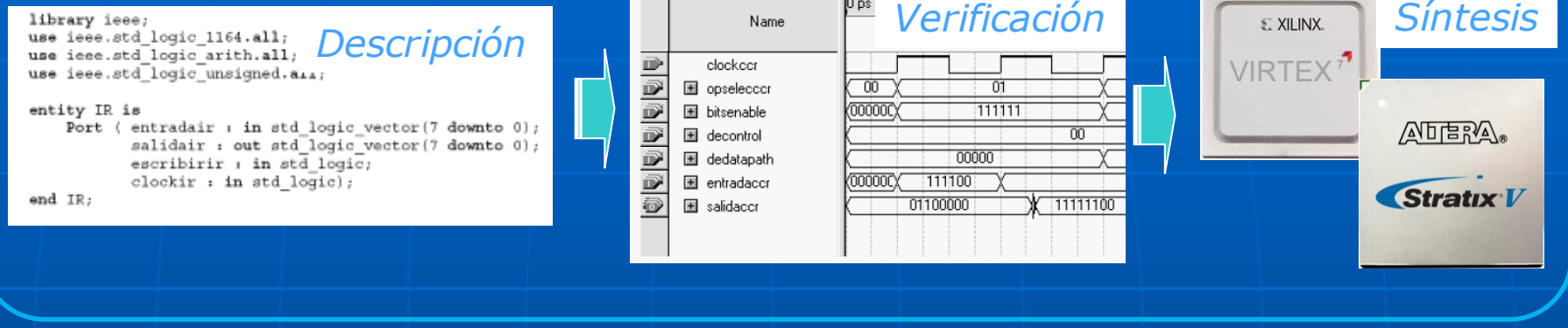



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity IR is
    Port ( entradair : in std_logic_vector(7 downto 0);
           salidair : out std_logic_vector(7 downto 0);
           escribirir : in std_logic;
           clockir : in std_logic);
end IR;
```

Introducción al diseño lógico con VHDL - Parte 1

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.).

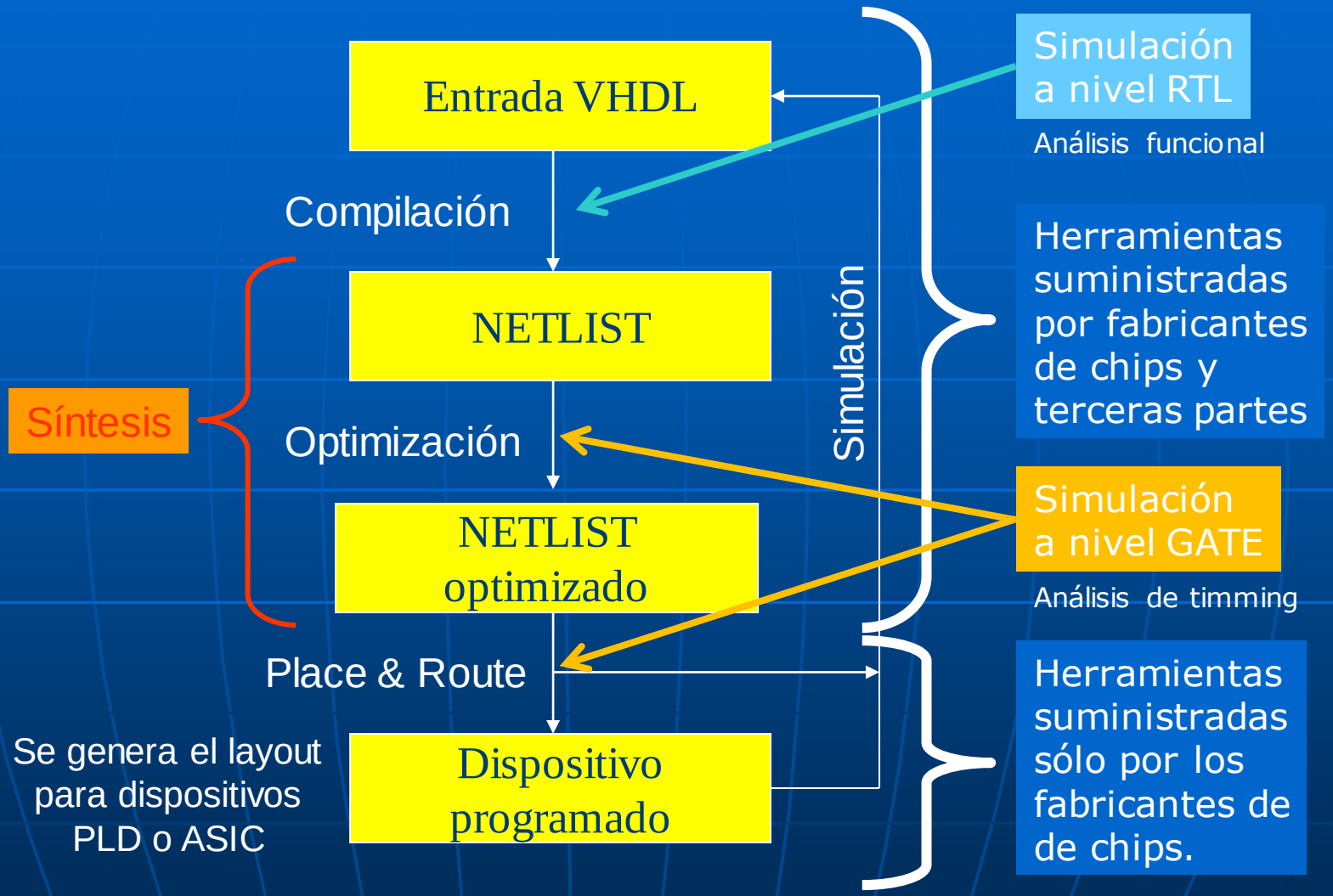
Desde su implementación en el año 1981, VHDL fue estandarizado por primera vez por la IEEE en 1987 (std. 1076) y se realizó un importante actualización en 1993 (con posteriores revisiones en 1994, 2000, 2002 y 2008).

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.

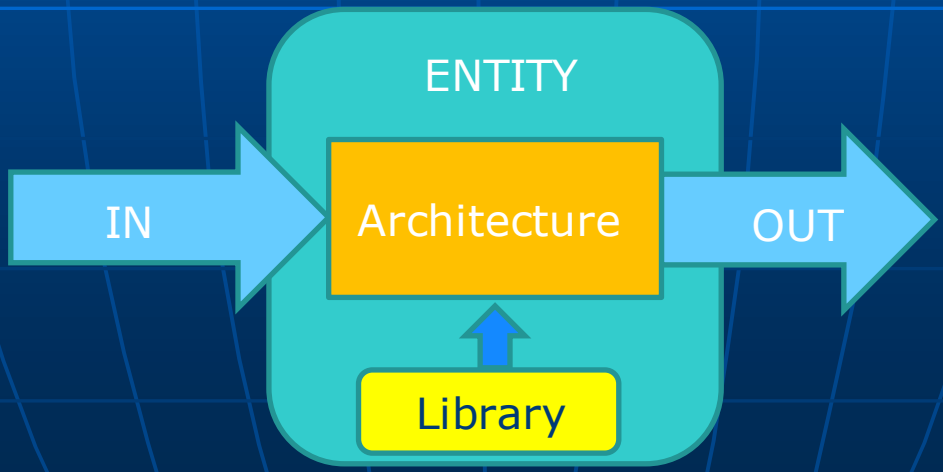
Estructuras en VHDL:

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.

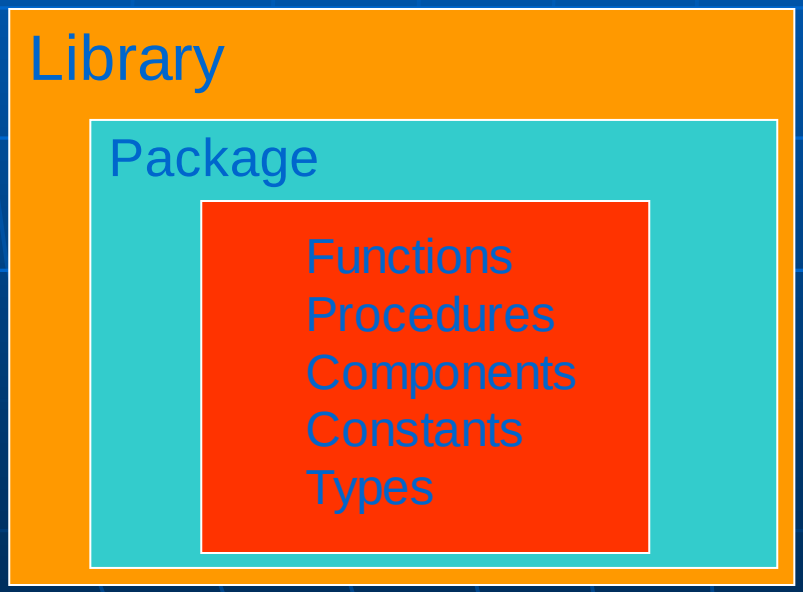
```
ENTITY mux IS
        PORT ( a, b, c, d : IN BIT;
                  s0, s1 : IN BIT;
                z : OUT BIT);
END mux;

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.

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.

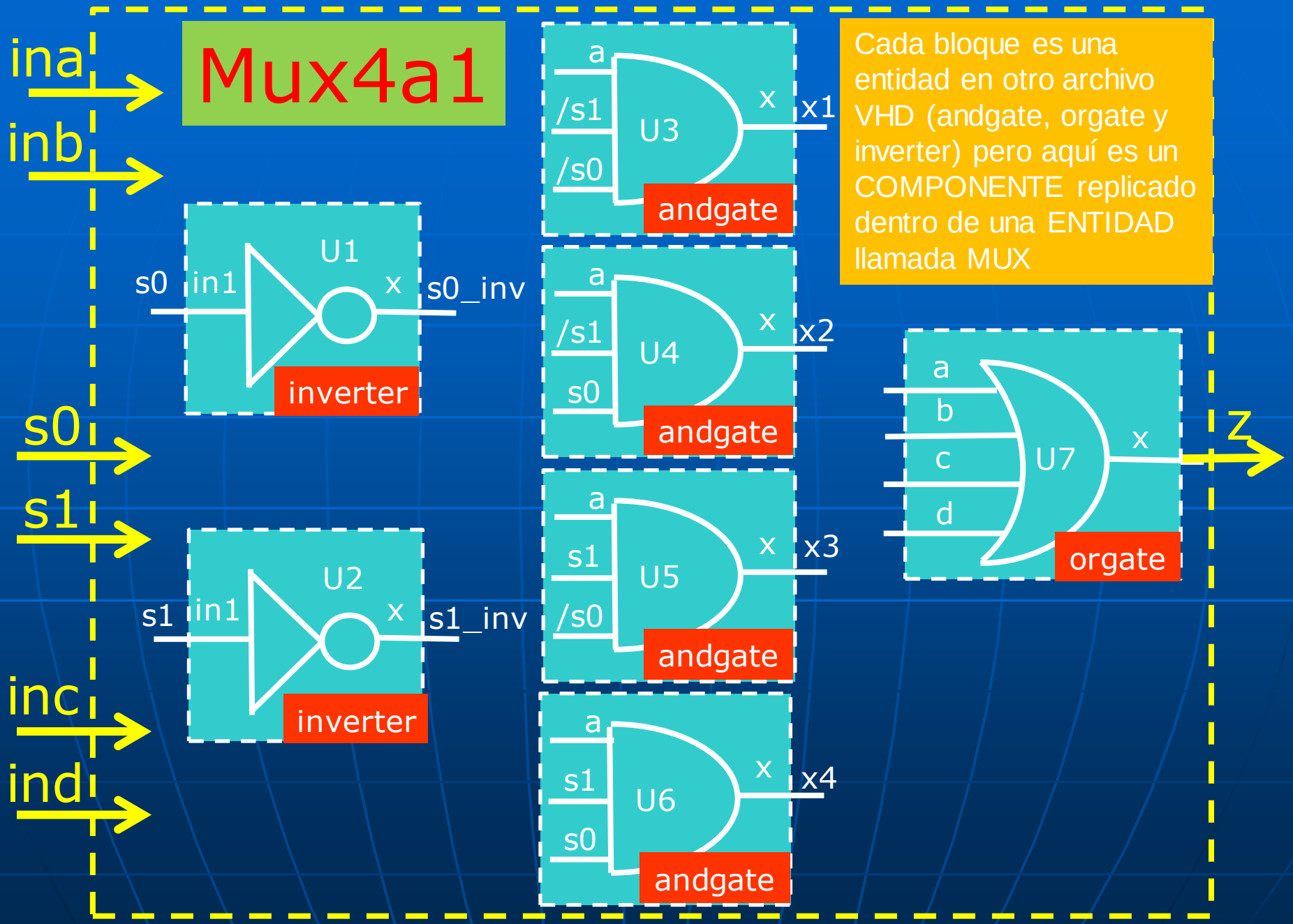
DISEÑO ESTRUCTURAL

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”;


Mux4a1
ina
inb
s0
s1
inc
ind
U1
s0
in1
x
s0_inv
inverter
U2
s1
in1
x
s1_inv
inverter
U3
a
/s1
/s0
x
x1
andgate
U4
a
/s1
s0
x
x2
andgate
U5
a
s1
/s0
x
x3
andgate
U6
a
s1
s0
x
x4
andgate
U7
a
b
c
d
x
z
orgate
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

EJEMPLO DE DISEÑO ESTRUCTURAL

inverter.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4
5  entity inverter is
6  Port ( in1     : in std_logic;
7         x       : out std_logic);
8  end inverter;
9
10
11 architecture Comportamiento of inverter is
12    begin
13
14    x <= NOT in1;
15
16    end Comportamiento;
```

orgate.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4
5  entity orgate is
6  Port ( a       : in std_logic;
7         b       : in std_logic;
8         c       : in std_logic;
9         d       : in std_logic;
10        x       : out std_logic);
11 end orgate;
12
13
14 architecture Comportamiento of orgate is
15    begin
16
17    x <= a OR b OR b OR c OR d;
18
19    end Comportamiento;
```

andgate.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4
5  entity andgate is
6  Port ( a       : in std_logic;
7         b       : in std_logic;
8         c       : in std_logic;
9         x       : out std_logic);
10 end andgate;
11
12
13 architecture Comportamiento of andgate is
14    begin
15
16    x <= a AND b AND b AND c;
17
18    end Comportamiento;
```

EJEMPLO DE DISEÑO ESTRUCTURAL

mux4a1.vhd

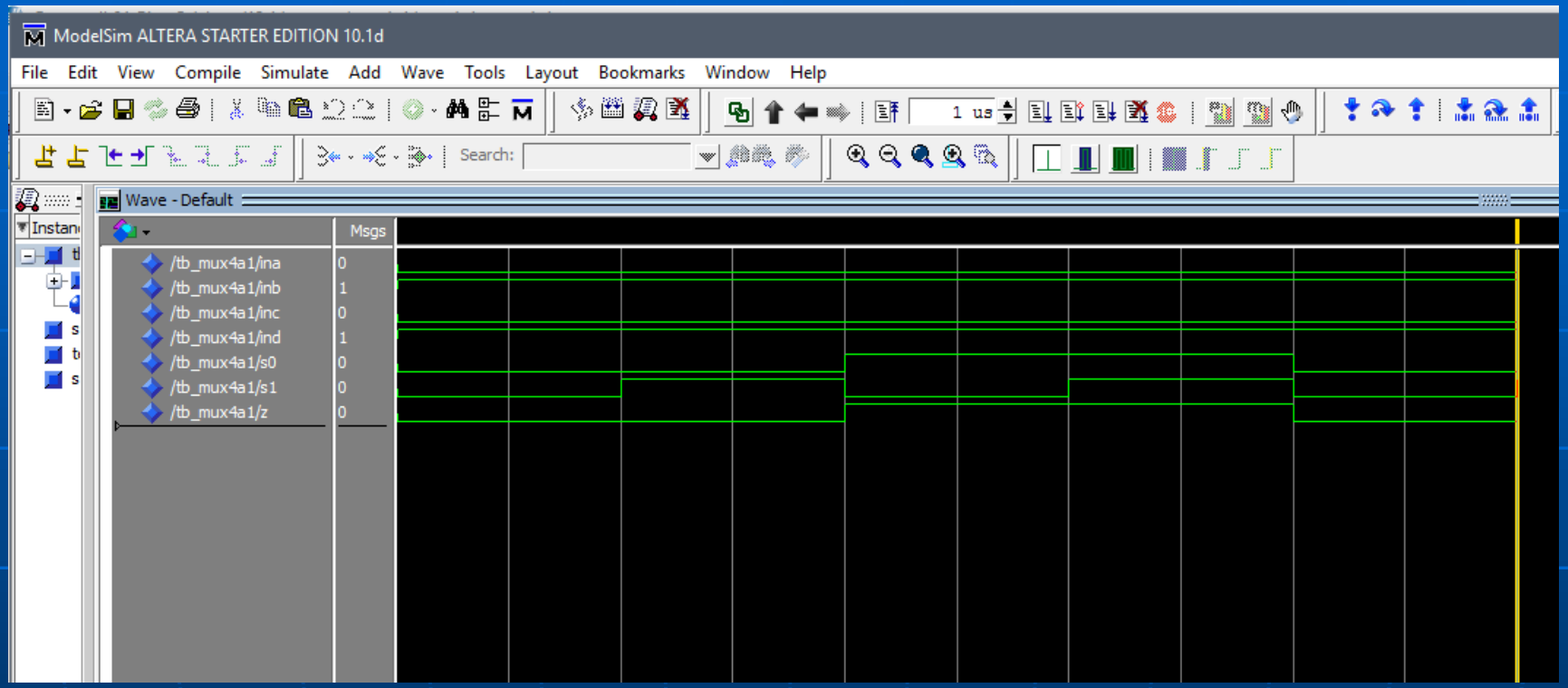
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity Mux4a1 is
4   Port ( ina      : in std_logic;
5          inb      : in std_logic;
6          inc      : in std_logic;
7          ind      : in std_logic;
8          s0       : in std_logic;
9          s1       : in std_logic;
10         z        : out std_logic
11         );
12  end Mux4a1;
13
14
15  ARCHITECTURE netlist OF Mux4a1 IS
16     COMPONENT andgate
17        PORT(a, b, c: IN std_logic; x : OUT std_logic);
18     END COMPONENT;
19
20     COMPONENT inverter
21        PORT(in1 : IN std_logic; x : OUT std_logic);
22     END COMPONENT;
23
24     COMPONENT orgate
25        PORT(a, b, c, d : IN std_logic; x : OUT std_logic);
26     END COMPONENT;
27
28  SIGNAL s0_inv, s1_inv, x1, x2, x3, x4, z1 : std_logic;
29
30  BEGIN
31    U1 : inverter port map (s0, s0_inv);
32    U2 : inverter port map ( s1,   s1_inv);
33    U3 : andgate port map ( ina,  s0_inv,  s1_inv, x1);
34    U4 : andgate port map (inb, s0 , s1_inv  , x2);
35    U5 : andgate port map (inc,  s0_inv,  s1  ,x3);
36    U6 : andgate port map ( ind, s0,  s1,  x4);
37    U7 : orgate port map ( x1, x2,  x3, x4, z1);
38    z <= z1;
39  END netlist;
40
```

EJEMPLO DE DISEÑO ESTRUCTURAL

tb_Mux4a1.vhd | inverter.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity tb_Mux4a1 is
5  end tb_Mux4a1;
6
7  architecture test of tb_Mux4a1 is
8
9     component Mux4a1
10       Port (    ina       : in std_logic;
11                 inb       : in std_logic;
12                 inc       : in std_logic;
13                 ind       : in std_logic;
14                 s0      : in std_logic;
15                 s1      : in std_logic;
16                 z       : out std_logic
17        );
18    end component;
19
20 signal ina, inb, inc, ind, s0, s1, z  : std_logic;
21
22 begin
23
24   uut: Mux4a1 port map ( ina, inb, inc, ind, s0, s1, z);
25
26    estimulos : process
27    begin
28          ina <= '0';
29          inb <= '1';
30          inc <= '0';
31          ind <= '1';
32          s0 <= '0';
33          s1 <= '0';
34          wait for 200 ns;
35          s0 <= '0';
36          s1 <= '1';
37          wait for 200 ns;
38          s0 <= '1';
39          s1 <= '0';
40          wait for 200 ns;
41          s0 <= '1';
42          s1 <= '1';
43          wait for 200 ns;
44
45
46               end process estimulos;
47
48   end test;
```

EJEMPLO DE DISEÑO ESTRUCTURAL



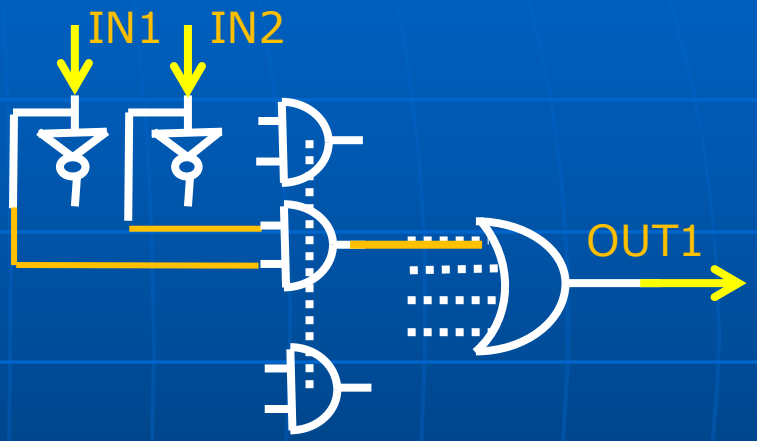
DISEÑO POR FLUJO DE DATOS

Ejemplo de compuerta AND de 2 entradas

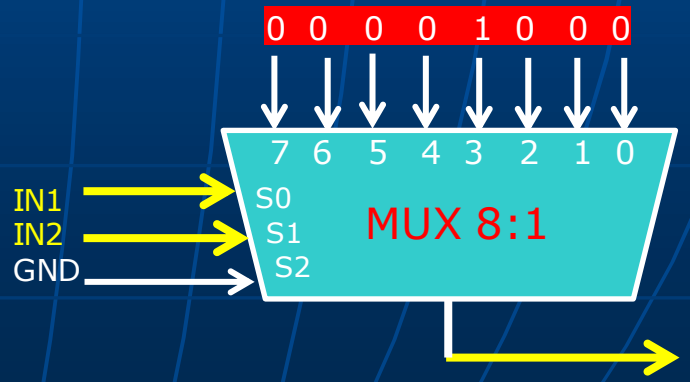
and2.vhd - Text Editor

```
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 sería 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

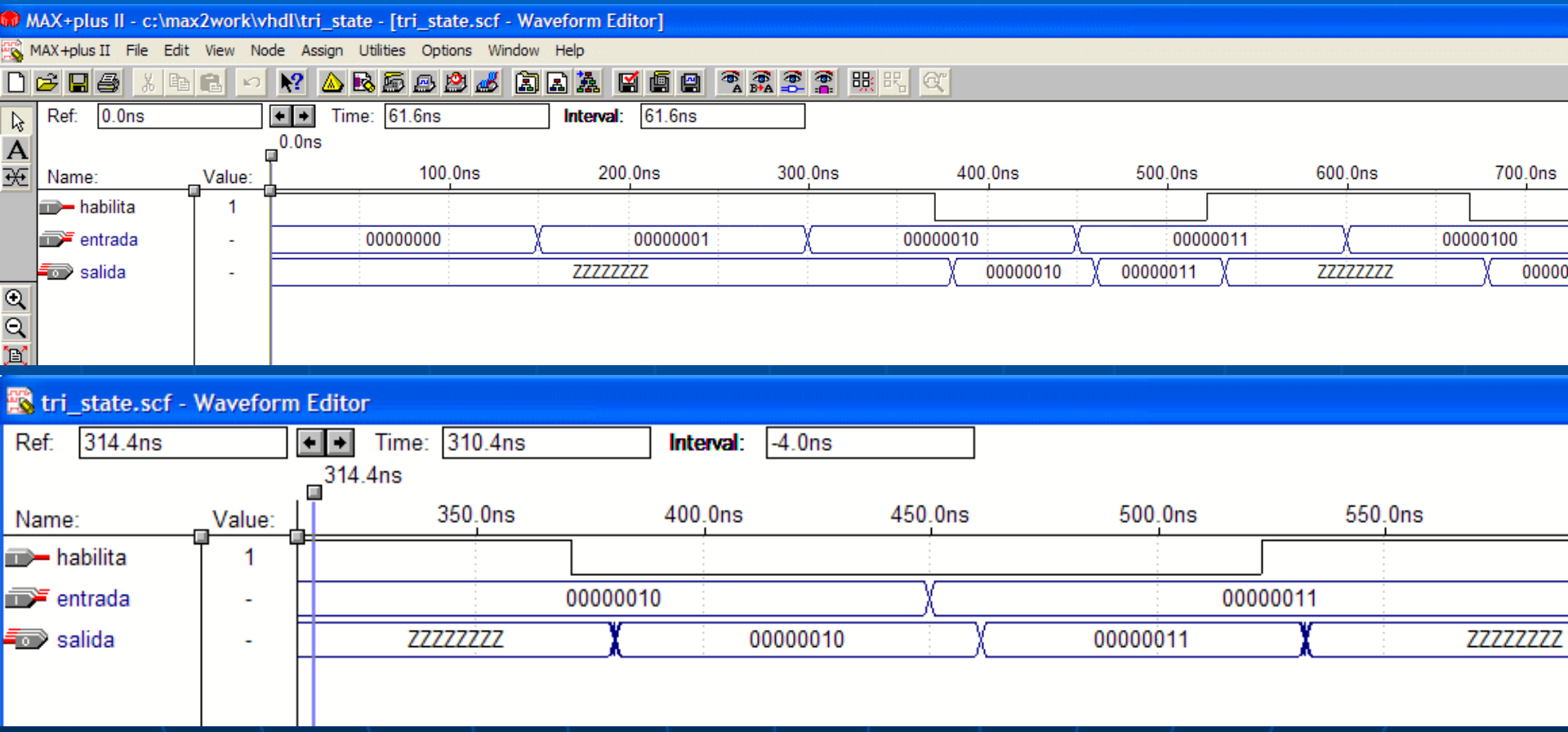
tri_state.vhd - Text Editor

```
-- 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;
```