# INTRODUCCIÓN A VHDL

CUDAR

UTN FRC

Ing. Paredes, Federico
Ing. Cayuela, Pablo

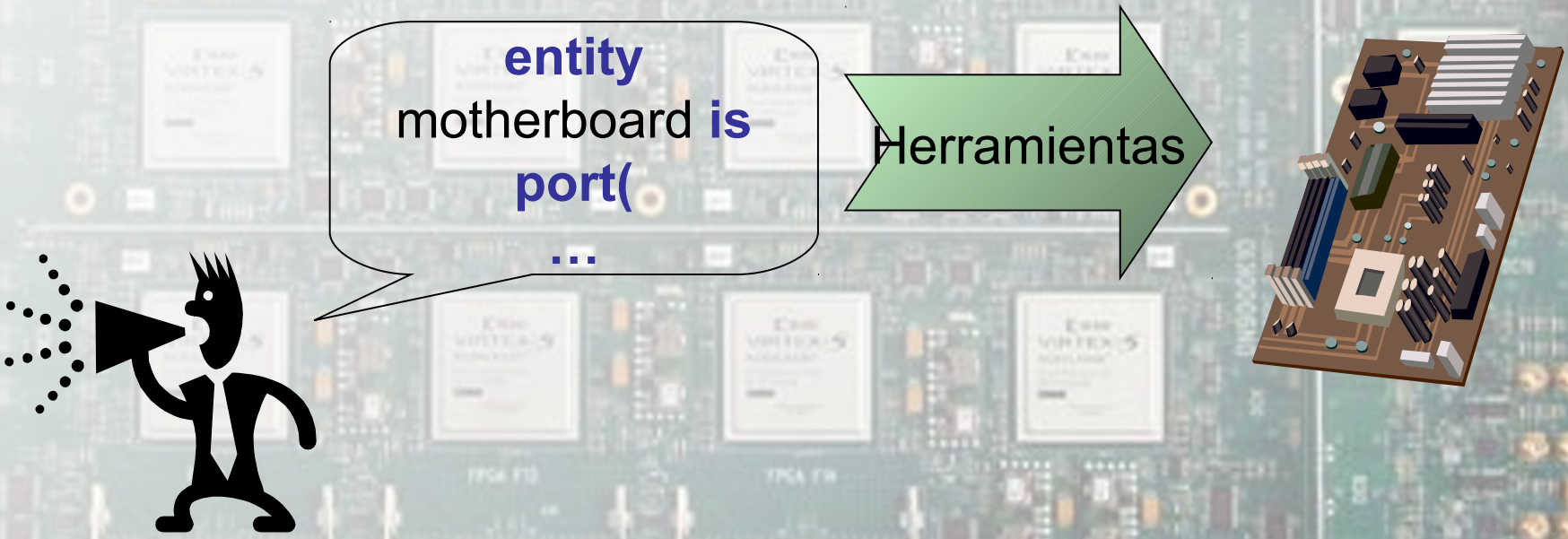
# INTRODUCCIÓN A VHDL

UTN FRC

¿Que es VHDL?

• VHDL es el acrónimo simplificado de “**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguaje”

• Permite modelar, simular y documentar sistemas digitales, a través de distintas herramientas que lo interpretan.



# INTRODUCCIÓN A VHDL

UTN FRC

Algunas características importantes de VHDL

- VHDL es un lenguaje concurrente, sus sentencias se “ejecutan” en paralelo.
- Es un lenguaje extenso con más de 100 palabras reservadas.
- Posibilita la integración de diferentes herramientas de CAD.
- Posee una sintaxis amplia y flexible, permitiendo distintos tipos de descripción.
- Permite dividir un diseño en unidades más pequeñas y la creación de librerías, facilitando la reutilización de código.
- Se encuentra estandarizado por el IEEE.
- Permite diseñar, modelar y simular un sistema desde altos niveles de abstracción hasta niveles muy cercanos al hardware final.

# INTRODUCCIÓN A VHDL

UTN FRC

Adelantos para ir perdiendo el miedo a programar

- Todas las sentencias terminan en punto y coma ;
- Los comentarios comienzan con –
- Todo identificador debe comenzar con una letra y contener solo letras, números y guiones bajos _
- Los números están por defecto en base 10.

# INTRODUCCIÓN A VHDL

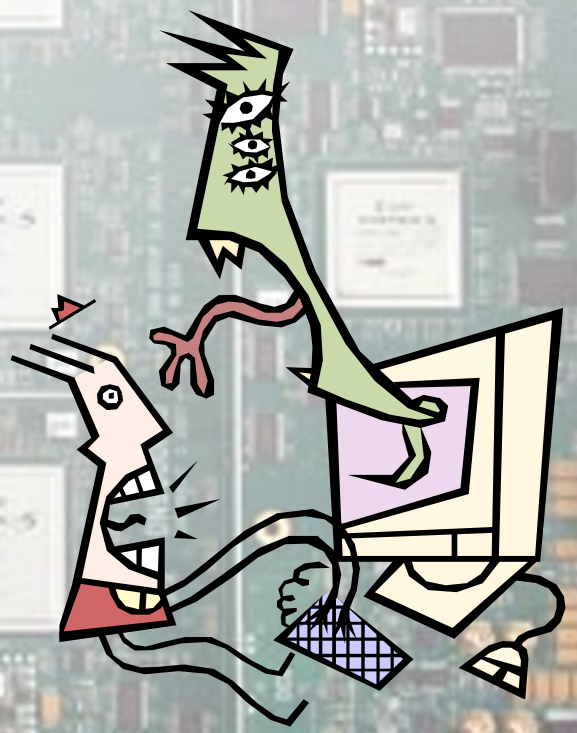
UTN FRC

Adelantos para ir perdiendo el miedo a programar

• Pueden especificarse otras bases con el caracter numeral #. Ej: 2#10110#  (binario).

• Mayúsculas y minúsculas se consideran equivalentes.

• Las cadenas se definen mediante comillas dobles, los caracteres con comillas simples. Ej “cadena”, ‘a’.



UTN FRC

Diseño de hardware: El concepto de entidad


Un diseño de hardware comienza como un bloque con entradas y salidas definidas. Este bloque se denomina **ENTITY** (Entidad).

# INTRODUCCIÓN A VHDL

UTN FRC

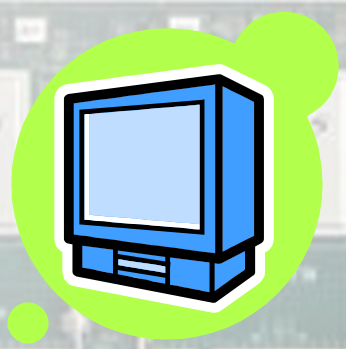
## La Entidad

Todo diseño en VHDL comienza con una entidad

```
entity celular is
...
end entity celular;
```


```
entity procesador is
...
end entity procesador;
```

```
entity televisor is
...
end entity televisor;
```

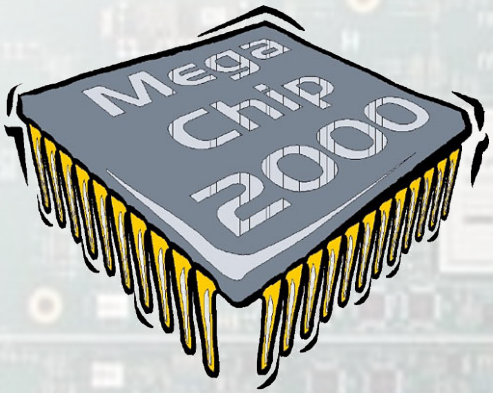


# INTRODUCCIÓN A VHDL

UTN FRC

## La Entidad

La entidad especifica las las entradas y salidas (puertos) y parámetros de un bloque.




```
entity megachip  is
  generic(
        N : integer := 8);
  port(
        entrada : in bit;
        salida  : in bit);
end entity megachip;
```

Con la palabra "**generic**" definimos los parámetros

Con la palabra "**port**" declaramos los puertos de nuestra entidad

UTN FRC

## La Entidad

La entidad especifica como se conecta el bloque al mundo exterior y permite establecer los parámetros del mismo.

Resumiendo

# ENTIDAD = INTERFASE

**Sin entidad, no existe hardware!**

# INTRODUCCIÓN A VHDL

UTN FRC

## La Arquitectura

El paso siguiente a la entidad es definir su contenido.
Esto se realiza en un bloque llamado **ARCHITECTURE** (Arquitectura).

Descripción de arquitectura

```
architecture contenido of mi_entidad is

[ Parte declarativa ]

begin

[ Sentencias ]

end architecture contenido;
```

Una arquitectura siempre se asocia con una entidad determinada.

# INTRODUCCIÓN A VHDL

UTN FRC

## La Arquitectura

Una entidad puede tener muchas arquitecturas.

```
entity teléfono is
...
end entity teléfono;
```

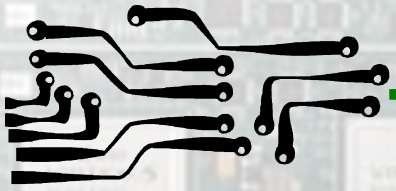
```
architecture móvil of teléfono is
...
end architecture móvil;
```

```
architecture fijo of teléfono is
...
end architecture fijo;
```

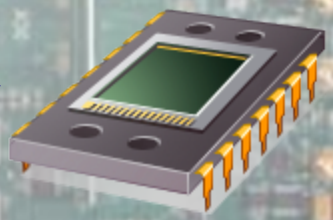
UTN FRC

## La Arquitectura

La porción declarativa de la arquitectura permite declarar señales, constantes y componentes entre otros elementos.



```
architecture contenido of mi_entidad is

constant retardo  : time := 1 ms;

signal conexión1 : integer;

component chip
…

begin
…
```

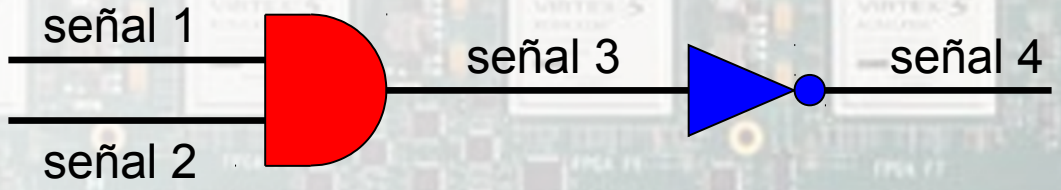


# INTRODUCCIÓN A VHDL

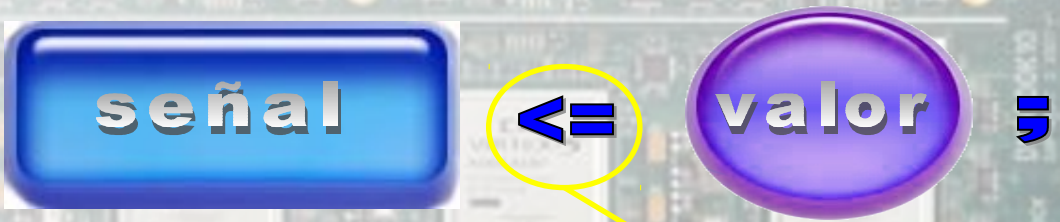
UTN FRC

## Señales

Una señal cumple en VHDL la misma función que una conexión en un circuito




Una señal puede sujeto de asignaciones

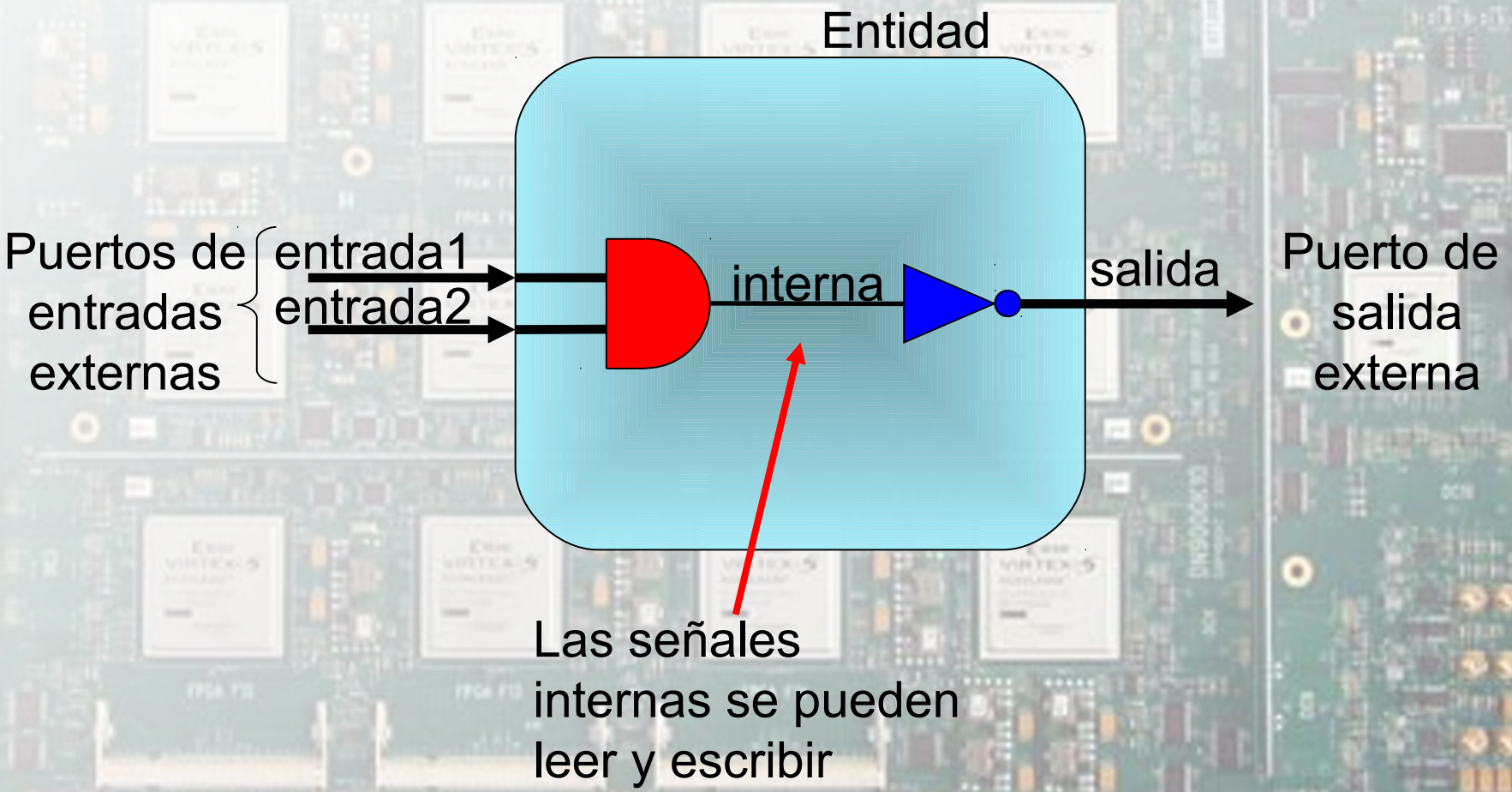



Operador de asignación para señales

UTN FRC

## Señales

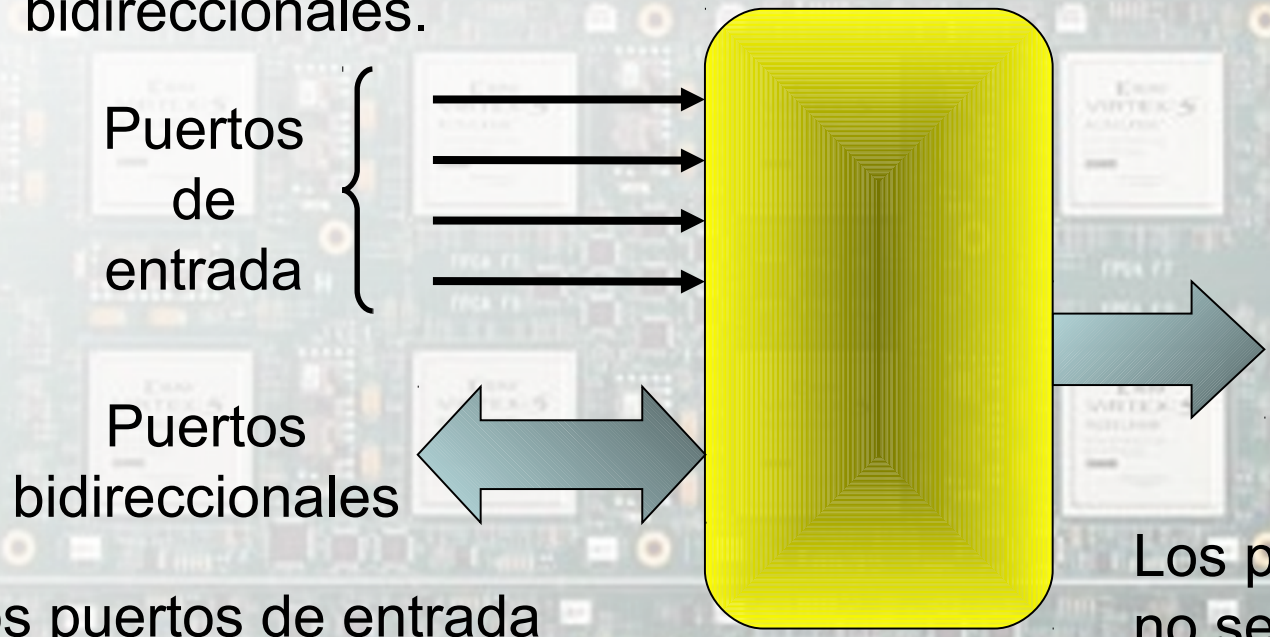
Las señales se pueden clasificar en señales internas y señales externas (puertos).



UTN FRC

## Señales

Las señales externas pueden ser de entrada, de salida o bidireccionales.




Los puertos de entrada no se pueden escribir (solo lectura).

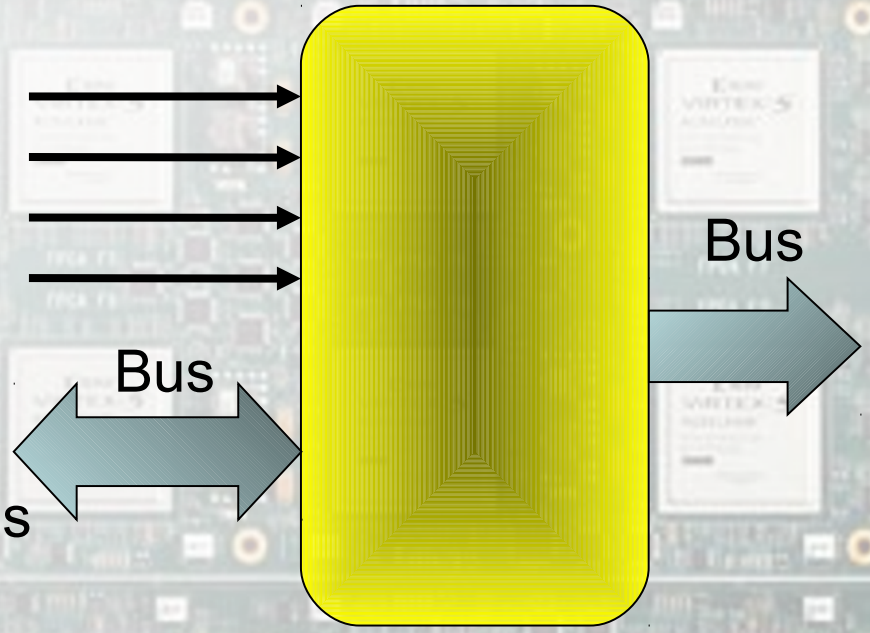
Los puertos de salida no se pueden leer (solo escritura).

- Las señales de entrada se especifican como **in**
- Las señales de salida se especifican como **out**
- Las bidireccionales se especifican como **inout**

UTN FRC

## Señales

También se pueden clasificar en señales individuales y buses.



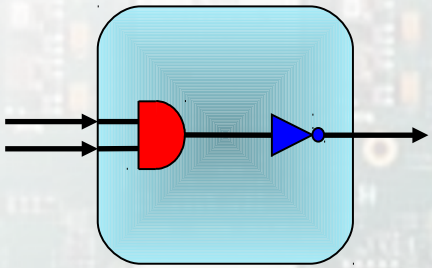

Los buses se declaran como arreglos de señales individuales

Varias señales individuales se pueden unir (concatenar) para formar un bus y éste también se puede descomponer en señales únicas.

UTN FRC

## Señales

Las señales externas se declaran en la entidad, entre los paréntesis de la sentencia **port**

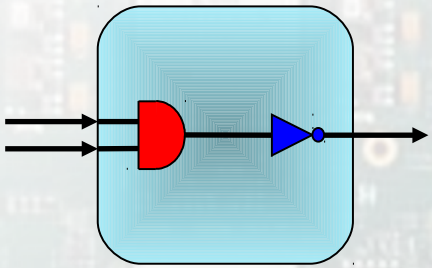


```
entity compuerta  is
  port(
        entrada1    : in     bit;
        entrada2    : in     bit;
        salida      : out    bit
      );
end entity compuerta;
```

UTN FRC

## Señales

Las señales internas se definen en la sección declarativa de la arquitectura



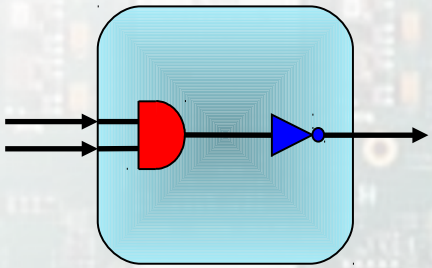
```
architecture estructura of compuerta is

   signal interna : bit;

begin

interna <= entrada1 and entrada2;
salida  <= not interna;

end architecture compuerta;
```

# INTRODUCCIÓN A VHDL

UTN FRC

## Señales

Podemos ver el diseño completo



```
entity compuerta  is
  port(
        entrada1   : in    bit;
        entrada2   : in    bit;
        salida     : out   bit);
end entity compuerta;
architecture estructura of compuerta is
  signal interna : bit;
begin
interna <= entrada1 and entrada2;
salida  <= not interna;
end architecture compuerta;
```

# INTRODUCCIÓN A VHDL

UTN FRC

## Tipos de datos

Todas las señales deben poseer un tipo de datos definido.

Ejemplos de datos predefinidos en VHDL

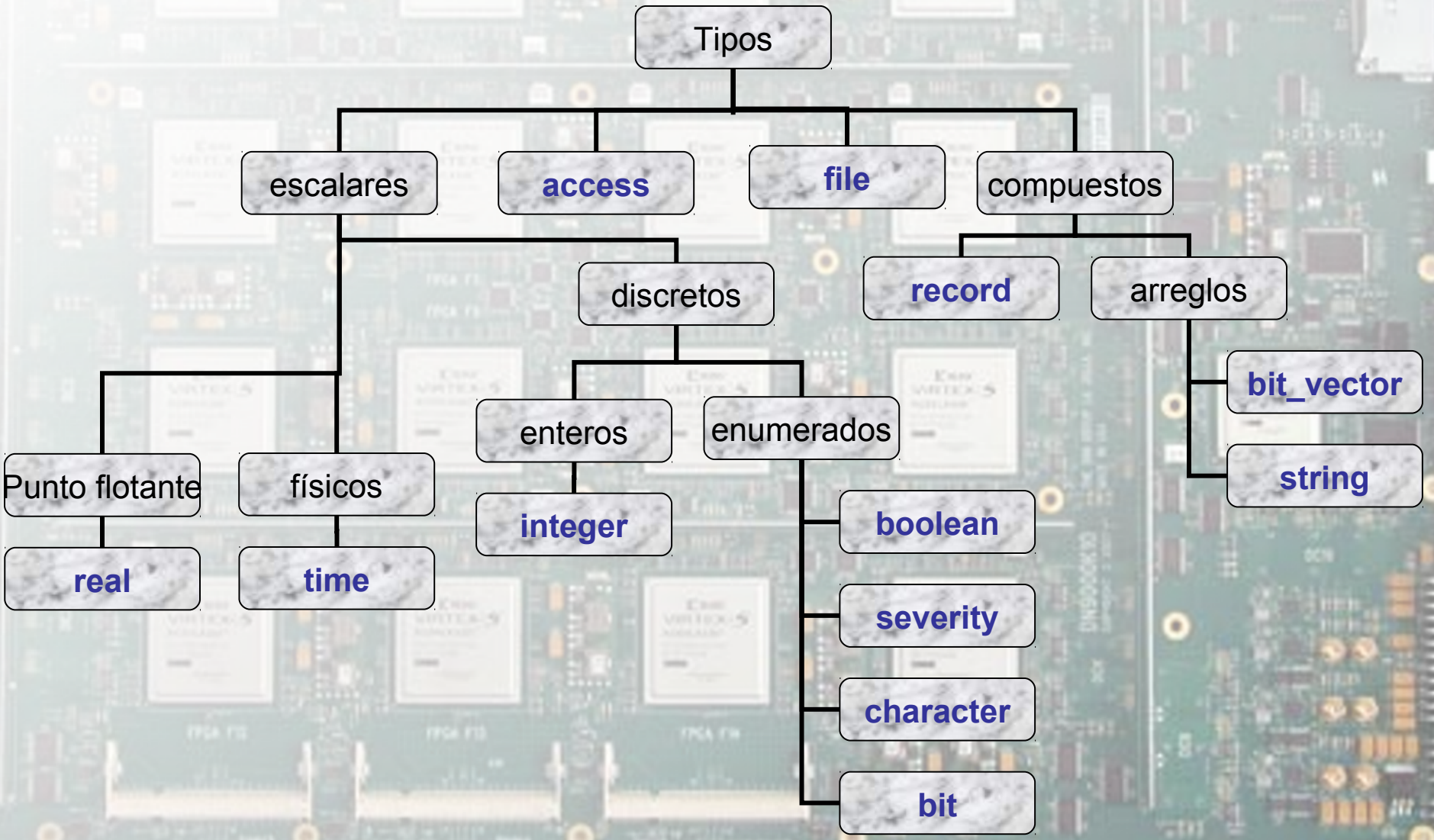
- bit 100110101
- integer -1 0 1 2 3 ...
- time 10 ns, 5 ms
- real $\pi = 3{,}14159265358979\ldots$
- file

*Archivo.vhd*

# INTRODUCCIÓN A VHDL

UTN FRC

## Clasificación de los tipos de datos predefinidos

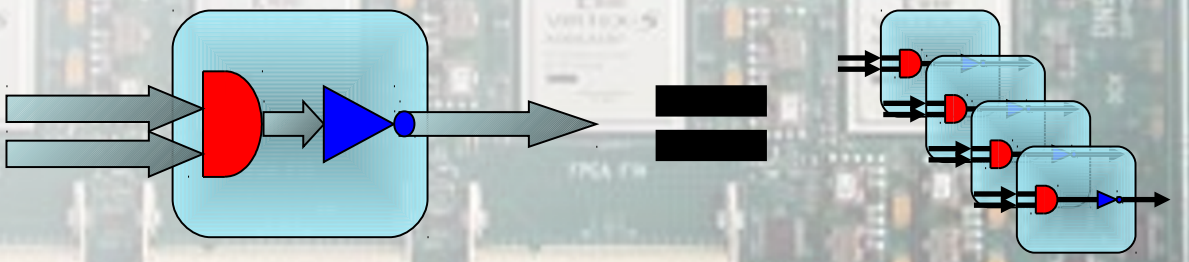


UTN FRC

## Tipos de datos

El tipo **bit** solo tiene dos valores posibles: ‘0’ y ‘1’, y se utiliza para señales individuales. Para ampliar la cantidad de bits se puede utilizar el tipo **bit_vector**.

```
entity compuerta  is
   port(
         entrada1   : in    bit_vector(3 downto 0);
         entrada2   : in    bit_vector(3 downto 0);
         salida     : out   bit_vector(3 downto 0));
end entity compuerta;
```



UTN FRC

## Tipos de datos

En toda declaración de vectores se utilizan las palabras reservadas **downto** y **to** para determinar el tamaño del vector y la ubicación del bit más significativo.

```
signal mi_vector : bit_vector(0 to 7)
```

MSB LSB

| bit 0 | bit 1 | bit 2 | bit 3 | bit 4 | bit 5 | bit 6 | bit 7 |
|---|---|---|---|---|---|---|---|

Para evitar confusiones utilice sólo un tipo de declaración

```
signal mi_vector : bit_vector(7 downto 0)
```

MSB LSB

| bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---|---|---|---|---|---|---|---|

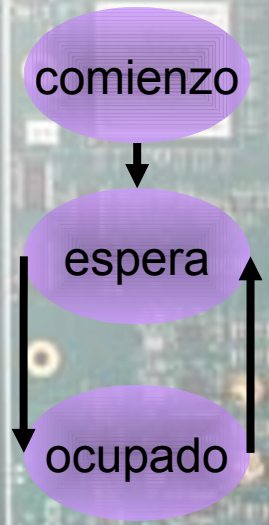
UTN FRC

## Tipos de datos

También es posible definir nuevos tipos. Para esto se utiliza la sentencia **type**. Se pueden definir subconjuntos de tipos existentes, tipos enumerados o arreglos de varias dimensiones. Las definiciones se realizan en la parte declarativa de la arquitectura.

```
type estados is (comienzo, espera, ocupado);
signal estado_actual : estados;
```

Los tipos enumerados suelen utilizarse para crear máquinas de estado

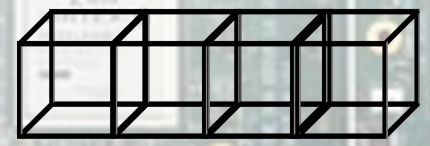


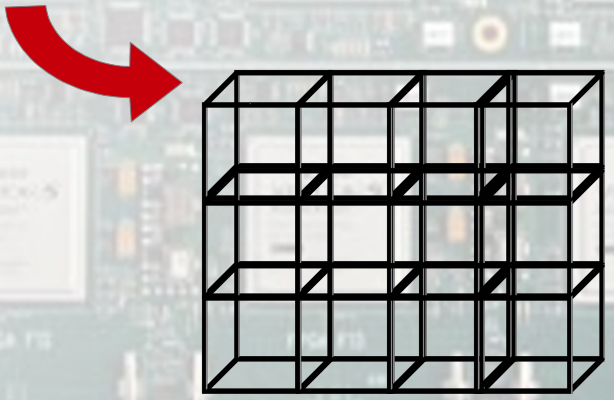
UTN FRC

## Tipos de datos

También se pueden definir arreglos de una o más dimensiones.

```
type vector is array (4 downto 0)  of integer;
type matriz is array (2 downto 0) of vector;

signal mi_vector : vector;

signal mi_matriz : matriz;
```



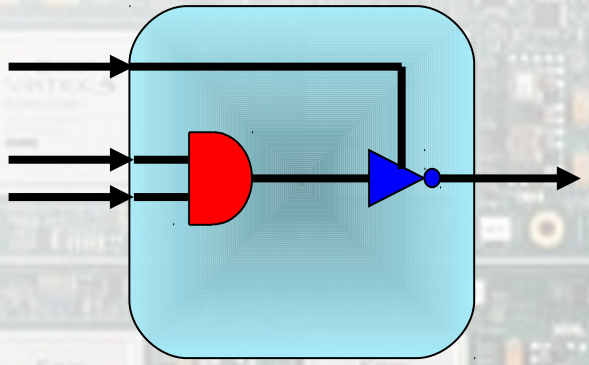


UTN FRC

## Tipos de datos

¿Son suficientes los tipos **bit** y **bit_vector** para los sistemas digitales?

Supongamos que deseamos agregar a nuestro circuito la capacidad de poner su salida en alta impedancia.



Pero los tipos de datos usados solo tiene los valores 0 y 1!


y Z?

# INTRODUCCIÓN A VHDL

UTN FRC

## Tipos de datos

Los tipos de datos predefinidos no alcanzan para describir un sistema digital en su totalidad. Por eso se agrega un tipo de datos llamado **STD_LOGIC,** agregado desde una librería.

Valores que pueden tomar las señales std_logic

std_logic

- U Sin inicializar
- X Desconocido fuerte
- 0 Nivel lógico bajo
- 1 Nivel lógico alto
- Z Alta impedancia
- W Desconocido débil
- L Cero débil
- H Uno débil
- — No importa

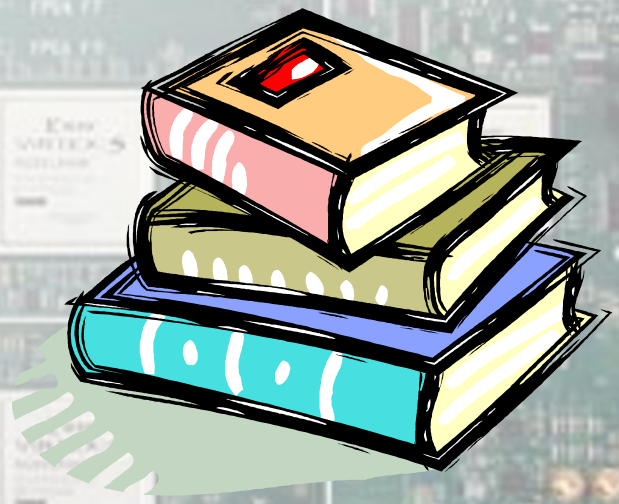
**std_logic** está definido en la librería **ieee.std_logic_1164**

# INTRODUCCIÓN A VHDL

UTN FRC

## Librerías y paquetes

Las librerías contienen declaraciones de tipos, definición de operadores, componentes, entidades, funciones, etc. Permiten incluir elementos predefinidos facilitando la programación y la reutilización de código.



Para utilizar una librería

**library** nombre_librería;
**use** nombre_librería.nombre_paquete.**all**

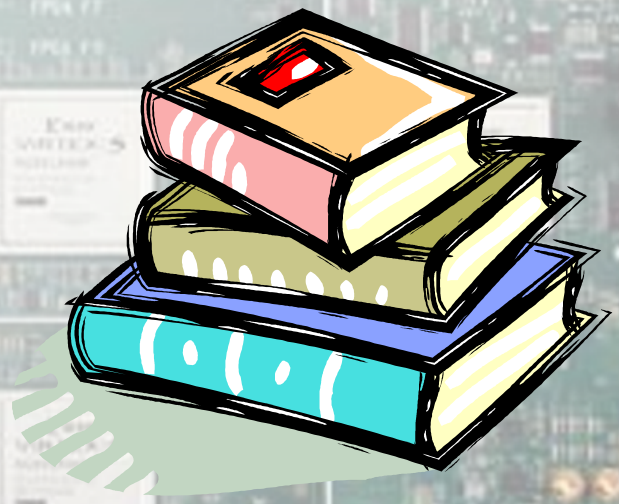
# INTRODUCCIÓN A VHDL

UTN FRC

## Librerías y paquetes

Una librería importante es la librería IEEE.
Ésta se encuentra subdivida en paquetes.
Algunos paquetes importantes:

- std_logic_1164
- std_logic_signed
- std_logic_arith
- numeric_std



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

# INTRODUCCIÓN A VHDL

UTN FRC

## Operadores

EnVHDL existe una amplia cantidad de operadores.
A continuación se detallan algunos.

### Operadores aritméticos:

** Exponencial

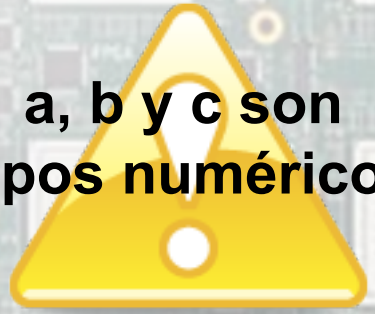
a <= b ** c;

abs() Valor absoluto

a <= abs(b);

* Multiplicación

a <= b * c;




/ División

a <= b / c;

mod Módulo

a <= b mod c;

\+ Suma

a <= b + c;

# INTRODUCCIÓN A VHDL

UTN FRC

## Operadores

### Operadores aritméticos:

**rem** Resto

a <= b **rem** c;

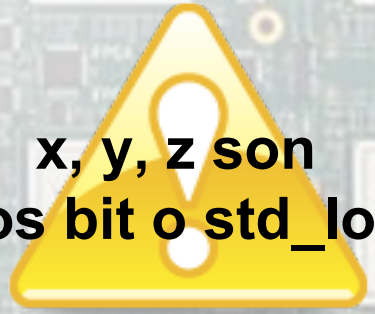
**-** Resta

a <= b **-** c;

### Operadores lógicos:

**not**

x <= **not** y;



**x, y, z son tipos bit o std_logic**

**and**

x <= y **and** z;

**nand**

x <= y **nand** z;

**or**

x <= y **or** z;

**nor**

x <= y **or** z;

**xor**

x <= y **xor** z;

# INTRODUCCIÓN A VHDL

UTN FRC

## Operadores

### Operador de concatenación

**&** Concatenación

```
x <= y & z;
```

### Operadores relacionales

**=** **=/** Igualdad / Desigualdad

**>** **<** Mayor / Menor

**>=** **<=** Mayor igual / Menor igual

Los operadores relacionales devuelven un valor de tipo booleano (true o false)

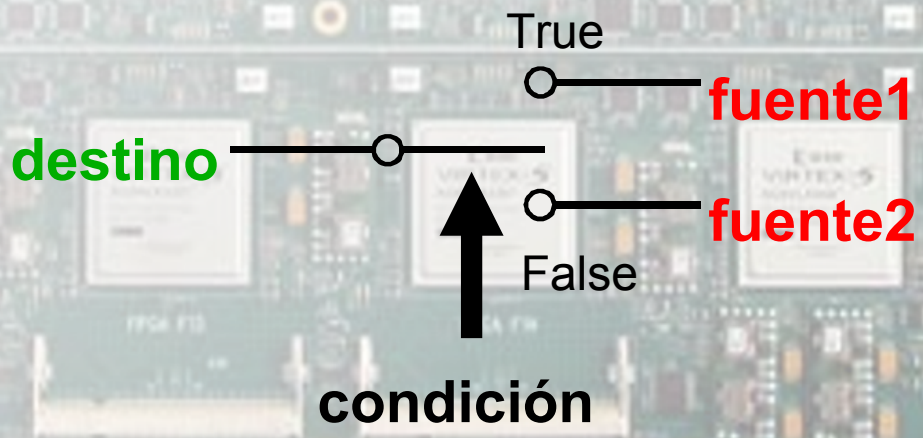
UTN FRC

## Asignaciones condicionales

**when … else**

destino <= *fuente1* **when** *condición* **else** *fuente2*

- *fuente1* y *fuente2* pueden ser expresiones, señales o valores ctes
- *condición* es una expresión con resultado booleano.

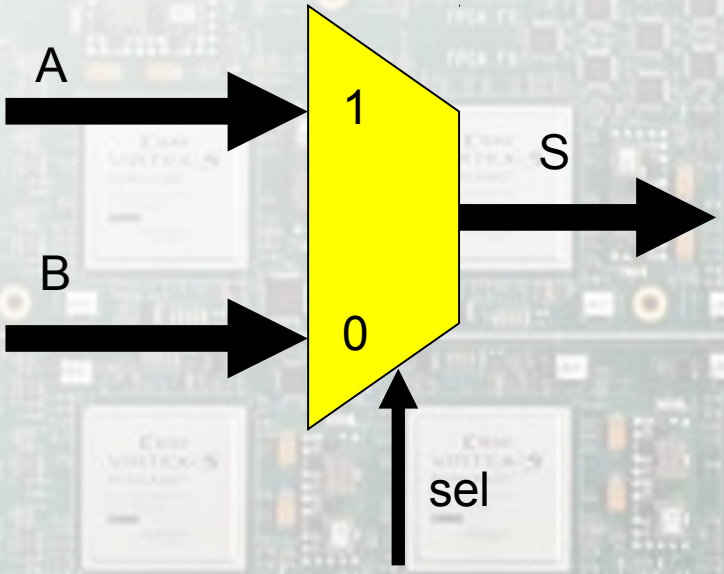


UTN FRC

## Asignaciones condicionales

### when … else

Es ideal para describir multiplexores




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

Se pueden anidar varias sentencias por ejemplo, para formas un mux más grande:

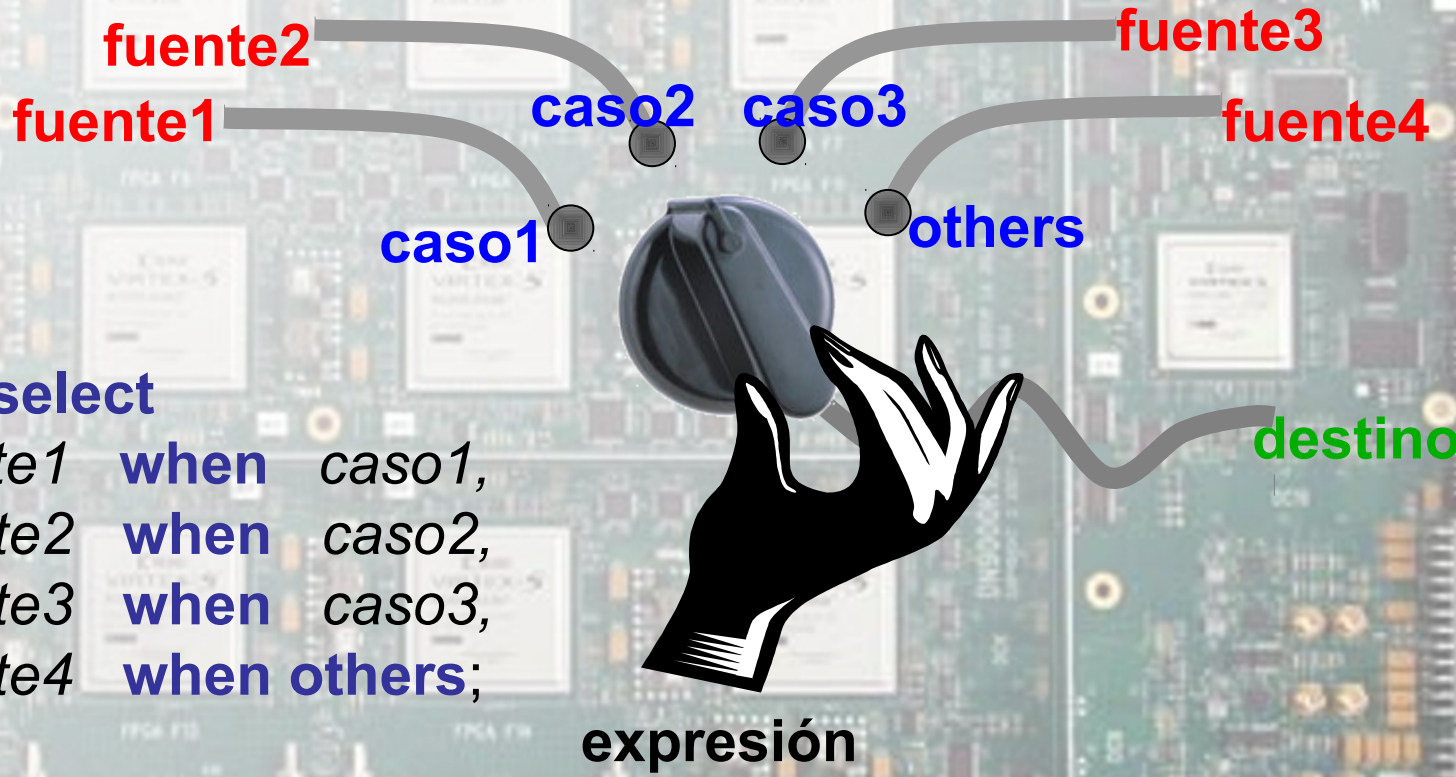
```
S <= A when sel="00" else
     B when sel="01" else
     C when sel="10" else
     D;
```

# INTRODUCCIÓN A VHDL

UTN FRC

## Asignaciones condicionales

**with … select …**




**with** *expresión* **select**
destino <= *fuente1* **when** *caso1,*
*fuente2* **when** *caso2,*
*fuente3* **when** *caso3,*
*fuente4* **when others**;

# INTRODUCCIÓN A VHDL

UTN FRC

Ejemplo de código completo

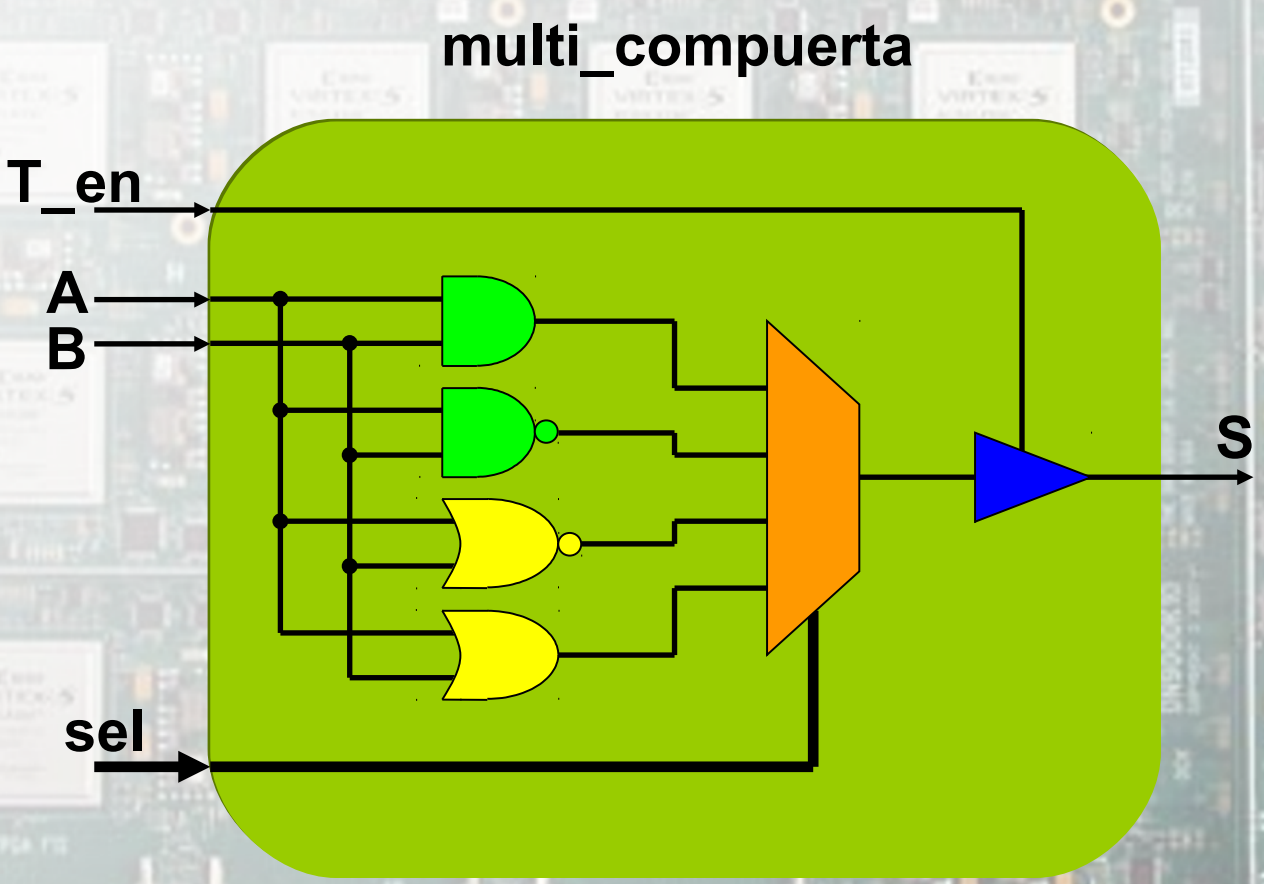
```
library IEEE;                                   -- Declaración de librerías
use IEEE.STD_LOGIC_1164.ALL;

entity compuerta is
    Port (
            entrada1   : in    STD_LOGIC_VECTOR (3 downto 0);
            entrada2   : in    STD_LOGIC_VECTOR (3 downto 0);
            triestado  : in    STD_LOGIC;
            salida     : out   STD_LOGIC_VECTOR (3 downto 0));
end compuerta;

architecture Behavioral of test is
  signal interna : STD_LOGIC_VECTOR (3 downto 0);
begin
  interna  <= entrada1 and entrada2;
  salida   <= not interna when triestado='0' else (others=>'Z');
end Behavioral;
```

# INTRODUCCIÓN A VHDL

UTN FRC

Ejercicio Práctico:

Describir el siguiente circuito



# INTRODUCCIÓN A VHDL

UTN FRC

```
-- Ejemplo de descripción flujo de datos
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity multi_compuerta is
   generic( N              : integer := 8);
   port ( sel              : in   std_logic_vector(1 downto 0);
          T_en             : in   std_logic;
          A                : in   std_logic_vector(N-1 downto 0);
          B                : in   std_logic_vector(N-1 downto 0);
          S                : out std_logic_vector(N-1 downto 0);
end entity multi_compuerta;
architecture flujo_de_datos of multi_compuerta is
   signal  salida_interna : std_logic_vector(N-1 downto 0);
begin
   with sel select
      salida_interna <= A and B   when   “00”,
                        A nand B when   “01”,
                        A nor B    when   “10”,
                        A or B      when others;
   S <= salida_interna when T_en=‘0’ else (others=>‘Z’);
end architecture flujo_de_datos;
```