DIGITAL CIRCUITS DESCRIPTION

Combinational circuits

Sequential circuits

Design organization. Generic Design

Repetitive operations

Authors: Celia López, Luis Entrena Arrontes, Mario García, Enrique San Millán, Marta Portela, Almudena Lindoso

Entities and Architectures

☞ **Already studied**

```
ENTITY entity_name IS
              [ GENERIC (...); ]
              [ PORT (...); ]
              [ Type declarations, components, etc. ; ]
BEGIN
              [ Passive concurrent statements ; ]
END [entity_name ] ;
```

```
ARCHITECTURE arch_name OF entity_name IS
              [Type declarations, components, etc. ; ]
BEGIN
              [ Concurrent statements; ]
END [arch_name] ;
```

Generic ports and parameters

- Ports are the circuit's interface
- Modes
  - IN: Input port. It only can be read.
  - OUT: Output port. It only can be written.
  - INOUT: Input/Output port. It can be read as well as written.
- Generic parameters are constants that are used to parameterize the designs. They are usually used to model constants, signal ranges and delay times.

Generic parameters: N-bit adder

- **The parameter is declared and used in the circuit description.**

```
ENTITY Adder IS
      GENERIC (
            n: INTEGER := 8 );  -- By default the adder is for
                                  --8-bit data
      PORT (
            a,b: IN STD_LOGIC_VECTOR (n-1 downto 0);
            s: OUT STD_LOGIC_VECTOR (n-1 downto 0);
            Cout: OUT STD_LOGIC );
END Adder ;

ARCHITECTURE functional OF Adder IS
      SIGNAL res: STD_LOGIC_VECTOR (n downto 0);
BEGIN
      res <= '0'&a + '0'&b;         -- Adding n+1 bits
      s <= res(n-1 downto 0);       -- N bits result
      Cout <= res(n);               -- Carry out
END functional ;
```

- **The generic parameter's value is assigned when the circuit is instantiated.**

- **When a value is not assigned, the value by defect is assumed.**

```
ARCHITECTURE … OF … IS
      ………….
BEGIN
      add1: Adder
            GENERIC MAP ( n => 16 );
            PORT MAP ( a => a, ….. );

END …;
```

Open Ports

- **Ports can be open. To this purpose we use the key word OPEN**

```
COMPONENT HalfAdder
                    PORT ( a: IN BIT;  b: IN BIT; s: OUT BIT; c: OUT BIT);
END COMPONENT;
          ......
SS0: HalfAdder PORT MAP (x, y, s_partial, OPEN);
```

We are not using the carry out

- **When an input port is open the initial value by default is assumed. This case is not recommended.**

Design organization

- VHDL language supports the use of design libraries to manage components and utilities (type and constant declarations, subprograms, etc.)
- How to create a library depends on the software tool used, but the VHDL standard does define how to use them.
- Predefined libraries:
  - Standard (STD): It contains the definitions of the standard types like BIT, BIT_VECTOR, TIME, as well as the functions to access ASCII files.
  - WORK: Current work library.

<u>Using libraries</u>

- To select components and utilities we must use the context clauses:
  - ❑ Clause LIBRARY: Used to select a library.
  - ❑ Clause USE: Used to select an entity, object, etc, that is defined in a given library.
- Context clauses must be written immediately before the component where we are going to use them.
- We assume that any design contains <u>implicitly</u> the following statements:
  - ❑ LIBRARY STD
  - ❑ USE STD.STANDARD.ALL
  - ❑ LIBRARY WORK

Example

```
LIBRARY UNISIM;                  --  Makes accessible the UNISIM
use UNISIM.vcomponents.ALL;      --  Makes accessible the package
                                  --Vcomponents
USE WORK.my_package.my_components;
                   --  Makes accessible the component my_component, that
                   --   is declared within the package my_package
ARCHITECTURE ... OF ....
```

Context clauses are written immediately before the entity that uses them

Subprograms: Functions and procedures

- As in other high level languages, VHDL provides the possibility of using subprograms to structure circuit descriptions.
- Two kinds of subprograms:
  - ❑ Functions: return always a value and the cannot modify the value of their parameters.
  - ❑ Procedures: they are used as sentences and they can modify the value of their parameters.
- Subprograms can be recursive, although in this case they are not synthesizable.

Example of a function

```
FUNCTION parity (a: IN STD_LOGIC_VECTOR) RETURN STD_LOGIC IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE  loop
                p:= p XOR a(i);
        END LOOP;
        RETURN p;
END parity;
```

```
-- Calling the function
        r <= parity( “00101101”);
```

Example of a procedure

```
PROCEDURE parity_proc (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC) IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE  LOOP
                p:= p XOR a(i);
        END LOOP;
        s := p;
END parity_proc;
```

```
-- Calling the procedure
        parity_proc( “00101101”, r);
```

Parameter associations in subprograms

- **Parameter associations could be either by position or by name as in the case of components.**
- **Procedures have parameters of mode IN, OUT or INOUT.**
- **Functions have only parameters of mode IN.**
- **When the subprogram is called, data types must match except in case of a constant parameter, which can be assigned to any expression.**
- **If the type of the object is not declared, the parameter is assumed as CONSTANT for parameters of mode IN, and VARIABLE for parameters of mode OUT or INOUT**

Packages

- **A package is a repository for storing commonly used declarations that can be used in several designs.**
- **A package declaration can contain:**
  - **Declaration of different kinds: types, subtypes, constants, signals, files, aliases, subprograms, attributes and components.**
  - **Clauses USE**
  - **Specifications of attributes**
- **A package declaration can have associated a package body that can contains:**
  - **Declarations of: types, subtypes, constants, signals, files, alias and subprograms.**
  - **Clauses USE**
  - **Subprogram bodies of functions and procedures declared in the package declarations**

Example: declaring and using packages

```
PACKAGE my_package IS
        COMPONENT HalfAdder
                PORT ( a: IN BIT;  b: IN BIT; s: OUT BIT; c: OUT BIT);
        END COMPONENT;
        TYPE int_numbers  IS ARRAY ( 0 TO 12 ) OF INTEGER;
        CONSTANT delay : TIME := 10 NS;
        PROCEDURE parity_proc (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC);
END my_package ;
```

```
USE WORK. my_package.ALL;
ARCHITECTURE using_my_package OF ... IS
...
END using_my_package;
```

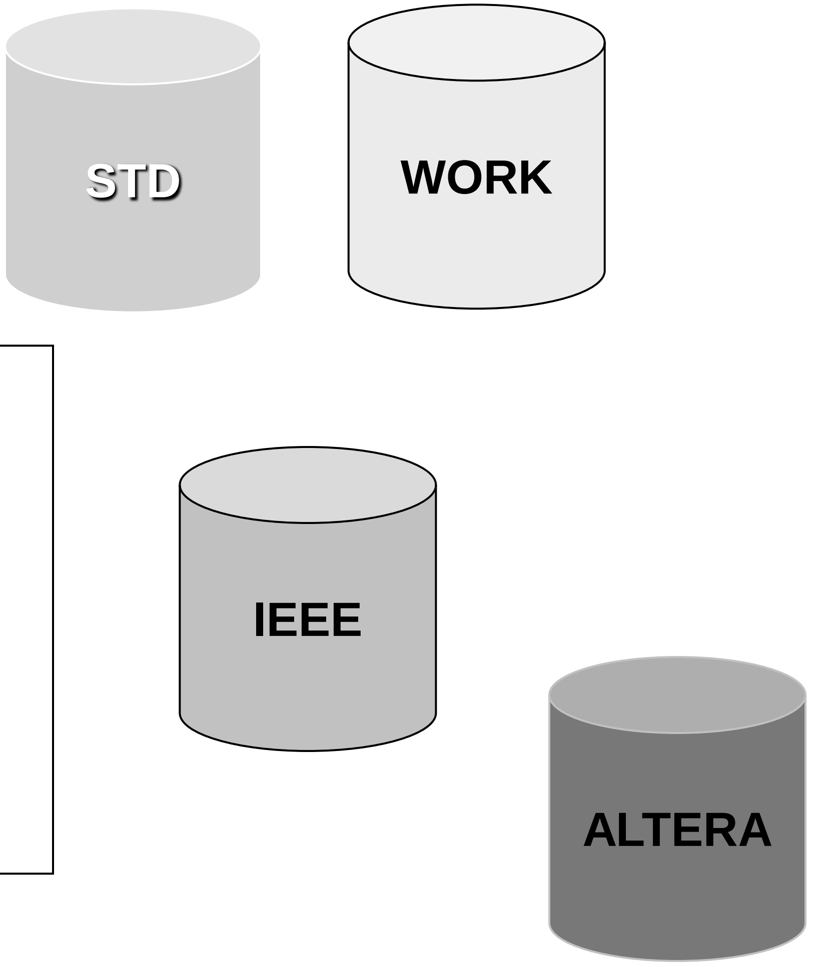
Example: Package body

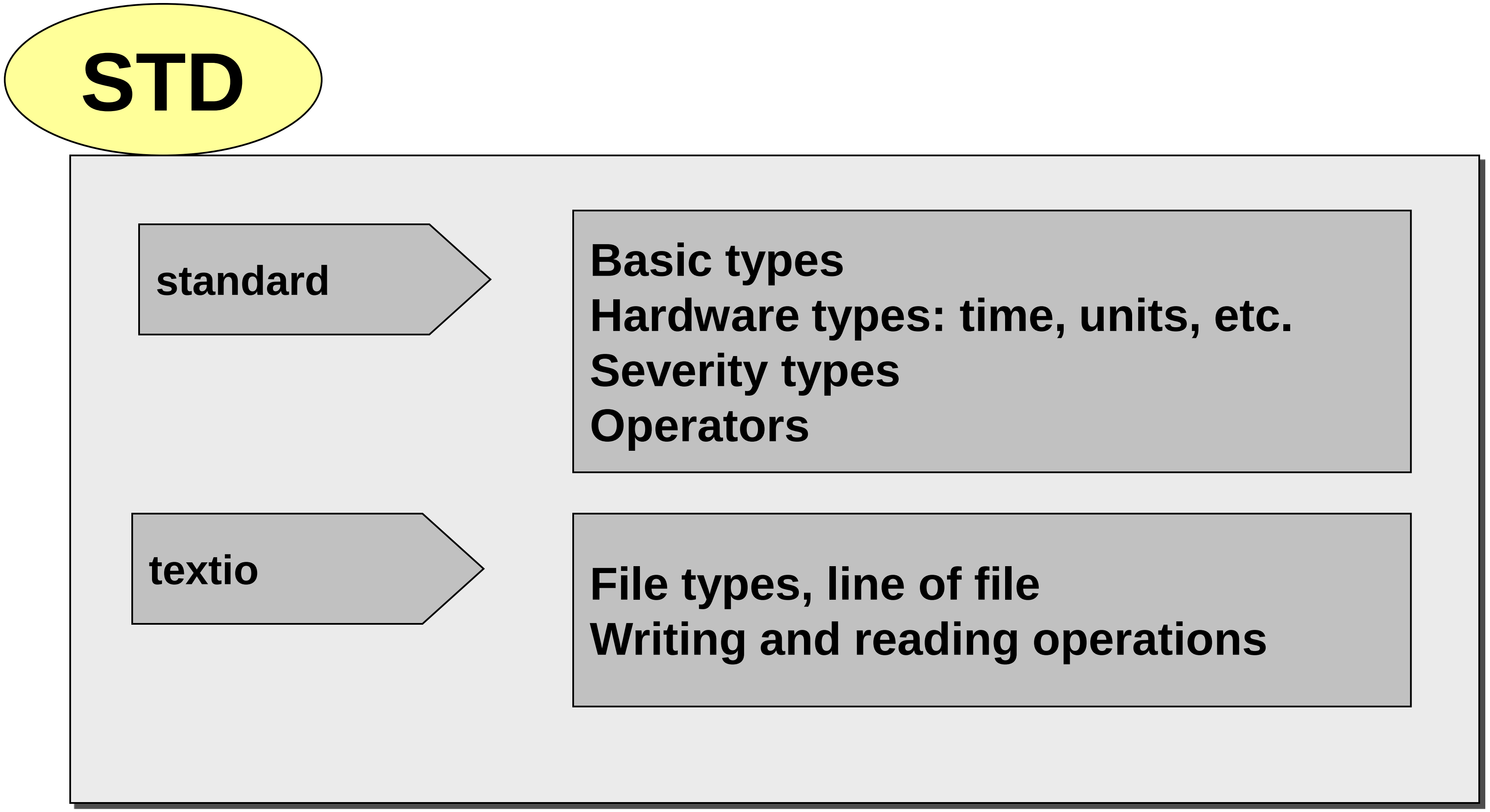
```
PACKAGE BODY my_package IS
PROCEDURE parity_proc (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC) IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE  LOOP
                p:= p XOR a(i);
        END LOOP;
        s := p;
END parity_proc;
END my_package ;
```

Standard libraries

```
library STD;
use STD.STANDARD.all;
library WORK;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_textio.all;
…
```





Basic types
Hardware types: time, units, etc.
Severity types
Operators

File types, line of file
Writing and reading operations

package STANDARD is

Package

```
    TYPE BOOLEAN IS (FALSE, TRUE);
    TYPE BIT IS ('0', '1');
    TYPE CHARACTER IS
(NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT,
FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
CAN, EM, SUB, ESC, FSP, GSP, RSP, USP, ' ', '!', '"',
'#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '-',
'.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8',
'9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z', '[', '\', ']', '^', '_', '`', 'a', 'b', 'c', 'd',
'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'{', '|', '}', '~', DEL);
    TYPE SEVERITY_LEVEL IS
                (NOTE, WARNING, ERROR, FAILURE);
    TYPE INTEGER IS
                RANGE -2147483648 TO 2147483647;
    TYPE REAL IS
                RANGE -1.7014110e+038 TO 1.7014110e+038;
```

```
TYPE TIME IS
    RANGE -9223372036854775808 TO 9223372036854775807
     UNITS fs;
        ps = 1000 fs;
        ns = 1000 ps;
        us = 1000 ns;
        ms = 1000 us;
        sec = 1000 ms;
        min = 60 sec;
        hr = 60 min;
     END UNITS;
  FUNCTION NOW RETURN TIME;
  SUBTYPE NATURAL IS
            INTEGER RANGE 0 TO INTEGER'HIGH;
  SUBTYPE POSITIVE IS
             INTEGER RANGE 1 TO INTEGER'HIGH;
  TYPE STRING IS
            ARRAY (POSITIVE RANGE <>) OF CHARACTER;
  TYPE BIT_VECTOR IS
            ARRAY (NATURAL RANGE <>) OF BIT;
END STANDARD;
```

IEEE

std_logic_1164	std_logic, std_logic_vector
std_logic_arith	signed, unsigned, operations
std_logic_signed	std_logic_vector used as signed
std_logic_unsigned	std_logic_vector used as signed
std_logic_…	

PACKAGE std_logic_1164 IS

Package

```
    ------------------------------------------------------------------
    -- logic state system  (unresolved)
    ------------------------------------------------------------------
    TYPE std_ulogic IS ( 'U',  -- Uninitialized
                         'X',  -- Forcing  Unknown
                         '0',  -- Forcing  0
                         '1',  -- Forcing  1
                         'Z',  -- High Impedance
                         'W',  -- Weak     Unknown
                         'L',  -- Weak     0
                         'H',  -- Weak     1
                       );'-'   -- Don't care
    attribute ENUM_ENCODING of std_ulogic : type is "U D 0 1 Z D 0 1 D";
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;
```

…

END std_logic_1164;

Package body

PACKAGE BODY std_logic_1164 IS

```
    -------------------------------------------------------------------
    -- local types
    -------------------------------------------------------------------
    --synopsys synthesis_off
    TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
    TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
    -------------------------------------------------------------------
    -- resolution function
    -------------------------------------------------------------------
    CONSTANT resolution_table : stdlogic_table := (
    --      ---------------------------------------------------------
    --      |  U    X    0    1    Z    W    L    H    -        |   |
    --      ---------------------------------------------------------
            ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
            ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
            ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
            ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
…
```

END std_logic_1164;

Arithmetic with and without sign

- Package: IEEE.STD_LOGIC_ARITH
  - Types: UNSIGNED, SIGNED
  - Operations: +,-,*
  - Functions: SHL, SHR, EXT, SEXT
  - Conversion functions:
    - STD_LOGIC_VECTOR(), SIGNED(), UNSIGNED()
    - CONV_INTEGER(), CONV_SIGNED(), CONV_UNSIGNED()
    - CONV_STD_LOGIC_VECTOR()
- Package: IEEE. STD_LOGIC_SIGNED
  - Operations with sign for data of type STD_LOGIC_VECTOR
- Package: IEEE. STD_LOGIC_UNSIGNED
  - Operations without sign for data of type STD_LOGIC_VECTOR

- Option 1:
  - To include STD_LOGIC_ARITH
  - To use the types SIGNED Y UNSIGNED
- Option 2:
  - To include STD_LOGIC_ARITH
  - To include STD_LOGIC_UNSIGNED or STD_LOGIC_SIGNED
  - To use STD_LOGIC_VECTOR, that are interpreted in base on the included package.

Typical usage of libraries

```
-- Minimum declaration required for
-- using STD_LOGIC and STD_LOGIC_VECTOR
-- types
library IEEE;
use IEEE.std_logic_1164.all;
```

```
-- Arithmetic operations
-- with SIGNED and UNSIGNED types
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
-- Arithmetic operations that use
-- STD_LOGIC_VECTOR data like
-- integers without sign
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
-- Arithmetic operations that use
-- STD_LOGIC_VECTOR data like
-- integers with sign
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
```

DIGITAL CIRCUITS DESCRIPTION

Combinational circuits

Sequential circuits

Design organization. Generic design

Repetitive operations

Loops

☞ **Bucles**

```
-- Infinite loop
LOOP
          ...
END LOOP;
```

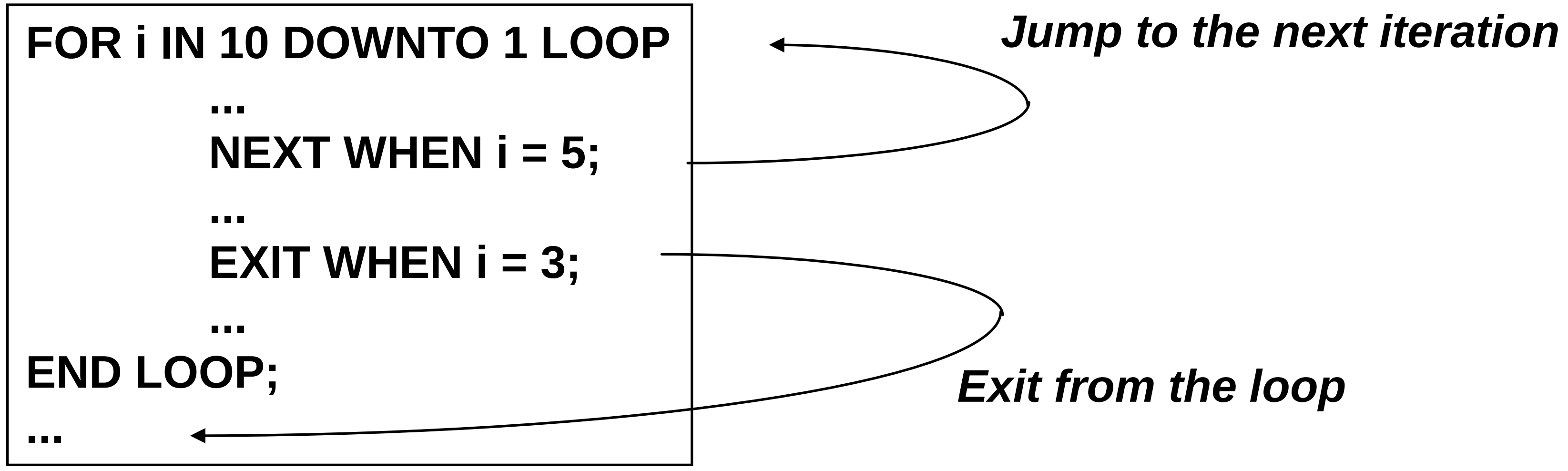
```
-- While loop
WHILE a < b LOOP
          ...
END LOOP;
```

```
-- For loop
FOR i IN 10 DOWNTO 1 LOOP
          ...
END LOOP;
```

☞ **Statement NEXT ... [WHEN]: it provokes a jump to the next loop iteration**

☞ **Statement EXIT ... [WHEN]: it causes an exit from the loop**

Example of using NEXT and EXIT



```
FOR i IN 10 DOWNTO 1 LOOP
            ...
            NEXT WHEN i = 5;
            ...
            EXIT WHEN i = 3;
            ...
END LOOP;
...
```

Synthesis of a loop

- **A loop can be synthesized only if the number of iterations to perform is a static value, and therefore it is possible to know the exact necessary hardware**
- **NEXT and EXIT are not syntesizable**
- **WHILE is syntesizable only in some syntesizers, and just when the number of iterations is a static value**
- **For synthesis, FOR loops must contain a constant range**

Example

```
SIGNAL a: STD_LOGIC_VECTOR (0 TO 7);
SIGNAL z: STD_LOGIC;
...
z <= a(0) AND
      a(1) AND
      a(2) AND
      a(3) AND
      a(4) AND
      a(5) AND
      a(6) AND
      a(7);
```

```
SIGNAL a: STD_LOGIC_VECTOR (0 TO 7);
SIGNAL z: STD_LOGIC;
...
PROCESS (a)
      VARIABLE aux: STD_LOGIC;
BEGIN
      aux := a(0);
      FOR i IN 1 to 7 LOOP
            aux := aux AND a(i);
      END LOOP;
      z <= aux;
END PROCESS;
```

Statement GENERATE

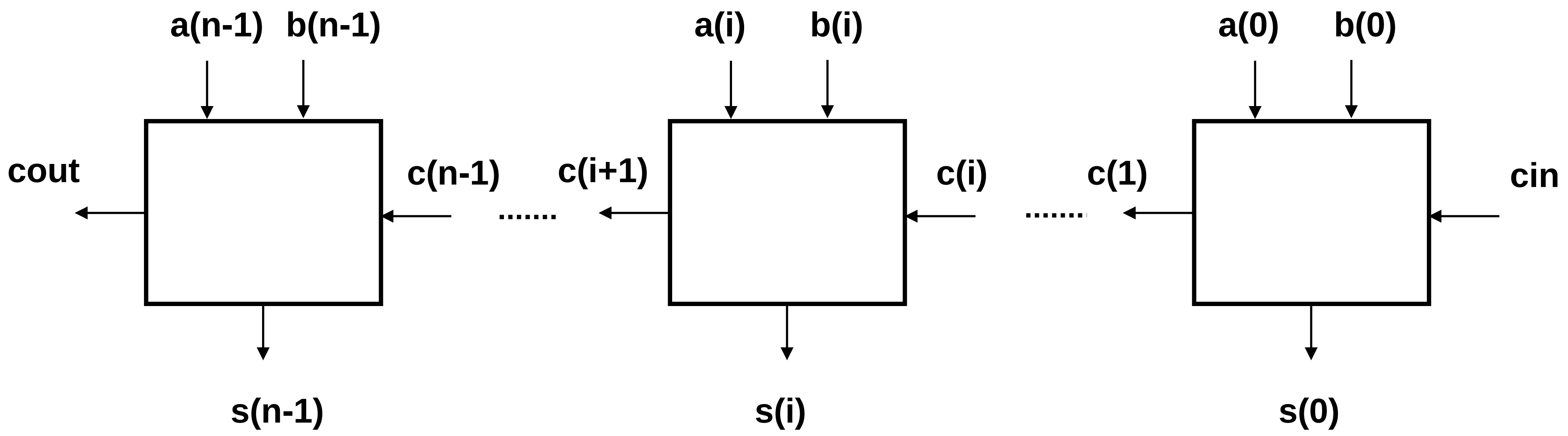
- It allows the designer to replicate components in an easy way in base on some parameters
- Two ways:
  - The statement *for…generate* allows the replication of concurrent statement several times

```
Label:  FOR name IN discrete_range GENERATE
                        ...-- concurrent statements
             END GENERATE [ Label];
```

  - *If… generate* generates concurrent statements only if a condition is met

```
Label:  IF condition GENERATE
                        ... -- concurrent statements
             END GENERATE [ Label];
```

Example: n-bits adder



N-bits adder in VHDL

```
ENTITY n_bits_adder  IS
      GENERIC (n: INTEGER);
      PORT ( a:         IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
                  b:         IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
                  cin:       IN STD_LOGIC;
                  s:         IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
                  cout:     OUT STD_LOGIC );
END n_bits_adder ;

ARCHITECTURE generated OF n_bits_adder IS
      COMPONENT FullAdder
      PORT ( x: IN STD_LOGIC; y: IN STD_LOGIC; cin: IN STD_LOGIC;
                        s: OUT STD_LOGIC; cout: OUT STD_LOGIC);
      END COMPONENT;
      SIGNAL c : STD_LOGIC_VECTOR (n DOWNTO 0);
BEGIN
              c(0) <= cin;
fors:         FOR i IN 0 to n-1 GENERATE
addx:                     FullAdder PORT_MAP (a(i), b(i), c(i), s(i), c(i+1));
              END GENERATE;
              cout <= c(n);
END generated;
```