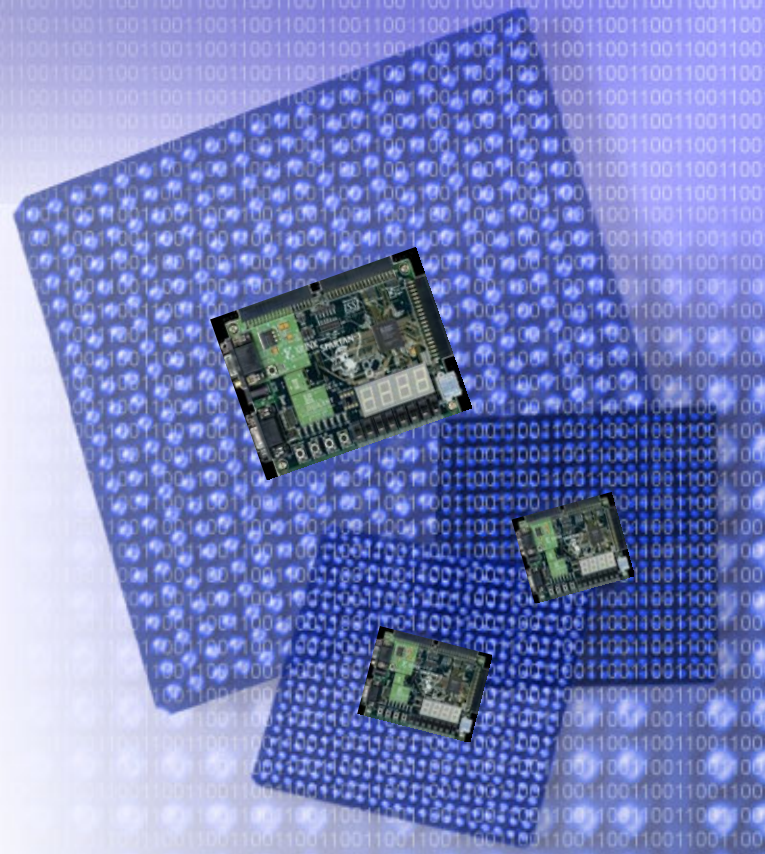
Universitat d'Alacant
Universidad de Alicante



VHDL. Lenguaje de descripción hardware

Tipos de datos

Tipos de objetos y datos

- VHDL predefine un conjunto relativamente limitado de tipos de datos, pero dispone de gran versatilidad para que el usuario los cree según sus necesidades.
- Todos los objetos tienen que declararse antes de su utilización.
- En VHDL NO se pueden asignar valores de una señal de un tipo a una señal de otro tipo.

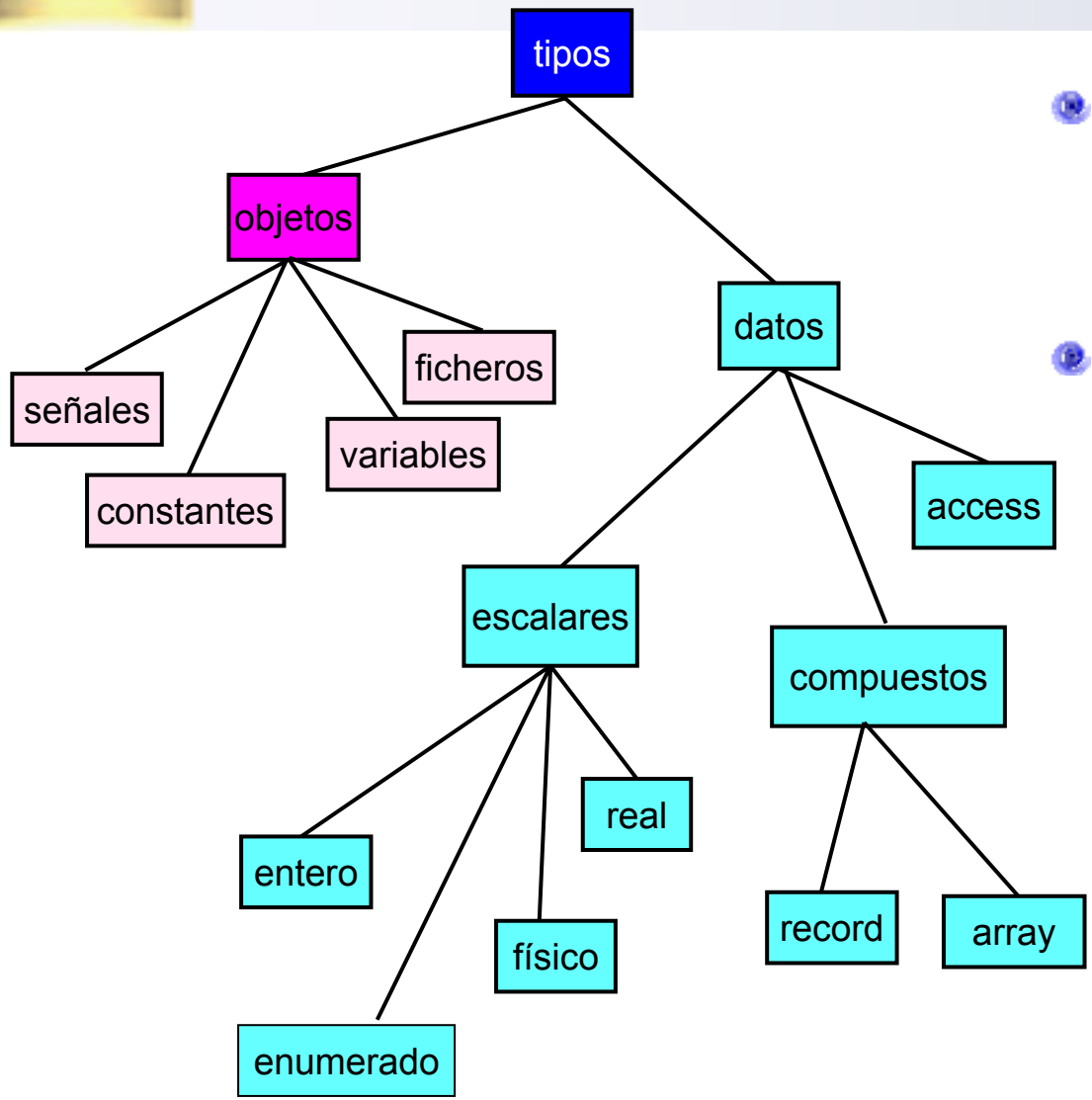
❑ **Tipos de objetos.**

- Constantes.
- Variables.
- Señales.
- Ficheros.

❑ **Tipos de datos.**

- Escalares.
  - Enumerados.
  - Enteros.
  - Físicos.
  - Coma flotante.
- Compuestos.
  - Vector/matriz.
  - Registro.
- Acceso.
  - Punteros.

Tipos



- TIPO: Es la definición de los valores posibles que pueden tomar los objetos y los datos
- VHDL es un lenguaje fuertemente tipado:
  - A los objetos se les asigna siempre un tipo cuando se declaran
  - Las asignación sólo pueden hacerse entre objetos del mismo tipo

Tipos básicos predefinidos
Tipos IEEE-1076

BIT: Puede tomar los valores ‘0’ ó ‘1’

BIT_VECTOR: Agrupación de bits

ejemplo:

```
signal salida :BIT_VECTOR (0 to 3);
        salida <=“1000”;
esto significa que
        salida(0)=‘1’
        salida(1)=‘0’
        salida(2)=‘0’
        salida(3)=‘0’
```

```
signal salida :BIT_VECTOR (3 downto 0);
        salida <=“1000”;
esto significa que
        salida(3)=‘1’
        salida(2)=‘0’
        salida(1)=‘0’
        salida(0)=‘0’
```

diferencia

Tipos básicos predefinidos
Tipos IEEE-1076

INTEGER: A veces utilizado para índices de loops, constantes, valores genéricos, etc

BOOLEAN: Pueden tomar los valores 'true' ó 'false'

ENUMERATED: Enumeración de valores definidos por el usuario

por ejemplo:

```
type estado is (inicio, arriba, abajo, stop)
```

Tipo STD_LOGIC

- Los dos valores del tipo bit se quedan cortos para modelar todos los estados que puede tomar una señal digital.
- El paquete IEEE.standard_logic_1164 define el tipo **std_logic**, que representa todos los posibles estados de una señal:

  U No inicializado, valor por defecto.

  X Desconocido fuerte, salida con múltiples fuentes en corto

  0 Salida de una puerta con nivel lógico bajo

  1 Salida de una puerta con nivel lógico alto

  Z Alta Impedancia

  W Desconocido débil, terminación de bus

  L 0 débil, resistencia de pull-down

  H 1 débil, resistencia de pull-up

  – No importa, usado como comodín para síntesis

Tipo STD_LOGIC

- Para describir buses se utiliza el tipo **std_logic_vector**, que es un array de **std_logic**
- Los tipos **std_logic** y **std_logic_vector** son los estándares industriales.
- Todos los valores son validos en un simulador VHDL, sin embargo solo: ‘0’, ‘1’, ‘Z’, ‘L’, ‘H’ y ‘–’ se reconocen para la síntesis.
- En el paquete IEEE.std_logic_1164 aparecen otros dos tipos: **std_ulogic** y **std_ulogic_vector**. Son los mismos, pero sin haber pasado por la función de resolución
  - Esta función decide cuál debe ser el valor de la señal cuando tiene dos fuentes que le asignan valores distintos
  - Por ejemplo, si una fuente asigna un ‘1’ y la otra una ‘L’, la función de resolución dice que la señal se queda a ‘1

Utilizando los tipos de objeto: Constantes

Constantes.

- Mantienen el valor, del tipo de dato especificado, durante toda la ejecución.
- Se declaran en la parte declarativa, antes del **“begin”**

```
ARCHITECTURE eps OF mi_prueba IS
   CONSTANT const1  : STD_LOGIC := ‘1';
   CONSTANT retardo : TIME := 15 ns;
   CONSTANT const3  : INTEGER := 8;
BEGIN
```

- Las constantes pueden ser de cualquier tipo

Utilizando los tipos de objeto: Variables

Variables.

- Almacenan valores del tipo de datos especificado, que pueden cambiar en la descripción.
- Sólo pueden declararse y utilizarse **dentro** de procesos o subprogramas, (aunque se pueden compartir mediante SHARED, no es muy recomendable)
- Se les puede asignar un valor inicial, si no se hace así, el descriptor le asigna el **menor** valor del tipo declarado
- La asignación se realiza con :=

  Ejemplo:

```
VARIABLE Estado: bit:=0;    --definición
Estado:= 1;                 --asignación
```

- No tienen significado físico directo.
  - Las **asignaciones ocurren inmediatamente** no tienen tiempo asociado

Utilizando los tipos de objeto: Variables

Variables compartidas

```
LIBRARY IEEE ;
USE IEE STD LOGIC 1164 .ALL ;
ENTITY ejemplo_variables IS
PORT ( definición de puertos) ;
END ejemplo_variables ;

ARCHITECTURE ejemplo OF ejemplo_variables IS
TYPE estados IS (rojo, verde, azul, naranja) ;
SHARED VARIABLE variable_compartida : estado;
BEGIN
proceso1 : PROCESS
BEGIN
      WAIT UNTIL variable_compartida := rojo ;
      variable_compartida := azul ;
      . . .
END PROCESS proceso1 ;
proceso2 : PROCESS
BEGIN
      WAIT UNTIL variable_compartida := v e r d e ;
       variable_compartida :=naranja
      . . .
END PROCESS proceso2 ;
END ARCHITECTURE;
```

No se recomienda el uso de este tipo de variables, ya que puede dar lugar a comportamientos no deterministas, es decir, resultados de la simulación de un mismo modelo pueden ser diferentes dependiendo de la herramienta de simulación que se utilice

Tipos de objetos. Señales

Señales.

- Almacenan valores que pueden cambiar.
- Representan las **conexiones físicas**.
- Se declaran en sentencias concurrentes y pueden aparecer también en procesos.
- Una señal mantiene una lista de valores.
  - La lista incluye su valor actual y un conjunto de posibles valores futuros.
- Las asignaciones no son instantáneas, se actualizan en el siguiente paso de simulación (sentencia WAIT o si ha terminado una sentencia concurrente).
- Sirven para comunicar procesos e interconectar componentes.
  - Si no se especifica un retardo (mediante sentencia AFTER), se aplica automáticamente un retardo de tipo delta (δ).

Utilizando los tipos de objeto: Señales

- El objeto básico en VHDL es la señal, que se utiliza para modelar los hilos del circuito
- Puesto que modela nodos físicos, incluye información de tiempo
  - No sólo contiene unos valores ('0', '1', 'Z', etc...) sino también el tiempo en el que se toman estos valores
- Se declaran antes del begin de la arquitectura (en la parte declarativa):

```
ARCHITECTURE eps OF prueba IS
   SIGNAL s1 : STD_LOGIC;
   SIGNAL s2 : INTEGER;
BEGIN
```

- Pueden tener un valor inicial (no soportado en síntesis)

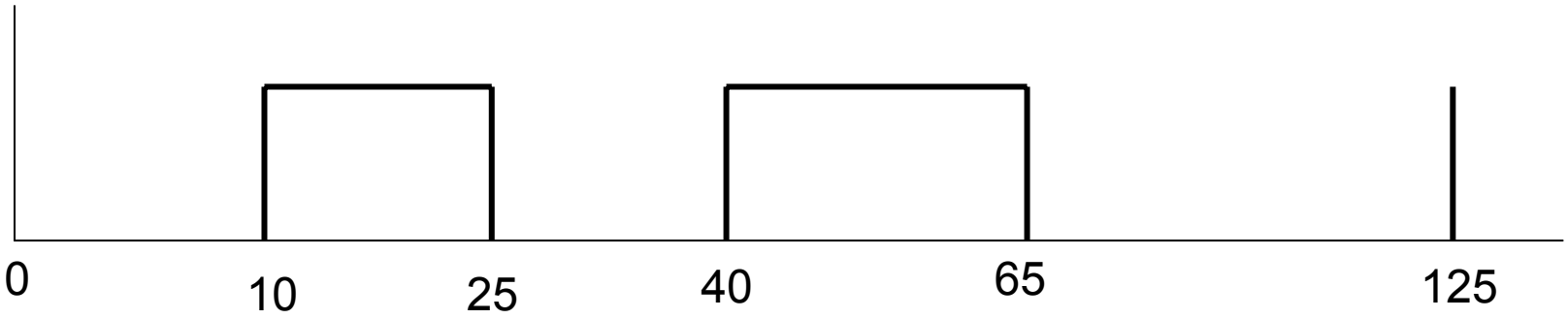
```
SIGNAL a : STD_LOGIC := '0';
```

- Para asignar valores a una señal se utiliza <=

Caracterización de las señales. Driver

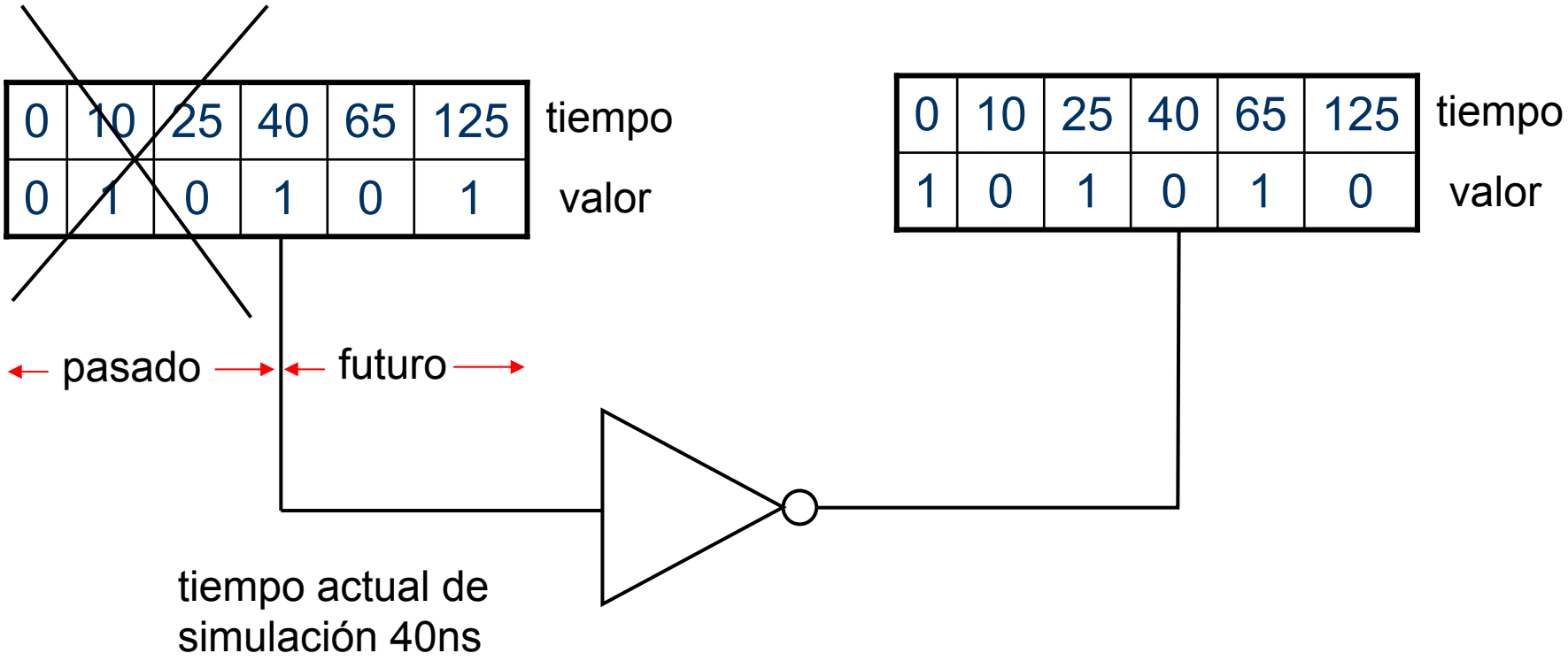
- ***Driver***. Es una cola (tabla) de transacciones que almacena la forma de onda de la señal
  - El driver ***proyecta*** los valores futuros de la señal, es decir, la señal está ***planificada*** para tomar el valor indicado en su correspondiente momento
  - Asignaciones diferentes de la misma señal comparten el mismo driver

0	10	25	40	65	125	tiempo
0	1	0	1	0	1	valor



Caracterización de las señales. Driver

Cuando avanza el tiempo de simulación las transacciones ya procesadas se eliminan de la cola



Modelo de retardos en señales

- Las señales permiten incluir información de los retardos para que los modelos se parezcan lo más posible al hardware.
- Existen tres modelos:
  - **Transporte**. Describe el comportamiento de una línea de transmisión ideal.
  - **Inercial**. Describe los circuitos reales.
  - **Delta**. Es el retardo por defecto.

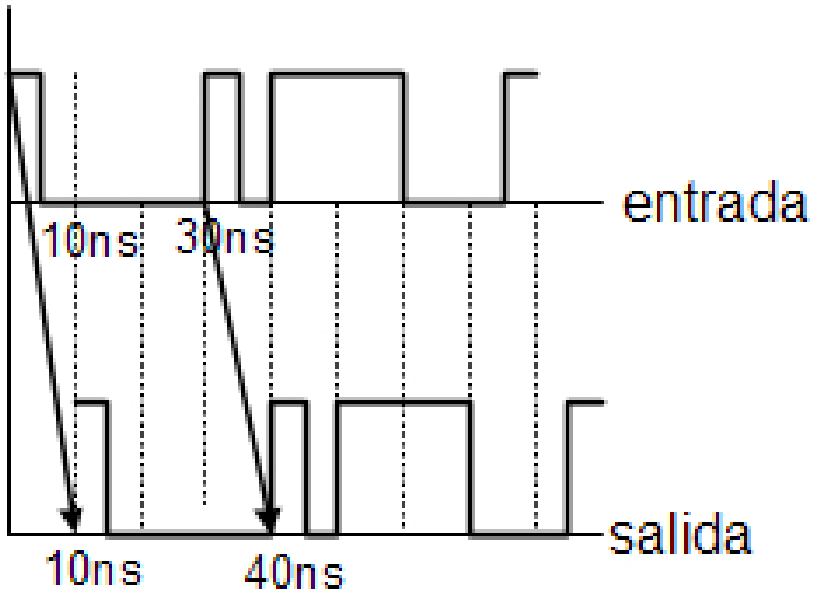
Modelo de retardos. Transporte

- **Transporte**. Permite modelar dispositivos que presentan un comportamiento en frecuencia casi infinito:
  - La entrada se propaga a la salida sin ninguna alteración.
  - No importa lo pequeña o lo grande que sea la duración de la entrada.
  - Este comportamiento es típico de las líneas de transmisión.

nom_señal <= TRANSPORT [expresión] AFTER tiempo;

Modelo de retardos. Transporte

salida <= **TRANSPORT** entrada **AFTER** 10ns;



Modelo de retardos. Inercial

- **Inercial**. Permite modelar el comportamiento temporal de la conmutación de los circuitos:
  - Una entrada debe tener un valor estable durante un cierto tiempo antes de que se propague a la salida.
  - Si la entrada no permanece estable durante el tiempo especificado, la salida no se ve afectada por la entrada, es decir, la entrada se ignora.

nom_signal_out <= [[REJECT tiempo] INERTIAL]
[expresión] nom_signal AFTER tiempo;

Modelo de retardos. Inercial

- Si no se incluye la opción REJECT, el límite para rechazar el pulso es igual al valor del retardo especificado.
- El retardo inercial es el que se considera por defecto.

ejemplo:

```
salida <= INERTIAL entrada AFTER 5ns;
salida <= entrada AFTER 5ns;
```

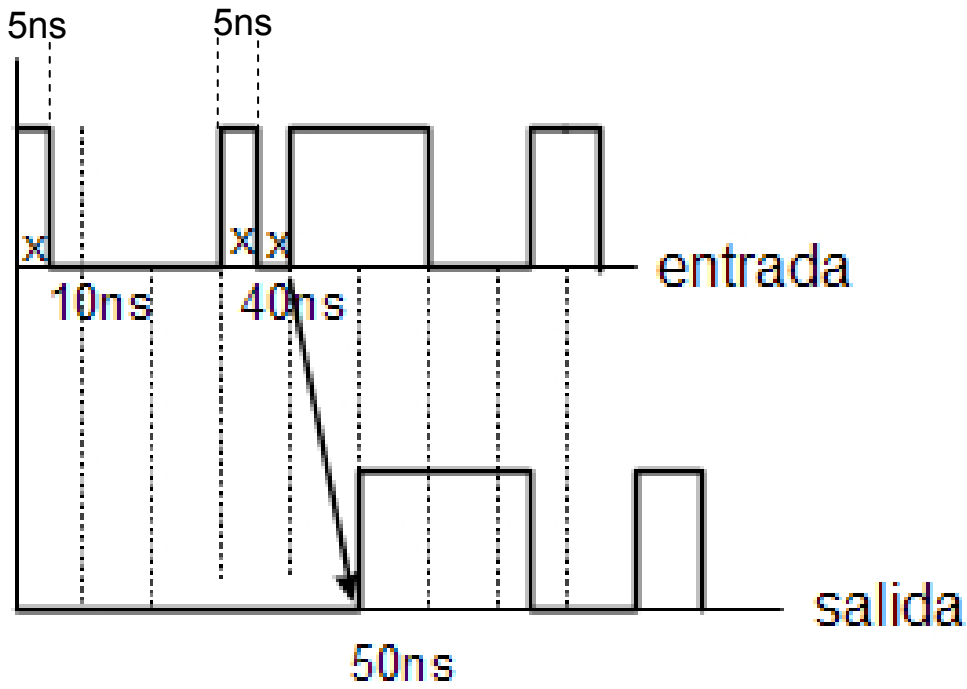
son expresiones equivalentes

```
salida <= REJECT 7ns INERTIAL entrada AFTER 10ns;
```

propaga con un retardo de 10ns los pulsos de la señal de entrada, eliminando aquellos que no tengan una duración superior a 7ns

Modelo de retardos. Inercial

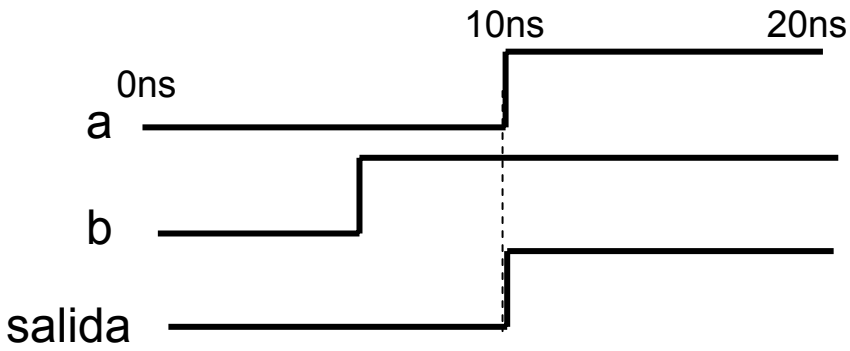
```
salida <= REJECT 7ns INERTIAL entrada AFTER 10ns;
```



Modelo de retardos. Delta

- **Delta**. (δ). Es el modelo que se asume por defecto cuando no se especifica un retardo específico para la señal.
  - Es un retardo infinitesimal, siempre despreciable respecto del tiempo de simulación.

salida <= a and b;



Resumen de señales y variables

- Recordad ….

  Las asignaciones a señales dentro de procesos sólo se ejecutan cuando se suspende el proceso.

- Su explicación …
  - La señales modelan conexiones físicas, y por tanto, no sólo deben tener en cuenta el valor, sino también el tiempo.
  - Para que un cable, conexión física, cambie de valor hace falta que el tiempo avance.
  - De la misma forma, para que una señal cambie de valor hace falta que el tiempo avance.
  - El tiempo sólo avanza cuando se suspende el proceso.

Resumen de señales y variables

- Sin embargo, a la hora de modelar un circuito nos puede venir bien tener un objeto cuyo valor se actualice inmediatamente.

  Sin tener que esperar a que avance el tiempo, como en las señales

- La solución son las variables
  - La variables se declaran dentro de los procesos
  - Sólo se ven dentro del proceso que las ha declarado
  - Toman el valor inmediatamente, son independientes del tiempo

Resumen de señales y variables

	Señales	Variables
Sintaxis	destino <= fuente	destino:= fuente
Utilidad	modelan nodos físicos del circuito	representan almacenamiento local
Visibilidad	global (comunicación entre procesos)	local (dentro del proceso)
Comportamiento	se actualizan cuando avanza el tiempo	se actualizan inmediatamente

Tipos de datos

- **Tipos de datos escalares.**
  - **Enumerados.**
    - Conjuntos ordenados de identificadores o caracteres definidos por el usuario, útiles para definir los estados de una máquina de estados finitos.

```
TYPE nombre_del_tipo IS (valor1, valor2,..., valorN);
TYPE variosLogicos IS ('X', '0', '1', 'Z');
TYPE semaforo IS (rojo, verde, ambar, apagado);
```

Ejemplos de declaración de objetos a partir de la declaración de tipos.

```
SIGNAL estadoActual: semaforo:= apagado;
VARIABLE logicaMitad: variosLogicos;
```

Tipos de datos

- **Tipos de datos escalares.**
  - **Enteros.**
    - Definidos en el paquete estándar

```
TYPE integer IS RANGE -2147483648 TO 2147483647;
```

    - Tienen asociados las operaciones matemáticas de suma (+), resta (-), multiplicación (*) y división (/).
    - El rango puede ser especificado en orden:
      - Ascendente: (límiteInferior TO límiteSuperior).
      - Descendente: (límiteSuperior DOWNTO límiteInferior).

Tipos de datos

- **Tipos de datos escalares.**
  - **Enteros**
    - Definidos por el usuario

```
TYPE nombre IS RANGE intervalo rango de enteros;
```

Ejemplos:

```
TYPE diasMes IS RANGE 31 DOWNTO 1;
TYPE diasSemama IS RANGE 1 TO 7;
TYPE diasAnno IS RANGE 1 TO 365;
```

Ejemplos de declaración de objetos a partir de la declaración de tipos.

```
SIGNAL aniversario: diaMes:= 22;
ENTITY fiestasPuentes IS
    PORT( mes      : IN diasMes;
          dia      : IN diasSemana;
          fiestas  : OUT diasAnno);
END fiestasPuentes;
```

Tipos de datos

- **Tipos de datos escalares.**
  - **Físico**
    - Es un tipo enumerado que se utiliza para representar magnitudes físicas (tiempo, distancia, capacidad).
      - Tienen un valor y unas unidades.
      - Internamente son considerados como enteros.

```
TYPE nombreMagnitud IS RANGE restricción de rango
UNITS
    Identificador= valores unidades;
END UNITS;
```

Tipos de datos

- Tipos de datos escalares.
  - Físico

Ejemplos:

```
TYPE tiempo IS RANGE 0 TO 10000000;
UNITS
    ns;              -- nanosegundo
    us = 1000 ns;    -- microsegundo
    ms = 1000 us;    -- milisegundo
    s  = 1000 ms;    -- segundo
END UNITS;
TYPE frecuencia IS RANGE 0 TO 10000000;
UNITS
    Hz;              -- Herzios
    KHz = 1000 Hz;   -- Kiloherzios
    MHz = 1000 KHz;  -- Megaherzios
    GHz = 1000 MHz;  -- Gigaherzios
END UNITS;
```

Tipos de datos

- **Tipos de datos escalares.**
  - **Coma flotante**
    - Números reales definidos en el paquete estándar

```
TYPE real IS RANGE -1.0E38 TO 1.0E38;
```

    - Tienen asociados las operaciones matemáticas de suma (+), resta (-), multiplicación (*) y división (/).
    - Declarados de igual forma que los enteros.

    - Números reales definidos por el usuario

```
TYPE nombre IS RANGE rango de números reales;
```

Ejemplos:

```
TYPE notas IS RANGE 10.0 DOWNTO 0.0;
TYPE probabilidad IS RANGE 0.0 TO 1.0;
```

Tipos de datos

- **Tipos de datos compuestos.**
  - **Vectores y matrices**
    - Es una colección indexada de elementos que son del mismo tipo.
    - Cada elemento de un vector o una matriz puede ser accedido por uno o más índices.
    - Son especialmente útiles para modelar memorias (RAM/ROM).

```
TYPE identificador IS ARRAY rango OF tipoDatos;
```

Tipos de datos

Ejemplos:

```
TYPE fila IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- array 1D
```

0	0	1	1	0	0	1	1

```
TYPE array1x1 IS ARRAY (0 TO 2) OF fila; -- array 1Dx1D
```

0	0	1	1	0	0	1	1
0	1	0	0	0	0	1	1
1	0	1	1	1	0	0	0

```
TYPE array2D IS ARRAY (0 TO 2, 7 DOWNTO 0) OF STD_LOGIC;
```

0	1	0	1	1	1	0	0
1	1	0	1	0	0	0	1
1	0	1	0	1	1	1	0

Tipos de datos

- **Subtipos.**
  - Se utilizan para definir subconjuntos ya declarados en un tipo de datos.
    - Son útiles para detectar si un objeto toma valores dentro del rango esperado.
  - Un tipo y un subtipo se consideran el mismo tipo de datos y pueden mezclarse en una operación.

SUBTYPE identificadorSubtipo IS identificadorTipo [RANGE valor1 O/DOWNTO valor2]

Ejemplo:

```
TYPE hexadecimal IS ('0', '1', '2', '3', '4',
 '5', '6', '7', '8', '9', 'A', 'B' , 'C', 'D',
 'E', 'F');
SUBTYPE octal IS hexadecimal RANGE '0' TO '7';
```

Uso de arrays para crear buses

- Los vectores se pueden definir tanto en rangos ascendentes como descendentes:

```
SIGNAL x: STD_LOGIC_VECTOR(0 TO 3);     --ascen.
SIGNAL y: STD_LOGIC_VECTOR(3 DOWNTO 0); --descen.
x <= “1010";
y <= “1010";
```

Produce como resultado:

x(3) = ‘0'; x(2) = ‘1'; x(1) = ‘0'; x(0) = ‘1';

y(3) = ‘1'; y(2) = ‘0'; y(1) = ‘1’; y(0) = ‘0';

- Una manera rápida y eficiente de asignar valores a vectores son los agregates:

```
a <= (0 => ‘0’, 1 => c and d, others=> ‘Z‘);
```

Asignación de señales en buses

- Permite una gran flexibilidad

ejemplo:

```
SIGNAL temporal STD_LOGIC_VECTOR(7 downto 0)
```

Todos los bits:
```
temporal <= “10110110”;
temporal <= x”B6”; --en hexadecimal
```

Un solo bit:
```
temporal(3) <=‘1’;
```

Un rango de bits:
```
temporal(3 downto 0) <=“0110”;
```

NOTA:

- **1 bit: comilla simple (‘)**
- **múltiples bits: comilla doble (“)**

Cómo definir nuevos tipos de datos

- VHDL permite definir nuevos tipos, bien a partir de tipos enumerados, o como subconjunto de tipos ya existentes, o tipos multidimensionales
- Las definiciones de tipos se deben hacer en la parte declarativa de la arquitectura
- Definir un tipo como una enumeración:

```
TYPE estados IS (inicio, cambiar, sumar, salir);
SIGNAL mi_maquina : estados;
```

- Definir un tipo bidimensional:

```
TYPE memoria IS ARRAY (1024 downto 0) OF
std_logic_vector(7 downto 0);
SIGNAL mi_memoria : memoria;
```

Operadores. Aritméticos

Aritméticos

Operación	Operador	Tipo datos	Explicación
Suma	+	cualquier tipo numérico	
Resta	-	cualquier tipo numérico	
Producto	*	entero, real	
División	/	entero, real	
Exponenciación	**	entero, real (exponente entero)	
Módulo	mod	ambos enteros	(X mod Y) = X – Y·N signo(X mod Y)=signo Y
Resto	rem	ambos enteros	(X rem Y) = X – pe[X/Y]·Y signo(X rem Y)=signo X
Valor absoluto	abs	cualquier tipo numérico	

Operadores. Relacionales

Devuelven un literal del tipo *boolean*, es decir, el resultado de la operación puede ser *true* o *false*

Operación	Operador	Tipo operando
Igual	=	cualquier tipo
Diferente	/=	cualquier tipo
Mayor	>	cualquier tipo
Mayor o igual	>=	cualquier tipo
Menor	<	cualquier tipo
Menor o igual	<=	cualquier tipo

Operadores. Lógicos

Operación	Operador
Y	and
Y negada	nand
O	or
O negada	nor
O exclusiva	xor
O exclusiva negada	xnor
negación	not
Concatenación	&

Operadores. Desplazamiento

Operación	Operador	Ejemplo	Explicación
Desplazamiento lógico a la izquierda	sll	“000111” sll 2 → “011100”	por la derecha se introducen 0’s
Desplazamiento lógico a la derecha	srl	“101100” srl 4 → “000010”	por la izquierda se introducen 0’s
Desplazamiento aritmético a la izquierda	sla	“000111” sla 2 → “011111”	desplaza a la izda recirculando el bit de mayor peso e introduce por la derecha *ope’right*
Desplazamiento aritmético a la derecha	sra	“011000” sra 3 → “000011”	desplaza a la dcha recirculando el bit de mayor peso e introduce por la izquierda *ope´left*
Desplazamiento circular a la izquierda	rol	“100111” rol 2 → “011110”	los bits que salen por la dcha. entran por la izda
Desplazamiento circular a la derecha	ror	“011001” ror 3 → “001011”	los bits que salen por la izda. entran por la dcha

Operadores

- No todos los operadores están definidos para todos los tipos
- En particular, para los std_logic habrá que obtenerlos de las librerías estándar:
  - std_logic_signed
  - std_logic_unsigned
  - std_logic_arith

- El operador de concatenación se utiliza muy a menudo

```
signal a: std_logic_vector( 3 downto 0);
signal b: std_logic_vector( 3 downto 0);
signal c: std_logic_vector( 7 downto 0);
a <= “1100";
b <= “0010";
c <= a & b; -- c =“11000010"
```

- Los operadores de desplazamiento básicos sólo funcionan con bit_vector. Es mucho mejor usar concatenación con std_logic

Acerca de las librerías en VHDL

- Librerías clásicas
  - std_logic_signed
  - std_logic_unsigned
  - std_logic_arith
- Las librerías **signed** y **unsigned** se deben emplean cuando se quiere que los std_logic_vector estén respectivamente en complemento a 2 o en binario natural
  - Aquí está el CONV_INTEGER
- La librería **arith** es más completa, y utiliza los tipos **signed** o **unsigned** (derivados de std_logic_vector)
  - Aquí está CON_STD_LOGIC_VECTOR
- Tendencia actual del IEEE: emplear la librería **numeric_std**
  - Pensada para trabajar con los tipos **signed** y **unsigned**
  - TO_INTEGER, TO_SIGNED, TO_UNSIGNED

Ejemplos

```
signal b : std_logic;
signal u1 : unsigned (3 downto 0);
signal s1 : signed (3 downto 0);
signal i1, i2, i3 : integer;
...
u1 <= "1001";
s1 <= "1001";
b <= 'X';
wait 10ns;
i1 <= conv_integer(u1);  -- 9
i2 <= conv_integer(s1);  -- -7 el 1001 es el -7
i3 <= conv_integer(b);   -- warning en el simulador
```

Ejemplos

```
signal u1 : unsigned (3 downto 0);
signal s1 : signed (3 downto 0);
signal v1, v2, v3, v4 : std_logic_vector (3 downto 0);
signal i1, i2 : integer;
...
u1 <= "1101";
s1 <= "1101";
i1 <= 13;
i2 <= -2;
wait for 10 ns;
v1 <= conv_std_logic_vector(u1, 4);   -- = "1101"
v2 <= conv_std_logic_vector(s1, 4);   -- = "1101"
v3 <= conv_std_logic_vector(i1, 4);   -- = "1101"
v4 <= conv_std_logic_vector(i2, 4);   -- = "1110"
```