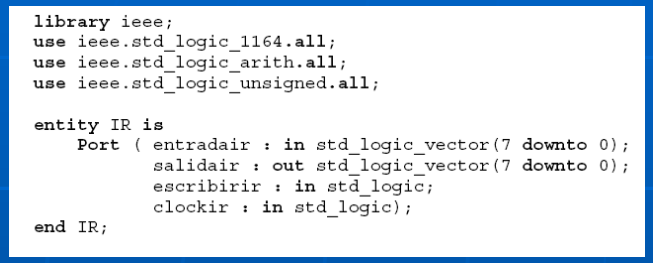
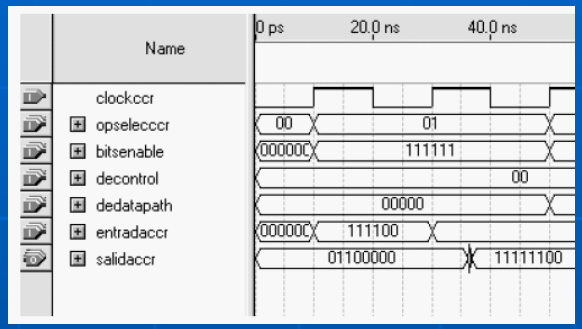




Introducción al diseño lógico con VHDL

Sergio Noriega 2015

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 fue 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 fue 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 y FPGA.

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.).

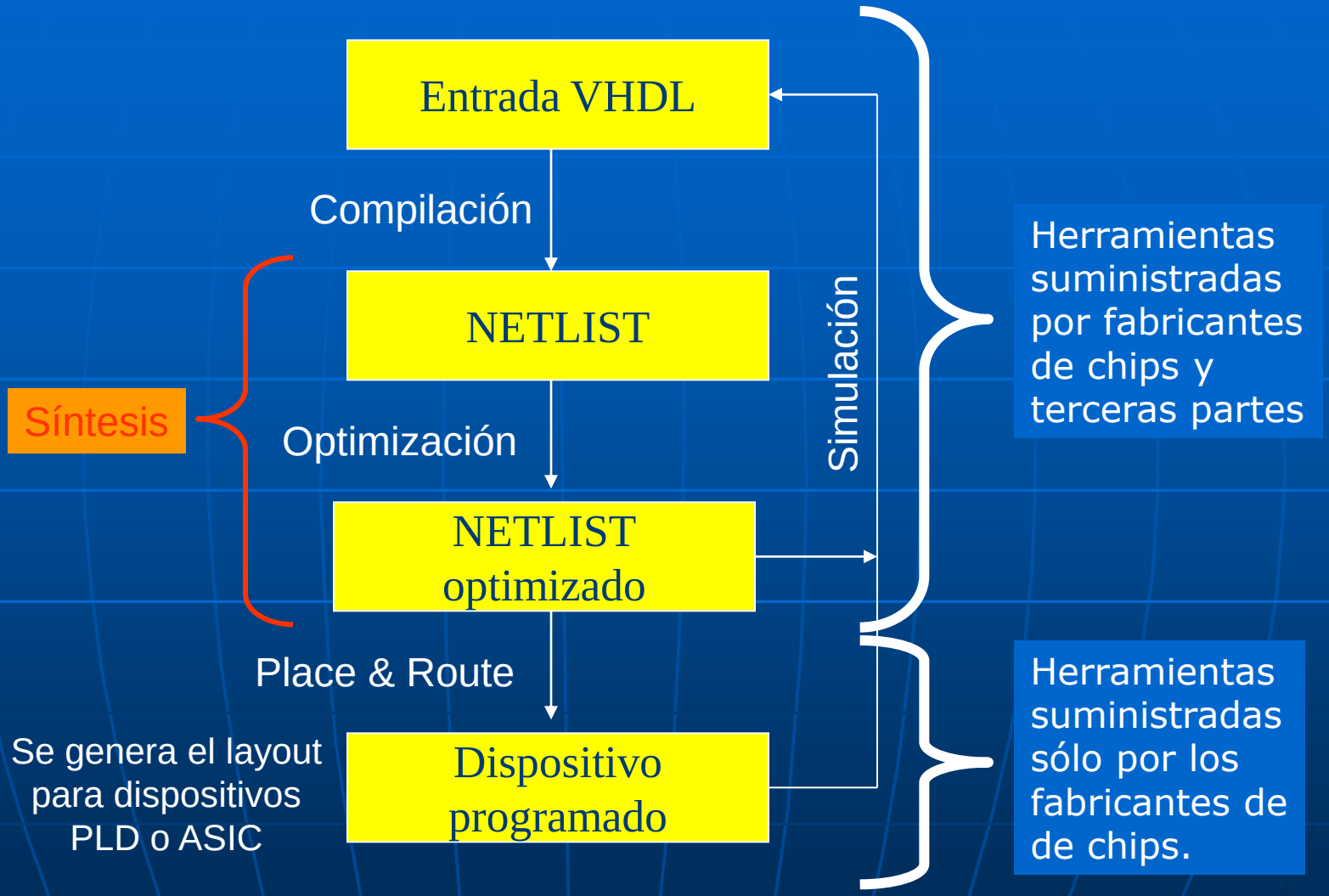
Desde su implementación en el año 1981, VHDL fue estandarizado por primera vez por la IEEE en 1987 (std. 1076) y se realizó un importante actualización en 1993 (con posteriores revisiones en 1994, 2000, 2002 y 2007).

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 , Altera, 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 ó MaxPlus II de Altera (aquí se usará el segundo),
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.

Estructuras en VHDL:

Entity: Define la vista externa de un modelo.

Architecture: Define una posible funcionalidad de un modelo.

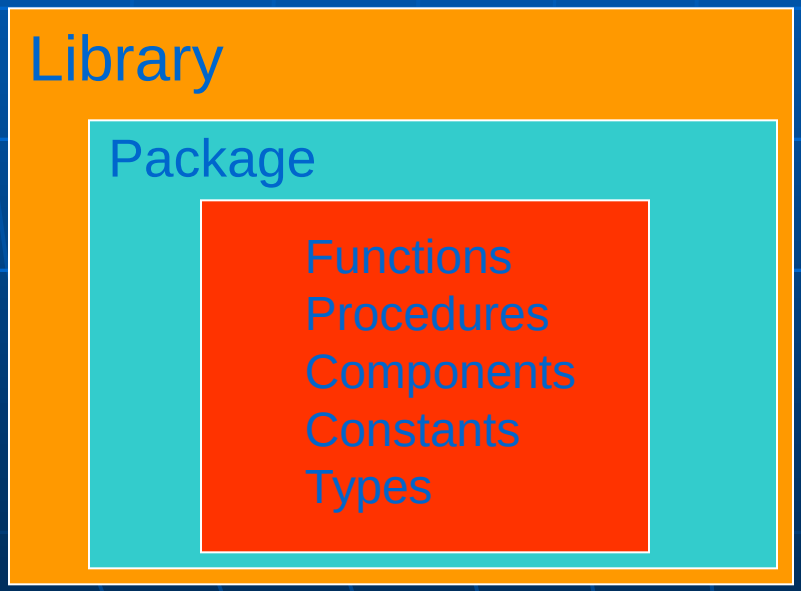
Library: Contiene un listado de todas las librerías utilizadas en el diseño.

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

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;
```

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.

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).

Arquitectura (Architecture)

La arquitectura en VHDL describe la funcionalidad de la entidad y contiene la información que modela el comportamiento de la misma.

Ejemplo (asociado al ejemplo anterior):

```
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;
```

Aquí especificamos QUE HACE el hardware existiendo diferentes maneras para hacerlo

IMPORTANTE:
EN ESTE EJEMPLO EN PARTICULAR, ESTA MANERA DE DESCRIPCIÓN TEMPORAL SE USA FUNDAMENTALMENTE EN SIMULACIÖN Y PUEDE NO SER SINTETIZABLE POR HARDWARE.

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.

SIGNAL:

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.

En el ejemplo anterior hay sólo una señal “select” declarada en una arquitectura denominada “dataflow”.

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.

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.

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 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”;

```
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;
```

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.

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 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;
```

DISEÑO POR FLUJO DE DATOS

Ejemplo de buffer tri_state octuple

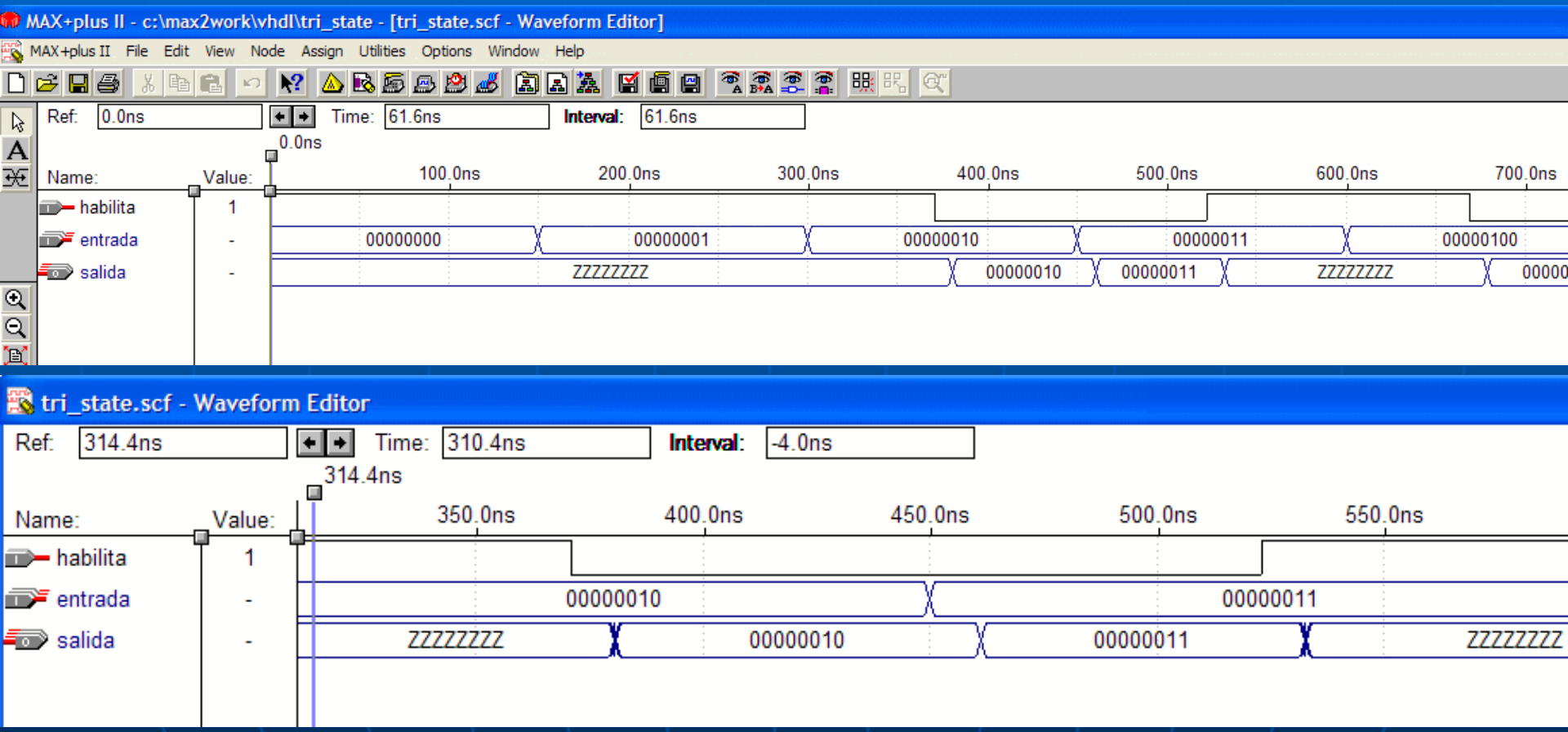
tri_state.vhd - Text Editor

```
-- 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;
```

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

```
ENTITY sumador2 IS PORT (
   a,b : IN BIT_VECTOR (4 DOWNTO 1);
   c_out : OUT BIT; sum : OUT BIT_VECTOR(4 DOWNTO 1));
END ENTITY sumador2;

ARCHITECTURE sum_con_gen OF sumador2 IS
  SIGNAL c: BIT_VECTOR (5 DOWNTO 1);
BEGIN
  c(1) <= `0´;
  adders: FOR i IN 1 TO 4 GENERATE
     sum(i) <= a(i) XOR b(i) XOR  c(i);
     c(i+1) <= (a(i) AND b(i));
                  OR (a(i) AND b(i))
               OR (b(i) AND c(i));
    END GENERATE;
    cout <= c(5);
END ARCHITECTURE sum_con_gen;
```

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

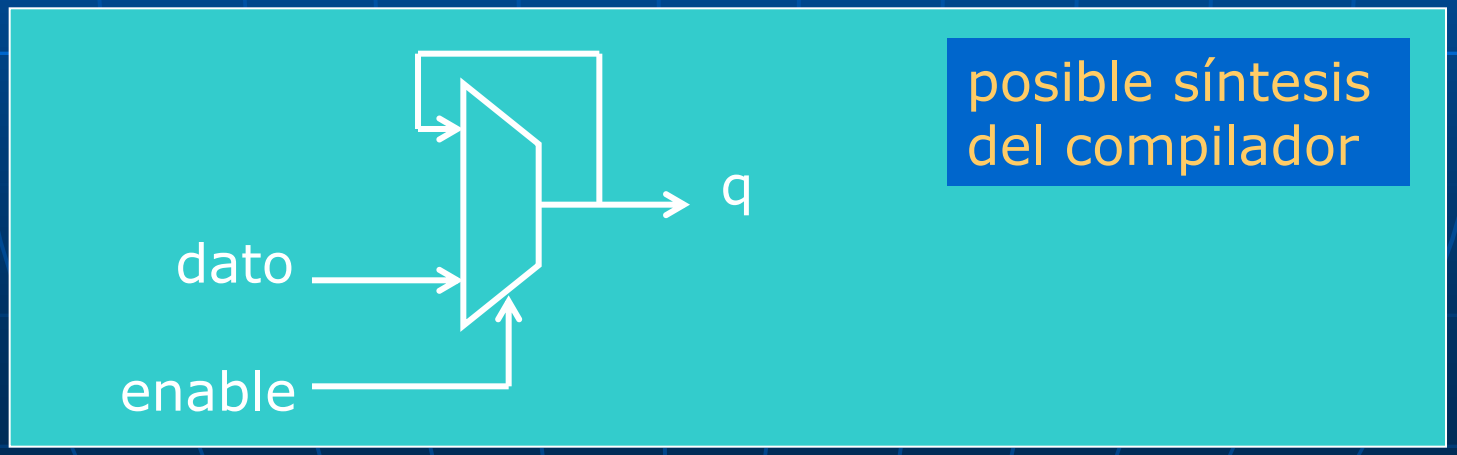
Diseño de Compuertas

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 tipo "D" con RESET asincrónico y habilitación de RELOJ

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;
```

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.

Diseño de Flip-Flops

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;
```

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

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 II

Registro de 8 bits con RESET sincrónico

registro8bits.vhd - Text Editor

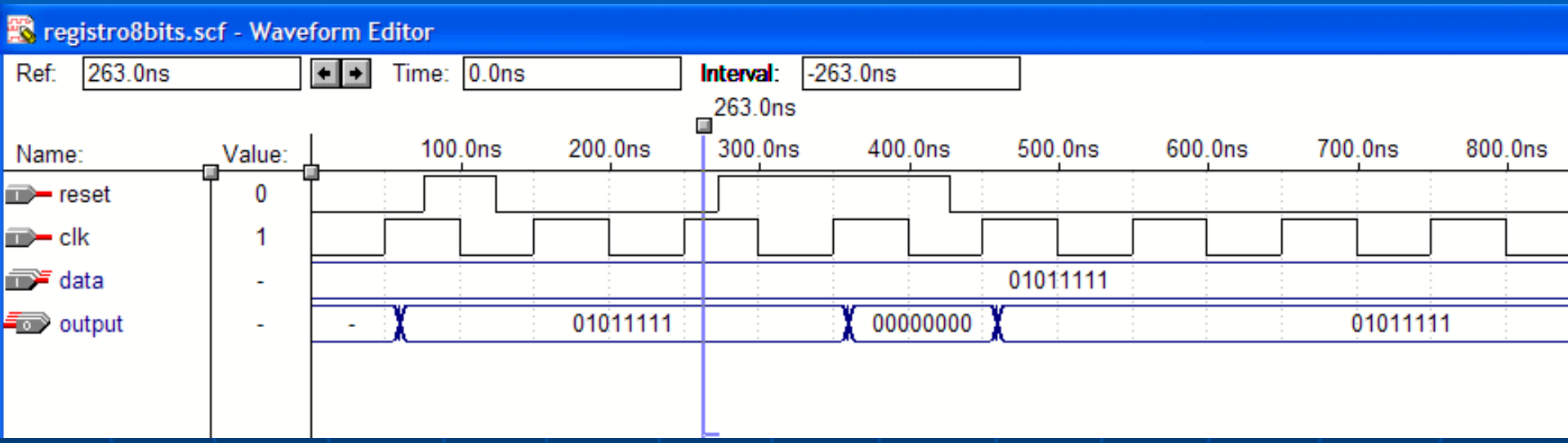
```
library IEEE;
use IEEE.std_logic_1164.all;

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 II

Registro de 8 bits con RESET sincrónico

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



Diseño de Contadores

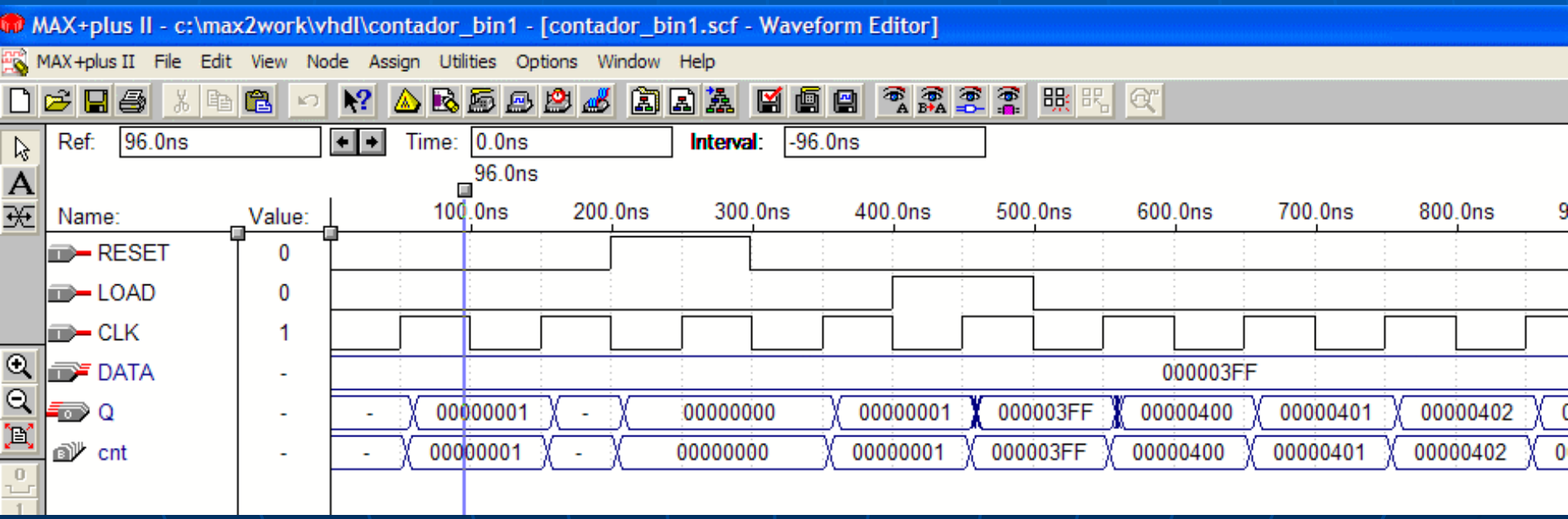
Contador binario progresivo de 32 bits con precarga y reset asincrónico

```
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 con el “waveform editor” del MAX+plus II

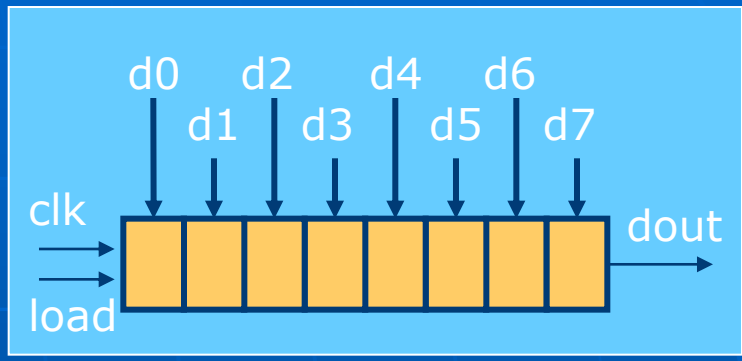


Diseño de Registro de Desplazamiento

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;
```



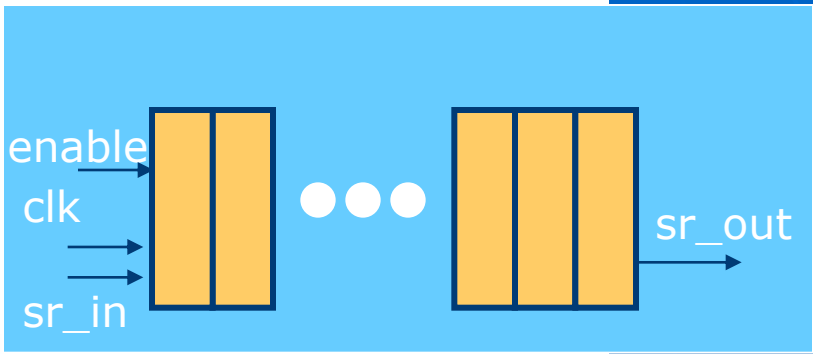
La carga del registro de desplazamiento es sincrónica ya que está dentro del proceso de control del flanco del reloj.

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 serie-serie.

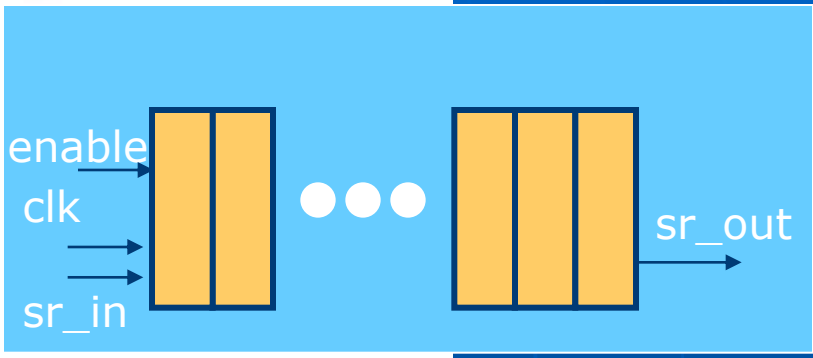
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity shift_1x64 is
5       port
6       (
7           sr_in : in std_logic;
8           enable   : in std_logic;
9           clk      : in std_logic;
10          sr_out   : out std_logic
11      );
12
13  end entity;
14
```



Diseño de Registro de Desplazamiento

RD serie-serie.

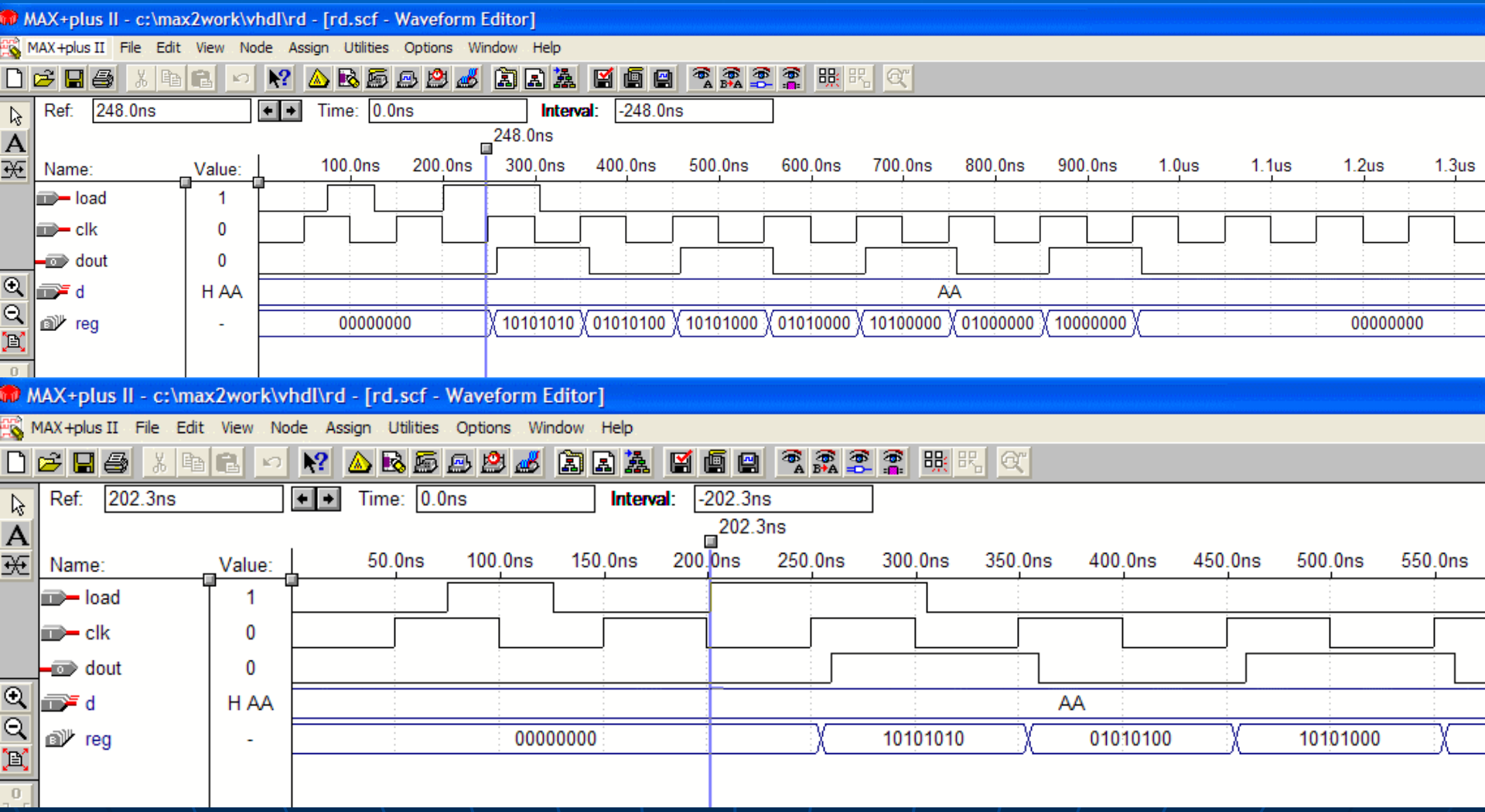
```
15  architecture rtl of shift_1x64 is
16
17      -- Build an array type for the shift register
18      type sr_length is array (63 downto 0) of std_logic;
19
20      -- Declare the shift register signal
21      signal sr: sr_length;
22
23  begin
24
25      process (clk)
26      begin
27          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 Registro de Desplazamiento

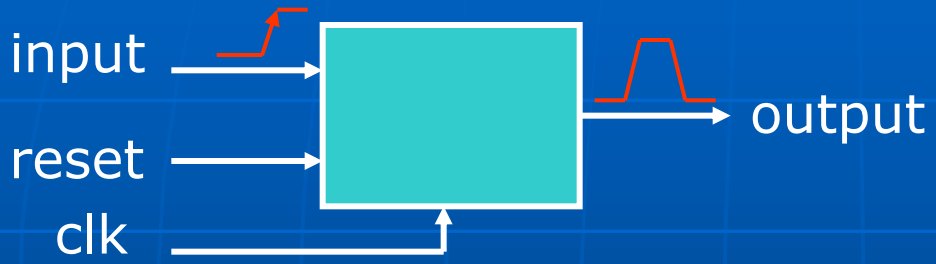
RD paralelo-serie con carga sinc.

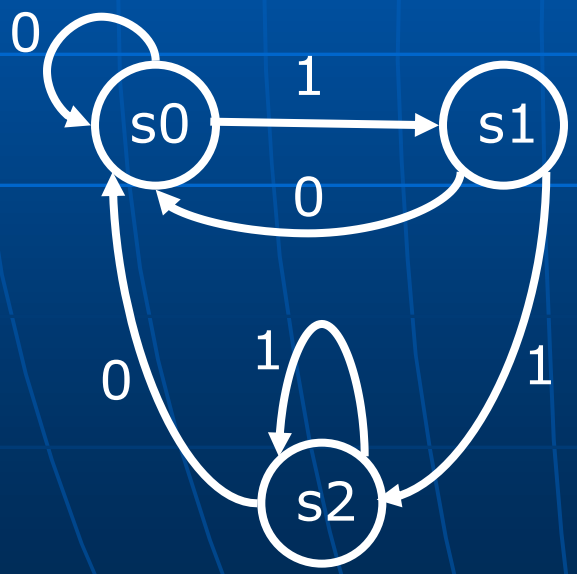
Simulación con el "waveform editor" del MAX+plus II



Diseño de Máquinas de Estado

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



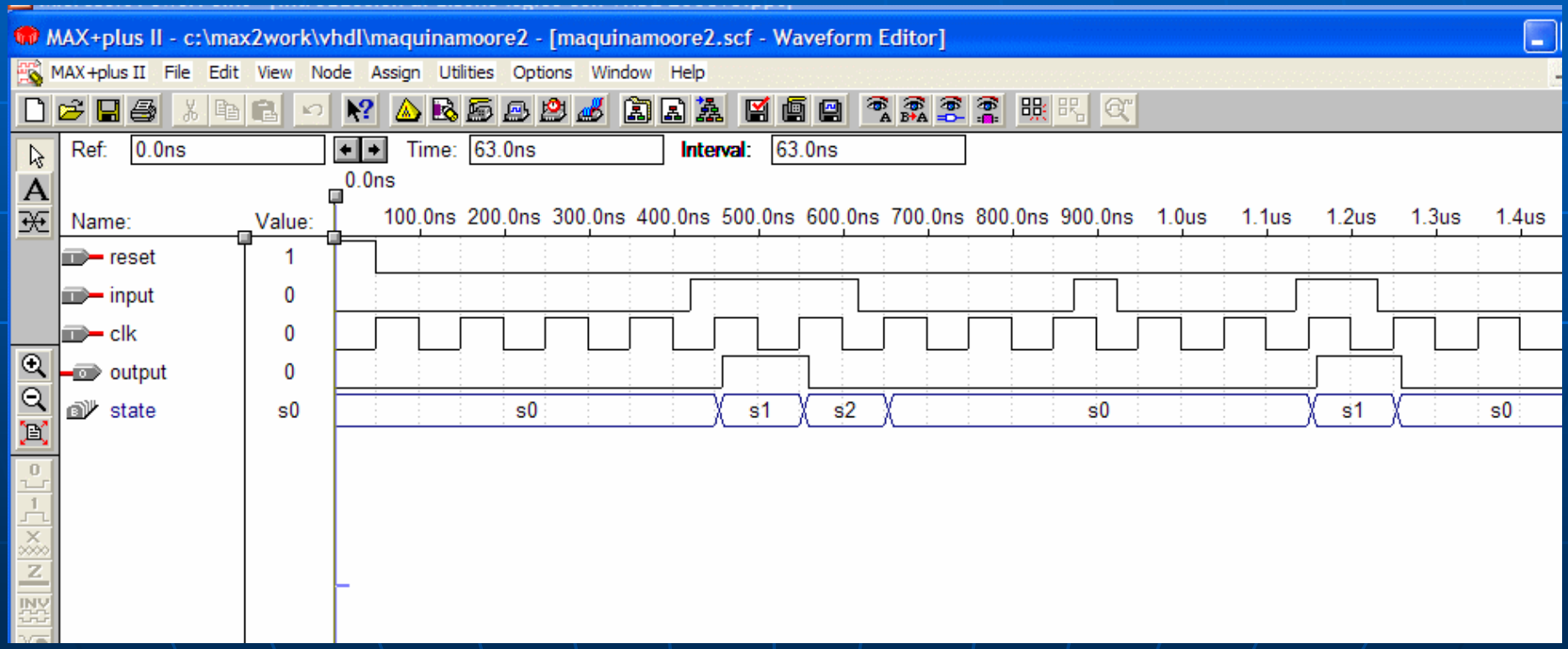



```
ENTITY maquinamoore2 IS
    PORT(
        clk     : IN    BIT;
        input   : IN    BIT;
        reset   : IN    BIT;
        output  : OUT   BIT);
END maquinamoore2;

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;
```

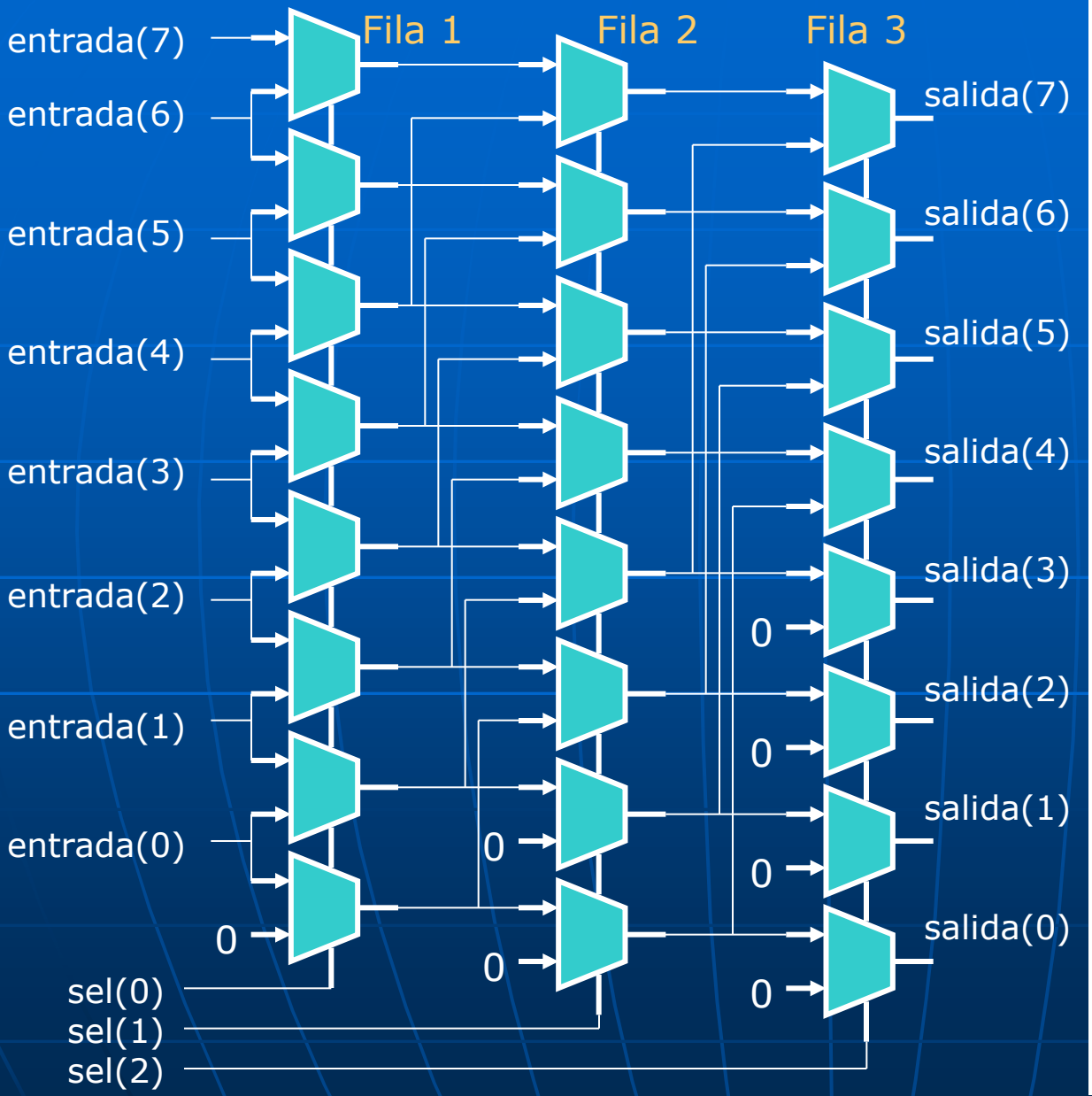
Diseño de Máquinas de Estado

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

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



Diseño de Barrel Shifters

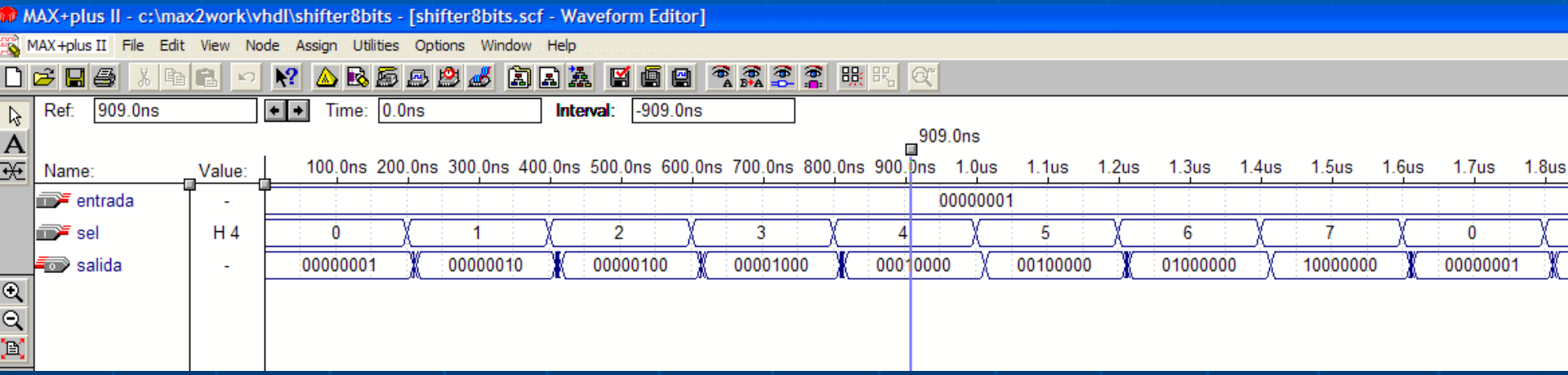



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

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

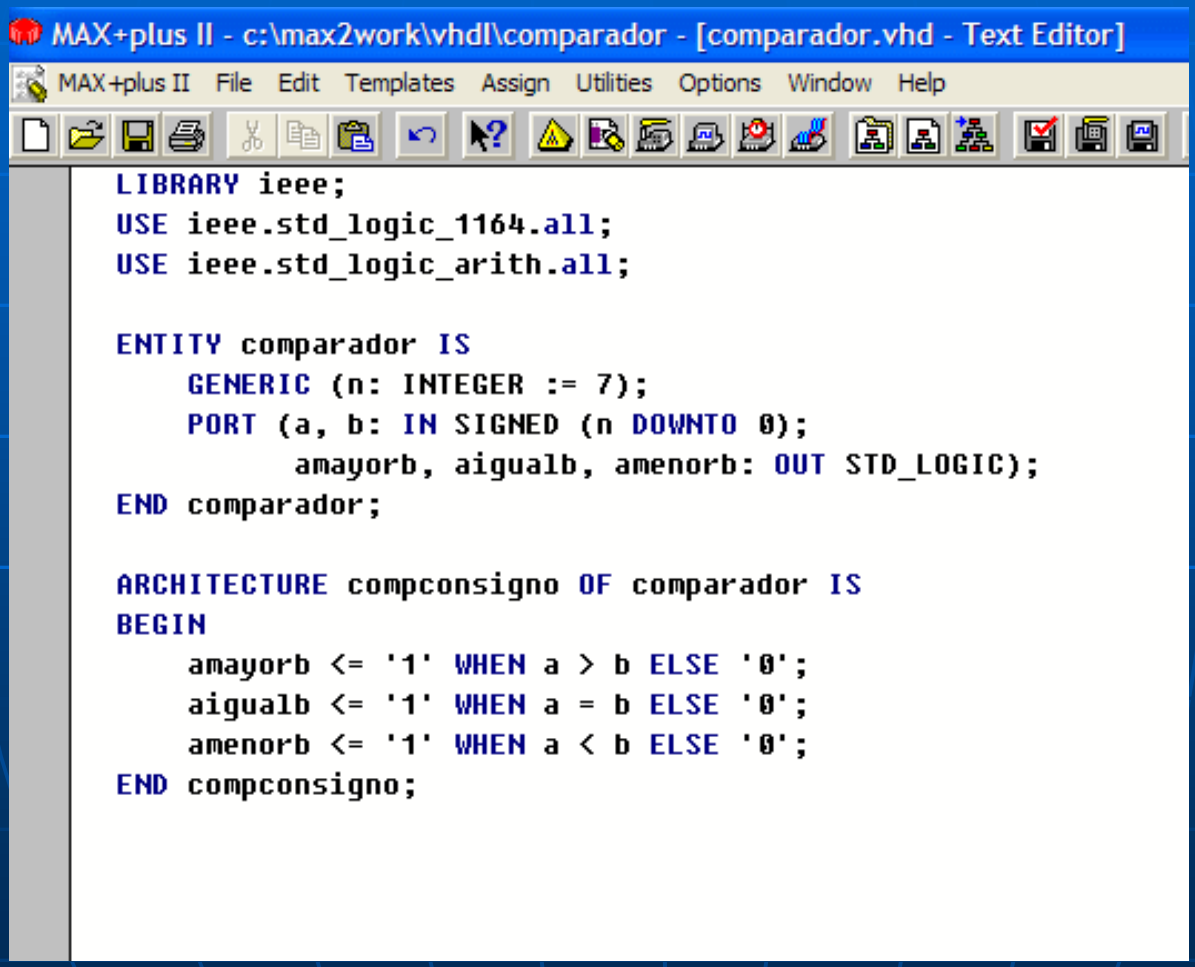
Ejemplo: Diseño de Shifter aritmético de 8 bits a izquierda.

Simulación con el “waveform editor” del MAX+plus II



Diseño de comparadores

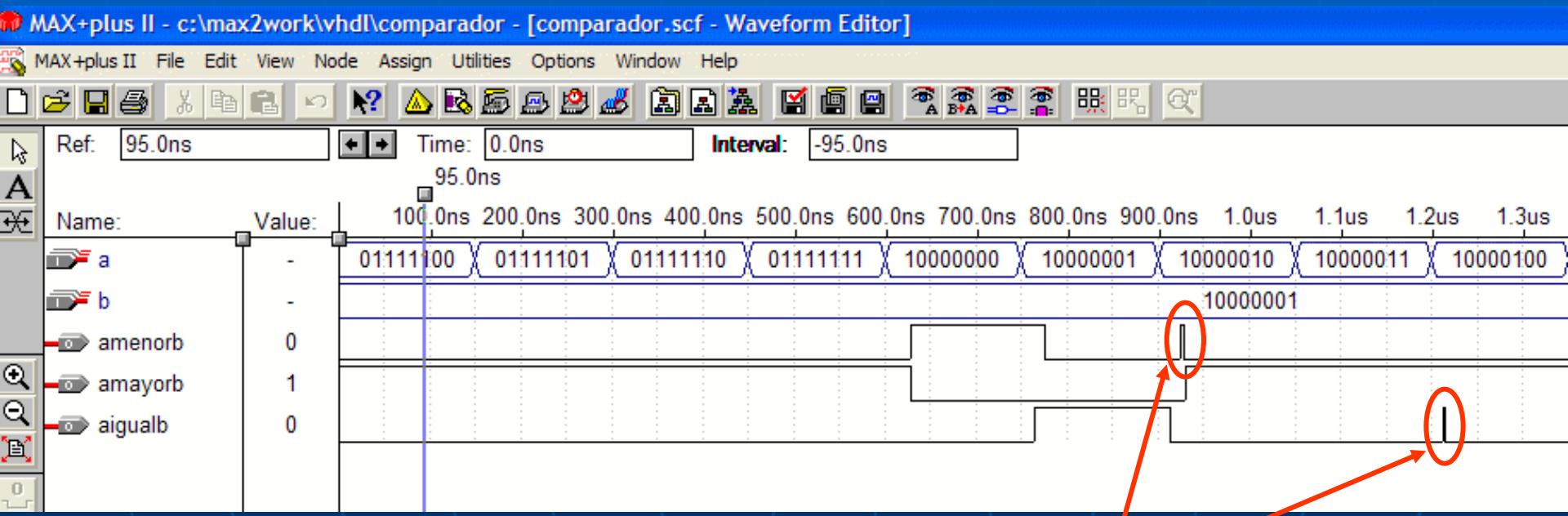
Ejemplo: Diseño de comparador binario con signo en punto fijo de dos números “a” y “b” de 8 bits de longitud de palabra



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 con el “waveform editor” del MAX+plus II



Porqué los glitches...???

Cómo se soluciona...???

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
```

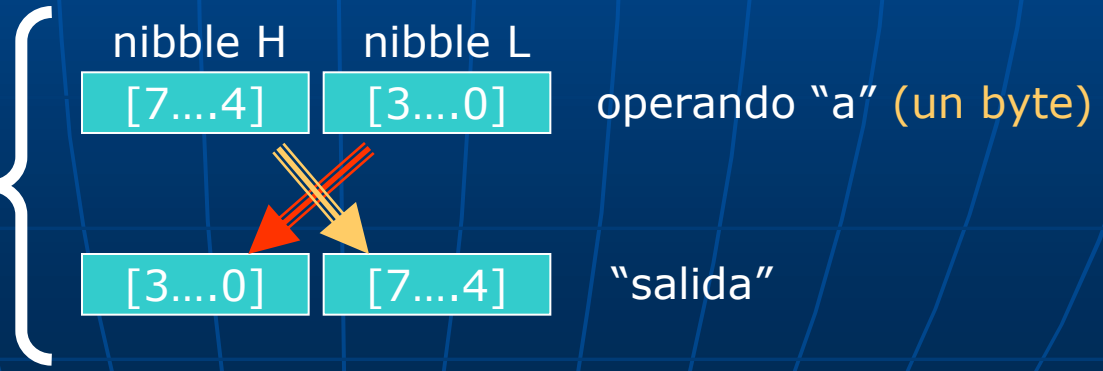
sentencias para implementar la operación “swap” del operando “a”

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;
```

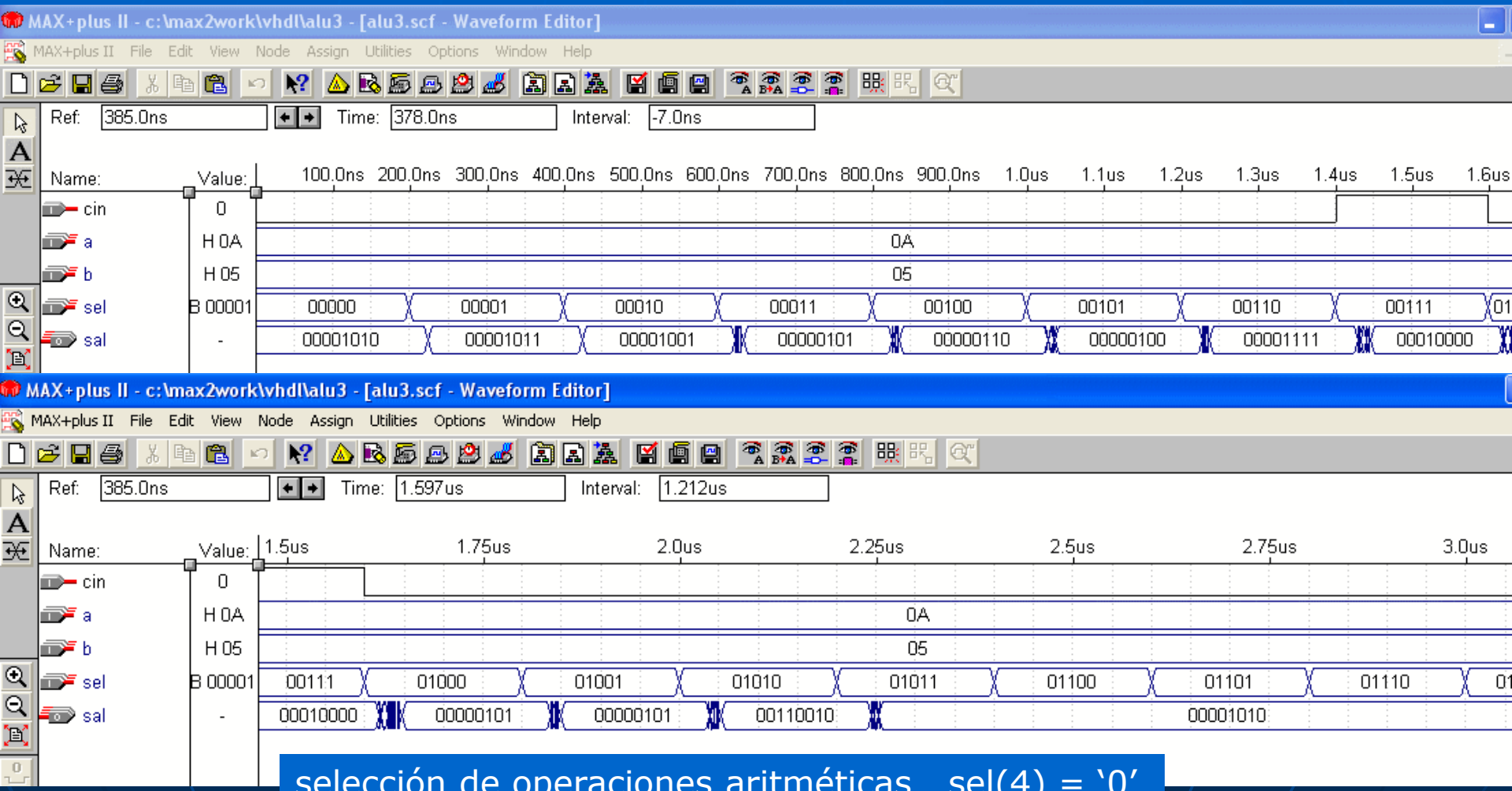
Ejemplo de operación swap para el operando "a" ("sel" = "11000")



Diseño de Unidad Aritmético-Lógica (ALU)

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

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

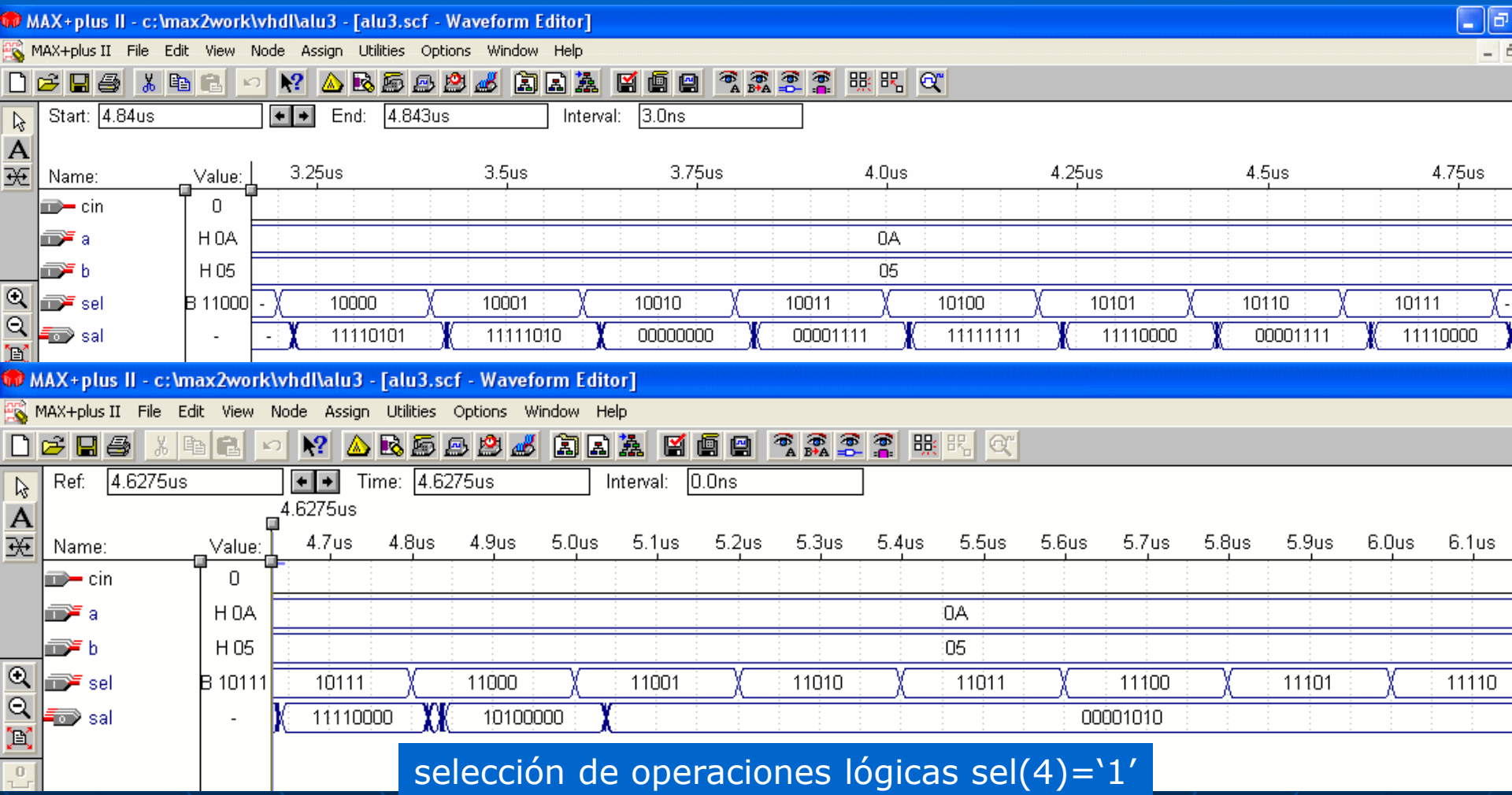


selección de operaciones aritméticas sel(4) = '0'

Diseño de Unidad Aritmético-Lógica (ALU)

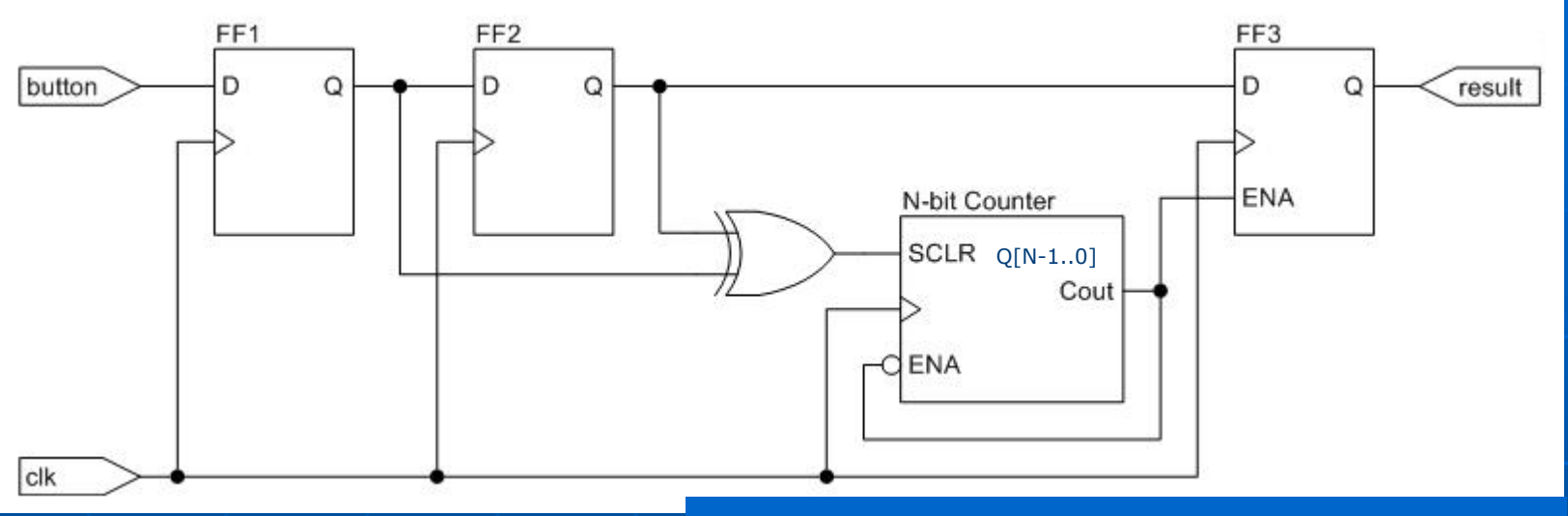
Diseño de ALU de 8 bits con operaciones aritméticas sin signo (continuación)

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



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

Ejemplo de diseño de circuito anti-rebote digital



Ejemplo:
clk = 50 MHz.
contador de 20 bits
Cout = Q19.

clk: reloj de circuito anti-rebote.
button: entrada desde pulsador ó llave.
result: salida procesada.
Cout: salida del bit MSB de Counter.

"result" sólo copia a Q(FF2) cuando lo habilita Cout con "1" y eso ocurre sólo si Counter llega hasta "1000....00000" (tiempo total de 1/50MHz x 2^{19} = 10,48 ms). Toda vez que difieran los valores de Q(FF1) y Q(FF2), el contador es borrado en forma sincrónica porque la OR-EXCL. se pone en "1" y Cout inhibe al FF3.

Ejemplo de diseño de circuito anti-rebote digital

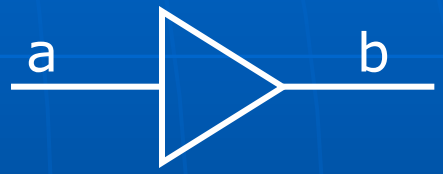
```
--------------------------------------------------------------------------------
--
--   FileName:         debounce.vhd
--   Dependencies:     none
--   Design Software:  Quartus II 32-bit Version 11.1 Build 173 SJ Full Version
--
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--   Version History
--   Version 1.0 3/26/2012 Scott Larson
--     Initial Public Release
--
--------------------------------------------------------------------------------
```

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

ENTITY debounce IS
  GENERIC(
    counter_size  :  INTEGER := 19); --counter size (19 bits gives 10.5ms with 50MHz clock)
  PORT(
    clk     : IN  STD_LOGIC;  --input clock
    button  : IN  STD_LOGIC;  --input signal to be debounced
    result  : OUT STD_LOGIC); --debounced signal
END debounce;

ARCHITECTURE logic OF debounce IS
  SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
  SIGNAL counter_set : STD_LOGIC;                    --sync reset to zero
  SIGNAL counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (OTHERS => '0'); --counter output
BEGIN

  counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/reset counter

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
      flipflops(0) <= button;
      flipflops(1) <= flipflops(0);
      If(counter_set = '1') THEN                  --reset counter because input is changing
        counter_out <= (OTHERS => '0');
      ELSIF(counter_out(counter_size) = '0') THEN --stable input time is not yet met
        counter_out <= counter_out + 1;
      ELSE                                        --stable input time is met
        result <= flipflops(1);
      END IF;
    END IF;
  END PROCESS;
END logic;
```

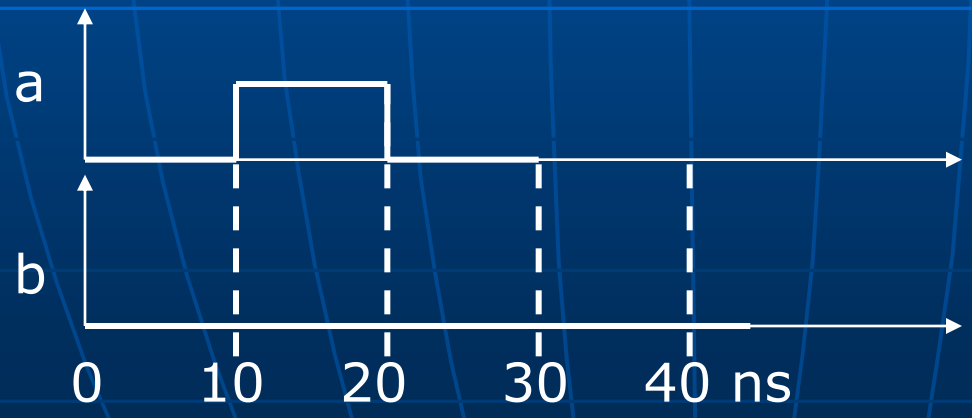
MODELADO POR COMPORTAMIENTO

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;
```



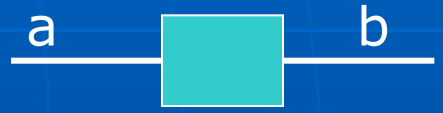
delay = 20 ns



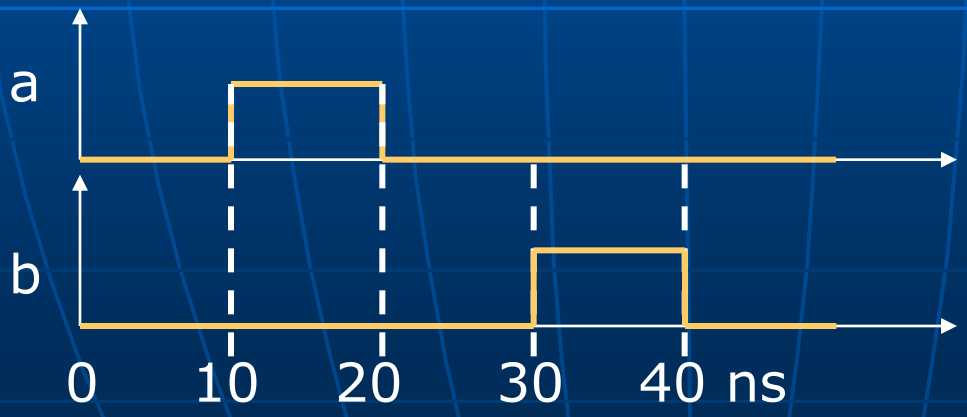
MODELADO POR COMPORTAMIENTO

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)

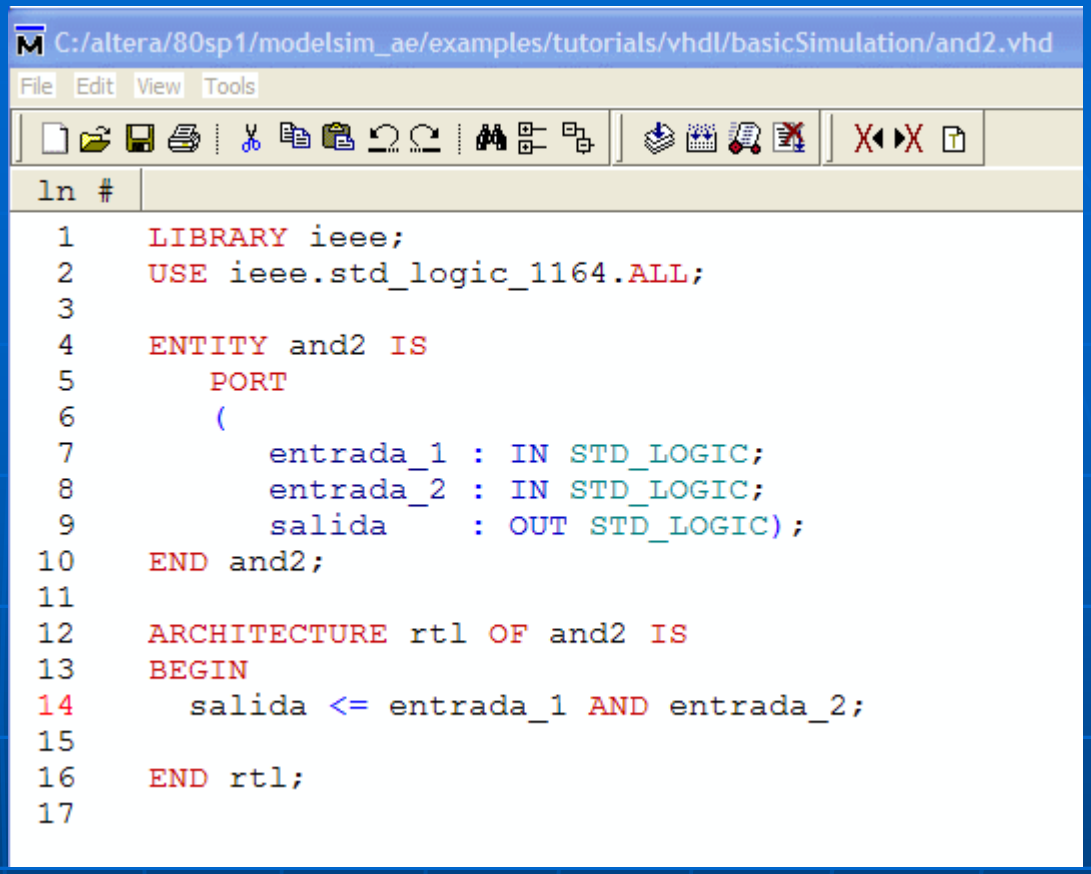
--Post-síntesis

--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 (and2.vhd):
En este caso una compuerta AND de 2 entrads.



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: (test_and2.vhd).

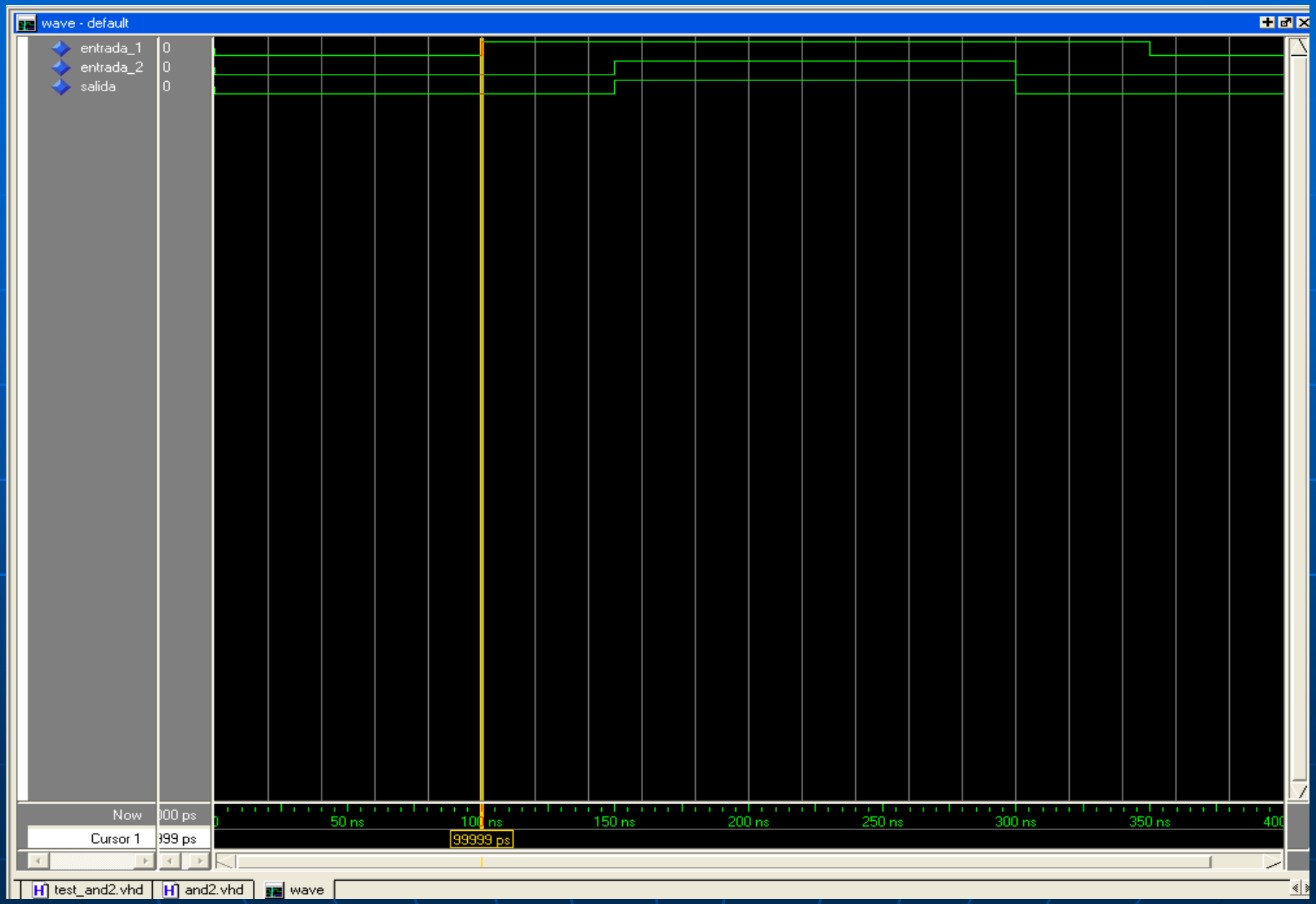
```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3
4   ENTITY test_and2 IS
5   END test_and2;
6
7   ARCHITECTURE test OF test_and2 IS
8
9   COMPONENT and2
10     PORT
11     (
12         entrada_1 : IN STD_LOGIC;
13         entrada_2 : IN STD_LOGIC;
14         salida    : OUT STD_LOGIC);
15  END COMPONENT;
16
17  SIGNAL entrada_1 : STD_LOGIC;
18  SIGNAL entrada_2 : STD_LOGIC;
19  SIGNAL salida  : STD_LOGIC;
20
21  BEGIN
22
23  dut : and2
24     PORT MAP (
25        entrada_1 => entrada_1,
26        entrada_2 => entrada_2,
27        salida => salida   );
28
29  stimulus : PROCESS
30
31     BEGIN
32     entrada_1 <= '0';
33     entrada_2 <= '0';
34     wait for 100 ns;
35     entrada_1 <= '1';
36     wait for 50 ns;
37     entrada_2 <= '1';
38     wait for 150 ns;
39     entrada_2 <= '0';
40     wait for 50 ns;
41     entrada_1 <= '0';
42  END PROCESS stimulus;
43
44  END test;
```

Instanciación de la entidad de la and ahora como un componente dentro de la entidad llamada “test_and2”

Sección para la generación de las señales de estímulo.

TEST BENCH EN VHDL

Ejemplo de simulación con ModelSim



BIBLIOGRAFÍA:

Libros (lista parcial):

-- 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.
Fernando Pardo – José Boluda. Alfaomega Grupo Editor, 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.