4.5 GENERIC

As the name suggests, GENERIC is a way of specifying a *generic* parameter (that is, a static parameter that can be easily modified and adapted to different applications). The purpose is to confer the code more flexibility and reusability.

A GENERIC statement, when employed, must be declared in the ENTITY. The specified parameter will then be truly global (that is, visible to the whole design, including the ENTITY itself). Its syntax is shown below.
Example: The GENERIC statement below specifies a parameter called n, of type INTEGER, whose default value is 8. Therefore, whenever n is found in the ENTITY itself or in the ARCHITECTURE (one or more) that follows, its value will be assumed to be 8.

ENTITY my_entity IS
  GENERIC (n : INTEGER := 8);
  PORT (...);
END my_entity;
ARCHITECTURE my_architecture OF my_entity IS
  ...
END my_architecture;

More than one GENERIC parameter can be specified in an ENTITY. For example:

GENERIC (n: INTEGER := 8; vector: BIT_VECTOR := "0001111");

Complete design examples, further illustrating the use of GENERIC and other attributes and operators, are presented below.

4.6 Examples

We show now a few complete design examples, with the purpose of further illustrating the use of operators, attributes and GENERIC. Recall, however, that so far we have just worked on establishing the basic foundations of VHDL, with the formal discussion on coding techniques starting only in the next chapter (chapter 5). Therefore, a first-time VHDL student should not feel discouraged if the constructs in the examples look still unfamiliar. Instead, you may have a look at the examples now, and then, after studying chapters 5 to 7, return and reexamine them.

Example 4.1: Generic Decoder

Figure 4.1 shows the top-level diagram of a generic m-by-n decoder. The circuit has two inputs, sel (m bits) and ena (single bit), and one output, x (n bits). We assume that n is a power of two, so \( m = \log_2 n \). If ena = ‘0’, then all bits of x should be high;
otherwise, the output bit selected by sel should be low, as illustrated in the truth table of figure 4.1.

The ARCHITECTURE below is totally generic, for the only changes needed to operate with different values of m and n are in the ENTITY (through sel, line 7, and x, line 8, respectively). In this example, we have used m = 3 and n = 8. However, though this works fine, the use of GENERIC would have made it clearer that m and n are indeed generic parameters. That is indeed the procedure that we will adopt in the other examples that follow (please refer to problem 4.4).

Notice in the code below the use of the following operators: ‘+’ (line 22), ‘*’ (lines 22 and 24), ‘:=’ (lines 17, 18, 22, 24, and 27), ‘<=’ (line 29), and ‘=>’ (line 17). Notice also the use of the following attributes: HIGH (lines 14–15) and RANGE (line 20).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY decoder IS
  PORT ( ena : IN STD_LOGIC;
         sel : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
         x : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END decoder;

ARCHITECTURE generic_decoder OF decoder IS
BEGIN
  PROCESS (ena, sel)
  BEGIN
    VARIABLE temp1 : STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);
    VARIABLE temp2 : INTEGER RANGE 0 TO x'HIGH;
    BEGIN

      -- Diagram of decoder

      sel (m-1:0) → m x n
      ena
      DECODER
      x(n-1) → x(n-2)
      ... → x(1)
      x(0)

      Table 4.1
      Decoder of example 4.1.

<table>
<thead>
<tr>
<th>ena</th>
<th>sel</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>1111</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>1110</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>1101</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1011</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0111</td>
</tr>
</tbody>
</table>

Figure 4.1
Decoder of example 4.1.

```
17    temp1 := (OTHERS => '1');
18    temp2 := 0;
19    IF (ena='1') THEN
20        FOR i IN sel'RANGE LOOP -- sel range is 2 downto 0
21            IF (sel(i)='1') THEN -- Bin-to-Integer conversion
22                temp2:=2*temp2+1;
23            ELSE
24                temp2 := 2*temp2;
25            END IF;
26        END LOOP;
27        temp1(temp2):='0';
28    END IF;
29    x <= temp1;
30 END PROCESS;
31 END generic_decoder;
32 ---------------------------------------------

The functionality of the encoder above can be verified in the simulation results of
figure 4.2. As can be seen, all outputs are high, that is, x = '11111111' (decimal 255),
when ena = '0'. After ena has been asserted, only one output bit (that selected
by sel) is turned low. For example, when sel = '000' (decimal 0), x = '11111110'
(decimal 254); when sel = '001' (decimal 1), x = '11111101' (decimal 253); when
sel = '010' (decimal 2), x = '11111011' (decimal 251); and so on.