

2.2. Diferentes aspectos de VHDL. Circuitos combinacionales

OBJETIVOS DEL TEMA:

- ☐ Concurrencia versus Secuencialidad
- ☐ Sentencias de asignación concurrentes:
 - a) Simple
 - b) Condicional
 - c) Selectiva
 - d) Procesos
- ☐ Tipo de datos en VHDL
- ☐ Objetos en VHDL: constantes
- ☐ Descripción de circuitos combinacionales básicos en VHDL
- ☐ Diseño jerárquico: Estilo de descripción estructural en VHDL

2.2.1. VHDL: Concurrencia vs Secuencialidad

VHDL: Concurrencia vs Secuencialidad

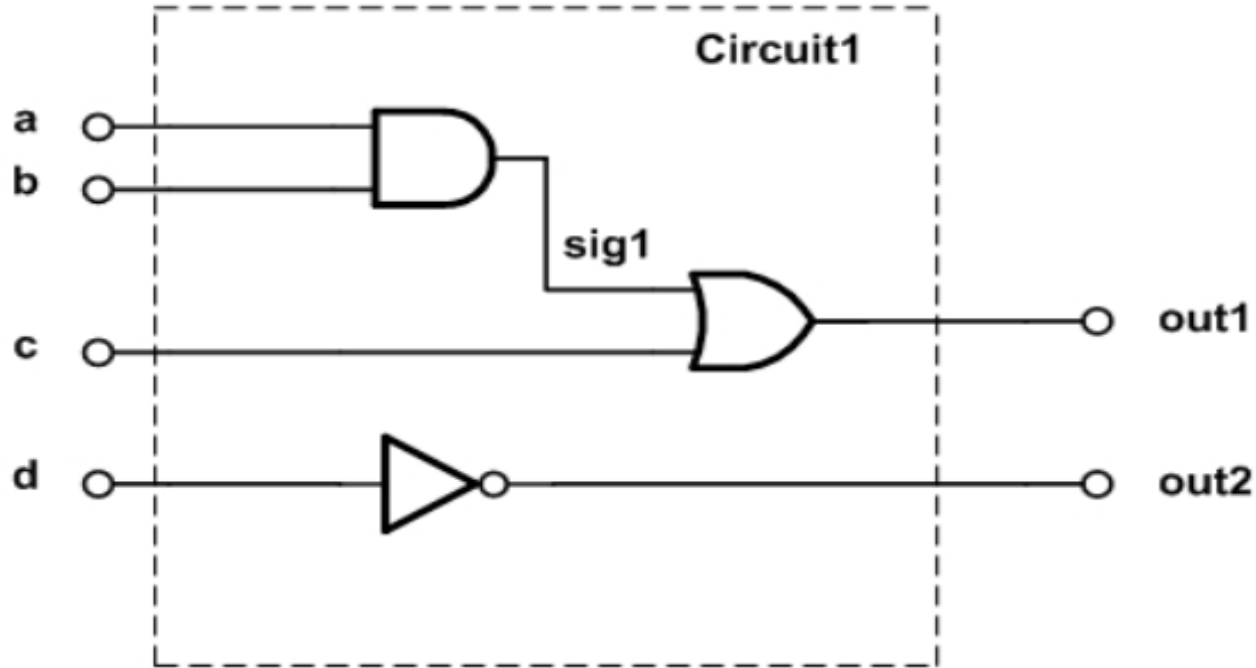
Secuencialidad

- Cada sentencia o instrucción en un lenguaje algorítmico representa una acción del procesador.
- Cuando concluye una acción avanza para leer y ejecutar la siguiente acción → Igual que los humanos

Concurrencia

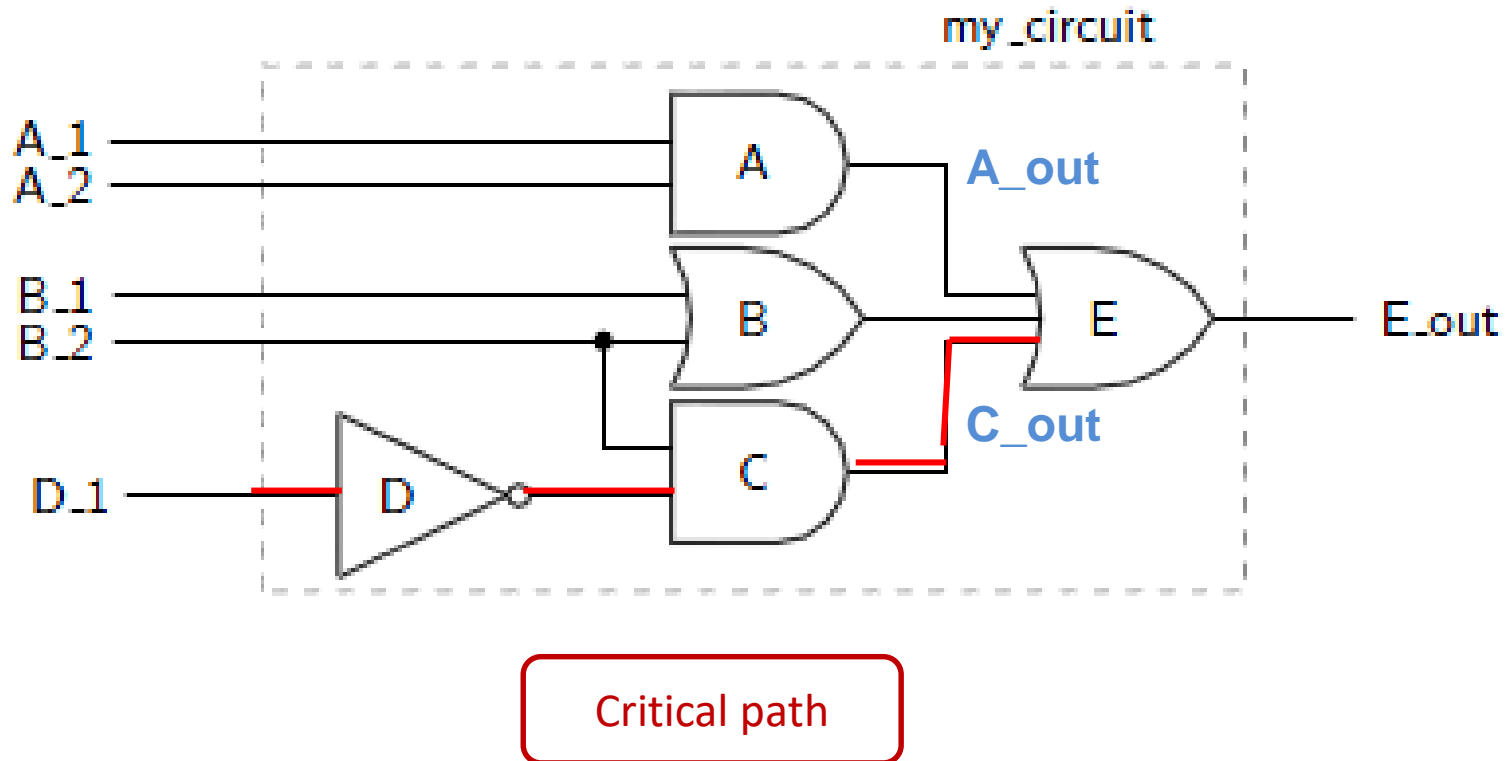
- Cada sentencia o instrucción en un lenguaje hardware representa una acción del circuito.
- Pero las ejecuta todas a la vez, es decir, en paralelo

VHDL: Concurrencia vs Secuencialidad



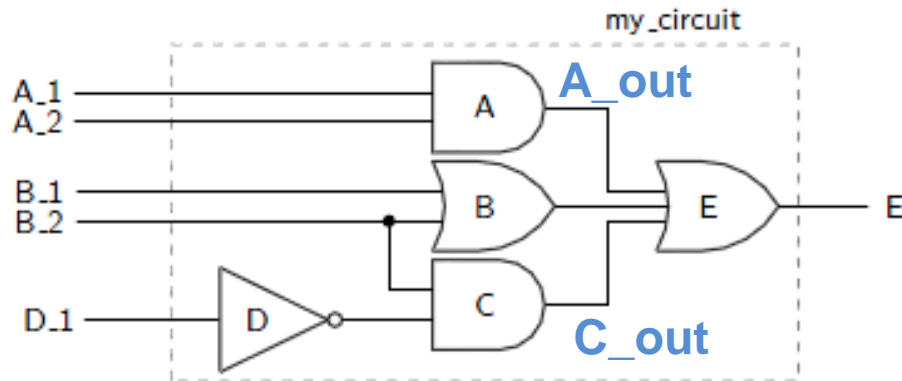
```
architecture Behavioral of comb1 is
    signal sig1: std_logic;
begin
    sig1 <= ( a and b );
    out1 <= ( sig1 or c );
    out2 <= (not d);
end Behavioral;
```

VHDL: Concurrencia vs Secuencialidad



VHDL: Concurrencia vs Secuencialidad

Concurrencia



```
-- architecture
architecture my_circuit_arc of my_circuit is
    signal A_out, B_out, C_out : std_logic;
begin
    A_out <- A_1 and A_2;
    B_out <- B_1 or B_2;
    C_out <- (not D_1) and B_2;
    E_out <- A_out or B_out or C_out;
end my_circuit_arc;
```

```
C_out <- (not D_1) and B_2;
A_out <- A_1 and A_2;
B_out <- B_1 or B_2;
E_out <- A_out or B_out or C_out;
```

```
A_out <- A_1 and A_2;
E_out <- A_out or B_out or C_out;
B_out <- B_1 or B_2;
C_out <- (not D_1) and B_2;
```

```
B_out <- B_1 or B_2;
A_out <- A_1 and A_2;
E_out <- A_out or B_out or C_out;
C_out <- (not D_1) and B_2;
```

Todos dan como
resultado el
mismo circuito

VHDL: Concurrencia vs Secuencialidad

CONCURRENCIA	SECUENCIALIDAD en VHDL
Las sentencias se ejecutan a la vez, en paralelo	Las sentencias se ejecutan una a continuación de otra (Programación Software)
Las sentencias se pueden escribir en cualquier orden	Se producen dentro de un proceso
Ejecutar más tareas a la vez incrementa el rendimiento	
Una instrucción se ejecuta solo cuando cambia alguna de sus entradas	El proceso se inicia cuando cambia alguno de los valores de la lista de sensibilidad
La asignación de los nuevos valores se realiza una vez ejecutadas todas las instrucciones cuyas entradas haya cambiado	La asignación de los nuevos valores solo se produce cuando se detiene el proceso
Si como consecuencia del nuevo valor cambia alguna de las entradas de otra instrucción, entonces se ejecuta dicha instrucción.	

2.2.2. VHDL: Sentencias en VHDL

VHDL: Sentencias en VHDL

Sentencias de descripción

a) Concurrentes:

➤ Asignación de señales

- ✓ Simple o incondicional (\leq)
- ✓ Condicional (when-else)
- ✓ Selectiva (with –select- when)

➤ Procesos

b) Secuenciales (dentro de un proceso)

➤ Asignación de señales

➤ Control del flujo de ejecución

- ✓ If-then-else
- ✓ case

2.2.2.1 VHDL: Sentencias Concurrentes

VHDL: Sentencias en VHDL

Sentencias concurrentes

➤ Asignación de señal:

1. Simple o incondicional

```
entity WAVEFORMS is
  port( Sal1_o : out std_logic_vector(1 downto 0);
        Sal2_o : out std_logic_vector(1 downto 0));
end WAVEFORMS;
```

```
architecture DATAFLOW of WAVEFORMS is
  Sal1_o <= "00"; -- Mismo tipo y tamaño
  Sal2_o <= "11";
end DATAFLOW ;
```

VHDL: Sentencias en VHDL

Sentencias concurrentes

➤ Asignación de señal:

2.Condicional (when – else). Tres opciones:

```
1: target <= thing1 when (condición1) ;  
  
2: target <= thing1 when (condición1) else thing2 ;  
  
3: target <= thing1 when (condición1) else  
    thing2 when (condición2) else  
    ...  
    thingn ;
```

Evitar LATCH
estableciendo todas las
opciones

VHDL: Sentencias en VHDL

Sentencias concurrentes

➤ Asignación de señal:

2. Condicional (when – else).

```
architecture BEHAVIORAL of MUX_8 is  
begin
```

```
    Y_o <=      E_i(0) when sel_i = "000" else  
                E_i(1) when sel_i = "001" else  
                E_i(2) when sel_i = "010" else  
                E_i(3) when sel_i = "011" else  
                E_i(4) when sel_i = "100" else  
                E_i(5) when sel_i = "101" else  
                E_i(6) when sel_i = "110" else  
                E (7) ;
```



STD_LOGIC

```
End BEHAVIORAL ;
```

Else cubre todos las restantes combinaciones, no solo "111"

VHDL: Sentencias en VHDL

Ejemplos:

```
architecture BEHAVIORAL of
    when_else_mal is
begin
    Z_o <= '1' when A_i='0' else
        '0' when B_i='1';
end BEHAVIORAL;
```

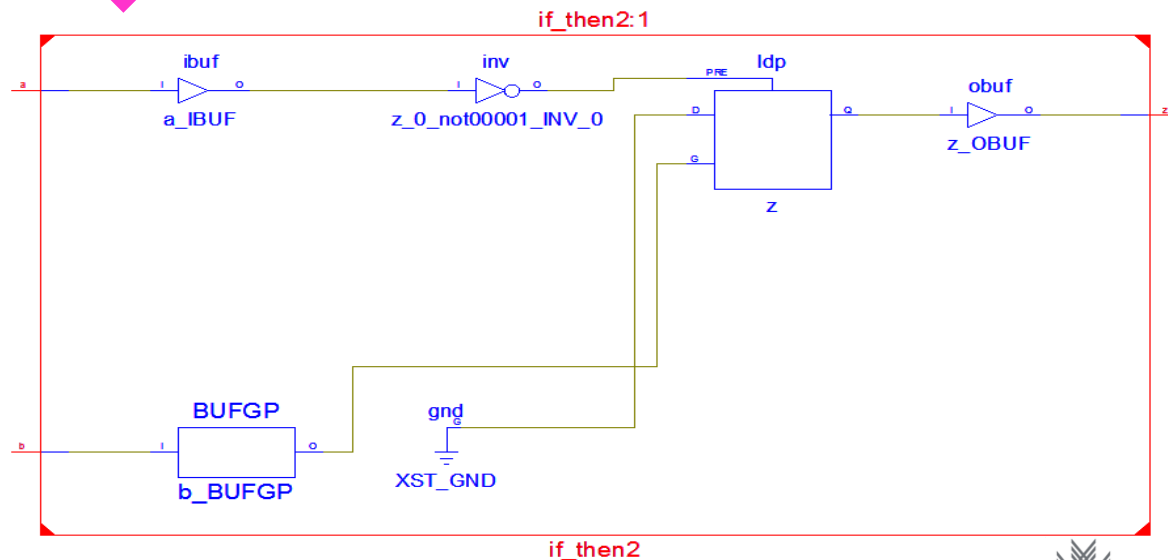
1

WARNING:Xst:737 - Found 1-bit latch for signal <Z_o>. Latches may be generated from incomplete case or if statements.

We do not recommend the use of latches in FPGA/CPLD designs, as they may lead to timing problems.

```
architecture BEHAVIORAL of
    when_else_bien is
begin
    Z_o<='1' when A_i='0'
else
    '0' when B_i='1'
else
    '0';
end BEHAVIORAL;
```

2



VHDL: Sentencias en VHDL

Sentencias concurrentes

➤ Asignación de señal:

3. Selectiva (with – select –when)

```
architecture BEHAVIORAL of MUX_8 is
begin
  with SEL_i select
    Y <= E_i(0) when "000",
        E_i(1)  when "001",
        E_i(2)  when "010",
        E_i(3)  when "011",
        E_i(4)  when "100",
        E_i(5)  when "101",
        E_i(6)  when "110",
        E_i(7)  when others;
end BEHAVIORAL ;
```

STD_LOGIC

When others cubre todas las restantes combinaciones, no solo "111"

VHDL: Sentencias en VHDL

Sentencias concurrentes

Ejemplos:

```
architecture BEHAVIORAL of
with_select_mal is
begin
with DATA_EN_i SELECT
    Y_o <= "00" when "11",
           "01" when "10",
           "10" when "01",
           "11" when "00";
end BEHAVIORAL ;
```

MAL

```
architecture BEHAVIORAL of
with_select_bien is
begin
with DATA_EN_i SELECT
    Y_o <= "00" when "11",
           "01" when "10",
           "10" when "01",
           "11" when others;
end BEHAVIORAL ;
```

BIEN

ERROR:HDLParasers:812 - "C:/PoyectosTDC/MUX_with_Select/with_select.vhd" Line 36.
A value is missing in select.

2.2.2.2 VHDL: Sentencias Secuenciales

VHDL: Sentencias en VHDL

Sentencias concurrentes

➤ Procesos

Un proceso (**process**) es una sentencia concurrente que agrupa en su interior sentencias que se ejecutan **secuencialmente**.

```
<ETIQUETA>: process (<LISTA_SEÑALES>)  
    <DECLARACIONES>  
    begin  
    <SENTENCIAS_SECUENCIALES>  
end process <ETIQ>;
```


Las asignaciones de señales hechas dentro tienen lugar cuando se para el proceso.

VHDL: Sentencias en VHDL

Sentencias secuenciales

➤ Asignación de señal:

```
architecture BEHAVIORAL of EXAMPLE is
begin
P1: process (A_i, C_i)
  Begin
    if A_i = '1' then
      B_o <= C_i;
    else
      B_o <= not C_i;
    end if;
  end process P1;
```


- 
- Siempre dentro de *Process*
 - Lista de sensibilidad: todas las señales que se leen
 - Los procesos pueden estar etiquetados

VHDL: Sentencias en VHDL


Sentencias secuenciales

EJEMPLO: Asignación concurrente vs asignación en un proceso

```
architecture BEHAVIORAL of xor_gate is
Begin
    Z_o <= A_i xor B_i;
end BEHAVIORAL ;
```



```
architecture BEHAVIORAL of xor_gate is
begin
    P1: process ( A_i, B_i)
    Begin
        Z_o <= A_i xor B_i;
    end process P1;
End BEHAVIORAL ;
```



VHDL: Sentencias en VHDL

Sentencias secuenciales

EJEMPLO: Asignación concurrente vs asignación en un proceso

1

```
architecture
BEHAVIORAL of
MULTISOURCE is begin
    Z_o <= A_i;
    Z_o <= B_i;
    Z_o <= C_i;
End BEHAVIORAL;
```

ERROR:Xst:528 - Multi-source in Unit <MULTISOURCE> **on signal <Z_o>**; this signal is connected to multiple drivers.

2

```
architecture BEHAVIORAL of
MULTISOURCE is
Begin
    process (A_i, B_i, C_i)
    begin
        Z_o <= A_i;
        Z_o <= B_i;
        Z_o <= C_i;
    end process;
End BEHAVIORAL;
```

WARNING:Xst:647 - Input <A_i> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <B_i> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.

VHDL: Sentencias en VHDL

Sentencias secuenciales

➤ Control del flujo de ejecución :

1. “IF – THEN – ELSE - END IF” ó “IF – THEN – ELSIF – ELSE - END IF”

```
if <CONDICIÓN> then  
    <SENT_SECUENCIALES>  
[elsif < CONDICIÓN > then  
    <SENT_SECUENCIALES>]  
else  
    <SENT_SECUENCIALES>  
end if;
```

VHDL: Sentencias en VHDL

Sentencias secuenciales

➤ Control del flujo de ejecución :

1. "IF – THEN – ELSE - END IF" ó "IF – THEN – ELSIF – ELSE - END IF"

```
architecture BEHAVIORAL of MUX8 is
begin
  COMB: process (SEL_i)
  begin
    if SEL_i = "000" then
      Y_o <= E_i(0);
    elsif SEL_i = "001" then
      Y_o <= E_i(1);
    elsif SEL_i = "010" then
      Y_o <= E_i(2);
    ...
    else
      Y_o <= E_i(7);
    end if,
```

Sintaxis 1

VHDL: Sentencias en VHDL

Sentencias secuenciales

➤ Control del flujo de ejecución :

1. "IF – THEN – ELSE - END IF" ó "IF – THEN – ELSIF – ELSE - END IF"

```
architecture BEHAVIORAL of MUX8 is
begin
  COMB: process (SEL_i)
  begin
    if      (SEL_i = "000") then Y <= E_i(0) ;
    elsif (SEL_i = "001") then Y <= E_i(1) ;
    elsif (SEL_i = "010") then Y <= E_i(2) ;
        . . .
    elsif (SEL_i = "111") then Y <= E_i(7) ;
    else   Y_o <= E_i(7) ;
  end if;
```

Sintaxis 2

VHDL: Sentencias en VHDL

Sentencias secuenciales

➤ Control del flujo de ejecución :

2. CASE

```
case <EXPR> is  
    when <CHOICE_1> =>  
        <SENTENCIAS_SECUENCIALES>  
    . . .  
    [when others =>  
        <SENTENCIAS_SECUENCIALES>]  
end case;
```

VHDL: Sentencias en VHDL

Sentencias secuenciales

➤ Control del flujo de ejecución :

2. CASE

```
COMB: process (E_i, SEL_i)
begin
    case SEL_i is
        when "000" =>
            Y_o <= E_i(0);
        when "001" =>
            Y_o <= E_i(1);
        ...

        when "110" =>
            Y_o <= E_i(6);
        when others =>
            Y_o <= E_i(7);
    end case;
end process COMB;
```

Sintaxis 1

VHDL: Sentencias en VHDL

➤ Control del flujo de ejecución :

2. CASE

Sintaxis 2

```
architecture BEHAVIORAL of MUX8 is
begin
  COMB: process (E_i, SEL_i)
  begin
    case SEL_i is
      when "000" => Y_o <= E_i(0) ;
      when "001" => Y_o <= E_i(1) ;
      ...
      when "110" => Y_o <= E_i(6) ;
      when "111" => Y_o <= E_i(7) ;
      when others => Y_o <= E_i(7) ;
    end case;
  end process COMB;
end BEHAVIORAL;
```

VHDL: Sentencias en VHDL

➤ Control del flujo de ejecución : Anidar IF y CASE

```
-- architecture
architecture my_case_ex of mux_8to1_ce is
begin
  my_mux: process (SEL, Data_in, CE)
  begin
    if (CE = '1') then
      case (SEL) is
        when "000" => F_CTRL <= Data_in(0);
        when "001" => F_CTRL <= Data_in(1);
        when "010" => F_CTRL <= Data_in(2);
        when "011" => F_CTRL <= Data_in(3);
        when "100" => F_CTRL <= Data_in(4);
        when "101" => F_CTRL <= Data_in(5);
        when "110" => F_CTRL <= Data_in(6);
        when "111" => F_CTRL <= Data_in(7);
        when others => F_CTRL <= '0';
      end case;
    else
      F_CTRL <= '0';
    end if;
  end process my_mux;
end my_case_ex;
```

No es posible
con sentencias
concurrentes

2.2.3. VHDL: Tipos de datos

VHDL: Tipos de datos

Types (Tipos de datos)

- ❑ Tipo de dato es el modo en que se representa la información
- ❑ Intrínsecamente una vez llevado a la FPGA el diseño, todos los datos son binarios. Existen tipos para tratar la información más cómodamente.
- ❑ El tipo de dato establece los valores que puede tomar una **señal** o puerto y las operaciones que pueden ser realizadas con ellos.
- ❑ Existen algunos tipos predefinidos por el lenguaje VHDL. No es necesario invocar ninguna librería adicional.
- ❑ Existen tipos de datos escalares (simples) y vectoriales (compuestos)

VHDL: Tipos de datos

Simples

Tipos de datos: BIT (Predefinido VHDL)

- ❑ Están incluidos en el lenguaje VHDL, en la librería Standard (Invocada por defecto en cualquier módulo VHDL).
- ❑ Establece dos valores posibles : '0' y '1'
- ❑ Fue reemplazado por el STD_LOGIC.

```
entity EJEMPLO1 is
    port ( A_i, B_i : in bit;
           F_o : out bit );
end EJEMPLO1 ;
```


VHDL: Tipos de datos

Simples

Tipos de datos: **BOOLEAN** (Predefinido VHDL)

- ❑ Está incluido en el lenguaje VHDL, en la librería Standard (Invocado por defecto en cualquier módulo VHDL).
- ❑ Establece dos valores posibles : 'Falso' y 'Verdadero'

```
entity BOOLE_4 is
    port (A_i, B_i : in boolean;
          Y_o      : out boolean);
end BOOLE_4;
```

```
architecture BEHAVIORAL of BOOLE_4 is
begin
    Y_o <= true when A_i = false and B_i = false
           else false;
end BEHAVIORAL;
```

VHDL: Tipos de datos (*)

Simples

Tipos de datos: INTEGER (Predefinido VHDL)

❑ Está incluido en el lenguaje VHDL, en la librería Standard (Invocado por defecto en cualquier módulo VHDL).

❑ Enteros positivos y negativos si no se define un subrango.

$[-(2^{31}-1), +(2^{31}-1)] = [-2.147.483.647, +2.147.483.647]$

❑ Para usarlo en síntesis es recomendable definir un subrango.

signal Entero: integer range -2 to 12;

❑ Se recomienda usar solo para constantes e índice en tipos bidimensionales.

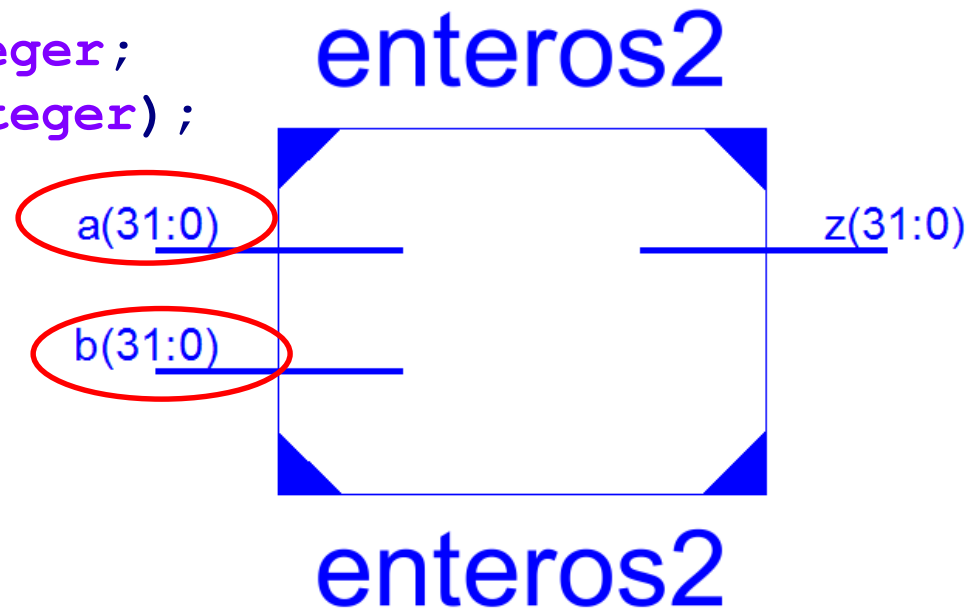
VHDL: Tipos de datos (*)

Simples

Tipos de datos: INTEGER (Predefinido VHDL)

```
--library IEEE;  
--use IEEE.STD_LOGIC_1164.ALL;  
entity ENTEROS is  
    Port (A_i,B_i: in integer;  
          Z_o  : out integer) ;  
end ENTEROS;
```

```
architecture Behavioral of  
ENTEROS is  
begin  
    Z_o <= A_i+B_i;  
end Behavioral;
```



VHDL: Tipos de datos

Simples

Tipos de datos: STD_LOGIC/STD_LOGIC_VECTOR (Estándar de IEEE – Package std_logic_1164)

- ❑ Librería “IEEE” , paquete “standard_logic_1164”
- ❑ Establece 9 valores posibles además del ‘0’ y el ‘1’.

- | | |
|-------|-------------------------------------------------------|
| ❑ ‘U’ | - Sin inicializar. Es el valor por defecto |
| ❑ ‘X’ | - Desconocido (fuerte) |
| ❑ ‘0’ | - Cero lógico (fuerte). Señal puesta a tierra |
| ❑ ‘1’ | - Uno lógico (fuerte). Señal puesta a alimentación |
| ❑ ‘Z’ | - Alta impedancia |
| ❑ ‘W’ | - Desconocido (débil) |
| ❑ ‘L’ | - Cero (débil). Resistencias pull-down |
| ❑ ‘H’ | - Uno (débil). Resistencias pull-up |
| ❑ ‘-’ | - Valor indiferente (“don’t-care”). Usado en síntesis |

VHDL: Tipos de datos

Simples

Tipos de datos: STD_LOGIC/STD_LOGIC_VECTOR (Estándar de IEEE – Package std_logic_1164)

- ❑ Para síntesis solo usaremos; '0', '1', ('Z' inout ports)
- ❑ Operaciones (“std_logic_1164”):
 - Lógicas: and, or, xor, nor, not, nand

VHDL: Tipos de datos

Compuestos (Arrays)

Conjuntos de datos del mismo tipo:

- ❑ BIT_VECTOR

- ❑ STD_LOGIC_VECTOR

`std_logic_vector (<MSB> downto <LSB>)`

`signal LED: std_logic_vector (6 downto 0);`

- ❑ SIGNED, UNSIGNED

VHDL: Tipos de datos

Compuestos (Arrays)

Añadir valores a un tipo compuesto (Aggregate)

❑ Varios caminos :

- A su totalidad **LED <= "1010100";**
- A un elemento **LED (6) <= '1';**
- A un subrango **LED (6 downto 3) <= "1010";**

Ojo a la notación
para 1 bit (' ') y
para más de 1 bit (" ").

VHDL: Tipos de datos

Compuestos (Arrays)

❑ Clausula “others”

```
signal DATABUS : std_logic_vector (6 downto 0) ;  
...  
DATABUS <= “0000000”;
```

```
signal DATABUS : std_logic_vector (6 downto 0) ;  
...  
DATABUS <= (others => '0');
```

“Others” allows clear a whole array
regardless its size

VHDL: Tipos de datos

Compuestos (Arrays)

Tipos de datos: SIGNED y UNSIGNED

- ❑ Es un vector de **std_logic** pero con **significado numérico**
- ❑ **UNSIGNED** → Entero sin signo → Binario puro → 0 to $2^N - 1$
- ❑ **SIGNED** → Entero con signo → Complemento a 2 → $-2^{(N-1)}$ to $2^{(N-1)} - 1$

```
signal A_unsigned      : unsigned(3 downto 0) ;  
signal B_signed        : signed  (3 downto 0) ;
```

```
A_unsigned <= "1111" ;
```

← = 15 decimal

```
B_signed    <= "1111" ;
```

← = -1 decimal

VHDL: Tipos de datos

Compuestos (Arrays)

Tipos de datos: **SIGNED** y **UNSIGNED**

- ❑ Para poder comparar, sumar(+) y restar(-) números binarios necesitamos saber si son números sin signo ó si están en complemento a dos (unsigned ó signed).
- ❑ Declarar el package numeric_std (Descomentar en el módulo de ISE)

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
-- Uncomment the following library declaration if using  
-- arithmetic functions with Signed or Unsigned values  
-- use IEEE.NUMERIC_STD.ALL;
```

VHDL: Tipos de datos

Tipos de datos: SIGNED y UNSIGNED

```
1  -- library declaration
2  library IEEE;
3      use IEEE.std_logic_1164.all; -- defines std_logic_vector type
4      use IEEE.numeric_std.all;    -- defines signed and unsigned types
5  -- entity
6  entity double_sum is
7      Port (
8          in1      : in  std_logic_vector (7 downto 0);
9          in2      : in  std_logic_vector (7 downto 0);
10         out1     : out std_logic_vector (7 downto 0));
11         unsig_in  : in  unsigned(7 downto 0);
12         unsig_out : out unsigned(7 downto 0));
13     end double_sum;
14 -- architecture
15 architecture arch of sum is
16     begin
17         out1 <= in1 + 1; -- ILLEGAL OPERATION, 1 is an integer
18         out1 <= in1 + in2; -- ILLEGAL OPERATION, addition is not defined
19         unsig_out <= unsig_in + 1; -- legal operation
20         unsig_out <= unsigned(in1) + 1; -- legal operation
21         out1 <= std_logic_vector(unsigned(in1) + 1); -- legal operation
22     end arch;
```

VHDL: Tipos de datos

Tipos de datos: SIGNED y UNSIGNED

Mejor práctica de codificación

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;
```

```
entity suma_unsigned is  
  Port (a : in STD_LOGIC_VECTOR (1 downto 0);  
        b : in STD_LOGIC_VECTOR (1 downto 0);  
        c : out STD_LOGIC_VECTOR (1 downto 0));  
end suma_unsigned;
```

```
architecture Behavioral of suma_unsigned is  
  signal sa_us4, sb_us4, resul_us4 : unsigned (1 downto 0);  
begin  
  -- Suma sin acarreo:  
  sa_us4<= unsigned(a); -- Cast de SLV a Unsigned  
  sb_us4<= unsigned(b); -- Cast de SLV a Unsigned  
  resul_us4 <= sa_us4 + sb_us4;  
  c<=std_logic_vector(resul_us4); -- Cast de Unsigned a SLV  
end Behavioral;
```



VHDL: Tipos de datos (*)

Tipos de datos: Definidos por el usuario (type)

❑ En VHDL el usuario puede definir tipos de datos nuevos:

```
type <nombre_tipo> is <tipo_de_dato>;
```

```
type NOMBRE is unsigned(2 downto 0);  
--0,1,2,...,7
```

VHDL: Tipos de datos (*)

Tipos de datos: Definidos por el usuario (type)

❑ En VHDL el usuario puede definir un array de elementos de un tipo compuesto:

```
type <nombre_array> is array <rango_array> of  
<tipo_vector>;
```

```
type RAM_TYPE is array(15 downto 0) of  
                      std_logic_vector(3 downto 0);
```

ram_type

15	"0110"
14	"1110"
...	
1	"0000"
0	"0101"

VHDL: Tipos de datos

Indicaciones sobre los tipos de datos

Tipo de objeto	Tipo de dato
Puertos	STD_LOGIC, STD_LOGIC_VECTOR
Signal	STD_LOGIC, STD_LOGIC_VECTOR SIGNED, UNSIGNED User defined (type)(*)
Constant	INTEGER

- Siempre que sean posible utilizar **tipos compuestos**
- **Limitar** el rango de los tipos INTEGER
- Usar Signed y Unsigned para efectuar **operaciones aritméticas**

2.2.4. VHDL: Conversión de tipos de datos

VHDL: Conversión de Tipos de datos

Signed & Unsigned (elements) <=> Std_Logic

Conversión automática

```
senal_sl <= senal_us(2); -- elemento de unsigned  
                        a std_logic
```

```
senal_slv(0) <= senal_s(1); -- 1 elemento de signed a  
                           1 elemento de std_logic
```

```
senal_us(1) <= senal_slv(0); -- elemento de  
                             std_logic_vector  
                             a bit de unsigned
```

```
senal_s(2) <= senal_us(1); -- elemento de unsigned a  
                           elemento de signed
```

VHDL: Conversión de Tipos de datos

Signed & Unsigned <=> Std_Logic_Vector

Conversión CAST

El ***cast*** es una reinterpretación del significado de la señal, pero la señal mantiene el mismo número de bits y el formato.

```
senal_slv <= std_logic_vector(senal_us); -- de unsigned a
                                           -- std_logic_vector
senal_slv <= std_logic_vector(senal_s);  -- de signed a
                                           -- std_logic_vector
senal_s <= signed (senal_slv); -- de std_logic_vector
                                -- a signed
senal_us <= unsigned (senal_slv); -- de std_logic_vector
                                -- a unsigned
```

P.e. Un vector de 8 bits (standard_logic_vector) sin ningún significado numérico pasa a ser un número codificado en binario con signo o sin signo según usemos el *cast signed* o *unsigned*

VHDL: Conversión de Tipos de datos (*)

- ❑ En VHDL el tipo de datos `std_logic_vector` puede convertirse a `integer`.
- ❑ Para ello primero debe efectuarse la conversión de `std_logic_vector` a `signed/unsigned`.
- ❑ **Aplicación:** actuar como índice en arrays de elementos.

```
RAM(to_integer(unsigned(ADDRESS))) <= DATA_IN;
```

2.2.5. VHDL: Operadores

VHDL: Operadores

Tipo de operadores	Tipo de datos	Operador
Lógicos	STD_LOGIC STD_LOGIC_VECTOR	And, or , nor, nand xor, xnor
Aritméticos	SIGNED/UNSIGNED INTEGER	+, -
Comparación	STD_LOGIC STD_LOGIC_VECTOR SIGNED/UNSIGNED INTEGER	=, /=, >, <, >=, <=
Concatenación	STD_LOGIC STD_LOGIC_VECTOR	&
Desplazamiento	SIGNED/UNSIGNED	sll, slr, rol, ror, sla, sra

VHDL: Operadores

Operadores de desplazamiento

Operator	Name		Example	Result
logical	sll	shift left logical	result <= "10010101" sll 2	"01010100"
	srl	shift right logical	result <= "10010101" srl 3	"00010010"
arithmetic	sla	shift left arithmetic	result <= "10010101" sla 3	"10101111"
	sra	shift right arithmetic	result <= "10010101" sra 2	"11100101"
rotate	rol	rotate left	result <= "101000" rol 2	"100010"
	ror	rotate right	result <= "101001" ror 2	"011010"

Operadores para tipo signed
Ca2 → Signo se conserva y si es
negativo se introducen '1'

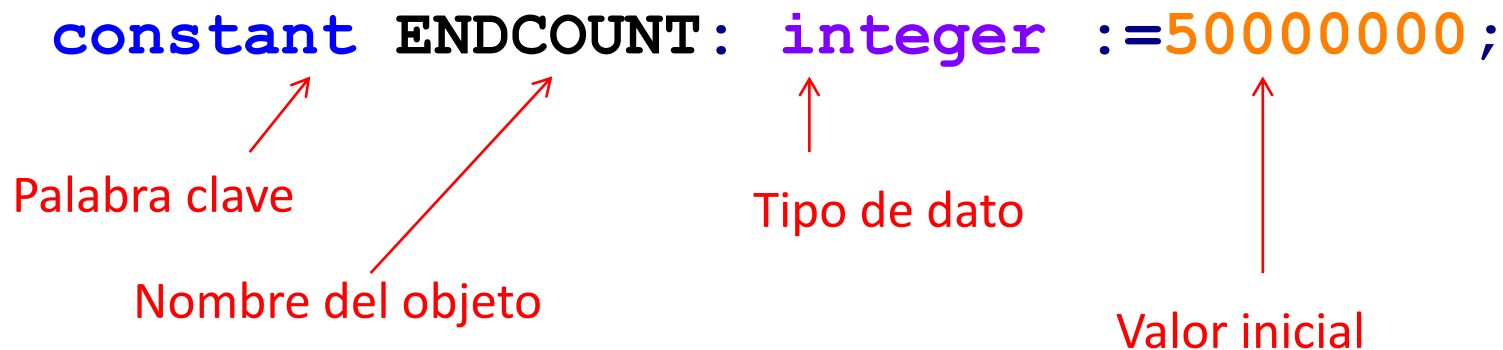
2.2.6. VHDL: Constantes

Tipos de objetos en VHDL: constantes

- ❑ Se pueden declarar en cualquier ámbito (arquitectura)
- ❑ Similar a una constante “software”
- ❑ Su uso es interesante cuando se prevé la reutilización del módulo
- ❑ Hacen más claro el código

constant **ENDCOUNT** : **integer** := **50000000** ;

Palabra clave Nombre del objeto Tipo de dato Valor inicial



The diagram illustrates the syntax of a VHDL constant declaration. It shows the code 'constant ENDCOUNT : integer := 50000000 ;' with four red arrows pointing from labels below to specific parts of the code: 'Palabra clave' points to 'constant', 'Nombre del objeto' points to 'ENDCOUNT', 'Tipo de dato' points to 'integer', and 'Valor inicial' points to '50000000'.

Tipos de objetos en VHDL: constantes

Useful tips:

```
constant ZERO: std_logic_vector  
    (WIDTH-1 downto 0) := (others => '0');
```

```
constant ONES: std_logic_vector  
    (WIDTH-1 downto 0) := (others => '1');
```

```
constant Z_DECO:  
std_logic_vector(WIDTH-1 downto 0) :=  
(0 => '1', others => '0');
```

2.2.7. VHDL: Signals

Tipos de objetos en VHDL: Signals

Tipos de Objetos: señales

- ❑ Tipo de objeto que más se usa en VHDL
- ❑ Son el principal método para mover información dentro de la FPGA.
- ❑ Se declaran dentro de una arquitectura, no se “conectan” con el exterior → No se declaran en la entidad
- ❑ Tras la síntesis las señales se convierten en “cables”.
- ❑ VHDL es fuertemente tipado , cada señal debe ser declarada como de un determinado tipo antes de ser usada. (p.e.: std_logic)

Tipos de objetos en VHDL: Signals

Tipos de Objetos: señales

`signal X,Y: std_logic := '0';`

Palabra clave

Lista de nombres

Tipo de dato

Valor inicial

```
architecture BEHAVIORAL of PRUEBA is
    signal X,Y: std_logic;
begin
    --
end BEHAVIORAL;
```

2.2.8. VHDL: Descripción estructural

VHDL: Descripción estructural

Descripción ESTRUCTURAL

- Hasta el momento solo se ha trabajado con arquitecturas descritas usando técnicas de modelado **DATAFLOW** y **BEHAVIORAL**
- La descripción **ESTRUCTURAL** es una práctica habitual en grandes diseños similar a la programación estructurada o modular software.
- En VHDL el diseño estructural permitirá el **diseño jerárquico**.
- Usar módulos con los elementos funcionales de menor nivel permite **mayor legibilidad** del código
- Se pueden construir librerías con los módulos VHDL
- En resumen expresa un conjunto de objetos y sus interconexiones

VHDL: Descripción estructural

TOP_Module

Entity TOP

Architecture TOP (**Estructural**)

Entity 01

Archit.
(Behavioral)

Module_01

Entity 02

Archit.
(Behavioral)

Module_02

...

Entity n

Archit. N
(Behavioral)

Module_n

VHDL: Descripción estructural

Componentes

- El módulo VHDL que se quiere invocar recibe el nombre de **COMPONENTE**
- Un **COMPONENTE** es la referencia al conjunto formado por una entidad y una arquitectura.
- Para utilizar un **COMPONENTE** en un nuevo módulo VHDL, es preciso declararlo.
- Es posible incluir un solo **COMPONENTE** o varios **COMPONENTE**.
- Cada **COMPONENTE** puede invocarse tantas veces como necesite el diseño de mayor jerarquía.

VHDL: Descripción estructural

Paso 1: Definir la entidad correspondiente al módulo TOP

Paso 2: Declarar los componentes/módulos a usar en el diseño TOP.

Paso 2.1. Añadir una copia del módulo VHDL ya creado en otros proyectos al proyecto TOP

Paso 3: Declarar las señales internas que interconectarán los módulos.

VHDL: Descripción estructural

Paso 4: Instanciar y “*mapear*” los módulos declarados para conseguir el diseño de mayor jerarquía

Instanciar:

- Llamar a un submódulo (unidad de diseño) para incluirlo en la arquitectura del módulo de mayor jerarquía.
- Se puede instanciar un módulo más de una vez, pero **cada instancia** recibirá **un nombre único**.

¿Cómo usaremos
los módulos?

Mapear:

- Asociar las entradas/salidas del submódulo con las entradas/salidas y señales internas del módulo de mayor jerarquía.

VHDL: Descripción estructural

```
begin
u0: MUX2 PORT MAP (
    a => A(0),
    b => B(0),
    sel => sel,
    y => C(0)
);
u1: MUX2 PORT MAP (
    a => A(1),
    b => B(1),
    sel => sel,
    y => C(1)
);
u2: MUX2 PORT MAP (
    a => A(2),
    b => B(2),
    sel => sel,
    y => C(2)
);
u3: MUX2 PORT MAP (
    a => A(3),
    b => B(3),
    sel => sel,
    y => C(3)
);
end Structural;
```

Nombres de las instancias

Nombre del Componente

Mapeo de puertos

VHDL: Descripción estructural

Sintaxis:

```
<nombre_instancia>: <nombre_componente> [port map  
(lista_conexiones)];
```

Las asociaciones dentro de la lista de conexiones o de parámetros genéricos pueden especificarse:

- de forma explícita: {parámetro componente=> parámetro local, }
- ímplicitamente: {parámetro local, }

VHDL: Descripción estructural

Mapeado Implícito

begin

```
N0: inversor port map (I0, I0n);  
N1: inversor port map (I1, I1n);  
A0: puerta_and port map (I0n, I1n, G, Y0);  
A1: puerta_and port map (I0, I1n, G, Y1);  
A2: puerta_and port map (I0n, I1, G, Y2);  
A3: puerta_and port map (I0, i1, G, Y3);
```

end estructural;

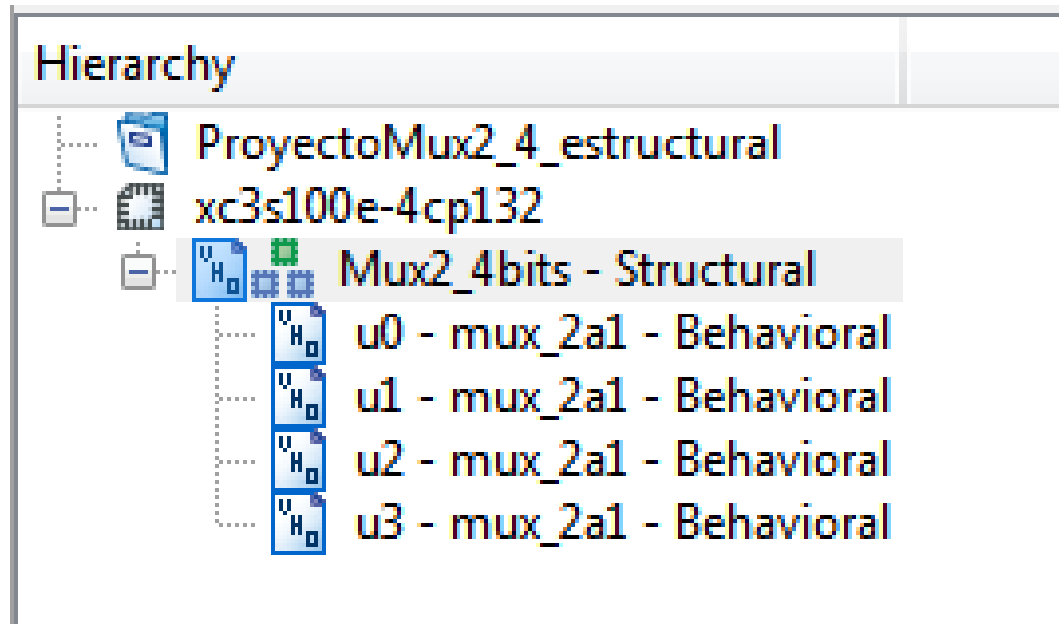
Mapeado Explícito

begin

```
N0: inversor port map (e=>I0,s=> I0n);  
N1: inversor port map (e=>I1,s=> I1n);  
A0: puerta_and port map (e0=>I0n, e1=> I1n, e2=> G, s=> Y0);  
A1: puerta_and port map (e0=> I0, e1=> I1n, e2=> G, s=> Y1);  
A2: puerta_and port map (e0=> I0n, e1=> I1, e2=> G, s=> Y2);  
A3: puerta_and port map (e0=> I0, e1=> i1, e2=> G, s=> Y3);
```

end estructural;

VHDL: Descripción estructural



2.2. VHDL: Bibliografía

- **VHDL FOR LOGIC SYNTHESIS.** Andrew Rushton. 2011 John Wiley & Sons, Ltd. Published
- **Free range VHDL.** Bryan Mealy, Fabrizio Tappero. (Creative Commons). <http://www.freerangefactory.org> (Mayo 2013)
- **VHDL 101.** William Kfig. Editorial Elsevier. 2011