Sistemas Electrónicos Digitales

Tema #4

Diseño mediante Lenguajes de Descripción Hardware

Parte 4.3

Índice

HDL. Motivación

- HDL (**Hardware Description Language**), Lenguajes de Descripción de Hardware:
  - Un mismo lenguaje para todo el diseño.
  - Favorecen la reutilización de los diseños.
  - Independientes de la tecnología.
  - Descripción con un alto nivel de abstracción que aumenta la productividad de los diseñadores.
  - Simulables desde el principio del diseño.

HDL. Lenguajes (I)

- Primeros pasos: lista de mallas (netlist):
  - EDIF.
- En cuanto aparecieron los primeros ordenadores personales se desarrollaron **herramientas CAD**, todas incompatibles entre sí.
- EDIF permitió la comunicación entre distintas herramientas de distintos fabricantes.
- Sólo puede usarse para descripción de circuitos.
- Casi todas las herramientas lo soportan incluso hoy en día.

Ejemplo EDIF

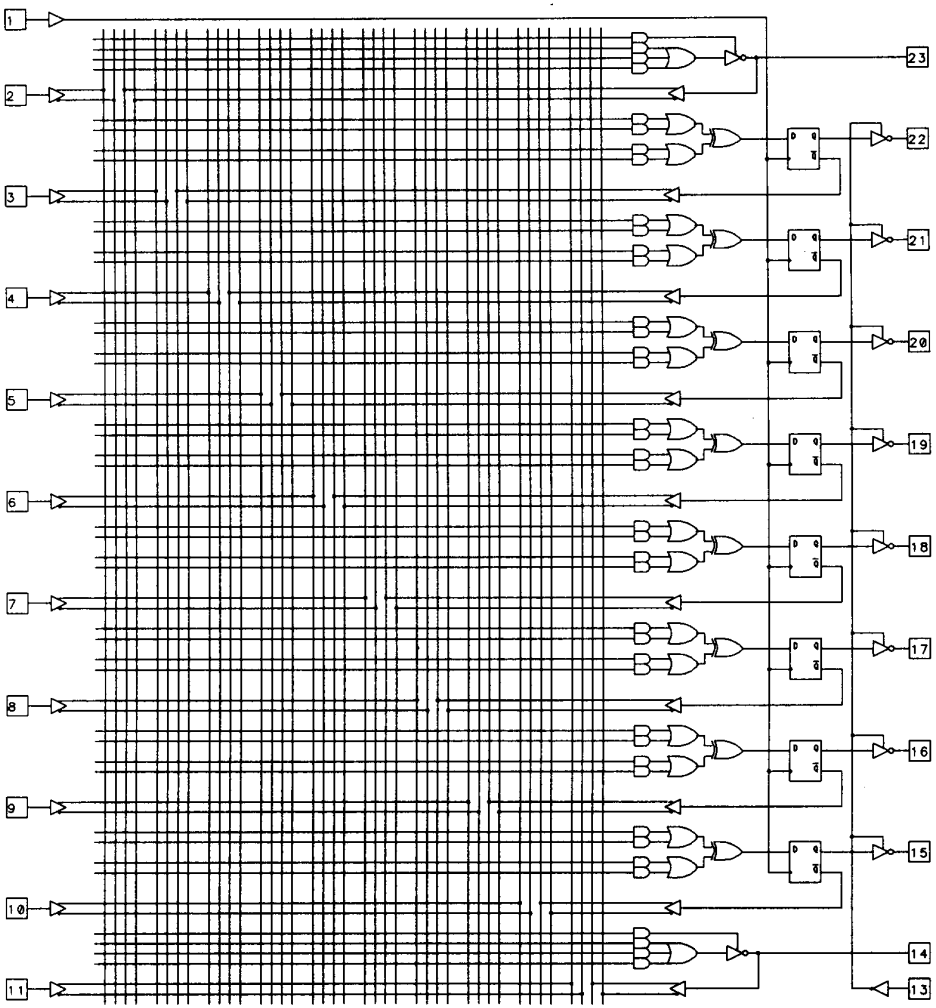
```
(cell NET_TOP_LEFT_RIP
    (cellType RIPPER)
    (view NET_TOP_LEFT_RIP
     (viewType SCHEMATIC)
     (interface
      (port
       (array WIRE 512))
      (port
       (array BUNDLE 512))
      (joined
       (portRef WIRE)
       (portRef BUNDLE))
      (symbol
       (figure
        (figureGroupOverride WIRE
         (fillPattern 1 1
          (boolean
           (boolean
            (false)))))
```

```
(path
          (pointList
           (pt 0 0)
           (pt 10 -10))))
        (portImplementation WIRE
         (connectLocation
          (figure PIN
           (dot
            (pt 0 0)))))
        (portImplementation BUNDLE
         (connectLocation
          (figure PIN
           (dot
            (pt 10 -10)))))))))
```

HDL. Lenguajes (II)

- Aparición de los primeros circuitos lógicos programables (SPLD):
  - Funciones lógicas más complejas, FSM.
  - Necesidad de crear mapas de fusibles.
- Hacerlo a mano era tedioso y proclive a errores.
- Primeros HDL para SPLD: soporte para máquinas de estados, optimización automática…
  - PALASM.
  - ABEL.

Ejemplo: mapa de fusibles (fuse map)




```
ABEL(tm) 3.20 Data I/O Corp.  JEDEC file for: P16R4 V8.0
4 bit counter with synchronous clear
Michael Holley and Dave Pellerin*
QP20* QF2048* QV6* F0*
NOTE Table of pin names and numbers*
NOTE PINS Q3:14 Q2:15 Q1:16 Q0:17 Clk:1 Clear:2 OE:11*
L0512 01111111111111111111111111111111*
L0544 11111111101111111111111111111111*
L0768 01111111111111111111111111111111*
L0800 11111111101110111111111111111111*
L0832 11111111111011101111111111111111*
L1024 01111111111111111111111111111111*
L1056 11111111101110111011111111111111*
L1088 11111111111011111101111111111111*
L1120 11111111111111101110111111111111*
L1280 01111111111111111111111111111111*
L1312 11111111101110111011101111111111*
L1344 11111111111011111111110111111111*
L1376 11111111111111101111111011111111*
L1408 11111111111111111101110111111111*
V0001 C1XXXXXXXXN0XXLLLLXXN*
V0002 C0XXXXXXXXN0XXLLLHXXN*
V0003 C0XXXXXXXXN0XXLLHLXXN*
V0004 C0XXXXXXXXN0XXLLHHXXN*
V0005 C0XXXXXXXXN0XXLHLLXXN*
V0006 C1XXXXXXXXN0XXLLLLXXN*
C337C*
```

Ejemplo: mapa de fusibles (fuse map)

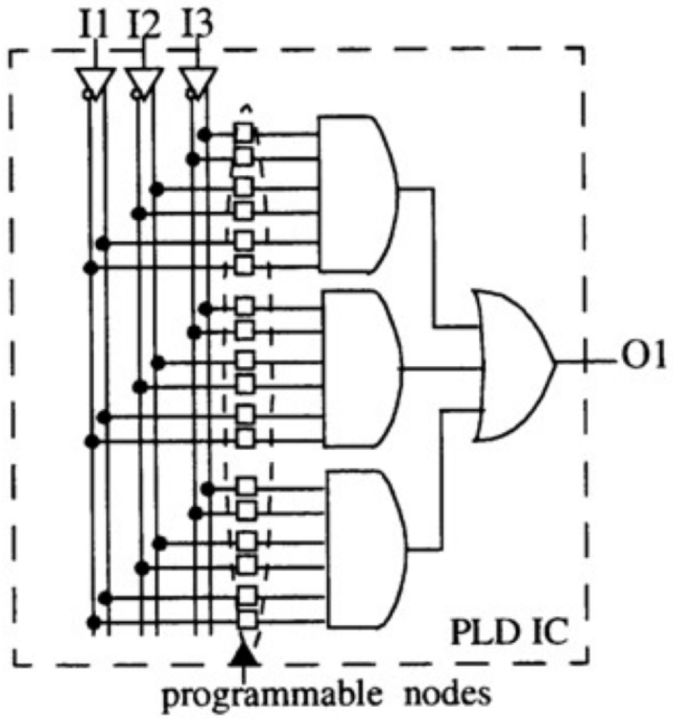


Figure 7.26: A basic example of a programmable logic device. (AND gates are wired-AND).

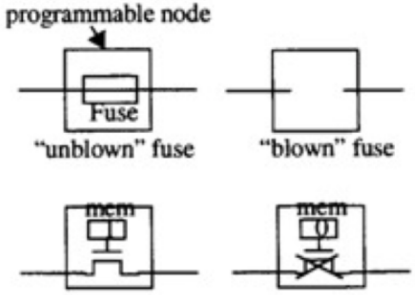


Figure 7.27: Two types of programmable nodes: (top) fuse based, (bottom) memory based.

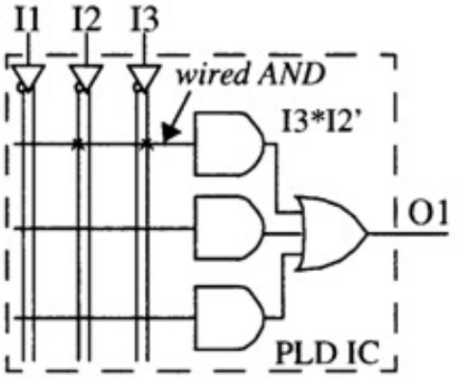


Figure 7.28: Simplified PLD drawing.

Ejemplo: mapa de fusibles (fuse map)

w = k p s'

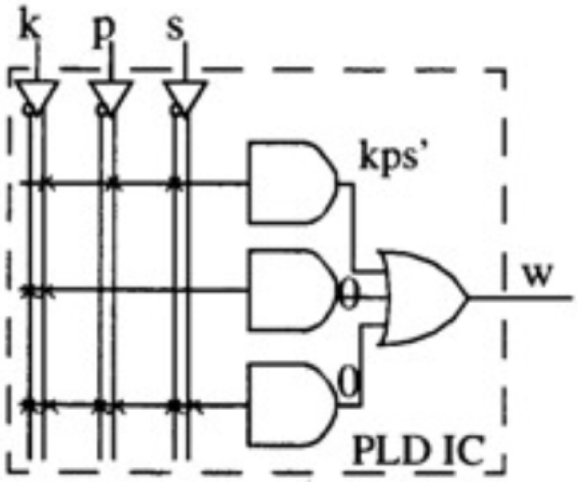


Figure 7.29: Seat-belt warning system on a simple PLD.

Ejemplo: PALASM

```
TITLE video ; shift register
CHIP video PAL20X8
CK /LD D0 D1 D2 D3 D4 D5 D6 D7 CURS GND NC REV Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0
/RST VCC
STRING Load 'LD*/REV*/CURS*RST' ; load data
STRING LoadInv 'LD*REV*/CURS*RST' ; load inverted of data
STRING Shift '/LD*/CURS*/RST' ; shift data from MSB to LSB
EQUATIONS
/Q0 := /D0*Load+D0*LoadInv:+:/Q1*Shift+RST
/Q1 := /D1*Load+D1*LoadInv:+:/Q2*Shift+RST
/Q2 := /D2*Load+D2*LoadInv:+:/Q3*Shift+RST
/Q3 := /D3*Load+D3*LoadInv:+:/Q4*Shift+RST
/Q4 := /D4*Load+D4*LoadInv:+:/Q5*Shift+RST
/Q5 := /D5*Load+D5*LoadInv:+:/Q6*Shift+RST
/Q6 := /D6*Load+D6*LoadInv:+:/Q7*Shift+RST
/Q7 := /D7*Load+D7*LoadInv:+:Shift+RST;
```

HDL. Lenguajes (III)

- La complejidad de los dispositivos y de los diseños siguió aumentando.
- La simulación se hizo imprescindible.
- Lenguajes actuales: descripción con alto nivel de abstracción y simulación:
  - VHDL.
  - Verilog.
- Cuando la potencia de los ordenadores lo permitió se añadió la síntesis.

HDL. Lenguajes (IV)

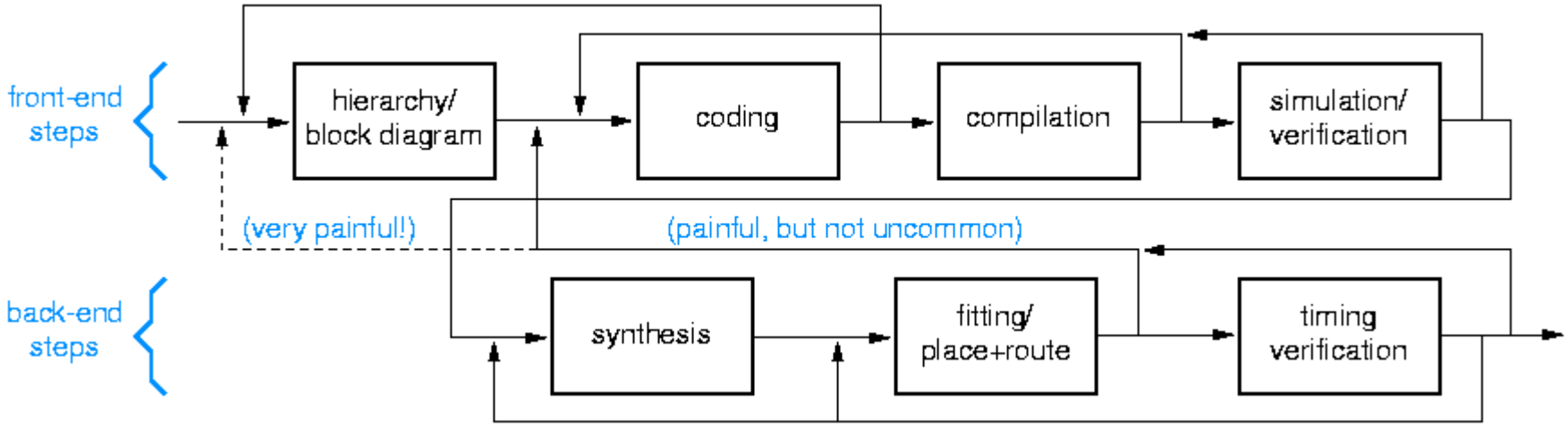
- Hoy en día, una vez que el diseño se ha descrito a nivel de transferencia de registros (RTL), el resto de etapas hasta la creación del mapa de fusibles o el *layout* está automatizado.
- En la actualidad se trabaja para que la programación (scheduling) de los recursos (quién, cuántos y cuándo) se automatice también (mayor nivel de abstracción).

VHDL - Introducción

- Desarrollado a mediados de los 80's por el DoD y el IEEE.
- VHDL es un acrónimo de un acrónimo: "**Very high speed integrated circuit Hardware Description Language**".
- Características:
  - Permite organizar los diseños jerárquicamente.
  - Cada elemento de diseño tiene una interfaz y una especificación de comportamiento.
  - Se puede especificar el comportamiento de forma algorítmica o estructural.
  - Se pueden modelar concurrencia, temporización y los relojes.
  - Se pueden simular la operación lógica y la temporización de un diseño.

VHDL. Flujo de diseño

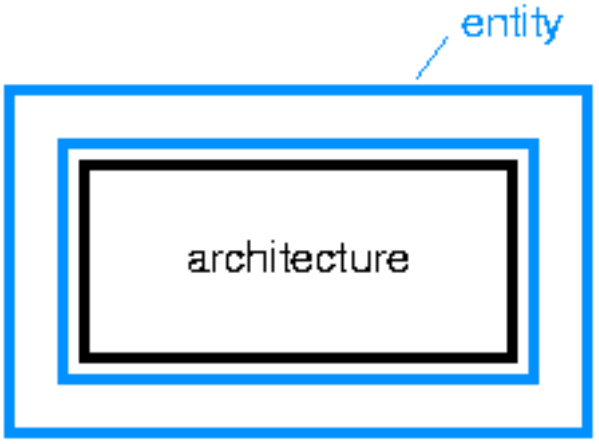
- El VHDL comenzó como un lenguaje para la documentación y el modelado, que permitía especificar y simular el comportamiento de los diseños.
- Ahora también existen herramientas de síntesis. Estas herramientas crean circuitos lógicos directamente de la descripción VHDL.



Recordar: **las herramientas excelentes aún no son un sustituto para una reflexión cuidadosa al principio de un diseño**.

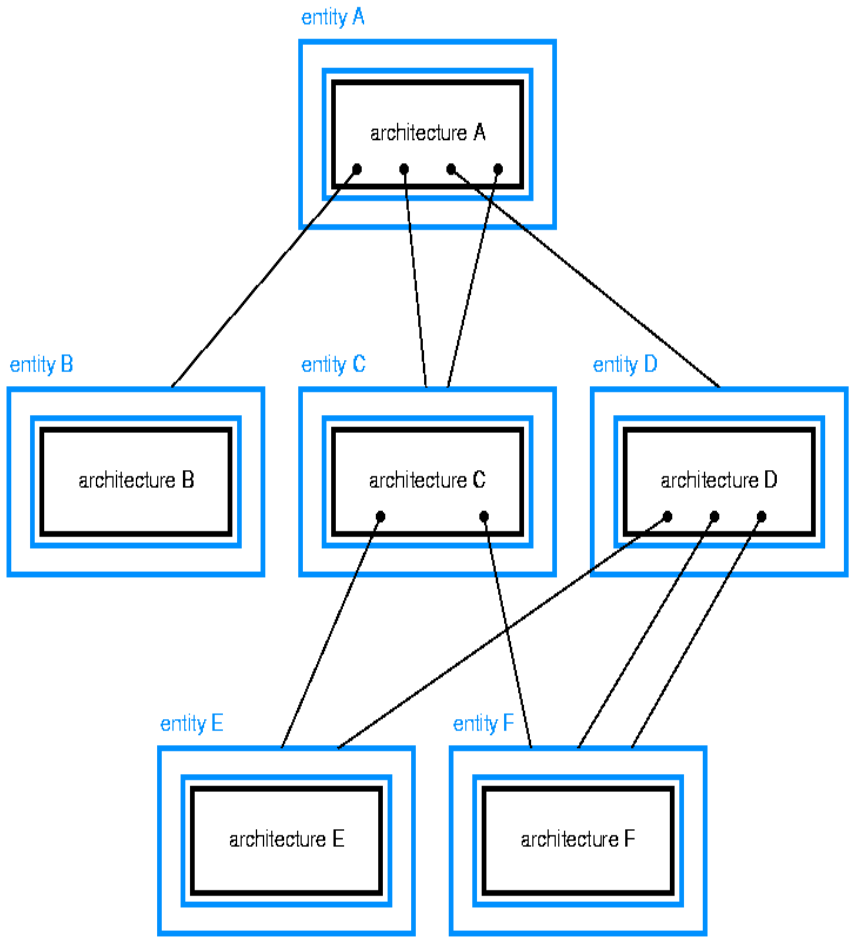
VHDL. Entidades y Arquitecturas (1)

- El VHDL se diseñó según los principios de la programación estructurada.
- Su diseño estuvo influido por los lenguajes Pascal y Ada.
- Un interfaz define los límites de un módulo hardware, a la vez que esconde sus detalles internos.
- En VHDL, una **entidad (entity)** es una declaración de las entradas y salidas de un módulo.
- En VHDL, una **arquitectura (architecture)** es una descripción detallada de la estructura interna o del comportamiento del módulo.



VHDL. Entidades y Arquitecturas (2)

- Dentro de una arquitectura se pueden usar otras entidades.
- Una arquitectura de alto nivel puede usar una entidad de bajo nivel varias veces.
- Distintas arquitecturas de alto nivel pueden usar la misma entidad de bajo nivel.
- Esta es la base que posibilita el diseño jerárquico de un sistema.

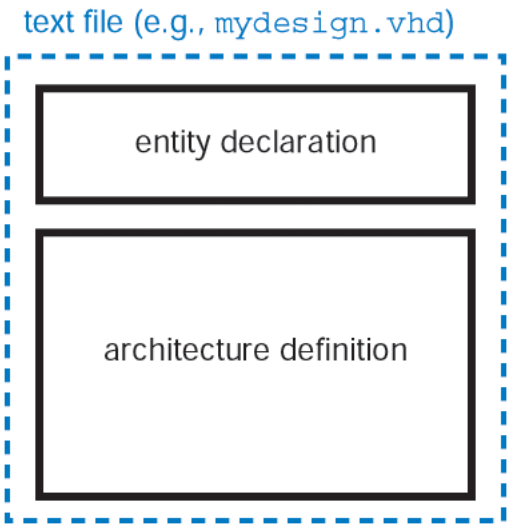


VHDL. Entidades y Arquitecturas (3)

- En el fichero de texto de una descripción VHDL, la declaración de la entidad y la definición de la arquitectura están separadas.

```
entity Inhibit is
  port (X,Y: in BIT;
        Z:   out BIT);
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```



VHDL. Entidades y Arquitecturas (4)

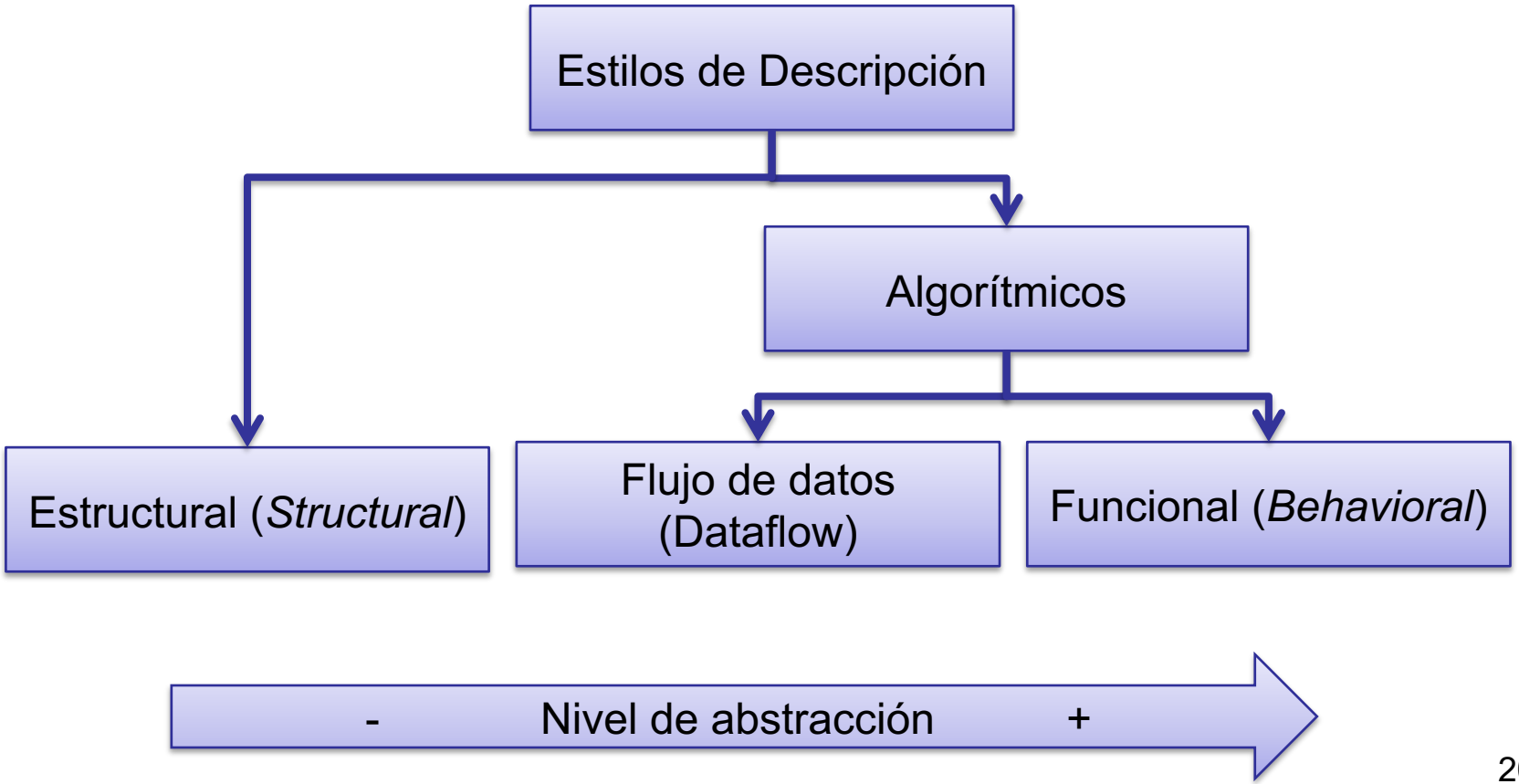
- Cada arquitectura está asociada a una única entidad.
- Una entidad puede tener múltiples arquitecturas:
  - Para simulación (más abstracta y rápida de simular).
  - Para sintetizar (más detallada).
  - Para distintas tecnologías (CMOS, GaAs, FPGA).

Estilos de descripción (I)

- VHDL permite descripciones con varios niveles de abstracción.
- Cada nivel de abstracción se relaciona con un “estilo” de descripción.
- Estos estilos no los impone VHDL y pueden mezclarse si resulta conveniente en un diseño.

Estilos de descripción (II)

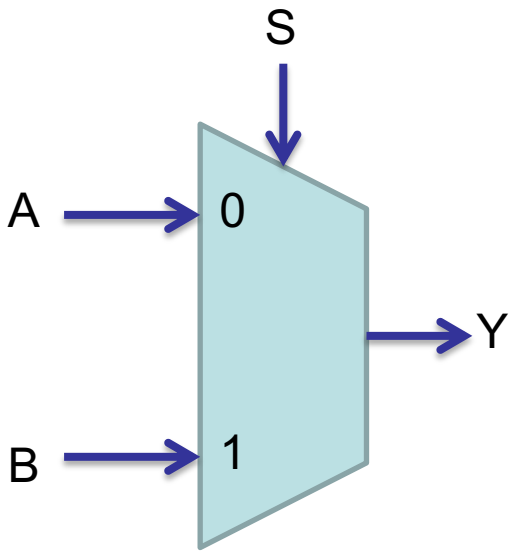
- Existen 3 estilos de descripción:



Ej.: Multiplexor (I)

Veamos un ejemplo con una entidad que representa a un multiplexor

```
entity MUX is
  port (
    A, B: in  std_logic;
    S   : in  std_logic;
    Y   : out std_logic
  );
end entity MUX;
```



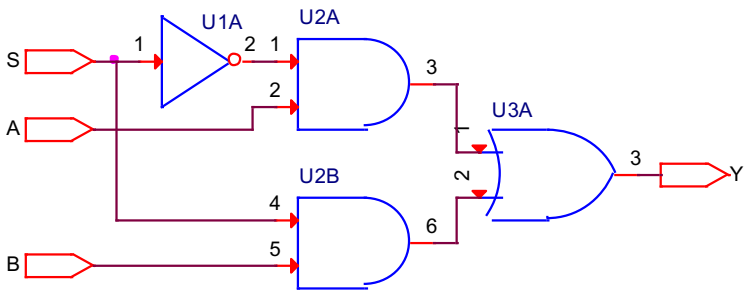
S	Y
0	A
1	B

Estilos de descripción (III)

- Estilo **estructural** (***Structural***):
  - Describe el diseño como una lista interconectada de nodos (*netlist*).
  - Mínimo nivel de abstracción.
  - Necesario para interconectar entre sí los componentes de un diseño.
  - Siempre sintetizable con las herramientas actuales.

Ej.: Multiplexor (II)

```
architecture STRUCTURAL of MUX is
  component INV is
    port (
      A: in std_logic;
      B: out std_logic
    );
  component AND2 is …
  component OR2 is …
  signal NS, A_NS, B_S:std_logic;
begin
  u1a: INV port map(S, NS);
  u2a: AND2 port map(NS, A, A_NS);
  u2b: AND2 port map(S, B, B_S);
  u3a: OR2 port map(A_NS, B_S, Y);
end architecture STRUCTURAL;
```

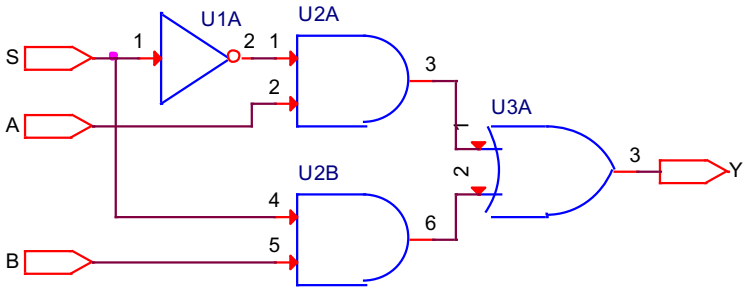


Estilos de descripción (IV)

- Estilo **flujo de datos (*Dataflow*)**:
  - Describe el sistema mediante diagramas de transferencia entre registros, tablas de verdad o ecuaciones booleanas. Los elementos básicos son: registros, memorias, lógica combinacional y *buses*.
  - Nivel de abstracción: transferencia de registros (*Register Transfer Level*).
  - Habitualmente sintetizable con las herramientas actuales.

Ej.: Multiplexor (III)

```
architecture DATAFLOW of MUX is
begin
  Y <= (A and not S) or
       (B and S);
end architecture DATAFLOW;
```




```
architecture DATAFLOW of MUX is
begin
  Y <= A when S = '0' else
       B;
end architecture DATAFLOW;
```

S	Y
0	A
1	B

Estilos de descripción (V)

- Estilo **funcional (*behavioral*)**:
  - Describe el circuito mediante el uso de algoritmos (procesos secuenciales).
  - Máximo nivel de abstracción.
  - Salvo diseños muy simples, es el más productivo y lo emplearemos siempre que podamos.
  - No siempre sintetizable con las herramientas actuales.

Ej.: Multiplexor (IV)

```
architecture BEHAVIORAL of MUX is
begin
  process (A, B, S)
  begin
    if S = '0' then
      Y <= A;
    else
      Y <= B;
    end if;
  end process;
end architecture BEHAVIORAL;
```

S	Y
0	A
1	B

VHDL sintetizable (I)

- El VHDL permite un nivel de abstracción muy alto:
  - Generación de retardos arbitrarios.
  - Bucles.
  - Acceso a ficheros…
- Es posible escribir un **código sintáctica y semánticamente correcto** (se puede simular y los resultados son correctos), **pero que no es sintetizable**.

VHDL sintetizable (II)

- Esto es debido a limitaciones en la capacidad actual de las herramientas de síntesis.
- Se denomina **“VHDL sintetizable”** al subconjunto de VHDL que es **sintetizable por una herramienta**.
- Una parte importante del diseño es refinar la descripción inicial hasta hacerla sintetizable.

Estándares VHDL

- Revisiones:
  - IEEE 1076-1987: primera versión
  - IEEE 1076-1993: mejoras significativas en el lenguaje. Probablemente el más soportado.
  - IEEE 1076-2000: revisión menor.
  - IEEE 1076-2002: revisión menor.
  - IEEE 1076-2008: última revisión. Cambios significativos.

Estándares VHDL

- Estándares relacionados:
  - IEEE 1076.1 VHDL Analog and Mixed-Signal (VHDL-AMS)
  - IEEE 1076.1.1 VHDL-AMS Standard Packages
  - IEEE 1076.2 VHDL Math Package
  - IEEE 1076.3 VHDL Synthesis Package
  - IEEE 1076.3 VHDL Synthesis Package-Floating Point
  - IEEE 1076.4 Timing (VHDL Initiative Towards ASIC Libraries: vital)
  - IEEE 1076.6 VHDL Synthesis Interoperability
  - IEEE 1164 VHDL Multivalue Logic Packages

VHDL: Generalidades

- El VHDL tiene una sintaxis parecida a la de los leguajes de programación SW.
- La mayor diferencia es que sus sentencias se ejecutan de forma concurrente.
- El código se organiza en sentencias acabadas por ‘;’.
- Una sentencia puede ocupar varias líneas siempre que no divida un identificador.

VHDL: Generalidades

- VHDL no distingue entre mayúsculas y minúsculas.
- No obstante, se recomienda ser consistente al usarlas.
- Los espacios al comienzo de cada línea se ignoran.
- Esto permite mejorar la legibilidad del código usando indentación.

VHDL: Comentarios

- Cualquier texto precedido por '--' se considera un comentario y se ignora hasta el final de la línea.

```
-- Asignación datos a bus
BUS <= DATA when OE = '1' else
       "ZZZZ"; -- Triestado
```

Toda la línea es un comentario

Sólo el final de la línea es un comentario

VHDL: Identificadores

- Nombres que damos a los distintos objetos de nuestro código: entidades, arquitecturas, señales, variables…
- Sólo letras, números y ‘_’.
- Deben empezar por letra.
- No más de un ‘_’ seguido.
- No pueden acabar en ‘_’.
- No pueden ser palabras reservadas.

VHDL: Identificadores

- Validos:
  - databus, data_bus, databus0
- Válidos, pero iguales:
  - databus, DataBus
- No válidos:
  - _databus, data__bus, databus_, 3databus

VHDL: Números

- Entero decimal (base 10):
  - 10, 1024, -3
- Entero otras bases:
  - 196 ≡ 2#11000100# ≡ 16#C4#
- Reales:
  - 1.25, -3.14, 5.9e24, -1.6e-19

VHDL: Caracteres y cadenas de caracteres

- Caracteres: letras, números o símbolos encerrados entre comillas simples:
  - ‘a’, ‘@’, ‘&’, ‘5’
- Cadenas de caracteres: secuencias de caracteres encerrados entre comillas dobles:
  - “Esto es una cadena”

VHDL: Cadenas de bits

- Existen tipos específicos para representar los estados de un bit.
- Los estados están representados por una enumeración de caracteres: ‘0’, ‘1’, ‘Z’, etc.
- Para facilitar su utilización se pueden unir formando cadenas de bits:
  - d <= ‘1’; -- asignación 1 bit
  - dbus <= “1100” – asignación 4 bits

VHDL: Cadenas de bits

- Es posible utilizar distintas bases:
  - B”11101001” ≡ ”11101001”
  - O”126”
  - X”FE”
- Ojo: no son enteros, sino cadenas de bits.

VHDL: Palabras reservadas

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

VHDL: operador &

- Operador de concatenación.
- Une dos o más cadenas en una mayor.
- Las cadenas se operan de izquierda a derecha.
- Ejemplos:
  - “1101” & “0011” ≡ “11010011”
  - “Hola ” & “mundo” ≡ “Hola mundo”

VHDL: operadores aritméticos

- Números enteros y reales:
  - Unarios
    - **+**: signo positivo, ej: +4 ≡ 4
    - **-**: signo negativo, ej: -3
    - **abs**: valor absoluto, ej: abs(-2.5) ≡ 2.5
  - Binarios
    - **: exponenciación, ej: 2**4 ≡ 16
    - **+**: suma, ej: 2 + 2 ≡ 4
    - **-**: resta, ej: 2.0 – 2.0 ≡ 0.0
    - *: multiplicación, ej: 3 * 2 ≡ 6
    - */*: división, ej: 25.0 / 2.5 ≡ 10.0

Sólo exponentes enteros

VHDL: Operadores aritméticos

- Números enteros:
  - **mod**: módulo, operación que cumple
    - a mod b = a - b * N
    - sig(a mod b) = sig(b)
    - Ej.: 34 mod -5 ≡ -4
  - **rem**: resto, operación que cumple
    - a rem b = a - b * (a / b)
    - sig(a rem b) = sig(a)
    - Ej.: 34 rem -5 ≡ 4

VHDL: operadores de desplazamiento

- Rotación:
  - **rol**: a la izquierda, ej.: “01101” rol 2 ≡ “01011”
  - **ror**: a la derecha, ej.: “11010” ror 3 ≡ “10110”

- Desplazamientos lógicos:
  - **sll**: a la izquierda, ej.: “01101” sll 2 ≡ “10100”
  - **slr**: a la derecha, ej.: “11010” slr 3 ≡ “00011”

VHDL: operadores de desplazamiento

- Desplazamientos aritméticos:
  - Similares a los desplazamientos lógicos, pero pensados para trabajar sobre números enteros con signo en representación de complemento a dos en lugar de enteros sin signo.
  - Equivale a multiplicar o dividir por 2^n
  - El desplazamiento aritmético hacia la izquierda es exactamente igual al lógico hacia la izquierda
  - El desplazamiento aritmético hacia la derecha es diferente
    - Si el entero con signo es positivo, (con el bit del signo igual a 0), se insertará el bit 0 del signo en el extremo izquierdo al desplazar un bit hacia la derecha (igual que el desplazamiento lógico hacia la derecha),
    - Si es un entero negativo, (con el bit del signo igual a 1), se insertará el bit 1 del bit del signo en el extremo izquierdo
    - **sla**: a la izquierda, ej.: “01101” sla 2 ≡ “10100”
    - **sra**: a la derecha, ej.: “11010” slr 3 ≡ “11111”

VHDL: operadores relacionales

- Devuelven un valor de tipo booleano:
  - **=**: igual a
    - ej.: 2 = 2 ≡ true, 2 = 3 ≡ false
  - **/=**: distinto a
    - ej.: 2 /= 2 ≡ false, 2 /= 3 ≡ true
  - <: menor que
    - ej.: 2 < 3 ≡ true
  - <=: menor o igual que
    - ej.: 4 <= 3 ≡ false
  - >: mayor que
    - ej.: 2 > 3 ≡ false
  - >=: mayor o igual que
    - ej.: 4 >= 3 ≡ true

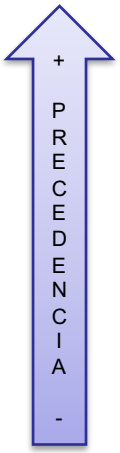
VHDL: operadores relacionales

- Matching Comparison Operators
  - Comparan valores lógicos
  - 1 Matching equality operator (?=)
  - 1 Matching inequality operator (?/=)
  - 1 Matching less than operator (?<)
  - 1 Matching greater than operator (?>)
  - 1 Matching less than or equal to operator (?<=)
  - 1 Matching greater than or equal to operator (?>=)
- Ejemplo:
  - IF ‘H’='1'..." devuelve FALSE
  - IF ‘H’?='1'..." devuelve TRUE (Si H está a nivel alto)

VHDL: operadores lógicos

- Operan sobre valores booleanos y bits.
- También sobre vectores de bits y vectores de valores booleanos (a nivel de bit o *bitwise*).
- **and**: “1101” and “0011” ≡ “0001”
- **nand**: “1101” nand “0011” ≡ “1110”
- **nor**: “1101” nand “0011” ≡ “0000”
- **not**: not “1101” ≡ “0010”
- **or**: “1101” or “0011” ≡ “1111”
- **xnor**: “1101” xnor “0011” ≡ “0001”
- **xor**: “1101” xor “0011” ≡ “1110”

VHDL: precedencia operadores



**	abs	not			
*	/	mod	rem		
+ (signo)	- (signo)				
+	-	&			
=	/=	<	<=	>	>=
and	nand	nor	or	xnor	xor

- Las expresiones se ejecutan de izquierda a derecha.
- Es posible cambiar el orden de ejecución usando paréntesis.

VHDL: tipos de datos

- De forma estricta, VHDL no tiene tipos predefinidos, sino los mecanismos para definir nuevos tipos.
- No obstante existen varios tipos normalizados en la biblioteca estándar:
  - bit, bit_vector, boolean, character, delay_length, file_open_kind, file_open_status, integer, natural, positive, real, severity_level, string, time

VHDL: tipos escalares (I)

- Enteros:
  - **type** integer **is range** -2**31 **to** 2**31 – 1;
- Enumerados:
  - **type** bit **is** ('0', '1');
  - **type** boolean **is** (false, true);
  - **type** color **is** (red, green, blue);
- Reales:
  - **type** real **is range** -1.0e38 **to** 1.0e38;

VHDL: tipos escalares (II)

- Físicos:

```
type time is range 0 to 1e19
    units
        fs; -- femtosecond
        ps = 1000 fs; -- picosecond
        ns = 1000 ps; -- nanosecond
        us = 1000 ns; -- microsecond
        ms = 1000 us; -- millisecond
        sec = 1000 ms; -- second
        min = 60 sec; -- minute
        hr = 60 min; -- hour
    end units;
```

VHDL: tipos compuestos (I)

- Matrices
  - Unidimensionales o multidimensionales.
  - Admiten rangos como índices.
  - **El índice puede quedar indefinido hasta la creación del objeto**.
  - Ejemplos:
    - **type** word **is array**(15 **downto** 0) of bit;
    - **type** bit_vector **is array** (natural **range** <>) **of** bit;
    - **type** string **is array** (positive **range** <>) **of** character;
    - **type** mat3x3 **is array**(1 **to** 3, 1 **to** 3) of real;
  - Uso:

```
signal data: bit_vector(7 downto 0);
q_n <= not data(3);
nibble <= data(3 downto 0);
```

VHDL: tipos compuestos (II)

- Agregados
  - Lista separada por comas encerrada entre paréntesis que se asigna por orden a una matriz.
  - Puede usarse **others** para asignar aquellos bits que no se hayan asignado hasta ese momento.
  - Uso:

```
data_bus <= (byte_high, byte_low);
signal reg: bit_vector(3 downto 0);
reg <= ('0', '1', others => 'Z'); ≡ "01ZZ"
```

VHDL: tipos compuestos (III)

- Registros
  - Semejantes a las estructuras de C/C++.
  - Permiten agrupar distintos tipos de datos en un solo objeto.
  - Uso:

```
type student is
        record
                first_name: string;
                course: integer;
        end record;
```

  - Acceso mediante '.':

```
variable person: student;
person.first_name := "John";
```

VHDL: subtipos

- Permiten acotar los valores de un tipo escalar o el rango de una matriz.
- Síntesis: permiten ahorrar recursos.
- Ejemplos:

```
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;
subtype byte is bit_vector(7 downto 0);
```

VHDL: atributos (I)

- Permiten añadir información adicional a casi cualquier objeto en VHDL: señales, variables, entidades, arquitecturas, tipos, subtipos, procedimientos, funciones…
- Existe un buen número predefinido y se pueden definir más.
- Algunas herramientas de síntesis los emplean para pasar meta-información: optimizaciones, encapsulados, etc.

VHDL: atributos (II)

- Para acceder al atributo de un objeto se usa el apóstrofe:

```
reg := databus'left
```

- Convenciones en el listado de atributos:
  - T representa cualquier tipo.
  - A representa cualquier matriz.
  - S representa cualquier señal.
  - E representa una entidad con nombre.

VHDL: atributos (III)

- T‘**base**: tipo base de T.
- T‘**image**(X): representación literal (string) del valor X del tipo T.
  - 25 -> “25”
- T‘**value**(X): convierte la cadena X en el valor correspondiente de tipo T.
  - “25” -> 25

VHDL: atributos (IV)

- T‘**left**: valor más a la izquierda de T.
- T‘**right**: valor más a la derecha de T.
- T‘**high**: valor más alto de T.
- T‘**low**: valor más bajo de T.
- T‘**ascending**: *true* si T definido con **to**.

VHDL: atributos (V)

- T‘**pos**(X) posición de X en el tipo discreto T.
- T'**val**(X) valor tipo discreto T en la posición X.
- T‘**succ**(X) valor del tipo discreto T que sucede al valor en la posición X.
- T‘**pred**(X) valor del tipo discreto T que precede al valor en la posición X.
- T'**leftof**(X) valor del tipo discreto T a la izquierda de X.
- T'**rightof**(X) valor del tipo discreto T a la derecha de X.

VHDL: atributos (VI)

- A'**left**[(N)] índice más a la izquierda de A o de la dimensión N de A.
- A'**right**[(N)] índice más a la derecha de A o de la dimensión N de A.
- A'**high**[(N)] índice más alto de A o de la dimensión N de A.
- A'**low**[(N)] índice más alto de A o de la dimensión N de A.

VHDL: atributos (VII)

- A'**range**[(N)]: rango de índices de A o de la dimensión N de A.
- A'**reverse_range**[(N)]: rango inverso de índices de A o de la dimensión N de A.
- A'**length**[(N)]: número de elemento de A o en la dimensión N de A.
- A'**ascending**[(N)]: *true* si el rango de A o de la dimensión N de A se definió con **to**.

VHDL: atributos (VIII)

- S'**delayed**(t): valor de S en **now** - t .
- S'**stable**: *true* si no está ocurriendo ningún *suceso* (cambio de valor) en S en este ciclo de simulación.
- S'**stable**(t): *true* si no ha ocurrido ningún *suceso* en S desde hace t unidades de tiempo.
- S'**quiet**: *true* si no ha habido asignaciones a S en este ciclo de simulación.
- S'**quiet**(t) *true* si no ha habido asignaciones a S desde hace t unidades de tiempo.

VHDL: atributos (IX)

- S'**transaction**: bit, conmuta su valor en cada ciclo en que S está activa.
- S'**event**: *true* si ha habido un *suceso* en S en el ciclo de simulación actual.
- S'**active**: *true* si S está *activa* (ha habido asignación a S) durante el ciclo de simulación actual.
- S'**last_event**: tiempo transcurrido desde el último suceso en S.
- S'**last_active**: tiempo transcurrido desde la última vez que S estuvo activa.
- S'**last_value**: valor anterior de S.

VHDL: atributos (X)

- S'**driving**: *false* sólo si el valor que gobierna S es una transacción nula.
- S'**driving_value**: valor actual que gobierna S.

VHDL: atributos (XI)

- E'**simple_name**: cadena con el nombre de E.
- E'**instance_name**: cadena conteniendo la jerarquía de diseño incluyendo E.
- E'**path_name**: cadena conteniendo la jerarquía de diseño de E hasta la raíz del diseño.

VHDL: atributos definidos por usuario (XII)

- Se pueden definir nuevos atributos:

  attribute nombre : tipo;

- Para asignar el atributo a un objeto:

  attribute nombre **of** objeto: tipo_objeto **is** valor;

- Ej:

  signal control: std_logic;

  attribute pin: integer;

  attribute pin **of** control: **signal is** 5;

VHDL: entidad

- **Define la interfaz del diseño:**

entity name **is**
    [**generic**(…);]
    [**port**(…);]
[entity_declarative_part]
[**begin**
    entity_statement_part]
end [**entity**] [name];

VHDL: genéricos

- **Parametrización del diseño.**
- **Facilitan la reutilización del código.**
- **Son visibles para la entidad y todas sus arquitecturas como constantes.**
- **Sintaxis:**

  $\{nombre,\}_1$: tipo [:= valor_por_defecto];

VHDL: puertos (I)

- **Interacción con el resto del diseño.**
- **Son visibles para la entidad y todas sus arquitecturas.**
- **Sintaxis:**

  {nombre,}$_1$: **in**|**out**|**inout**|**buffer**|**linkage** tipo;

  – **in**: entrada, sólo lectura.
  – **out**: salida, sólo asignación.
  – **inout**: entrada/salida, lectura y asignación.
  – **buffer**: salida, lectura y asignación.
  – **linkage**: enlazado con otros lenguajes.

Ejemplo entidad

```
entity REG_D is
  generic(WIDTH : positive := 8);
  port(
    RESET, CLK: in std_logic;
    D: in std_logic_vector(width – 1 downto 0);
    Q: out std_logic_vector(width – 1 downto 0)
  );
constant T_SETUP : time := 10 ns;
begin
  process
  begin
    wait until clk = '1'
    assert D'stable(T_SETUP)
      report "Setup violation"
      severity failure;
  end process;
end entity REG_D;
```

VHDL: puertos (II)

- **Para patillas bidireccionales, incluyendo triestado, se usa *inout*.**
- **No debe usarse *inout* si la patilla no es bidireccional.**
- **El uso de *buffer* no es frecuente y está desaconsejado por algunos fabricantes.**
- **Si es necesario leer una salida es mejor definir una señal interna, trabajar con ella, y, finalmente, asignarla al puerto de salida.**

VHDL: arquitectura (I)

- Define la funcionalidad del diseño.

architecture name **of** entity_name **is**
[architecture_declarative_part]
begin
    architecture_statement_part
end [**architecture**] [name];

VHDL: arquitectura (II)

- Las sentencias dentro de una arquitectura se ejecutan de forma concurrente (en paralelo).
- En simulación, no hay garantía del orden de ejecución, aunque sí de causalidad.
- El funcionamiento del diseño no debe depender del orden de ejecución.

VHDL: constantes

- Constantes:
  - Una vez inicializadas no pueden cambiarse.
  - Mejoran la legibilidad del código.
  - Facilitan el mantenimiento del código.
- Sintaxis:

  constant nombre : tipo := valor;
- Ej.:

  constant PI: real := 3.14159265;

VHDL: variables

- Variables:
  - Concepto semejante a otros lenguajes.
  - Pueden leerse, escribirse y tener un valor inicial.
  - La asignación de valor tiene efecto inmediato.
  - Declaración/uso sólo en procesos secuenciales.
- Sintaxis:

  variable nombre : tipo [:= valor_inicial];
- Ej.:

  variable contador : natural := 0;

  variable buffer : bit_vector(31 **downto** 0);

VHDL: señales (I)

- Señales:
  - Representan líneas reales del circuito.
  - Pueden leerse, escribirse y tener un valor inicial.
  - Funcionan de forma semejante a una FIFO.
  - La asignación no tiene efecto inmediato.
- Sintaxis:

  signal nombre : tipo [**bus** | **register**] [:= valor_inicial];
- Ej.:

  signal acumulador : bit_vector(3 **downto** 0) = “1011”;

  signal clock : std_logic;

VHDL: señales (II)

- Las señales pueden ser de 3 clases:
  - Normales (): no se pueden “desconectar”.
  - Bus (**bus**): pueden desconectarse y entonces toman un valor por defecto.
  - Register (**register**): pueden desconectarse y entonces mantienen su último valor.
- Ej.:

  signal data_bus : bit_vector(31 **downto** 0) **register**;

VHDL: alias (I)

- Permiten declarar otro nombre para un identificador ya declarado:
  - **alias** nombre2 **is** nombre1;
  - **alias** nombre2 [: indicación_subtipo] : **is** [firma];
- *nombre2* puede ser un identificador, un caracter literal o el símbolo de un operador.
- Permite:
  - Dar localmente otro nombre a una señal.
  - Acceder a partes de un dato por nombre.
  - “Sobrecargar” operadores para tipos no estándar.
  - Acortar rutas complejas.

VHDL: alias (II)

- Ej. de alias:

```
alias rs is my_reset_signal ; -- No recomendado

signal real32 : std_logic_vector(31 downto 0);
alias mantissa : std_logic_vector(23 downto 0) is  real32(8 to 31);
alias exponent is real32(0 to 7);

alias "<" is my_compare [ my_type, my_type, return boolean ] ;

alias 'H' is STD.standard.bit.'1' [ return bit ] ;
```