

# Objetos de VHDL

- Un objeto en VHDL es un elemento que contiene un valor de tipo específico de dato

Objetos que se pueden manipular en VHDL y sus tipos

*-3 clases principales de objetos:*

- **SEÑALES:** similares a las señales encontradas en los esquemas. Los “ports” declarados dentro de una entidad son señales. Pueden ser declaradas como bus.
- **CONSTANTES:** Permiten definir valores permanentes
- **VARIABLES:** utilizadas solamente dentro de los “PROCESS”  
Su valor puede ser cambiado en cualquier momento

*- Una declaración de objeto comprende:*

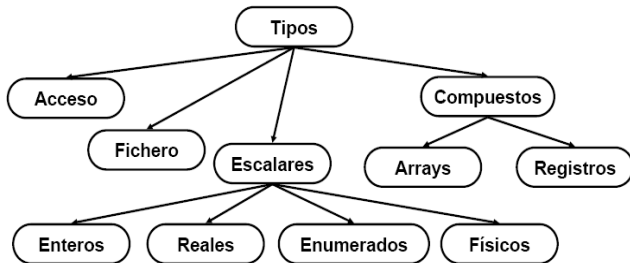
- **clase:** señal, constante o variable
- **nombre:** cualquiera excepto palabras reservadas
- **modo:** (sólo para señales en los *ports*): in, out, inout.
- **tipo:** bit, bit\_vector, boolean, sdt\_logic, integer,...

# Tipos de datos

- No existen tipos propios del lenguaje (ej. real o integer)
- Cada objeto deberá ser de un tipo concreto de dato
- Esto determinará el conjunto de valores que puede asumir y las operaciones que se podrán realizar con este objeto
- Se declaran con la siguiente sintaxis:

⇒ **type identificador is definición\_tipo;**

- ✓ Hay un conjunto de tipos pre-definidos por el sistema (integran bibliotecas que se cargan al comienzo).
- ✓ También pueden ser definidos por el usuario



➤ Tipo de datos escalares

➤ Tipos de datos compuestos

➤ Tipo de datos escalares

Sus valores están formados por una sólo unidad indivisible. Ejemplos:

- **enteros**
- **reales**
- **enumerados**
- **fisicos**

➤ Tipos de datos compuestos

Sus valores pueden dividirse en unidades atómicas más pequeñas

- **vector** : unidades atómicas del mismo tipo
- **registro** : unidades atómicas de tipo heterogéneo

### Tipo de datos escalares

**type** identificador **is range** literal **to/ downto** literal;  
**type** identificador **is** definicion\_tipo;

Ejemplo:

- **TYPE** indice **IS RANGE 7 DOWNTO 1**;
- **TYPE integer IS RANGE -2147483648 TO 2147483647**;  
-- este último está ya *Predefinido* en el lenguaje

- **TYPE** nivel **IS RANGE 5.0 DOWNTO 0.0**;
- **TYPE real IS RANGE -1.0E38 TO 1.0E38**;  
-- este último está ya *Predefinido* en el lenguaje

### Tipo de datos escalares

- Los tipos **enteros** y **reales** son tipos pre-definidos. Usaremos sólo los enteros.

• **integer**: valor entero codificado con 32 bits  
un entero puede estar limitado en su declaración, a fin de evitar utilizar los 32 bits

-----  
Ejemplo:

**signal VALOR : integer range 0 to 255**;  
**begin**  
**VALOR <= 212 when INICIO = '1' else 100**

### Tipo de datos escalares (cont.)

- Tipos **enumerados**: Se define el conjunto de posibles valores del tipo, especificando una lista. Se deben enumerar todos y cada uno.

▪ Ejemplo:

-- declaración del tipo

**type** desplazamiento **is** (arriba, abajo, derecha, izquierda);

-- declaración de una variable de este tipo

**variable** flecha: **desplazamiento :=** arriba;

tipo de dato

asignación de un valor  
inicial

### Tipo de datos escalares (cont.)

- Tipos **enumerados** (cont.)

Principalmente utilizados en síntesis para definir los estados de las máquinas de estado

-----  
architecture ARQUI of MAQ\_ESTADO is

**type ESTADOS is (REPOSO, LECTURA, ESCRITURA);**

**signal ESTADO\_ACTUAL, ESTADO\_SIGUIENTE: ESTADOS;**

-- Las señales ESTADO\_ACTUAL y ESTADO\_SIGUIENTE podrán

-- tomar los valores "REPOSO", "LECTURA" o "ESCRITURA"

**begin**

-- siguen las asignaciones

## Tipo de datos escalares (cont.)

### ▪ Tipo **enumerados** (cont.)

Tipos pre\_definidos principales:

- **boolean** puede tomar los valores **true** ó **false**
- **bit**: puede tomar el valor **'0'** ó **'1'**

type boolean is (false, true);

type bit is ('0', '1');

} no hace falta esta  
declaración. Ya  
está hecha

## Tipo de datos escalares (cont.)

Resumen de los tipos de datos pre-definidos en VHDL:

- BIT
- BOOLEAN
- INTEGER
- REAL

## Tipo de datos escalares (cont.)

El tipo bit puede resultar insuficiente, por no tener en cuenta las propiedades eléctricas de las señales. *IEEE* ha estandarizado un paquete llamado *std\_logic\_1164*

- **STD\_LOGIC** : Extensiones del tipo BIT, pueden tomar 9 valores diferentes:

'U'	--no inicializado
'X'	--forzando valor desconocido (en simul.: conflicto)
'0'	--forzando 0
'1'	--forzando 1
'Z'	--alta impedancia - tres estados
'W'	--valor débil desconocido
'L'	--0 débil
'H'	--1 débil
'-'	--sin importancia (don't care)

(Extraído del código fuente del package "STD\_LOGIC\_1164")

### Tipo enumerado del paquete *std\_logic\_1164* de la librería *IEEE*.

```
library IEEE;
use IEEE.std_logic_1164.all

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
                    );
```

- Valores lógicos: '0', '1', 'L' y 'H'.
- Valores metalógicos: 'U', 'X', 'W', '-'

std\_logic: Tipo resuelto derivado de std\_ulogic.

## Tipo de datos escalares (cont.) Modelo de resolución

- ◆ Basado en *fuerzas y valores lógicos*.
- ◆ El estado de una línea en un momento dado es una combinación de una fuerza y un nivel lógico.
- ◆ Los niveles lógicos básicos son tres: 0; 1; y X (desconocido)
- ◆ Las fuerzas pueden ser *S, R, Z* e *I*.

## Tipo de datos escalares (cont.) Modelo de resolución

- ◆ La fuerza S (Strong) es una salida con conexión a alimentación o tierra a través de un transistor → *buffer tótem-pole*.
- ◆ La fuerza R (Resistiva), es una salida con conexión a alimentación o tierra a través de una resistencia (pull\_up, pull\_down) → *buffer colector abierto*.
- ◆ La fuerza Z (alta impedancia) es una salida con una conexión a alimentación o tierra a través de una alta impedancia, → *buffer triestado*.
- ◆ La fuerza I (indeterminada) sirve para indicar que no se sabe que fuerza hay en la línea.

## Tipo de datos escalares (cont.) Modelo de resolución

- ◆ En una contención (dos señales contienden por una línea):
  - Se mira primero el valor de la fuerza ( $S > R > Z$ ). La línea toma la fuerza y valor lógico de la señal ganadora.
  - Si las fuerzas son iguales, se mira el valor lógico, si son iguales, la línea toma el valor de cualquiera de ellos.
  - Si son valores lógicos distintos, las opciones son varios según la lógica que implemente el bus.

## Tabla de resolución

<i>Std_logic</i>	<i>Fuerza-Valor</i>	<i>Descripción</i>	<i>Puerta típica</i>
'U'		No inicializado	
'X'	<i>SX</i>	Desconocido	<i>tótem-pole</i>
'0'	<i>S0</i>	0 fuerte	<i>tótem-pole</i>
'1'	<i>S1</i>	1 fuerte	<i>tótem-pole</i>
'Z'	<i>Z0, Z1</i>	Alta impedancia	triestado
'L'	<i>R0</i>	0 resistivo	pull_down
'H'	<i>R1</i>	1 resistivo	pull_up
'-'		No importa	

### Tipo de datos escalares (cont.)

El tipo STD\_LOGIC da una mayor potencia operacional que el tipo BIT, tanto para la simulación como para la síntesis.

*Para utilizar estos tipos de datos, debemos declarar la utilización de la biblioteca IEEE que contiene el package particular (STD\_LOGIC\_1164), en el encabezado del archivo vhd.*

```
library IEEE;  
use IEEE.STD_LOGIC_1164 . all;
```

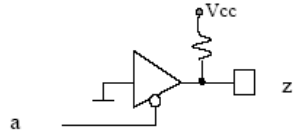
### Tipo de datos escalares (cont.)

**•EJEMPLO de declaración de la biblioteca IEEE y del package STD\_LOGIC\_1164:**

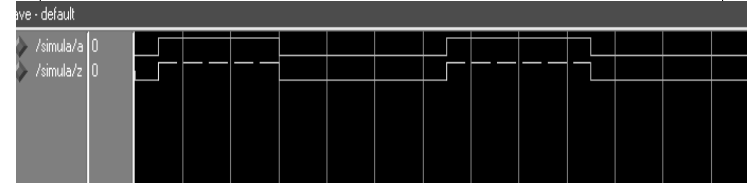
```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
Entity EJEMPLO is  
  port    ( A,B : in STD_LOGIC;  
            SEL : in STD_LOGIC;  
            MUX : out STD_LOGIC);  
  
end EJEMPLO;  
architecture ARCHI of EJEMPLO is      -- sigue la definición
```



```
13 -- Dependencies:  
14 --  
15 -- Revision:  
16 -- Revision 0.01 - File Created  
17 -- Additional Comments:  
18 --  
19 -----  
20 library ieee ;  
21 use      ieee.std_logic_1164.all;  
22  
23  
24 ENTITY bufer IS  
25   PORT(  
26     a  : IN  std_logic;  
27     z  : OUT std_logic  
28   );  
29 END ENTITY bufer;  
30  
31 ARCHITECTURE comportamiento OF bufer IS  
32 BEGIN  
33  
34  
35 z<= '0' when (a= '0') else 'Z';  
36  
37 z<= 'H';  
38  
39 END ARCHITECTURE comportamiento;  
40  
41
```



`z<='H'; --simula la R pull-up`  
`z <= '0' when (a = '0') else 'Z'; --simula la salida OC`



### Tipo de datos escalares (cont.)

- ◆ Para la síntesis con XST los valores '0' y el 'L' son tratados en forma idéntica, del mismo modo para '1' y 'H'.
- ◆ Los valores 'U' y 'W' no son aceptados.
- ◆ El valor 'Z' es tratado como alta impedancia.

### Tipo de datos compuestos

- Vectores : conjunto de objetos del mismo tipo ordenados mediante uno o más índices que indican la posición de cada objeto dentro del vector.
- Vectores de una dimensión (un índice) → vectores
- Vectores de más de una dimensión (varios índices) → matrices
- Sintaxis:

**type** identificador **is array** (rango {, ...}) **of** tipo\_objetos;

- El rango especifica el margen de valores del vector.
- El rango será un tipo discreto (entero o enumerado).

Ejemplos:

**type** Byte **is array** ( 7 downto 0) **of** bit;

### Tipo de datos compuestos (cont.)

**bit\_vector**: grupo de BITS (bus) “0001110” Está pre-definido, no hace falta que yo lo defina

-----  
Ejemplo:

signal A : bit\_vector (7 downto 0);

-- El bit de mayor peso a la izquierda

**begin**

A<= “01001011” -- equivalente a A<= X “4B”

-- la X indica un valor en hexa

### Tipo de datos compuestos (cont.)

**type** Memoria **is array** (0 to 7, 0 to 63) **of** bit;

-- Una vez declarado el tipo *Memoria* se pueden

-- declarar objetos de este tipo

**variable** Ram\_A, Ram\_B : Memoria;

**begin**

Ram\_A (4,7) := ‘1’

### Tipo de datos compuestos (cont.)

También acá, para dar una mayor potencia operacional, se puede usar el paquete de *IEEE* para datos compuestos:

- **STD\_LOGIC\_VECTOR**

Grupo de objetos similar a BIT\_VECTOR, pero con con los 9 estados posibles del STD\_LOGIC para cada uno de los bits.

*Para utilizar estos tipos de datos, debemos declara la utilización de la biblioteca IEEE que contiene el package particular (STD\_LOGIC\_1164), en el encabezado del archivo vhd.*

**library** IEEE;

**use** IEEE.STD\_LOGIC\_1164.all;

### Tipo de datos compuestos (cont.)

- **EJEMPLO de declaración de la biblioteca IEEE y del package STD\_LOGIC\_1164:**

-----  
library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

Entity EJEMPLO is

port ( A,B : in STD\_LOGIC\_VECTOR (7 DOWNT0 0);

SEL : in STD\_LOGIC;

MUX : out STD\_LOGIC\_VECTOR (7 DOWNT0 0);

end EJEMPLO;

architecture ARCHI of EJEMPLO is

-- sigue la definición



## Asignación de valores a objetos

Asignación simple de un valor a un objeto según su clase:

- **Para una señal:** `<=`

Ejemplos:

```
signal A, B, C : std_logic ;  
signal CS : boolean ;  
signal DATO : std_logic_vector (7 downto 0);  
signal ADR : std_logic_vector (7 downto 4);  
signal ENT_VAL : integer range 0 to 15;
```

```
-----  
A <= '1';  
B <= C xor D;  
DATO <= "10100101";  
ENT_VAL <= 13;
```

Asignación simple de un valor a un objeto según su clase:

- **Para una constante o variable:** `:=`

Ejemplos:

```
constant A : std_logic := '0' ;
```

```
constant VECTOR : std_logic_vector (3 downto 0) := "0101";
```

```
constant NUMERO : integer := 32;
```

Asignación simple de un valor a un objeto según su clase:

- **Bit ó std\_logic:**

DATO (0) <= '1'; -- comillas simples

- **Bit\_vector ó std\_logic\_vector :**

-- Base por defecto : **binaria**

DATO (7 downto 0) <= "11001010"; -- comillas dobles  
-- equivalente a DATO ( 7 downto 0) <= B"11001010";  
-- equivalente a DATO ( 7 downto 0) <= B"1100\_1010";

-- Asignación en base Hexadecimal:

DATO (3 downto 0) <= X"5";

Asignación simple de un valor a un objeto según su clase:

- **Enteros:**  
signal A, S, T, U, V : integer range 0 to 255;  
-- Base por defecto: **decimal**  
s <= A + 83 ; -- **sin comillas**  
-- Especificación de base:  
T <= **16#B3#** ; -- Hexadecimal  
U <= **10#45#** ; -- Decimal  
V <= **2#10110011#** ; -- Binaria  
-- equivalente a V <= 2#1011\_0011#; con separadores

Reglas de asignación de los vectores de datos:

- **El orden en el cual se utiliza el vector (bus) debe ser el mismo que en la declaración del vector -- (7 downto 0)**  
(valores creciente o decreciente de los índices)
- **No es necesario utilizar el vector entero**  
(utilización de una parte de las señales de un bus)
- **El ancho del bus (tamaño del vector) debe corresponder para la mayoría de las operaciones.**  
(excepto para la comparación)

architecture ARQUI of VECTOR is

-- parte declaratoria

**signal D\_IN, MASCARA, D\_OUT : std\_logic\_vector (7 downto 0);**

**signal Q\_OUT : std\_logic\_vector (7 downto 0);**

**constant FIJA : std\_logic\_vector (2 downto 0) := "101";**

-- la asignación de un valor a una constante o variable se realiza por el

-- símbolo ":", diferente a asignación de señal

begin

**D\_OUT <= D\_IN and not (MASCARA)** -- todas las oper. son s/ 8 bits

**Q\_OUT <= (D\_IN(6 downto 2) and not (MASCARA(7 downto 3)))**

**& FIJA;**

-- El signo "&" es un operador denominado de concatenación

end (ARQUI)

Ejemplo de código equivalente

**D\_OUT(7) <= D\_IN (7) and not (MASCARA(7));**

**D\_OUT(6) <= D\_IN(6) and not (MASCARA(6));**

..... ;

**D\_OUT(0) <= D\_IN(0) and not (MASCARA(0));**

**Q\_OUT(7) <= D\_IN(6) and not (MASCARA(7));**

**Q\_OUT(6) <= D\_IN(5) and not (MASCARA(6));**

..... ;

**Q\_OUT(3) <= D\_IN(2) and not (MASCARA(3));**

**Q\_OUT(2) <= FIJA(2);** -- ó Q\_OUT(2) <= '0';

**Q\_OUT(1) <= FIJA(1);** -- ó Q\_OUT(1) <= '1';

**Q\_OUT(0) <= FIJA(0);** -- ó Q\_OUT(0) <= '0';

# Operadores

## Operadores frecuentemente utilizados en síntesis:

- **Operadores lógicos pre definidos:**

- **and, or, nor, xor y not**

Operan sobre todos los objetos (señales, constantes y variables) y de tipo:

- bit
- bit\_vector
- std\_logic
- std\_logic\_vector
- boolean

Los operandos deben ser del mismo tipo y contener el mismo número de bits.

## Operadores frecuentemente utilizados en síntesis:

- **Operadores lógicos pre definidos:**

- *Ejemplo de utilización:*

```
library IEEE; -- declaración de la biblioteca IEEE, seguida de la
-- palabra use para indicar que se quiere utilizar de ésta el
-- package std_logic_1164 sin limitación de las funciones o ele-
-- mentos de la biblioteca (.all)
use IEEE.STD_LOGIC_1164.ALL;
entity OPERA is
  port (a, b, c : in std_logic_vector (6 downto 0);
        s :      out std_logic_vector (6 downto 0);
  end OPERA;
architecture ARQUI of OPERA is
begin
  s <= (a and b) and not (c);
end ARQUI;
```

## Operadores frecuentemente utilizados en síntesis:

- **Operadores relacionales:**

- = (igual a)                      /= (diferente de)
- < (menor a)                    <= (menor o igual a)
- > (mayor a)                    >= (mayor o igual a)

Operan sobre todos los objetos (señales, constantes y variables) y de tipo:

- bit
- bit\_vector
- std\_logic
- std\_logic\_vector
- integer
- boolean

Los operandos deben ser del mismo tipo, pero el número de bits comparados puede ser diferente. (cuidado !!!)

## Operadores frecuentemente utilizados en síntesis:

### Operadores relacionales: Comparación de vectores.

La comparación se hace bit a bit entre los dos vectores comenzando por el MSB.

Los resultados pueden traer sorpresa, cuando no son del mismo tamaño. Ejemplo:

```
signal REGIS : std_logic_vector (4 downto 0);
signal CONT  : std_logic_vector (3 downto 0);
begin
```

```
REGIS <= "01111" ;      CONT <= "1000";
```

- Si realizáramos una comparación entre vectores, el sistema encontrará
- que **CONT ES MAYOR QUE REGIS** pues compara 0111 con 1000 y
- encuentra que el MSB de CONT es mayor que el MSB de REGIS.

## Operadores frecuentemente utilizados en síntesis:

### Operadores relacionales: información complementaria.

El resultado de una comparación de tipo "boolean" puede tomar sólo los valores TRUE o FALSE.

-----  
Ejemplo:

```
signal IGUAL, VENTANA : boolean;
signal CONT  : integer range 0 to 31;
begin
CONT := 22 ;
IGUAL <= (CONT = 37);
VENTANA <= (CONT >= 13) and (cont <= 25);
```

## Operadores frecuentemente utilizados en síntesis:

### Operadores aritméticos:

+ (suma)                      - (resta)  
\* (multiplicación)            / (división)  
\*\* (potencia)                abs () (valor absoluto)

MOD --  $a = b \cdot N + (a \text{ MOD } b)$ ,  $N \rightarrow$  entero

REM (resto) --  $a = (a/b) * b + (a \text{ REM } b)$

- Operan sobre objetos tipo INTEGER
- Pueden también operar sobre STD\_LOGIC\_VECTOR utilizando el package STD\_LOGIC\_ARITH.
- **Restricción:** La mayoría de las herramientas de síntesis sólo autorizan las operaciones de multiplicación y división entre CONSTANTES, o una constante potencia de 2 y una SEÑAL

## Operadores frecuentemente utilizados en síntesis:

### Implementación física de funciones descritas por medio de operadores aritméticos.

- **Algunas arquitecturas** de componentes programables tienen recursos de lógica específicos para implementación eficaz de funciones aritméticas y similares (en términos de velocidad y área de silicio usado).
- Las herramientas de síntesis pueden hacer uso de estos recursos y opciones para elegir.

Ejemplo, XILINX dispone de los recursos LOGI BLOX y CORE.

## Operadores frecuentemente utilizados en síntesis:

### Operadores de desplazamiento y rotación.

- **rol** (rota a izquierda)
- **ror** (rota a derecha)
- **sll** (desplazamiento lógico izquierda)
- **srl** (desplazamiento lógico derecha)

Hacer visibles **IEEE.numeric\_std.ALL**  
**IEEE.std\_logic\_unsigned.ALL**

Operan sobre objetos de tipo:

- unsigned
- bit\_vector
- std\_logic\_vector (!)

```
signal a,b,c,d,e,f,g:unsigned(7 downto 0);
```

```
a<="11000101";
```

```
b<= a sll 2;
```

```
c<= a srl 2;
```

```
f<=a rol 2;
```

```
g<=a ror 2;
```

a	11000101
b	00010100
c	00110001
f	00010111
g	01110001

## Operadores frecuentemente utilizados en síntesis:

### Operador de concatenación (&).

Permite la creación de vectores a partir de bit o de vectores

Ejemplo:

```
library IEEE;
use IEEE.STD_LOGIC_VECTOR_1164.all;
entity MI_BUS is
    port ( A, B, C, D in std_logic;
           SALIDA out: std_logic_vector (7 downto 0));
architecture ARQUI of MI_BUS is
begin
    SALIDA <= "0000" & A & B & C & D;
    -- equivalente a : SALIDA (7 DOWNT0 4) <= "0000"; SALIDA (3) <= A;
    -- SALIDA (2) <= B ; SALIDA (1) <= C ; SALIDA (0) <= D ;
end ARQUI;
```