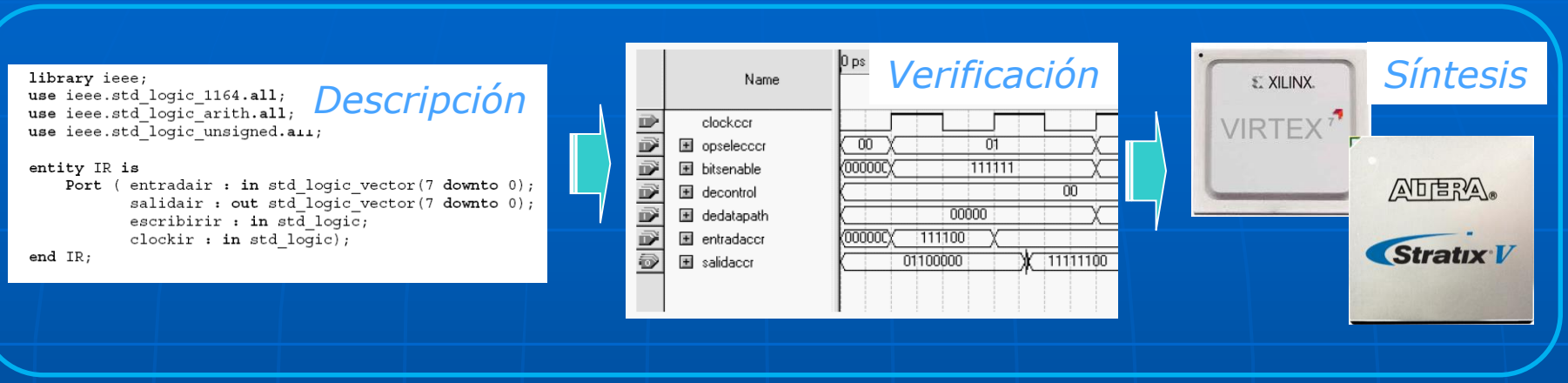


Introducción al diseño lógico con VHDL

Sergio Noriega 2018

VHDL :
Very High Speed Integrated Circuits Hardware Description Language

Qué es?:

Herramienta formal para describir el comportamiento y la estructura de un sistema usando un lenguaje textual.

Qué permite?:

Describir las operaciones de un sistema empleando las siguientes posibilidades:

- Indicar QUE debe hacer el sistema modelando por "comportamiento" (behavior).
- Indicar COMO debe funcionar utilizando "algoritmos".
- Indicar CON QUE hacerlo, empleando "estructuras" ó "flujo de datos".

Historia de de VHDL

VHDL fué diseñado originariamente por el Departamento de Defensa de los Estados Unidos de Norteamérica como una forma de documentar las diversas especificaciones y el comportamiento de dispositivos ASIC de diversos fabricantes que incluían en sus equipos.
Con la posterior posibilidad de simular dichos dispositivos, comenzaron a crearse compiladores que pudieran llevar a cabo esta tarea leyendo los archivos VHDL.
El paso siguiente fué el de desarrollar software capaz de sintetizar las descripciones generadas y obtener una salida apta para su posterior implementación, ya sea en ASIC como en dispositivos CPLD (Dispositivos Lógicos Programable Complejo) y FPGA Arreglo de Compuertas Programable por el Usuario).
CPLD => Complex Programmable Logic Device.
FPGA => Field Programmable Gate Array.

Historia de VHDL (continuación)

La gran masificación de VHDL permite que un mismo diseño sea portable, pudiendo utilizarlo no sólo en varios tipos de dispositivos PLD sino además de diferentes proveedores, donde con el mismo código VHDL se puede sintetizar un diseño para optimizar uno o mas parámetros críticos (área de silicio, velocidad de respuesta, consumo de energía, etc.).

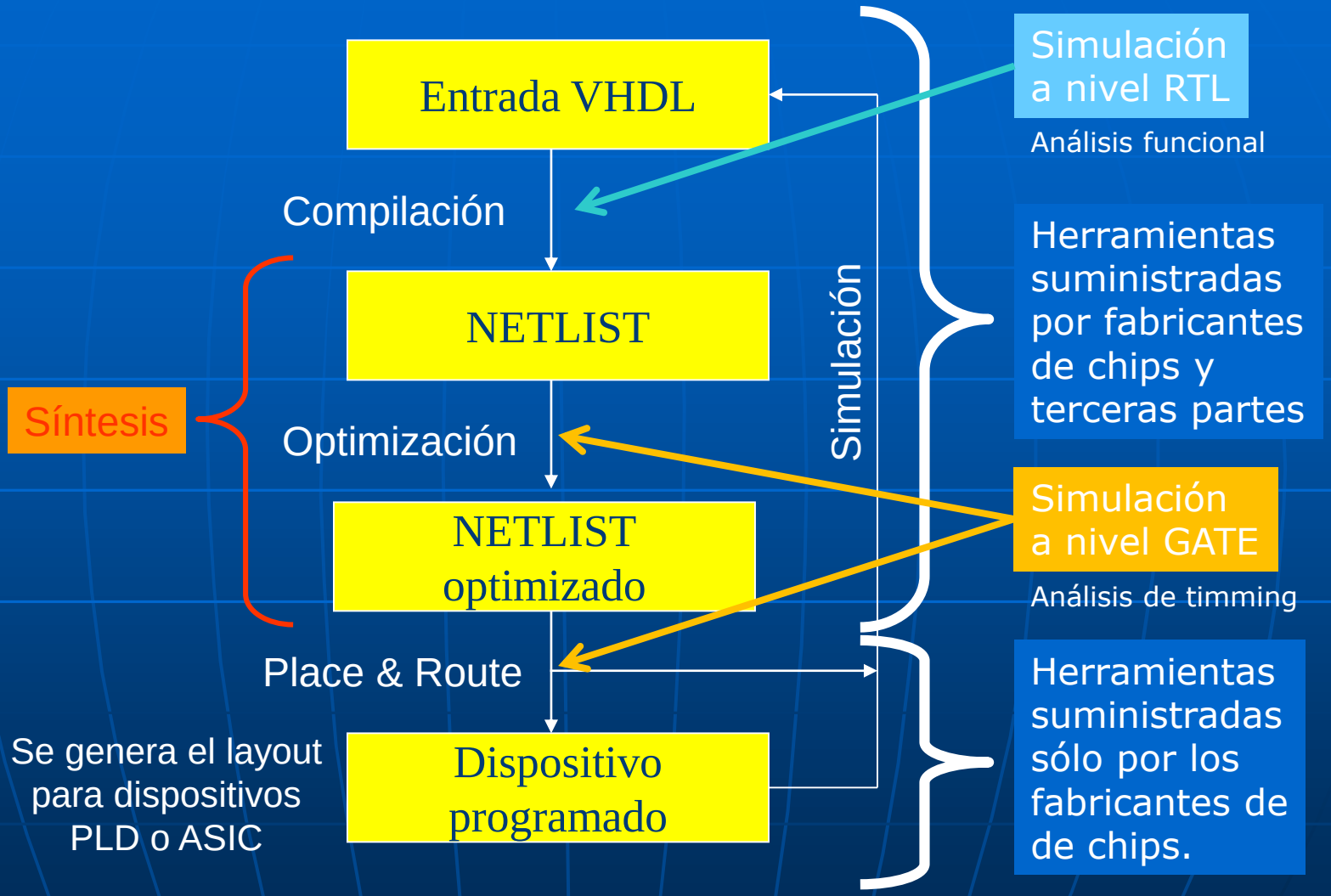
Desde su implementación en el año 1981, VHDL fue estandarizado por primera vez por la IEEE en 1987 (std. 1076) y se realizó un importante actualización en 1993 (con posteriores revisiones en 1994, 2000, 2002 y 2008).

Si bien su uso se aplica fundamentalmente para la descripción de sistemas digitales, en 1999 la IEEE aprobó el standard 1076.1 conocido como VHDL-AMS el cual incluye extensiones para entradas analógicas y mixtas.

Características y Ventajas de VHDL

- Sirve como herramienta de diseño lógico, posibilitando la documentación de los proyectos y su reutilización.
- Sirve como herramienta de especificación de proyectos.
- Permite generar proyectos con estructura del tipo jerárquica.
- Posibilita modelizar el concepto de tiempo.
- Permite describir módulos con acciones que serán evaluadas luego en forma secuencial.
- Permite la parametrización de componentes y portabilidad de los diseños para independizarse de la tecnología.
- Permite implementación de test-bench para simulación de de diseños.

Diagrama de Flujo en el Diseño con VHDL



VHDL es un lenguaje portable y reusable ya que es independiente de la tecnología ó fabricante (Xilinx , Intel (ex-Altera), Microsemi (ex-Actel), QuickLogic, etc.).
Sus sentencias a diferencia de un programa de software se ejecutan inherentemente en forma “concurrente” salvo aquellas que se incluyan dentro de un Procedimiento (Procedure), Proceso (Process) ó Función (Function) donde se ejecutarán en forma secuencial.

Software para diseño:

Existen varias herramientas EDA (Electronic Design Automation) para la síntesis, implementación y simulación de circuitos digitales.
Algunas las proveen los fabricantes de chips:
Quartus II de Altera (ahora INTEL FPGA),
ISE suite de Xilinx, etc.
Otras son de terceras partes por ejemplo los sintetizadores:
Leonardo Spectrum de Mentor Graphics,
Synplify de Synplicity,
ModelSim de Model Technology,
etc.

TIPOS y SUBTIPOS en VHDL

Escalares: Por enumeración, enteros, de punto flotante y físicos.
Compuestos: Matrices y vectores. Ej: (is array .... of).
De acceso: punteros.
Archivos: Para manejo de Entrada-salida en archivos.

Ejemplo de tipos:

```
type word is array (0 to 31) of BIT;
type byte is array (NATURAL range 7 downto 0) of BIT;
type MVL4 is (`X´, `0´, `1´, `Z´);
```

Los subtipos son casos restringuidos de TIPOS.
Ejemplo el subtipo “nibble” del tipo “byte”
(subtype nibble is byte (3 downto 0);.

LITERALES

- Enteros.
- Punto flotante.
- Literales físicos (ejemplo ns).
- Bases (ejemplo 2#1011#, 8#17#, 16#9FD#).
- Caracteres ASCII (ejemplo `M´, `5´).
- Cadena de caracteres (ejemplo: “esto es un string”).
- otros.

CONSTANTES, VARIABLES y SEÑALES

- Constantes (ejemplo: CONSTANT retardo: tiem:=3ns;)

- Variables: Locales en un proceso. No salen del entorno de declaración en procesos ó subprogramas. Son ejecutadas secuencialmente.

- Señales: Se modifican mediante sentencias de asignación pero no se hace efectivo hasta que todos los procesos terminan. Son de uso global.

OPERADORES EN VHDL

- Lógicos: AND, OR, NAND, NOR, XOR, XNOR, NOT para tipos BIT, BOOLEAN y arrays de ellos.
- Relacionales: =, /=, <, <=, >, >= donde los operandos deben ser del mismo tipo y el resultado es BOOLEAN.
- Shift: SLL, SRL, SLA, SRA, ROL, ROR donde el operando izquierdo debe ser BIT ó BOOLEAn y el derecho INTEGER.
- Suma y resta: + y -.
- MULT y DIV: *, /, MOD y REM.
- miscelaneos: exponenciación (**) y valor absoluto (ABS).

Los comentarios comienzan con doble línea "- -".
Las sentiencias terminan con ";".

IDENTIFICADORES

No hay longitud máxima.
No hay diferencia entre mayúsculas y minúsculas.
Deben empezar con caracter alfabético.
No pueden terminar con underline.

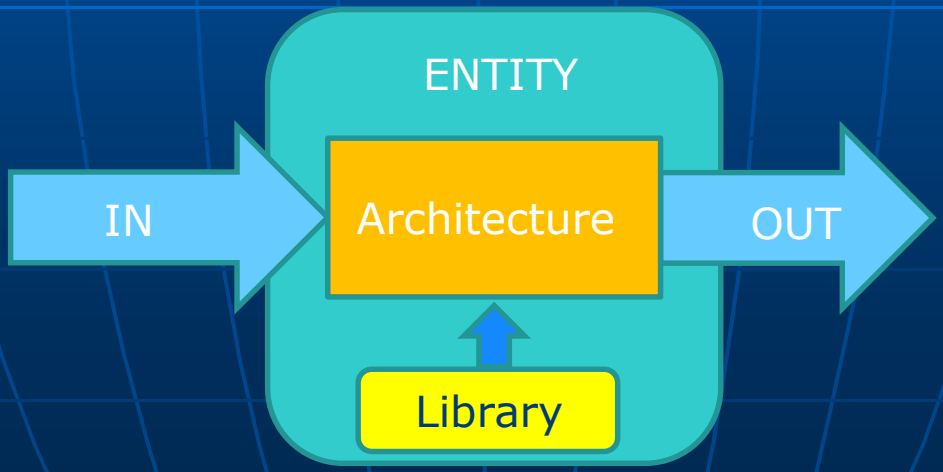
Estructuras en VHDL:

Entity: Define la vista externa de un modelo.

Architecture: Define una posible funcionalidad de un modelo.

Library: Contiene un listado de todas las librerías utilizadas en el diseño.

Package: Es una forma para almacenar y usar información útil que describe a un modelo (relacionada con Library).



PUERTOS EN VHDL

Los puertos de una entidad se declaran con la palabra PORT seguida de una lista formal de señales.
Cada señal ó grupo de señales de igual tipo se define con su "identificador", su modo de operación (in, out, inout, buffer), y un eventual valor por default en caso de ser del tipo "in" ó "out", que queden sin conectar.

Un puerto "in" puede ser leído pero no modificado.
Un puerto "out" puede ser modificado pero no leído.
Un puerto "buffer" es una salida siempre activa.
Un puerto "inout" se asocia a una compuerta bidireccional (por ejemplo con TRISTATE).

Entidad (Entity)

Una entidad VHDL especifica el nombre de la entidad , sus puertos y toda aquella información relacionada con ella.

Ejemplo:

```
ENTITY mux IS
        PORT ( a, b, c, d : IN BIT;
                    s0, s1 : IN BIT;
                    z : OUT BIT);
END mux;
```

En este ejemplo Entity describe la interface con el mundo externo de un multiplexor, el número, tipos puertos empleados y dirección de los mismos (entrada ó salida).

NADA sobre COMO funciona o QUE hace se ha especificado aún.

Arquitectura (Architecture)

CASO SINTETIZABLE

La arquitectura en VHDL describe la funcionalidad de la entidad y contiene la información que modela el comportamiento de la misma. En este caso permite la síntesis.

```
ENTITY mux IS
        PORT ( a, b, c, d : IN BIT;
                    s0, s1 : IN BIT;
                z : OUT BIT);
END mux;

ARCHITECTURE dataflow of mux IS
        SIGNAL select : INTEGER;
 BEGIN
   select <= 0 WHEN s0 = '0' AND s1 = `0´ ELSE
             1 WHEN s0 = '1' AND s1 = `0´ ELSE
             2 WHEN s0 = '0' AND s1 = `1´ ELSE
             3 ;

   z <=   a WHEN select = 0 ELSE
          b WHEN select = 1 ELSE
          c WHEN select = 2 ELSE
          d ;
 END dataflow;
```

Aquí especificamos QUE HACE el hardware existiendo diferentes maneras para hacerlo

ESTA FORMA DE DESCRIPCIÓN SE USA FUNDAMENTALMENTE EN SINTESIS DE HARDWARE.

Arquitectura (Architecture)

CASO **NO** SINTETIZABLE

La arquitectura en VHDL describe la funcionalidad de la entidad y contiene la información que modela el comportamiento de la misma. En este caso sólo sirve para verificación.

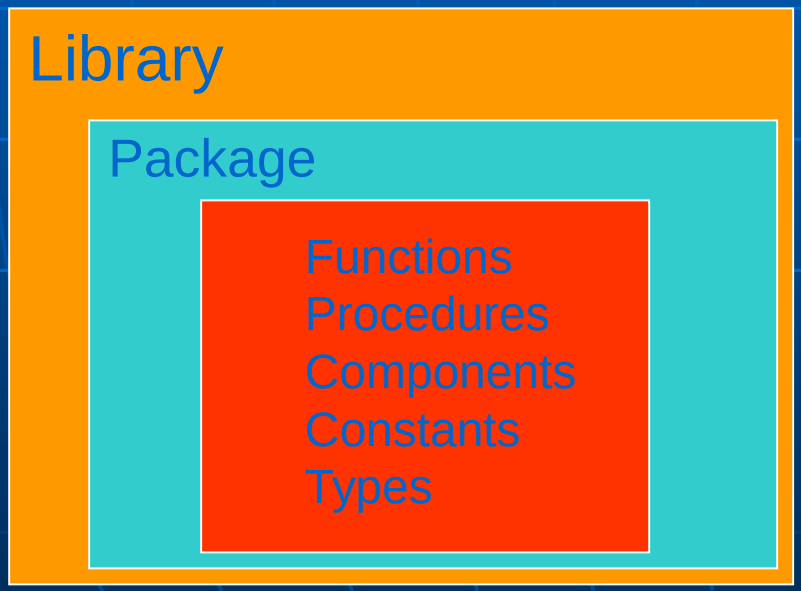
```
ENTITY mux IS
        PORT ( a, b, c, d : IN BIT;
                  s0, s1 : IN BIT;
                z : OUT BIT);
END mux;

ARCHITECTURE dataflow of mux IS
        SIGNAL select : INTEGER;
BEGIN
  select <= 0 WHEN s0 = '0' AND s1 = `0´ ELSE
            1 WHEN s0 = '1' AND s1 = `0´ ELSE
            2 WHEN s0 = '0' AND s1 = `1´ ELSE
            3 ;

  z <=    a  AFTER 0.5 NS  WHEN select = 0 ELSE
          b  AFTER 0.5 NS  WHEN select = 1 ELSE
          c  AFTER 0.5 NS  WHEN select = 2 ELSE
          d  AFTER 0.5 NS;
END dataflow;
```

IMPORTANTE:
ESTE EJEMPLO ES ANALOGO AL ANTERIOR EXCEPTO QUE SE HACE REFERENCIA TEMPORAL EN LA ACTUALIZACIÓN DE LAS SEÑALES.
ESTA MANERA DE DESCRIPCIÓN SE USA FUNDAMENTALMENTE EN SIMULACIÓN Y NO PUEDE SER SINTETIZABLE YA QUE LOS RETARDOS LOS DEFINE EL TIPO DE CHIP A EMPLEAR.

Una librería es una colección de piezas de código usualmente empleadas. Esto permite poder reusar esas piezas ó compartirlas con otros diseños.

Sintáxis:
LIBRARY <nombre de la librería>;
USE <nombre de un package>;

Ejemplo: LIBRARY ieee;
USE ieee.std_logic_1164;



El código es escrito en forma de Funciones (Functions), Procesos (Process), Procedimientos (Procedures) ó Componentes (Components) y luego ubicados dentro de Paquetes (Packages) para ser compilados dentro de la Librería destino.

Librerías mas comunes del paquete VHDL actualizado en 1993:

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;
```

LIBRARY std;

Librería que no requiere ser declarada en un diseño.
Contiene declaraciones de tipos de datos y funciones de entrada-salida de texto entre otros.

```
USE std.standard.all;
USE std.textio.all;
```

Standard: donde se definen los tipos lógicos y numéricos básicos
TEXTIO: Define tipos para la creación de texto y procedimeintos para el ingreso e impresión de textos.

LIBRARY work;

```
USE work.all;
```

Librería que no requiere ser declarada en un diseño.
Es donde se salvan todos los archivos relacionados con el diseño en curso (creados por el compilador, simulador, etc.).

USE ieee.std_logic_1164:
Especifica el STD_LOGIC (8 niveles) y el STD_ULOGIC (9 niveles) para sistemas lógicos multinivel.
De todos estos niveles sólo 3 son sintetizables sin restricciones; el resto sirven para simulación.

USE ieee.std_logic_arith:
Especifica tipos de datos con y sin signo, operaciones aritméticas y de comparación numérica y funciones para conversión de datos.

USE ieee.std_logic_signed:
Permite operaciones con signo con datos tipo STD_LOGIC_VECTOR.

USE ieee.std_logic_unsigned:
Permite operaciones sin signo con datos tipo STD_LOGIC_VECTOR.

Librería en VHDL: Contenido del archivo std_logic_arith.vhd

```
library IEEE;
use IEEE.std_logic_1164.ALL;

package std_logic_arith is

    type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
    type SIGNED is array (NATURAL range <>) of STD_LOGIC;
    subtype SMALL_INT is INTEGER range 0 to 1;

    function "+"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
    function "+"(L: SIGNED; R: SIGNED) return SIGNED;
    function "+"(L: UNSIGNED; R: SIGNED) return SIGNED;
    function "+"(L: SIGNED; R: UNSIGNED) return SIGNED;
    function "+"(L: UNSIGNED; R: INTEGER) return UNSIGNED;
    function "+"(L: INTEGER; R: UNSIGNED) return UNSIGNED;
    function "+"(L: SIGNED; R: INTEGER) return SIGNED;
    function "+"(L: INTEGER; R: SIGNED) return SIGNED;
    function "+"(L: UNSIGNED; R: STD_ULOGIC) return UNSIGNED;
    function "+"(L: STD_ULOGIC; R: UNSIGNED) return UNSIGNED;
    function "+"(L: SIGNED; R: STD_ULOGIC) return SIGNED;
    function "+"(L: STD_ULOGIC; R: SIGNED) return SIGNED;

    function "+"(L: UNSIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: SIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: UNSIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: SIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: UNSIGNED; R: INTEGER) return STD_LOGIC_VECTOR;
    function "+"(L: INTEGER; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: SIGNED; R: INTEGER) return STD_LOGIC_VECTOR;
    function "+"(L: INTEGER; R: SIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: UNSIGNED; R: STD_ULOGIC) return STD_LOGIC_VECTOR;
    function "+"(L: STD_ULOGIC; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: SIGNED; R: STD_ULOGIC) return STD_LOGIC_VECTOR;
    function "+"(L: STD_ULOGIC; R: SIGNED) return STD_LOGIC_VECTOR;

    function "-"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
    function "-"(L: SIGNED; R: SIGNED) return SIGNED;
    function "-"(L: UNSIGNED; R: SIGNED) return SIGNED;
    function "-"(L: SIGNED; R: UNSIGNED) return SIGNED;
    function "-"(L: UNSIGNED; R: INTEGER) return UNSIGNED;
    function "-"(L: INTEGER; R: UNSIGNED) return UNSIGNED;
    function "-"(L: SIGNED; R: INTEGER) return SIGNED;
    function "-"(L: INTEGER; R: SIGNED) return SIGNED;
    function "-"(L: UNSIGNED; R: STD_ULOGIC) return UNSIGNED;
    function "-"(L: STD_ULOGIC; R: UNSIGNED) return UNSIGNED;
    function "-"(L: SIGNED; R: STD_ULOGIC) return SIGNED;
    function "-"(L: STD_ULOGIC; R: SIGNED) return SIGNED;

    function "-"(L: UNSIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: SIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: UNSIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: SIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: UNSIGNED; R: INTEGER) return STD_LOGIC_VECTOR;
    function "-"(L: INTEGER; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: SIGNED; R: INTEGER) return STD_LOGIC_VECTOR;
    function "-"(L: INTEGER; R: SIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: UNSIGNED; R: STD_ULOGIC) return STD_LOGIC_VECTOR;
    function "-"(L: STD_ULOGIC; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: SIGNED; R: STD_ULOGIC) return STD_LOGIC_VECTOR;
    function "-"(L: STD_ULOGIC; R: SIGNED) return STD_LOGIC_VECTOR;

    function "+"(L: UNSIGNED) return UNSIGNED;
    function "+"(L: SIGNED) return SIGNED;
    function "-"(L: SIGNED) return SIGNED;
    function "ABS"(L: SIGNED) return SIGNED;

    function "+"(L: UNSIGNED) return STD_LOGIC_VECTOR;
    function "+"(L: SIGNED) return STD_LOGIC_VECTOR;
    function "-"(L: SIGNED) return STD_LOGIC_VECTOR;
    function "ABS"(L: SIGNED) return STD_LOGIC_VECTOR;

    function "*"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
    function "*"(L: SIGNED; R: SIGNED) return SIGNED;
    function "*"(L: SIGNED; R: UNSIGNED) return SIGNED;
    function "*"(L: UNSIGNED; R: SIGNED) return SIGNED;

    function "*"(L: UNSIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "*"(L: SIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
    function "*"(L: SIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
    function "*"(L: UNSIGNED; R: SIGNED) return STD_LOGIC_VECTOR;

    function "<"(L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
    function "<"(L: SIGNED; R: SIGNED) return BOOLEAN;
    function "<"(L: UNSIGNED; R: SIGNED) return BOOLEAN;
    function "<"(L: SIGNED; R: UNSIGNED) return BOOLEAN;
    function "<"(L: UNSIGNED; R: INTEGER) return BOOLEAN;
    function "<"(L: INTEGER; R: UNSIGNED) return BOOLEAN;
    function "<"(L: SIGNED; R: INTEGER) return BOOLEAN;
    function "<"(L: INTEGER; R: SIGNED) return BOOLEAN;
```

Librería en VHDL: Contenido del archivo std_logic_arith.vhd (continuación)

```
function "<="(L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
function "<="(L: SIGNED; R: SIGNED) return BOOLEAN;
function "<="(L: UNSIGNED; R: SIGNED) return BOOLEAN;
function "<="(L: SIGNED; R: UNSIGNED) return BOOLEAN;
function "<="(L: UNSIGNED; R: INTEGER) return BOOLEAN;
function "<="(L: INTEGER; R: UNSIGNED) return BOOLEAN;
function "<="(L: SIGNED; R: INTEGER) return BOOLEAN;
function "<="(L: INTEGER; R: SIGNED) return BOOLEAN;

function ">"(L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
function ">"(L: SIGNED; R: SIGNED) return BOOLEAN;
function ">"(L: UNSIGNED; R: SIGNED) return BOOLEAN;
function ">"(L: SIGNED; R: UNSIGNED) return BOOLEAN;
function ">"(L: UNSIGNED; R: INTEGER) return BOOLEAN;
function ">"(L: INTEGER; R: UNSIGNED) return BOOLEAN;
function ">"(L: SIGNED; R: INTEGER) return BOOLEAN;
function ">"(L: INTEGER; R: SIGNED) return BOOLEAN;

function ">="(L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
function ">="(L: SIGNED; R: SIGNED) return BOOLEAN;
function ">="(L: UNSIGNED; R: SIGNED) return BOOLEAN;
function ">="(L: SIGNED; R: UNSIGNED) return BOOLEAN;
function ">="(L: UNSIGNED; R: INTEGER) return BOOLEAN;
function ">="(L: INTEGER; R: UNSIGNED) return BOOLEAN;
function ">="(L: SIGNED; R: INTEGER) return BOOLEAN;
function ">="(L: INTEGER; R: SIGNED) return BOOLEAN;

function "="(L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
function "="(L: SIGNED; R: SIGNED) return BOOLEAN;
function "="(L: UNSIGNED; R: SIGNED) return BOOLEAN;
function "="(L: SIGNED; R: UNSIGNED) return BOOLEAN;
function "="(L: UNSIGNED; R: INTEGER) return BOOLEAN;
function "="(L: INTEGER; R: UNSIGNED) return BOOLEAN;
function "="(L: SIGNED; R: INTEGER) return BOOLEAN;
function "="(L: INTEGER; R: SIGNED) return BOOLEAN;

function "/="(L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
function "/="(L: SIGNED; R: SIGNED) return BOOLEAN;
function "/="(L: UNSIGNED; R: SIGNED) return BOOLEAN;
function "/="(L: SIGNED; R: UNSIGNED) return BOOLEAN;
function "/="(L: UNSIGNED; R: INTEGER) return BOOLEAN;
function "/="(L: INTEGER; R: UNSIGNED) return BOOLEAN;
function "/="(L: SIGNED; R: INTEGER) return BOOLEAN;
function "/="(L: INTEGER; R: SIGNED) return BOOLEAN;

function SHL(ARG: UNSIGNED; COUNT: UNSIGNED) return UNSIGNED;
function SHL(ARG: SIGNED; COUNT: UNSIGNED) return SIGNED;
function SHR(ARG: UNSIGNED; COUNT: UNSIGNED) return UNSIGNED;
function SHR(ARG: SIGNED; COUNT: UNSIGNED) return SIGNED;

function CONV_INTEGER(ARG: INTEGER) return INTEGER;
function CONV_INTEGER(ARG: UNSIGNED) return INTEGER;
function CONV_INTEGER(ARG: SIGNED) return INTEGER;
function CONV_INTEGER(ARG: STD_ULOGIC) return INTEGER;

function CONV_UNSIGNED(ARG: INTEGER; SIZE: INTEGER) return UNSIGNED;
function CONV_UNSIGNED(ARG: UNSIGNED; SIZE: INTEGER) return UNSIGNED;
function CONV_UNSIGNED(ARG: SIGNED; SIZE: INTEGER) return UNSIGNED;
function CONV_UNSIGNED(ARG: STD_ULOGIC; SIZE: INTEGER) return UNSIGNED;

function CONV_SIGNED(ARG: INTEGER; SIZE: INTEGER) return SIGNED;
function CONV_SIGNED(ARG: UNSIGNED; SIZE: INTEGER) return SIGNED;
function CONV_SIGNED(ARG: SIGNED; SIZE: INTEGER) return SIGNED;
function CONV_SIGNED(ARG: STD_ULOGIC; SIZE: INTEGER) return SIGNED;

function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
                               return STD_LOGIC_VECTOR;
function CONV_STD_LOGIC_VECTOR(ARG: UNSIGNED; SIZE: INTEGER)
                               return STD_LOGIC_VECTOR;
function CONV_STD_LOGIC_VECTOR(ARG: SIGNED; SIZE: INTEGER)
                               return STD_LOGIC_VECTOR;
function CONV_STD_LOGIC_VECTOR(ARG: STD_ULOGIC; SIZE: INTEGER)
                               return STD_LOGIC_VECTOR;
------------------------------------------------------------------
-- attributes for conversion functions
------------------------------------------------------------------
attribute Synth_Conversion_Function of CONV_INTEGER: function is "INTEGER";
attribute Synth_Conversion_Function of CONV_UNSIGNED: function is "UNSIGNED";
attribute Synth_Conversion_Function of CONV_SIGNED: function is "SIGNED";
attribute Synth_Conversion_Function of CONV_STD_LOGIC_VECTOR: function is "STD_LOGIC_VECTOR";
------------------------------------------------------------------
   -- zero extend STD_LOGIC_VECTOR (ARG) to SIZE,
-- SIZE < 0 is same as SIZE = 0
-- returns STD_LOGIC_VECTOR(SIZE-1 downto 0)
function EXT(ARG: STD_LOGIC_VECTOR; SIZE: INTEGER) return STD_LOGIC_VECTOR;

-- sign extend STD_LOGIC_VECTOR (ARG) to SIZE,
-- SIZE < 0 is same as SIZE = 0
-- return STD_LOGIC_VECTOR(SIZE-1 downto 0)
function SXT(ARG: STD_LOGIC_VECTOR; SIZE: INTEGER) return STD_LOGIC_VECTOR;

end std_logic_arith;
```

STD_ULOGIC

Generalmente utilizado para simulación.
Posee 9 niveles diferentes de valores.

'U': No inicializado. Esta señal no ha sido definida todavía.
'X': Desconocido. Imposible de determinar su valor o resultado.
'0': Nivel lógico 0.
'1': Nivelo lógico 1.
'Z': Alta Impedancia.
'W': Señal Débil, no puede definirse como = o 1.
'L': Señal Débil que debería ir a 0.
'H': Señal Débil que debería ir a 1.
'-': Don't care.

STD_LOGIC

Generalmente utilizado para síntesis y simulación.
Posee 3 niveles diferentes de valores.

'0': Nivel lógico 0.
'1': Nivelo lógico 1.
'Z': Alta Impedancia.

Tipos de descripciones en VHDL

En HDL se describen en general sucesos inherentemente concurrentes pues en la arquitectura de una entidad (entre BEGIN y END) se definen sentencias concurrentes.
Sin embargo en VHDL aparece la noción de PROCESOS (Process), que si bien describen eventos que se producen como cualquier sentencia concurrente, son analizados internamente en forma secuencial para su síntesis y/o simulación.
Es por eso que se encuentran sentencias de asignación propias de acciones concurrentes y otras exclusivas para procesos secuenciales.

Ejemplos:
Concurrentes: Declaración de señales.
Sentencias WHEN..ELSE, PROCESS, etc.
Secuenciales: Declaración de variables.
Sentencias IF..THEN..ELSE, CASE, LOOP, etc.
Ambas: Asignación a señales, declaración de tipos y constantes, sentencia ASSERT, retardos (AFTER), etc.

Tipos de descripciones en VHDL

CONCURRENTES	SECUENCIALES	CONCURRENTES Y SECUENCIALES
Declaración de señales. Sentencia WHEN ...ELSE. Sentencia WITH ...SELECT. Sentencia PROCESS. Sentencia BLOCK.	Declaración de variables Asignación a variables. Sentencia IF THEN ELSE Sentencia CASE. Sentencia FOR … LOOP. Sentencia RETURN. Sentencia NULL. Sentencia WAIT.	Asignación de señales. Declaración de Tipos y Ctes. Declaración de atributos de señales. Invocación a Funciones y Procedimientos. Sentencia ASSERT. Sentencia AFTER.

SEÑALES VERSUS VARIABLES

SEÑALES

Las señales representan vínculos físicos entre procesos concurrentes.
Se pueden declaran en:
Packages: al describir componentes.
Entity: al describir los puertos de una entidad.
Architecture: al definir líneas internas.

Cuando una señal es del tipo INTEGER es importante definir su rango, sino el compilador asignará 32 bits por default.

En el caso de inferirse registros se generará gran cantidad de recursos innecesarios.

Las señales tienen vigencia en toda la descripción del circuito.
Sirven de nexo para comunicación entre componentes.
Se actualizan dentro de un ‘Process’ al final.

VARIABLES

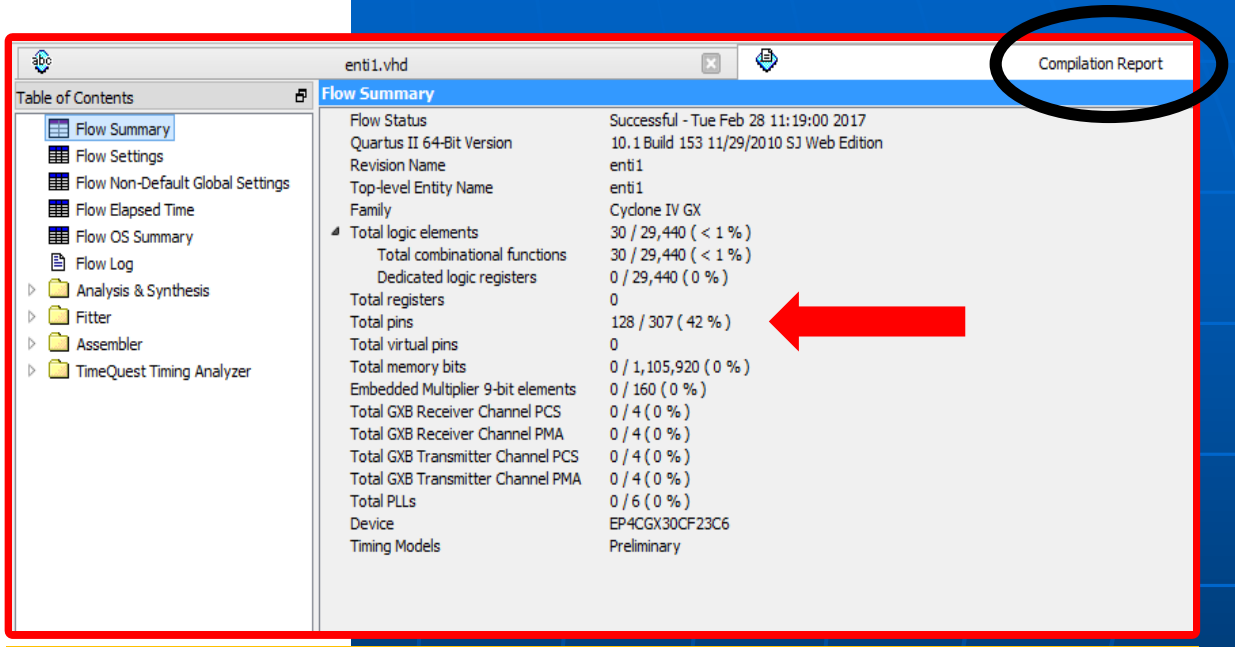
Las variables sólo tienen vigencia dentro del proceso donde están declaradas.
Proveen un mecanismo conveniente para almacenamiento local de información.
Se actualizan dentro de un ‘Process’ inmediatamente al invocarlas.
Utiles cuando se quiere escribir un código serializado
(Ver ejemplo de diseño de sumador ripple-carry con
FOR-LOOP ó FOR GENERATE)

SEÑALES

Cuando una señal es del tipo INTEGER es importante definir su rango, sino el compilador asignará 32 bits por default.

enti1.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity enti1 is
6
7     port
8     (
9        a1, b1   : in integer;
10       d1, e1   : out integer
11    );
12
13 end entity;
14
15 architecture maldef of enti1 is
16 SIGNAL a,b,c,d,e: integer := 0;
17 BEGIN
18  PROCESS (a,b,c)
19  BEGIN
20     c <= a;
21     d <= c + 12;
22     c <= b;
23     e <= c + 12;
24  END PROCESS;
25
26  a <= a1;
27  b <= b1;
28  d1 <= d;
29  e1 <= e;
30
31 END maldef;
32
```



Dos entradas (a1 y b1) y dos salidas (d1 y e1) especificadas como ENTEROS.
El compilador salvo alguna inicialización, las define como de 32 bits cada una (128pines /32 = 4 ports).

Una Solución es definir PORTS como arreglos de bits BIT_VECTOR ó STD_LOGIC_VECTOR).

TIPOS DE DISEÑO EN VHDL

ESTRUCTURAL:
En forma similar a las herramientas de diseño que trabajan con lenguajes de NETLIST, VHDL puede ser utilizado para diseñar ó simular un sistema digital, especificando por un lado sus componentes y por el otro sus interconexiones.

POR COMPORTAMIENTO (ó FUNCIONAL):
VHDL puede ser usado para diseñar un sistema digital, describiendo el comportamiento del mismo a través de dos formas diferentes: "algorítmica" y por "flujo de datos".
Esta modalidad es muy utilizada en procesos de simulación ya que permite simular un sistema sin necesidad de conocer su estructura interna.

DISEÑO ESTRUCTURAL

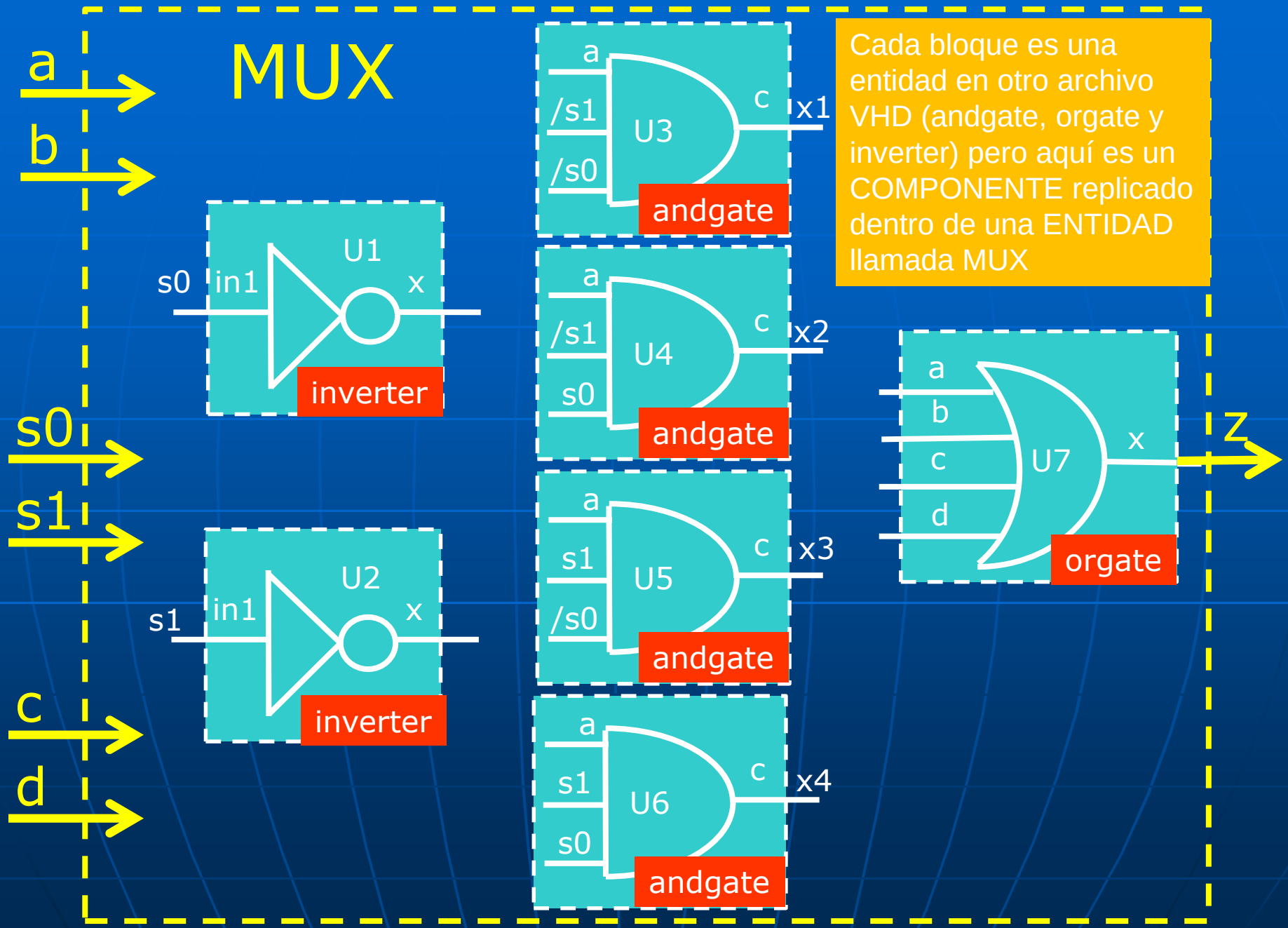
Es una forma de diseñar instanciando subcomponentes que realizan operaciones mas pequeñas del modelo completo.

Ejemplo:
Diseño del mismo mux 4:1 pero con una descripción “estructural” de la arquitectura que ahora denominaremos “netlist”.
La misma está formada por compuertas de tres tipos diferentes (andgate, inverter y orgate) interconectadas convenientemente.
Cada tipo de compuerta está especificado como un COMPONENTE diferente.
“andgate” es de 3 puertos de entrada y uno de salida.
“orgate” es de 4 puertos de entrada y uno de salida.
“inverter” es de un puerto de entrada y uno de salida.

La forma de describir que hace cada compuertas está en la sección: BEGIN ..... END de la estructura “netlist”;

EJEMPLO DE DISEÑO ESTRUCTURAL

```
ARCHITECTURE netlist OF mux IS
        COMPONENT andgate
                PORT(a, b, c: IN BIT; x : OUT BIT);
        END COMPONENT;
        COMPONENT inverter
                PORT(in1 : IN BIT; x : OUT BIT);
        END COMPONENT;
        COMPONENT orgate
                PORT(a, b, c, d : IN BIT; x : OUT BIT);
        END COMPONENT;

SIGNAL s0_inv, s1_inv, x1, x2, x3, x4 : BIT;

BEGIN
  U1 : inverter (s0, s0_inv);
  U2 : inverter (s1, s1_inv);
  U3 : andgate(a, s0_inv, s1_inv, x1);
  U4 : andgate(b, s0, s1_inv, x2);
  U5 : andgate(c, s0_inv, s1_inv, x3);
  U6 : andgate(d, s0_inv, s1_inv, x4);
  U7 : orgate(x2 => b, x1 => a, x4 => d, x3 => c, x => z);
END netlist;
```


MUX
a
b
s0
s1
c
d
U1
s0
in1
x
inverter
U2
s1
in1
x
inverter
U3
a
/s1
/s0
c
x1
andgate
U4
a
/s1
s0
c
x2
andgate
U5
a
s1
/s0
c
x3
andgate
U6
a
s1
s0
c
x4
andgate
U7
a
b
c
d
x
orgate
z
Cada bloque es una entidad en otro archivo VHD (andgate, orgate y inverter) pero aquí es un COMPONENTE replicado dentro de una ENTIDAD llamada MUX

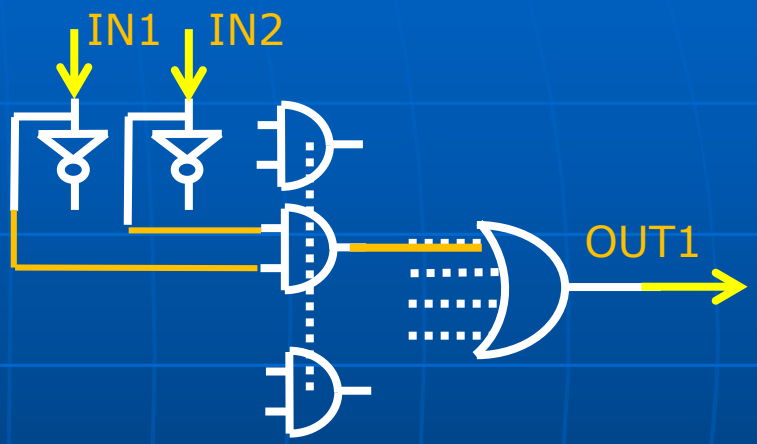
DISEÑO POR FLUJO DE DATOS

Ejemplo de compuerta AND de 2 entradas

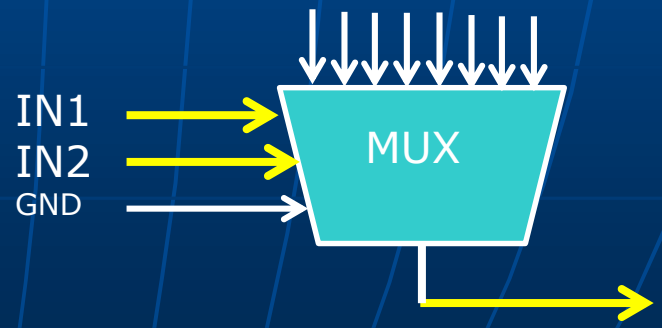
and2.vhd - Text Editor

```
library IEEE;
use IEEE.std_logic_1164.all;

-- Sección de ENTITY
entity and2 is
   port (
         IN1 : in std_logic;
         IN2 : in std_logic;
         OUT1: out std_logic);
end and2;

--Sección de ARCHITECTURE

architecture RTL of and2 is
begin

  OUT1 <= IN1 and IN2;

end RTL;
```

La síntesis en una EPLD será algo así:



La síntesis en una FPGA será algo así:



DISEÑO ALGORÍTMICO

Otra forma de describir la funcionalidad de un dispositivo es la de hacerlo algorítmicamente dentro de una sentencia "PROCESS".
La sentencia PROCESS consta de una serie de partes:
La primera es la lista de sensibilidad.
La segunda la declarativa
La tercera es la de statement. Comienza en "BEGIN".... .

Ejemplo:
Diseño del mismo multiplexor "mux" en una arquitectura ahora denominada "secuencial" donde la misma contiene sólo una sentencia "PROCESS....END PROCESS".
(a, b, c, d, s0, s1) es la lista de sensibilidad.
"sel" es una variable local que se declara (similar a NODE en AHDL).
La ejecución de la sentencia PROCESS comienza en "BEGIN" y termina en "END PROCESS".
En esta sección se han utilizado las funciones "IF" y "CASE" para describir el comportamiento del multiplexor.

Process (.....lista de sensitividad....)

Se activa cada vez que cambia el estado de una de las entradas descriptas en la "lista de sensitividad".
Al variar cualquiera de ellas, se Ejecuta todo su contenido y termina, a la espera de un próximo cambio en el estado de las señales «sensitivas».
Si no hay lista de sensitividad debe ponerse una sentencia WAIT para que no se ejecute indefinidamente.
Las declaraciones de TIPOS, FUNCIONES, PROCEDIMIENTOS y VARIABLES son locales al Proceso.
Las funciones admitidas son IF, CASE y FOR … LOOP.

Puede describir circuitos «COMBINATORIOS» como «SECUENCIALES».

Las SEÑALES son la interface entre el dominio concurrente y el secuencial dentro de un proceso. Mientras un proceso está ejecutándose, las señales permanecen inalterables (como si fueran constantes).

Las VARIABLES en cambio, pueden modificarse dentro de un Process cuando está corriendo.

DISEÑO ALGORITMICO

Ejemplo multiplexor 2:1 con sentencia CASE

mux2_ejemplo2.vhd - Text Editor

```
ENTITY mux2_ejemplo2 IS
    PORT
    (
        input0, input1, sel : IN  BIT;
        output              : OUT BIT
    );
END mux2_ejemplo2;

ARCHITECTURE mux2 OF mux2_ejemplo2 IS
BEGIN
process(input0, input1)
begin
  case sel is
    when '1'    => output <= input1;
    when others => output <= input0;
  end case;
end process;
END mux2;
```

DISEÑO ALGORITMICO

Ejemplo multiplexor 2:1 con sentencia IF..ELSE

mux2_ejemplo3.vhd - Text Editor

```
ENTITY mux2_ejemplo3 IS
    PORT
    (
        input0, input1, sel : IN  BIT;
        output              : OUT BIT
    );
END mux2_ejemplo3;

ARCHITECTURE mux3 OF mux2_ejemplo3 IS
BEGIN
  process(input0, input1)
    begin
        if sel = '1' then
            output <= input1;
        else
            output <= input0;
        end if;
  end process;
END mux3;
```

DISEÑO POR FLUJO DE DATOS

Ejemplo de buffer tri_state octuple

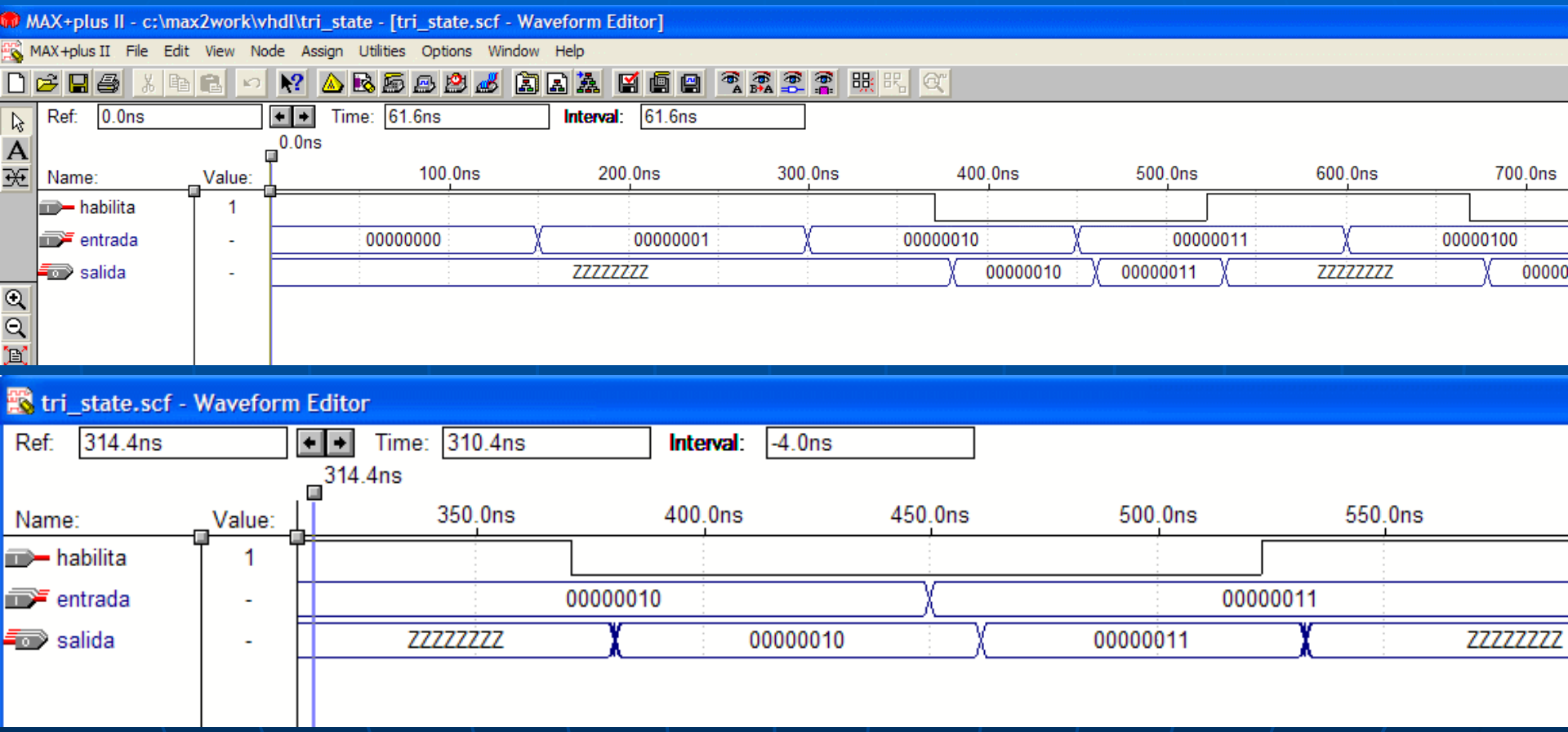
tri_state.vhd - Text Editor

```
-- buffer tri_state octuple
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tri_state IS PORT (
  entrada : IN std_logic_vector ( 7 DOWNTO 0);
  habilita : IN std_logic;
  salida : OUT std_logic_vector ( 7 DOWNTO 0));
END ENTITY tri_state;

ARCHITECTURE buf OF tri_state IS
 BEGIN
   salida <= entrada WHEN (habilita = '0') ELSE
                     (OTHERS => 'Z');
END ARCHITECTURE buf;
```

DISEÑO POR FLUJO DE DATOS

Ejemplo de buffer tri_state octuple

Simulación con el "waveform editor" del MAX+plus II



DISEÑO POR FLUJO DE DATOS

Ejemplo de multiplexor 2:1 con sentencia WHEN..ELSE

mux2_ejemplo1.vhd - Text Editor

```
ENTITY mux2_ejemplo1 IS
    PORT
    (
        input0, input1, sel : IN  BIT;
        output              : OUT BIT
    );
END mux2_ejemplo1;

ARCHITECTURE mux1 OF mux2_ejemplo1 IS
BEGIN

    output <= input0 WHEN sel = '0' ELSE input1;

END mux1;
```

DISEÑO POR FLUJO DE DATOS

Ejemplo de multiplexor 2:1 con sentencia WITH..SELECT

mux2_ejemplo4.vhd - Text Editor

```
ENTITY mux2_ejemplo4 IS
    PORT
    (
        input0, input1, sel : IN  BIT;
        output              : OUT BIT
    );
END mux2_ejemplo4;

ARCHITECTURE mux4 OF mux2_ejemplo4 IS
BEGIN

    with sel select output <= input1 when '1',
                              input0 when '0';

END mux4;
```

EJEMPLO de asignación concurrente con WITH..SELECT

```
-- decodificador BCD a 7 segmentos
ENTITY decobcda7s IS PORT (
  bcdin : IN INTEGER RANGE 0 TO 9;
  salida : OUT BIT_VECTOR ( 6 DOWNTO 0));
END ENTITY decobcda7s;

ARCHITECTURE deco OF decobcda7s IS
 BEGIN
 WITH bcdin SELECT
  salida <= B"1111110" WHEN 0, B"0110000" WHEN 1,
            B"1101101" WHEN 2, B"1111001" WHEN 3,
            B"0110011" WHEN 4, B"1011011" WHEN 5,
            B"1011111" WHEN 6, B"1110000" WHEN 7,
            B"1111111" WHEN 8, B"1111011" WHEN 9,
            B"0000000" WHEN OTHERS;
END ARCHITECTURE deco;
```

EJEMPLO de asignación concurrente con WITH..SELECT

deco2a4.vhd - Text Editor

```
-- decodificador 2 a 4
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY deco2a4 IS PORT (
  decoin : IN std_logic_vector (1 DOWNTO 0);
  salida : OUT std_logic_vector ( 3 DOWNTO 0));
END ENTITY deco2a4;

ARCHITECTURE decodi OF deco2a4 IS
 BEGIN
 WITH decoin SELECT
  salida <= "1000" WHEN "11",
                "0100" WHEN "10",
                "0010" WHEN "01",
                "0001" WHEN "00",
                "XXXX" WHEN OTHERS;
END ARCHITECTURE decodi;
```

PARA QUÉ SIRVEN LAS SEÑALES ?

Un ejemplo

```
entity and_nand is
      port (a, b : in bit;
            z, zbar : out bit);
end;
architecture illegal of and_nand is
begin
      z <= a and b;
      zbar <= not z;
end;
```

Así no puede ser usado ‘z’, para leer su estado y modificar a zbar, porque ‘z’ es una salida.

```
entity and_nand is
      port (a, b : in bit;
            z, zbar : out bit);
end;
architecture behaviour of and_nand is
      signal result : bit;
begin
      result <= a and b;
      z <= result;
      zbar <= not result;
end;
```

La solución es usar una ‘señal’ (result) para leer su estado y modificar zbar (y de paso a ‘z’ en este ejemplo).

CONVIENE SIEMPRE USAR SEÑALES Y AL FINAL DE «ARCHITECTURE» ASIGNARLAS A LOS PORTS DE SALIDA.

SEÑALES VERSUS VARIABLES

EJEMPLO

Definición de ´C´ como SEÑAL

enti1.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity enti1 is
6
7     port
8     (
9        a1, b1   : in integer;
10       d1, e1   : out integer
11    );
12
13 end entity;
14
15 architecture maldef of enti1 is
16    SIGNAL c, d, e : integer := 0;
17 BEGIN
18    PROCESS (a1,b1)
19    BEGIN
20       c <= a1;
21       d <= c + 12;
22       c <= b1;
23       e <= c + 12;
24    END PROCESS;
25
26    d1 <= d;
27    e1 <= e;
28
29 END maldef;
30
```

Defnición de ´C´ como VARIABLE

enti2.vhd

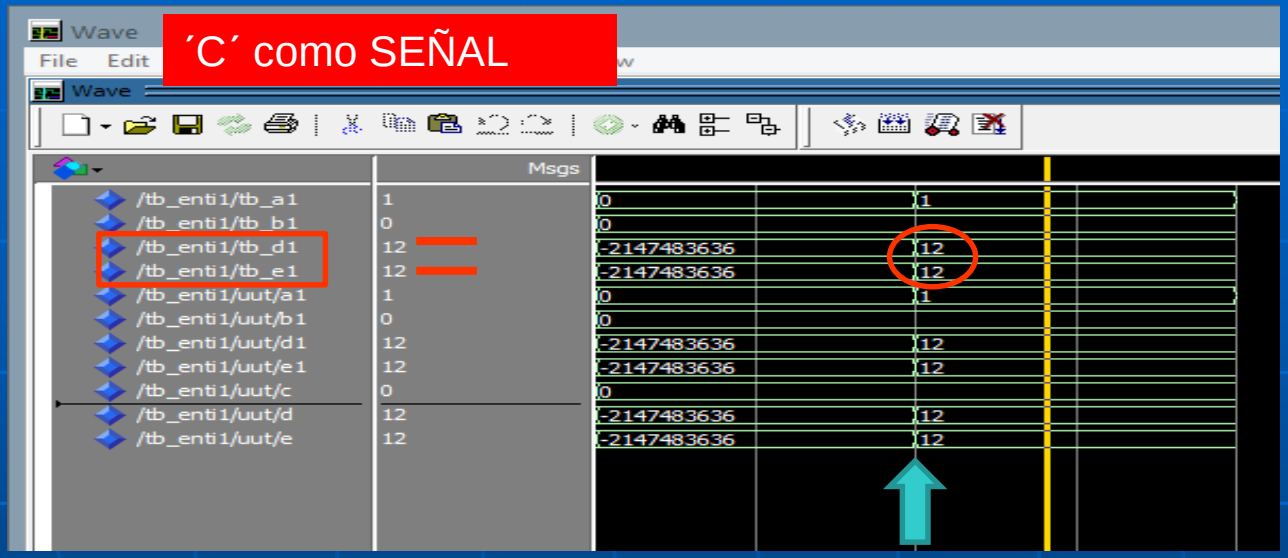
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity enti2 is
6
7     port
8     (
9        a1, b1   : in integer;
10       d1, e1   : out integer
11    );
12
13 end entity;
14
15 architecture biendef of enti2 is
16    SIGNAL d, e: integer := 0;
17 BEGIN
18    PROCESS (a1,b1)
19    variable c : integer := 0;
20    BEGIN
21       c := a1;
22       d <= c + 12;
23       c := b1;
24       e <= c + 12;
25
26    END PROCESS;
27
28    d1 <= d;
29    e1 <= e;
30
31 END biendef;
32
```

SEÑALES VERSUS VARIABLES

EJEMPLO (Continuación)

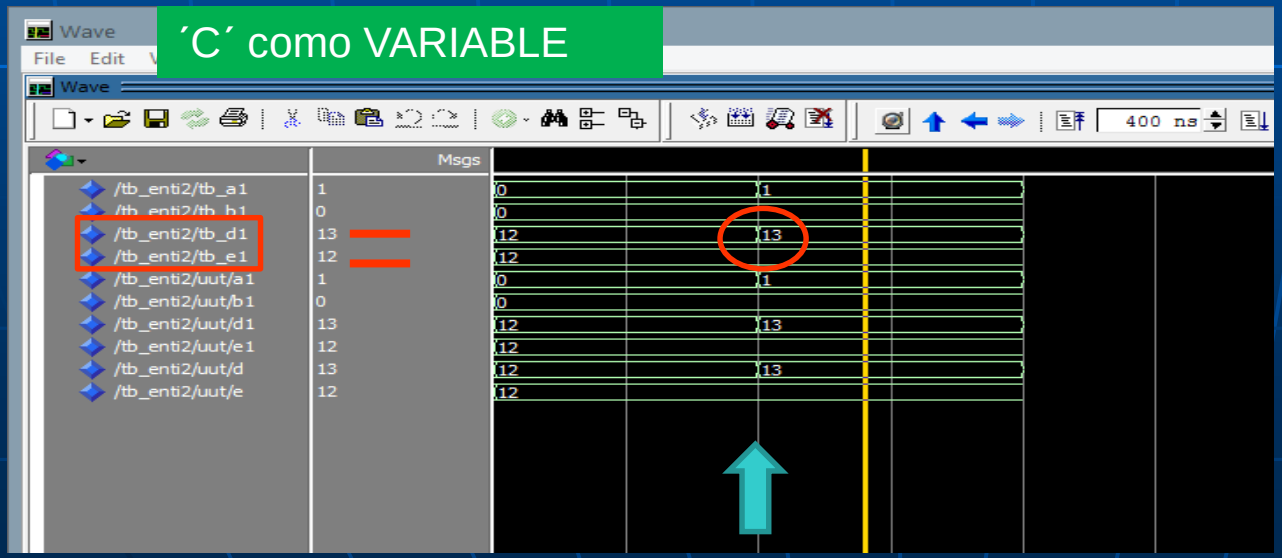
En t= 0 ns => a1 = 0 y b1 = 0.
En t= 200 ns => a1 = 1 y b1 = 0 (el cambio se muestra con la flecha ↑ .

´C´ como SEÑAL



d1 y e1 finalizan en 12.

´C´ como VARIABLE



e1 finaliza en 12
y
d1 finaliza en 13.

Ejemplo 1: DESCRIPCIÓN COMBINACIONAL CON PROCESS y señales

```
library ieee;
use ieee.std_logic_1164.all,
ieee.numeric_std.all;
entity sum is
  port (a, b : in unsigned(3 downto 0);
        sum : out unsigned(3 downto 0));
end;
architecture behaviour of sum is
  begin
```

CON LISTA DE SENSITIVIDAD

```
    process (a,b)
    begin
    sum <= a + b;
    end process;

end;
```

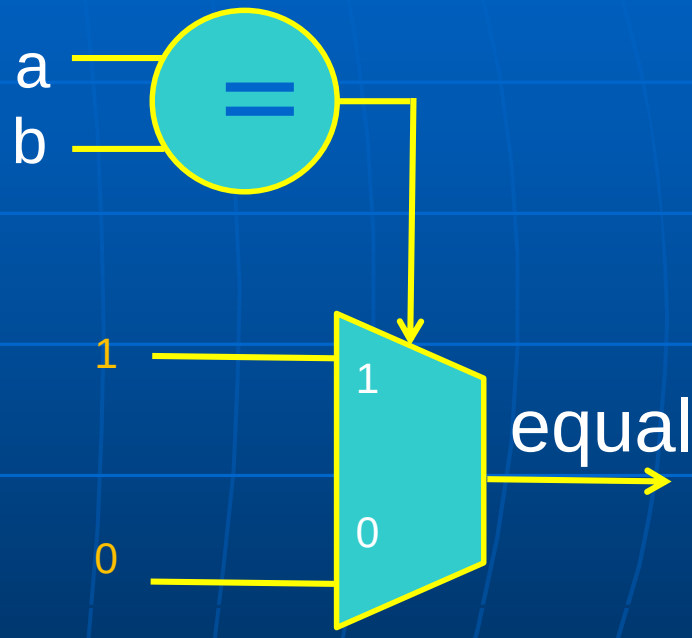
```
process
 begin
  sum <= a + b;
  wait on a, b;
end process;
```

SIN LISTA DE SENSITIVIDAD

WAIT puede estar al principio o al final en una descripción combinacional

Ejemplo 2: DESCRIPCIÓN COMBINACIONAL CON PROCESS y señales

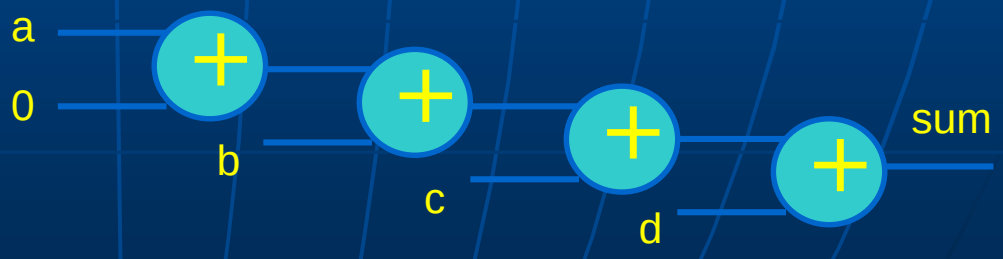
```
library ieee;
use ieee.std_logic_1164.all,
ieee.numeric_std.all;
entity compare is
   port (a, b : in unsigned (7 downto 0);
   equal : out std_logic);
end;
architecture behaviour of compare is
  begin
    process (a, b)
     begin
        if a = b then
           equal <= '1';
        else
           equal <= '0';
        end if;
     end process;
end;
```



Ejemplo 3: DESCRIPCIÓN COMBINACIONAL CON PROCESS y variables

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity adder is
    port (a, b, c, d : in signed(7 downto 0);
          result : out signed(7 downto 0));
end entity;

architecture behaviour of adder is
   begin
      four_add: process (a, b, c, d)
         variable sum : signed(7 downto 0);
      begin
         sum := a;
         sum := sum + b;
         sum := sum + c;
         sum := sum + d;
         result <= sum;
      end process;
end;
```

Cada vez que cambia alguna señal de entrada, se dispara el proceso y se recalcula la suma de a+b+c+d.



EJEMPLO de uso de LOOP (asignación secuencial)

Diseño de un sumador ripple-carry de 4 bits

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL;
ENTITY sumador IS
  GENERIC (n_bits : INTEGER :=4);
  PORT (a : IN std_logic_vector (n_bits DOWNTO 1);
        b : IN std_logic_vector (n_bits DOWNTO 1);
        c_in : IN std_logic;
        c_out : OUT std_logic;
        suma : std_logic_vector (n_bits DOWNTO 1));
END ENTITY sumador;

ARCHITECTURE sumador_rc OF sumador IS
BEGIN
   PROCESS (a, b, c_in)
     VARIABLE vsuma : std_logic_vector (n_bits DOWNTO 1);
     VARIABLE carry : std_logic;
    BEGIN
       carry := c_in;
         FOR i IN 1 TO n_bits LOOP
           vsuma(i) := a(i) XOR b(i) XOR carry;
           carry := (a(i) AND b(i)) OR (carry AND (a(i) OR b(i)));
         END LOOP;
         suma <= vsuma; c_out <= carry;
    END PROCESS;
END ARCHITECTURE sumador_rd;
```

EJEMPLO de uso de GENERATE

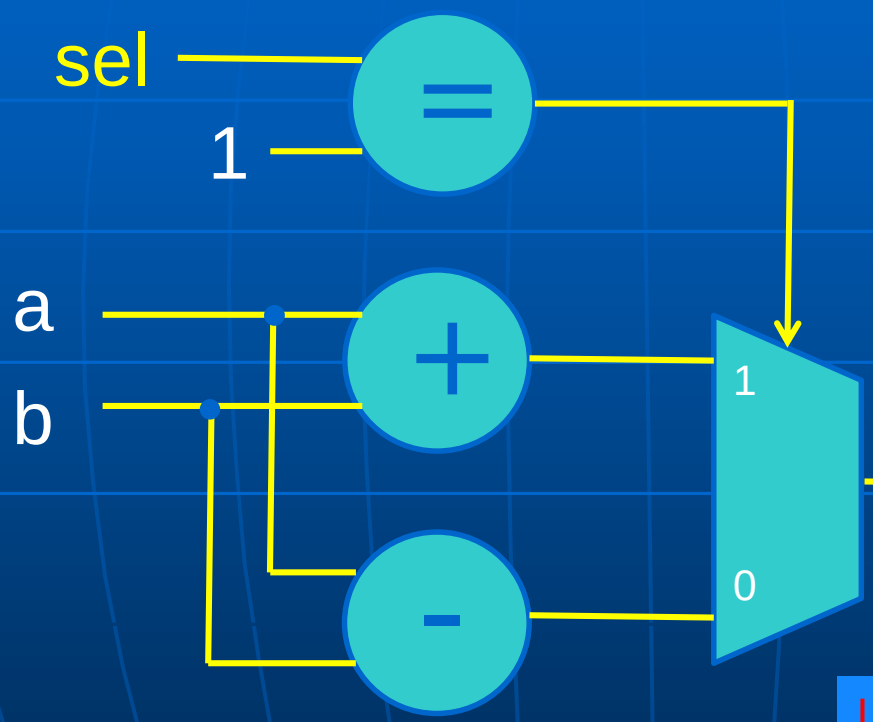
Diseño de un sumador ripple-carry de 4 bits

```
1  ENTITY sumador2 IS
2     PORT ( a,b : IN BIT_VECTOR (4 DOWNTO 1);
3            sum : OUT BIT_VECTOR(4 DOWNTO 1);
4            cout  : OUT BIT );
5
6  END ENTITY sumador2;
7
8  ARCHITECTURE sum_con_gen OF sumador2 IS
9
10 SIGNAL c: BIT_VECTOR (5 DOWNTO 1);
11
12    BEGIN
13       c(1) <= '0';
14       adders: FOR i IN 1 TO 4 GENERATE
15          sum(i) <= a(i) XOR b(i) XOR c(i);
16          c(i+1) <= (a(i) AND b(i)) OR (a(i) AND b(i))
17          OR (b(i) AND c(i));
18       END GENERATE;
19
20       cout <= c(5);
21
22 END ARCHITECTURE sum_con_gen;
```

Generate es una sentencia concurrente empleada usualmente para describir estructuras que tienen un patrón repetitivo en su diseño.

Resultado posible de una descripción antes de PLACE and ROUTE

```
sum <= a + b when sel = '1' else a - b;
```



Posible circuito basado en una pseudosíntesis con bloques genéricos

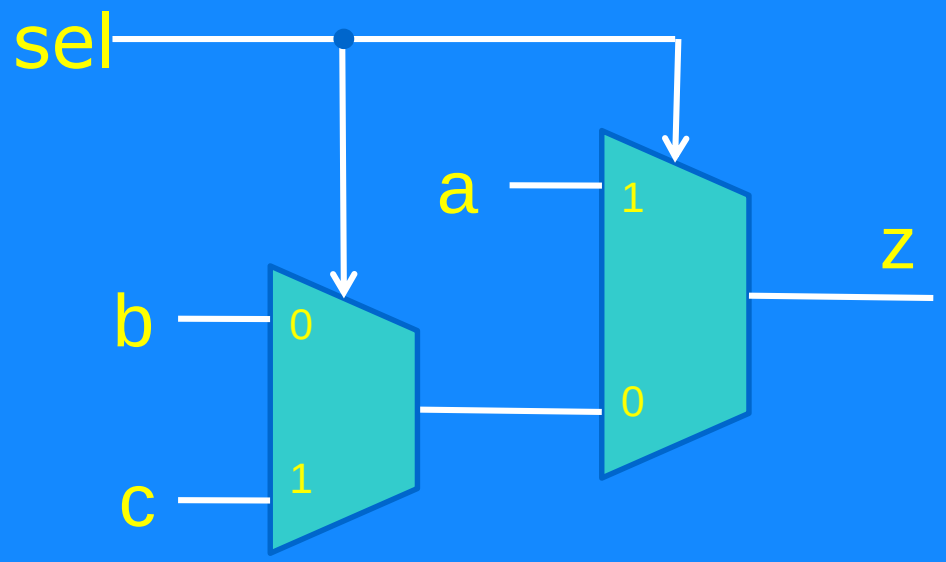
Una simulación a este nivel sólo dará información de si el circuito cumple con lo especificado funcionalmente

Luego del PLACE and ROUTE

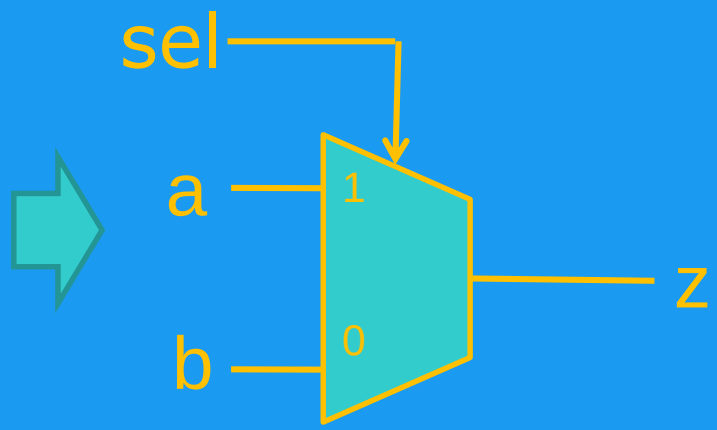
EN BASE A LOS RECURSOS QUE TENGA LA FPGA, A LA INTELIGENCIA DEL COMPILADOR Y EVENTUALES DIRECTIVAS DEL DISEÑADOR, SERÁ EL HARDWARE FINALMENTE IMPLEMENTADO. Aquí SI se tendrá información confiable del TIMING.

Ejemplo donde existe una mala descripción de un circuito.

```
z <= a when sel = '1' else
     b when sel = '0' else
     c;
```



AL COMPILAR … ESPERAMOS QUE SE MINIMICE A ESTO .....



Es muy probable que el COMPILADOR lo advierta y simplifique el circuito.

En casos mas complejos NO sabemos si el COMPILADOR lo hará.

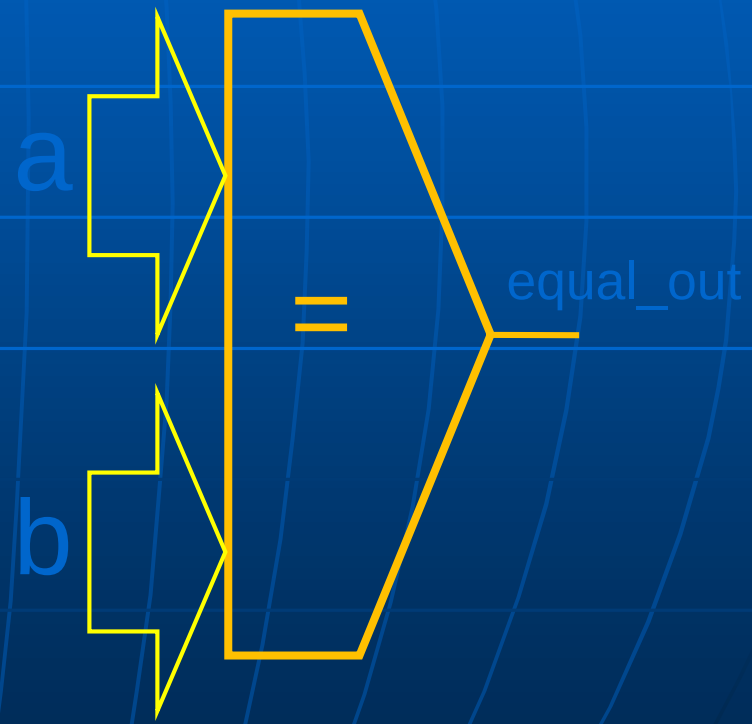
PROCESS: USO DE SENTENCIA FOR … LOOP

EJEMPLO: COMPARADOR DE IGUALDAD BIT A BIT

Circuito Combinatorio

```
library ieee;
use ieee.std_logic_1164.all;
entity match_bits is
port (a, b : in std_logic_vector(7 downto 0);
equal_out : out std_logic_vector(7 downto 0));
end;
```

```
architecture behaviour of match_bits is
   begin
     process (a, b)
       begin
          for i in 7 downto 0 loop
            equal_out(i) <= a(i) xnor b(i);
          end loop;
     end process;
end;
```



Otra manera de expresar el lazo:

```
for i in integer range 7 downto 0 loop
```

PROCESS: USO DE SENTENCIA FOR … LOOP

EJEMPLO: CONTADOR DE UNOS.

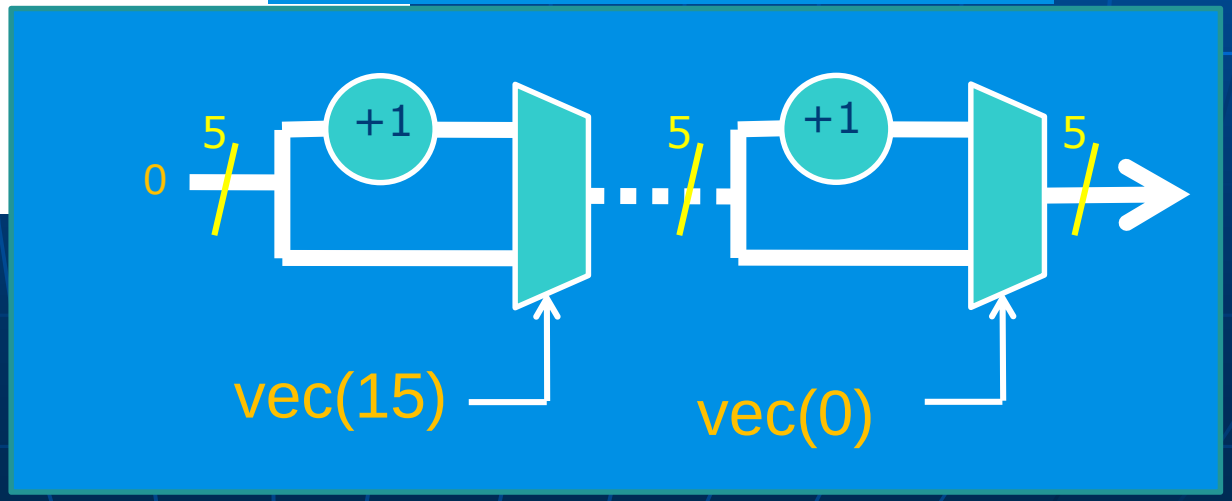
Circuito Combinatorio

contador_de_unos.vhd* | Compilation Report

```
1  library ieee;
2  use ieee.std_logic_1164.all, ieee.numeric_std.all;
3  entity contador_de_unos is
4    port (vec : in std_logic_vector(15 downto 0);
5          count : out unsigned(4 downto 0));
6  end;
7  architecture behaviour of contador_de_unos is
8    begin
9      process (vec)
10       variable result : unsigned(4 downto 0);
11       begin
12          result := "00000";
13          for i in 15 downto 0 loop
14             if vec(i) = '1' then
15                result := result + 1;
16             else result := result;
17             end if;
18          end loop;
19       count <= result;
20     end process;
21 end;
22
```

Inicialización de VARIABLE

POSIBLE SÍNTESIS DEL CIRCUITO



PROCESS: USO DE SENTENCIA FOR … LOOP

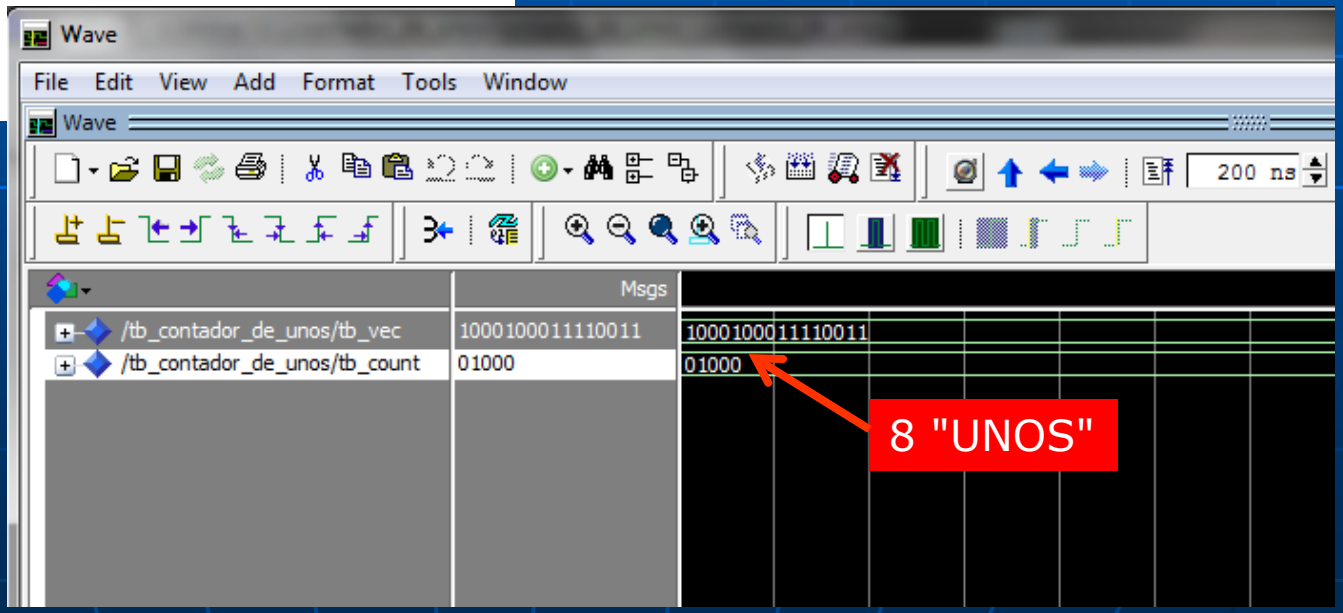
EJEMPLO: CONTADOR DE UNOS.

contador_de_unos.vhd* | Compilation Report | tb_contador_de_unos.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all, ieee.numeric_std.all;
3
4  entity tb_contador_de_unos is
5  end tb_contador_de_unos;
6
7  architecture test of tb_contador_de_unos is
8
9     component contador_de_unos
10       Port (    vec : in std_logic_vector(15 downto 0);
11                 count : out unsigned(4 downto 0));
12    end component;
13
14 signal tb_vec       : std_logic_vector(15 downto 0);
15 signal tb_count     : unsigned(4 downto 0);
16
17 begin
18
19   uut: contador_de_unos port map (  tb_vec, tb_count);
20
21    estimulos : process
22    begin
23                  tb_vec <= "1000100011110011";
24
25                  wait for 200 ns;
26
27               end process estimulos;
28
29   end test;
```

TEST-BENCH

VALOR INICIAL DE LA ENTRADA PARA CORRER LA SIMULACIÓN



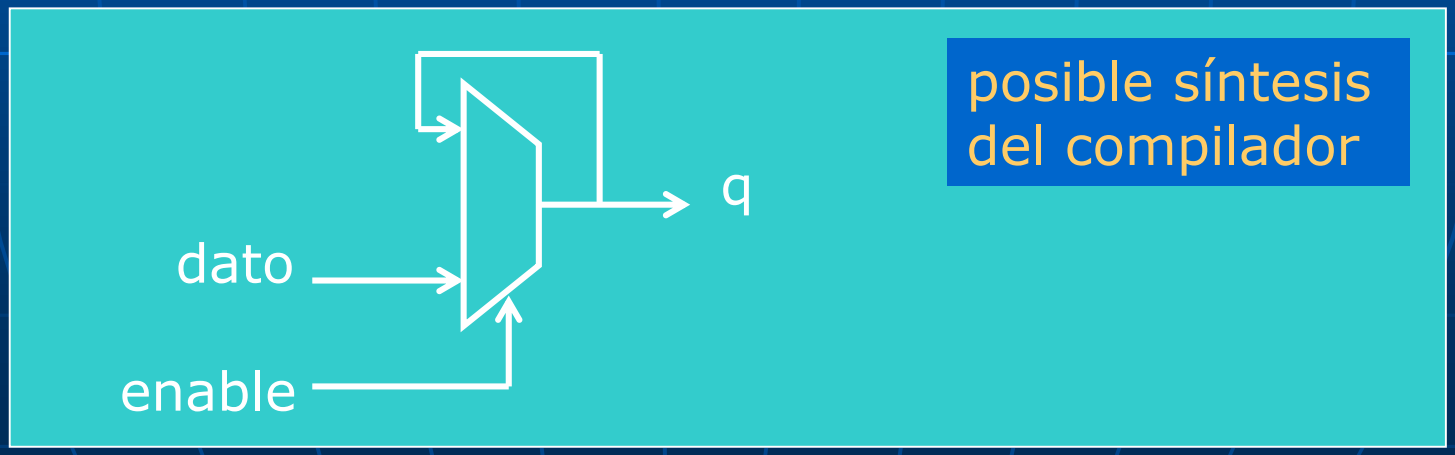
Diseño de Compuertas

Ejemplo: modelo de una compuerta AND

```
ENTITY and2 IS
        PORT (a:in bit; b:in bit; z:out bit);
END and2;

ARCHITECTURE funcional OF and2 IS
        BEGIN
            PROCESS (a,b)
                BEGIN
                        IF a=`1´and b=`1´ THEN z<=`1´;
                        ELSE z<=`0´;
            END PROCESS;
END funcional;
```

Diseño de Latches

```
ENTITY unlatch IS PORT ( enable, dato : IN std_logic;
        q : OUT std_logic);
END unlatch;

ARCHITECTURE tipo_latch1 OF unlatch IS BEGIN
  latch: PROCESS (enable, dato)
    BEGIN
        IF enable = `1´ THEN q <= dato; END IF;
  END PROCESS latch;
END tipo_latch1;
```



Diseño de Flip-Flops

Flip-Flop tipo "D"

```
LIBRARY ieee;
USE ieee.std_logc_1164.all;

ENTITY ff_tipod IS PORT ( d, reloj : IN std_logic;
            q : OUT std_logic);
END ff_tipod;

ARCHITECTURE tipo_reloj1 OF ff_tipod IS BEGIN
  PROCESS
    BEGIN
          WAIT UNTIL reloj = `1´; q <= d; --sensible a flanco ascendente
  END PROCESS;
END tipo_reloj1;
```

Otra forma de declarar la sensibilidad al flanco ascendente:

```
PROCESS (reloj) BEGIN
  IF reloj´EVENT AND reloj=`1´ THEN q <= d; END IF;
  END PROCESS;
```

Diseño de Flip-Flops

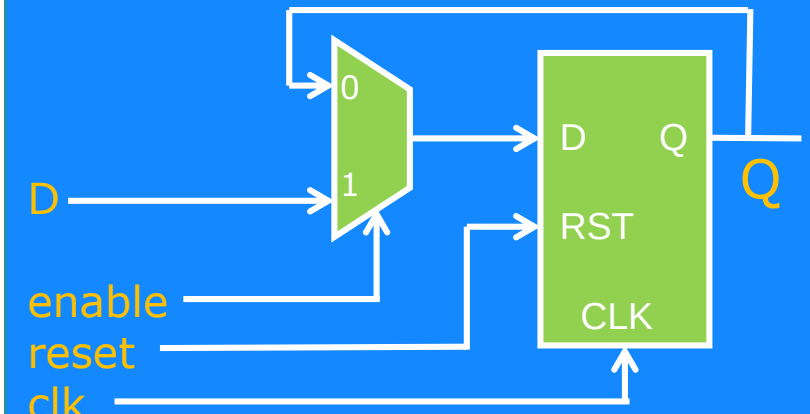
flip_flop_d1.vhd - Text Editor

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY flip_flop_d1 IS
    PORT
    (
        D, reset, clk, enable       : IN  std_logic;
        Q                           : OUT std_logic
    );
END flip_flop_d1;

ARCHITECTURE ffd1 OF flip_flop_d1 IS
BEGIN
    process(CLK, RESET, ENABLE)
        begin
            if RESET = '1' then
                Q <= '0';
            elsif rising_edge(CLK) then
            if Enable = '1' then
                Q <= D;
            end if;
            end if;
    end process;
END ffd1;
```

Flip-Flop tipo "D" con RESET asincrónico y habilitación de RELOJ

Aquí el RESET está fuera del proceso donde se evalúa que se hace cuando viene el flanco ascendente del reloj.
Como está antes de la sentencia de detección del flanco de clk el "reset" funciona como asincrónico.

Posible síntesis del compilador



Diseño de Flip-Flops

Flip-Flop tipo "D" con RESET sincrónico y habilitación de RELOJ

flip_flop_d2.vhd - Text Editor

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY flip_flop_d2 IS
    PORT
    (
        D, reset, clk, enable       : IN  std_logic;
        Q                           : OUT std_logic
    );
END flip_flop_d2;

ARCHITECTURE ffd2 OF flip_flop_d2 IS
BEGIN
    process(CLK, RESET, ENABLE)
        begin
            if rising_edge(CLK) then
                if RESET = '1' then
                    Q <= '0';
                elsif Enable = '1' then
                    Q <= D;
                end if;
            end if;
    end process;
END ffd2;
```

Ahora RESET está dentro del proceso y además se evalúa luego de detectar el flanco ascendente del reloj.
Por lo tanto el "reset" es sincrónico.

Diseño de Flip-Flops

Registro de 8 bits con RESET sincrónico

registro8bits.vhd - Text Editor

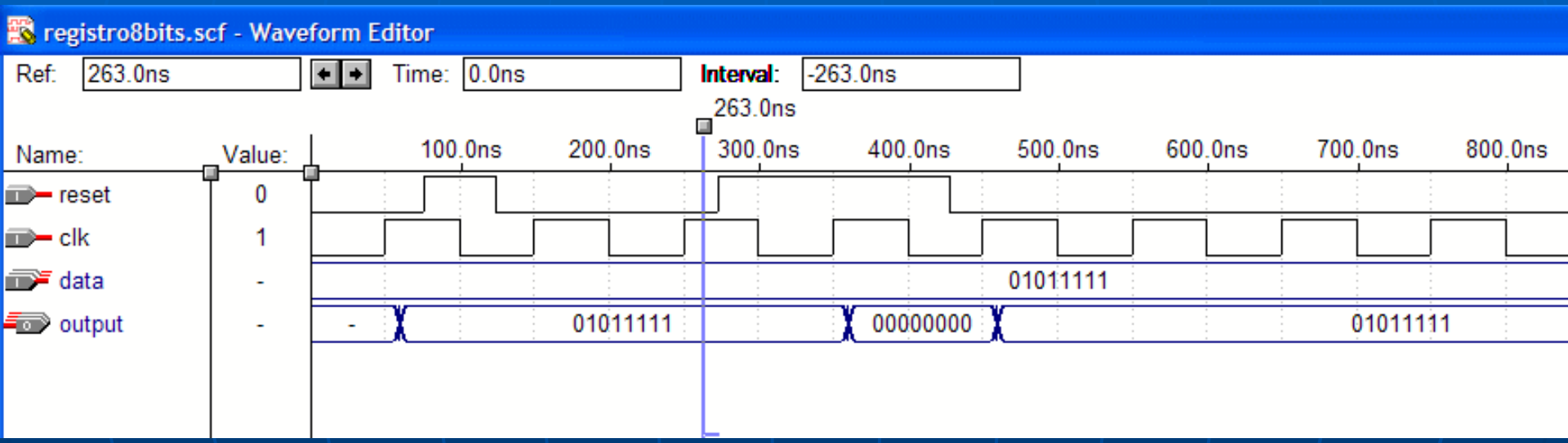
```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY registro8bits IS
    PORT
    (
        data         : IN  std_logic_vector(7 DOWNTO 0);
        reset, clk   : IN  std_logic;
        output           : OUT std_logic_vector(7 DOWNTO 0)
    );
END registro8bits;

ARCHITECTURE ffdx8 OF registro8bits IS
BEGIN
    process(CLK, RESET)
        BEGIN
            WAIT UNTIL (clk'EVENT AND clk ='1');
                IF RESET = '1' then
                     output <= "00000000";
                ELSE output <= data;
                END IF;
    END process;
END ffdx8;
```

El RESET está dentro del proceso donde se evalúa cuando viene el flanco ascendente del reloj

Diseño de Flip-Flops

Registro de 8 bits con RESET sincrónico

Simulación



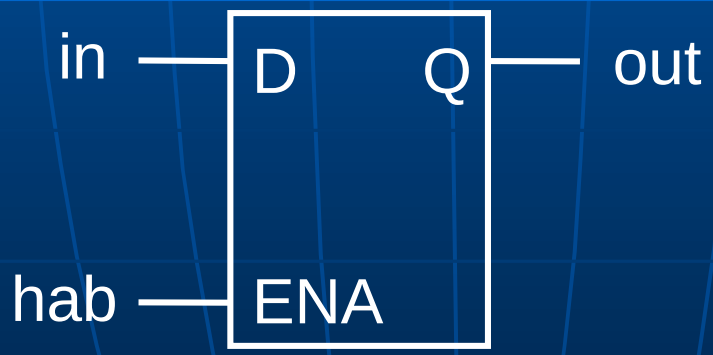
INFERENCIA DE ‘LATCH’ POR ESPECIFICACIÓN INCOMPLETA

ejemplo 1

```
signal in, out : std_logic_vector(3 downto 0);
signal hab : std_logic

process (hab, in)
    begin
      if hab = '1' then
       out <= in;
      end if;
end process;
```

No se definió que pasa cuando ‘hab’ es ‘0’ …!!!

El compilador, entonces, mantiene el último valor de la salida si pasa hab a ‘0’.



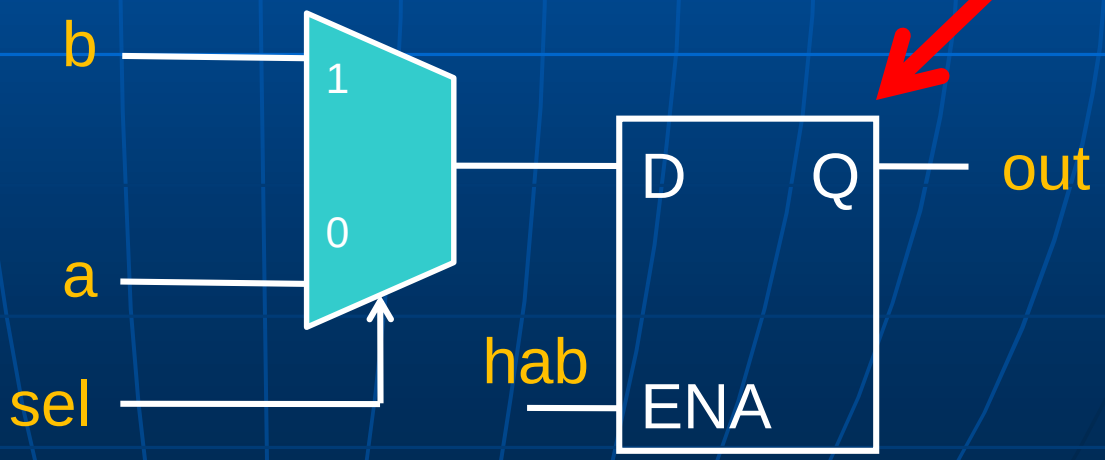
INFERENCIA DE ‘LATCH’ POR REQUERIMIENTO

ejemplo 2

```
process (hab, sel, a, b)
  begin
    if hab = '1' then
       if sel = '0' then
          z <= a;
       else
          z <= b;
       end if;
    end if;
end process;
```

En caso de requerir la habilidad de memorizar la salida del MUX, se omite exprofeso la opción de ‘hab = 0’ …

El compilador, entonces, mantiene el último valor de la salida si pasa hab a ‘0’.



MAS SOBRE DIFERENCIAS ENTRE VARIABLES Y SEÑALES

Otro ejemplo con un circuito secuencial

Mismo proceso con diferentes asignaciones

```
signal a,b : std_logic_vector (3 downto 0) := "0000";
process (reloj)
begin
if (event´reloj and reloj='1') then a <= "1010";
b <= a;
end if;
end process;
```

En este caso cuando se ejecute el proceso por primera vez, la señal «a» adoptará el valor "1010",pero «b» seguirá con el valor "0000". Recién en el siguiente evento cambiará a "1010".

Las señales se actualizan luego de pasado el evento.

```
signal a,b : std_logic_vector (3 downto 0) := "0000";
process (reloj)
variable var : std_logic_vector (3 downto 0);
begin
if (event´reloj and reloj='1') ) then var := "1010";
a <= var;
b <= var;
end if;
end process;
```

En este caso cuando se ejecute el proceso, tanto «a» como «b» adoptarán el valor "1010".

Las variables se actualizan dentro del proceso.
Puede ser varias veces si hay sentencia FOR-LOOP, por ejemplo.

Diseño de Contadores

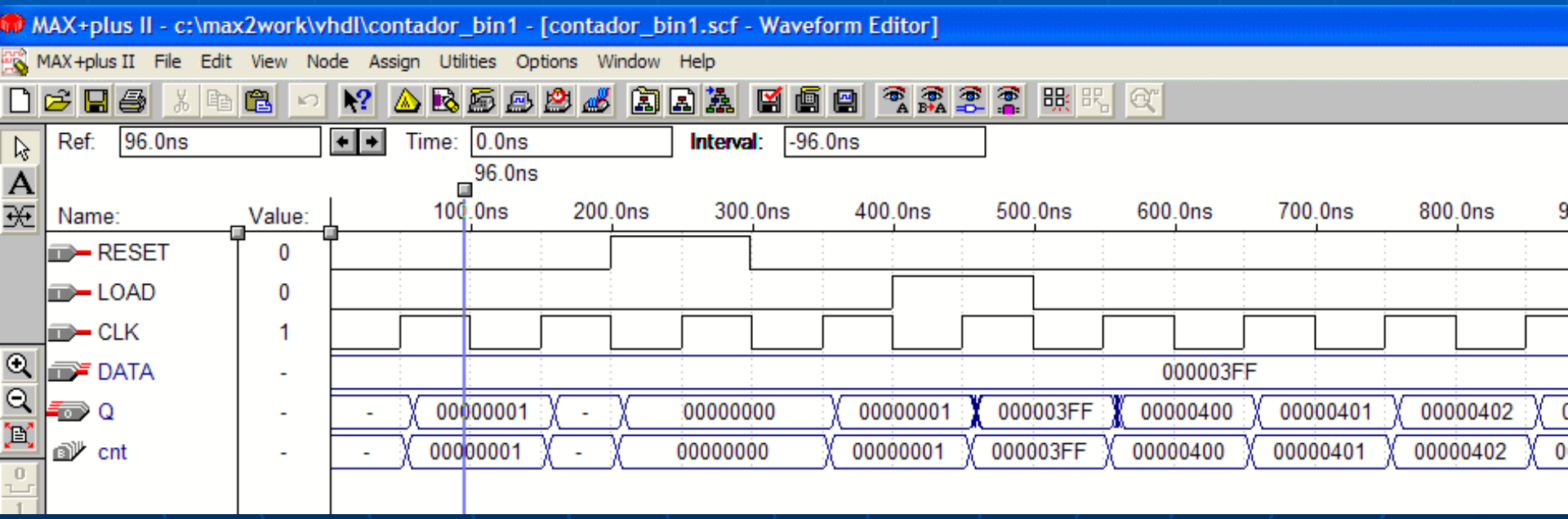
Contador binario progresivo de 32 bits con precarga y reset asincrónico

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity contador_bin1 is
generic ( WIDTH : integer := 32);
port (
  CLK, RESET, LOAD : in std_logic;
  DATA : in  unsigned(WIDTH-1 downto 0);
  Q    : out unsigned(WIDTH-1 downto 0));
end entity contador_bin1;

architecture contador_32 of contador_bin1 is
signal cnt : unsigned(WIDTH-1 downto 0);
begin
  process(RESET, CLK)
  begin
    if RESET = '1' then
      cnt <= (others => '0');
    elsif rising_edge(CLK) then
      if LOAD = '1' then
        cnt <= DATA;
      else
        cnt <= cnt + 1;
      end if;
    end if;
  end process;

  Q <= cnt;

end architecture contador_32;
```

Diseño de Contadores

Contador binario progresivo de 32 bits con precarga y reset asincrónico

Simulación

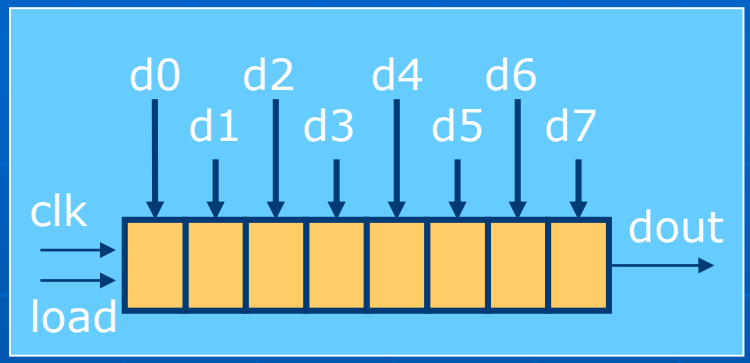


Diseño de Registro de Desplazamiento

RD paralelo-serie con carga sinc.

rd.vhd - Text Editor

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

ENTITY rd IS PORT (
  d : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
  clk, load : IN STD_LOGIC;
  dout : OUT STD_LOGIC);
END rd;

ARCHITECTURE rdpara_serie OF rd IS
   SIGNAL reg : STD_LOGIC_VECTOR (7 DOWNTO 0);
   BEGIN
   PROCESS (clk)
     BEGIN
       IF (clk'EVENT AND clk='1') THEN
         IF (load='1') THEN reg <= d;
            ELSE reg <= reg(6 DOWNTO 0) & '0';
         END IF;
       END IF;
   END PROCESS;
   dout <= reg(7);
END rdpara_serie;
```



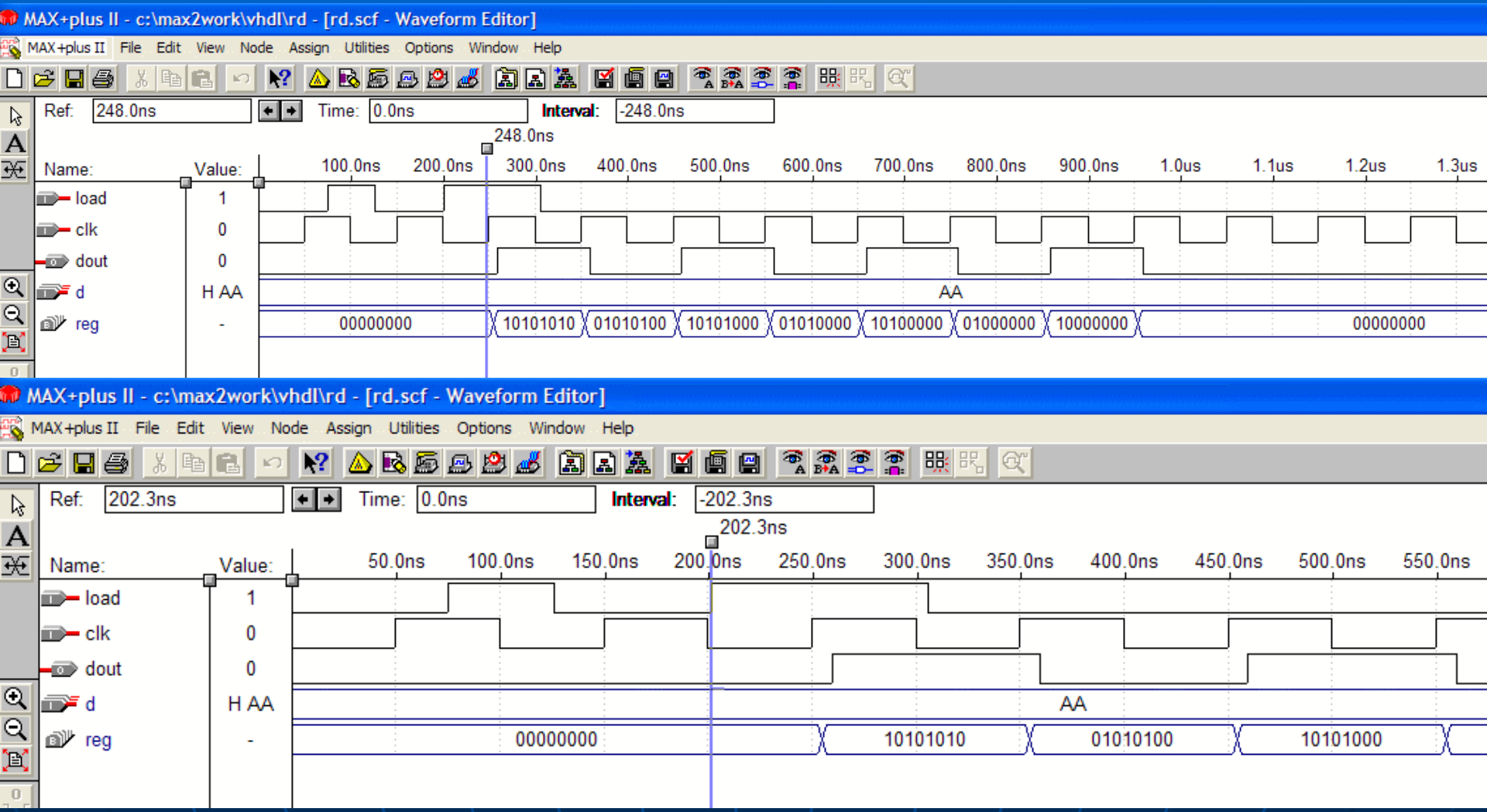
La carga del registro de desplazamiento es sincrónica ya que está dentro del proceso de control del flanco del reloj.

Esta operación con el operador de concatenación agrega en este caso un “0” en el bit LSB del RD.

Diseño de Registro de Desplazamiento

RD paralelo-serie con carga sinc.

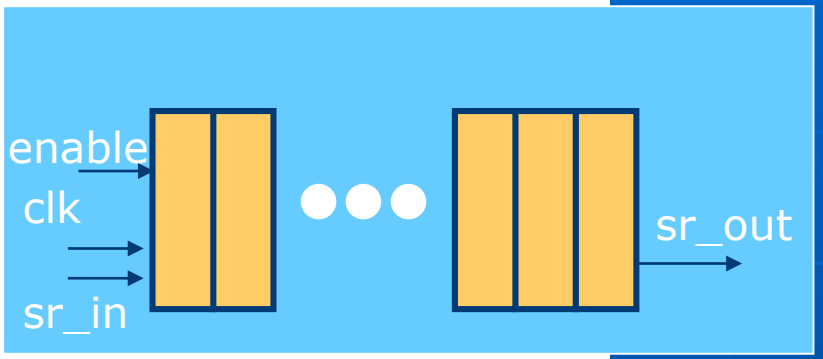
Simulación



Diseño de Registro de Desplazamiento

RD serie-serie.

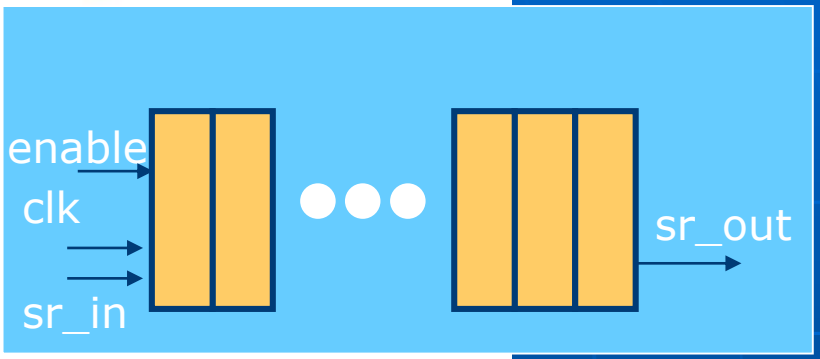
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity shift_1x64 is
5       port
6       (
7           sr_in : in std_logic;
8           enable   : in std_logic;
9           clk      : in std_logic;
10          sr_out   : out std_logic
11      );
12
13  end entity;
14
```



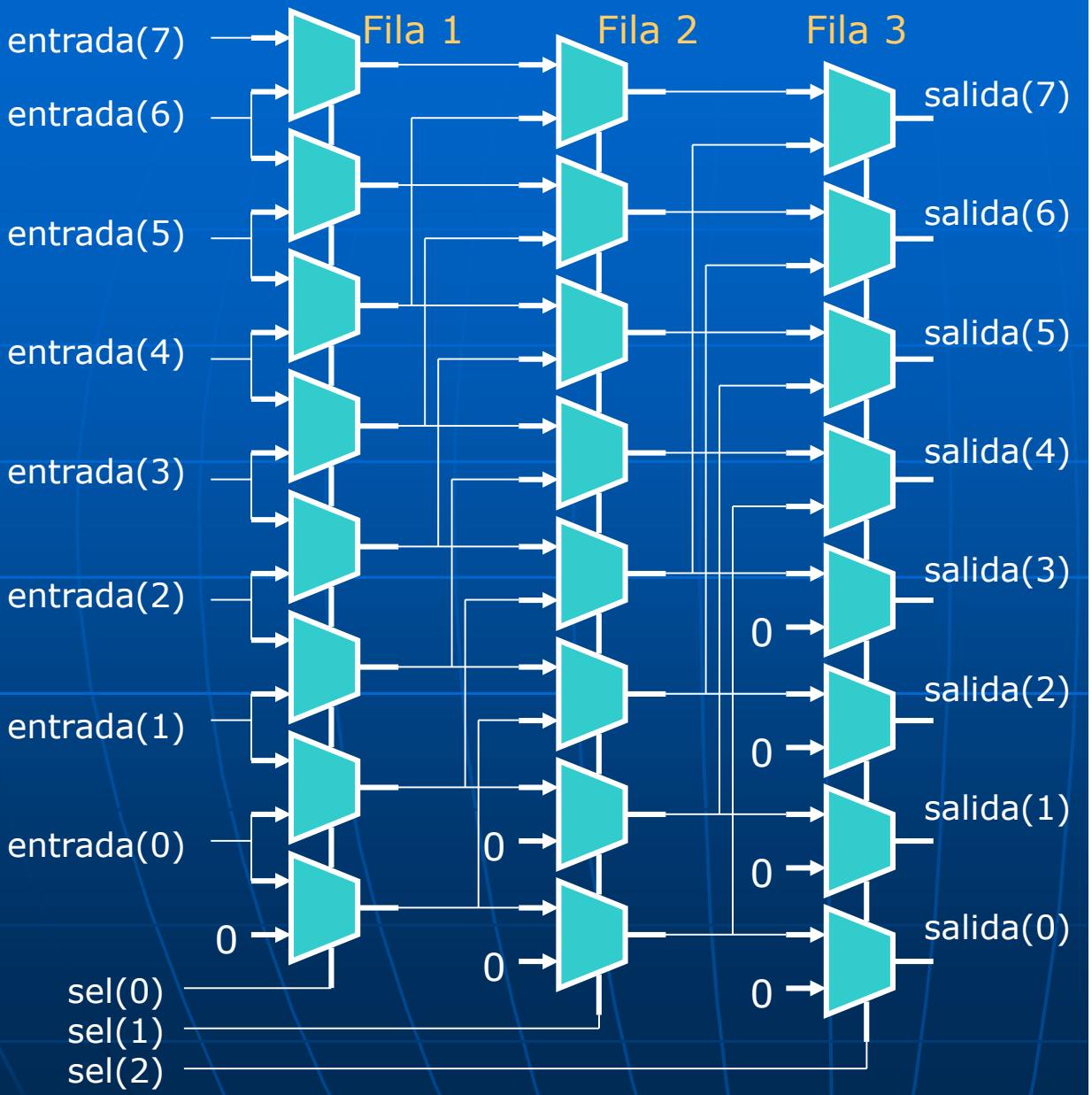
Diseño de Registro de Desplazamiento

RD serie-serie.

```
15  architecture rtl of shift_1x64 is
16
17      -- Build an array type for the shift register
18      type sr_length is array (63 downto 0) of std_logic;
19
20      -- Declare the shift register signal
21      signal sr: sr_length;
22
23  begin
24
25      process (clk)
26      begin
27          if (rising_edge(clk)) then
28              if (enable = '1') then
29
30                  -- Shift data by one stage; data from last stage is lost
31                  sr(63 downto 1) <= sr(62 downto 0);
32
33                  -- Load new data into the first stage
34                  sr(0) <= sr_in;
35
36              end if;
37          end if;
38      end process;
39
40      -- Capture the data from the last stage, before it is lost
41      sr_out <= sr(63);
42
```



Diseño de Barrel Shifters

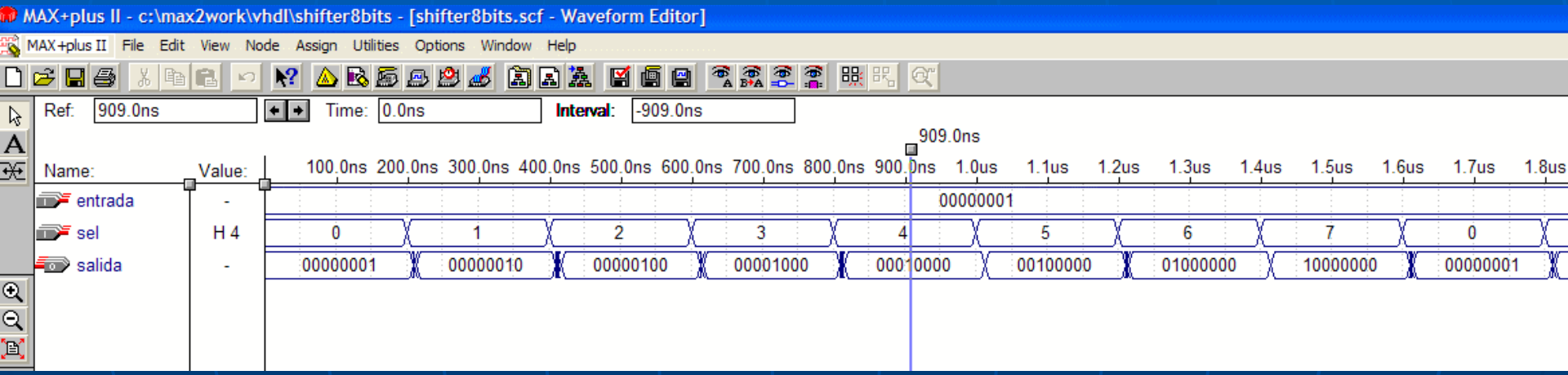



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY shifter8bits IS
    PORT(
        entrada: IN STD_LOGIC_VECTOR (7 downto 0);
        sel: IN STD_LOGIC_VECTOR (2 downto 0);
        salida: OUT STD_LOGIC_VECTOR (7 downto 0));
END shifter8bits;

ARCHITECTURE barrel OF shifter8bits IS
BEGIN
PROCESS (entrada, sel)
    VARIABLE temp1: STD_LOGIC_VECTOR(7 downto 0);
    VARIABLE temp2: STD_LOGIC_VECTOR(7 downto 0);
BEGIN
--Primera fila
IF (sel(0)='0') THEN
    temp1 := entrada;
ELSE
    temp1(0) := '0';
    FOR i IN 1 TO entrada'HIGH LOOP
        temp1(i) := entrada(i-1);
    END LOOP;
END IF;
--Segunda fila
IF (sel(1)='0') THEN
    temp2 := temp1;
ELSE
    FOR i IN 0 TO 1 LOOP
        temp2(i) := '0';
    END LOOP;
    FOR i IN 2 TO entrada'HIGH LOOP
        temp2(i) := temp1(i-2);
    END LOOP;
END IF;
--Tercera fila
IF (sel(2)='0') THEN
    salida <= temp2;
ELSE
    FOR i IN 0 TO 3 LOOP
        salida(i) <= '0';
    END LOOP;
    FOR i IN 4 TO entrada'HIGH LOOP
        salida(i) <= temp2(i-4);
    END LOOP;
END IF;
END PROCESS;
END barrel;
```

Diseño de Barrel Shifters

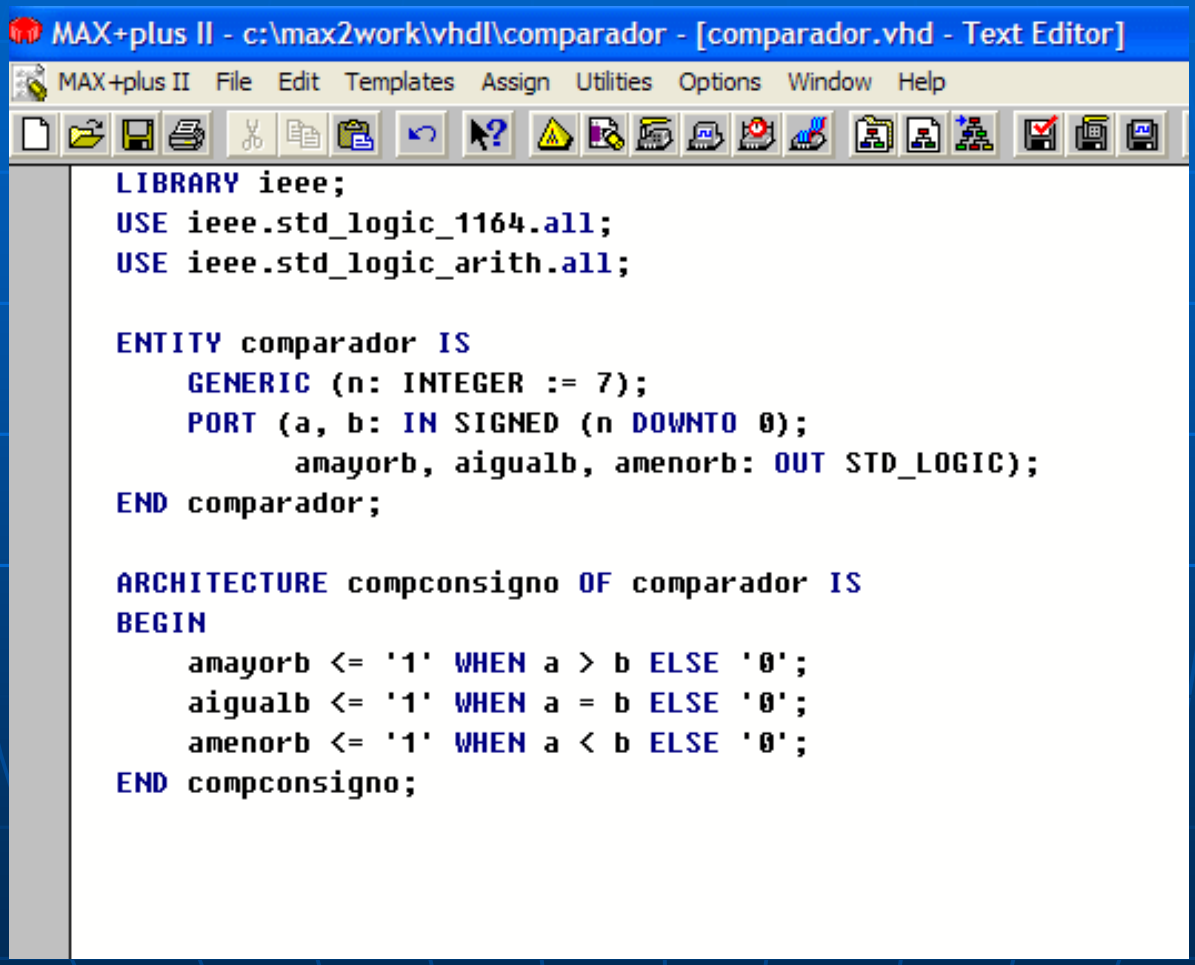
Ejemplo: Diseño de Shifter aritmético de 8 bits a izquierda.

Simulación



Diseño de comparadores

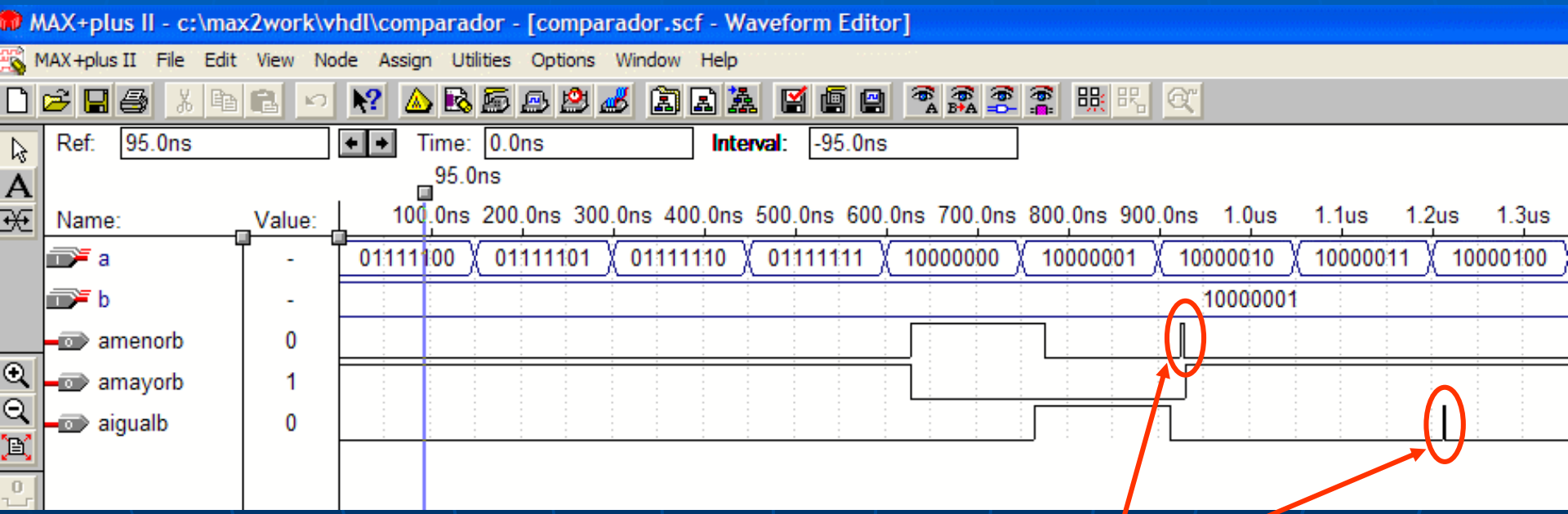
Ejemplo: Diseño de comparador binario con signo en punto fijo de dos números “a” y “b” de 8 bits de longitud de palabra



Diseño de comparadores

Ejemplo: Diseño de comparador binario con signo en punto fijo de dos números “a” y “b” de 8 bits de longitud de palabra

Simulación

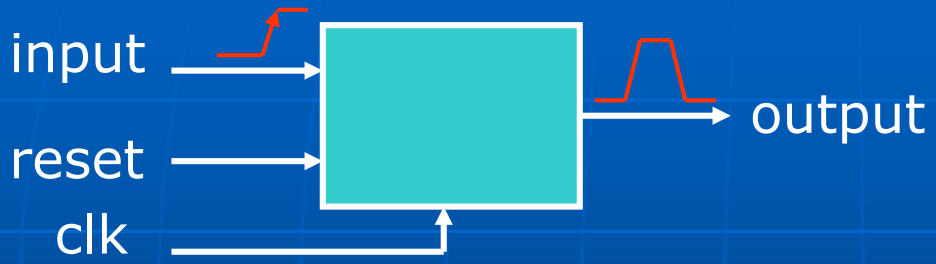


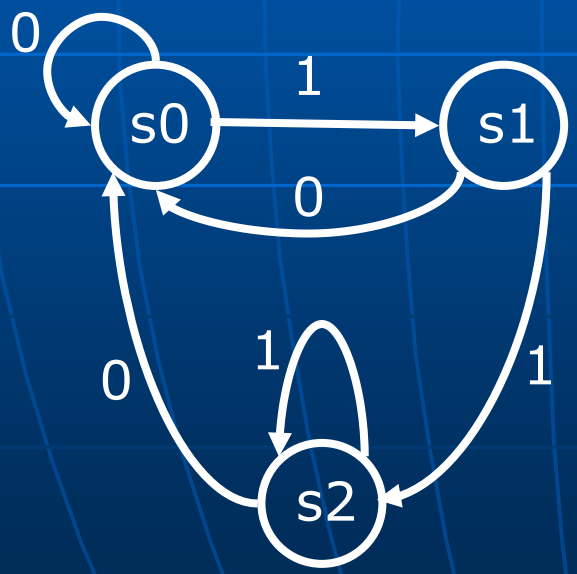
Porqué los glitches...???

Cómo se soluciona...???

Diseño de Máquinas de Estado

Ejemplo: Diseño de un monoestable disparado por flanco ascendente empleando "Máquina de Moore".



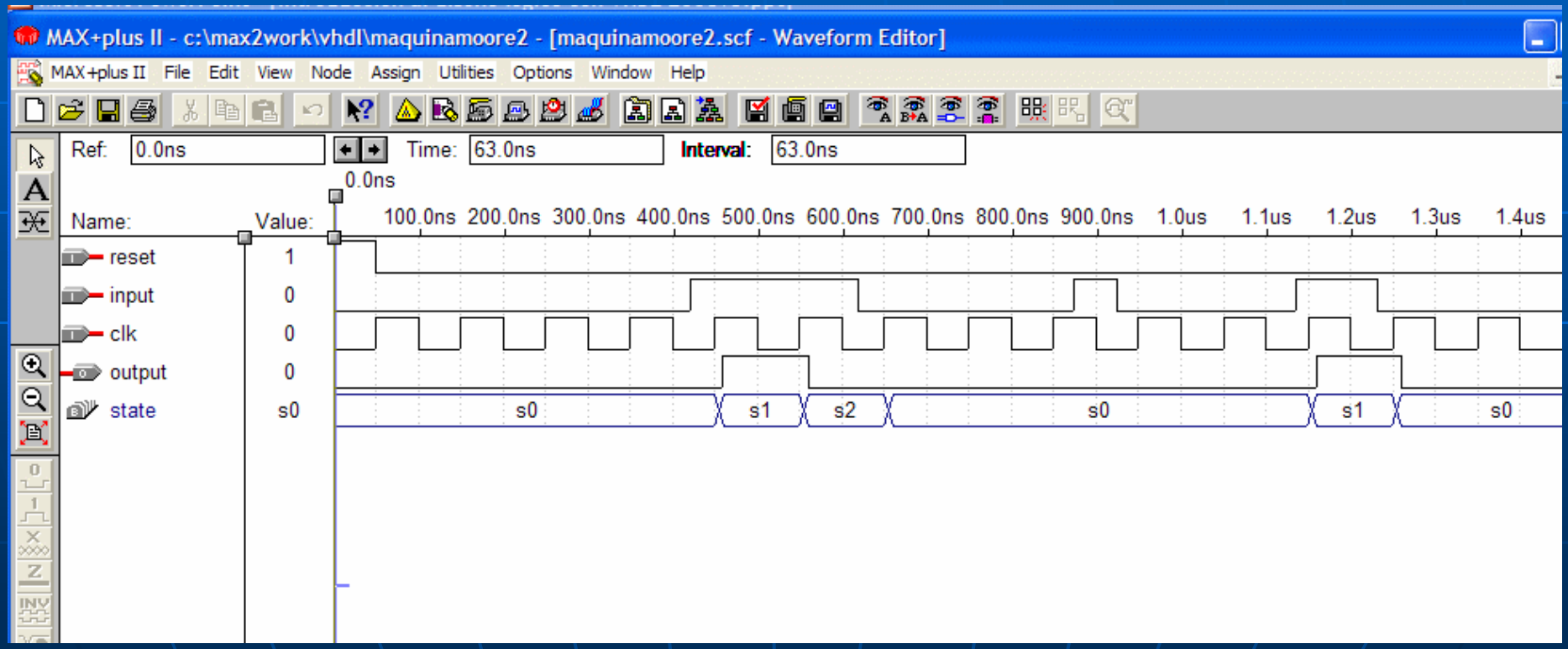



```
ENTITY maquinamoore2 IS
    PORT(
        clk     : IN    BIT;
        input   : IN    BIT;
        reset   : IN    BIT;
        output  : OUT   BIT);
END maquinamoore2;

ARCHITECTURE moore2 OF maquinamoore2 IS
    TYPE STATE_TYPE IS (s0, s1,s2);
    SIGNAL state    : STATE_TYPE;
BEGIN
    PROCESS (clk)
    BEGIN
        IF reset = '1' THEN
            state <= s0;
        ELSIF (clk'EVENT AND clk = '1') THEN
            CASE state IS
                WHEN s0=>
                    IF input = '1' THEN
                        state <= s1;
                    ELSE
                        state <= s0;
                    END IF;
                WHEN s1=>
                    IF input = '1' THEN
                        state <= s2;
                    ELSE
                        state <= s0;
                    END IF;
                WHEN s2=>
                    IF input = '1' THEN
                        state <= s2;
                    ELSE
                        state <= s0;
                    END IF;
            END CASE;
        END IF;
    END PROCESS;
output <= '1' WHEN state = s1 ELSE '0';
END moore2;
```

Diseño de Máquinas de Estado

Ejemplo: Diseño de un monoestable disparado por flanco ascendente empleando "Máquina de Moore".(continuación).

Simulación

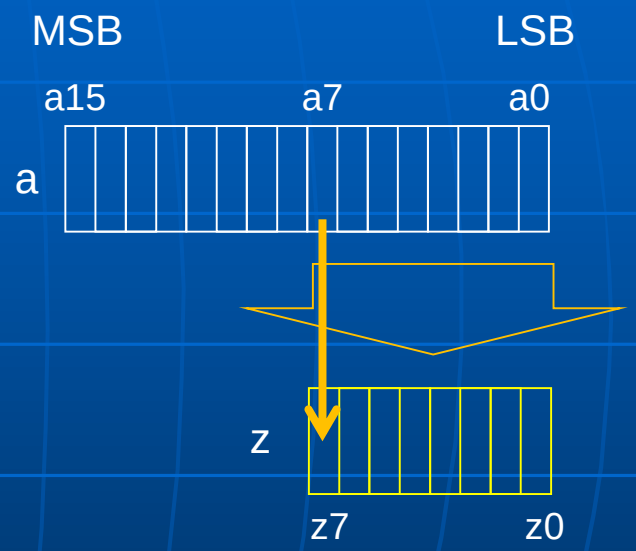


FUNCIONES

(SÓLO ALGUNOS EJEMPLOS)

RESIZE (señales sin signo)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity unsigned_resize is
  port (a : in unsigned(15 downto 0);
        z : out unsigned(7 downto 0));
end;
```



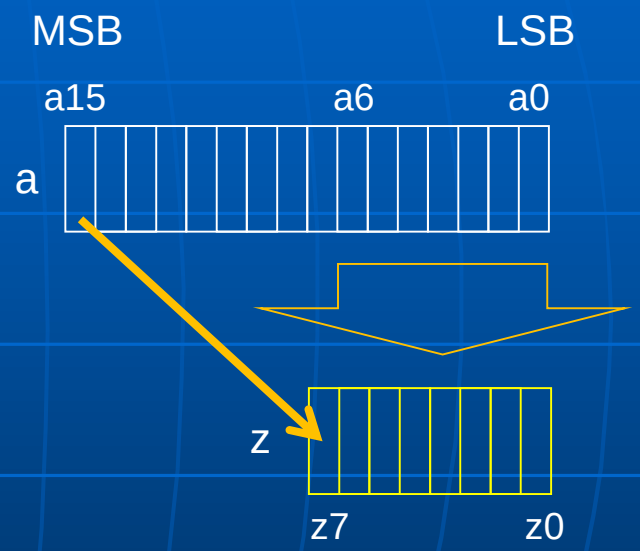

```
architecture behaviour of unsigned_resize is
  begin
    z <= resize(a, 8);
end;
```

(SÓLO ALGUNOS EJEMPLOS)

FUNCIONES

RESIZE (señales con signo)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity signed_resize is
port (a : in signed(15 downto 0);
z : out signed(7 downto 0));
end;
```




```
architecture behaviour of signed_resize is
begin
z <= resize(a, 8);
end;
```

(SÓLO ALGUNOS EJEMPLOS)

FUNCIONES

CONV_STD_LOGIC_VECTOR (ARG, SIZE)

CONVIERTE UN DATO EN STD_LOGIC_VECTOR

```
function conv_std_logic_vector(arg: integer, size: integer) return std_logic_vector;
function conv_std_logic_vector(arg: unsigned, size: integer) return std_logic_vector;
function conv_std_logic_vector(arg: signed, size: integer) return std_logic_vector;
function conv_std_logic_vector(arg: std_ulogic, size: integer) return std_logic_vector;
```

Si ‘arg’ es unsigned ó positvo es tratado como unsigned.
Si es negativo es convertido a signed en complemento a 2.

```
signal u1 : unsigned (3 downto 0);
signal s1 : signed (3 downto 0);
signal v1, v2, v3, v4 : std_logic_vector (3 downto 0);
signal i1, i2 : integer;
...
u1 <= "1101";
s1 <= "1101";
i1 <= 13;
i2 <= -3;
wait for 10 ns;
v1 <= conv_std_logic_vector(u1, 4); -- = "1101"
v2 <= conv_std_logic_vector(s1, 4);-- = "1101"
v3 <= conv_std_logic_vector(i1, 4);-- = "1101"
v4 <= conv_std_logic_vector(i2, 4);-- = "1101"
```

(SÓLO ALGUNOS EJEMPLOS)

FUNCIONES

CONV_INTEGER (STD_LOGIC_VECTOR)

CONVIERTE UN DATO EN ENTERO

```
function conv_integer(arg: integer) return integer;
function conv_integer(arg: unsigned) return integer;
function conv_integer(arg: signed) return integer;
function conv_integer(arg: std_ulogic) return small_int;
```

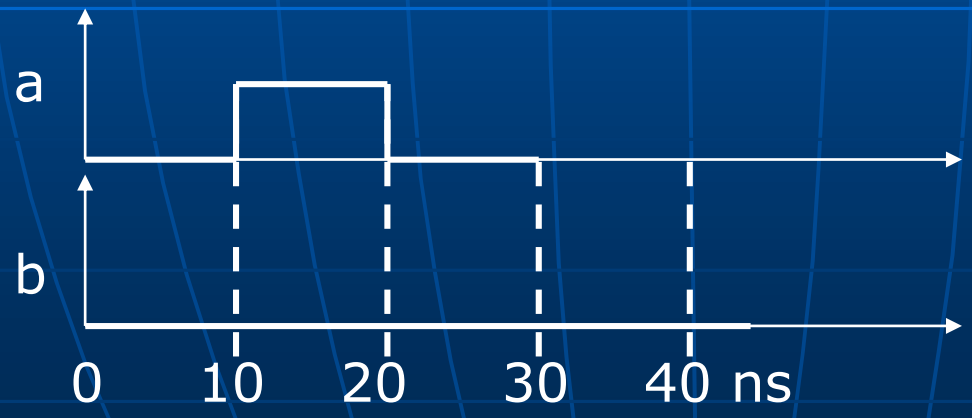
```
signal b : std_logic;
signal u1 : unsigned (3 downto 0);
signal s1 : signed (3 downto 0);
signal i1, i2, i3 : integer;
...
u1 <= "1001";
s1 <= "1001";
b <= 'X';
wait for 10 ns;
i1 <= conv_integer(u1); -- 9
i2 <= conv_integer(s1); -- -7
i3 <= conv_integer(b); -- warning generado por el simulador (valor indefinido)
```

MODELADO POR COMPORTAMIENTO

Modelado por RETARDO INERCIAL (default en VHDL)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY buf IS
PORT ( a : IN std_logic;
       b : OUT std_logic);
END buf;

ARCHITECTURE buffer_ine OF buf IS
BEGIN
       b <= a AFTER 20 ns;
END buffer_ine;
```

delay = 20 ns

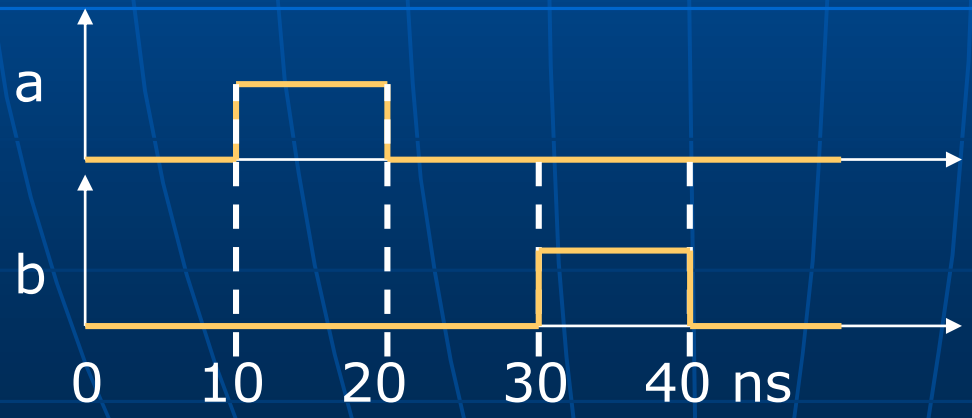


MODELADO POR COMPORTAMIENTO

Modelado por RETARDO de TRANSPORTE

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY linea_ret IS
PORT ( a : IN std_logic;
       b : OUT std_logic);
END linea_ret;

ARCHITECTURE delay_line OF buf IS
BEGIN
       b <= a TRANSPORT AFTER 20 ns;
END delay_line;
```

delay = 20 ns



TEST BENCH EN VHDL

Con VHDL es posible modelar no sólo el hardware sino también realizar un “banco de pruebas” (test bench) para aplicar estímulos al diseño a fin de analizar los resultados ó comparar los mismos de dos simulaciones diferentes.

VHDL es entonces además de un lenguaje de descripción de hardware un lenguaje para la definición de estímulos a los modelos descriptos en el mismo entorno de diseño.

A diferencia de los simuladores propietarios de las empresas proveedoras de dispositivos lógicos programables, los test bench realizados en VHDL permiten no sólo describir los estímulos en forma textual sino que es posible también comparar los resultados obtenidos al ensayar un DUT (dispositivo bajo prueba) con otros ya previamente cargados y generar entonces reportes de errores en caso de disparidad.
Esto se denomina “simulación automática”.

TEST BENCH EN VHDL

La simulación en VHDL permite mayor nivel de abstracción que con la síntesis.
Es posible por lo tanto modelizar el comportamiento de dispositivos sin que ellos sean luego sintetizados.
Por ejemplo en el caso de la simulación de un microprocesador donde se puede describir el comportamiento de una memoria aunque ésta no sea luego físicamente sintetizada.

Al ser VHDL un lenguaje portable es posible trabajar con simuladores de distintas empresas.

Ejemplo: El software ModelSim de Mentor Graphics.
(Existe una versión de trabajo para su uso con el Quartus II de Altera)

TEST BENCH EN VHDL

Diagrama de flujo de las opciones de simulación con el ModelSim-Altera

Es posible realizar simulaciones del tipo:

--Funcional RTL (Register Transfer Level)

--Post-síntesis

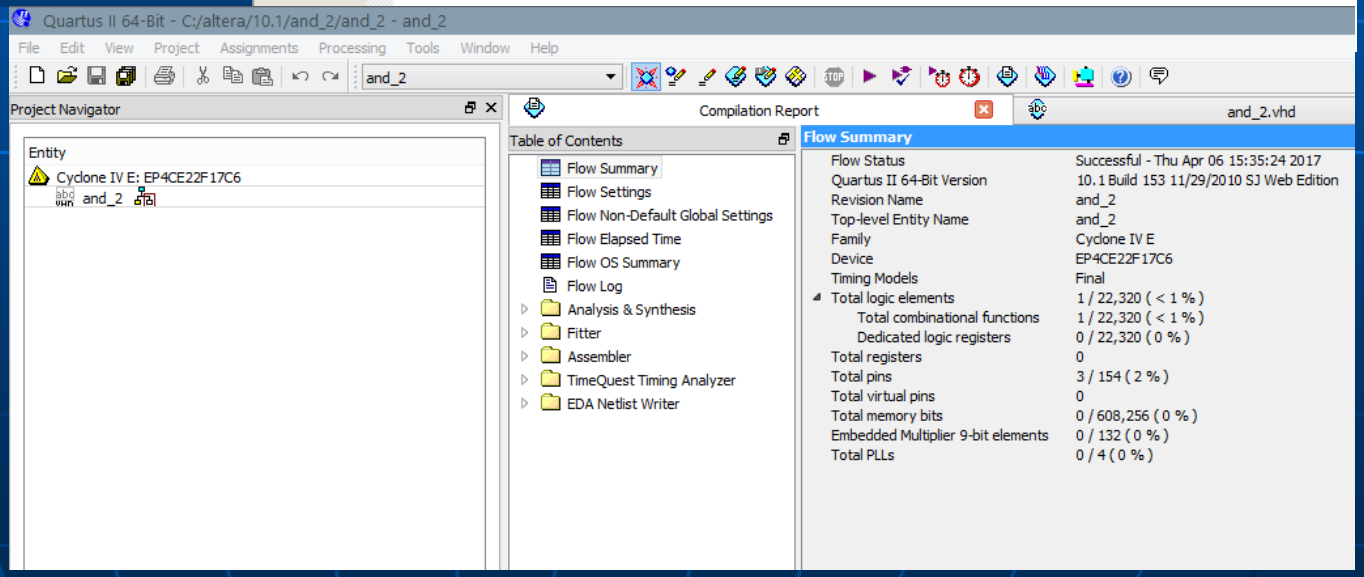
--Post-place&route (simulación temporal con información de timing precisa la cual debe ser provista por el fabricante del PLD ó ASIC)

TEST BENCH EN VHDL

Ejemplo de simulación sencilla con el software ModelSim de (Mentor Graphics)

Archivo VHDL que contiene el dispositivo a simular (and_2.vhd):
En este caso una compuerta AND de 2 entrads.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity and_2 is
5   Port ( entrada_a       : in std_logic;
6          entrada_b       : in std_logic;
7          salida_c        : out std_logic
8       );
9   end and_2;
10
11  architecture Comportamiento of and_2 is
12     begin
13        process(entrada_a, entrada_b)
14           begin
15              if (entrada_a = '1' and entrada_b = '1') then
16                 salida_c <= '1';
17              else
18                 salida_c <= '0';
19              end if;
20        end process;
21     end Comportamiento;
```



TEST BENCH EN VHDL

Ejemplo de simulación sencilla con ModelSim (continuación)

Archivo VHDL que contiene el test-bench de la "and" descripto anteriormente en otro archivo: (tb_and_2.vhd).

Sección para la generación de las señales de estímulo y acciones a tomar en caso de detectar inconsistencias en el funcionamiento.

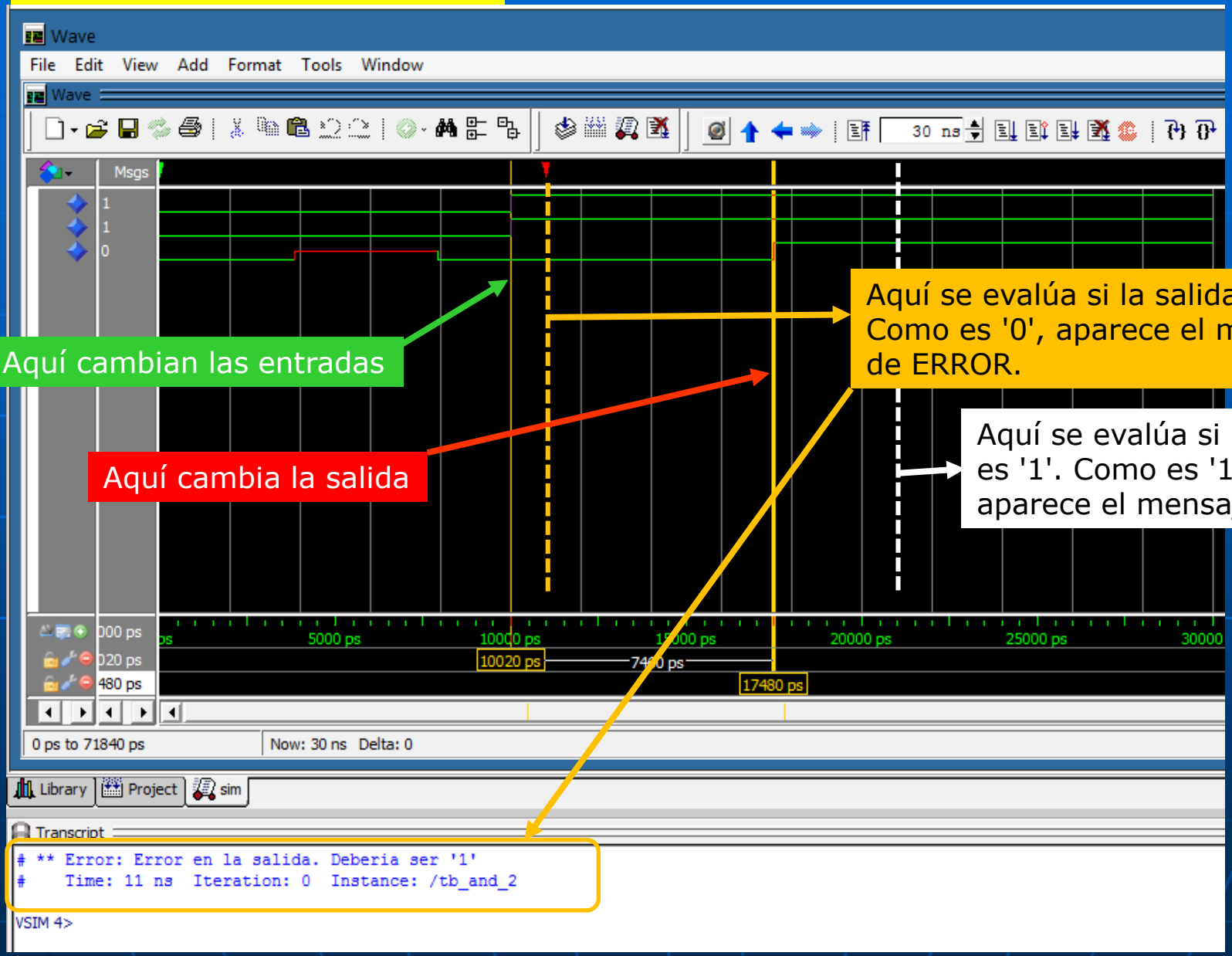
En este caso se chequea si la salida está en '1' luego de 1ns y 21 ns al cambiar ambas entradas a '1'.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY tb_and_2 IS
5  END tb_and_2;
6
7  ARCHITECTURE behavior OF tb_and_2 IS
8
9      COMPONENT and_2 is
10     Port ( entrada_a    : in  std_logic;
              entrada_b    : in  std_logic;
              salida_c : out std_logic
            );
           end COMPONENT;

       signal entrada_a, entrada_b, salida_c : std_logic;

   BEGIN
19
20     uut: and_2 PORT MAP (entrada_a, entrada_b, salida_c);
21
22     estimulo_proc: process
23        begin
24           entrada_a <= '0';
25           entrada_b <= '0';
26           wait for 10 ns;
27           entrada_a <= '1';
28           entrada_b <= '1';
29           wait for 1 ns;
             assert (salida_c = '1')
31           report "Error en la salida. Deberia ser '1'"
32           severity ERROR;
33           wait for 20 ns;
34           assert (salida_c = '1')
35           report "Error en la salida. Deberia ser '1'"
36           severity ERROR;
37           wait for 10 ns;
38     end process;
39
40 END;
```

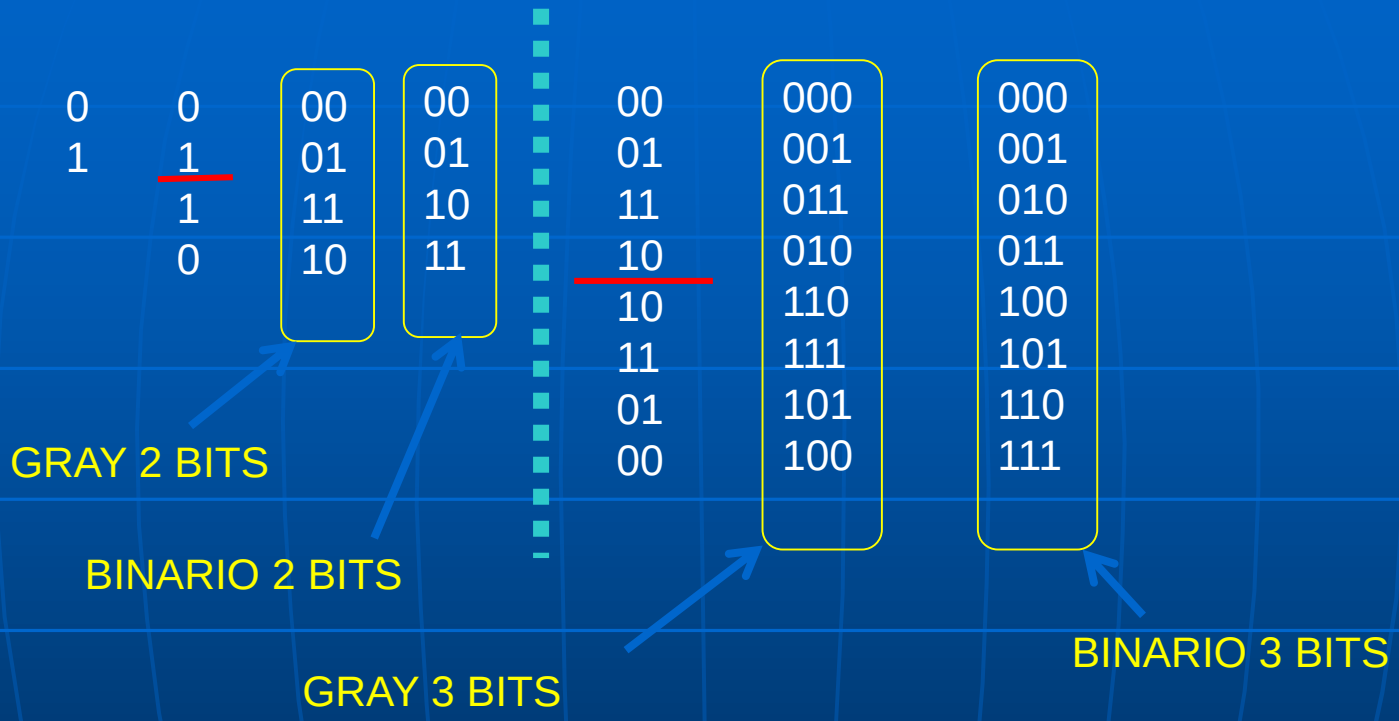
Instanciación de la entidad de la "and" ahora como un componente dentro de la entidad llamada "test_and_2"

TEST BENCH EN VHDL

Ejemplo de simulación con ModelSim



CONVERSIÓN BINARIO A GRAY



CONVERSIÓN BINARIO A GRAY EN 3 BITS

CONCEPTO DE MEMORIA ROM (sólo lectura)

bin_a_gray_3bits.vhd | tb_bin_a_gray_3bits.vhd

```
1  --Conversor binario a gray en formato paralelo
2  --Noriega ISLD 2016
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6  entity bin_a_gray_3bits is
7  port (bin     : in unsigned(2 downto 0);
8        gray    : out std_logic_vector(2 downto 0));
9  end;
10
11 architecture behaviour of bin_a_gray_3bits is
12 type memory_type is array (0 to 7) of
13 std_logic_vector(2 downto 0);
14
15 constant memory : memory_type :=
16 ("000", "001", "011", "010", "110", "111", "101", "100");
17
18 begin
19      gray <= memory(to_integer(bin));
20 end;
21
```

BIN => GRAY

BIN	GRAY
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Función «to_integer»
Se necesita pasar a INTEGER para indexar en un ARRAY

Desde el punto de vista funcional, ‘memory’ es una memoria de sólo lectura (ROM) la cual se direcciona por ‘bin’ para leer la información de uno de los 8 datos almacenados en ella.

CONVERSIÓN BINARIO A GRAY EN 3 BITS

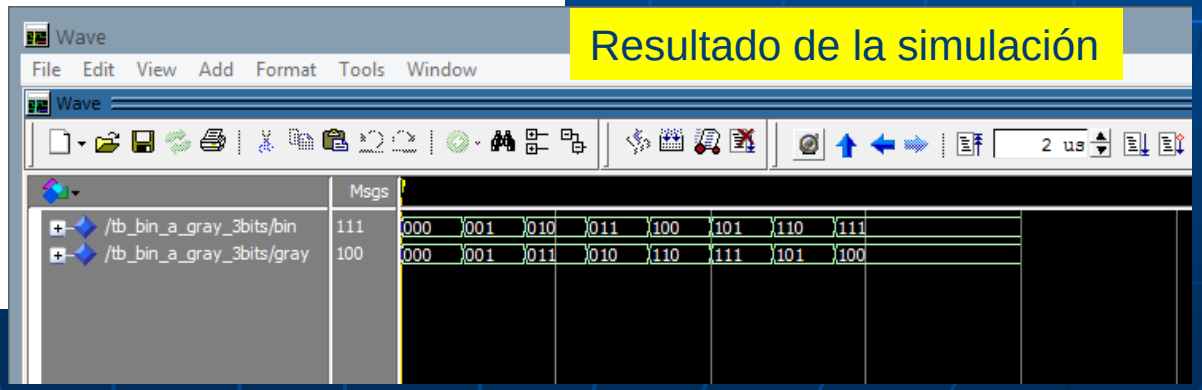
bin_a_gray_3bits.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_bin_a_gray_3bits IS
6  END tb_bin_a_gray_3bits;
7
8  ARCHITECTURE behavior OF tb_bin_a_gray_3bits IS
9
10     COMPONENT bin_a_gray_3bits is
11        port (bin     : in unsigned(2 downto 0);
12              gray    : out std_logic_vector(2 downto 0)
13              );
14     end COMPONENT;
15
16     signal bin    : unsigned(2 downto 0);
17     signal gray   : std_logic_vector(2 downto 0);
18
19 BEGIN
20
21    uut: bin_a_gray_3bits PORT MAP (bin, gray);
22
23    estimulo_proc: process
24       begin
25          for i in 0 to 7 loop
26             bin <= to_unsigned (i,3);
27             wait for 200 ns;
28          end loop;
29          wait;
30    end process;
31 END;
```

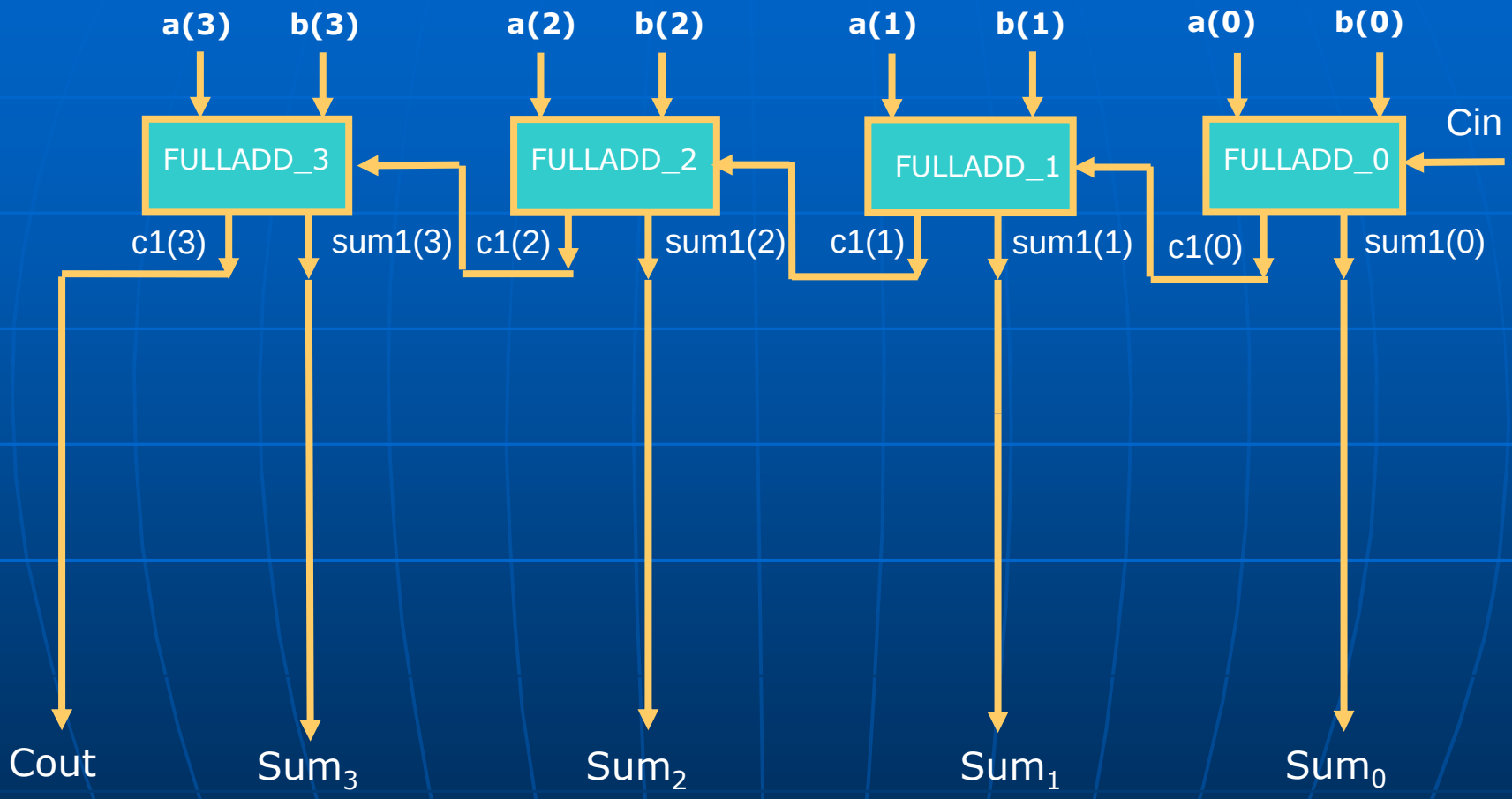
tb_bin_a_gray_3bits.vhd

Test bench en VHDL

Resultado de la simulación



Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits



Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

SUMADOR FULL-ADDER DE UN BIT

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity full_adder_1bit is
5       port (  x    : in std_logic;
6               y    : in std_logic;
7               ci   : in std_logic;
8               s    : out std_logic;
9               co   : out std_logic
10          );
11  end full_adder_1bit;
12
13  architecture behavior of full_adder_1bit is
14
15  begin
16
17  s  <= x xor y xor ci;
18  co <= (x and y) or (ci and (x xor y));
19
20  end;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

Descripción en VHDL

```vhdl
--Sumador de 2 operandos de 4 bits sin signo.
--Tipo: Ripple-Carry en formato Paralelo.
--Se emplea el metodo de descripcion ESTRUCTURAL para describir el
--sumador Ripple-Carry formado por sumadores completos de un bit.
--Se emplea descripcion de COMPORTAMIENTO para describir al sumador
--de un bit.
--SERGIO NORIEGA - ISLD - 2016
--Nombre del archivo TOP LEVEL: sum_ripple_carry_4bits.
--Este archivo se emplea junto con el denominado "full_adder_1bit"
--el cual describe un sumador completo de 1 bit y en este proyecto
--sera invocado 4 veces.


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sum_ripple_carry_4bits is
    Port ( a    : in  unsigned (3 downto 0);
           b    : in  unsigned (3 downto 0);
           cin  : in std_logic;
           cout : out std_logic;
           sum  : out unsigned (3 downto 0)
              );
end sum_ripple_carry_4bits;

architecture Behavioral of sum_ripple_carry_4bits is

component full_adder_1bit is
    port (  x    : in std_logic;
            y    : in std_logic;
            ci   : in std_logic;
            s    : out std_logic;
            co   : out std_logic
         );
end component;

signal c1,sum1,c2 : unsigned (3 downto 0) := (others => '0');

begin

fulladd_0 : full_adder_1bit port map(a(0),b(0),cin,sum1(0),c1(0));
fulladd_1 : full_adder_1bit port map(a(1),b(1),c1(0),sum1(1),c1(1));
fulladd_2 : full_adder_1bit port map(a(2),b(2),c1(1),sum1(2),c1(2));
fulladd_3 : full_adder_1bit port map(a(3),b(3),c1(2),sum1(3),c1(3));

sum(0) <= sum1(0);
sum(1) <= sum1(1);
sum(2) <= sum1(2);
sum(3) <= sum1(3);
cout <= c1(3);


end Behavioral;
```

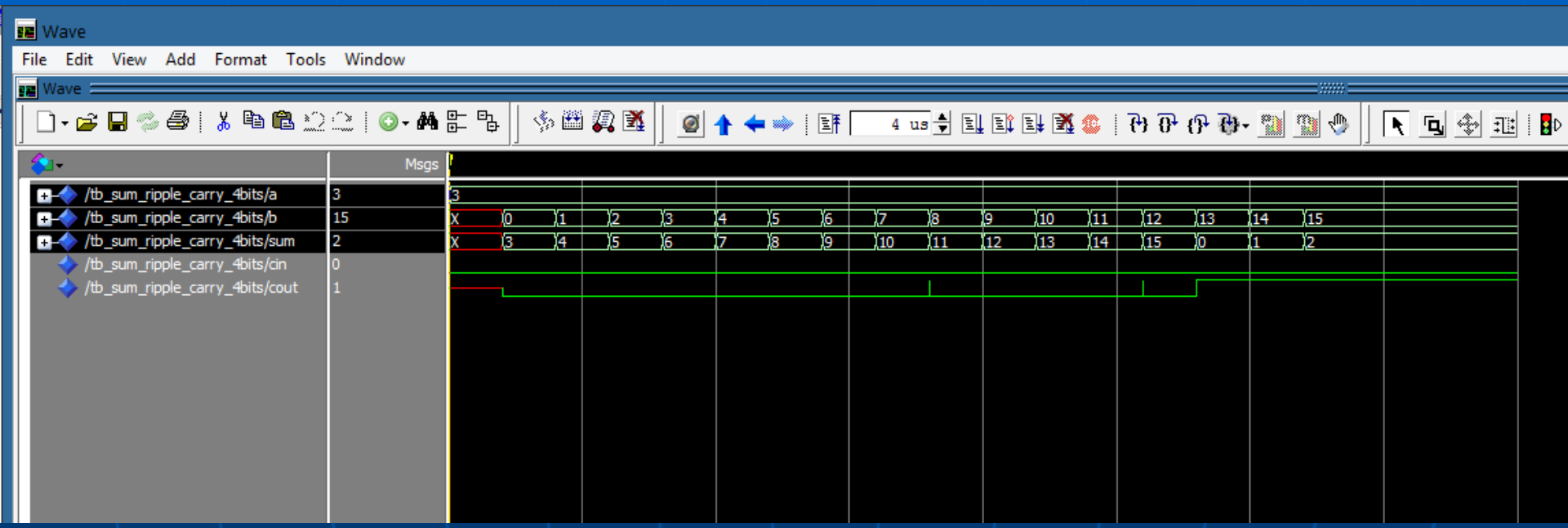
Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

Test bench en VHDL

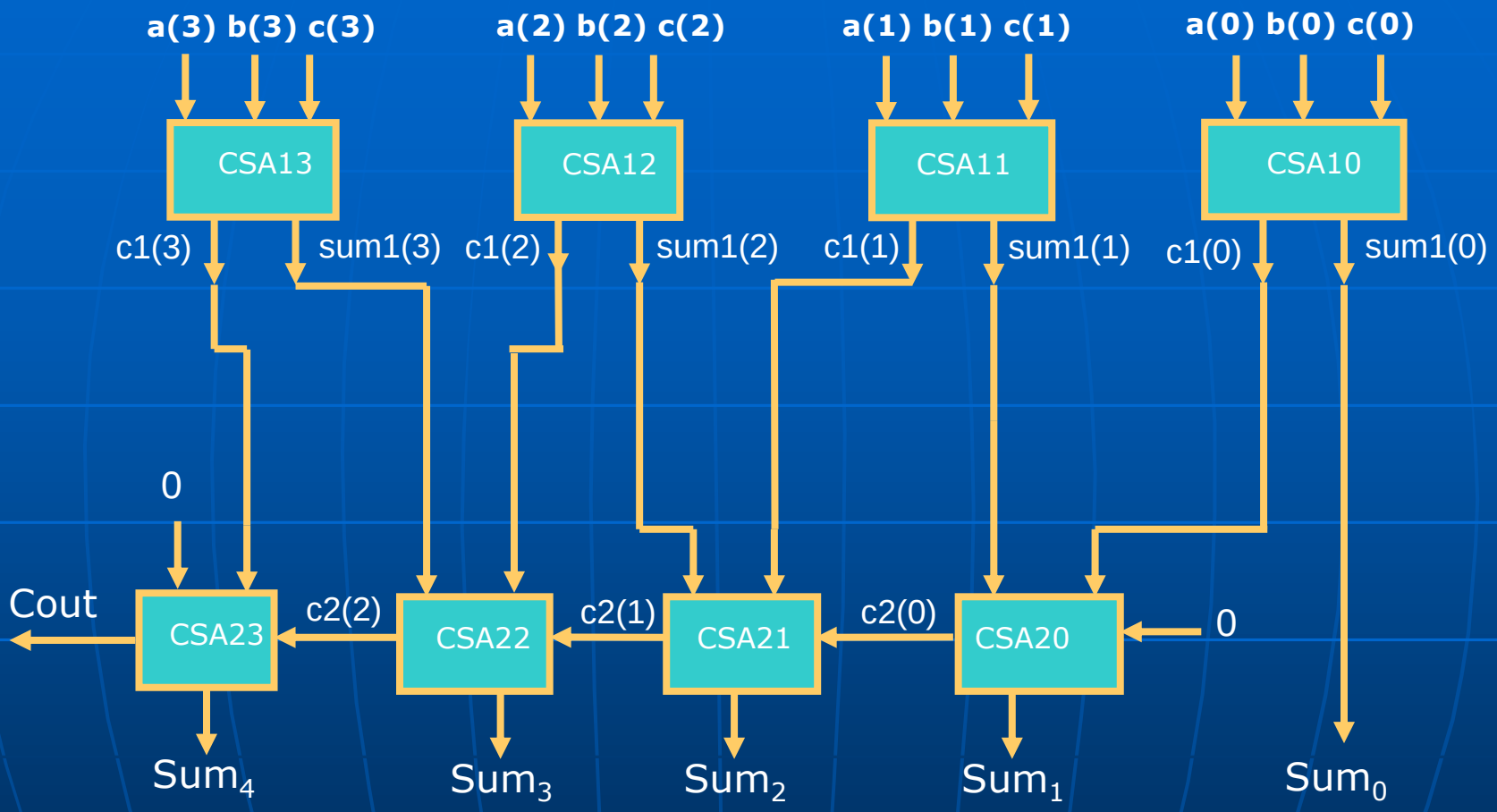
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_sum_ripple_carry_4bits IS
6  END tb_sum_ripple_carry_4bits;
7
8  ARCHITECTURE behavior OF tb_sum_ripple_carry_4bits IS
9
10     COMPONENT sum_ripple_carry_4bits is
11     Port ( a    : in  unsigned (3 downto 0);
12            b    : in  unsigned (3 downto 0);
13            cin  : in std_logic;
14            cout : out std_logic;
15            sum  : out unsigned (3 downto 0)
16               );
17         end COMPONENT;
18
19     signal a, b : unsigned(3 downto 0);
20     signal sum  : unsigned(3 downto 0) := (others => '0');
21     signal cin  : std_logic;
22     signal cout : std_logic := '0';
23
24 BEGIN
25
26    uut: sum_ripple_carry_4bits PORT MAP (a, b, cin, cout, sum);
27
28    estimulo_proc: process
29       begin
30          cin <= '0';
31          a <= to_unsigned (3,4);
32          for i in 0 to 15 loop
33             wait for 200 ns;
34             b <= to_unsigned(i,4);
35          end loop;
36          wait;
37    end process;
38
39 END;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

Resultado de la simulación



Diseño de un sumador tipo CARRY-SAVE de 3 operandos de 4bits



Diseño de un sumador de 3 operandos tipo carry-save

Descripción en VHDL

sum_carry_save_4bits.vhd | tb_sum_carry_save_4bits.vhd

```vhdl
--Sumador de 3 operandos de 4 bits sin signo.
--Tipo: Carry-Save en formato Paralelo.
--Se emplea el metodo de descripcion ESTRUCTURAL para describir el
--sumador Carry-Save formado por sumadores completos de un bit.
--Se emplea descripcion de COMPORTAMIENTO para describir al sumador
--de un bit.
--SERGIO NORIEGA - ISLD - 2016
--Nombre del archivo TOP LEVEL: sum_carry_save_4bits.
--Este archivo se emplea junto con el denominado "full_adder_1bit"
--el cual describe un sumador completo de 1 bit y en este proyecto
--sera invocado 8 veces.


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sum_carry_save_4bits is
    Port ( a    : in  unsigned (3 downto 0);
           b    : in  unsigned (3 downto 0);
           c    : in  unsigned (3 downto 0);
           cout : out std_logic;
           sum  : out unsigned (4 downto 0)
              );
end sum_carry_save_4bits;

architecture Behavioral of sum_carry_save_4bits is

component full_adder_1bit is
    port (  x    : in std_logic;
            y    : in std_logic;
            ci   : in std_logic;
            s    : out std_logic;
            co   : out std_logic
         );
end component;

signal c1,sum1,c2 : unsigned (3 downto 0) := (others => '0');

begin

csa_10 : full_adder_1bit port map(a(0),b(0),c(0),sum1(0),c1(0));
csa_11 : full_adder_1bit port map(a(1),b(1),c(1),sum1(1),c1(1));
csa_12 : full_adder_1bit port map(a(2),b(2),c(2),sum1(2),c1(2));
csa_13 : full_adder_1bit port map(a(3),b(3),c(3),sum1(3),c1(3));

csa_20 : full_adder_1bit port map(sum1(1),c1(0),'0',sum(1),c2(1));
csa_21 : full_adder_1bit port map(sum1(2),c1(1),c2(1),sum(2),c2(2));
csa_22 : full_adder_1bit port map(sum1(3),c1(2),c2(2),sum(3),c2(3));
csa_23 : full_adder_1bit port map('0',c1(3),c2(3),sum(4),cout);

sum(0) <= sum1(0);

end Behavioral;
```

Diseño de un sumador de 3 operandos tipo carry-save

SUMADOR FULL-ADDER DE UN BIT

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity full_adder_1bit is
5       port (  x     : in std_logic;
6               y     : in std_logic;
7               ci    : in std_logic;
8               s     : out std_logic;
9               co    : out std_logic
10          );
11  end full_adder_1bit;
12
13  architecture behavior of full_adder_1bit is
14
15  begin
16
17  s  <= x xor y xor ci;
18  co <= (x and y) or (ci and (x xor y));
19
20  end;
```

Diseño de un sumador de 3 operandos tipo carry-save

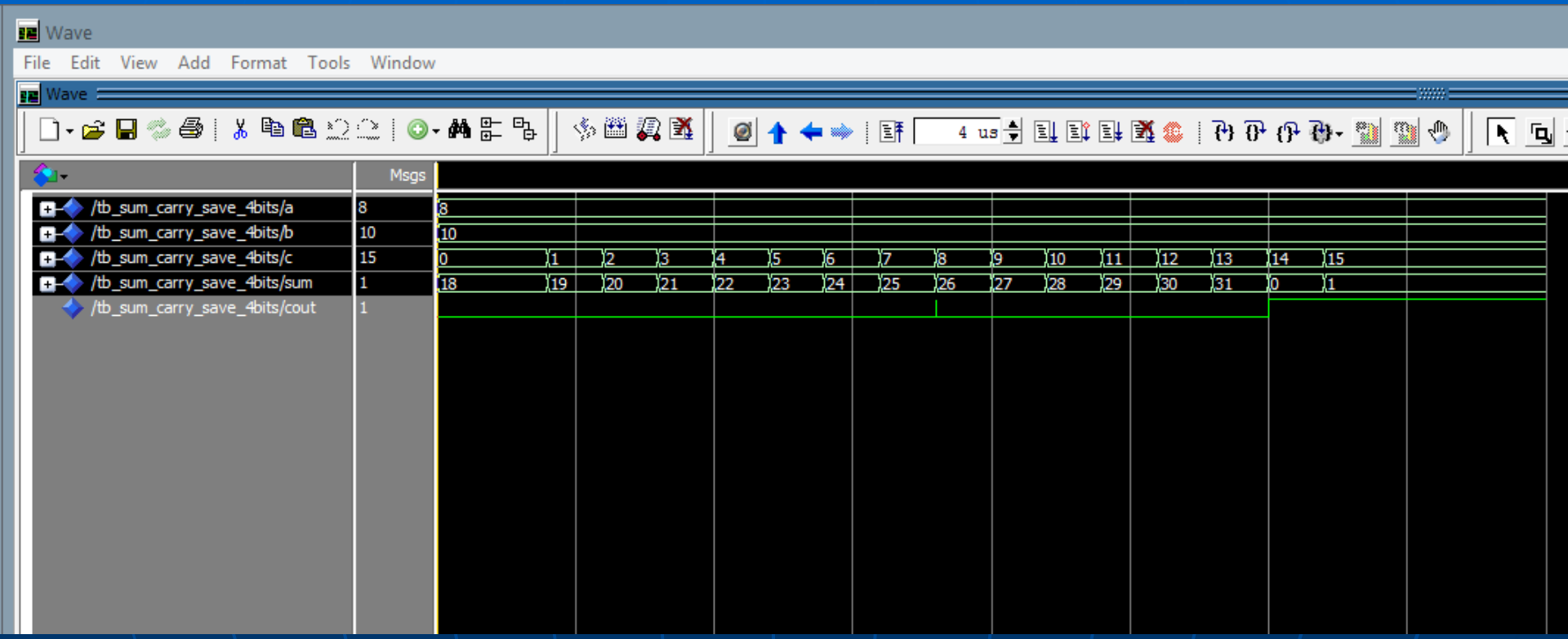
Test bench en VHDL

sum_carry_save_4bits.vhd | tb_sum_carry_save_4bits.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_sum_carry_save_4bits IS
6  END tb_sum_carry_save_4bits;
7
8  ARCHITECTURE behavior OF tb_sum_carry_save_4bits IS
9
10     COMPONENT sum_carry_save_4bits is
11     Port ( a    : in  unsigned (3 downto 0);
12            b    : in  unsigned (3 downto 0);
13            c    : in  unsigned (3 downto 0);
14            cout : out std_logic;
15            sum  : out unsigned (4 downto 0)
16               );
17         end COMPONENT;
18
19     signal a, b : unsigned(3 downto 0);
20     signal c    : unsigned(3 downto 0) := (others => '0');
21     signal sum  : unsigned(4 downto 0) := (others => '0');
22     signal cout : std_logic := '0';
23
24 BEGIN
25
26    uut: sum_carry_save_4bits PORT MAP (a, b, c, cout, sum);
27
28    estimulo_proc: process
29       begin
30          a <= to_unsigned (8,4);
31          b <= to_unsigned (10,4);
32          for i in 0 to 15 loop
33             wait for 200 ns;
34             c <= to_unsigned(i,4);
35          end loop;
36          wait;
37    end process;
38
39 END;
```

Diseño de un sumador de 3 operandos tipo carry-save

Resultado de la simulación



Diseño de un complementador a 2 (CA2) tipo serie

Descripción en VHDL

El chequear el valor de din después de procesar a data_out dentro del proceso, hace que se modifique data_out recién en el próximo ciclo de reloj, condición necesaria para ejecutar el algoritmo de complemento.

complementador_a2_serie.vhd | Compilation Report

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity complementador_a2_serie is
5     Port ( clock   : in std_logic;
6            reset   : in std_logic;
7            din     : in std_logic_vector(7 downto 0);
8            dout    : out std_logic_vector(7 downto 0)
9          );
10 end complementador_a2_serie;
11
12 architecture Comportamiento of complementador_a2_serie is
13
14    signal data_out, data_in  : std_logic_vector (7 downto 0);
15    signal lock               : integer range 0 to 1:= 0;
16    signal var                : integer range 0 to 8:= 0;
17
18    begin
19
20       data_in <= din;
21
22       process (reset, clock)
23          begin
24             if reset = '1' then
25             var <= 0;
26             lock <= 0;
27             elsif (clock'event and clock = '1') then
28                if var < 8 then
29                   if lock = 0 then data_out(var) <= din(var);
30                   else data_out(var) <= not din(var);
31                   end if;
32                   if din(var) = '1' then lock <= 1;
33                   end if;
34                   var <= var + 1;
35                end if;
36             end if;
37       end process;
38                   dout <= data_out;
39
40    end Comportamiento;
```

Diseño de un complementador a 2 (CA2) tipo serie

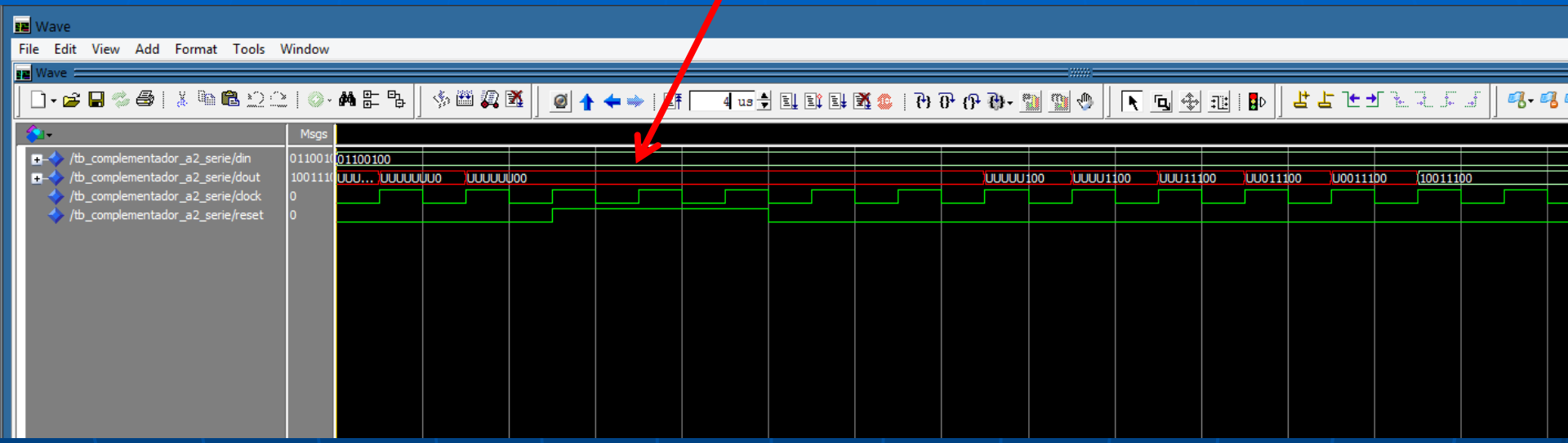
Test bench en VHDL

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity tb_complementador_a2_serie is
7   end tb_complementador_a2_serie;
8
9   architecture test of tb_complementador_a2_serie is
10
11
12    component complementador_a2_serie
13       Port (   clock        : in std_logic;
14                reset        : in std_logic;
15                din    :in std_logic_vector (7 downto 0);
16                dout   : out std_logic_vector (7 downto 0)
17
18             );
19    end component;
20
21    signal din, dout        : STD_LOGIC_VECTOR( 7 downto 0);
22    signal clock, reset     : STD_LOGIC;
23
24
25  begin
26
27    uut: complementador_a2_serie port map    ( clock => clock, reset => reset,
28                                               din => din, dout => dout
29                                  );
30
31     gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
32        begin
33           clock <= '0';
34           wait for 100 ns;
35           clock <= '1';
36           wait for 100 ns;
37     end process gen_reloj;
38
39     estimulos : process
40
41     begin
42              din <= "01100100";
43              reset    <= '0';
44              wait for 500 ns;
45              reset    <= '1';
46              wait for 500 ns;
47              reset    <= '0';
48              wait for 4 us;
49
50    end process estimulos;
51
52  end test;
```

Diseño de un complementador a 2 (CA2) tipo serie

Resultado de la simulación

"U" indica "no definido"



Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos SERIE

Descripción en VHDL

sum_ripple_carry_serie.vhd | Compilation Report | tb_sum_ripple_carry_serie.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity sum_ripple_carry_serie is
   Port ( clock   : in std_logic;
          reset   : in std_logic;
          a       : in std_logic_vector (7 downto 0);
          b       : std_logic_vector(7 downto 0);
          cin     : in std_logic;
          sum     : out std_logic_vector(7 downto 0);
          cout    : out std_logic
        );
end sum_ripple_carry_serie;

architecture Comportamiento of sum_ripple_carry_serie is

   signal suma, a_in, b_in : std_logic_vector(7 downto 0);
   signal cin_in, carry    : std_logic;

   begin

      a_in <= a;
      b_in <= b;
      cin_in <= cin;

      process (reset, clock)
         variable i      : integer range 0 to 8:= 0;
         begin
            if reset = '1' then
            i := 0;
            suma <= x"00";
            carry <= cin_in;
            elsif (clock'event and clock = '1') then
               if i < 8 then
                     suma(i) <= a_in(i) XOR b_in(i)XOR carry;
                     carry <= (a_in(i) AND b_in(i)) OR (carry AND (a_in(i) OR b_in(i)));
                     i := i + 1;
               end if;
            end if;
      end process;
      cout <= carry;
      sum <= suma;

   end Comportamiento;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos SERIE

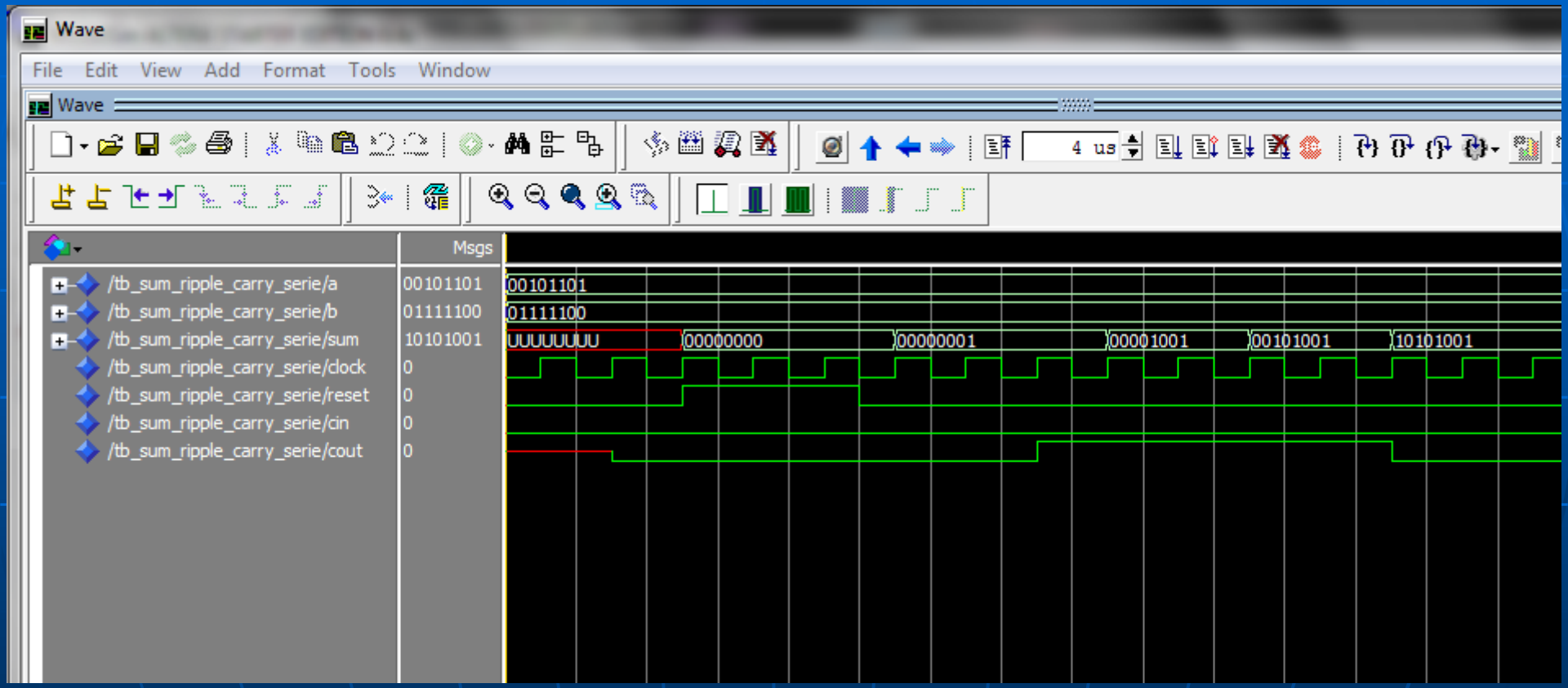
Test bench en VHDL

sum_ripple_carry_serie.vhd | Compilation Report | tb_sum_ripple_carry_serie.vhd

```
 6  entity tb_sum_ripple_carry_serie is
 7  end tb_sum_ripple_carry_serie;
 8
 9  architecture test of tb_sum_ripple_carry_serie is
10
11
12     component sum_ripple_carry_serie
13        Port (   clock : in std_logic;
14                 reset : in std_logic;
15                 a     : in std_logic_vector (7 downto 0);
16                 b     : in std_logic_vector (7 downto 0);
17                 cin   : in std_logic;
18                 sum   : out std_logic_vector (7 downto 0);
19                 cout  : out std_logic
20              );
21     end component;
22
23    signal a, b, sum                : std_logic_vector( 7 downto 0);
24    signal clock, reset, cin, cout  : std_logic;
25
26
27  begin
28
29    uut: sum_ripple_carry_serie port map     ( clock => clock, reset => reset,
30                                               a => a, b => b, cin => cin,
31                                               sum =>  sum, cout => cout
32                                    );
33
34     gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
35        begin
36           clock <= '0';
37           wait for 100 ns;
38           clock <= '1';
39           wait for 100 ns;
40     end process gen_reloj;
41
42     estimulos : process
43
44     begin
45              cin <= '0';
46              a <= "00101101";
47              b <= "01111100";
48              reset    <= '0';
49              wait for 500 ns;
50              reset    <= '1';
51              wait for 500 ns;
52              reset    <= '0';
53              wait for 4 us;
54
55    end process estimulos;
56
57  end test;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos SERIE

Resultado de la simulación



Diseño de un multiplicador sin signo paralelo con «template»

Descripción en VHDL

Usando el template se obtiene una plantilla de inicio, la cual puede ser modificada a conveniencia.

En este caso se cargó un multiplicador sin signo con longitud de palabra genérica ajustada inicialmente a 8 bits de entrada de datos.

multiplicador1.vhd | tb_multiplicador1.vhd

```
1  -- Quartus II VHDL Template
2  -- Unsigned Multiply
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.numeric_std.all;
7
8  entity multiplicador1 is
9
10    generic
11    (
12       DATA_WIDTH : natural := 8
13    );
14
15    port
16    (
17       a        : in unsigned ((DATA_WIDTH-1) downto 0);
18       b        : in unsigned ((DATA_WIDTH-1) downto 0);
19       result   : out unsigned ((2*DATA_WIDTH-1) downto 0)
20    );
21
22 end entity;
23
24 architecture rtl of multiplicador1 is
25 begin
26
27    result <= a * b;
28
29 end rtl;
30
```

Diseño de un multiplicador sin signo paralelo con «template»

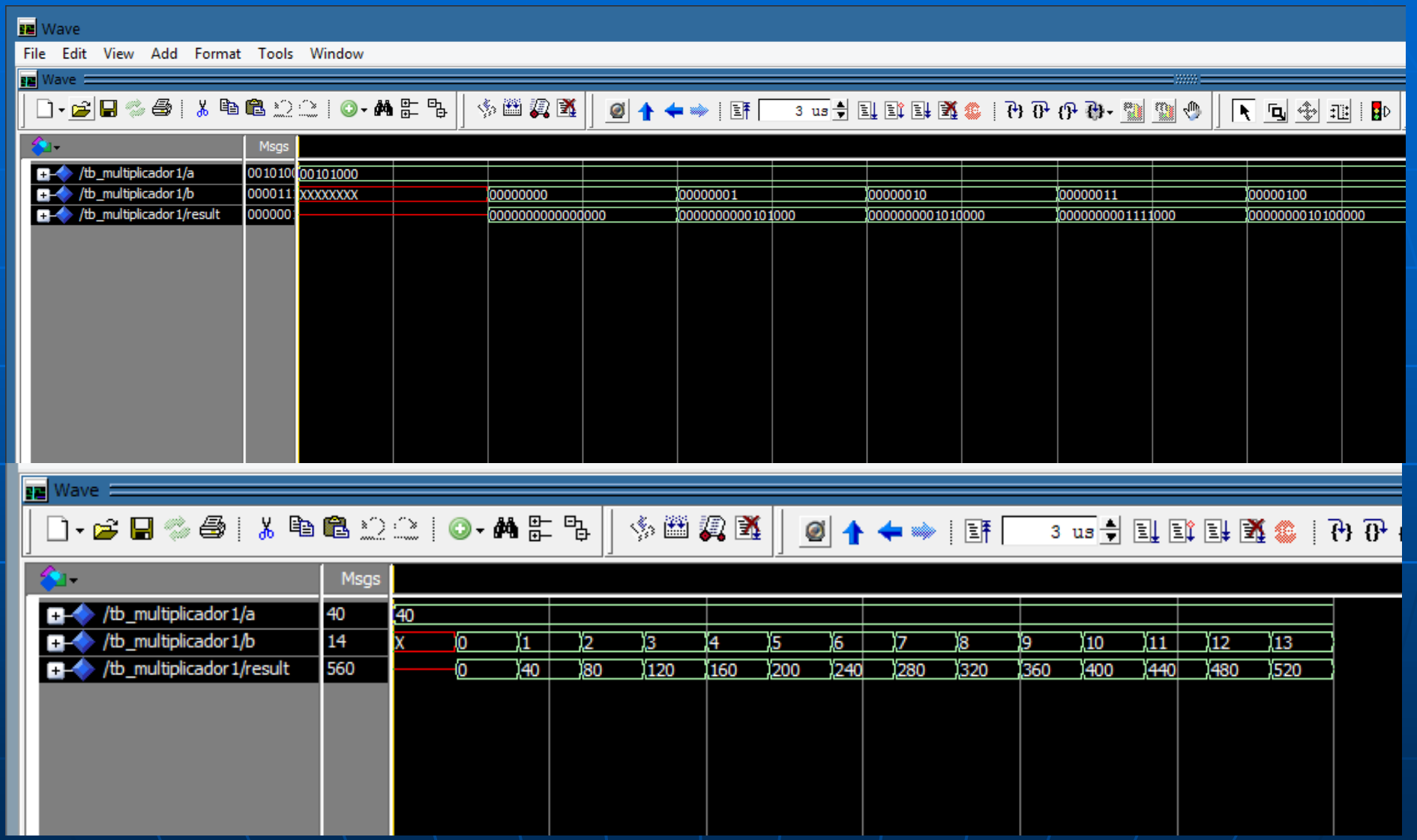
Test bench en VHDL

multiplicador1.vhd | tb_multiplicador1.vhd

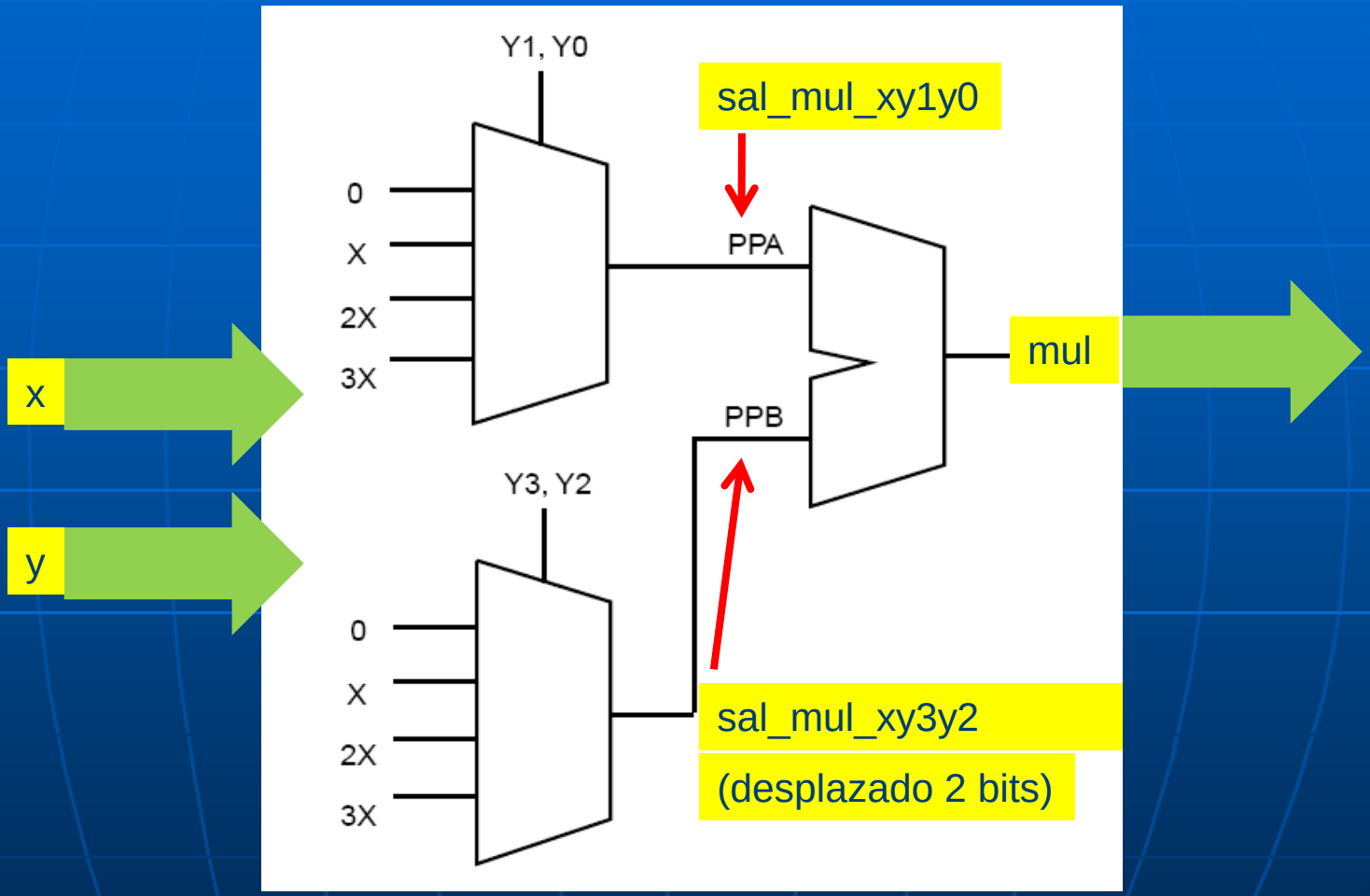
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_multiplicador1 IS
6
7     generic
8     (
9        DATA_WIDTH : natural := 8
10    );
11
12
13 END tb_multiplicador1;
14
15 ARCHITECTURE behavior OF tb_multiplicador1 IS
16
17    COMPONENT multiplicador1 is
18    Port ( a        : in unsigned ((DATA_WIDTH-1) downto 0);
19           b        : in unsigned ((DATA_WIDTH-1) downto 0);
20           result   : out unsigned ((2*DATA_WIDTH-1) downto 0)
21              );
22    end COMPONENT;
23
24    signal a, b    : unsigned((DATA_WIDTH-1) downto 0);
25    signal result  : unsigned((2*DATA_WIDTH-1)downto 0) := (others => '0');
26
27 BEGIN
28
29    uut: multiplicador1 PORT MAP (a, b, result);
30
31    estimulo_proc: process
32       begin
33          a <= to_unsigned(40,DATA_WIDTH);
34          for i in 0 to 15 loop
35             wait for 200 ns;
36             b <= to_unsigned(i,DATA_WIDTH);
37          end loop;
38          wait;
39    end process;
40
41 END;
```

Diseño de un multiplicador sin signo paralelo con «template»

Resultado de la simulación



Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits



Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits

Descripción en VHDL

```
6   library IEEE;
7   use IEEE.STD_LOGIC_1164.ALL;
8   use IEEE.NUMERIC_STD.ALL;
9
10  entity mul_booth_4bits is
11      Port ( x       : in  unsigned (3 downto 0);
12             y       : in  unsigned (3 downto 0);
13             s_xy1y0 : out unsigned (7 downto 0);
14             s_xy3y2 : out unsigned (7 downto 0);
15             mul     : out unsigned (7 downto 0)
16                );
17  end mul_booth_4bits;
18
19  architecture Behavioral of mul_booth_4bits is
20
21  signal sal_mul_x_y1y0, sal_mul_x_y3y2,xr,yr : unsigned (7 downto 0);
22  signal sal : unsigned (7 downto 0);
23
24  begin
25
26          xr <= resize(x,8);
27          yr <= resize(y,8);
28
29  mul_x_y1y0 : process (xr, yr)
30              begin
31                 if ((not yr(1) and not yr(0)) = '1')  then sal_mul_x_y1y0 <= "00000000";
32                 elsif ((not yr(1) and yr(0))  = '1')  then sal_mul_x_y1y0 <= xr;
33                 elsif ((yr(1) and not yr(0))  = '1')  then sal_mul_x_y1y0 <= shift_left(xr,1);
34                 elsif ((yr(1) and yr(0))  = '1')      then sal_mul_x_y1y0 <= xr + shift_left(xr,1);
35                 end if;
36          end process;
37
38  mul_x_y3y2 : process (xr, yr)
39              begin
40                 if ((not yr(3) and not yr(2)) = '1' ) then sal_mul_x_y3y2 <= "00000000";
41                 elsif ((not yr(3) and yr(2)) = '1' ) then sal_mul_x_y3y2 <= xr;
42                 elsif ((yr(3) and not yr(2)) = '1' ) then sal_mul_x_y3y2 <= shift_left(xr,1);
43                 elsif ((yr(3) and yr(2)) = '1' ) then sal_mul_x_y3y2 <= xr + shift_left(xr,1);
44                 end if;
45          end process;
46
47  --suma  : process (sal_mul_x_y1y0, sal_mul_x_y3y2)
48  --          begin
49                sal <= sal_mul_x_y1y0 + shift_left(sal_mul_x_y3y2,2);
50  --        end process;
51               mul <= sal;
52               s_xy1y0 <= sal_mul_x_y1y0;
53               s_xy3y2 <= sal_mul_x_y3y2;
54  end Behavioral;
```

Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits

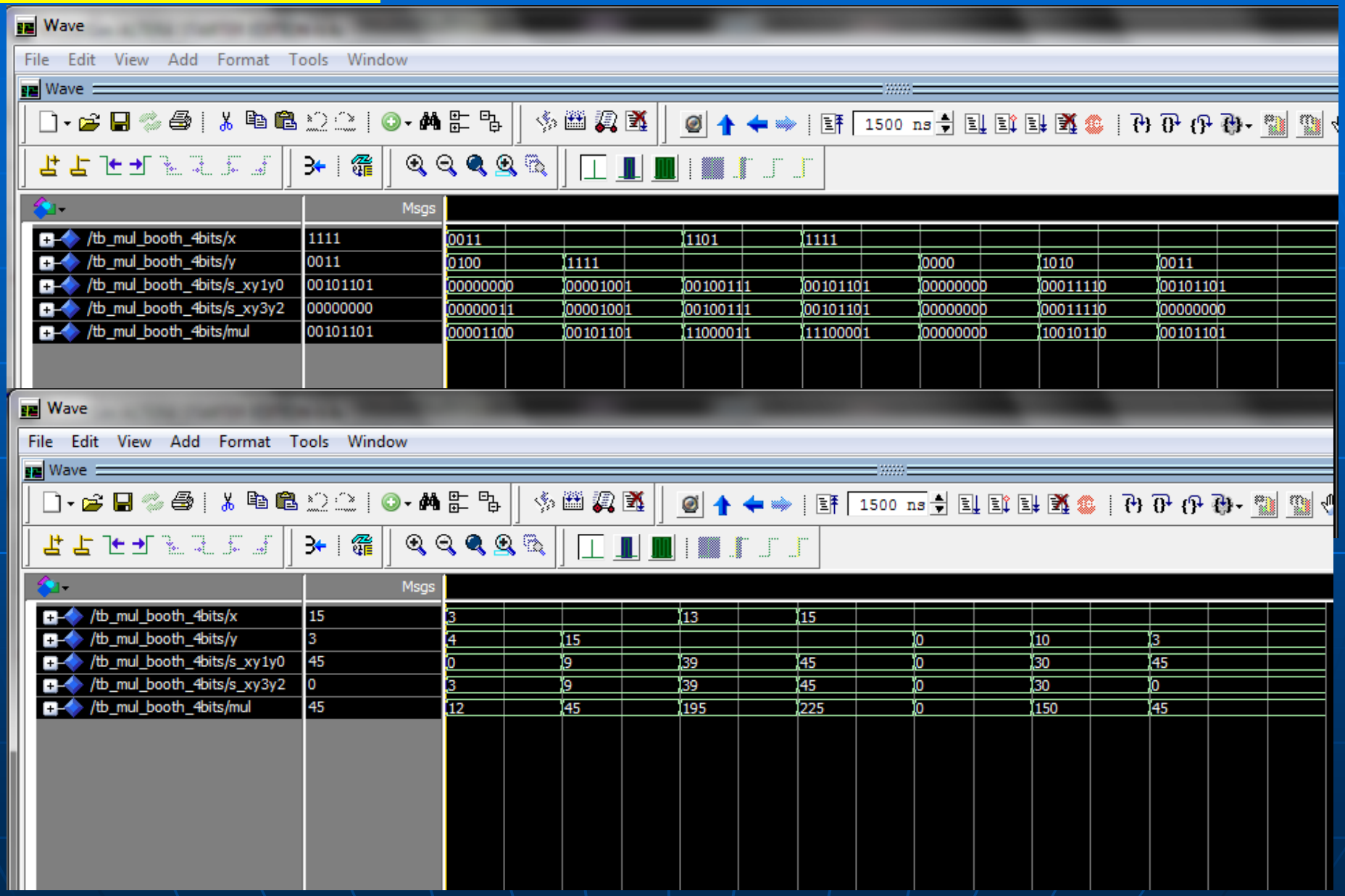
tb_mul_booth_4bits.vhd

Test bench en VHDL

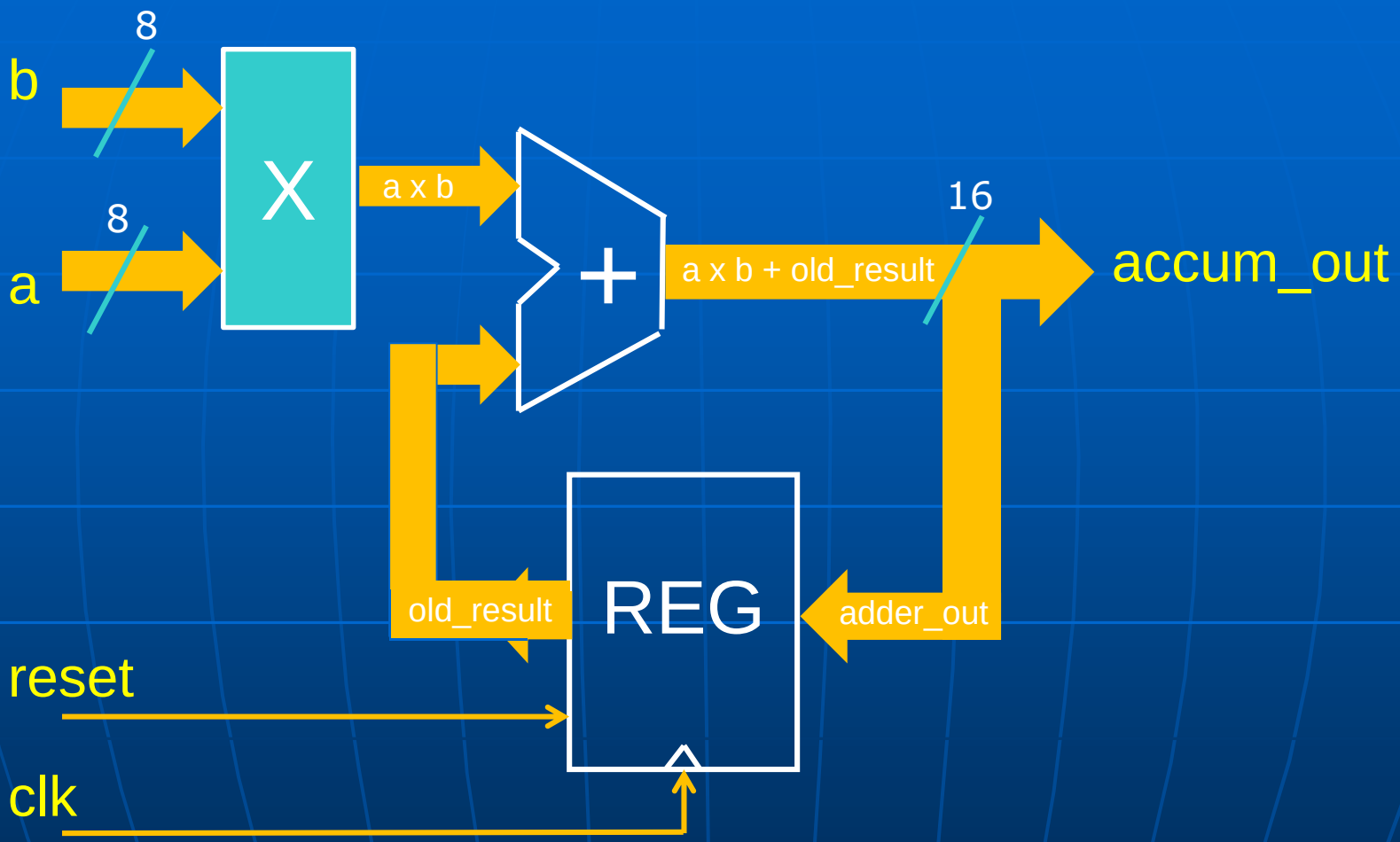
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_mul_booth_4bits IS
6  END tb_mul_booth_4bits;
7
8  ARCHITECTURE behavior OF tb_mul_booth_4bits IS
9
10     COMPONENT mul_booth_4bits is
11     Port ( x       : in  unsigned (3 downto 0);
12            y       : in  unsigned (3 downto 0);
13            s_xy1y0 : out unsigned (7 downto 0);
14            s_xy3y2 : out unsigned (7 downto 0);
15            mul     : out unsigned (7 downto 0)
16               );
17         end COMPONENT;
18
19     signal x, y             : unsigned(3 downto 0);
20     signal s_xy1y0, s_xy3y2 : unsigned(7 downto 0);
21     signal mul              : unsigned(7 downto 0);
22
23 BEGIN
24
25    uut: mul_booth_4bits PORT MAP (x, y, s_xy1y0, s_xy3y2, mul);
26
27    estimulo_proc: process
28       begin
29          x <= "0011";
30          y <= "0100";
31          wait for 200 ns;
32          y <= "1111";
33          wait for 200 ns;
34          x <= "1101";
35          wait for 200 ns;
36          x <= "1111";
37          wait for 200 ns;
38          y <= "0000";
39          wait for 200 ns;
40          y <= "1010";
41          wait for 200 ns;
42          y <= "0011";
43          wait;
44    end process;
45
46 END;
```

Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits

Resultado de la simulación



Diseño de un multiplicador – acumulador paralelo con signo de 8 bits



Diseño de un multiplicador – acumulador paralelo con signo de 8 bits

Descripción en VHDL

mult_acum_con_signo.vhd | tb_mult_acum_con_signo.vhd

```
1  --Ejemplo de multiplicador mas acumulador
2  -- Sergio Noriega ISLD 2016
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  entity mult_acum_con_signo is
8     port
9     (
10       a           : in signed(7 downto 0);
11       b           : in signed (7 downto 0);
12       clk         : in std_logic;
13       reset       : in std_logic;
14       accum_out   : out signed (15 downto 0)
15    );
16
17 end entity;
18
19 architecture rtl of mult_acum_con_signo is
20
21    signal a_reg : signed (7 downto 0);
22    signal b_reg : signed (7 downto 0);
23    signal mult_reg : signed (15 downto 0);
24    signal adder_out : signed (15 downto 0);
25    signal old_result : signed (15 downto 0);
26
27 begin
28          mult_reg <= a_reg * b_reg;
29          adder_out <= old_result + mult_reg;
30
31          a_reg <= a;
32          b_reg <= b;
33
34    process (clk, reset)
35    begin
36       if (reset = '1') then
37          old_result <= (others => '0');
38       elsif (rising_edge(clk)) then
39       old_result <= adder_out;
40       accum_out <= adder_out;
41       end if;
42
43    end process;
44
45 end rtl;
46
```

Diseño de un multiplicador – acumulador paralelo con signo de 8 bits

Test bench en VHDL

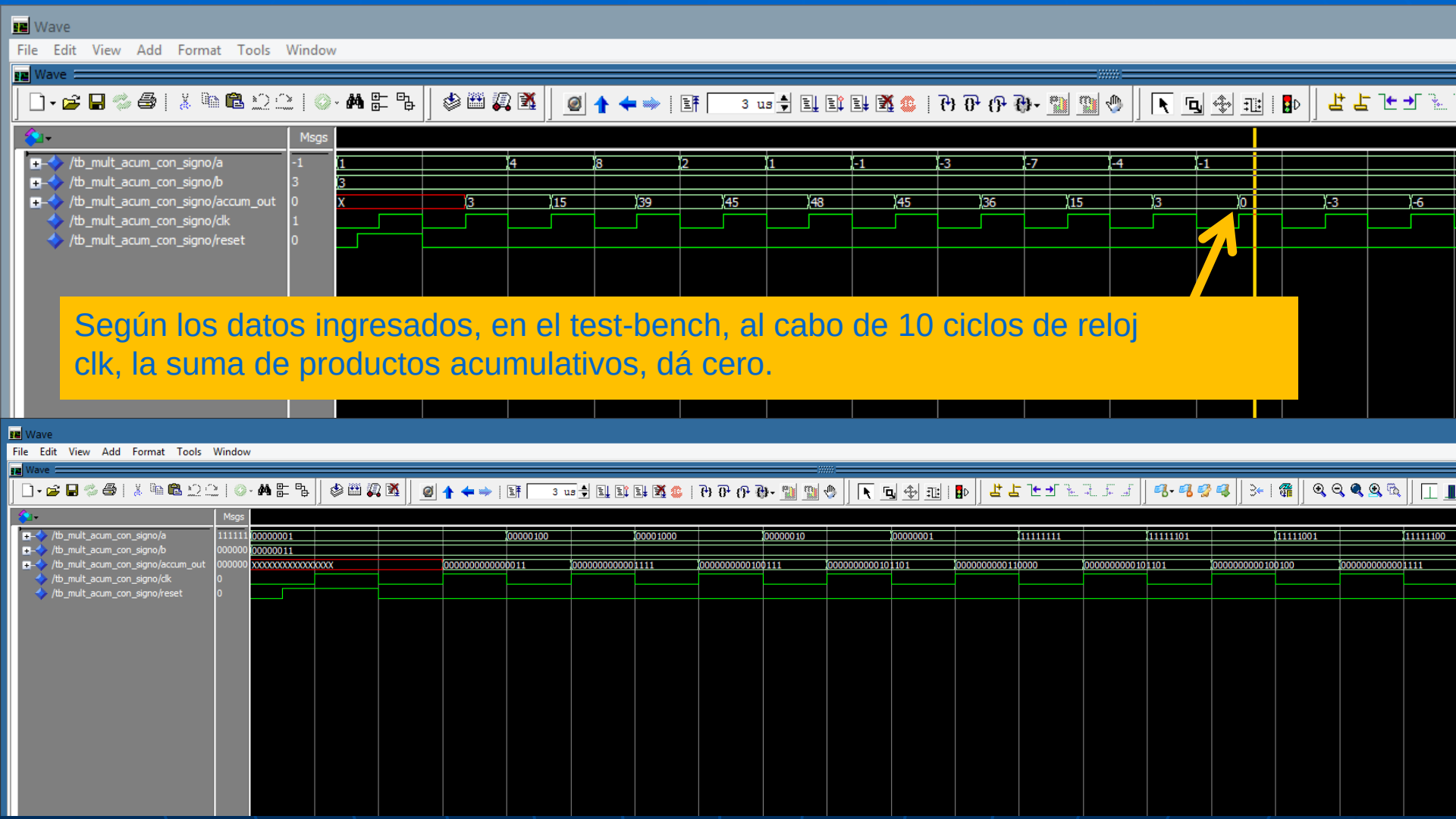
En este test-bench se mantiene constante la entrada ‘b’ en el valor +3 y se entran 10 valores de ‘a’ en sincronía con el reloj: +1, +4, +8, +2, +1, -1, -3, -7, -4, -1.

mult_acum_con_signo.vhd | tb_mult_acum_con_signo.vhd

```
1  LIBRARY ieee; USE ieee.std_logic_1164.ALL; use ieee.numeric_std.all;
2
3  ENTITY tb_mult_acum_con_signo IS
4  END tb_mult_acum_con_signo;
5
6  ARCHITECTURE behavior OF tb_mult_acum_con_signo IS
7
8      COMPONENT mult_acum_con_signo is
9      port (
10      a           : in signed(7 downto 0);
11      b           : in signed (7 downto 0);
12      clk         : in std_logic;
13      reset       : in std_logic;
14      accum_out   : out signed (15 downto 0));
15   end COMPONENT;
16
17    signal a, b             : signed(7 downto 0);
18    signal accum_out        : signed(15 downto 0);
19    signal clk, reset       : std_logic;
20
21 type datos is array (integer range 0 to 9) of signed(7 downto 0);
22
23 BEGIN
24
25  uut: mult_acum_con_signo PORT MAP (a, b, clk, reset, accum_out);
26
27    gen_reloj : process -- Reloj de 200 ns de periodo
28       begin
29          clk <= '0';
30          wait for 100 ns;
31          clk <= '1';
32          wait for 100 ns;
33    end process gen_reloj;
34
35    estimulo_proc: process
36    variable muestras: datos;
37             begin
38          muestras := ("00000001", "00000100", "00001000","00000010","00000001",
39                       "11111111", "11111101", "11111001","11111100","11111111");
40                       --(1, 4, 8, 2, 1, -1, -3, -7, -4, -1)
41          a <= muestras (0);
42          b <= "00000011";
43          reset <= '0';
44          wait for 50 ns;
45          reset <= '1';
46          wait for 150 ns;
47          reset <= '0';
48          wait for 200 ns;
49          for i in 1 to 9 loop
50             a <= muestras(i);
51             wait for 200 ns;
52          end loop;
53          wait;
54    end process;
55 END;
```

Diseño de un multiplicador – acumulador paralelo con signo de 8 bits

Resultado de la simulación



Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754

Single-Precision Format

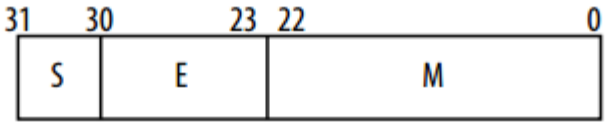
The single-precision format contains the following binary patterns:

- The MSB holds the sign bit.
- The next 8 bits hold the exponent bits.
- 23 LSBs hold the mantissa.

The total width of a floating-point number in the single-precision format is 32 bits. The bias for the single-precision format is 127.

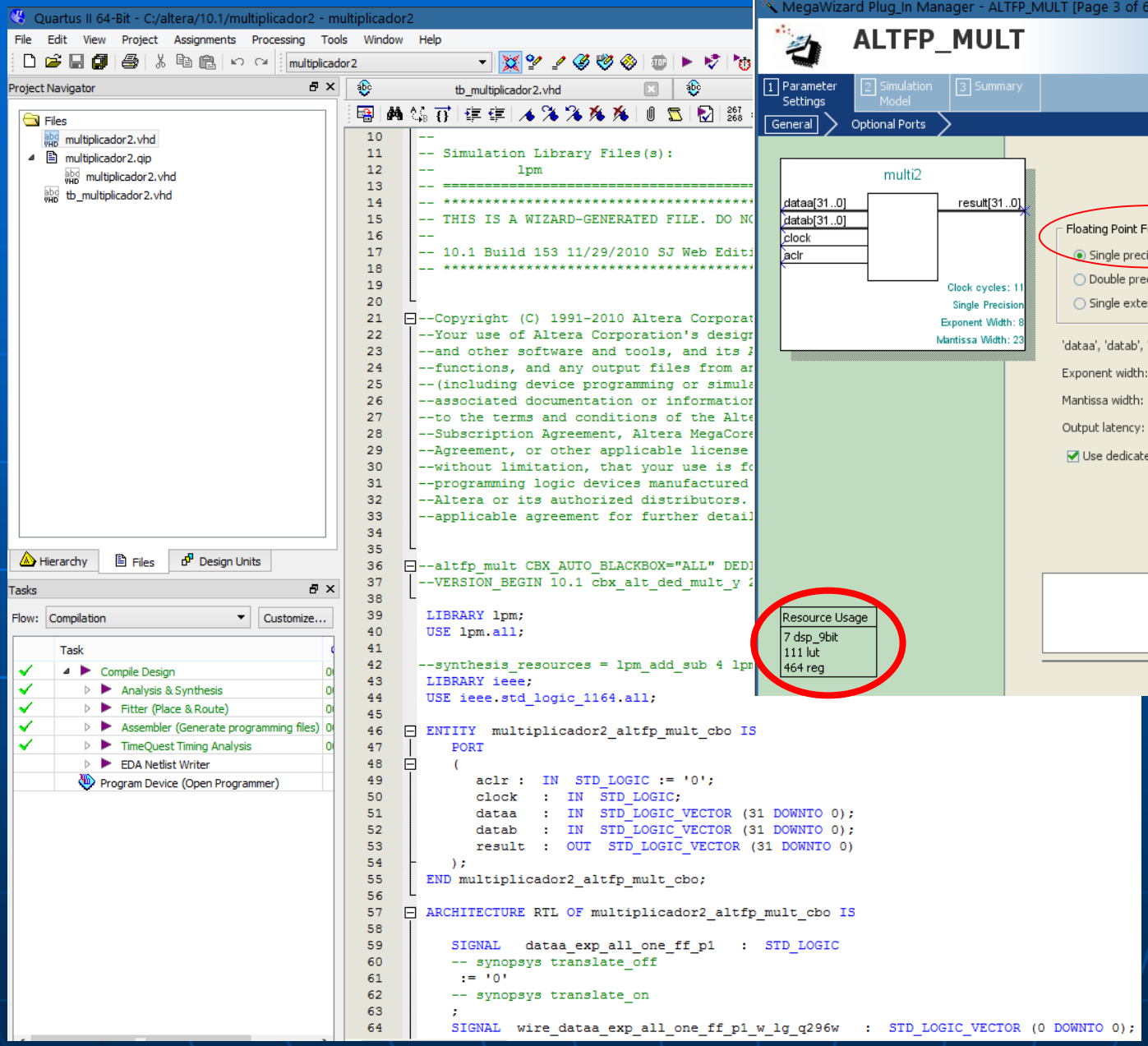
Figure 1-10: Single-Precision Representation

This figure shows a single-precision representation.



Referencia: Nota de aplicación de Altera 'Floating-Point IP Cores User Guide': 'ug_altfp_mfug.pdf' año 2015.

Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754



Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754

Test bench en VHDL

tb_multiplicador2.vhd | multiplicador2.qip | multiplicador2.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4  LIBRARY lpm;
5  USE lpm.all;
6
7  ENTITY tb_multiplicador2 IS
8  END tb_multiplicador2;
9
10 ARCHITECTURE behavior OF tb_multiplicador2 IS
11
12
13     COMPONENT multiplicador2 is
14     PORT
15     (
16        aclr :  IN  STD_LOGIC := '0';
17        clock   :  IN  STD_LOGIC;
18        dataa   :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
19        datab   :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
20        result  :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
21     );
22  END COMPONENT;
23
24
25
26     signal dataa, datab, result   : STD_LOGIC_VECTOR (31 DOWNTO 0);
27     signal aclr, clock : STD_LOGIC;
28
29 BEGIN
30
31    uut: multiplicador2 PORT MAP (aclr, clock, dataa, datab, result);
32
33    gen_reloj : process -- Reloj de 20 ns de periodo y 50% de ciclo de trabajo
34       begin
35          clock <= '0';
36          wait for 100 ns;
37          clock <= '1';
38          wait for 100 ns;
39    end process gen_reloj;
40
41    estimulo_proc: process
42       begin
43          dataa <= "00111111111000000000000000000000";
44          datab <= "11000000010000000000000000000000";
45          aclr <= '0';
46          wait for 300 ns;
47          aclr <= '1';
48          wait for 150 ns;
49          aclr <= '0';
50          wait for 5 us;
51    end process;
52
53 END;
```

$+1,75_{10}$

-3_{10}

Positivo — equivale a ‘1,11’

dataa = 00111111111000000……0

equivale a ’2^0’

Negativo — equivale a ‘1,10’

datab = 11000000011000000……0

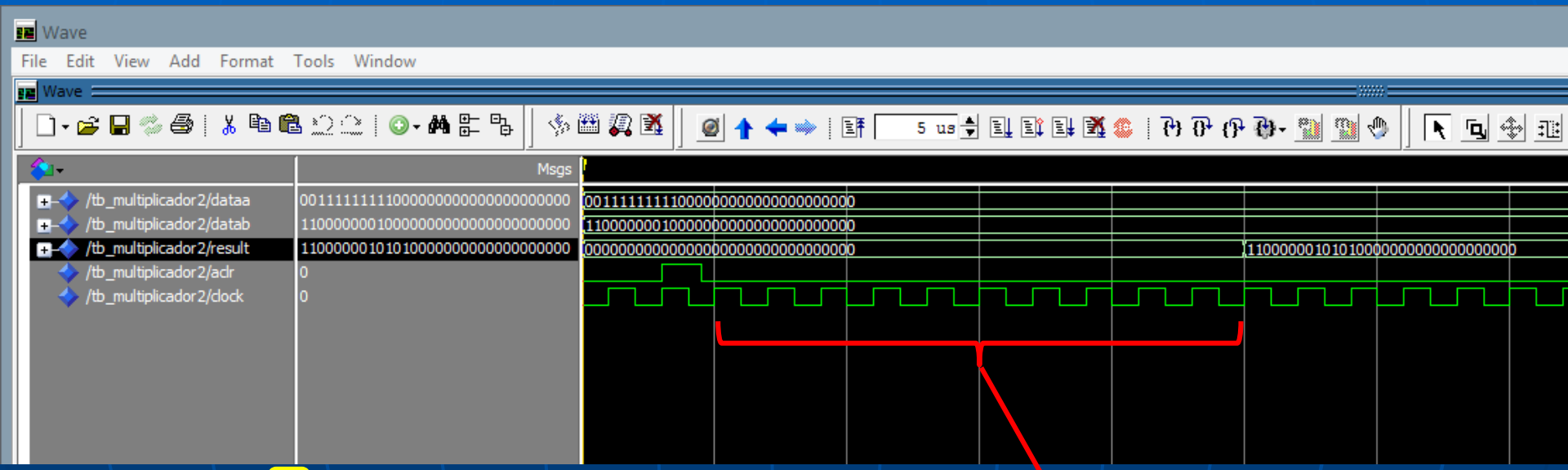
equivale a ’2^1’

Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754

Resultado de la simulación

$(1,75) \times (-3) = -5,25_{10}$

dataa datab result



result : 1 10000001 01010.....0

equivale a '2^{2}' equivale a '1,0101'

(-) 101,01

11 ciclos de reloj de latencia para generar el resultado de la multiplicación.

Diseño de Unidad Aritmético-Lógica (ALU)

Ejemplo: Diseño de ALU de 8 bits con operaciones aritméticas sin signo

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

ENTITY alu3 IS
    PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
          cin: IN STD_LOGIC;
          sal: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END alu3;

ARCHITECTURE alucomp3 OF alu3 IS
    SIGNAL aritme, logica, swap_a: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
    swap_a(7 DOWNTO 4) <= a(3 DOWNTO 0);
    swap_a(3 DOWNTO 0) <= a(7 DOWNTO 4);
--Sección de operaciones aritméticas (sin signo)
    WITH sel(3 DOWNTO 0) SELECT
        aritme <= a WHEN "0000",        --Transferencia del operando "a"
                  a+1 WHEN "0001",      --Incremento del operando "a"
                  a-1 WHEN "0010",      --Decremento del operando "a"
                  b WHEN "0011",        --Transferencia del operando "b"
                  b+1 WHEN "0100",      --Incremento del operando "b"
                  b-1 WHEN "0101",      --Decremento del operando "b"
                  a+b WHEN "0110",      --Suma sin signo
                  a+b+cin WHEN "0111", --Suma sin signo con carry
                  a-b WHEN "1000",      --Resta sin signo
                a-b-cin WHEN "1001",    --Resta sin signo con borrow
                a*b WHEN "1010",        --Multiplicación sin signo
                a WHEN OTHERS;          --Evita generación de lógica extra
```

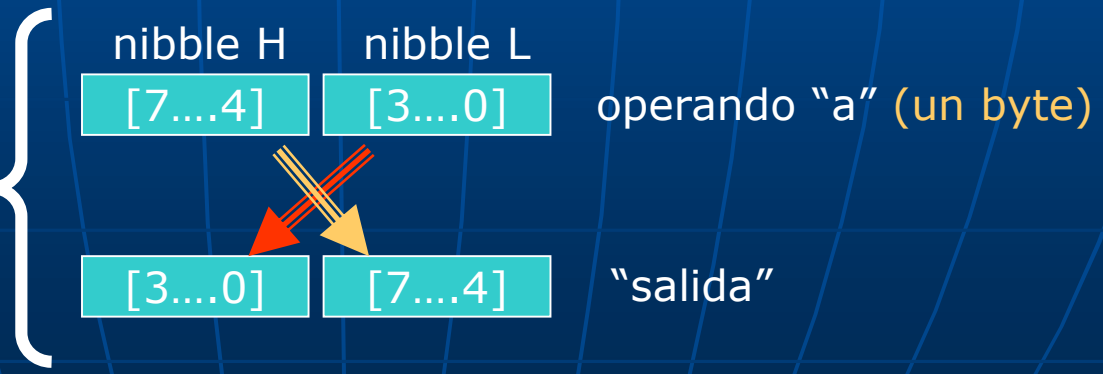
sentencias para implementar la operación “swap” del operando “a”

Diseño de Unidad Aritmético-Lógica (ALU)

Ejemplo: Diseño de ALU de 8 bits con operaciones aritméticas sin signo (continuación)

```
--Sección de operaciones lógicas (operaciones bit a bit)
    WITH sel(3 DOWNTO 0) SELECT
        logica <= NOT a WHEN "0000",   --Negación del operando "a"
                  NOT b WHEN "0001",   --Negación del operando "b"
                  a AND b WHEN "0010", --Operación AND e/ "a" y "b"
                  a OR b WHEN "0011",  --Operación OR e/ "a" y "b"
                  a NAND b WHEN "0100",--Operación NAND e/ "a" y "b"
                  a NOR b WHEN "0101", --Operación NOR e/ "a" y "b"
                  a XOR b WHEN "0110", --Operación XOR e/ "a" y "b"
                  a XNOR b WHEN "0111",--Operación NOT(XOR) e/ "a" y "b"
                  swap_a WHEN "1000",  --Intercambio de nibles en "a"
                  a WHEN OTHERS;       --Evita generación de lógica extra
    WITH sel(4) SELECT
        sal <= aritme WHEN '0',
               logica WHEN OTHERS;
END alucomp3;
```

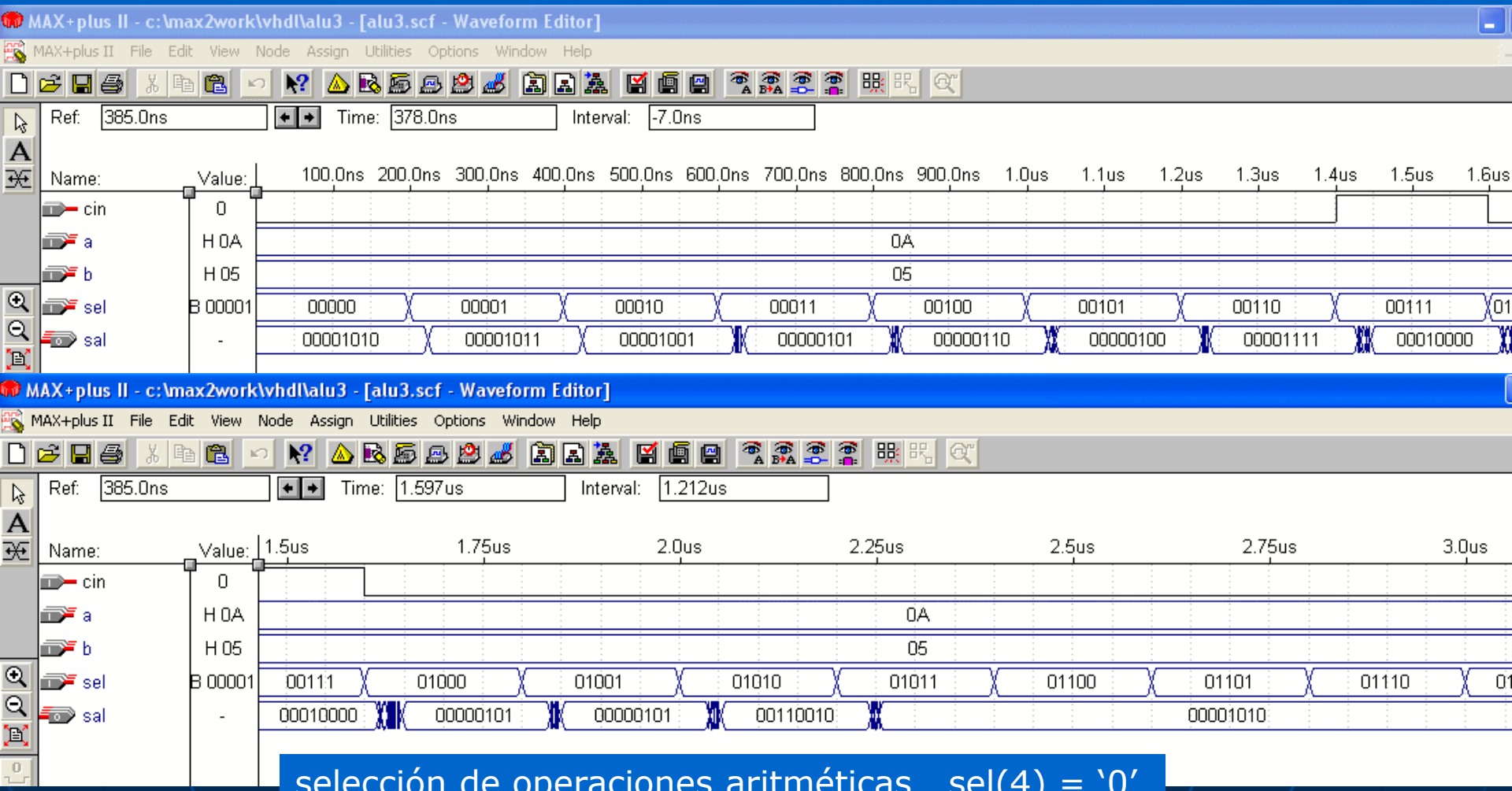
Ejemplo de operación swap para el operando "a" ("sel" = "11000")



Diseño de Unidad Aritmético-Lógica (ALU)

Diseño de ALU de 8 bits con operaciones aritméticas sin signo (continuación)

Simulación

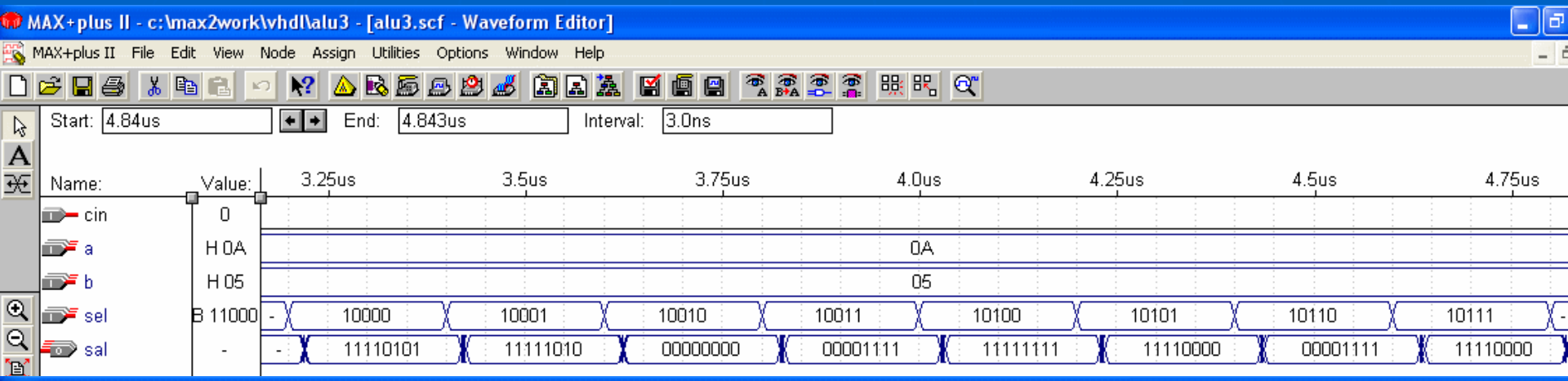


selección de operaciones aritméticas sel(4) = ‘0’

Diseño de Unidad Aritmético-Lógica (ALU)

Diseño de ALU de 8 bits con operaciones aritméticas sin signo (continuación)

Simulación



selección de operaciones lógicas sel(4)='1'

Diseño de un control para dos SEMÁFOROS

semaforo.vhd

```
1  --*************************************************************************************************
2  -- Descripcion en VHDL de un controlador de semaforos
3  -- En este proyecto se describe un controlador de dos semaforos "1" y "2"con modos
4  -- diurno y nocturno, controlado por la entrada "modo".
5  -- Las salidas de las 3 lamparas por semaforo se presentan en "salida" donde:
6  -- salida(5) = luz verde semaforo 2, V2.
7  -- salida(4) = luz amarilla semaforo 2, A2.
8  -- salida(3) =  luz roja semaforo 2, R2.
9  -- salida(2) = luz verde semaforo 1, V1.
10 -- salida(1) = luz amarilla semaforo 1, A1.
11 -- salida(0) =  luz roja semaforo 1, R1.
12 -- La secuencia de encendido de las lmaparas se hace de la siguiente manera:
13 -- Modo diurno (modo = '0'): R2-R1 -> R2-V1 -> R2-A1 -> R2-R1 -> V2-R1 -> A2-R1 -> REPITE
14 -- Modo nocturno (modo = '1'): apagago-apagado -> A2-A1 -> REPITE.
15 -- Para hacer el sistema mas realistico se definen ciclos de reloj variable segun en que
16 -- estado se encuentre:
17 -- R2-R1 dura  3000 ms.          0BB8 EN HEXADECIMAL.
18 -- R2-V1 dura 20000 ms.          4E20 "       "     .
19 -- R2-A1 y A2-R1 duran 2000 ms.  07D0 "       "     .
20 -- V2-R1 dura 40000 ms.          9C40 "       "     .
21 -- apagado-apagado dura 1000 ms. 03E8 EN HEXADECIMAL.
22 -- A2-A1 dura 1000 ms.           03E8 EN HEXADECIMAL.
23 -- Para ello se usa un reloj base de 50 MHz que se divide hasta 1KHz (1 ms de periodo).
24 -- Ese reloj entra a un contador preseteable con un dato que se modifica (count_max) cada vez
25 -- que se pasa a un nuevo estado.
26 -- Resultado: Funciona OK.
27 -- Archivo: semaforo.vhd
28 -- Sergio Noriega ISLD 2016
29 -- *************************************************************************************************
30
31 library ieee;
32 use ieee.std_logic_1164.all;
33 use ieee.std_logic_arith.all;
34 use ieee.std_logic_unsigned.all;
35
36 entity semaforo is
37
38    Port ( clock    : in std_logic;     --ENTRADA DE 50 MHZ
39           modo     : in std_logic;     --Entrada para ajuste diurno-nocturno
40           reset    : in std_logic;     --Reset
41           salida   : out std_logic_vector (5 downto 0)
42         );
43
44 end semaforo;
45
46 architecture Comportamiento of semaforo is
47
48    signal clock_1ms, clock_div, temporal, clock_prog : std_logic;
49    signal count_max                    : std_logic_vector (15 downto 0); --Hasta 65 segundos
50    signal counter                      : integer range 0 to 24999 := 0;
51    signal counter2                     : integer range 0 to 60000 := 0;
52
53    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7);
54    signal estado : state_type;
55
```

Diseño de un control para dos SEMÁFOROS

Para que sea mas realista se deben obtener diferentes tiempos en cada etapa de la secuencia. Además en este ejemplo necesitamos que el semáforo 1 esté en VERDE unos 20 segundos y el VERDE del semáforo 2, encendido durante 40 segundos.

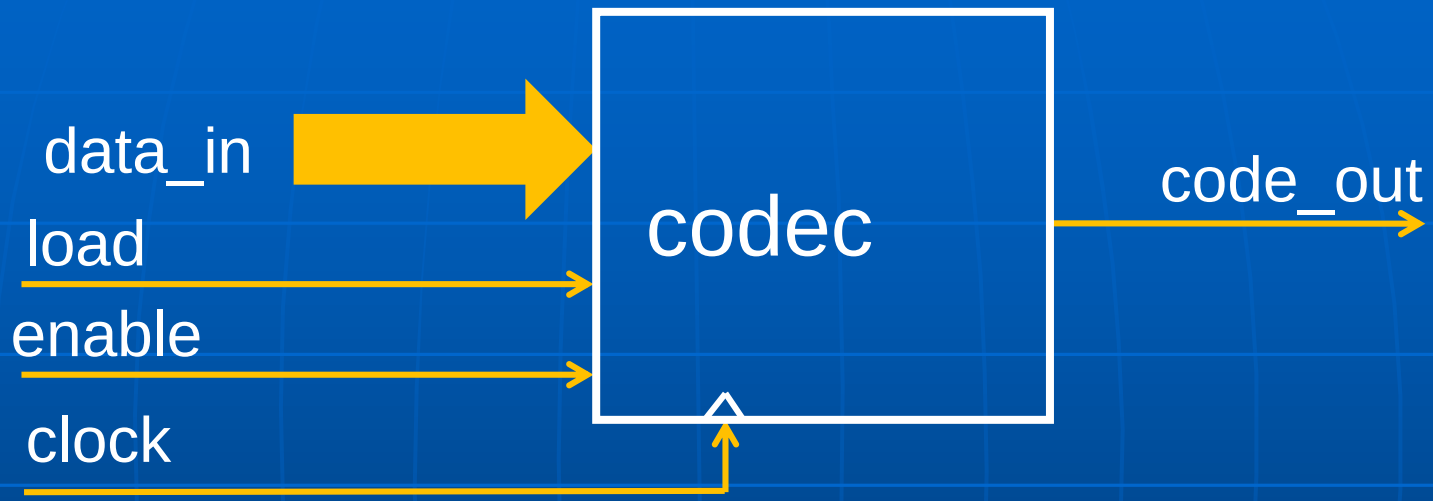
Cómo lograr con máquina de estados, modificar el tiempo en el que se está en cada estado?

```
56  begin
57
58   divisor_de_frec: process (reset, clock) -- Genera un reloj de 1ms de periodo
59                       begin
60                        if reset = '1' then temporal <= '0';
61                                            counter <= 0;
62                         elsif rising_edge(clock) then
63                          if (counter = 24999) then
64                             temporal <= NOT(temporal);
65                             counter <= 0;
66                          else
67                             counter <= counter + 1;
68                          end if;
69                          end if;
70                    end process;
71
72    clock_1ms <= temporal; --Onda cuadrada de 1KHz y 50% de duty cycle.
73
74  clock_var :   process (clock_1ms, reset)  --Genera el reloj de periodo variable
75                 begin
76                    if rising_edge (clock_1ms) then
77                     if counter2 < count_max  then
78                     clock_prog <= '0';
79                    counter2 <= counter2 + 1;
80                    else counter2 <= 0; clock_prog <= '1';
81                    end if;
82                    end if;
83               end process;
84
85         clock_div <= clock_prog;
86
```

Diseño de un control para dos SEMÁFOROS

```
86
87  gen_salidas:  process (clock_div, reset, modo)
88                begin
89                   if reset = '1' then estado <= s0;
90                   elsif (clock_div'event and clock_div = '1') then
91                      case estado is
92                         when s0 => if modo = '0' then estado <= s1;
93                                    else estado <= s7; end if;
94                         when s1 => if modo = '0' then estado <= s2;
95                                    else estado <= s0; end if;
96                         when s2 => if modo = '0' then estado <= s3;
97                                    else estado <= s0; end if;
98                         when s3 => if modo = '0' then estado <= s4;
99                                    else estado <= s0; end if;
100                        when s4 => if modo = '0' then estado <= s5;
101                                   else estado <= s0; end if;
102                        when s5 => if modo = '0' then estado <= s6;
103                                   else estado <= s0; end if;
104                        when s6 => if modo = '0' then estado <= s1;
105                                   else estado <= s0; end if;
106                        when s7 => if modo = '0' then estado <= s0;
107                                   else estado <= s0; end if;
108                     end case;
109                  end if;
110               end process;
111
112         process (estado)
113            begin
114               case estado is
115                  when s0 => salida <= "000000"; --Todas las luces apagadas
116                             count_max <= x"03E8"; -- 1 segundo
117                  when s1 => salida <= "001001"; --Prendidas R2 y R1
118                             count_max <= x"0BB8"; -- 3 segundos
119                  when s2 => salida <= "001100"; --Prendidas R2 y V1
120                             count_max <= x"4E20"; --  20 segundos
121                  when s3 => salida <= "001010"; --Prendidas R2 y A1
122                             count_max <= x"07D0";  -- 2 segundos
123                  when s4 => salida <= "001001"; --Prendidas R2 y R1
124                             count_max <= x"0BB8";  -- 3 segundos
125                  when s5 => salida <= "100001"; --Prendidas V2 y R1
126                             count_max <= x"9C40";  -- 40 segundos
127                  when s6 => salida <= "010001"; --Prendidas A2 y R1
128                             count_max <= x"07D0";  -- 2 segundos
129                  when s7 => salida <= "010010"; --Prendidas A2 y A1
130                             count_max <= x"03E8";  -- 1 segundo
131               end case;
132         end process;
133
134   end Comportamiento;
```

Diseño de un codificador manchester de 8 bits



El dato paralelo se lo serializa y se envían dos bits por bit de dato.
Se envía '01' si el dato es '0' y '10' si el dato es '1'.
Esto garantiza la posibilidad en el receptor de recuperación del reloj para grandes cadenas de datos con '0s' o con '1s'.

Diseño de un codificador manchester de 8 bits

Descripción en VHDL

'load' carga y convierte el dato de 8 bits en un arreglo de 16 bits con la conversión requerida.
'enable' habilita a que se envíen los 16 bits ya convertidos en forma serie a través del registro 'shiftreg'.

Un contador 'count' se encarga de parar la transferencia una vez que se hayan transmitido los 16 bits.

```
1  --Codificador Manchester
2  --Archivos: codec_man (proyecto) y tb_codec_man (test bench)
3  --Sergio Noriega 2016
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.std_logic_arith.all;
7  use ieee.std_logic_unsigned.all;
8
9  entity codec_man is
10
11    Port ( clock    : in std_logic;
12           enable   : in std_logic;
13           data_in  : in std_logic_vector (7 downto 0);
14           load     : in std_logic;
15           code_out : out std_logic
16         );
17
18 end codec_man;
19
20 architecture Comportamiento of codec_man is
21
22    signal shiftreg : std_logic_vector (15 downto 0);
23    signal count    : integer range 0 to 16;
24
25    begin
26
27
28    genera_code: process (clock, enable, load, count)
29                         begin
30                           if clock'event and clock = '1' then
31                            if load = '1' then
32                               count <= 0;
33                               for i in 0 to 7 loop
34                                  shiftreg (2*i) <= not data_in(i);
35                                  shiftreg (2*i +1) <= data_in(i);
36                               end loop;
37                               else if (enable = '1' and count < 16)then
38                                   shiftreg <= shiftreg(14 downto 0)&'0';
39                                   count <= count +1;
40                           end if;
41                          end if;
42                          end if;
43                end process;
44
45    code_out <= shiftreg(15);
46
47
48 end Comportamiento;
49
```

Diseño de un codificador manchester

Test bench en VHDL

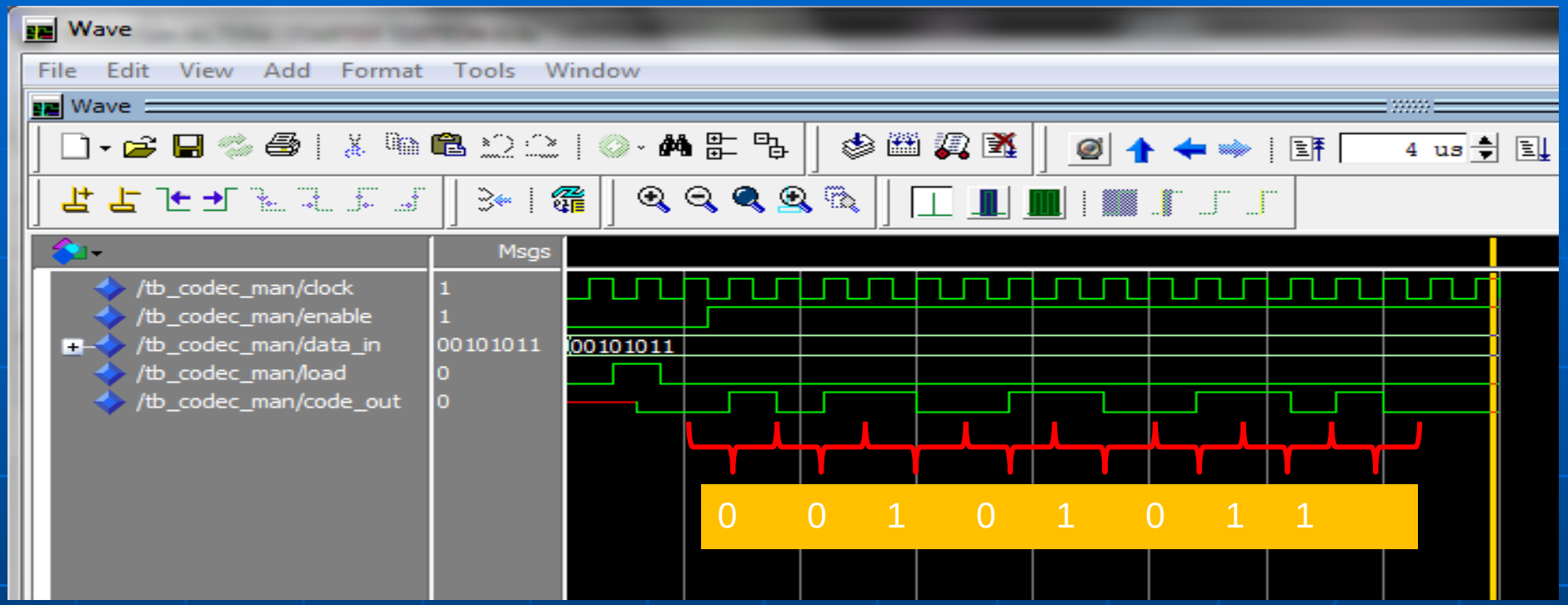
Flow Summary

Flow Status	Successful - Wed Jun 01 13:41:37 2016
Quartus II Version	10.1 Build 153 11/29/2010 SJ Web Edition
Revision Name	codec_man
Top-level Entity Name	codec_man
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	24 / 22,320 (< 1 %)
Total combinational functions	24 / 22,320 (< 1 %)
Dedicated logic registers	21 / 22,320 (< 1 %)
Total registers	21
Total pins	12 / 154 (8 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

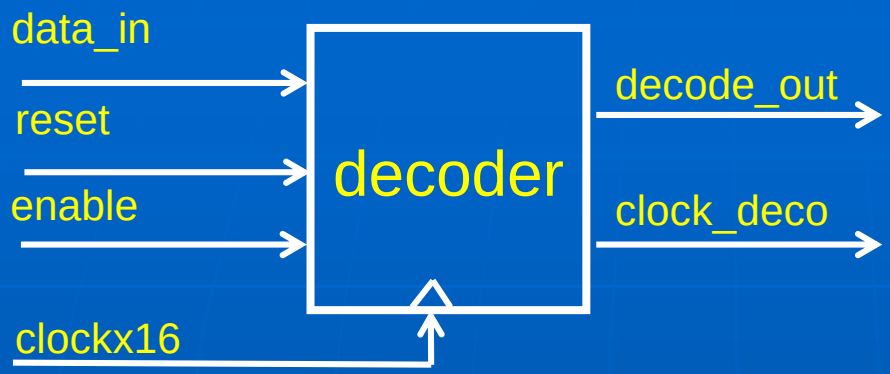
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity tb_codec_man is
7  end tb_codec_man;
8
9  architecture test of tb_codec_man is
10
11    component codec_man
12    Port (clock    : in std_logic;
13          enable   : in std_logic;
14          data_in  : in std_logic_vector (7 downto 0);
15          load     : in std_logic;
16          code_out : out std_logic );
17    end component;
18
19    signal clock      : std_logic;
20    signal enable     : std_logic;
21    signal data_in    : std_logic_vector (7 downto 0);
22    signal load       : std_logic;
23    signal code_out   : std_logic;
24
25 begin
26
27   uut: codec_man port map  (  clock => clock, enable => enable,
28                               data_in => data_in, load  => load,
29                               code_out => code_out );
30
31    gen_reloj : process
32       begin
33          clock <= '0';
34          wait for 100 ns;
35          clock <= '1';
36          wait for 100 ns;
37    end process gen_reloj;
38
39    estimulos : process
40    begin
41       enable <= '0';
42       load <= '0';
43       data_in <= "00101011";
44       wait for 200 ns;
45       load <= '1';
46       wait for 200 ns;
47       load <= '0';
48       wait for 200 ns;
49       enable <= '1';
50       wait for 3500 ns;
51   end process estimulos;
52 end test;
```

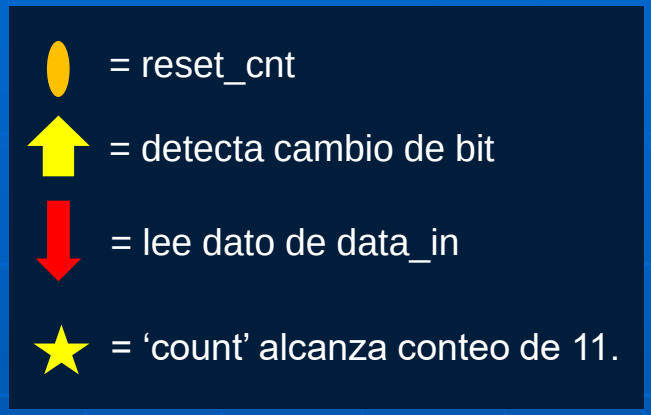
Diseño de un codificador manchester de 8 bits

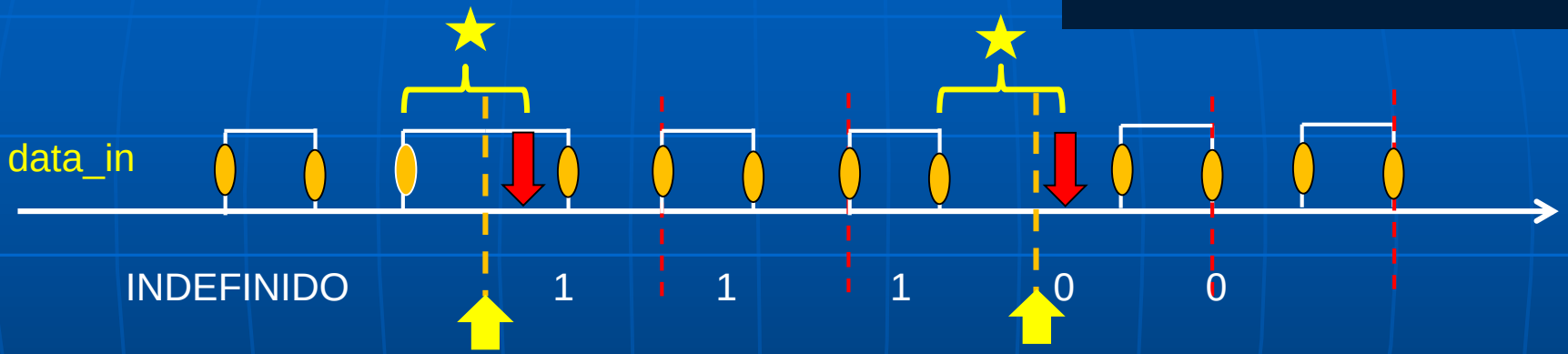
Resultado de la simulación



Diseño de un decodificador manchester





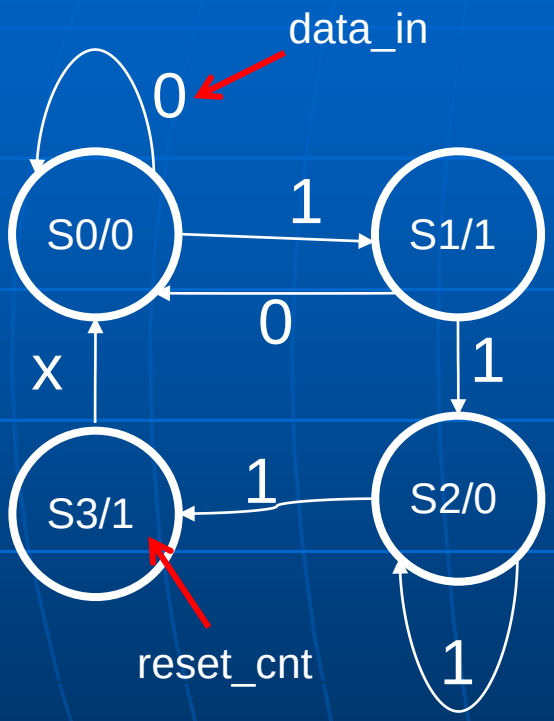


El dato del serializador se procesa a una frecuencia 16 veces mayor a la del reloj original del transmisor (TX). Esto garantiza que el receptor (RX) pueda recuperar el reloj aún existiendo falta de sincronismo entre TX y RX. Se comienza buscando una transición de datos 0 a 1 ó 1 a 0, donde el TX debe transmitir 0110 ó 1001 y en tal condición hay un lapso donde tenemos en dos tiempos de bit, un '1' ó un '0' respectivamente. Para ello se dispara constantemente un reloj que comienza en '0' al detectar una transición en la trama recibida. Ese contador trabaja con un reloj 16 veces más rápido que el del transmisor. Está preparado para que si cuenta hasta 12, significa que está en alguna de las dos condiciones anteriores donde se recibe 0110 ó 1001. En ese caso lee el dato de entrada para rescatar el dato originalmente enviado. En paralelo a esto, se reconstruye la señal de reloj a fin de mandar a la siguiente etapa del receptor, el dato y reloj originales antes de ser codificados.

Diseño de un decodificador manchester

Descripción en VHDL

Modelo de descripción por MOORE del sincronizador.



Esta etapa con máquina de estado detecta cada transición en la trama de entrada y genera un pulso cada vez que esto sucede.
Sirve de señal de reset para el proceso ‘counter’.

```
1  --Decodificador Manchester
2  --ISLD Sergio Noriega 2016
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use IEEE.NUMERIC_STD.all;
6  use ieee.std_logic_arith.all;
7  use ieee.std_logic_unsigned.all;
8
9  entity decoder_man1 is
10
11    Port ( clockx16     : in std_logic;
12           enable       : in std_logic;
13           data_in      : in std_logic;
14           reset        : in std_logic;
15           out_rst_cnt  : out std_logic;
16           out_count    : out std_logic_vector (4 downto 0);
17           out_count2   : out std_logic_vector (3 downto 0);
18           clock_deco   : out std_logic;
19           decoder_out  : out std_logic
20         );
21 end decoder_man1;
22
23 architecture Comportamiento of decoder_man1 is
24
25    signal count               : integer range 0 to 15;
26    signal count2              : std_logic_vector (3 downto 0):= "0000";
27    constant countmax          : integer :=11;
28    signal reset_cnt, out_dec, clock2, enable2   : std_logic := '0';
29    type state_type is (s0, s1, s2, s3); signal state   : state_type;
30
31    begin
32    gen1_rst_cnt : process (clockx16, reset)
33                   begin
34                      if reset = '1' then state <= s0;
35                      elsif (rising_edge(clockx16)) then
36                         case state is
37                            when s0=>
38                               if data_in = '1' then state <= s1;
39                               else state <= s0;
40                               end if;
41                            when s1=>
42                               if data_in = '1' then state <= s2;
43                               else state <= s0;
44                               end if;
45                            when s2=>
46                               if data_in = '1' then state <= s2;
47                               else state <= s3;
48                               end if;
49                            when s3 =>
50                               if data_in = '1' then state <= s0;
51                               else state <= s0;
52                               end if;
53                         end case;
54                      end if;
55                   end process;
```

Diseño de un decodificador manchester

Descripción en VHDL

```
56
57    gen2_rst_cnt: process (state)
58                   begin
59                     case state is
60                        when s0 => reset_cnt <= '0';
61                        when s1 => reset_cnt <= '1';
62                        when s2 => reset_cnt <= '0';
63                        when s3 => reset_cnt <= '1';
64                     end case;
65                  end process;
66
67           out_rst_cnt <= reset_cnt;
68
69    counter   : process (clockx16, reset_cnt)
70                  begin
71                   if reset_cnt = '1' then count <= 0;
72                     elsif clockx16'event and clockx16 = '1' then
73                      if enable = '1' then
74                        if count = countmax then out_dec <= data_in; count <= 0; enable2 <='1';
75                        else count <= count + 1;
76                      end if;
77                     end if;
78                   end if;
79                end process;
80
81    gen_clkr   : process(clockx16, count, enable2)
82                  begin
83                   if (reset = '1' or count = 11) then count2 <= "0000";
84                     elsif clockx16'event and clockx16 = '1' then
85                       if (count2 = "0000" and enable2 = '1') then  clock2 <= '1';
86                       else clock2 <= '0';
87                       end if;
88                       count2 <= count2 + "0001";
89                     end if;
90                end process;
91
92    clock_deco <= clock2;
93    decoder_out <= out_dec;
94    out_count <= CONV_STD_LOGIC_VECTOR(count, 5);
95    out_count2 <= count2;
96 end Comportamiento;
97
```

Etapa para lectura del dato desde la trama recibida.
Esto se hace sólo si el contador llega a contar hasta 11.

Etapa para reconstruir la señal de reloj necesaria para que la próxima etapa del receptor procese el dato original en sincronismo.

‘clock_deco’ y ‘decoder_out’ son las señales de salida para la etapa posterior del receptor que debe realizar el sincronismo de ‘frame’ y posterior des-serialización.

Diseño de un decodificador manchester

Test bench en VHDL

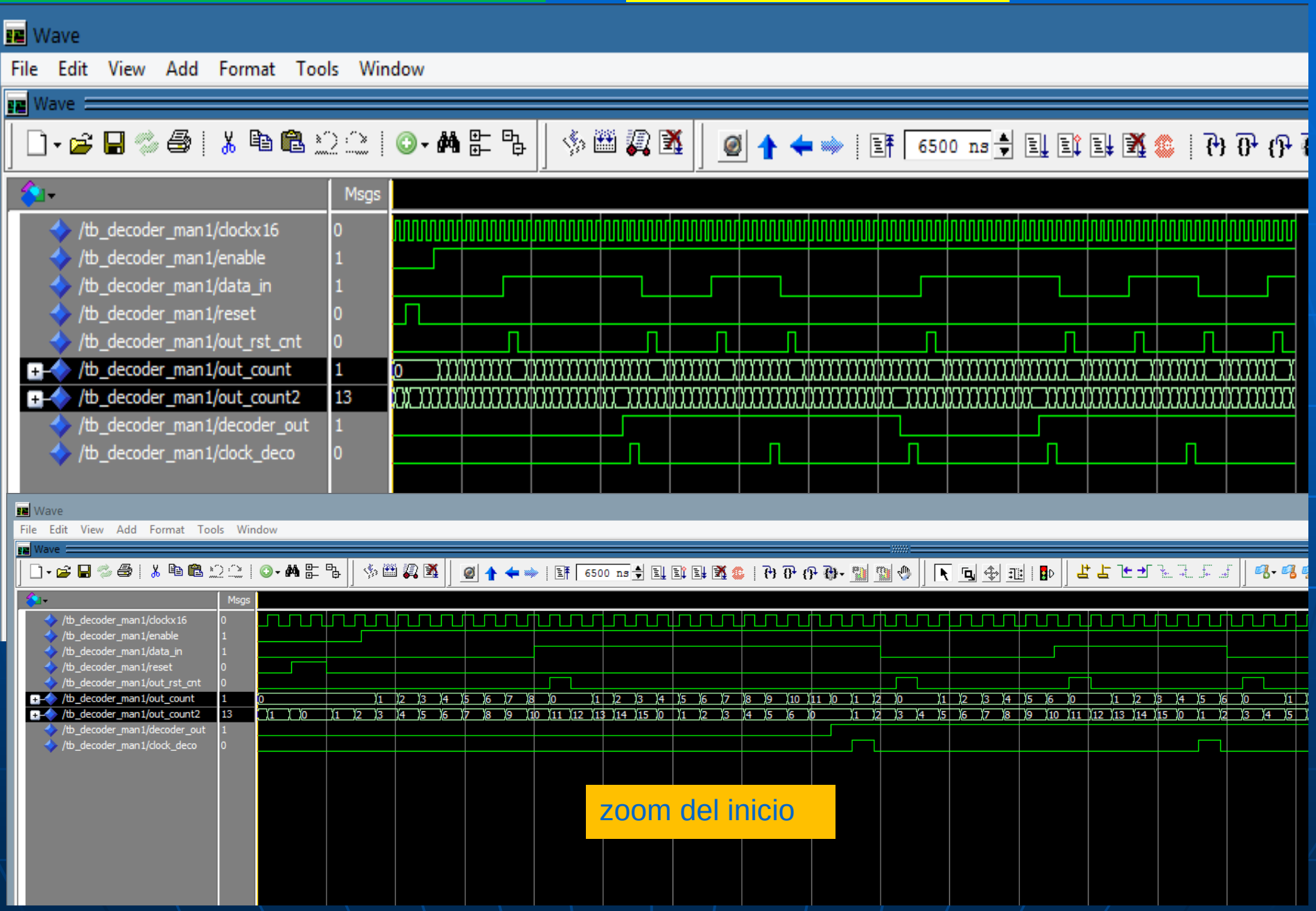
Se genera la secuencia: 10-10-01-10-10-.... que corresponde a un dato binario: 11011....

Se puede optimizar, usando un arreglo para 'data_in' y lazo de retardo con 'for-loop'.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity tb_decoder_man1 is
7  end tb_decoder_man1;
8
9  architecture test of tb_decoder_man1 is
10
11    component decoder_man1
12    Port (clockx16     : in std_logic;
13          enable       : in std_logic;
14          data_in      : in std_logic;
15          reset        : in std_logic;
16          out_rst_cnt  : out std_logic;
17          out_count    : out std_logic_vector (4 downto 0);
18          out_count2   : out std_logic_vector (3 downto 0);
19          clock_deco   : out std_logic;
20          decoder_out  : out std_logic );
21    end component;
22
23    signal clockx16, enable, data_in, reset, out_rst_cnt  : std_logic;
24    signal out_count    : std_logic_vector (4 downto 0);
25    signal out_count2   : std_logic_vector (3 downto 0);
26    signal decoder_out, clock_deco : std_logic;
27
28 begin
29   uut: decoder_man1 port map  (  clockx16 => clockx16, enable => enable,
30                                  data_in => data_in, reset => reset,
31                                  out_rst_cnt => out_rst_cnt,
32                                  out_count => out_count,
33                                  out_count2 => out_count2,
34                                  clock_deco => clock_deco,
35                                  decoder_out => decoder_out );
36    gen_reloj : process
37       begin
38          clockx16 <= '0'; wait for 31.25 ns;
39          clockx16 <= '1'; wait for 31.25 ns;
40    end process gen_reloj;
41
42    estimulos : process
43    begin
44       enable <= '0'; reset  <= '0'; data_in <= '0'; wait for 100 ns;
45       reset <= '1'; wait for 100 ns; reset <= '0'; wait for 100 ns;
46       enable <= '1'; wait for 500 ns; data_in <= '1'; wait for 500 ns;
47       data_in <= '1'; wait for 500 ns; data_in <= '0'; wait for 500 ns;
48       data_in <= '1'; wait for 500 ns; data_in <= '0'; wait for 500 ns;
49       data_in <= '0'; wait for 500 ns; data_in <= '1'; wait for 500 ns;
50       data_in <= '1'; wait for 500 ns; data_in <= '0'; wait for 500 ns;
51       data_in <= '1'; wait for 500 ns; data_in <= '0'; wait for 500 ns;
52       data_in <= '1'; wait for 500 ns; data_in <= '1'; wait for 500 ns;
53       data_in <= '1'; wait;
54    end process estimulos;
55 end test;
```

Diseño de un decodificador manchester

Resultado de la simulación



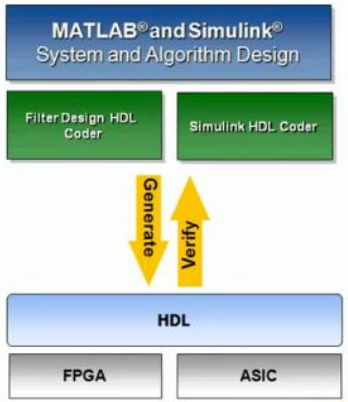
OTRAS HERRAMIENTAS DE DISEÑO PARA FPGAs

Ejemplo: Diseño y simulación en Matlab => Generación de código HDL (VHDL y Verilog).
=> Verificación de implementación de FPGA contra el modelo inicial.

The MathWorks MATLAB&SIMULINK

The System Design Solution

- Design, simulate, and validate system models and algorithms in MATLAB and Simulink
- Automatically generate HDL code for FPGAs and ASICs
- Verify the hardware implementation against the system model



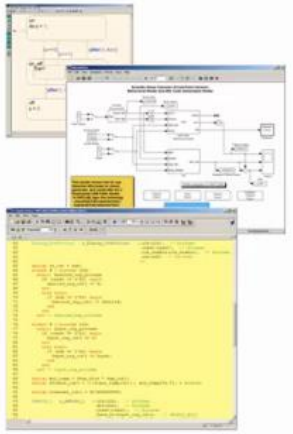
3

Feedback

The MathWorks MATLAB&SIMULINK

Simulink HDL Coder key features

- HDL code generation
  - IEEE 1376 compliant VHDL® code
  - IEEE 1364-2001 compliant Verilog® code
  - Area, Power, Speed choices for select blocks
- Verification
  - Generate HDL test-bench
  - Generate EDA Simulator Link co-simulation block
- Design automation
  - Synthesis and EDA simulator script generation via GUI
  - Control files for greater automation



13

Webseminar en website de Mathworks:
http://www.mathworks.com/videos/rapid-design-and-implementation-using-automatic-hdl-code-generation-82015.html

VERILOG

HISTORIA

VERILOG al igual que VHDL se ha convertido en la actualidad en un standard para el diseño y verificación de sistemas digitales como lenguaje de descripción de hardware (HDL) de alto nivel de abstracción.
En 1995 fue incorporado al IEEE (std. 1364-1995) como una herramienta de descripción de hardware y en forma paulatina fue siendo aceptada por los grandes fabricantes circuitos digitales programables.
Debido a varios problemas, en 2001 apareció una importante actualización a fin de subsanarlos, aunque todavía para realizar descripción y verificación de un diseño, se tenía que emplear un software adicional aparte de VERILOG.
Finalmente esta falencia se subsanó y en el 2005 se formalizó la nueva evolución de VERILOG, la cual se estandarizó en IEEE denominándose SYSTEM VERILOG (std. 1800-2005).
Este nuevo tipo de software HDL permitió entre otras cosas, como se comentó, poder realizar las verificaciones de sistemas en el mismo paquete del software como VHDL.

VERILOG

EJEMPLO CON SYSTEM VERILOG

DESCRIPCIÓN DE UN MUX EMPLEANDO SENTENCIA «IF»

```
7 module mux_usando_if(
8 input wire din_0 , // Primera entrada del Mux
9 input wire din_1 , // Segunda entrada del Mux
10 input wire sel , // Entrada de Selección del MUX
11 output reg mux_out // Salida del Mux
12 );
13 //-------------Comienza la Descripción del Código---------
14 always_comb
15 begin : MUX
16 if (sel == 1'b0) begin
17 mux_out = din_0;
18 end else begin
19 mux_out = din_1 ;
20 end
21 end
22
23
endmodule
```

http://www.asic-world.com/systemverilog/.....

VERILOG

EJEMPLO CON SYSTEM VERILOG

DESCRIPCIÓN DE UN FF «D» CON RESET SINCRÓNICO

```
7 module dff_sync_reset (
8 input wire data , // Entrada del Dato del FF
9 input wire clk , // Entrada del Clock del FF
10 input wire reset , // Entrada del Reset del FF
11 output reg q // Salida Q del FF
12 );
13 //------------- Comienza la Descripción del Código---------
14 always_ff @ ( posedge clk)
15 if (~reset) begin
16 q <= 1'b0;
17 end else begin
18 q <= data;
19 end
20
21 endmodule
```

http://www.asic-world.com/systemverilog/.....

VERILOG

EJEMPLO CON SYSTEM VERILOG

DESCRIPCIÓN DE UN CONTADOR PROGRESIVO CON CARGA PARALELA

```
7 module up_counter_load (
8 output reg [7:0] out , // Salida de counter
9 input wire [7:0] data , // Entrada Paralela de counter
10 input wire load , // Habilitación de Carga Paralela de  counter
11 input wire enable , // Habilitación de conteo de counter
12 input wire clk , // Entrada de clock de counter
13 input wire reset // Entrada de reset de counter
14 );
15 //------------- Comienza la Descripción del Código------
16 always_ff @ (posedge clk)
17 if (reset) begin
18 out <= 8'b0 ;
19 end else if (load) begin
20 out <= data;
21 end else if (enable) begin
22 out <= ++;
23 end
24
25 endmodule
```

http://www.asic-world.com/systemverilog/.....

BIBLIOGRAFÍA:

Libros (lista parcial):

-- VHDL for Logic Synthesis. A. Rushton. John Wiley 3rd Edition, 2011.

-- Circuit Design with VHDL. Volnei A. Pedroni. MIT Press, 2004.

-- VHDL: Programming by Example. Douglas Perry. McGraw-Hill, 2002.

-- VHDL: Lenguaje para Síntesis y modelado de circuitos. Pardo/Boluda. Alfaomega, 2000.

Internet (lista parcial):

-- The VHDL Cookbook. Peter Ashenden. Peter Ashenden 1990.
tech-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf

-- Hamburg VHDL Archives: http://tams-www.informatik.uni-hamburg.de/research/vlsi/vhdl/

-- EDA (Electronic Design Automation) Industry Working Groups:
http://www.vhdl.org/

NOTA: La disponibilidad de material sobre este tema es muy grande.
Aquí sólo se dan algunas referencias como para empezar a interesarse en el tema.