Universidad de Alcalá

Departamento de Electrónica

Modelado de Sistemas Computacionales

Grado en Ingeniería de Computadores

Tema 2: Lenguajes de descripción hardware. El VHDL

HDLs

- Los lenguajes de descripción Hardware (HDLs) son "*similares*" a los lenguajes de descripción software (C- Verilog, Ada-VHDL).
- Son lenguajes formal para especificar el comportamiento y estructura de un circuito digital.
- Los HDL`s son lenguajes de alto nivel que permiten definir (modelar) ***hardware***.
- Se utilizan para ***modelar***, no ***programar***.
- Son lenguajes que se utiliza tanto para síntesis de circuitos digitales como para su simulación:
  - Síntesis: el lenguaje describe el comportamiento o la estructura deseada del ***circuito digital*** que debe inferir una herramienta.
  - Simulación: el lenguaje permite crear el entorno donde el ***circuito digital*** está integrado de tal forma que permite generar estímulos que se aplican al modelo descrito para verificar su funcionamiento.

HDLs

- Características más relevantes:
  - Se puede usar durante todo el ciclo de diseño del circuito/sistema incluyendo la especificación (modelado) y simulación del mismo.
  - Soporta diferentes niveles de abstracción (nivel de detalle): comportamiento o funcional, RTL, flujo de datos y estructural.
  - Intrínsecamente soporta paralelismo ya que incluye construcciones que permiten modelar concurrencia (los modelos HW son concurrentes por naturaleza) y un modelo de ejecución, que para simulación, proporciona resultados deterministas.
  - Soporta programación secuencial para descripciones algorítmicas usando construcciones similares a las que incluyen los lenguajes de programación del SW (de propósito general).

UAH
Departamento de Electrónica

HDLs

- Comparativa entre los HDLs y los lenguajes de propósito general.

	HDLs	**Lenguajes SW**
Propósito	Hardware	Programas
Entrada	Editor de texto	
Desarrollo	Compilar para la simulación y sintetizar	Compilar y linkar.
Depuración	Simulación y vista de forma de ondas.	Ejecutar y ver resultados (texto o gráficos).

HDLs

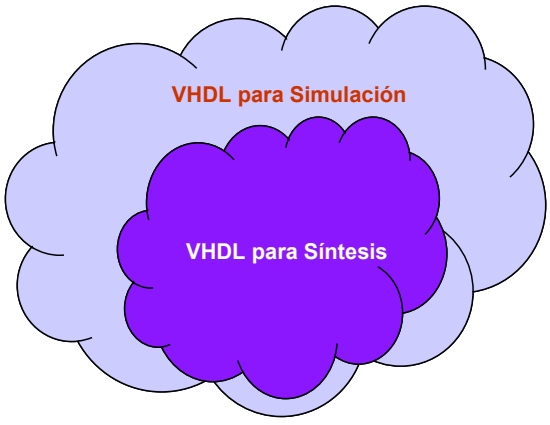
❑ Comparativa entre los HDLs y los lenguajes de propósito general.

	HDLs	**Lenguajes SW**
Instrucciones	Incluye construcciones secuenciales y concurrentes	Incluyen solo construcciones secuenciales
Estilos de descripción	Register Transfer Level (RTL). Es imprescindible tener un buen conocimiento de los circuitos digitales.	La descripción es siempre a nivel de comportamiento que para codificarlo se utiliza algoritmia y analítica.
Uso de recursos	El diseñador siempre tiene en mente el **área**, la **velocidad** y el **consumo**.	Normalmente el programador no tiene en cuenta el uso de recursos como memoria (área) y la velocidad del programa.
Aplicación	Se utiliza para describir circuitos digitales. El usuario es **diseñador**.	Se utiliza como lenguaje de programación para describir funcionalidad. Es usuario es **programador**.

El VHDL

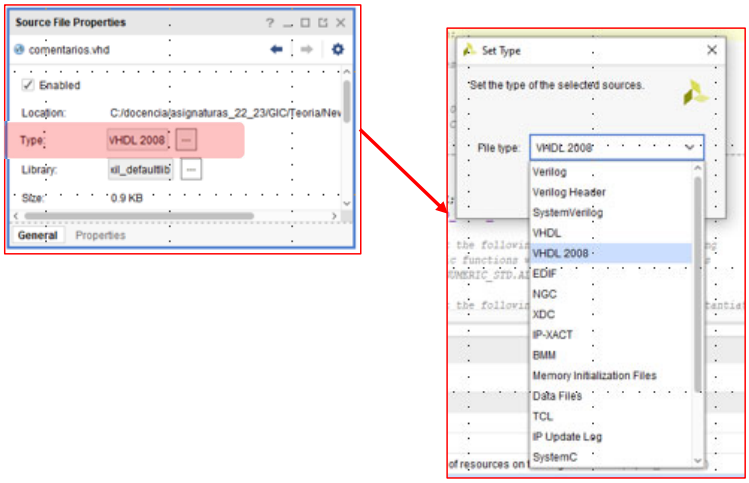
❑ VHDL: (**V**HSIC -**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit- **H**ardware **D**escription **L**anguage) fue promovido en 1982 por el DoD, para la especificación y documentación de los sistemas electrónicos digitales.

1981	Initiated by US DoD to address hardware life-cycle crisis
1983-85	Development of baseline language by Intermetrics, IBM and TI
1986	All rights transferred to IEEE
1987	Publication of IEEE Standard
1987	Mil Std 454 requires comprehensive VHDL descriptions to be delivered with ASICs
1994	**Revised standard (named VHDL 1076-1993)**
2000	Revised standard (named VHDL 1076 2000, Edition)
2002	Revised standard (named VHDL 1076-2002)
2007	VHDL Procedural Language Application Interface standard (VHDL 1076c-2007)
2009	**Revised Standard (named VHDL 1076-2008)**
2019	https://vhdlwhiz.com/vhdl-2019/

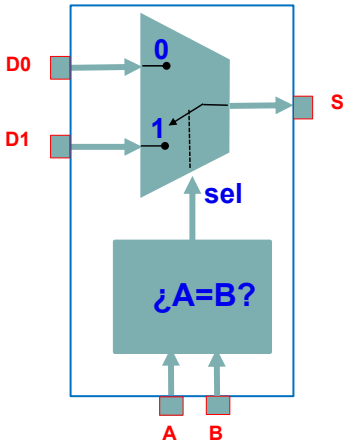


VHDL-08. aunque fue publicada hace ya bastantes años, muchas de las "mejoras" todavía están en fase de implementación por las herramientas de diseño y otras no se incluirán porque no se consideran necesarias. Constituye la versión del estándar con más cambios desde 1993.

- Consideraciones para trabajar con VHDL-08.
- Por defecto, los sistemas de desarrollo consideran que se trabaja con el estándar del 93 (VHDL-93)
- Para cada archivo se debe seleccionar el estándar del 2008 (VHDL-08)
  - Para Vivado.



- Estructura de un archivo VHDL.



sel	S
0	D0
1	D1

mux2_1.vhd

Bibliotecas

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

Entidad

```
entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;
```

Arquitectura

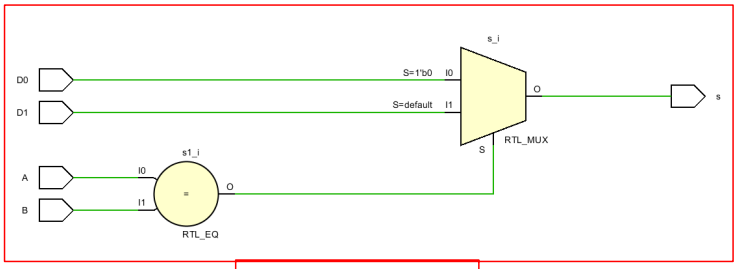
Declaración

```
architecture RTL of mux2_1 is
  signal sel : std_logic;
begin
```

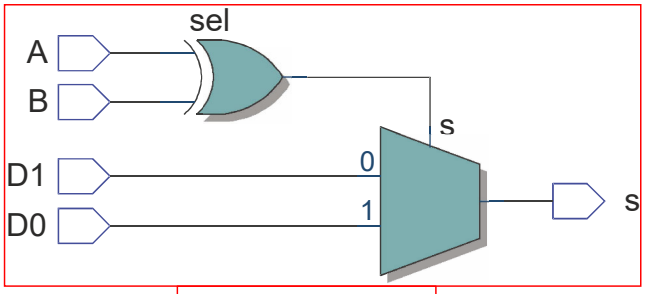
Cuerpo

```
sel <= '1' when A = B else  '0';

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end RTL;
```

❑ Síntesis del archivo VHDL.



Vivado (AMD-Xilinx)

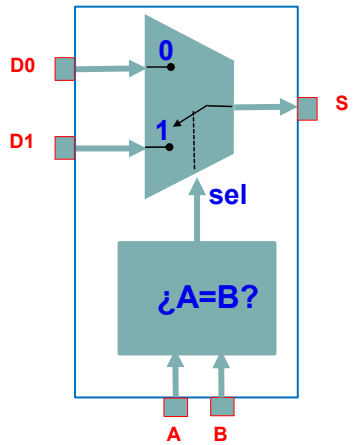


Quartus (Intel_Altera)

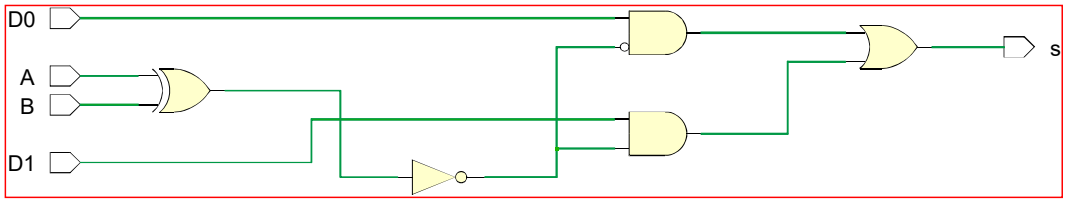
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

sel <= '1' when A = B else  '0';

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end RTL;
```

Dependiendo de la tecnología destino (FPGA o CPLD) y la herramienta de síntesis, la implementación del archivo VHDL se puede realizar de formas diferentes.

❑ Otra forma de modelar el circuito (nivel de abstracción).




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

entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

 sel <=  not(A xor B) ;

 S <= (D0 and (not sel)) or (D1 and sel);

end RTL;
```



❑ Estilos descriptivos.

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

entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

sel <= '1' when A = B else  '0';

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end RTL;
```

Estilo descriptivo Algorítmico

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

entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

 sel <=  not(A xor B) ;

 S <= (D0 and (not sel)) or (D1 and sel);

end RTL;
```

Estilo descriptivo Fujo de Datos

Hay un tercer Estilo descriptivo llamado Estructural que se verá más adelante

UAH
Departamento de Electrónica
El VHDL

❑ Estilos descriptivos.

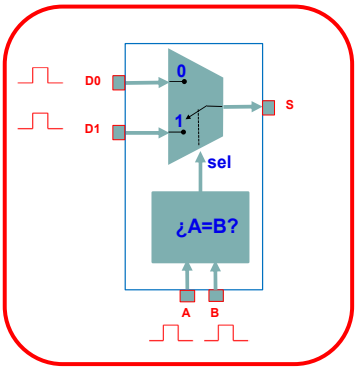
¡Los estilos se pueden y **se deben** mezclar!

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

entity mux2_1c is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1c;

architecture RTL of mux2_1c is
  signal sel : std_logic;
begin

sel <=  not(A xor B) ;

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end RTL;
```

UAH

Departamento de Electrónica

❑ Simulación del archivo VHDL.



mux2_1_tb.vhd

Bibliotecas

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

Entidad

```
entity mux2_1_tb is
end mux2_1_tb;
```

Arquitectura

Declaración

```
architecture sim of mux2_1_tb is
  signal A, B, D0, D1 : std_logic ;
  signal S            : std_logic;

begin  -- sim
```

Cuerpo

Generación de estímulos

```
D0 <= not D0 after 50 ns;
D1 <= not D1 after 10 ns;

A <= '0' ,'1' after 400 ns, '0' after 1200 ns;
B <= '0' ,'1' after 800 ns, '0' after 1600 ns;
```

Instanciación del componente.

```
DUT :entity work.mux2_1
  port map (
    A  => A,
    B  => B,
    D0 => D0,
    D1 => D1,
    s  => S);
end sim;
```

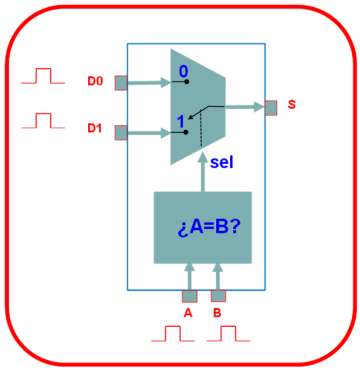
¡Este archivo es un *test bench* o banco de pruebas!

¡Este archivo **no se sintetiza**, es sólo para la simulación!

UAH

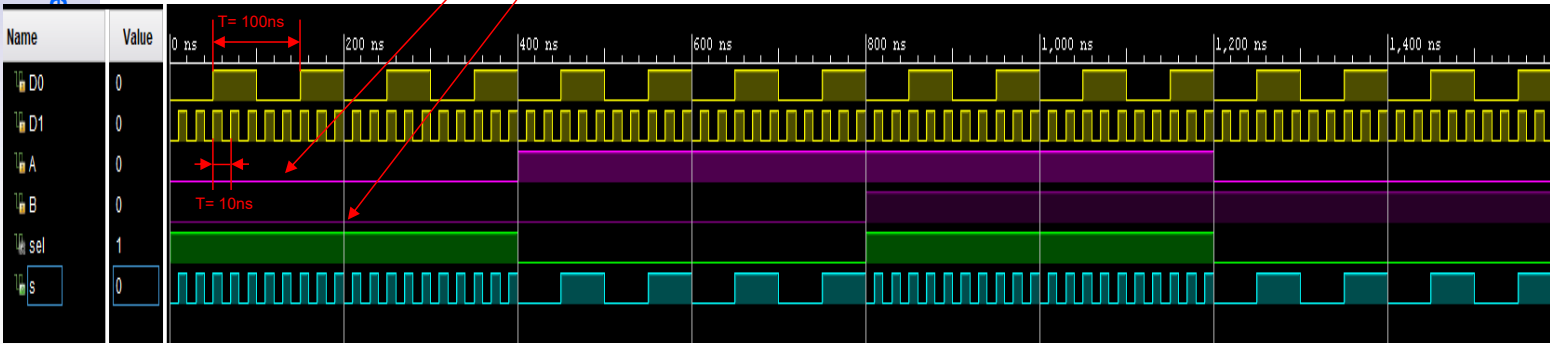
ectrónica

❑ Simulación funcional del archivo VHDL.

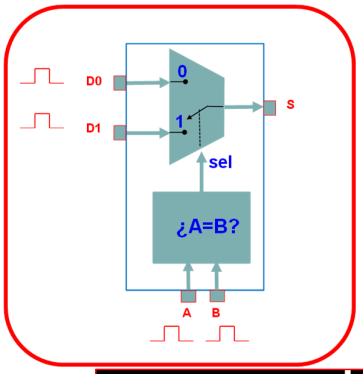



```
D0 <= not D0_i after 50 ns;
D1 <= not D1_i after 10 ns;

A <= '0', '1' after 400 ns, '0' after 1200 ns;
B <= '0', '1' after 800 ns, '0' after 1600 ns;
```

Simulación Funcional Es una simulación ideal en la que no existen retardos

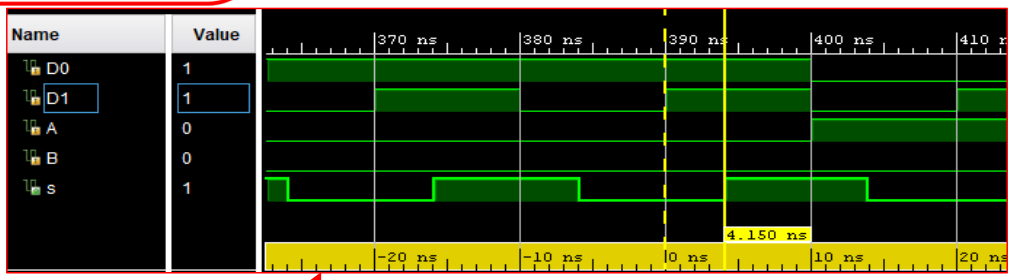


Simulación del archivo VHDL.



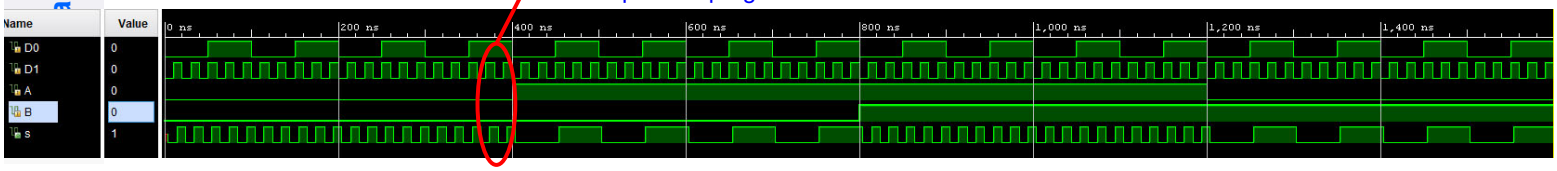
Se utiliza el mismo banco de pruebas

```
D0 <= not D0_i after 50 ns;
D1 <= not D1_i after 10 ns;

A <= '0', '1' after 400 ns, '0' after 1200 ns;
B <= '0', '1' after 800 ns, '0' after 1600 ns;
```



Simulación Temporal

Es una simulación más realista al tenerse en cuenta los retardo de internos des dispositivo programable



Identificadores.

- Se utilizan para asignar un nombre a un elemento de VHDL.
- No existen restricciones en cuanto al máximo número de caracteres a emplear.
  - Válidos los caracteres:
    - Alfabéticos (a-z, A-Z) (no ñ si w)
    - numéricos (0-9)
    - Subrayado (_).
- No se hace distinción entre mayúsculas y minúsculas (*case insensitive*).
- Restricciones:
  - Deben empezar por un carácter alfabético
  - No pueden tener dos subrayados seguidos
  - No pueden acabar en subrayado.
  - No se pueden utilizar las palabras clave

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

entity mux2_1 is
  port ( A, B, D0, D1 : in  std_logic;
         s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

sel <= '1' when A = B else  '0';

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end RTL;
```

PALABRAS RESERVADAS

- Palabras clave.

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

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

entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

sel <= '1' when A = B else  '0';

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end RTL;
```


SÍMBOLOS ESPECIALES

- Símbolo especiales.
  - Pueden ser operadores.
  - Pueden formar parte de sentencias.
  - Un carácter:

    & ' ( ) * + , - / : ; < > = | \ #
  - Dos caracteres:

    => ** := /= >= <= <> -- /* */
  - Todas las sentencias terminan en ;
  - Los comentarios van precedidos por --

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

entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

sel <= '1' when A = B else  '0';

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end RTL;
```

SÍMBOLOS ESPECIALES

- Comentarios.

```
-- Company:
-- Engineer:
--
-- Create Date: 03.11.2022 23:58:54
-- Design Name:
-- Module Name: comentarios - rtl
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

VHDL-93

```
/*Company:
Engineer:

Create Date: 03.11.2022 23:58:54
Design Name:
Module Name: comentarios - rtl
Project Name:
Target Devices:
Tool Versions:
Description:

Dependencies:

Revision:
Revision 0.01 - File Created
Additional Comments:
*/
```

VHDL-08

VHDL2008 permite comentar bloques completos de código con /* y */.

Estructura de un archivo VHDL

Bibliotecas

```
library nombre_librería
use nombre_librería.nombre_paquete.all
```

Entidad

```
entity nombre_entidad  is
        Listado_puertos
        -- Declaración de pines
end nombre_entidad;
```

Arquitectura

```
architecture nombre_arquitectura of  nombre_entidad
-- Declaración de señales internas
-- Declaración de componentes en caso de instanciación
begin
-- Cuerpo de la arquitectura:
        -- Se define la funcionalidad del diseño con:
                -- asignaciones concurrentes
                -- procesos
                -- instanciación de componentes
end nombre_arquitectura;
```

Bibliotecas

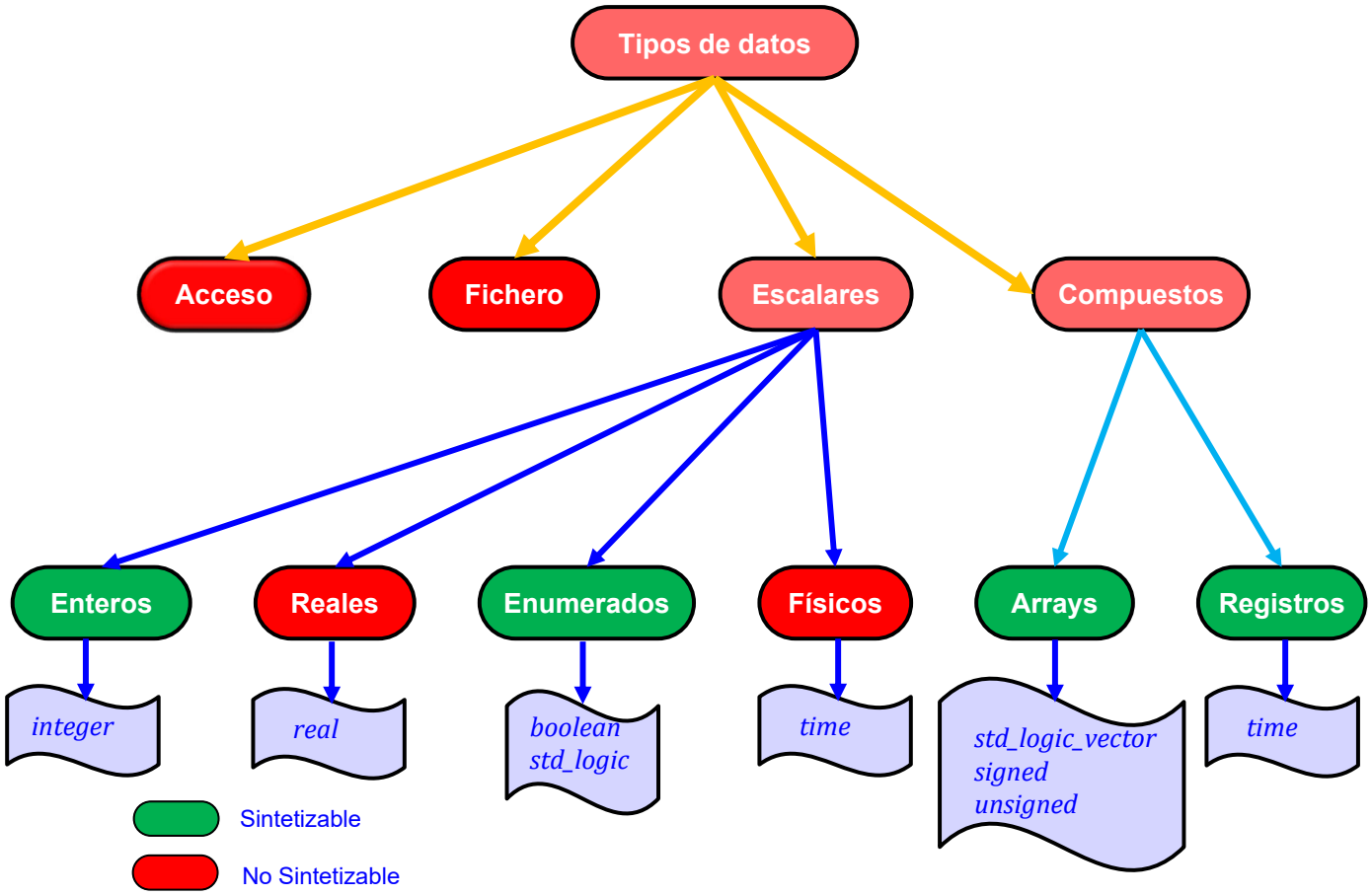
- Contienen las unidades de diseño (entidades, arquitecturas, configuraciones y paquetes) ya analizadas (*compiladas y elaboradas*).
- Se acceden mediante su nombre lógico.
- Para dar visibilidad a una biblioteca se usa la sentencia ***library***.
- Librerías utilizadas.
  - **work**. Es la librería donde se almacenan por defecto las unidades de diseño una vez compiladas. *Es siempre visible.*
  - **std**. Contiene los paquetes *standard* y *textio*. *Es siempre visible.*
  - **ieee**. Contiene paquetes para tipos *std_logic*, *unsigned/signed,* paquetes aritméticos, etc. Las proporciona la propia herramienta de desarrollo. *No es visible.*

Bibliotecas/Librerías

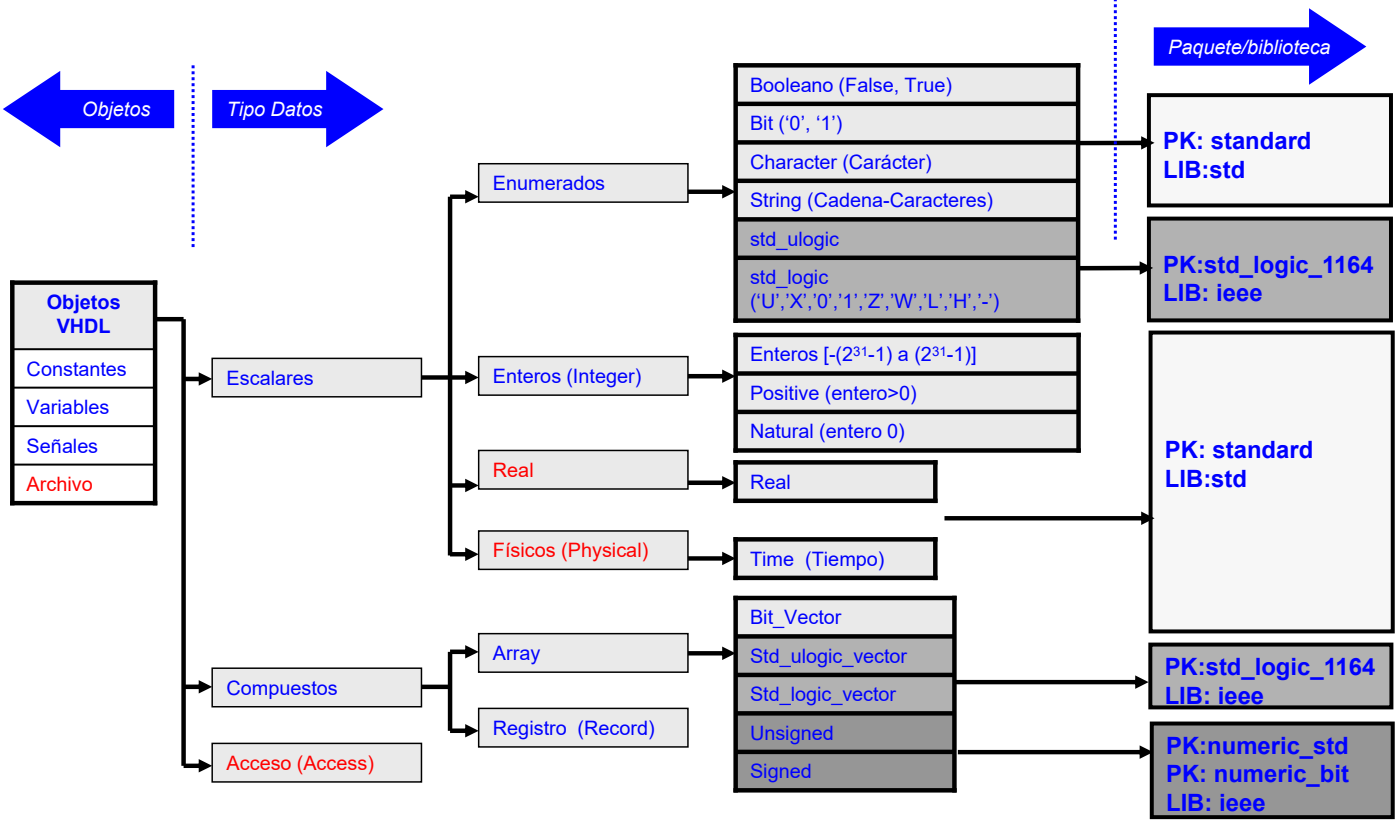
- Paquetes mas utilizados.

Librería	Paquete	Descripción
std	standard	Define los tipos y subtipos básicos de VHDL: ***boolean***, ***bit***, ***bit_vector***, ***character***, ***string***, ***integer***, ***real***, ***time***, …
std	textio	Define tipo de datos como ***line*** y ***text*** y procedimientos de lectura y escritura en archivos
ieee	Std_logic_1164	Define los tipos ***std_ulogic***, ***std_logic***, ***std_ulogic_vector*** y ***std_logic_vector.***
ieee	Numeric_std	Define los tipos ***signed*** y ***unsigned*** y los operandos aritméticos para ellos
ieee	Std_logic_arith	Define los tipos ***signed*** y ***unsigned*** y los operandos aritméticos para ellos
ieee	Std_logic_signed	Define operaciones aritméticas con signo, operaciones de conversión y comparación para ***std_ulogic_vector*** y ***std_logic_vector***
ieee	Std_logic_unsigned	Define operaciones aritméticas sin signo, operaciones de conversión y comparación para ***std_ulogic_vector*** y ***std_logic_vector***
ieee	Std_logic_textio	Define tipos de datos y procedimientos de lectura y escritura de archivos para ***std_ulogic***, ***std_logic***, ***std_ulogic_vector*** y ***std_logic_vector***

Tipos de datos



Tipos de datos



(Sintetizable)

(No Sintetizable)

Tipos de datos: escalares

- Enteros.

```
type integer is range -2147483647 to +2147483647;
```

$-(2^{31}-1)$ $(2^{31}-1)$

```
signal a integer;

Begin

  a <= 23;
  a <= -37;
  a <= 23456789;
  a <= 23_456_789;
  a <= 12e2;
  a <= 23.0;
```

a(0⁻)= -2147483647

```
signal c: integer range 0 to 122;
signal d: integer range -34 to 456;

Begin

c <= 90.0;   ← Error, no es un entero.
c <= 90;
d <= -37;    ← Error, esta fuera del rango de c
c <= -90;
c <= d;      ← Error si d toma un valor fuera del rango de c.
c <= d;
```

Para síntesis es recomendable acotar los ***integer***,

El valor inicial (t=0⁻) de una variable, señal o puerto, sino se especifica, es el igual al primer elemento del tipo.

c(0⁻)= 0
d(0⁻)= -34

UAH
Departamento de Electrónica

Tipos de datos: escalares

- Enteros.

Hay dos subtipos predefinidos de tipo entero:

```
subtype natural is integer range 0 to integer'high;
```

```
subtype positive is integer range 1 to integer'high;
```

- A un objeto sólo se le puede asignar un dato que sea del tipo utilizado en su declaración.
  - Un tipo y un subtipo no pueden considerarse como tipos diferentes

```
signal a: integer;
signal b: natural range 6 to 122;
signal c: positive;
 . . .
Begin

c<=a;   ← Error si a toma un valor menor que 1.
a<=b;
b<=c;
```

Tipos de datos: escalares

Reales

```
type real is range -1.0E38 to 1.0E38;
```

El tipo real no es aceptado por la mayoría de las herramientas de síntesis

```
signal a:real;
signal b: real range -34.0 to 456.0;
signal c: integer;
signal d: integer range -34 to 456;

begin
a<=b;
b<=a;   <-- Error si a toma un valor fuera del rango de b.
a<=c;   <-- Error, tipos de datos diferentes.
d<=b;   <-- Error, tipos de datos diferentes.
a<=2.3;
a<=-3.7;
a<=1_012.267_001e-3;
a<=2.3e3;
a<=2e2;   <-- Error, no es real.
a<=23;    <-- Error, no es real.
```


Tipos de datos: escalares

Enteros y Reales

- Los enteros y reales admiten la conversión tipo *CAST*. Comparten un conjunto de valores.

```
signal a:real;
signal b: real range -34.0 to 456.0;
signal c: integer;
signal d: integer range -34 to 456;

Begin

a<=real(c);
c<=integer(a);
d<=integer(b);
b<=real(d);
```

Tipos de datos: escalares

- Tipos para representar valores lógicos.
  - Son tipos enumerados en los que se define el conjunto de posibles valores.
  - Definidos en el paquete ***Standar*** de la librería ***std***.

```
type BOOLEAN is (FALSE, TRUE);
```

```
type BIT is ('0', '1');
```

← Se puede utilizar para síntesis. Aunque no lo vamos a hacer.

  - Definidos en el paquete ***std_logic_1164*** de la librería ***ieee***.

```
type std_ulogic is ( 'U', -- Uninitialized
                     'X', -- Forcing Unknown
                     '0', -- Forcing 0
                     '1', -- Forcing 1
                     'Z', -- High Impedance
                     'W', -- Weak Unknown
                     'L', -- Weak 0
                     'H', -- Weak 1
                     '-' -- Don't Care
                   );
```

```
SUBTYPE std_logic IS resolved std_ulogic;
```

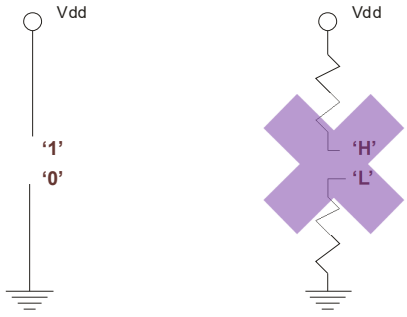
¡Es el que se utiliza!.

Para poder utilizar el tipo ***std_logic*** hay que declarar el paquete ***std_logic_1164***

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


Tipos de datos: escalares

- El tipo de datos ***std_logic***.
  - Valores lógicos: '0', '1', 'L' 'H'.



- Valores metalógicos: 'U', 'X', 'W', '-'

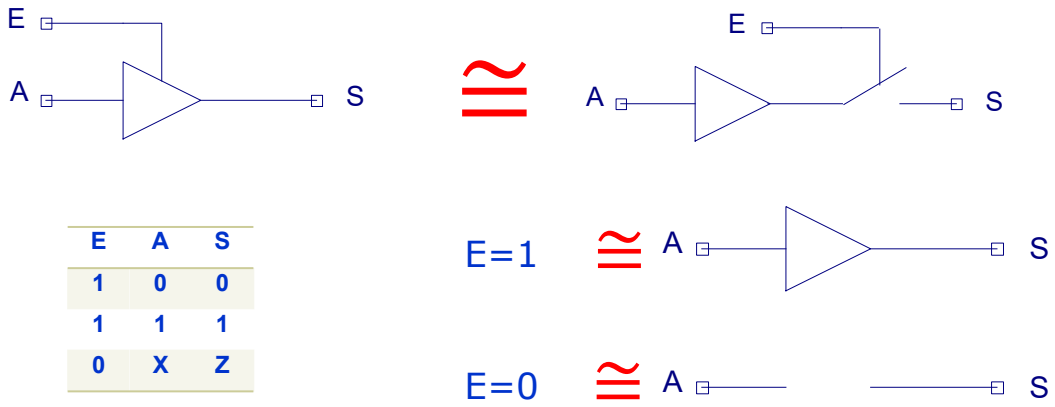
Para VHDL-93, no se pueden utilizar en síntesis.

Para VHDL-08, '-' se pueden utilizar en síntesis, con ciertas particularidades, pero sólo este valor

Tipos de datos: escalares

- ¿Qué es la alta impedancia ('Z')?

Es un tercer estado (tristate, Z), aparte de los niveles alto (1) y bajo (0).

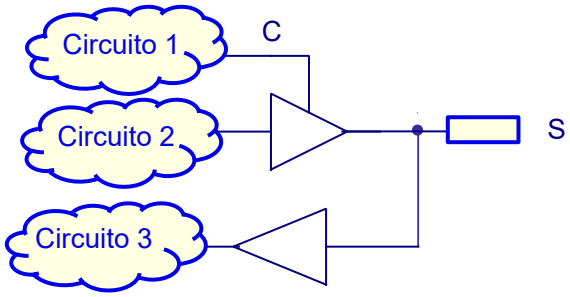


E	A	S
1	0	0
1	1	1
0	X	Z

UAH
Departamento de Electrónica

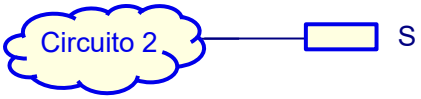
Tipos de datos: escalares

- ¿Qué es la alta impedancia ('Z')?



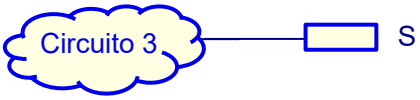
¡Se suele utilizar con puertos ***inout***!

C=1



¡El pin funciona como salida!

C=0



¡El pin funciona como entrada!

S tiene múltiples drivers.

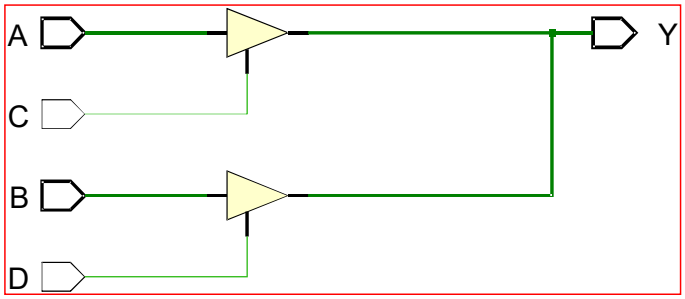
Esto sólo es posible si el tipo de datos de S es un tipo resuelto.

¡El *std_logic*, es un tipo resuelto!

Tipos de datos: escalares

```
library ieee;
use ieee.std_logic_1164.all;
entity buff_triestate is
  port(a, b : in  std_logic;
       c, d : in  std_logic;
       y    : out std_logic);
end entity;
architecture RTL of buff_triestate is
begin
  y <= a when c = '1' else 'Z';
  y <= b when d = '1' else 'Z';
end;
```

```
architecture RTL of buff_triestate is
begin

  process (all)
  begin
    if C = '1' then
      Y <= A;
    else
      Y <= 'Z';
    end if;
  end process;

  process (all)
  begin
    if D = '1' then
      Y <= B;
    else
      Y <= 'Z';
    end if;
  end process;
end;
```



Vectores

- Vectores.
  - Están formados por varios elementos o unidades atómicas a las cuales se puede acceder de forma individual, parcial o conjunta.

```
type bit_vector is array (natural range <>) of bit;
```

```
type std_logic_vector is array (natural range <>) of std_logic;
```

⚠ No admiten los operadores aritméticos

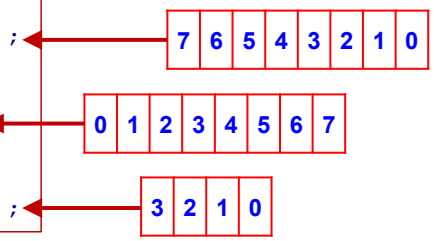
  - VHDL-08 añade dos nuevos vectores en el paquete ***standard***.

```
type integer_vector is array (natural range <>) of integer;
type boolean_vector is array (natural range <>) of boolean;
```

Vectores.

El tamaño del vector se indica en la declaración del objeto

```
signal bus_1 : std_logic_vector(7 downto 0);
signal bus_2 : std_logic_vector(0 to 7);
signal a     : std_logic_vector(3 downto 0);
```



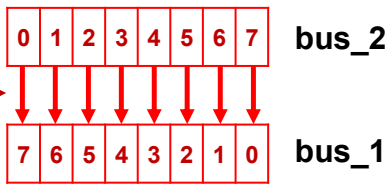
El elemento de la izquierda siempre es el de mayor peso.

Vectores

```
signal bus_1 : std_logic_vector(7 downto 0);
signal bus_2 : std_logic_vector(0 to 7);
signal a     : std_logic_vector(3 downto 0);
signal b     : bit_vector(3 downto 0);
```

- La asignación se hace elemento a elemento, independientemente de los índices.
- El dato asignado debe tener **igual** tamaño que el objeto al que se le asigna.
- El tipo de los elementos de los vectores debe ser igual.

```
...
bus_1<=bus_2;

...

bus_1<= a;
...
a     <= b;
```



Error: tamaños diferentes

Error: tipos diferentes

Asignación de valores.

```
signal a, b : std_logic_vector(7 downto 0);
...
    a              <= "00001111";
    a              <= "0000_1111";
    a              <= x"0f";
    a(7)           <= '0';
    a(7 downto 4)  <= "1000";
    a(4 to 7)      <= "1000";
    a(7 downto 5)  <= x"5";
    a              <= (others => '0');
    a(3 downto 0)  <= b(7 downto 4);
```

Error: no se puede utilizar el carácter _ con vectores

Error: hay que utilizar el mismo sentido que en la declaración.

Error: tamaños diferentes.

La palabra reservada ***others***: asigna un valor a todos los elementos.

```
signal a,b,s: std_logic_vector(7 downto 0);
...
s(7 downto 4)<=a(7 downto 4);
...

s(3 downto 0)<=b(3 downto 0);
...
s(7 downto 0)<=b(7 downto 0);
```

Es equivalente a: `s<=b;`

Asignación por posición.

```
signal a,b : std_logic_vector(3 downto 0);
...
a<=(b(2),a(0), b(1),'1');

b<=('1','0','Z','0');
```

3	2	1	0
b(2)	a(0)	b(1)	'1'

3	2	1	0
'1'	'0'	'Z'	'0'

```
b<=('1',"101");
```

3	2	1	0
'1'	'1'	'0'	'1'

¡Sólo es valido para VHDL-08!

```
b<=('1',a(3 downto 1));
```

Error: no se pueden utilizar un vectores.

- Asignación por identificador.

  Los índices de los elementos de los vectores son siempre enteros.

```
signal a, b : std_logic_vector(7 downto 0);
...
    a <= (7 => '1', 5 => '1');                          -- Error: tamaños diferentes.
    a <= (7 => '1', 5 => '1', others => '0');           -- 1010_0000
    a <= (7|5 => '1');                                  -- Error: tamaños diferentes.
    a <= (7|5 => '1', others => '0');                   -- 1010_0000
    a <= (3 downto 0=>'0', others =>'1');               -- 1111_0000
    a <= ( '1', '0', '1', others => '0');               -- 1010_0000
    a <= ('1', '0', others => '0');                     -- 1000_0000
    a <=(4 => '1', others => '0');                      -- 0001_0000
```

- VHDL-08 permite asignar un grupo de valores

```
a <= ( "1010",others => '0');          -- 1010_0000
a <= (others => '0', "1010");          -- Error, asignación incorrecta
```


- Descomposición de vectores.

  Solo es válido para VHDL-08.

```
signal S, D: std_logic_vector(7 downto 0);
. . .
( S(3 downto 0), S(7 downto 4)) <= D;
```

```
S(3 downto 0)<= D(7 downto 4);
S(7 downto 4)<= D(3 downto 0);
```

```
( 3 downto 0 => S, 7 downto  4 => S) <= D;
```

```
signal a, b, sum : unsigned(15 downto 0);
signal c_out : std_logic;
...
(c_out, sum) <= ('0' & a) + ('0' & b);
```

```
signal a, b,sum: unsigned(15 downto 0);
signal aux : unsigned(16 downto 0);
signal c_out : std_logic;
...
aux<= ('0' & a) + ('0' & b);
c_out <= aux (16);
sum <= aux(15 downto 0)
```

```
signal S0 : std_logic_vector(7 downto 0);
signal S1, S2: std_logic_vector(2 downto 0);
signal S3, S4 : std_logic;
...
( 2 downto 0 => S1,6 downto 4 => S2,3 => S3, 7 => S4) <= S0;
```

```
S1 <= S0(2 downto 0);
S2 <= S0(6 downto 4) ;
S3 <= S0(3);
S4 <= S0(7);
```

Vectores

- Otra forma de representación de números en VHDL (Sólo para VHDL-08).
  - Para los vectores tipo *std_logic_vector* o *unsigned*/*signed* el formato es ***NB"valor"***, donde ***N*** representa el tamaño en bits, ***B*** indica la base. Cuando se omite la base, por defecto se considera binario. Bases:

    B o b para binario, ***O u o*** para octal, ***D o d*** para decimal y ***X o x*** para hexadecimal

Número	Bits	Base	Valor decimal	Valor en binario
4B"1100"	4	2	12	1100
B"110"	3	2	6	110
8B"10"	8	2	2	00000010
3D"7"	3	2	7	111
6O"41"	6	8	33	100001
8X"AA"	8	16	170	10101010
"100"	3	2	4	100
X"AB"	8	16	171	10101011
7X"3E"	7	16	62	00111110
7O"141"	7	8	33	01100001
8X"3"	8	16	3	00000011
8SX"5"	8	16	+5	00000101
8SX"F"	8	16	-1	11111111
8UX"3"	8	16	3	00000011
8UX"A"	8	16	10	00001001

Por defecto se considera la base binaria.

Para los datos especificados en hexadecimal u octal se pueden asignar a vectores con tamaños no múltiplos de 4 o 3 respectivamente

Para los datos especificados con signo (*signed*) el bit de mayor peso del dato representa el signo. El valor es en complemento a 2.

Para los datos especificados sin signo (*unsigned*) el valor se considera binario natural.

Vectores.

- Vectores tipo *unsigned/signed.*

Los operadores aritméticos no soportan los vectores binarios (*bit_vector* y *std_logic_vector*).

En la biblioteca *ieee* se definen dos paquetes *numeric_bit* y *numeric_std* en los que se declaran los operadores aritméticos para soportar los vectores binarios: el *numeric_std* para el tipo *std_logic_vector* y el *numeric_bit* para el tipo *bit_vector.*

Se definen dos nuevos vectores:

```
type unsigned is array (natural range <>) of bit;
type signed   is array (natural range <>) of bit;
```

```
library ieee;
use ieee.numeric_bit.all;
```

```
type unsigned is array (natural range <>) of std_logic;
type signed   is array (natural range <>) of std_logic;
```

```
library ieee;
use ieee.numeric_bit.all;
```

Permiten utilizar todos los operadores aritméticos

Vectores.

- Vectores tipo *unsigned/signed.*

```
 ....
signal suma,sumando_1:unsigned (7 downto 0);
signal       sumando_2:unsigned (3 downto 0);

....
suma<= sumando_1 + sumando_2;


suma<= sumando_1 + 12;
```

El tamaño del vector resultado debe ser igual al tamaño del mayor de los operandos.

Se permite operar con enteros

```
if suma  > 45 then
```

Se pueden comparar con enteros

```
suma<= 12;
```

Error, Tipos diferentes

Las asignaciones son iguales que las vistas para el tipo *std_logic_vector*

Vectores

- Conversión entre *signed/unsigned* y *std_logic_vector*.
  - *Cast* : por ser arrays similares que tienen la misma longitud y tienen elementos idénticos o convertibles.

```
type std_logic_vector is array (NATURAL range <>) of STD_LOGIC;
```

```
type unsigned is array (natural range <>) of std_logic;
type signed   is array (natural range <>) of std_logic;
```

```
signal ent_a, ent_b: std_logic_vector(7 downto 0);
signal res_1,res_2: std_logic_vector(7 downto 0);
........
    ent_a<="11111101";
    ent_b<="00001001";
    res_1<= std_logic_vector(unsigned(ent_a)+unsigned(ent_b));
........
    res_2<= std_logic_vector(signed(ent_a)+signed(ent_b));
```

```
signal a,b :  std_logic_vector (7 downto 0);
signal s: std_logic_vector(8 downto 0));

s<=std_logic_vector(unsigned(a)+unsigned('0'&b));
```

Sumador sin overflow

Conversión entre vectores

- Conversión entre *signed/unsigned* y enteros (*integer*).
  - Hay que utilizar funciones de conversión

```
function TO_UNSIGNED (  ARG,SIZE: NATURAL) return UNSIGNED;
     -- Result subtype: UNSIGNED (SIZE-1 downto 0)
     -- Result: Converts a non-negative INTEGER to an UNSIGNED vector with
     --         the specified SIZE.

     -- Id: D.4
  function TO_SIGNED ( ARG: INTEGER; SIZE: NATURAL) return SIGNED;
     -- Result subtype: SIGNED (SIZE-1 downto 0)
     -- Result: Converts an INTEGER to a SIGNED vector of the specified SIZE.

     -- Id: D.5
  function TO_UNSIGNED ( ARG: STD_LOGIC_VECTOR) return UNSIGNED;
     -- Result subtype: UNSIGNED, same range as input ARG
     -- Result: Converts STD_LOGIC_VECTOR to UNSIGNED.

     -- Id: D.6
  function TO_SIGNED ( ARG: STD_LOGIC_VECTOR) return SIGNED;
     -- Result subtype: SIGNED, same range as input ARG
     -- Result: Converts STD_LOGIC_VECTOR to SIGNED.
```

```
for i in 7 downto 0 loop
   if E(i) = '0' then
     P <= "01";
     S <= std_logic_vector(to_unsigned(i,3));
   end if;
end loop;
```

```
function TO_INTEGER ( ARG: UNSIGNED) return NATURAL;
     -- Result subtype: NATURAL. Value cannot be
negative since parameter is an
     --                  UNSIGNED vector.
     -- Result: Converts the UNSIGNED vector to an
INTEGER.

     -- Id: D.2
function TO_INTEGER ( ARG: SIGNED) return INTEGER;
     -- Result subtype: INTEGER
     -- Result: Converts a SIGNED vector to an INTEGER.

     -- Id: D.3
```


Vectores

- Formación de vectores.

Se utiliza el operador **&**

Es igual para los vectores de tipo *std_logic_vector* y *unsigned*/*signed*.

```
signal sel1,sel2 : std_logic;
signal bus_std_1 : std_logic_vector(6 downto 0);
signal bus_std_2 : std_logic_vector(7 downto 0);
signal bus_std_3 : std_logic_vector(7 downto 0);
signal ax ,bx: std_logic_vector(3 downto 0);
signal sel   : std_logic_vector(1 downto 0);
signal aux   : unsigned(1 downto 0);
signal cx ,dx: unsigned(3 downto 0);
signal  bus_unsg_1  : unsigned(7 downto 0);
signal bus_unsg_2   : unsigned(4 downto 0);
```

```
sel<='`1'&'0';
```

```
aux<='1'&'0';
```

```
sel<=sel1&sel2;
```

```
aux<=sel1&sel2;
```

```
bus_std_1<= bus_std_2(3 downto 1)&"0010";
```

```
bus_unsg_1<= cx&dx;
```

```
bus_std_2<= ax&bx;
```

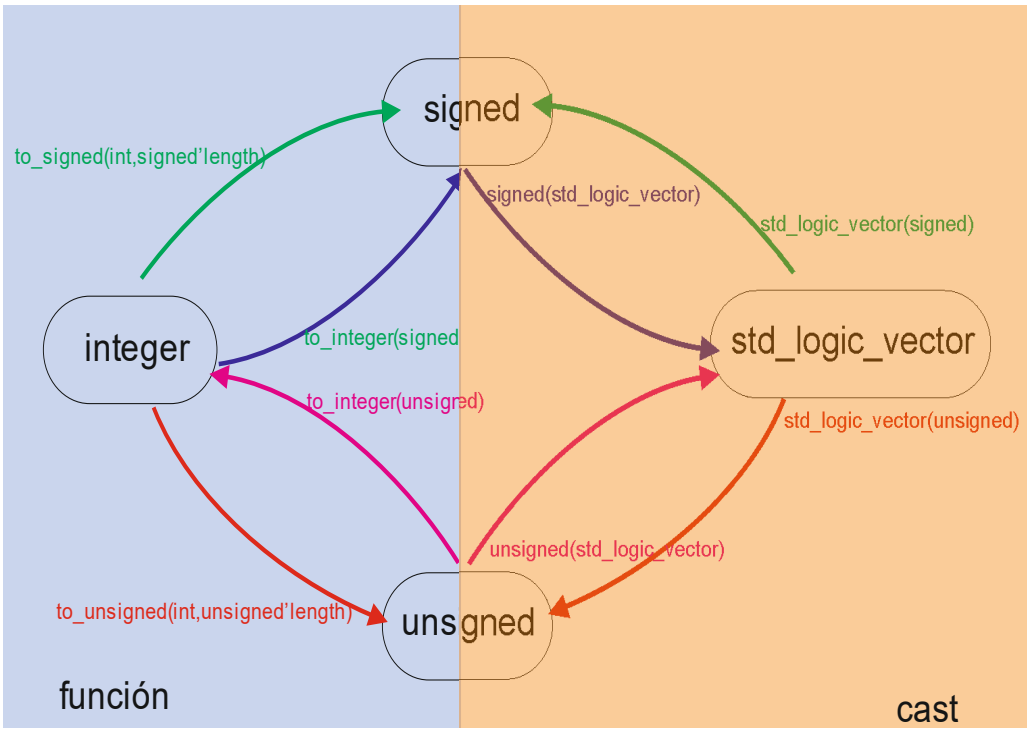
```
bus_unsg_2<=bus_unsg_1(4 downto 1)&'0';
```

```
bus_std_3<=bus_std_3(6 downto 0)&'0';
```

```
bus_unsg_2<=bus_unsg_2(3 downto 0)&bus_unsg_2(4);
```

```
bus_std_3<= bus_std_3(0)& bus_std_3(7 downto 1);
```

Conversión de tipos



Vectores

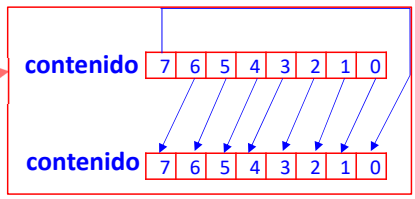
- Array de arrays

```
type t_mem2 is array (15 downto 0) of std_logic_vector(7 downto 0);
signal Ram1,Ram2       : t_mem2;
signal contenido : std_logic_vector(7 downto 0);
...
Ram2            <= (others => (others=>'1'));
Ram1            <= Ram2;
Ram1(4)( 7)      <= '0';
contenido       <= Ram1(5)(7 downto 0);
contenido       <= Ram2(1);
```

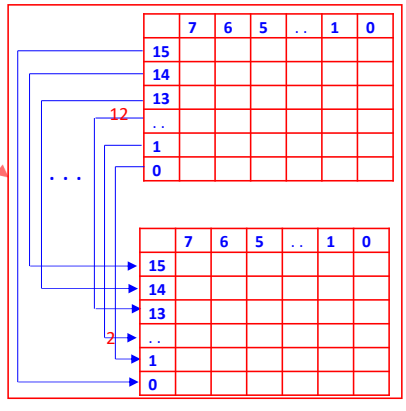
	7	6	5		1	0
15	'0'	'1'	'0'	...	'0'	'1'
14	'1'	'0'	'1'	...	'0'	'1'
13	'1'	'0'	'0'	...	'1'	'1'
	...	...	...	...	...	...
1	'1'	'1'	'0'	...	'1'	'0'
0	'0'	'1'	'1'	...	'1'	'0'

Error: los índices no pueden ser un rango

```
contenido<= contenido(6 downto 0)& contenido(7);
```




```
Ram1<= Ram1(14 downto 0)& Ram1(15);
```



Estructura de un archivo VHDL

Bibliotecas

```
library nombre_librería
use nombre_librería.nombre_paquete.all
```

Entidad

```
entity nombre_entidad  is
        Listado_puertos
        -- Declaración de pines
end nombre_entidad;
```

Arquitectura

```
architecture nombre_arquitectura of  nombre_entidad
-- Declaración de señales internas
-- Declaración de componentes en caso de instanciación
begin
-- Cuerpo de la arquitectura:
        -- Se define la funcionalidad del diseño con:
                -- asignaciones concurrentes
                -- procesos
                -- instanciación de componentes
end nombre_arquitectura;
```


Entidades

- Describen el nombre del diseño y la interfaz con el exterior.

```
entity identificador_entidad is
        generic (lista_genericos);
        port (lista_puertos);
end [entity][identificador_entidad];
```

- Indica el nombre que recibe la entidad.
- Se utiliza para referenciar a la entidad.

- Son constantes que permiten la parametrización del código.

- Son señales para comunicar el circuito con el exterior.
- Son vistos por todas las arquitecturas de la entidad.

Entidades

- Puertos: representan los terminales de nuestro modelo.

```
port (puerto_1: <modo> <tipo>;
      puerto_2: <modo> <tipo>;
         ....
      puerto_N: <modo> <tipo>);
```

```
port (puerto_a, puerto_c, puerto_d : <modo> <tipo>;
      puerto_2: <modo> <tipo>;
         ....
      puerto_N: <modo> <tipo>);
```

- ***In:*** Puerto de entrada. → Su valor sólo puede ser leído.
- ***Out:*** Puerto de salida. → Su valor sólo puede ser escrito (VHDL-93).
  Su valor puede ser leído o escrito (VHDL-08).
- ***Inout:*** Puerto bidireccional. → Su valor puede ser tanto leído como escrito.

Entidades

```
port (puerto_1: <modo> <tipo>;
      puerto_2: <modo> <tipo>;
         ....
      puerto_N: <modo> <tipo>);
```

- Terminales individuales:

```
std_logic
```

```
'U' -- Uninitialized
'X' -- Forcing Unknown
'0' -- Forcing 0
'1' -- Forcing 1
'Z' -- High Impedance
'W' -- Weak Unknown
'L' -- Weak 0
'H' -- Weak 1
'-' -- Don't Care
```

- Terminales múltiples (bus): Vectores.

```
std_logic_vector (7 downto 0)
```

¡Con los valores de los índices se define el tamaño del vector!

Entidades

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity eje_generic is
  generic (
    N               :     positive :=8);
  port (
    A               : in  std_logic_vector(N-1 downto 0);
    B               : in  std_logic_vector(N-1 downto 0);
    RESULTADO       : out std_logic_vector(N downto 0));
end eje_generic;
```

¡Es necesario asignar un valor por defecto a ***N*** para poder realizar la síntesis!

```
architecture RTL of dis_1 is
. . . . .
constant N : positive := 32;
signal CH_A   : std_logic_vector(N-1 downto 0);
signal CH_B   : std_logic_vector(N-1 downto 0);
signal CH_OUT : std_logic_vector(N downto 0);
. . . . .

begin
. . . . .
  DUT:entity work.eje_generic
    generic map (
        N => N)
    port map (
        A         => CH_A,
        B         => CH_B,
        RESULTADO => CH_OUT);
. . . . .
```

¡Cuando se llama (instancia) en otra entidad de nivel superior se le asigna valor a los genéricos y puertos!

Para síntesis: A los genéricos se le debe asignar una constante

A los puertos se le debe asignar una señal o un puerto de la entidad de nivel superior.

UAH
Departamento de Electrónica

Entidades

- En VHDL-08 el valor de los genéricos pueden referenciar a otros genéricos.

```
entity ejemplo is
  generic (
    SIZE  :     natural := 8;
    D     :     std_logic_vector(2*SIZE -1 downto 0));
  port (
    A     : in  std_logic_vector(D downto 0);
    S     : out std_logic_vector(D downto 0));
end ejemplo;
```

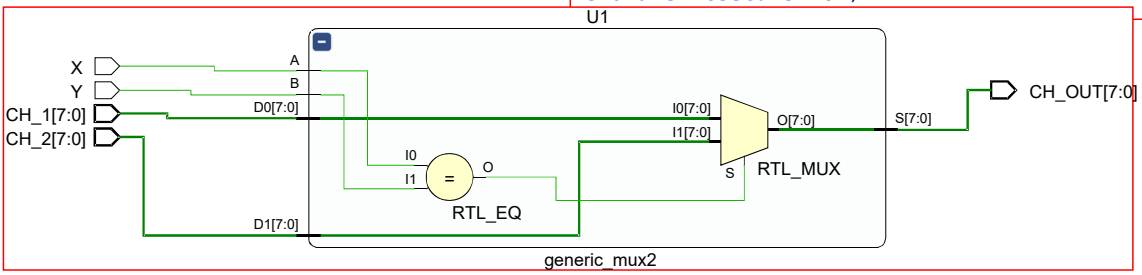
Entidades

- En VHDL-08 los genéricos pueden tipos, subprogramas y paquetes.

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

entity generic_mux2 is
  generic ( type data_type );
  port ( A, B   : in  std_logic;
         D0, D1 : in  data_type;
         S      : out data_type );
end entity generic_mux2;

architecture rtl of generic_mux2 is
begin
  process (all)
  begin  -- process
    if A = B then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;

end architecture rtl;
```

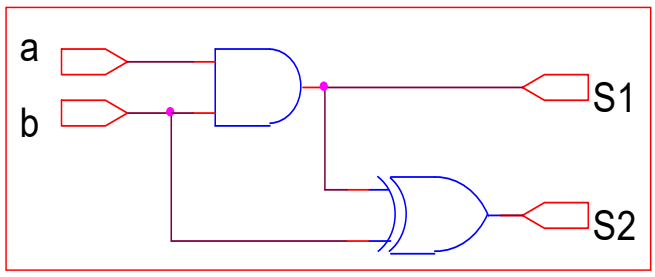
¡Al tipo no se la asigna un valor por defecto!

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

entity mux2_8b is
  port ( X, Y   : in  std_logic;
         CH_1   : in  std_logic_vector(7 downto 0);
         CH_2   : in  std_logic_vector(7 downto 0);
         CH_OUT : out std_logic_vector(7 downto 0 ));
end entity mux2_8b;

architecture rtl of mux2_8b is
begin

U1 : entity work.generic_mux2
  generic map(
    data_type => std_logic_vector(7 downto 0))
  port map (
    A  => X,
    B  => Y,
    D0 => CH_1,
    D1 => CH_2,
    S  => CH_OUT);

end architecture rtl;
```



Entidades

- Ejemplo.



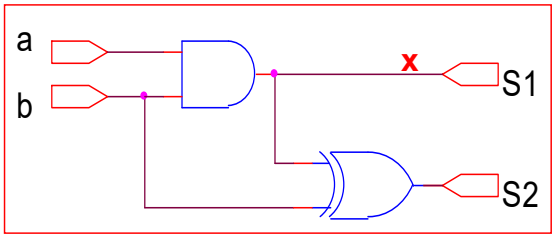

```
library ieee;
use ieee.std_logic_1164.all;


entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;


architecture rtl of ejemplo1 is
begin
  s1 <= a and b;
  s2 <= s1 xor b;
end rtl;
```

Esta línea:

Es errónea para VHDL-93

No es errónea para VHDL-08

- Ejemplo (VHDL-93).

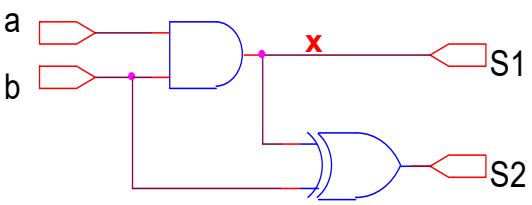



```
library ieee;
use ieee.std_logic_1164.all;

entity ejemplo1 is
  port(a, b : in  std_logic;
       s1, s2  : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin
  x <= a and b;
  s1<= x;
  s2<= x xor b;
end rtl;
```

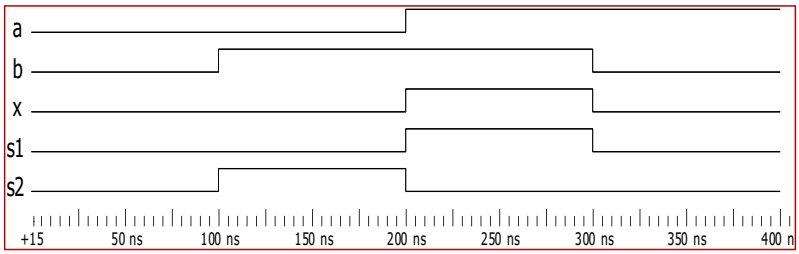
¡Ahora el código es correcto para VHDL-93!

¡Y también para VHDL-08!

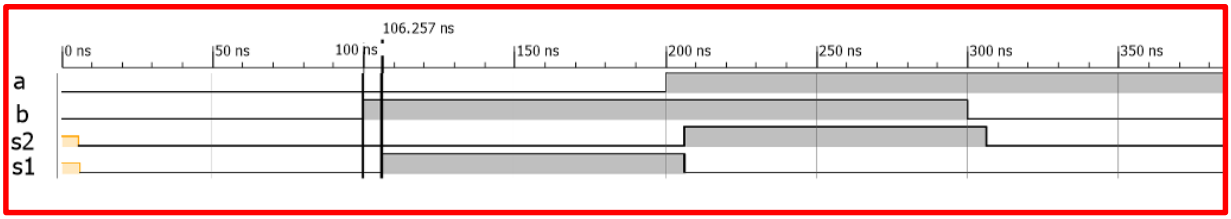
- Ejemplo (VHDL-93).



Simulación funcional



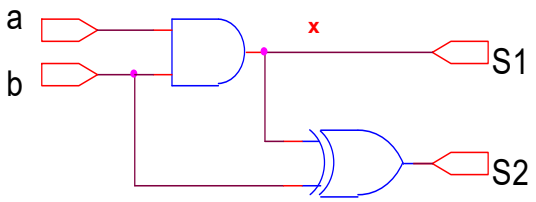
Simulación temporal



UAH

Departamento de Electrónica

- Ejemplo (VHDL-93).




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

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;
   s1  <= x;
   x   <= a and b;
end rtl;
```

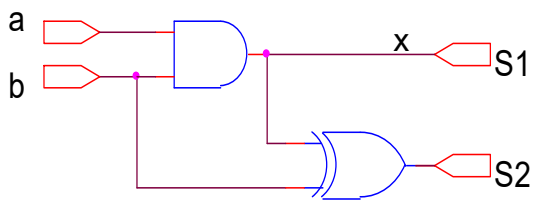
¿ES CORRECTO ESTE CÓDIGO ?

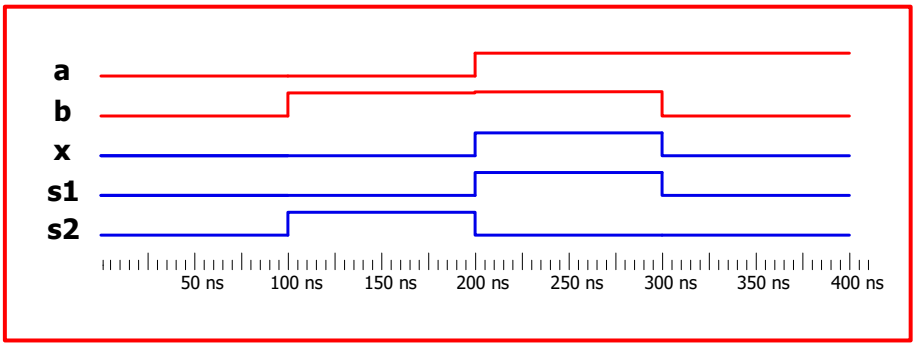
¡SI!

UAH

Departamento de Electrónica

- Ejemplo (VHDL-93).





Concurrencia y secuencialidad

- La concurrencia se consigue por la asignación diferida de las señales (***signal***) y los puertos.
- Las señales se pueden leer y escribir
- Las señales tienen 2 campos:

Campo de valor	Cola de eventos
Refleja el valor actuar	Contiene el conjunto de posibles valores futuros a tomar
Es de sólo lectura	Es de sólo escritura

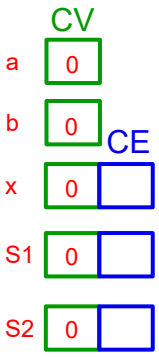
Si el puerto es de modo ***in***, *sólo tiene Campo de valor*

Si el puerto es de modo ***out*** *tiene: Cola de eventos (VHDL-93) o los dos campos (VHDL-08)*

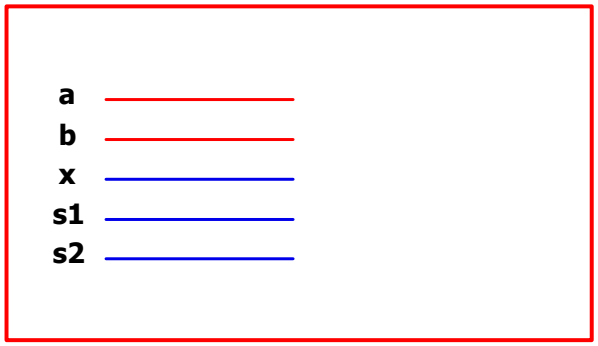
Si el puerto es de modo ***inout***, *tiene los dos campos*

Concurrencia y secuencialidad

- ¿Cómo se simula el código *(VHDL-93)*?

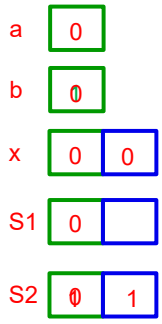


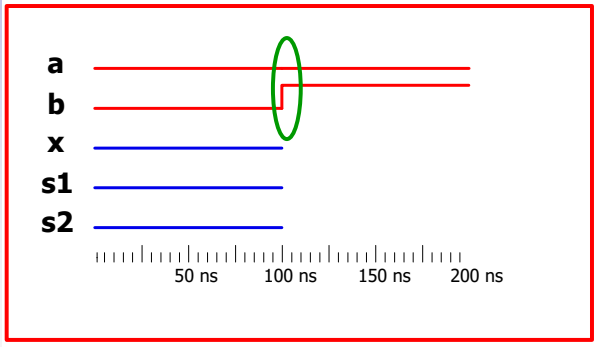

```
library ieee;
use ieee.std_logic_1164.all;

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;
   s1  <= x;
   x   <= a and b;
end rtl;
```



Concurrencia y secuencialidad

- ¿Cómo se simula el código *(VHDL-93)*?






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

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;  ✓
   s1  <= x;        ✘
   x   <= a and b;  ✓
end rtl;
```

La simulación se ejecuta paso a paso hasta que se produzca un evento en un puerto de entrada (*in*).

Al haber un evento se para el tiempo de simulación y se ejecuta (elabora) toda la arquitectura.

Sólo se ejecutan las sentencias que dependen del elemento (objeto) que ha sufrido el evento.

Los valores se asignan a la *Cola de Eventos* del objeto.

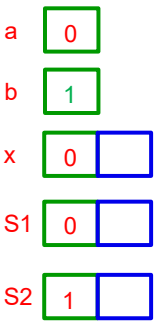
Al llegar al final de la arquitectura se actualizan todas las señales y puertos.

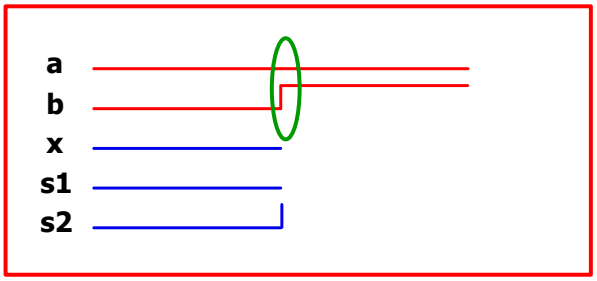
Esta ejecución se conoce como “un ciclo Δ”. Δ= 0ns

UAH
Departamento de Electrónica

Concurrencia y secuencialidad

- ¿Cómo se simula el código *(VHDL-93)*?






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

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;  ✘
   s1  <= x;        ✘
   x   <= a and b;  ✘
end rtl;
```

Al producirse un nuevo evento (S2) se vuelve a elaborar toda la arquitectura.

Se ejecutan las sentencias que dependen del objeto (S2)

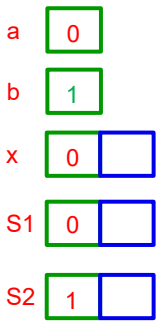
Al llegar al final de la arquitectura se vuelven a actualizan todas las señales y puertos.

Se ha producido otro ciclo Δ.

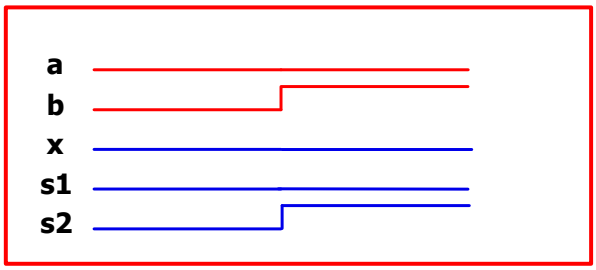
Como ya no hay eventos se actualiza el valor de los objetos.

Concurrencia y secuencialidad

- ¿Cómo se simula el código *(VHDL-93)*?




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

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;
   s1  <= x;
   x   <= a and b;
end rtl;
```

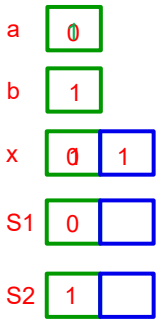


Como no hay eventos en los puertos de entrada no se vuelve elaborar la arquitectura

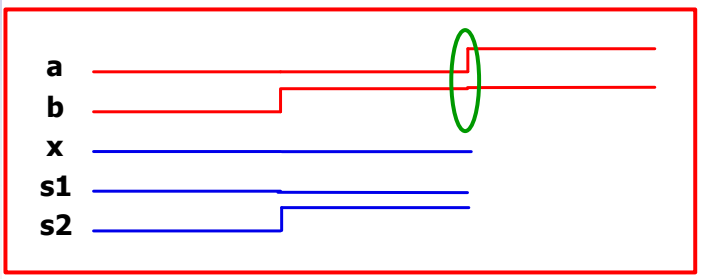
Se continua con la simulación paso a paso.

Concurrencia y secuencialidad

- ¿Cómo se simula el código *(VHDL-93)*?




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

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;  ✘
   s1  <= x;        ✘
   x   <= a and b;  ✓
end rtl;
```

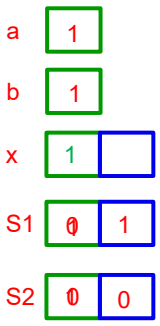


Primer ciclo Δ.

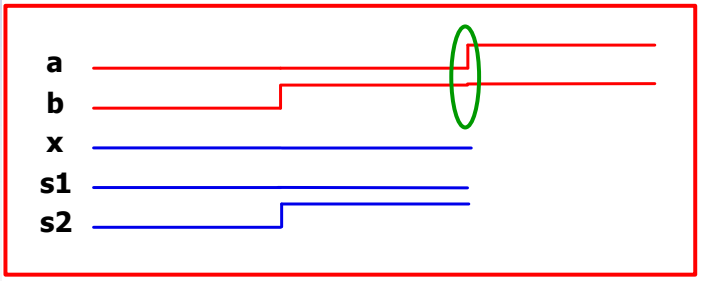
Como hay eventos (x) se vuelve a elaborar la arquitectura

Concurrencia y secuencialidad

- ¿Cómo se simula el código *(VHDL-93)*?




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

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;  ✓
   s1  <= x;        ✓
   x   <= a and b;  ✗
end rtl;
```



Segundo ciclo Δ.

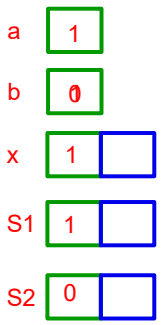
Como hay eventos (S1 y S2) se volvería a elaborar la arquitectura.

No tiene sentido porque son salidas.

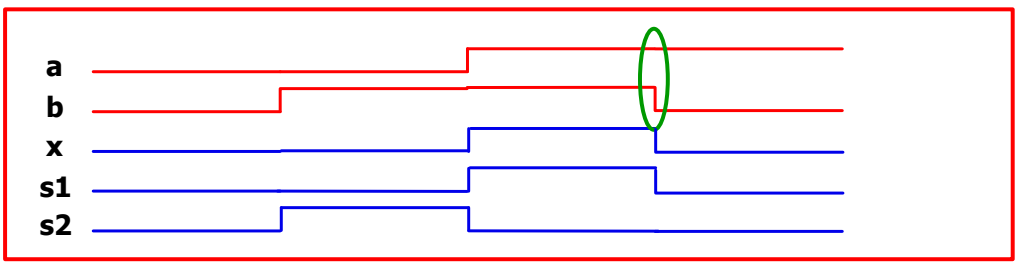
UAH
Departamento de Electrónica

Concurrencia y secuencialidad

- ¿Cómo se simula el código *(VHDL-93)*?




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

entity ejemplo1 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   s2  <= x xor b;
   s1  <= x;
   x   <= a and b;
end rtl;
```



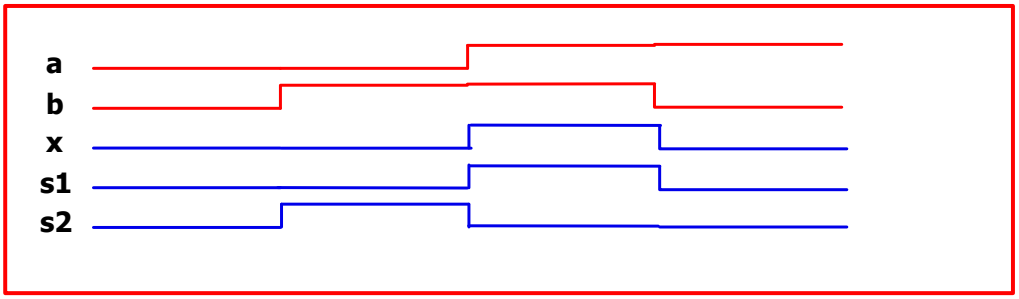
❑ Ejercicio: Simular el código (VHDL-93).

Las sentencias así construidas son concurrentes:

Se ejecutan todas a la vez.

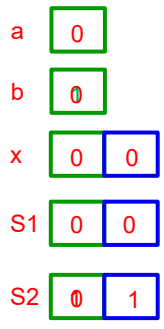
El orden en el que se escriben no importa.

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

entity ejemplo1 is
  port(a, b    : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo1;

architecture rtl of ejemplo1 is
  signal x    :     std_logic;
begin

   x    <= a and b;
   s1   <= x;
   s2   <= x xor b;
end rtl;
```



❑ Asignación secuencial.

Es idéntica para ambos estándares (VHDL-93 o VHDL-08)

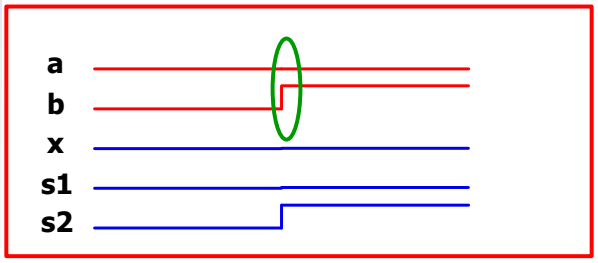


¡El código que hay en el proceso se ejecuta siembre que haya un evento en uno de estos objetos!

¡A esto se le llama lista de sensibilidad!

Todas las sentencias son secuenciales: se ejecutan una a continuación de la otra.

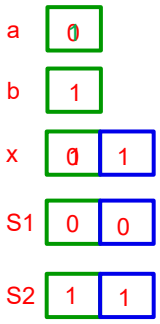
```
library ieee;
use ieee.std_logic_1164.all;

entity ejemplo2 is
  port(a, b    : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo2;

architecture rtl _2 of ejemplo2 is
  signal x    :     std_logic;
begin

 process(a, b)
 begin
   x    <= a and b;  ✓
   s1   <= x;        ✓
   s2   <= x xor b;  ✓
 end process;

end rtl _2;
```



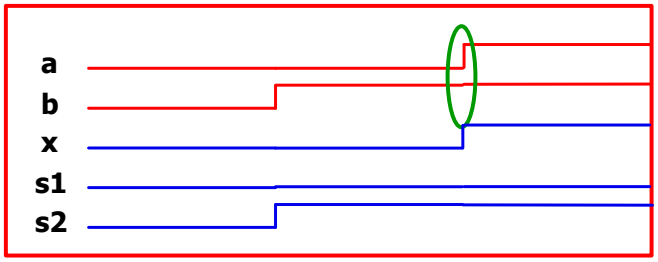
Como el evento está en una salida, no se producen ciclos Δ.

Concurrencia y secuencialidad

- Asignación secuencial.




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

entity ejemplo2 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo2;

architecture rtl _2 of ejemplo2 is
  signal x   :     std_logic;
begin

 process(a, b)
 begin
   x   <= a and b;  ✓
   s1  <= x;        ✓
   s2  <= x xor b;  ✓
 end process;

end rtl _2;
```



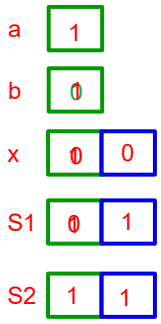
Como los eventos se producen en objetos que no están en la lista de sensibilidad, no se vuelve a ejecutar el proceso.

No se producen más ciclos Δ.

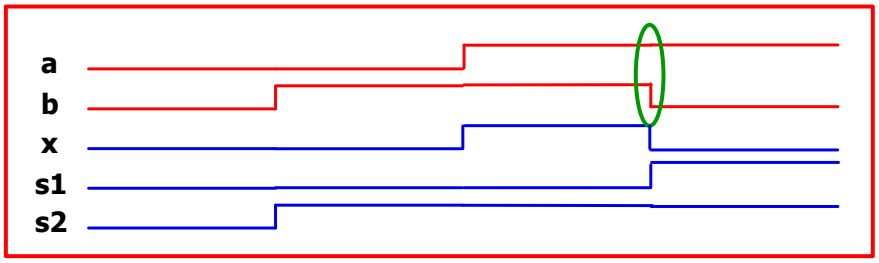
UAH
Departamento de Electrónica

Concurrencia y secuencialidad

- Asignación secuencial.




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

entity ejemplo2 is
  port(a, b   : in  std_logic;
       s1, s2 : out std_logic);
end ejemplo2;

architecture rtl_2 of ejemplo2 is
  signal x   :     std_logic;
begin

 process(a, b)
 begin
   x   <= a and b;  ✓
   s1  <= x;        ✓
   s2  <= x xor b;  ✓
 end process;

end rtl_2;
```



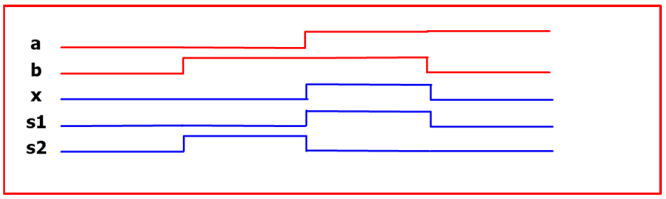
Concurrencia y secuencialidad

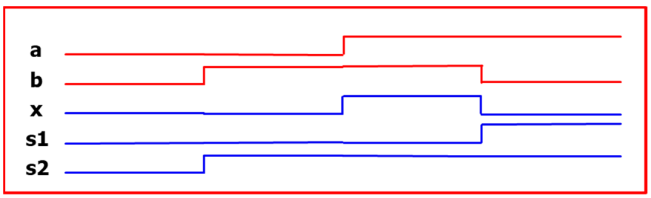
Asignación secuencial. Comparativa

```
architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   x   <= a and b;
   s1  <= x;
   s2  <= x xor b;
end rtl;
```

```
architecture rtl_2 of ejemplo2 is
  signal x   :     std_logic;
begin

 process(a, b)
  begin
   x   <= a and b;
   s1  <= x;
   s2  <= x xor b;
 end process;

end rtl_2;
```

¡Este código es erróneo!





¡Hay que tener mucho cuidado cuando en un proceso se le asigna un valor a una señal y a continuación se examina su valor!

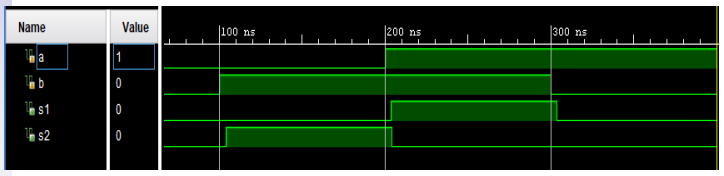
¡Volveremos sobre ello!

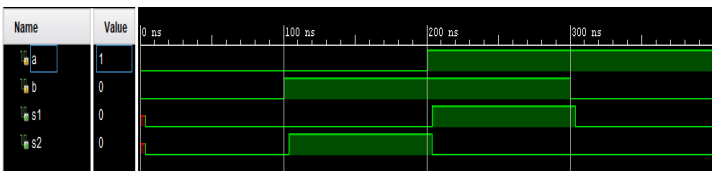
Concurrencia y secuencialidad

Asignación secuencial. Comparativa. Simulación temporal

```
architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   x   <= a and b;
   s1  <= x;
   s2  <= x xor b;
end rtl;
```

```
architecture rtl_2 of ejemplo2 is
  signal x   :     std_logic;
begin

 process(a, b)
  begin
   x   <= a and b;
   s1  <= x;
   s2  <= x xor b;
 end process;

end rtl_2;
```





¿Cómo es posible que ahora sean iguales?

¡El sintetizador completa la lista de sensibilidad!

```
process(a, b, x)
```

```
process(all)
```

Para VHDL-08

- ¿Una posible solución?

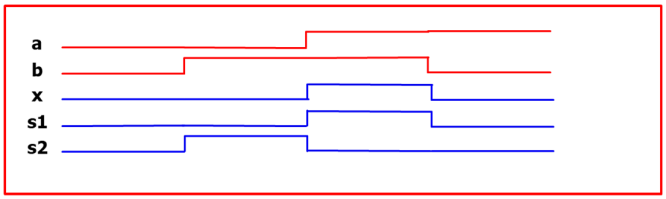
```
architecture rtl_2 of ejemplo2 is
  signal x   :     std_logic;
begin

 process(a, b, x)
  begin
    x   <= a and b;
    s1  <= x;
    s2  <= x xor b;
 end process;

end rtl_2;
```

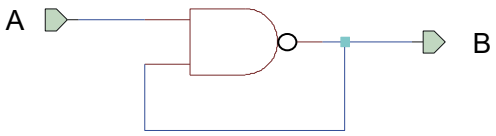
process(all)

¡Una salida actúa como entrada!

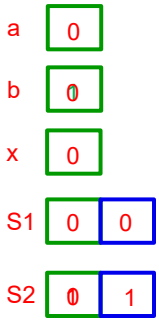
¡Esto puede dar problemas!

¡Se debe evitar SIEMPRE!



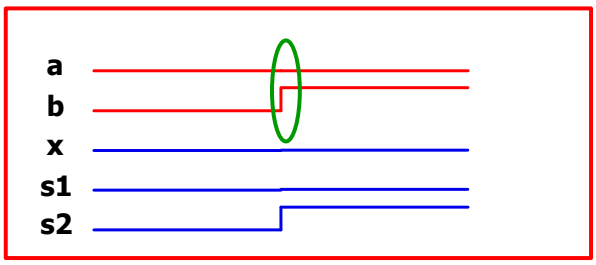


- Asignación secuencial.




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

entity ejemplo3 is
  port(a, b    : in  std_logic;
       s1, s2  : out std_logic);
end ejemplo3;

architecture rtl _3 of ejemplo3 is
begin

 process(a, b)
   variable x   :     std_logic;
 begin
   x   := a and b;  ✓
   s1  <= x;        ✓
   s2  <= x xor b;  ✓
 end process;

end rtl _3;
```

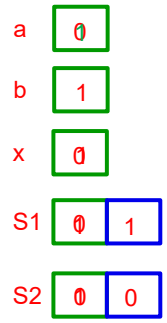
Las variable no tienen asignación diferida .



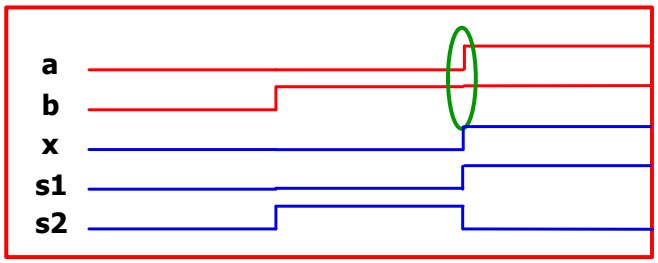
Las variables tienen el mismo comportamiento en ambos estándares (VHDL-93 o VHDL-08)

Concurrencia y secuencialidad

- Asignación secuencial.



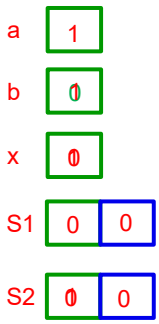

```
library ieee;
use ieee.std_logic_1164.all;

entity ejemplo3 is
  port(a, b    : in  std_logic;
       s1, s2  : out std_logic);
end ejemplo3;

architecture rtl _3 of ejemplo3 is
begin

 process(a, b)
   variable x    :      std_logic;
 begin
   x    := a and b;  ✓
   s1   <= x;        ✓
   s2   <= x xor b;  ✓
 end process;

end rtl _3;
```



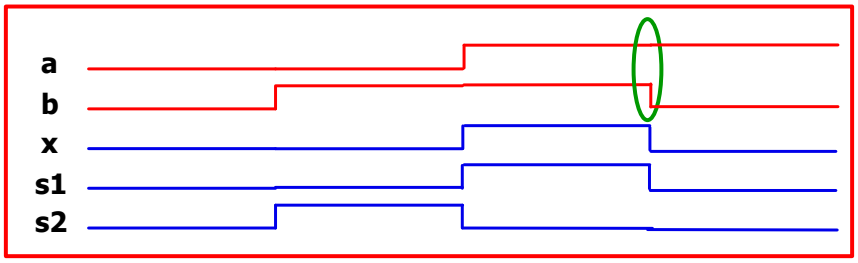
UAH
Departamento de Electrónica

Concurrencia y secuencialidad

- Asignación secuencial.




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

entity ejemplo3 is
  port(a, b    : in  std_logic;
       s1, s2  : out std_logic);
end ejemplo3;

architecture rtl _3 of ejemplo3 is
begin

 process(a, b)
   variable x    :      std_logic;
 begin
   x    := a and b;  ✓
   s1   <= x;        ✓
   s2   <= x xor b;  ✓
 end process;

end rtl _3;
```

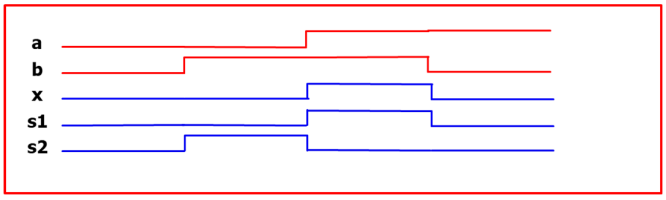


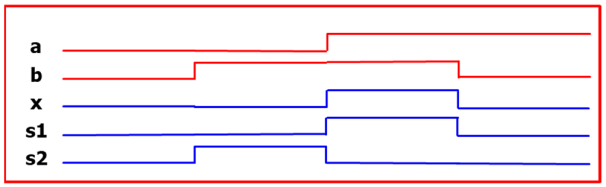
Comparativa

```
architecture rtl of ejemplo1 is
  signal x   :     std_logic;
begin

   x   <= a and b;
   s1  <= x;
   s2  <= x xor b;
end rtl;
```

```
architecture rtl _3 of ejemplo3 is
begin

 process(a, b)
   variable x   :     std_logic;
 begin
   x   := a and b;
   s1  <= x;
   s2  <= x xor b;
 end process;

end rtl _3;
```

¡Este código es correcto!

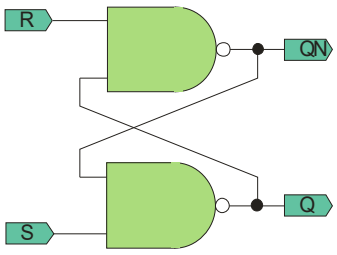




¡En un proceso, para almacenar un valor que después se va a examinar se utilizan variables !

¡Las variables son locales al procesos en el que se declara!

Ejercicio.




```
library ieee;
use ieee.std_logic_1164.all;
entity rs is
  port(R, S : in  std_logic;
       Q    : out std_logic;
       QN   : out std_logic );
end entity;

architecture arq1 of rs is
  signal aux_q, aux_qn : std_logic;
begin
  aux_q  <= S nand aux_qn;
  aux_qn <= R nand aux_q;

  Q  <= aux_q;
  QN <= aux_qn;
end arq1;
```

VHDL-93

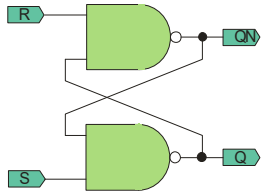
```
library ieee;
use ieee.std_logic_1164.all;
entity rs is
  port(R, S : in    std_logic;
       Q    : inout std_logic;
       QN   : inout std_logic );
end entity;

architecture arq1 of rs is

begin
  Q  <= S nand QN;
  QN <= R nand Q;
end arq1;
```

VHDL-08

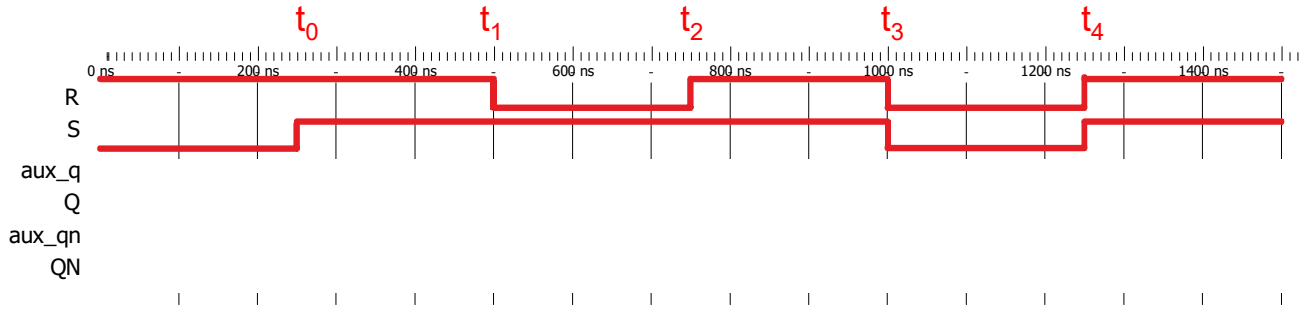
UAH

Departamento de Electrónica

- Ejercicio.




```
library ieee;
use ieee.std_logic_1164.all;
entity rs is
  port(R, S : in  std_logic;
       Q    : out std_logic;
       QN   : out std_logic );
end entity;

architecture arq1 of rs is
  signal aux_q, aux_qn : std_logic;
begin
  aux_q  <= S nand aux_qn;
  aux_qn <= R nand aux_q;

  Q  <= aux_q;
  QN <= aux_qn;
end arq1;
```

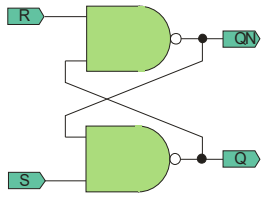


Completa el cronograma

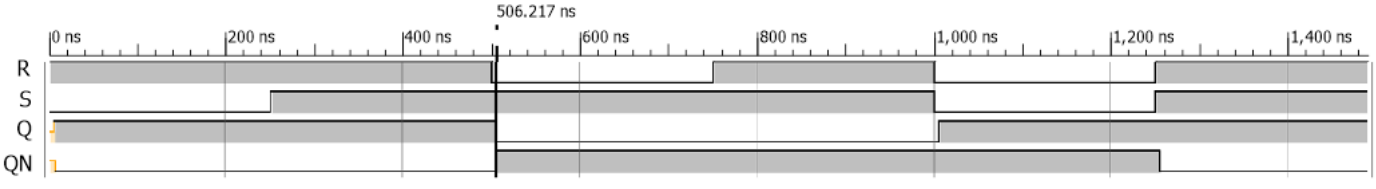
UAH

Departamento de Electrónica

- Ejercicio.

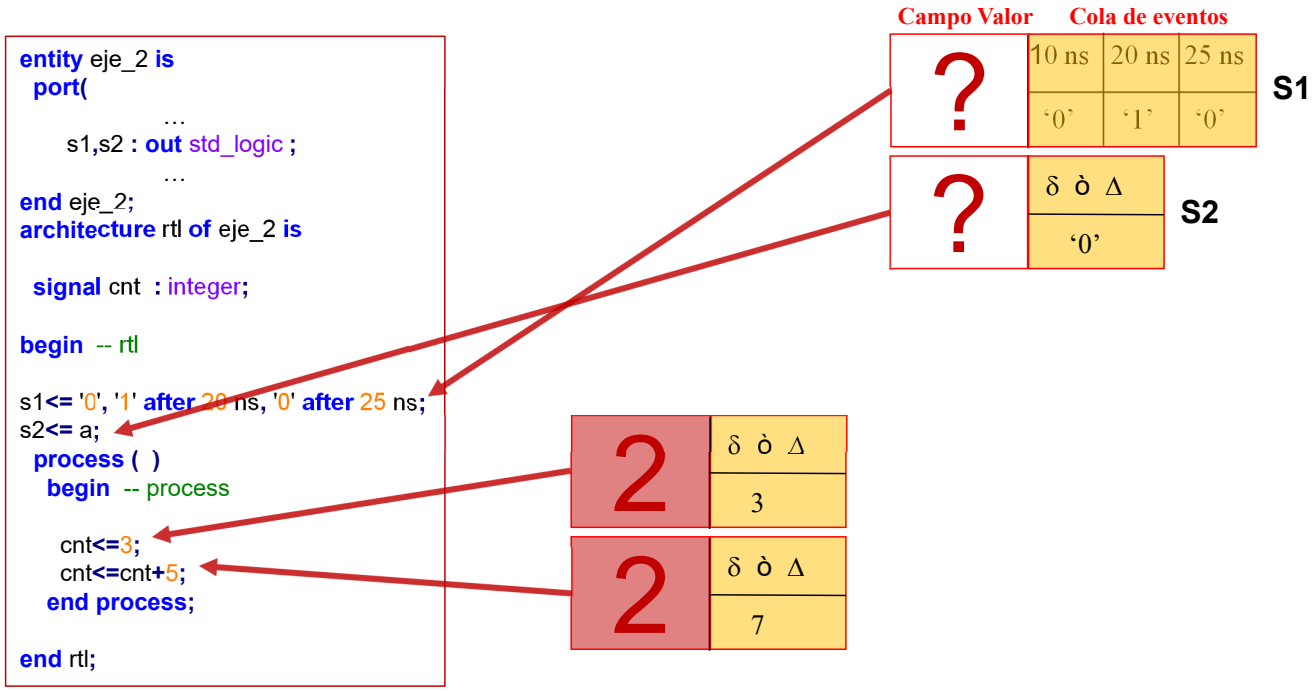


Simulación temporal



Concurrencia y secuencialidad

- La concurrencia se consigue por la asignación diferida de las señales (*signal*) y puertos (*port*)



Otros elementos del lenguaje

- Objetos.
  - Son elementos que toma un valor de algún tipo de dato.
  - Todo objeto se deben declarar antes de ser utilizado.

- Tipos:
  - Constantes. Son objetos que mantiene siempre su valor inicial. Son de sólo lectura
  - Variables. Son objetos que permite almacenan valores que pueden cambiar, mediante una sentencia (secuencial) de asignación. Son de lectura/escritura.
  - Señales. Son objetos que permite almacenan valores que pueden cambiar, mediante una sentencia (concurrente o secuencial) de asignación. Son de lectura/escritura

Constantes

- Es un objeto que mantiene siempre su valor inicial.
- Su valor no puede modificarse.
- Se emplean para asignar un valor a un identificador.
- Facilitan la legibilidad y depuración del código.
- Se pueden declarar en la arquitectura o en los procesos, de ello depende su visibilidad.

```
constant identificador{,..}: tipo [:= valor_inicial];
```

```
architecture rtl of dis is
 constant ROMsize: integer := 10;
 constant bus_dir: std_logic_vector(7 downto 0):="10010111";

begin
. . .

process(. . . )
  constant bus_dat: std_logic_vector(7 downto 0):=x"33";
  constant ROM_Width: integer := ROMsize*2;

 begin
. . .

end process;
. . .
```


Variables

- Almacenan valores que pueden cambiar
- Proporciona un mecanismo para un almacenamiento local.
- Tienen una asignación inmediata;
- No tiene analogía en el hardware.
- Sólo se pueden utilizar en estructuras secuenciales: dentro procesos.
- Se declaran el la parte declarativa de procesos.
- Son locales al proceso en el que se declara.

```
variable identificador{,..}: tipo [:= valor_inicial];
```

```
architecture rtl of eje is
. . .
begin

process(. . .)
 variable cnt_1,cnt_2 : natural;
 variable adrs : std_logic_vector (3 downto 0) := "0000";
 variable data : std_logic_vector (0 to 15);
 variable aux:   std_logic;
. . .
  begin
. . .

end process;
```

En la síntesis se ignora el valor inicial

NO SE DEBE HACER

Señales

- Permiten que se modifique su valor.
- Tienen una analogía directa con el hardware. Modela una conexión física.
- Se declaran en la parte declarativa de las arquitecturas.
- Son visibles en la arquitectura en que se declaran.
- Se utilizan como elemento de comunicación entre procesos.
- Puede recibir asignaciones concurrentes o secuenciales .
- La asignación de las señales es diferida .
  - Llevan asociados una lista o cola)de eventos con el conjunto de posibles valores futuros a tomar.
  - La asignación no se hace efectiva hasta que todos los procesos terminan el ciclo actual de simulación
- Los puertos de una entidad se consideran señales.

```
signal identificador{,..}: tipo [:= valor_inicial];
```

```
architecture rtl_1 of mux4_1_if is
signal rst : std_logic;
signal clk_1,clk_2,clk_3 : std_logic := '0';
. . .
begin
```

En la síntesis se ignora el valor inicial.

NO SE DEBE HACER

Señales

- Mal uso de las señales.
  - Señales que son constantes.

```
signal valor_1: std_logic_vector(3 downto 0);
 ---
begin
---
valor_1<="1010";
```

¡Esto nunca se debe hacer!

  - Asignación innecesaria de un puerto de entrada.

```
entity ejemplo is
  port( SEL: in  std_logic_vector(1 downto 0);
    . . .
end ejemplo;

architecture rtl of ejemplo is
signal aux : std_logic_vector(1 downto 0);
 . . .
begin
 . . .
aux <= SEL;

  process(. . .)
 . . .
  begin
    if aux = "00" then
      . .
  end process;
. . .
```

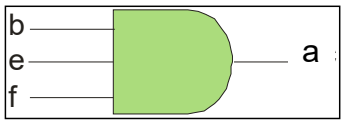
¡Esto es innecesario!

Asignación a señal/variable

¡Este es el código correcto!

```
architecture rtl of ejemplo is
  signal a, b, e, f : std_logic;
  . . .
begin

  process(b,e,f)
    variable c : std_logic;
  begin
    a <= b and c;
    c := e and f;
  end process;
```

```
architecture rtl of ejemplo is
  signal a, b, e, f : std_logic;
  . . .
begin

process(b,e,f)
    variable c : std_logic;
  begin
    c := e and f;
    a <= b and c;
  end process;
```



¡Volveremos a hablar de esto!

```
architecture rtl of ejemplo is
  signal a, b, c, e, f : std_logic;
  . . .
begin

  process(b,e,f)
  begin
    a <= b and c;
    c <= e and f;
  end process;
```

```
architecture rtl of ejemplo is
  signal a, b, c, e, f : std_logic;
  . . .
begin
process(b,e,f)

  begin
    c <= e and f;
    a <= b and c;
  end process;
```


Tipos de sentencias

❑Sentencias concurrentes:

- Se ejecutan al mismo tiempo (en paralelo).
- No hay prioridad entre unas y otras.
- Describen paralelismo.
- El comportamiento es independiente del orden en el cual son escritas.
- Las sentencias concurrentes equivalen a procesos.
- Forman el cuerpo de las arquitecturas y bloques, algunas en las entidades

❑Sentencias secuenciales:

- Se ejecutan ordenadamente, una después de otra.
- El orden de las sentencias es relevante.
- Sólo se pueden utilizar dentro de los procesos y los subprogramas (funciones y procedimientos).

Tipos de sentencias

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

entity mux2_1 is
  port( A, B, D0, D1 : in  std_logic;
        s            : out std_logic);
end mux2_1;

architecture RTL of mux2_1 is
  signal sel : std_logic;
begin

  sel <= A xor B;

  process(sel, D0, D1)
  begin
    if sel = '0' then
      S <= D0;
    else
      S <= D1;
    end if;
  end process;
end RTL;
```

Asignación a señal = concurrente

Proceso = concurrente

If= secuencial

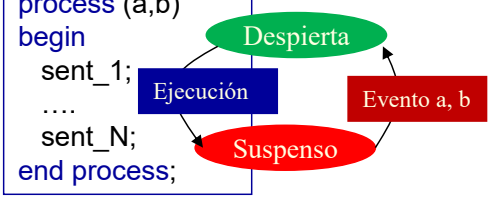
Asignación a señal= secuencial

Procesos

- Son sentencias concurrentes que definen su comportamiento a través de sentencias secuenciales.
- No hay restricciones en cuanto al número de procesos en una arquitectura.
- El número de líneas/sentencias de un proceso viene limitado por su legibilidad.
- Los procesos se ejecutan (despiertan) cuando se producen eventos en las señales/puertos presentes en la lista de sensibilidad.
- El proceso se suspende al alcanzar el final.
- Los objetos (variables o constantes) declarados en un proceso son locales a él.
- Las variables no pierden su valor una vez suspendido el proceso.
- Las variables declaradas en un proceso son locales a ese proceso

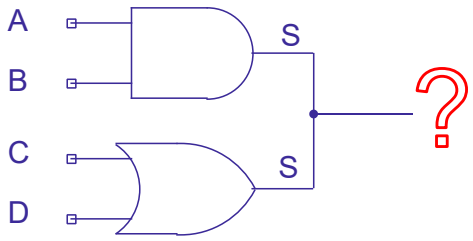
```
[etiqueta: ] process ( lista_sensibilidad)  [ is ]
   declaración_constantes
   declaración_variables
begin
   {sentencias_secuenciales}
end process  [ etiqueta] ;
```

```
process (a,b)
begin
  sent_1;
  ….
  sent_N;
end process;
```



❑ Errores de modelado

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

architecture rtl of driver_error is
  signal A, B, C, D, S : std_logic ;

begin

  process (A, B)
  begin
    S <= A and B;
  end process;

  process (C, D)
  begin
    S <= C or D;
  end process;

end rtl;
```




```
library ieee;
use ieee.std_logic_1164.all;
architecture rtl of driver_error is
  signal A, B, C, D, S : std_logic ;

begin

  S <= A and B;
  S <= C or D;
end rtl;
```

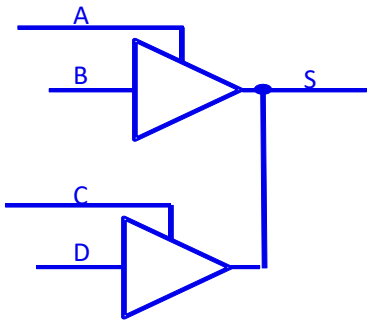
¡Desde dos o mas sentencias concurrentes no se le puede asignar valores a una señal o puerto!

❑ Errores de modelado

```
process (clk)
 begin  -- process

   if clk'event and clk = '1' then
     if ce = '1' then
       if direction = '1' then
         aux <= aux-1;
       else
         aux <= aux+1;
       end if;
     end if;
   end if;
 end process;

 process (rst)
 begin  -- process
   if rst = '1' then
     aux <= x"0";
   end if;
 end process;
```

- Errores de modelado

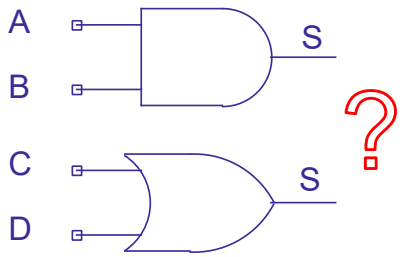
```
library ieee;
use ieee.std_logic_1164.all;
...
architecture rtl of HZ is
  signal A, B, C, D, S : std_logic;
...
begin
...
  process (A, B)
  begin
    if A = '1' then
      S <= B;
    else
      S <= 'Z';
    end if;
  end process;
...
  process (C, D)
  begin
    if C = '1' then
      S <= D;
    else
      S <= 'Z';
    end if;
...
  end process;
```



¡El código es correcto pero:

A y C no pueden ser las dos '1' a la vez!


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

architecture rtl of driver_error is
  signal A, B, C, D, S : std_logic ;

begin

  process (A, B,C,D)
  begin
    S <= A and B;
    S <= C or D;
  end process;

end rtl;
```



¡Dentro de un mismo proceso se pueden asignar diferentes valores a una señal o puerto!

¡Tomará el último valor asignado!