Paquetes y componentes [1]

Sistemas Digitales Avanzados

Universidad Técnica Particular de Loja

Prof: Diego Barragán Guerrero

Oct. 2014 - Feb. 2015

Introducción.

- Durante el primer bimestre, se estudió temas como:
  - Estructura de código: declaración de librerías, entidad, arquitectura.
  - Tipos de datos.
  - Operadores y atributos.
  - Declaraciones concurrente y código concurrente.
  - Declaraciones secuenciales y código secuencial.
  - Señales, variables y constantes.
  - Diseño de máquinas de estado finitas.
- En este bimestre, se añadirán nuevos elementos principalmente para asignación de librerías, tales como:
  - *Packages* (paquetes).
  - *Components* (componentes).
  - *Functions* (punciones).

Introducción.

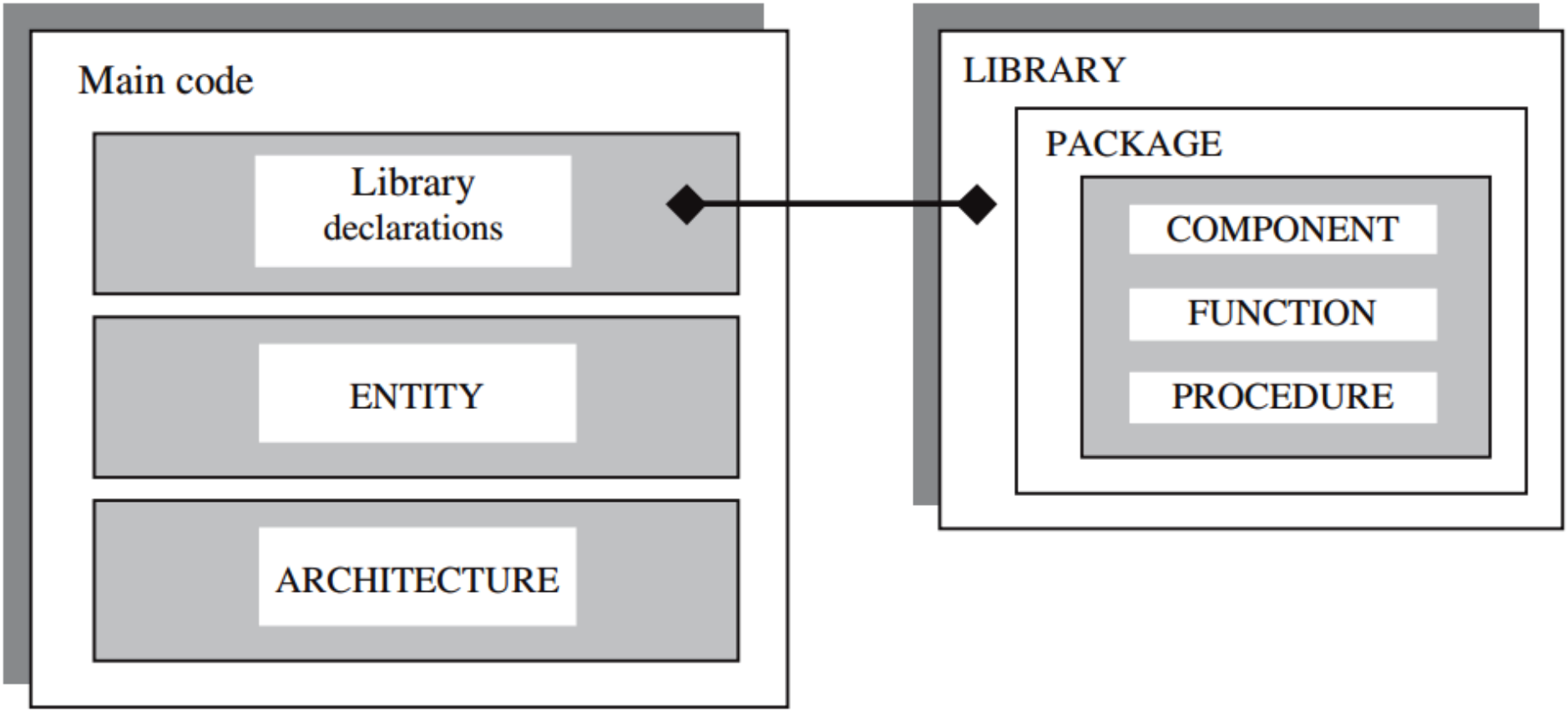

Figura: Unidades fundamentales de VHDL.

Introducción.

- Estas nuevas unidades pueden ser colocadas dentro del mismo código.
- Sin embargo, debido a que su propósito principal es permitir que piezas comunes de código sean reusadas y compartidas, es más común colocarlas en una librería.
- Esto permite particionar el código, lo cual es de mucha ayuda en código extensos.
- Por lo tanto, piezas comunes de código son escritas en forma de COMPONENTES, FUNCIONES o PROCEDIMIENTOS, luego colocados en un PAQUETE, el cual es finalmente compilado dentro de una LIBRERÍA.

PACKAGE

Sintaxis

```
PACKAGE package_name IS
     (declarations)
END package_name;
[PACKAGE BODY package_name IS
     (FUNCTION and PROCEDURE descriptions)
END package_name;]
```

- La sintaxis está compuesta de dos partes: PACKAGE y PACKAGE BODY.
- La primera parte es obligatoria y contiene todas las declaraciones.
- La segunda parte es necesaria solo cuando uno o más subprogramas son declarados. PACKAGE y PACKAGE BODY deben tener el mismo nombre.
- La lista de declaraciones del proceso puede contener: COMPONENTES, FUNCIONES, PROCEDIMIENTOS, TYPOS, CONSTANTES, etc.

Package: Ejemplo 1.

- El siguiente ejemplo muestra un paquete llamado my_package. Contiene solo declaraciones de tipos y constantes, de modo que el cuerpo del paquete (package body) no es necesario.

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 ------------------------------------------------
4 PACKAGE my_package IS
5     TYPE state IS (st1, st2, st3, st4);
6     TYPE color IS (red, green, blue);
7     CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
8 END my_package;
```

Package: Ejemplo 2

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ------------------------------------------------
4  PACKAGE my_package IS
5      TYPE state IS (st1, st2, st3, st4);
6      TYPE color IS (red, green, blue);
7      CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
8      FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN;
9  END my_package;
10 ------------------------------------------------
11 PACKAGE BODY my_package IS
12     FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS
13     BEGIN
14         RETURN (s'EVENT AND s='1');
15     END positive_edge;
16 END my_package;
```

Package.

- Cualquiera de los paquetes anteriores pueden ahora ser compilados, llegando a ser parte de la librería *work* (o cualquier otra).
- Para hacer uso de esta librería en el código VHDL, se añade una nueva sentencia USE al código principal, tal como se muestra:

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE work.my_package.all;
4 ------------------------------------
5 ENTITY...
6 ...
7 ARCHITECTURE...
8 ...
9 ------------------------------------
```

Component.

- Al declarar un código como un componente, puede ser utilizado dentro de otro circuito, lo que permite la construcción de diseños jerárquicos.
- Un COMPONENT es también otra forma de particionar el código de tal forma que pueda ser compartido y reutilizado. Por ejemplo, circuitos comunes como FF, multiplexores, sumadores, compuerta básica pueden ser colocados dentro de una librería, de modo que cualquier proyecto puede hacer uso de ellos sin necesidad de volver a escribirlos.
- Para hacer uso de un COMPONENT, debe ser declarado. La sintaxis de declaración se muestra a continuación:

```
1 --DeclaraciOn del componente:
2 COMPONENT component_name IS
3 PORT (
4 port_name : signal_mode signal_type;
5 port_name : signal_mode signal_type;
6 ...);
7 END COMPONENT;
8 --COMPONENT instanciacion:
9 label: component_name PORT MAP (port_list);
```

Modos de declaración de componentes.

- Existen dos maneras para declarar un componente.
- Una es diseñarlo y colocarlo en una librería (luego declarar el componente en el código principal).
- Otra forma es usando un paquete (PACKAGE), lo que evita el uso de repetición de declaraciones cada vez que se haga uso del componente.

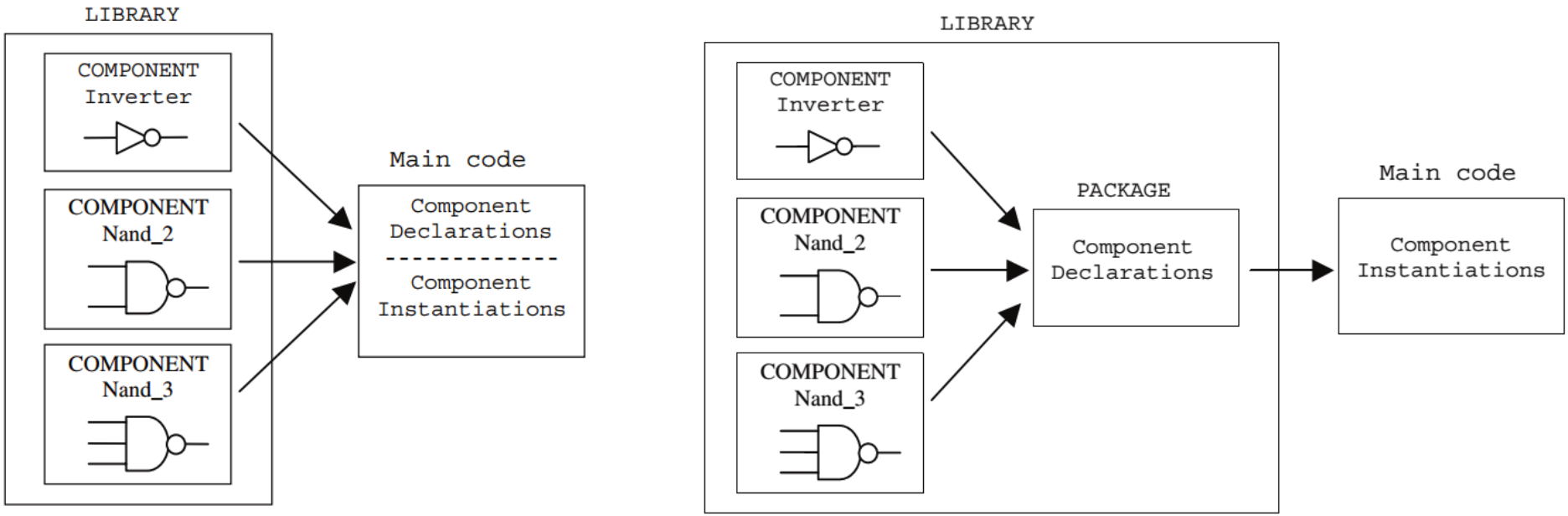

Figura: Declaración de componente: en código y en paquete.

Ej: declaración del componente en el código principal.

- Se implementará el diseño de la figura usando solo COMPONENTS (inverter, nand_2, nand_3), sin crear un PACKAGE.
- Serán necesarios cuatro módulos VHDL: uno para cada componente y otro para el proyecto.
- Debido a que no se crea un paquete, los componentes deben ser declarados en el código principal (parte declarativa de la arquitectura).

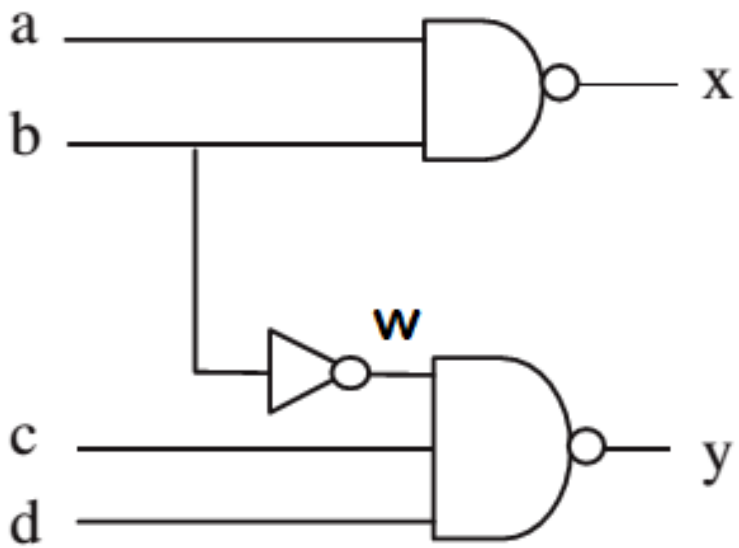

Figura: Circuito de ejemplo.

Ej: declaración del componente en el código principal.

```
1  ------ File inverter.vhd: -------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  ENTITY inverter IS
6  PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
7  END inverter;
8  ------------------------------------
9  ARCHITECTURE inverter OF inverter IS
10 BEGIN
11 b <= NOT a;
12 END inverter;
```

```
1  ------ File nand_2.vhd: ---------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  ENTITY nand_2 IS
6  PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
7  END nand_2;
8  ------------------------------------
9  ARCHITECTURE nand_2 OF nand_2 IS
10 BEGIN
11 c <= NOT (a AND b);
12 END nand_2;
```

Ej: declaración del componente en el código principal.

```
1  ----- File nand_3.vhd: ----------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  ENTITY nand_3 IS
6  PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
7  END nand_3;
8  ------------------------------------
9  ARCHITECTURE nand_3 OF nand_3 IS
10 BEGIN
11 d <= NOT (a AND b AND c);
12 END nand_3;
```

Ej: declaración del componente en el código principal.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ------------------------------------
4  ENTITY project IS
5  PORT (a, b, c, d: IN STD_LOGIC;
6  x,y: OUT STD_LOGIC);
7  END project;
8  ------------------------------------
9  ARCHITECTURE structural OF project IS
10 COMPONENT inverter IS
11 PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
12 END COMPONENT;
13 -------------
14 COMPONENT nand_2 IS
15 PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
16 END COMPONENT;
17 -------------
18 COMPONENT nand_3 IS
19 PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
20 END COMPONENT;
21 -------------
22 SIGNAL w: STD_LOGIC;
23 BEGIN
24 U1: inverter PORT MAP (b, w);
25 U2: nand_2 PORT MAP (a, b, x);
26 U3: nand_3 PORT MAP (w, c, d, y);
27 END structural;
```

Declaración del componente en un PACKAGE.

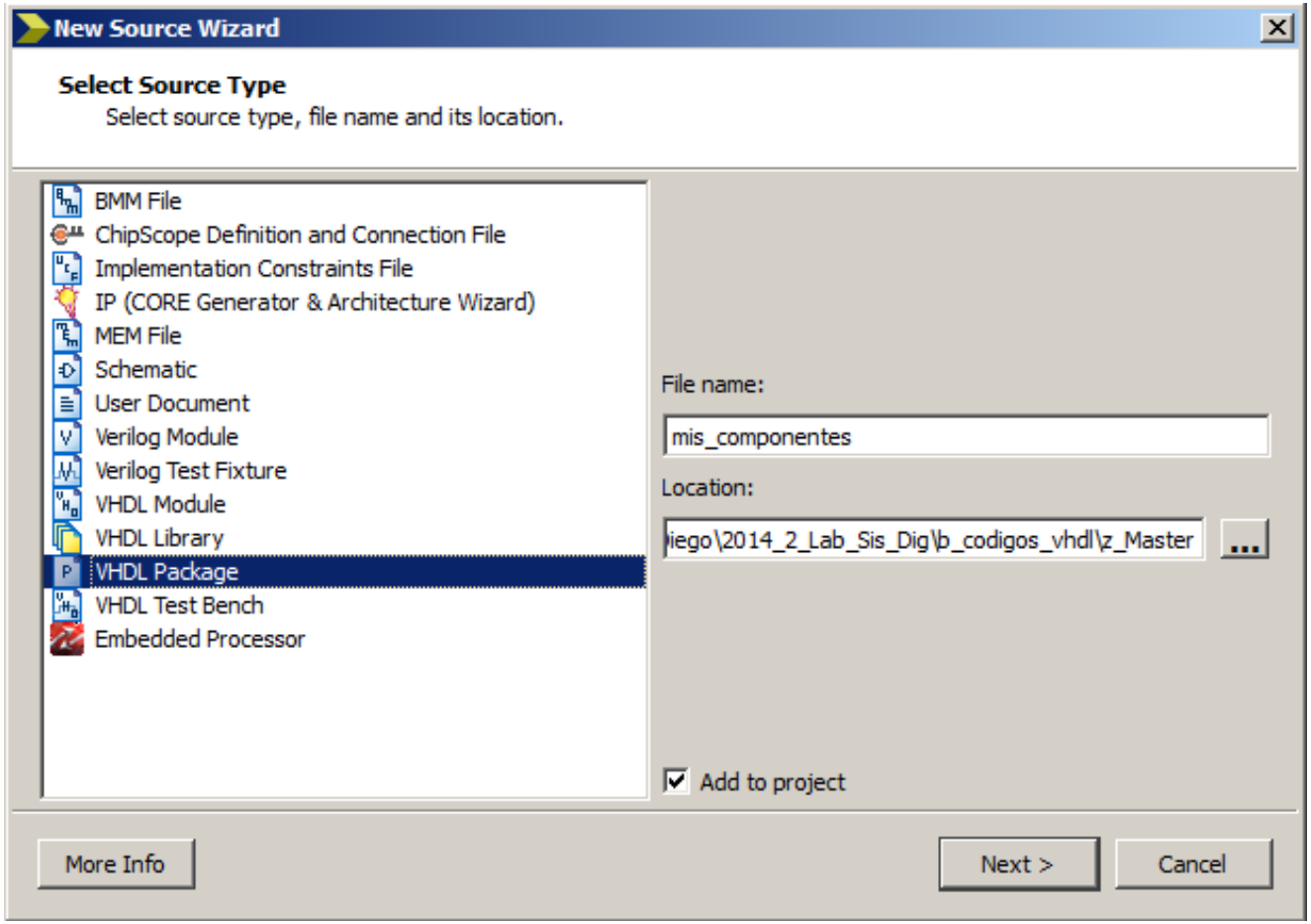

Figura: Crear plantilla de paquete.

Declaración del componente en un PACKAGE.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  -------------
4  package mis_componentes is
5  -------------
6  COMPONENT inverter IS
7  PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
8  END COMPONENT;
9  -------------
10 COMPONENT nand_2 IS
11 PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
12 END COMPONENT;
13 -------------
14 COMPONENT nand_3 IS
15 PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
16 END COMPONENT;
17 -------------
18 end mis_componentes;
```

Declaración del componente en un paquete.

- A pesar de crear un módulo adicional (PACKAGE), el módulo extra solo se crea una vez, así se evita la necesidad de declarar los componentes en el código principal cada vez que sean de usarse.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use work.mis_componentes.all; --Incluir el paquete
4  ------------------------------------
5  ENTITY project IS
6  PORT (a, b, c, d: IN STD_LOGIC;
7                x,y: OUT STD_LOGIC);
8  END project;
9  ------------------------------------
10 ARCHITECTURE structural OF project IS
11 -------------
12 SIGNAL w: STD_LOGIC;
13 BEGIN
14 U1: inverter PORT MAP (b, w);
15 U2: nand_2 PORT MAP (a, b, x);
16 U3: nand_3 PORT MAP (w, c, d, y);
17 END structural;
18 ----------------------------------------
```

PORT MAP

- Existen dos formas de mapear los puertos de un componente durante su instalación: mapeo posicional y mapeo nominal.

```
1 COMPONENT inverter IS
2 PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
3 END COMPONENT;
4 ...
5 U1: inverter PORT MAP (x, y);
```

- Este mapeo es posicional, esto es, los puertos 'x' y 'y' corresponden a 'a' y 'b', respectivamente.
- Por otro lado, el mapeo nominal es:

```
1 U1: inverter PORT MAP (x=>a, y=>b);
```

- El mapeo posicional es fácil de escribir, pero el mapeo nominal tiene menos posibilidad de error.
- Los puertos pueden quedar sin conexión (usando la sentencia OPEN). Por ejemplo:

```
1 U2: my_circuit PORT MAP (x=>a, y=>b, w=>OPEN, z=>d);
```

GENERIC MAP

- El uso de genéricos en componentes se logra con la sentencia GENERIC MAP para pasar información al parámetro genérico. La sintaxis es la siguiente:

```
1 label: compon_name GENERIC MAP (param. list) PORT MAP (port list);
```

- Vamos a considerar un generador de paridad genérico el cual añade un bit al vector de entrada (en su parte izquierda). Tal bit será '0' si el número de unos en el vector de entrada es par, y será '1' si el número de bits es impar, de modo que el vector siempre tenga un número par de unos.

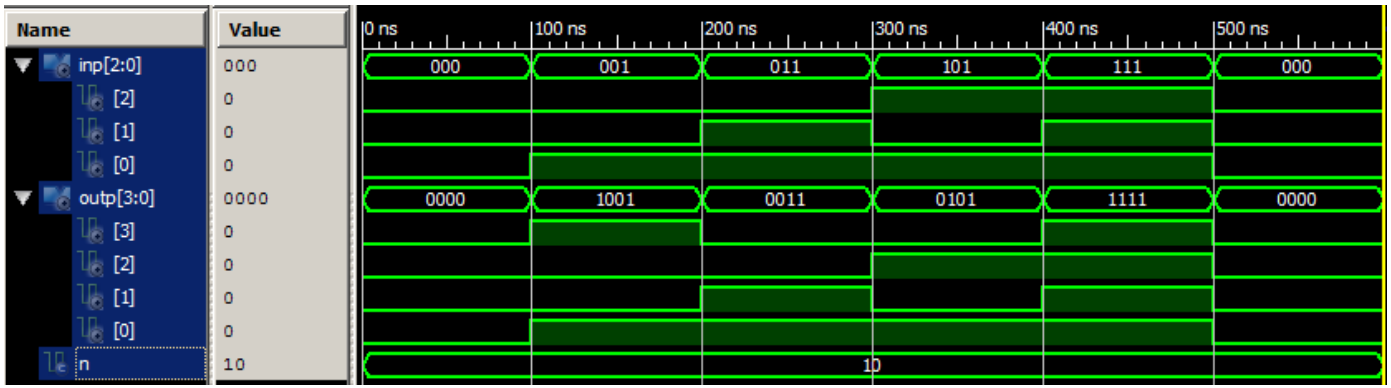

Figura: Generador de paridad.

GENERIC MAP

```
1  ------ File parity_gen.vhd (component): ------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------
5  ENTITY parity_gen IS
6  GENERIC (n : INTEGER := 7); -- default is 7
7  PORT (input: IN std_logic_vector (n DOWNTO 0);
8       output: OUT std_logic_vector (n+1 DOWNTO 0));
9  END parity_gen;
10 ---------------------------------
11 ARCHITECTURE parity OF parity_gen IS BEGIN
12 PROCESS (input)
13 VARIABLE temp1: std_logic;
14 VARIABLE temp2: std_logic_vector (output'RANGE);
15 BEGIN
16     temp1 := '0';
17     FOR i IN input'RANGE LOOP
18         temp1 := temp1 XOR input(i);
19         temp2(i) := input(i);
20     END LOOP;
21     temp2(output'HIGH) := temp1;
22     output <= temp2;
23 END PROCESS;
24 END parity;
```

GENERIC MAP

```
1  ------ File my_code.vhd (actual project): ------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------
5  ENTITY my_code IS
6  GENERIC (n : POSITIVE := 2); -- 2 will overwrite 7
7  PORT ( inp: IN std_logic_vector (n DOWNTO 0);
8        outp: OUT std_logic_vector (n+1 DOWNTO 0));
9  END my_code;
10 ---------------------------------
11 ARCHITECTURE my_arch OF my_code IS
12 ------------------------
13 COMPONENT parity_gen IS
14 GENERIC (n : POSITIVE);
15 PORT (input: IN std_logic_vector (n DOWNTO 0);
16       output: OUT std_logic_vector (n+1 DOWNTO 0));
17 END COMPONENT;
18 ------------------------
19 BEGIN
20 C1: parity_gen GENERIC MAP(n) PORT MAP(inp, outp);
21 END my_arch;
```

Bibliografía

[1] V. A. Pedroni.
Circuit Design and Simulation with VHDL.
The MIT Press, 2nd edition, 2010.