## Lexical elements

## **Declarations**

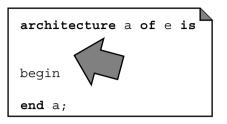
Reserved words

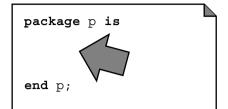
Type declaration
Subtype declaration
Constant declaration
Signal declaration
Variable declaration

page 1 page 3

#### Type declaration

#### Reserved words





```
in the STD.STANDARD package:

type boolean is (false, true);
type bit is ('0', '1');
type character is (NUL, SOH, <...> '}', '~', DEL);
type string is array(positive range <>) of character;
type bit_vector is array(natural range <>) of bit;
```

```
in the IEEE.NUMERIC_STD package:
    type unsigned is array(natural range <>) of std_Logic;
    type signed is array(natural range <>) of std logic;
```

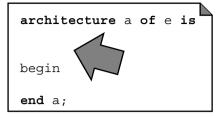
abs access after alias all and architecture arrav assert attribute begin block bodv buffer bus case component configuration constant disconnect downto else elsif end entity exit file for function generate generic group quarded

if impure in inertial inout is label library linkage literal 1000 map mod nand new next nor not ทนไไ of on open or others out package port procedure process pure

range record register reject rem report return rol ror select severity shared signal sla sll sra srl subtype then t.o transport type unaffected units until use variable wait when while with xnor xor

#### Subtype declaration

#### Signal declaration



```
package p is
end p;
```

```
end a;
      signal s1, s2, s3: std ulogic;
      signal sig1: std ulogic;
     signal sig2: std uloqic;
     signal sig3: std ulogic;
                   signal logic out: std uLogic;
                   signal open drain out: std logic;
                   signal tri state out: std logic;
    signal counter: unsigned(nb bits-1 downto 0);
    signal double: unsigned(2*nb bits-1 downto 0);
    signal sine: signed(nb bits-1 downto 0);
  signal clock internal: std ulogic := '1'
```

architecture a of e is

begin

```
in the STD.STANDARD package:
```

```
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;
```

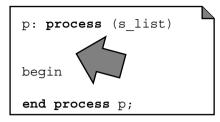
```
in the IEEE.STD_LOGIC_1164 package:
subtype std_logic is resolved std_uLogic;
subtype X01 is resolved std_uLogic range 'X' to '1';
subtype X01Z is resolved std_uLogic range 'X' to 'Z';
subtype UX01 is resolved std_uLogic range 'U' to '1';
subtype UX01Z is resolved std_uLogic range 'U' to 'Z';
```

```
subtype byte is std_uLogic_vector(7 downto 0);
subtype word is std_uLogic_vector(15 downto 0);
subtype long_word is std_uLogic_vector(31 downto 0);
```

```
subtype BCD_digit is unsigned(3 downto 0);
subtype my_counter_type is unsigned(9 downto 0);
subtype sine_wave_type is signed(15 downto 0);
```

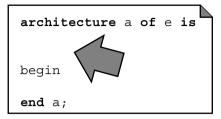
#### Variable declaration

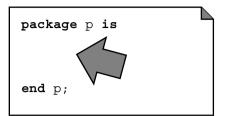
#### Constant declaration



```
variable v1, v2, v3: std_ulogic;
variable var1: std_ulogic;
variable var2: std_ulogic;
variable var3: std_ulogic;
```

```
variable counter: unsigned(nb_bits-1 downto 0);
variable double: unsigned(2*nb_bits-1 downto 0);
variable sine: signed(nb_bits-1 downto 0);
```



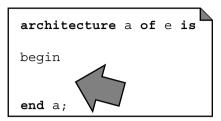


```
constant bit nb: positive := 4;
   constant min value: positive := 0;
   constant max_value: positive := 2**bit nb - 1;
constant bit nb: positive := 4;
constant patt1: unsigned(bit nb-1 downto 0) := "0101";
constant patt2: unsigned(bit nb-1 downto 0) := "1010";
   constant address nb: positive := 4;
   constant data register address
                                      : positive := 0;
   constant control register address : positive := 1;
   constant interrupt register address: positive := 2;
   constant status_register_address
                                      : positive := 3;
       constant clock period: time := 5 ns;
       constant access time: time := 2 us;
       constant duty cycle: time := 33.3 ms;
       constant reaction time: time := 4 sec;
       constant teaching period: time := 45 min;
```

#### Process statement

# Concurrent statements

Signal assignment Process statement When statement With statement



```
mux: process(sel, x0, x1)
         begin
           if sel = '0' then
             y \ll x0;
           elsif sel = '1' then
             y \ll x1;
           else
             y <= 'X';
           end if;
         end process mux;
count: process(reset, clock)
begin
  if reset = '1' then
    counter <= (others => '0');
  elsif rising edge(clock) then
    counter <= counter + 1;</pre>
  end if;
end process count;
```

page 9 page 11

#### When statement

#### Signal assignment

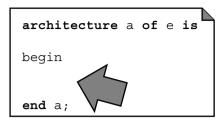
```
architecture a of e is
begin
end a;
```

```
y <= x0 when sel = '0' else
     x1 when sel = '1' else
     'X';
  y <= x0 after 2 ns when sel = '0' else
        x1 after 3 ns when sel = '1';
```

```
architecture a of e is
begin
end a;
```

```
y1 \ll a;
      y2 <= a and b;
      y3 <= to integer(a);
 y <= "0000011";
 y <= "0000" & "0011";
 y \leftarrow ('0', '0', '0', '0', '0', '0', '1', '1');
 y \le (7 \text{ downto } 2 = \ '0', \ 1 \mid 0 = \ '1');
 y <= (7 downto 2 => '0', others => '1');
           y1 \ll a;
           y2 <= a after 2 ns;
           y3 <= inertial a after 1 ns;
           y4 <= transport a after 4 ns;
           y5 <= reject 1 ns inertial a after 5 ns;
                                  y <= '0',
y <= a and b after 5 ns;
                                        '1' after 2 ns,
                                              cer 10 ns,
                                        '1' arcer 15 ns,
                                        '-' after 23 ns;
```

#### With statement



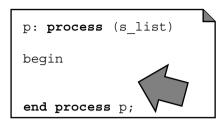
```
mux: with sel select
         y <= x0 when "00",
              x1 when "01",
              x2 when "10",
              x3 when "11",
              'X' when others;
decoder: with binary code select
 y <= transport "0001" after 2 ns when "00",
                 "0010" arc 5 ns when "01",
                         ce: 3 ns when "10",
                 "1000" after 4 ns when "11",
                 "XXXX" when others;
```

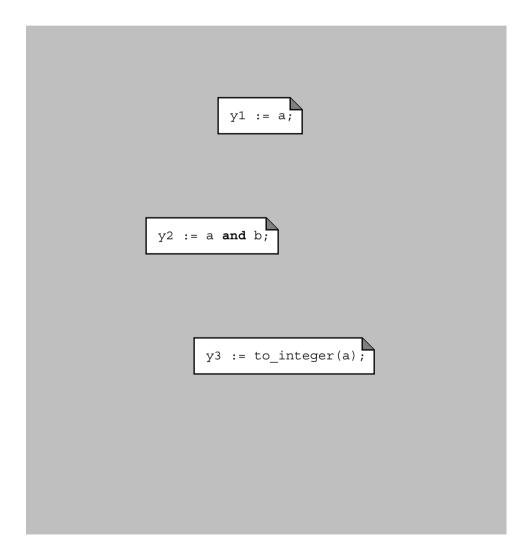
## Sequential statements

Variable assignment
If statement
Case statement
Loop statement

page 13 page 15

#### Variable assignment





page 16 page 14

If statement Loop statement

```
p: process (s_list)
begin
end process p;
```

```
if gate = '1' then
    q \ll d;
                                     if sel = '0' then
   end if;
                                       v1 <= x0:
                                      y2 <= x1;
                                       y3 <= '0';
                                     else
                                       y1 <= x1;
         if sel = '0' then
                                       y2 <= x0;
                                      y3 <= '1';
           y \ll x0;
         else
                                     end if;
           y \ll x1;
         end if;
if sel = 0 then
y \ll x0;
elsif sel = 1 then
y \ll x1;
elsif sel = 2 then
y \ll x2;
                       if (a = '0') and (b = '0') then
else
 y \ll x3;
                         y <= '1';
end if;
                       else
                        y <= '0';
                       end if;
```

```
p: process (s_list)
begin
end process p;
```

```
for xIndex in 1 to xSize loop
        for yIndex in 1 to ySize loop
          if xIndex = yIndex then
            y(xIndex, yIndex) <= '1';
          else
            y(xIndex, yIndex) <= '0';
          end if:
        end loop;
      end loop;
multipl: for indexB in 0 to nBits-1 loop
 partialProd: for indexA in nBits-1 downto 0 loop
    partProd(indexA) <= a(indexA) and b(indexB);</pre>
  end loop partialProd;
 cumSum(indexB+1) <= cumSum(indexB) + partProd;</pre>
end loop multipl;
```

page 17 page 19

#### Case statement

```
p: process (s_list)
begin
end process p;
```

```
case sel is
 when "00" => y <= x0;
                             case opCode is
 when "01" => y <= x1;
                               when add \Rightarrow y1 \iff x0;
 when "10" => y <= x2;
                                           y2 <= x1;
 when "11" => y <= x3;
                               when sub \Rightarrow y1 \iff x1;
 when others => null;
                                           y2 <= x0;
end case;
                               when others => null;
                             end case;
    case value is
      when 1
               => nBits <= 1;
      when 2 | 3 => nBits <= 2;</pre>
      when 4 to 7 => nBits <= 3;
      when 8 to 15 => nBits <= 3;</pre>
      when others => nBits <= 0;</pre>
    end case;
               case to integer(sel) is
                 when 0 \Rightarrow y \ll x0 after 1 ns;
                 when 1 => y <= x1 arr 1 ns;
                 when 3 => y <= x3 after 1 ns;
                 when others => y <= 'X';
               end case;
```

page 20 page 18

## **Operators**

Logic operators
Arithmetic operators
Comparisons
Concatenation

#### Arithmetic operators

operator	description
+	addition
-	substraction
*	multiplication
/	division
**	power
abs	absolute value
mod	modulo
rem	reminder of the division
sla	arithmetic shift left
sra	arithmetic shift right

```
maxUnsigned <= 2**nBits - 1;
maxSigned <= 2**(nBits-1) - 1;</pre>
```

page 21 page 23

#### Comparisons

operator	description
=	equal to
/=	not equal to
<	smaller than
>	greater than
<=	smaller than or equal to
>=	greater than or equal to

```
if counter > 0 then
  counter <= counter -1;
end if;</pre>
```

```
if counter /= 0 then
  counterRunning <= '1';
else
  counterRunning <= '0';
end if;</pre>
```

#### Logic operators

operator	description
not	inversion
and	logical AND
or	logical OR
xor	exclusive-OR
nand	NAND-function
nor	NOR-function
xnor	exclusive-NOR
sll	logical shift left
srl	logical shift right
rol	rotate left
ror	rotate right

```
if (a = '1') and (b = '1') then
    y <= '1';
else
    y <= '0';
end if;

if (a and b) = '1' then
    y <= '1';
else
    y <= '0';
end if;</pre>
count <= count sll 3;
```

page 24 page 22

#### Concatenation

operator	description
&	concatenation

```
address <= "1111" & "1100";
```

```
constant CLR: std logic vector(1 to 4) := "0000";
constant ADD: std logic vector(1 to 4) := "0001";
constant CMP: std logic vector(1 to 4) := "0010";
constant BRZ: std logic vector(1 to 4) := "0011";
constant R0 : std logic vector(1 to 2) := "00";
constant DC : std logic vector(1 to 2) := "--";
constant reg : std logic := '0';
constant imm : std logic := '1';
type ROMArrayType is array(1 to 255)
        of std logic vector(1 to 9);
constant ROMArray: ROMArrayType := (
  0 = > (CLR \& DC \& R0 \& req),
  1 = > (ADD &"01"& R0 & imm),
  2 = (CMP \&"11"\& R0 \& imm),
  3 = (BRZ \& "0001" \& '-'),
  4 to romArray'length-1 => (others => '0') );
```

## **Attributes**

Type related attributes
Array related attributes

page 25 page 27

#### Type related attributes

attribute	result
T'base	the base type of T
T'left	the left bound of T
T'right	the right bound of T
T'high	the upper bound of T
T'low	the lower bound of T
T'pos(X)	the position number of X in T
T'val(N)	the value of position number N in T
T'succ(X)	the successor of X in T
T'pred(X)	the predecessor of X in T
T'leftOf(X)	the element left of X in T
T'rightOf(X)	the element right of X in T

```
signal counterInt: unsigned;
signal count1: unsigned(counter'range);
signal count2: unsigned(counter'length-1 downto 0);
...

flip: process(count1)
begin
   for index in count1'low to count1'high loop
        count2(index) <= count1(count1'length-index);
   end loop;
end process flip;</pre>
```

page 28 page 26

#### Array related attributes

attribute	result
A'left	the left bound of A
A'right	the right bound of A
A'high	the upper bound of A
A'low	the lower bound of A
A'range	the range of A
A'reverse_range	the range of A in reverse order
A'length	the size of the range of A

```
type stateType is (reset, wait, go);
signal state: stateType;
...

evalNextState: process(reset, clock)
begin
   if reset = '1' then
       state <= stateType'left;
   elsif rising_edge(clock) then
       ...
   end if;
end process evalNextState;</pre>
```

#### Wait statement

```
p: process
begin
end process p;
```

```
test: process
begin
  testMode <= '0';
  dataByte <= "11001111";
  startSend <= '1';
  wait for 4*clockPeriod;
  startSend <= '0';
  wait for 8*clockPeriod;
  testMode <= '1';
  dataByte <= "11111100";
  startSend <= '1';
  wait for 4*clockPeriod;
  startSend <= '0';
  wait;
end process test;</pre>
```

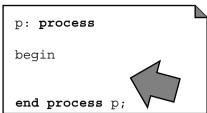
```
test: process
begin
  a <= '0';
  b <= '0';
  wait for simulStep;
  error <= y xor '0';

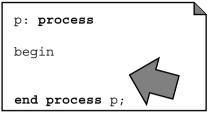
a <= '1';
  b <= '1';
  wait for simulStep;
  error <= y xor '1';

end process test;</pre>
```

```
test: process
begin
  playVectors: for index in stimuli'range
  dataByte <= stimuli(index);
  wait for clockPeriod;
  assert codedWord = expected(index);
  wait for clockPeriod;
  end loop playVectors;
  wait;
end process test;</pre>
```

#### Assert statement







## Simulation elements



```
assert output = '1';
      assert output = '1'
        report "output was '0'!";
                     assert output = '1'
                       report "output was '0'!"
                       severity error;
    in the STD.STANDARD package:
    type severity level is (note,
                             warning,
                             error,
                             failure);
```

Wait statement Assert statement

page 32 page 30

## Index

Lexical elements	1
Declarations	3
Concurrent statements	9
Sequential statements	15
Operators	21
Attributes	27
Simulation elements	29

Arithmetic operators	23
Array related attributes	29
Assert statement	31
Case statement	18
Comparisons	24
Concatenation	25
Constant declaration	6
If statement	17
Logic operators	22
Loop statement	19
Process statement	11
Reserved words	2
Type declaration	4
Type related attributes	28
Signal assignment	10
Signal declaration	7
Subtype declaration	5
Variable assignment	16
Variable declaration	8
Wait statement	30
When statement	12
With statement	13
	_

# VHDL syntax shortform