Lexical elements

Reserved words

Reserved words

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

```
if
impure
in
inertial
inout
is
label
library
linkage
literal
loop
map
mod
nand
new
next
nor
not
null
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
to
transport
type
unaffected
units
until
use
variable
wait
when
while
with
xnor
xor
```

Declarations

Type declaration
Subtype declaration
Constant declaration
Signal declaration
Variable declaration

Type declaration

```
architecture a of e is


begin

end a;
```

```
package p is


end p;
```

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.STD_LOGIC_1164 package:

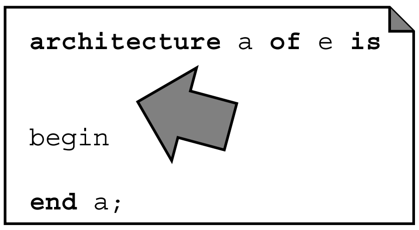
```
type std_uLogic is ('U', 'X', '0', '1', 'Z',
                    'W', 'L', 'H', '-');

type std_uLogic_vector is
       array(natural range <>) of std_uLogic;
```

in the IEEE.NUMERIC_STD package:

```
type unsigned is array(natural range <>) of std_Logic;
type signed is array(natural range <>) of std_logic;
```

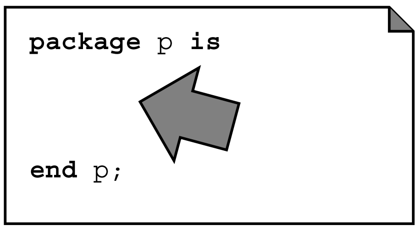
Subtype declaration



```
architecture a of e is

begin

end a;
```



```
package p is


end p;
```

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:

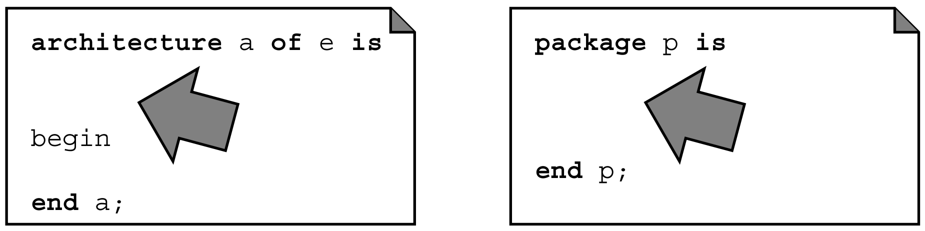
```
subtype std_logic is resolved std_uLogic;

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);
```

Constant declaration



```
architecture a of e is

begin

end a;
```

```
package p is

end p;
```

```
constant bit_nb: positive := 4;
constant min_value: positive := 1;
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     : natural:= 0;
constant control_register_address  : natural:= 1;
constant interrupt_register_address: natural:= 2;
constant status_register_address   : natural:= 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;
```

Signal declaration

```
architecture a of e is

begin

end a;
```

```
signal s1, s2, s3: std_ulogic;
signal sig1: std_ulogic;
signal sig2: std_ulogic;
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';
```

Variable declaration

```
p: process (s_list)

begin

end process p;
```

```
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);
```

Concurrent statements

Signal assignment
Process statement
When statement
With statement

Signal assignment

```
architecture a of e is

begin


end a;
```

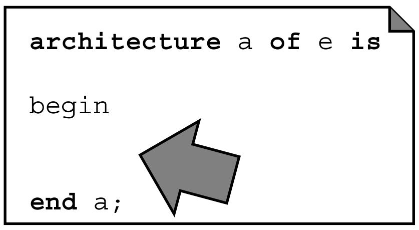
```
y1 <= a;
y2 <= a and b;
y3 <= to_integer(a);
```

```
y1 <= 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 <= a and b after 5 ns;
```

```
y <= '0',
     '1' after 2 ns,
     '0' after 4 ns,
     'X' after 10 ns,
     '1' after 15 ns,
     '-' after 23 ns;
```

Process statement



```
architecture a of e is

begin


end a;
```

```
mux: process(sel, x0, x1)
begin
  if sel = '0' then
    y <= x0;
  elsif sel = '1' then
    y <= 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;
  end if;
end process count;
```

When statement

```
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';
```

With statement

```
architecture a of e is

begin


end a;
```

```
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" after 5 ns when "01",
                 "0100" after 3 ns when "10",
                 "1000" after 4 ns when "11",
                 "XXXX" when others;
```

Sequential statements

Variable assignment
If statement
Case statement
Loop statement

Variable assignment

```
p: process (s_list)

begin


end process p;
```

```
y1 := a;
```

```
y2 := a and b;
```

```
y3 := to_integer(a);
```

If statement

```
p: process (s_list)

begin


end process p;
```

```
if gate = '1' then
  q <= d;
end if;
```

```
if sel = '0' then
  y1 <= x0;
  y2 <= x1;
  y3 <= '0';
else
  y1 <= x1;
  y2 <= x0;
  y3 <= '1';
end if;
```

```
if sel = '0' then
  y <= x0;
else
  y <= x1;
end if;
```

```
if sel = 0 then
  y <= x0;
elsif sel = 1 then
  y <= x1;
elsif sel = 2 then
  y <= x2;
else
  y <= x3;
end if;
```

Case statement

```
p: process (s_list)

begin


end process p;
```

```
case sel is
  when "00" => y <= x0;
  when "01" => y <= x1;
  when "10" => y <= x2;
  when "11" => y <= x3;
  when others => null;
end case;
```

```
case opCode is
  when add => y1 <= x0;
              y2 <= x1;
  when sub => y1 <= x1;
              y2 <= x0;
  when others => null;
end case;
```

```
case value is
  when 1       => nBits <= 1;
  when 2|3     => nBits <= 2;
  when 4 to 7  => nBits <= 3;
  when 8 to 15 => nBits <= 4;
  when others  => nBits <= 0;
end case;
```

```
case to_integer(sel) is
  when 0 => y <= x0 after 1 ns;
  when 1 => y <= x1 after 1 ns;
  when 2 => y <= x2 after 1 ns;
  when 3 => y <= x3 after 1 ns;
  when others => y <= 'X';
end case;
```

Loop statement

```
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);
  end loop partialProd;
  cumSum(indexB+1) <= cumSum(indexB) + partProd;
end loop multipl;
```

Operators

Logic operators
Arithmetic operators
Comparisons
Concatenation

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

```
y <= a and b;
```

```
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;
```

```
count <= count sll 3;
```

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;
```

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;
```

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

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(0 to 255)
        of std_logic_vector(1 to 9);

constant ROMArray: ROMArrayType := (
  0 => ( CLR & DC & R0 & reg ),
  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

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;
```

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;
```

🚫 Simulation elements 🚫

Wait statement
Assert statement

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;
```

```
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;
```

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

Assert statement

```
p: process

begin


end process p;
```

```
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);
```

Index

Lexical elements	1
Declarations	3
Concurrent statements	9
Sequential statements	15
Operators	21
Attributes	27
Simulation elements	30

Arithmetic operators	23
Array related attributes	29
Assert statement	32
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	31
When statement	12
With statement	13