла**. Если это поле пустое – оператор относится к циклу, внутри которого он непосредственно находится.

В случае наличия **условия**, оператор **exit** выполняется, если условие возвращает **TRUE**.

Примеры использования операторов изменения последовательности выполнения циклов (**next** и **exit**)

```
variable arr   : std_logic_vector(0 to 3,0 to 3);
variable j     : std_logic;
      ...            ...
ext: for i in 0 to 3 loop
   j := 0;
   int: while true loop
      next ext when i = j;
      exit when j = 2;
      arr(i,j) = '1';
      j := j + 1;
   end loop int;
end loop ext;
```

	0	1	2	3
0				
1	'1'			
2	'1'	'1'		
3	'1'	'1'	'1'	

Последовательный оператор присваивания значения переменно (:=)

Оператор **:=** присваивает новое значение переменной.

Синтаксис оператора **:=** :

```
[метка:] имя_переменной := значение;
```

Переменная и значение должны быть одного и того же типа

Пример :

```
var1 := '1';
arr := “0001”;
```

Пустой оператор (**null**)

Пустой оператор **null** означает отсутствие действия.

Синтаксис оператора **null** :

```
[метка:] null;
```

Чаще всего используется в операторах выбора и условия для указания неиспользуемых условий.

Пример :

```
case X is
    when '1' => Y <= '0';
    when '0' => Y <= '1';
    when others => null;
end case;
```

Оператор ожидаия (**wait**)

Оператор **wait** позволяет приостанавливать выполнение последовательных операторов в процессе. Дальнейшее продолжение выполнения операторов зависит от различных условий, задаваемых в самом операторе **wait**.

В случае использовании оператора **wait** список чувствительности оператора **process** должен отсутствовать.

Существует несколько вариантов оператора **wait** :

- **wait** ;
- **wait on** (параметры) ;
- **wait until** (параметры) ;
- **wait for** (параметры) .

Бесконечный останов оператором **wait**

Оператор **wait** без дополнительных параметров соответствует «бесконечному останову», то есть дальнейшее выполнение последовательных операторов прекращается до конца текущего сеанса моделирования.

Чаще всего используется при написании тестовых воздействий, для описания процесса начальной инициализации устройств.

Пример использования оператора **null** :

```
init_process : process
    begin
        en      <= '0';
        init    <= '1', '0' after 10 ns;
        wait    ;
    end process;
```

Останов до изменения сигнала при помощи оператора **wait**

Оператор **wait on** приостанавливает выполнение последовательных операторов до изменения сигналов, указанных в качестве параметра.

С помощью этого варианта оператора **wait** можно реализовать функционал списка чувствительности параллельного оператора **process**.

Синтаксис : [метка :] wait on список_чувствительности ;

```
not_logic : process
    begin
        Y  <= not X;
        wait on X;
    end process;
```

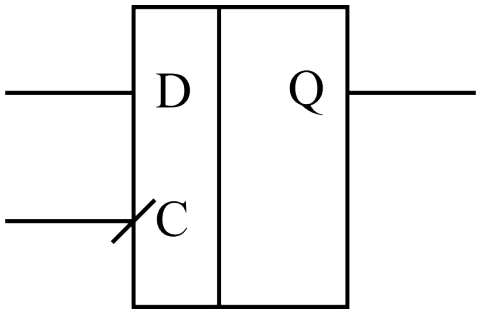
```
not_logic : process (X)
    begin
        Y  <= not X;
    end process;
```

Оператор **wait** с продолжением по условию

Оператор **wait until** приостанавливает выполнение последовательных операторов до наступления условия, заданного в качестве параметра.

Синтаксис : [метка :] wait until условие;

Пример :




```
DFF : process
    begin
        wait until C='1' and C'event;
        Q  <= D;
end process;
```

Останов на заданное время при помощи оператора **wait**

Оператор **wait for** приостанавливает выполнение последовательных операторов на время, указанное в качестве параметра.

Синтаксис : [метка :] wait for выражение времени ;

Пример :

```
generator : process
    begin
        clk  <= '0';
        while true loop
            wait for 10 ns;
            clk <= not clk;
        end loop;
    end process;
```

Комбинирование различных вариантов оператора **wait**

При использовании оператора **wait** можно комбинировать несколько вариантов в одну конструкцию. Это упрощает описание сложного поведения разрабатываемых устройств.

Пример :

```
wait on nmi, interrupt
    until ((nmi = true) or (interrupt = true))
    for 5 us;
```

Выполнение операторов приостанавливается до изменения сигналов **nmi** или **interrupt** и продолжится только в случае если значение одного из этих сигналов равно **true** или с момента их изменения прошло 5 мс.

Подпрограммы языка VHDL

Подпрограммы в языке VHDL подразделяются на два класса :

- процедуры ;
- функции .

Кроме этого, подпрограммы в языке VHDL могут быть :

- последовательными ;
- параллельными .

Параллельные подпрограммы определяются вне параллельного оператора **process** и других подпрограмм.

Последовательные подпрограммы определяются только внутри оператора **process** или других подпрограмм.

Подпрограммы языка VHDL

Подпрограммы содержат в своем теле набор последовательных операторов, которые задают совокупность действий, исполняемых после вызова этой подпрограммы.

Процедуры – не имеют возвращаемого значения и могут использоваться только в качестве отдельных операторов.

Функции – имеют одно возвращаемое значение и могут использоваться в выражениях.

Подпрограмм могут объявляться в декларативных частях следующих объектов языка VHDL:

- Entity
- Architecture
- Подпрограммы
- Process
- Package

Подпрограммы языка VHDL (объявления процедур и функций)

Синтаксис объявления процедуры :

```
procedure имя_процедуры [(список_формальных_параметров)];
```

Синтаксис объявления функции :

```
[pure | impure] function имя_функции
  [(список_формальных_параметров)]
  return тип_возвращаемого_значения;
```

Имя процедуры всегда является идентификатором.

Имя функции может быть либо идентификатором, либо оператором. Если имя функции является оператором – это означает что реализуется «перегрузка» заданного оператора.

Подпрограммы языка VHDL (список формальных параметров)

Список формальных параметров имеет следующий формат :

```
(параметр1 [,параметрN]);
```

Формальные параметры подпрограмм могут быть :

- константами ;
- переменными ;
- сигналами ;
- файлами .

Подпрограммы языка VHDL (объявления параметров)

Синтаксис объявления параметров :

```
signal имя_параметра1 [,имя_параметраN] : [режим]
  тип_параметра [:= статическое выражение])

variable имя_параметра [,имя_параметраN]  : [режим]
  тип_параметра [:= статическое выражение])

file имя_параметра [,имя_параметраN]  : тип_параметра

constant имя_параметра [,имя_параметраN]  : [in]
  тип_параметра [:= статическое выражение])
```

Подпрограммы языка VHDL (формальные параметры)

Формальные параметры процедуры могут использоваться в следующих режимах : **in**, **out**, **inout**.

Формальные параметры функции могут использоваться только в режиме **in**.

Свойства формальных параметров :

- Если режим параметра **in**, а класс не указан – предполагается что класс параметра – **constant**.
- Если режим параметра **inout** или **out**, а класс не указан – предполагается что класс параметра – **variable**.
- Если определено статическое выражение, то при вызове подпрограммы значение параметра может быть заменено ключевым словом **open**.
- Если не указан режим использования параметра – подразумевается режим **in**.

Подпрограммы языка VHDL (определение процедур)

Синтаксис определения функции:

```
[объявление_процедуры] is
        [декларативная часть]
  begin
        [операторная часть]
  end [имя_процедуры];
```

Декларативная часть используется для определения новых типов, переменных, констант и т.д.

Операторная часть содержит набор последовательных операторов.

Подпрограммы языка VHDL (процедуры)

Особенности :

- Если режим формального параметра **in**, а его класс – **variable**, то на время выполнения процедуры значение этого параметра не изменяется.
- Если режим формального параметра **in**, а его класс – **signal**, то значение этого параметра может изменяться в ходе выполнения процедуры.
- Процедура может содержать операторы **wait**, во время которых могут изменяться значения входных сигналов.
- Переменные, объявленные в декларативной части процедуры, создаются и инциализируются при каждом вызове процедуры.

Подпрограммы языка VHDL (пример процедуры)

Пример процедуры для записи байта данных по протоколу RS-232:

```
procedure send_uart_byte ( data : std_logic_vector) is
    constant baud_rate : time := 25 us;
begin
        for i in data'reverse_range loop
                wait for baud_rate;
                rxd <= data(i);
        end loop;
        wait for baud_rate * 10;
end procedure;
```

Подпрограммы языка VHDL (определение функций)

Синтаксис определения функции:

```
[объявление_функции] is
        [декларативная часть]
  begin
        [операторная часть]
  end [имя_функции];
```

Декларативная часть используется для определения новых типов, переменных, констант и т.д.

Операторная часть содержит набор последовательных операторов.

Подпрограммы языка VHDL (функции)

Особенности :

- Функция всегда должна возвращать значения с использованием оператора **return**.
- Функция объявления с идентификатором **pure** не может обращаться к внешним переменным и сигналам.
- Функция объявления с идентификатором **impure** может обращаться к внешним переменным и сигналам.
- По умолчанию функция рассматривается как **pure**.

Подпрограммы языка VHDL (пример функции)

Присер функции, определяющей наличие неопределенных значений в массиве типа **std_logic_vector**.

```
function Is_X(s : std_logic_vector) return boolean is
  begin
      for i in s'range loop
          case s(i) is
              when 'U' | 'X' | 'Z' | 'W' | '-' => return true;
              when others => null;
          end case;
      end loop;
      return false;
  end;
```