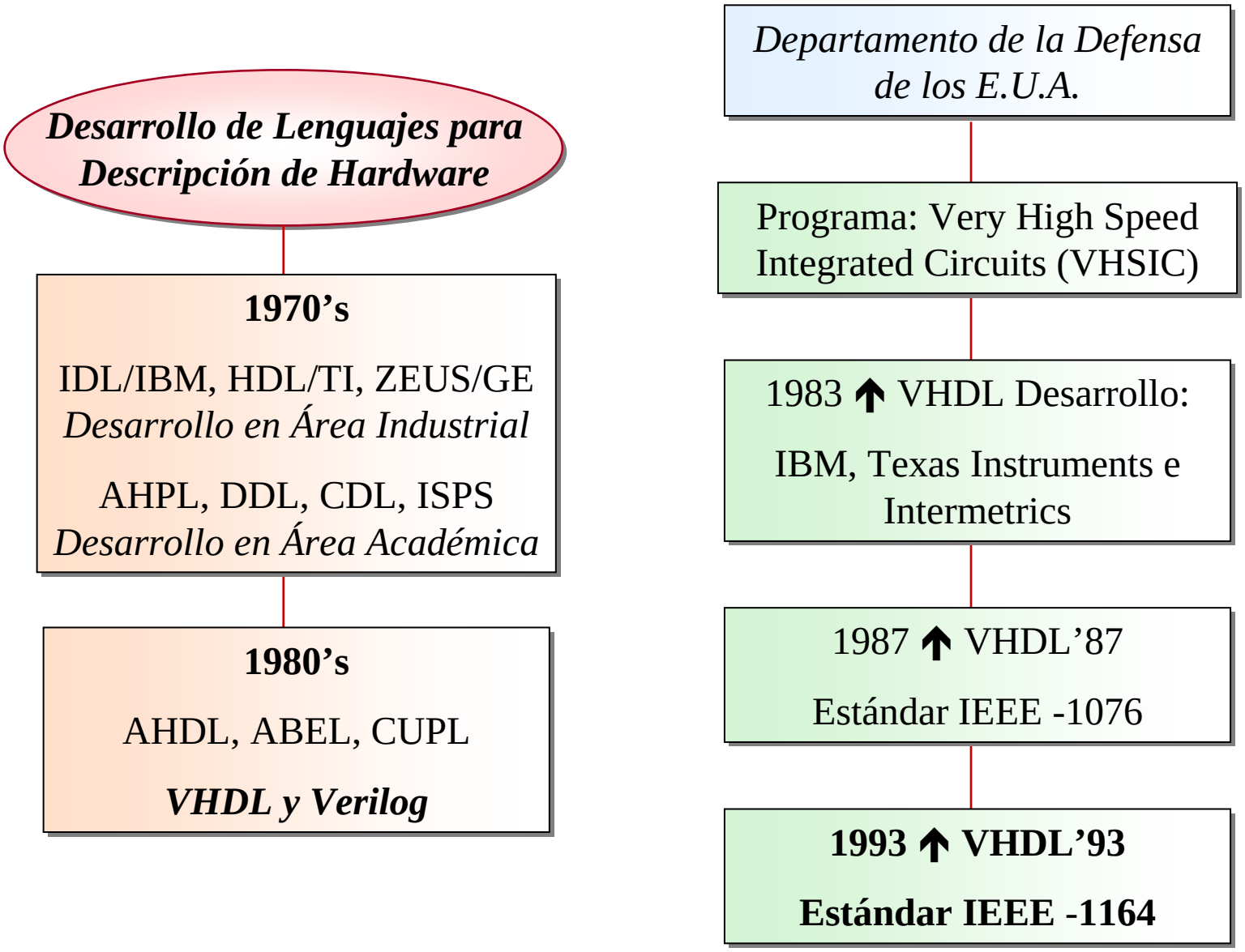
Curso:

VHDL (VHSIC Hardware Description Language)

VHSIC – Very High Speed Integrated Circuit


Desarrollo de Lenguajes para Descripción de Hardware
1970's
IDL/IBM, HDL/TI, ZEUS/GE
Desarrollo en Área Industrial
AHPL, DDL, CDL, ISPS
Desarrollo en Área Académica
1980's
AHDL, ABEL, CUPL
VHDL y Verilog
Departamento de la Defensa de los E.U.A.
Programa: Very High Speed Integrated Circuits (VHSIC)
1983 VHDL Desarrollo:
IBM, Texas Instruments e Intermetrics
1987 VHDL'87
Estándar IEEE -1076
1993 VHDL'93
Estándar IEEE -1164

Ventajas de VHDL
Notación Estandarizada
Disponibilidad al Público
Independencia del Sistema de Desarrollo
Independencia de la Metodología de Diseño (PLD's, ASIC's, FPGA's)
Independencia de la Tecnología y Proceso de Fabricación (CMOS, Bipolar, BiCMOS)
Reutilización de Código
Capacidad descriptiva del comportamiento del sistema en distintos niveles de abstracción: ***Algoritmico***, ***RTL*** (Register Transfer Logic) ***o concurrente***, ***estructural*** *(Lógico)*, ***Netlist.***
Facilitar la Verificación/Prueba y puesta a punto del sistema a diseñar.
Adición de la extensión analógica (IEEE1076.1) que permite la especificación, simulación y síntesis de sistemas digitales, analógicos y mixtos

Elementos sintácticos del VHDL	
Comentarios	Se consideran comentarios después de dos guiones medios seguidos “--”.
Símbolos especiales	Existen caracteres especiales sencillos como (&, #, +, *, =) o dobles como (:=, <=).
Identificadores	Es lo que se usa para dar nombre a los diferentes objetos del lenguaje.
Números	Se considera que se encuentra en base 10, se admite la notación científica convencional es posible definir números en otras bases utilizando el símbolo # : 2#11000100#
Caracteres	Es cualquier letra o carácter entre comillas simples: ‘3’, ‘t’
Cadenas	Son un conjunto de caracteres englobados por comillas dobles: “hola”
Cadenas de bits	Los tipos bit y bit_vector son en realidad tipo carácter y arreglo de caracteres respectivamente, se coloca un prefijo para indicar la base : O”126”, X”FE”
Palabras reservadas	Son las instrucciones, órdenes y elementos que permiten definir sentencias.

Identificadores
Nombres o etiquetas que se usan para referirse a: Variables, Constantes, Señales, Procesos, Entidades, etc.
Están formados por números, letras (mayúsculas o minúsculas) y guión bajo "_" con las reglas especificadas en la tabla siguiente.
Longitud (Número de Caracteres): Sin restricciones
Palabras reservadas por VHDL no pueden ser identificadores
En VHDL, un identificador en mayúsculas es igual a su contraparte en minúsculas

Reglas para especificar un identificador	*Incorrecto*	*Correcto*
Primer carácter debe ser siempre una letra mayúscula o minúscula	4Suma	Suma4
Segundo carácter no puede ser un guión bajo (_)	S_4bits	S4_bits
Dos guiones bajos no son permitidos	Resta__4	Resta_4_
Un identificador no puede utilizar símbolos especiales	Clear#8	Clear_8

Lista de palabras reservadas en VHDL

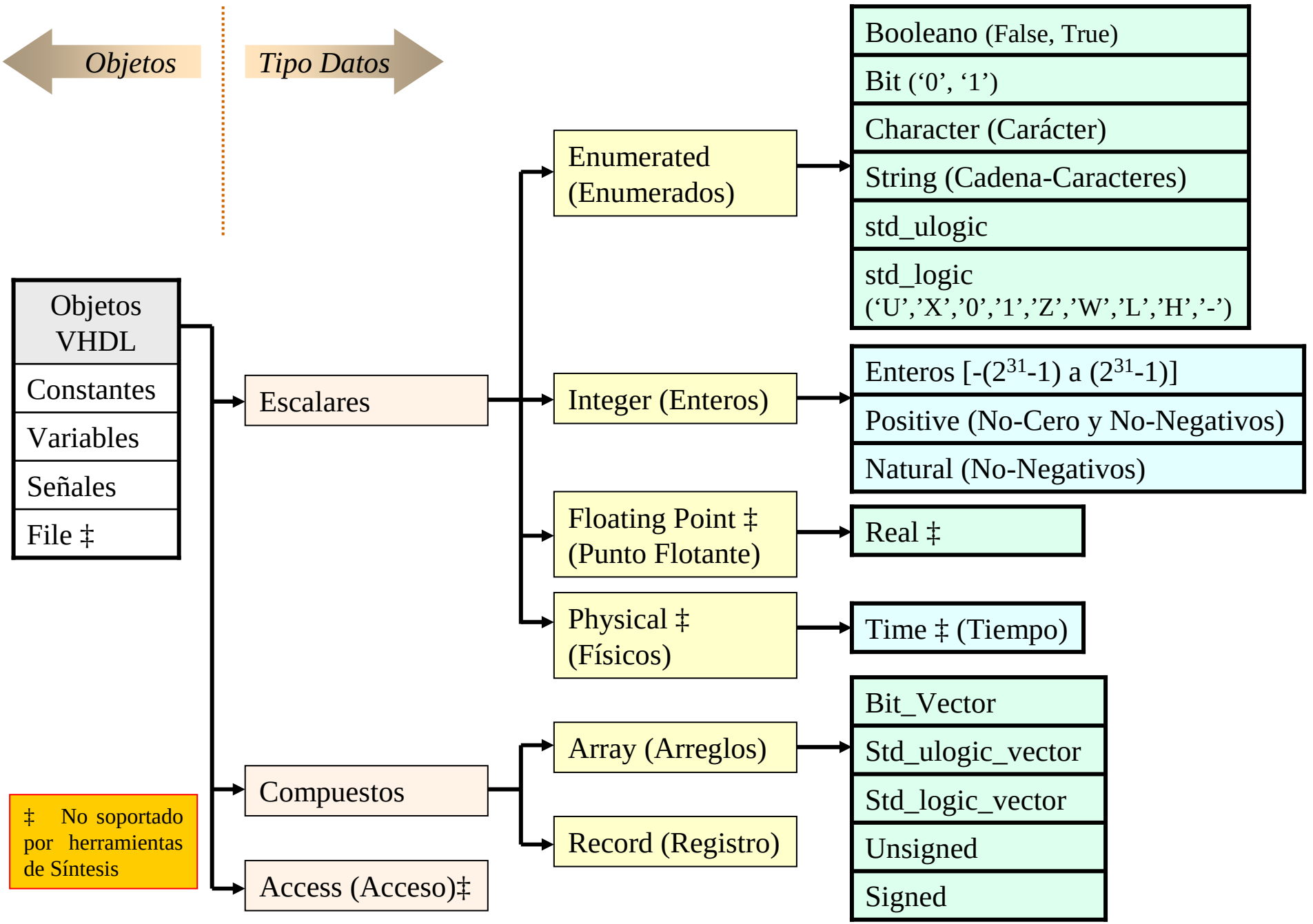
abs	downto	library	postponed	subtype
access	else	linkage	procedure	then
after	elsif	literal	process	to
alias	end	loop	pure	transport
all	entity	map	range	type
and	exit	mod	record	unaffected
architecture	file	nand	register	units
array	for	new	reject	until
assert	function	next	rem	use
attribute	generate	nor	report	variable
begin	generic	not	return	wait
block	group	null	rol	when
body	guarded	of	ror	while
buffer	if	on	select	with
bus	impure	open	severity	xnor
case	in	or	shared	xor
component	inertial	others	signal	
configuration	inout	out	sla	
constant	is	package	sra	
disconnect	label	port	srl	

Mayor

Menor

Precedencia de operadores					
**	ABS	NOT			
*	/	MOD	REM		
+ (signo)	- (signo)				
+	-	&			
=	/=	<	<=	>	<=
AND	OR	NAND	NOR	XOR	XNOR

La precedencia de operadores se encuentran ordenados de mayor (arriba) a menor (abajo), los operadores que se encuentran en la misma fila tienen la misma precedencia y serán evaluados siguiendo el orden de izquierda a derecha.


Objetos
Tipo Datos
Objetos VHDL
Constantes
Variables
Señales
File ‡
Escalares
Compuestos
Access (Acceso)‡
Enumerated (Enumerados)
Integer (Enteros)
Floating Point ‡ (Punto Flotante)
Physical ‡ (Físicos)
Array (Arreglos)
Record (Registro)
Booleano (False, True)
Bit (‘0’, ‘1’)
Character (Carácter)
String (Cadena-Caracteres)
std_ulogic
std_logic (‘U’,’X’,’0’,’1’,’Z’,’W’,’L’,’H’,’-’)
Enteros [-(2^31-1) a (2^31-1)]
Positive (No-Cero y No-Negativos)
Natural (No-Negativos)
Real ‡
Time ‡ (Tiempo)
Bit_Vector
Std_ulogic_vector
Std_logic_vector
Unsigned
Signed
‡ No soportado por herramientas de Síntesis

Objetos de Datos

Un objeto de datos en VHDL es un elemento que toma un valor de algún tipo de dato determinado, según sea el tipo de dato, el objeto poseerá un conjunto de propiedades. En VHDL los objetos de datos son generalmente una de las tres clases siguientes:

Constantes

Una constante es un elemento que puede tomar un único valor de un tipo dato, las constantes pueden ser declaradas dentro de entidades, arquitecturas, procesos y paquetes.

CONSTANT identidicador : tipo := valor;

Ejemplo

CONSTANT byte: integer := 8;

Variables

Las variables pueden ser modificadas cuando sea necesario, pueden ser declaradas solamente dentro de los procesos y subprogramas.

VARIABLE identidicador : tipo [:= valor];

Ejemplo

VARIABLE aux1, aux2: bit;

Señales

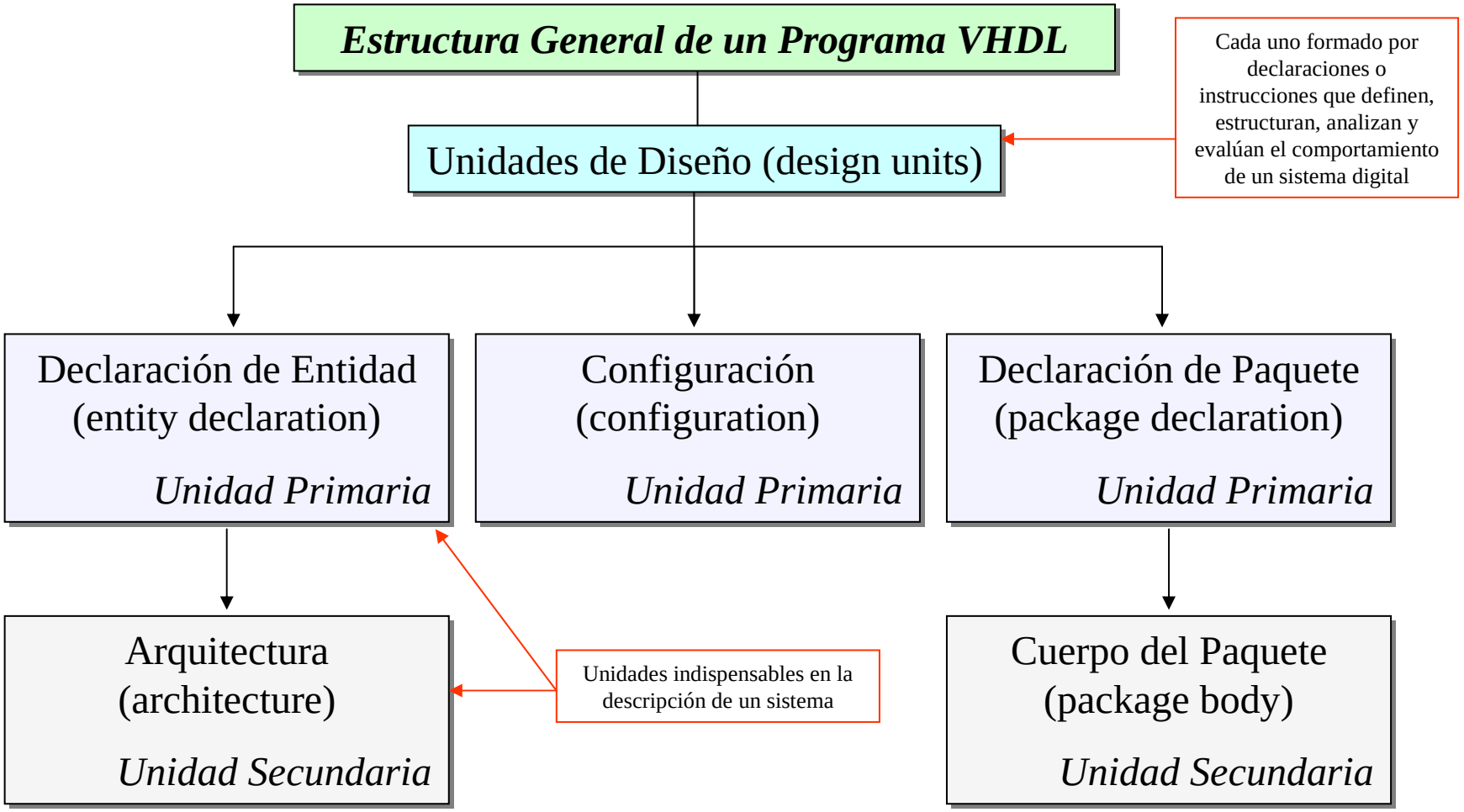
Las señales si pueden almacenar o pasar valores lógicos, por lo tanto, representan elementos de memoria o conexiones y si pueden ser sintetizadas. Son declaradas en las arquitecturas antes del BEGIN.

SIGNAL identidicador : tipo [:= valor];

Ejemplo

SIGNAL A, B : bit := ‘0’;

SIGNAL dato: bit_vector (7 **downto** 0);

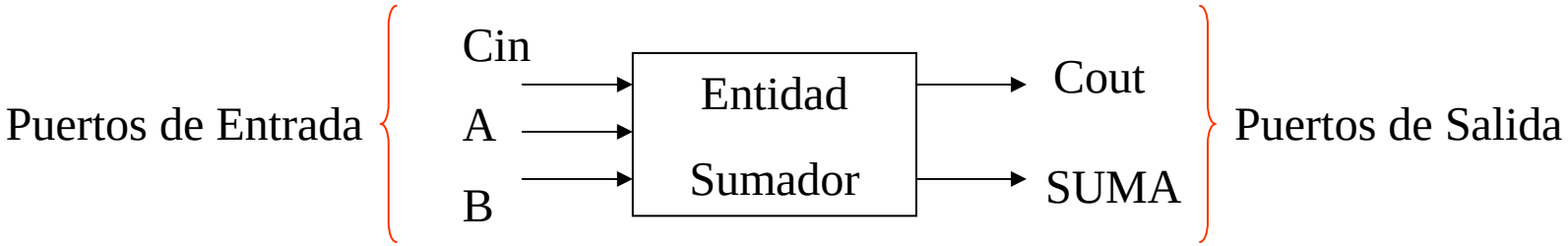

Estructura General de un Programa VHDL
Unidades de Diseño (design units)
Cada uno formado por declaraciones o instrucciones que definen, estructuran, analizan y evalúan el comportamiento de un sistema digital
Declaración de Entidad (entity declaration)
Unidad Primaria
Configuración (configuration)
Unidad Primaria
Declaración de Paquete (package declaration)
Unidad Primaria
Arquitectura (architecture)
Unidad Secundaria
Unidades indispensables en la descripción de un sistema
Cuerpo del Paquete (package body)
Unidad Secundaria

entidad (**entity**) ➔ Bloque elemental de diseño

Circuitos elementales digitales que forman de manera individual o en conjunto un sistema digital

Ejemplos: Compuertas, Flip-Flops, Sumadores/Restadores, Multiplexores, Contadores, Multiplicadores, ALUs, Neurona-Digital, etc.

Ejemplo-1: **Sumador**



Declaración de una entidad ➔ Consiste en la descripción de los puertos de entrada o salida de un circuito, el cual es identificado como una entidad (**entity**)

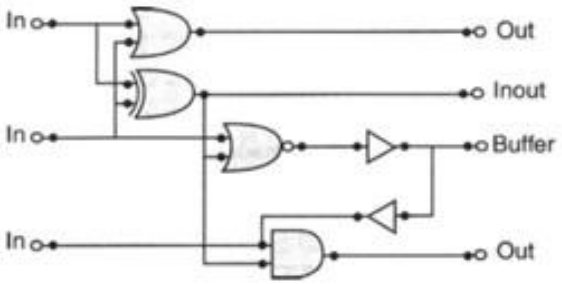
¡Importante!

No se describe cómo será realizado o implementado el circuito, es decir, su Arquitectura

Descripción de un Puerto

Nombre

Identificador



Modo

in = Entrada

out = Salida

inout

- Puerto de Entrada (Lectura) y Salida (Escritura)
- El valor leído (Entrada) es aquél que llega al puerto, y no el valor que se le asigna (Salida), en caso de existir.

buffer

- Similar al Puerto de Salida (Escritura), pero además puede ser leído.
- El valor leído (Entrada) es el mismo valor asignado (Salida) al puerto.

Tipo de Dato

Conjuntos de Valores que se les ha asignado un nombre (p.ej. *bit, boolean, bit_vector*, etc), de tal forma que un objeto (p.ej. una Señal) de un determinado Tipo (p.ej. el tipo bit_vector) pueda tomar cualquier valor dentro del conjunto de valores que define al Tipo especificado.

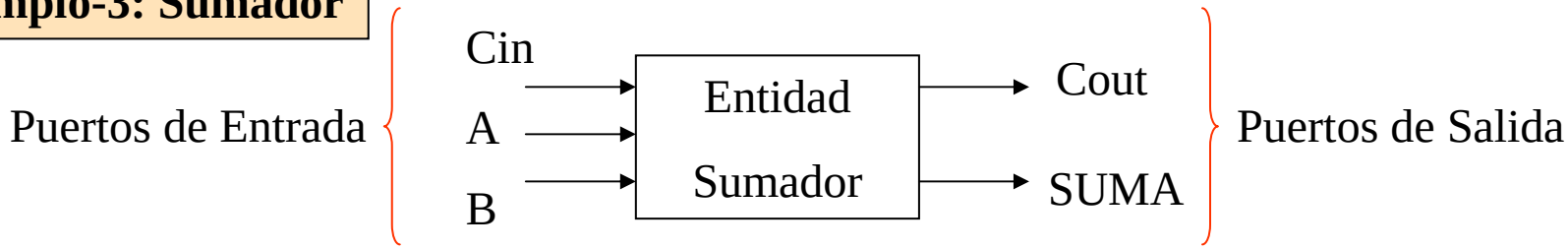
bit (pkg.*standard*)	Valores de '0' o '1' Lógico
boolean (pkg.*standard*)	Define valores de cierto o falso de acuerdo con una expresión
bit_vector (pkg.*standard*)	Conjunto de bits que representa a un grupo de señales de ent. o sal.
integer (pkg.*standard*)	Números enteros
std_logic (pkg.*std_logic_1164*)	Valores 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'
std_logic_vector (pkg.*std_logic_1164*)	Arreglos de std_logic

Más tipos

Se irán introduciendo conforme avance el curso

Paquete (pkg.) en el cual es definido el tipo.
Ver: "Uso de Librerías y Paquetes"

Ejemplo-3: Sumador



(entity) Inicia declaración de la entidad

(--) Indica Comentario

Identificador de la entidad

Nombres de los puertos

Línea N°.	Sumador-completo de dos datos con longitudes de 1-bit (Declaración de Entidad)
1	--Declaración de la entidad de un circuito sumador
2	**entity** sumador **is**
3	**port** (A, B, Cin: **in** bit;
4	SUMA, Cout: **out** bit);
5	**end** sumador;

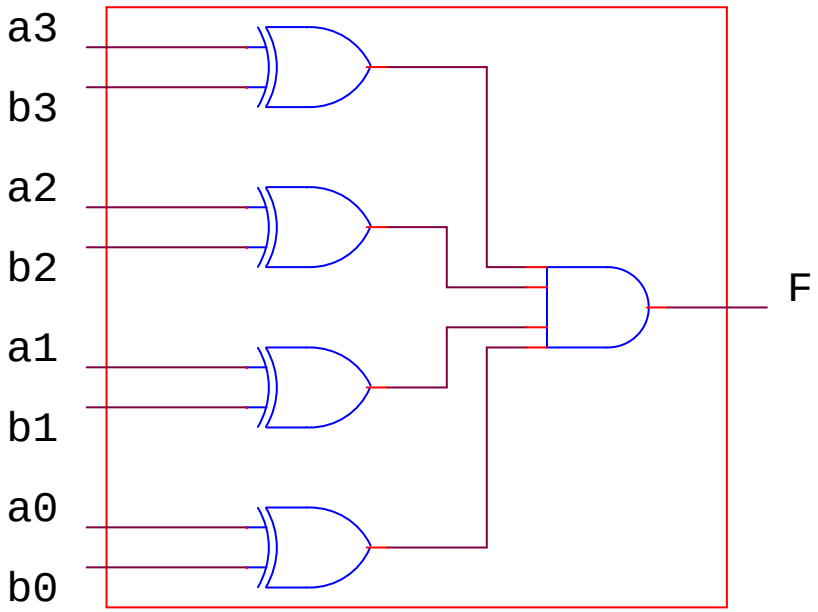
Modo de Operación

Tipo de Dato

(;) Finaliza declaración o subdeclaración

(end) Finaliza declaración de la entidad

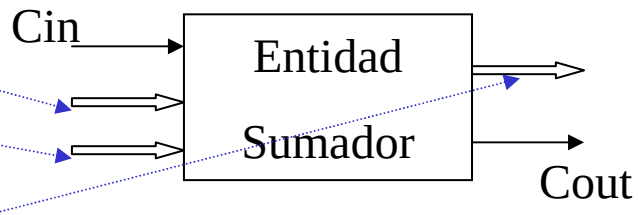
Ejemplo-4



Línea N°.	Detector – Uso de dos datos con longitudes de 4-bit (Declaración de Entidad)
1	--Declaracion de la entidad
2	**entity** circuito **is**
3	**port** (a3, b3, a2, b2, a1, b1, a0, b0: **in** bit;
4	F: **out** bit);
5	**end** circuito;

Ejemplo-5

vector_A	= [A3, A2, A1, A0]
vector_B	= [B3, B2, B1, B0]
vector_SUMA	= [S3, S2, S1, S0]



Declaración de Puertos Tipo-Vector

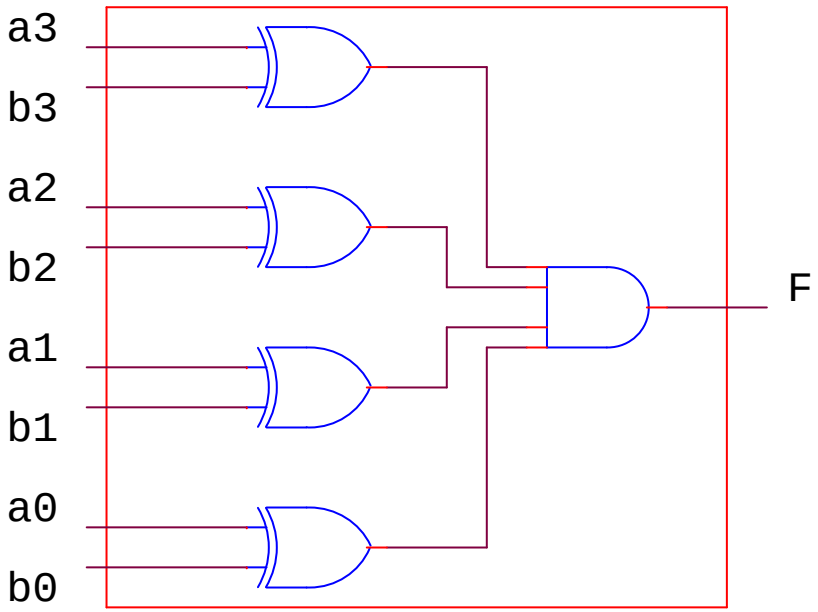
```
port (vector_A, vector_B: in bit_vector (3 downto 0);
      vector_SUMA: out bit_vector (3 downto 0));
```

Para ordenar en forma ascendente utilizar **to** en lugar de **downto** (p.ej. 0 to 3)

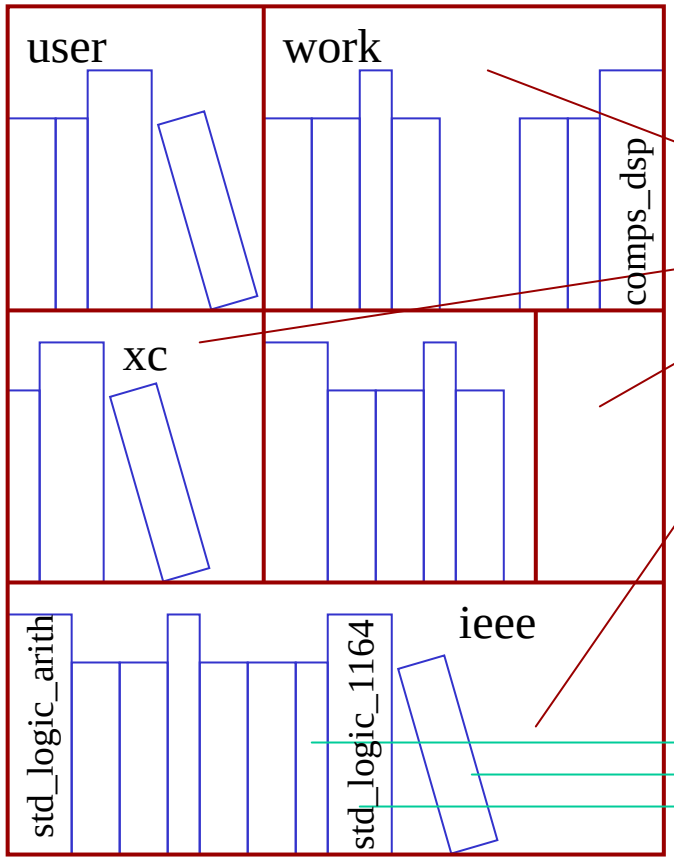
Sumador-completo de dos datos con longitudes de 4-bit (Declaración de Entidad – Uso de Vectores)

```
entity sumador is
    port (A, B: in bit_vector (3 downto 0);
          Cin: in bit;
          Cout: out bit;
          SUMA: out bit_vector (3 downto 0));
end sumador;
```

Ejemplo-6



Línea N°.	Detector – Uso de dos datos con longitudes de 4-bit (Declaración de Entidad – Uso de Vectores)
1	--Declaracion de la entidad
2	**entity** circuito **is**
3	**port** (a, b: **in** bit_vector (3 downto 0);
4	F: **out** bit);
5	**end** circuito;



Biblioteca (library)

•Lugar donde se almacenan los Paquetes definidos por el fabricante de la herramienta de desarrollo o el usuario.

•Lugar donde se permite almacenar resultados de la compilación de diseños, con el fin de utilizarlos en otros.

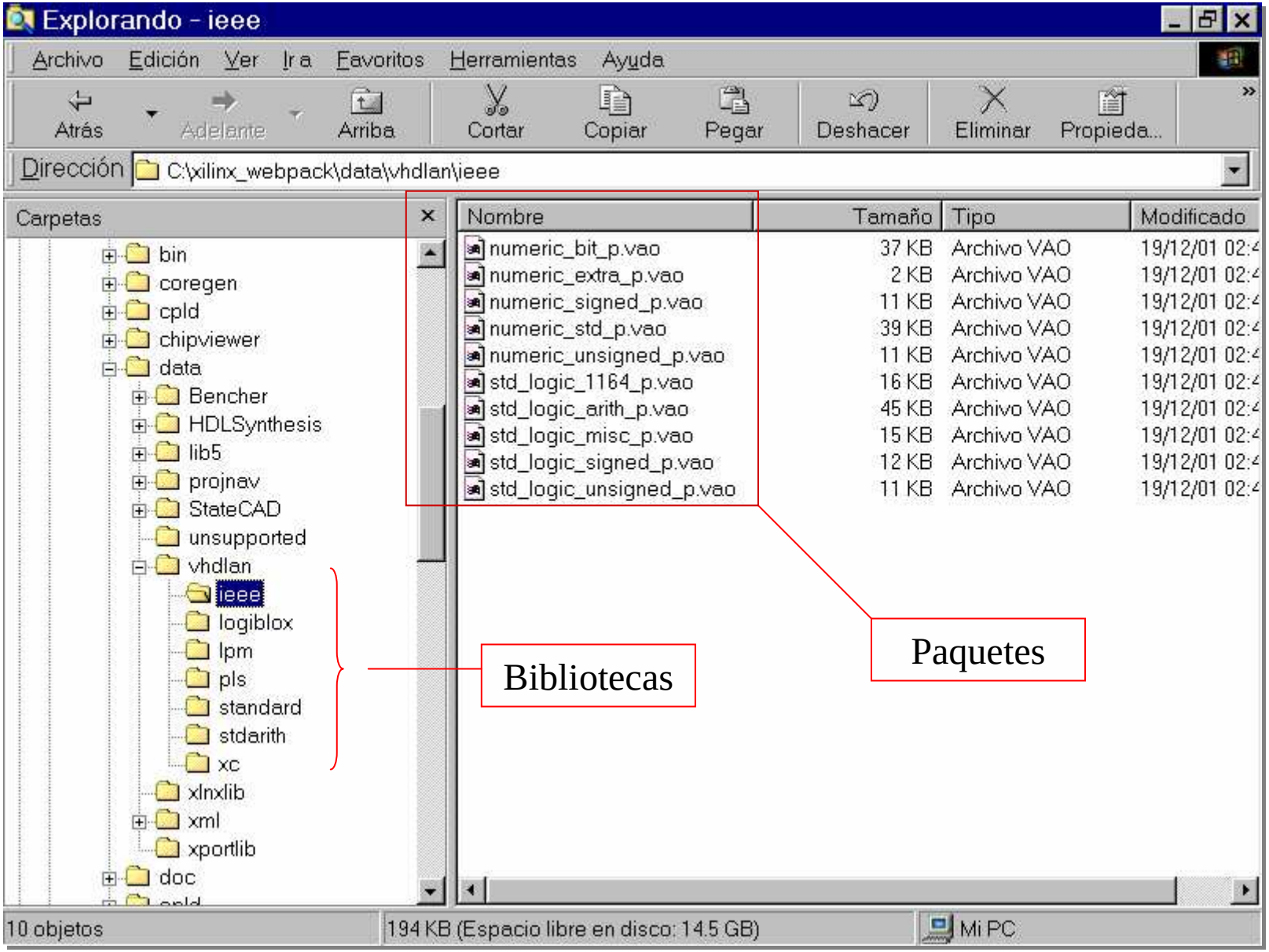
Paquetes

•Un paquete contiene:

- Declaraciones de Tipos y Subtipos de Datos
- Definiciones de Constantes
- Definiciones de Funciones y Procedimientos
- Declaraciones de Componentes (Sumadores, Restadores, Contadores, Multiplicadores, etc)

Un Paquete = Macro-Unidad de Diseño

Objetivo: Facilitar el diseño


Explorando - ieee
Archivo Edición Ver Ir a Favoritos Herramientas Ayuda
Atrás Adelante Arriba Cortar Copiar Pegar Deshacer Eliminar Propieda...
Dirección C:\xilinx_webpack\data\vhdlan\ieee
Carpetas
bin
coregen
cpld
chipviewer
data
Bencher
HDLSynthesis
lib5
projnav
StateCAD
unsupported
vhdlan
ieee
logiblox
lpm
pls
standard
stdarith
xc
xlnxlib
xml
xportlib
doc
Nombre Tamaño Tipo Modificado
numeric_bit_p.vao 37 KB Archivo VAO 19/12/01 02:4
numeric_extra_p.vao 2 KB Archivo VAO 19/12/01 02:4
numeric_signed_p.vao 11 KB Archivo VAO 19/12/01 02:4
numeric_std_p.vao 39 KB Archivo VAO 19/12/01 02:4
numeric_unsigned_p.vao 11 KB Archivo VAO 19/12/01 02:4
std_logic_1164_p.vao 16 KB Archivo VAO 19/12/01 02:4
std_logic_arith_p.vao 45 KB Archivo VAO 19/12/01 02:4
std_logic_misc_p.vao 15 KB Archivo VAO 19/12/01 02:4
std_logic_signed_p.vao 12 KB Archivo VAO 19/12/01 02:4
std_logic_unsigned_p.vao 11 KB Archivo VAO 19/12/01 02:4
Bibliotecas
Paquetes
10 objetos
194 KB (Espacio libre en disco: 14.5 GB)
Mi PC

Para llamar un paquete es necesario llamar a la librería/biblioteca que lo contiene (donde ha sido compilado)

Sintaxis: **use** nombre_librería**.**nombre_paquete**.all**;

Ejemplo: **use** ieee.std_logic_1164.**all**;

Uso del paquete **std_logic_1164** incluido en la biblioteca **ieee**

Permite el uso de todos los componentes almacenados en el paquete

Paquetes predefinidos comúnmente utilizados	
Standard	
standard	•Contiene tipos básicos: bit, bit_vector, integer •Paquete incluido por omisión.
IEEE	
std_logic_1164	•Define los tipos: std_logic, std_ulogic, std_logic_vector, std_ulogic_vector •Define funciones de conversión basadas sobre estos tipos.
numeric_bit	•Define tipos de vectores signados y no-signados basados en el tipo bit y todos los operadores aritméticos sobre estos tipos. •Define funciones extendidas y de conversión para dichos tipos.
numeric_std	Define tipos de vectores signados y no-signados basados en el tipo std_logic. Paquete equivalente al Paquete std_logic_arith
Synopsys	
std_logic_arith	•Define tipos de vectores signados y no-signados, y todos los operadores aritméticos sobre estos tipos. •Define funciones extendidas y de conversión para dichos tipos.
std_logic_unsigned	•Define operadores aritméticos sobre el tipo std_ulogic_vector y los considera como operadores no-signados.
std_logic_signed	•Define operadores aritméticos sobre el tipo std_logic_vector y los considera como operadores signados.
std_logic_misc	•Define tipos, subtipos, constantes y funciones complementarios para el paquete std_logic_1164.

Ejemplo-7

Multiplicador de dos datos con longitudes de 2-bit
(Declaración de Entidad – Uso de Biblioteca y Paquete)

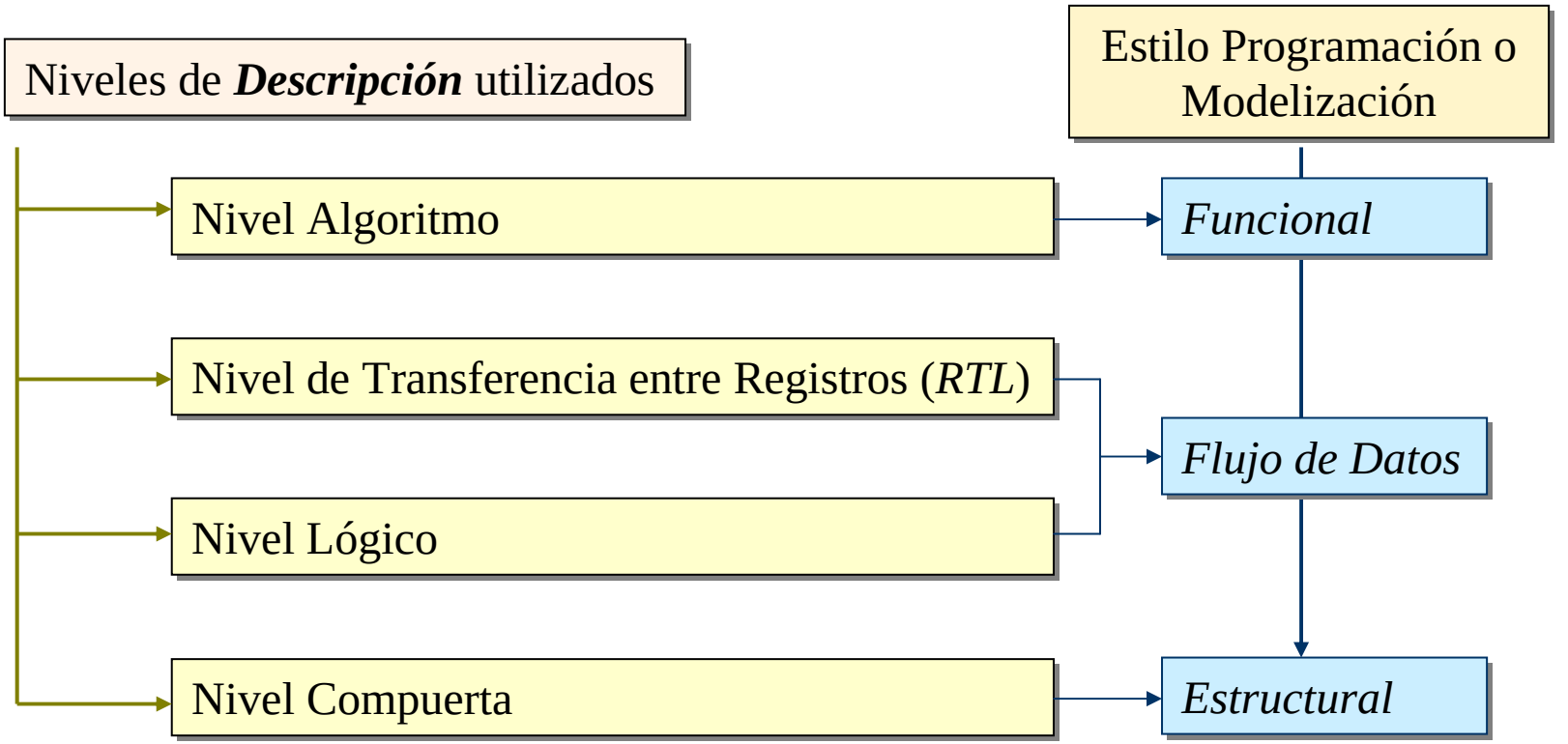
```
library ieee;
use ieee.std_logic_1164.all;
entity multiplica is
        port (X0, X1, Y0, Y1: in std_logic;
                Z3, Z2, Z1, Z0: out std_logic);
end multiplica;
```

arquitectura (**architecture)**

Unidad de Diseño Secundaria que describe el comportamiento interno de una entidad.

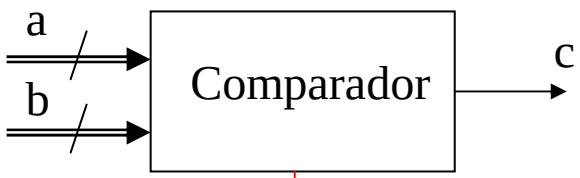
¿Cómo? - A través de la programación de varios procedimientos que permitan que la entidad (**entity**) cumpla con las condiciones de operación o comportamiento deseadas.

Niveles de ***Descripción*** utilizados



Funcional - En este caso, se describen las relaciones entre las entradas y salidas, sin importar la estructura o implementación física del sistema o circuito.

Ejemplo-8



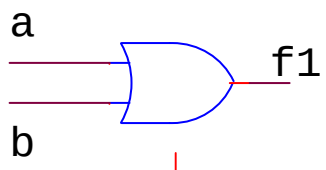
Uso de **if-then-else** (construcción secuencial)

si a = b entonces c = 1

si a ≠ b entonces c = 0

Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	--Ejemplo de una descripción abstracta (funcional)
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (1 **downto** 0);
6	c: **out** bit);
7	**end** comp;
8	**architecture** funcional **of** comp **is**
9	**begin**
10	**process** (a,b)
11	**begin**
12	**if** a = b **then**
13	c <= ‘1’;
14	**else**
15	c <= ‘0’;
16	**end if**;
17	**end process**;
18	**end** funcional;

Ejemplo-9



a	b	f1
0	0	0
0	1	1
1	0	1
1	1	1

Línea Nº	Arquitectura - Compuerta OR de dos entradas
1	--Ejemplo de una descripción abstracta (funcional)
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** COMP_OR **is**
5	**port** (a,b: **in** std_logic;
6	f1: **out** std_logic);
7	**end** COMP_OR;
8	**architecture** ACOMP_OR **of** COMP_OR **is**
9	**begin**
10	**process** (a,b) **begin**
11	**if** (a = ‘0’ **and** b=‘0’) **then**
12	f1 <= ‘0’;
13	**else**
14	f1 <= ‘1’;
15	**end if**;
16	**end process**;
17	**end** ACOMP_OR;

Flujo de Datos - En este caso, se describe la forma en la que los datos se pueden transferir entre los diferentes módulos operativos que constituyen la entidad (sistema o circuito)

➢ La construcción **when-else**

Ejemplo-10

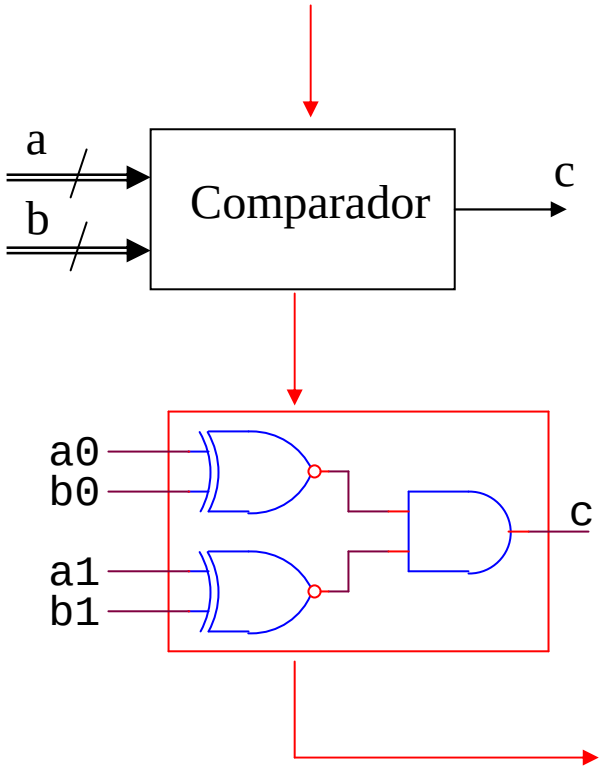
Comparador (Ejemplo-8)

Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	--Ejemplo de una arquitectura usando when-else
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (1 **downto** 0);
6	c: **out** bit);
7	**end** comp;
8	**architecture** f_datos **of** comp **is**
9	**begin**
10	c <= ‘1’ **when** (a = b) **else** ‘0’;
11	**end** f_datos;
12	

➢ Uso de **ecuaciones booleanas**

Ejemplo-11

Comparador (Ejemplo-8)



Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	--Ejemplo de una arquitectura usando ecs. booleanas
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (1 **downto** 0);
6	c: **out** bit);
7	**end** comp;
8	**architecture** booleana **of** comp **is**
9	**begin**
10	c <= (a(1) **xnor** b(1)) **and** (a(0) **xnor** b(0));
11	**end** booleana;
12	

Estructural - En este caso, el comportamiento de un sistema o circuito es descrito mediante modelos lógicos establecidos de los componentes que conforman al sistema o circuito, como son: Compuertas, Sumadores, Contadores, etc.

Modelos lógicos pueden ser:

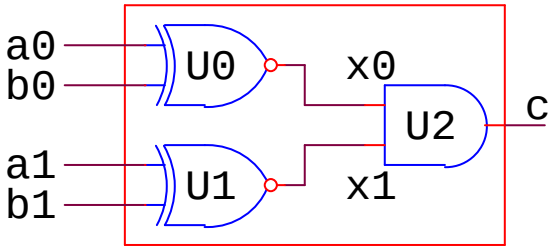
Diseñados por el Usuario

Predefinidos por el Fabricante

Almacenados en Paquetes contenidos en las bibliotecas de la Herramienta de Desarrollo

Ejemplo-12

Comparador (Ejemplo-8)



Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**use** work.compuertas.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (0 **to** 1);
6	c: **out** bit);
7	**end** comp;
8	**architecture** estructural **of** comp **is**
9	**signal** x: bit_vector (0 **to** 1);
10	**begin**
11	U0: xnor2 **port map** (a(0), b(0), x(0));
12	U1: xnor2 **port map** (a(1), b(1), x(1));
13	U2: and2 **port map** (x(0), x(1), c);
14	**end** estructural;