

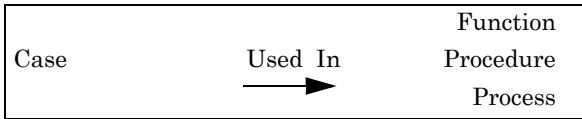


# **The VHDL Reference Guide**

**Version 4.2**

# Case Statement

The case statement evaluates an expression and selects one of a number of sequences of statements depending on the expression's value. A case statement is a **sequential statement**:



## Syntax

```
[label]:  
case expression is  
  when choice =>  
    -- sequential statements  
  when choice =>  
    -- sequential statements  
  when choice | choice =>  
    -- sequential statements  
  when choice to choice =>  
    -- sequential statements  
end case;
```

See LRM section 8.8. The label is optional.

## Rules and Examples

All possible values of the case expression must be included in one (and only one) of the choice branches. The **others** clause can be used as the final choice to cover all expression values not explicitly listed.

```
case SEL is  
  when "01" => Z <= A;  
  when "10" => Z <= B;  
  when others => Z <= 'X';  
end case;
```

A range or a selection may be specified as a choice:

```
case INT_A is  
  when 0 => Z <= A;  
  when 1 to 3 => Z <= B;  
  when 4|6|8 => Z <= C;  
  when others => Z <= 'X';  
end case;
```

# Case Statement

---

Choices may not overlap:

```
case INT_A is
  when 0      => Z <= A;
  when 1 to 3 => Z <= B;
  when 2|6|8  => Z <= C;      -- ERROR
  when others => Z <= 'X';
end case;
```

A range may not be used with a vector type:

```
case VEC is
  when "000" to "010" => -- ERROR
    Z <= A;
  when "111"  => Z <= B;
  when others => Z <= 'X';
end case;
```

See also the **null** statement.

## Synthesis Issues

The case statement is generally synthesizable, typically implemented with separate multiplexer logic for each output.

With multiple targets and embedded `if` statements, the case statement may be used to synthesize a general mapping function (e.g., next state and output generation for a finite state machine). For example:

```
case READ_CPU_STATE is
  when WAITING =>
    if CPU_DATA_VALID = '1' then
      CPU_DATA_READ <= '1';
      READ_CPU_STATE <= DATA1;
    end if;
  when DATA1 =>
    -- etc.
end case;
```

## Compatibility Issues in VHDL'87

In VHDL-87, a case statement label is not allowed.

# Article A: Expressions

---

An expression is a formula for computing a value. It consists of **operators** and operands (literals, attributes and objects).

Expressions can be as simple as a single literal, or as complex as required:

```
3
OP + 1;
((128-DECAY)*ENV + DECAY*abs(FILT))/128;
```

Expressions can be used in many places in VHDL code. There may be restrictions on the type of an expression depending where it is used.

## Declaration Expressions

Expressions are used in a **constant** declaration to define the value, for assigning **initial values** in the declaration of signals and variables and for assigning **default values** in the declaration of ports, generics or subprogram parameters.

```
constant ROWS : natural := 2**LINE_WIDTH;
signal CLK : std_logic := '1';
component PARITY
  generic (N :integer := 8);
  ...
procedure DLEN (signal RES : out integer;
                LEN : in natural := 16) ...
```

Note: initial values are applied at the beginning of simulation and persist until the object is assigned a new value. Default values apply only if the object is otherwise left unassigned, i.e. not mapped in a port or generic map, or a subprogram call.

## Assignment Expressions

Expressions are used to calculate values for assignment to signals and variables.

```
-- assignment expressions
OP <= A and not(B or C);
TEMP := not(TEMP_IN(7 downto 1)) + 1;
```

Expressions can also be used to map values to ports and generics in component instantiations or configurations and to map values to subprogram parameters in subprogram calls.

# A: Expressions

---

```
-- expression in procedure call
PSIG_AB <= PARITY(SIG_A & SIG_B);

U1 : PARITY    -- component instantiation
      generic map (N => DWORD'length)
      port map (A => unsigned(DWORD),
                ODD => PWORD);
```

Assignment expressions can also be used with the **return** statement of a function declaration to calculate a function result.

## Simple Expressions

A simple expression is one which uses only arithmetic operators, together with the concatenation and logical not operators. Simple expressions are used in the following constructs:

Aggregate *item* (see example), case choice, generate and for loop iteration ranges

```
OP <= (OP'high downto OP'high-2 => '1',
      others => '0');
```

## Expression Elements

Expressions can be used in the following locations which are elements of a larger expression:

Aggregate value (see example), array index, qualified expression, type conversion target

```
OP <= (6 => (A and B), 5 => (A or B),
      others => not(C));
BYTE <= BYTE(BTYE'high-1 downto 0) & '0';
```

## Boolean Expressions

Expressions returning a boolean value are required for conditional clauses in the following statements:

assertion, exit, if, if generate, next, wait until, while

Remember relational operators return boolean values and are usually part of boolean expressions:

```
if (A and B and C) = '1' then
    ...
exit L1 when DATA(I-1) = '1';
```

# A: Expressions

---

## String Expressions

Expressions returning a string value are required for clauses in the following statements:

Assertion report, report, attribute 'value, file declaration

Remember, the concatenation operator can join together strings and characters

```
report "Bad value in" &
      ADDR'path_name;
file VEC_FILE : text open write_mode
      is "../output/" & FILENAME;
```

## Time Expressions

Expressions returning a time value are required for clauses in the following statements:

attributes 'delayed, 'stable and 'quiet, wait for

## Expressions in Case Statements

Expressions can also be used in case select clauses.

```
signal AI, BI, OPI : integer;
...
case AI + BI is      -- select
  when 64 => ...
```

For **array** types, the select expression must have a **locally static subtype**. An example of a non-static subtype is obtained when using the concatenation operator - the result is an unconstrained array:-

```
signal A, B: unsigned(3 downto 0);
...
case A & B is      -- ERROR non-static subtype
```

The work-around for this issue is to either use a temporary variable to store the concatenation, or to constrain the result with a qualified expression:-

```
subtype S8 is unsigned(7 downto 0);
signal A, B: unsigned(3 downto 0);
...
case S8'(A & B) is  -- OK
```

Case choice expressions must also be **locally static**.

# A: Expressions

---

## Locally and Globally Static Expressions

A dynamic expression is one which can only be evaluated during simulation, e.g. is dependent on variable or signal objects.

A static expression is one which can be evaluated during the compilation or elaboration of a VHDL design.

A locally static expression can be evaluated during the compilation of the design unit which contains the expression, i.e. the expression only depends on literals and locally declared constants.

A globally static expression can only be evaluated during the elaboration phase of compilation, when the design hierarchy is linked together. Globally static expressions may depend on constants declared in packages or generics.

Locally static restrictions apply to case statement expressions. For case statements, if the case select expression is of an **array type**, then the type must be locally static.

Case choice expressions must always be locally static.

```
entity NLS is
  generic (N : integer);
  port (AGEN : in
        std_logic_vector(N downto 0));
end NLS;
architecture A of NLS is
  constant TWON : integer := 2*N;
  constant FIVE : integer := 5;
  ...
  -- syntax:
  -- case <select expression> is
  --   when <choice expression> =>

case AGEN is          -- ERROR input AGEN not
  when ...           -- locally static
  ...
case AINT is
  when N =>          -- ERROR generic N not
    ...            -- locally static
  when TWON =>      -- ERROR constant TWON
    ...            -- not locally static
  when FIVE =>      -- OK
    ...
```