

Arquitecturas Hardware de Comunicaciones

Diseño y Síntesis de Sistemas Digitales con VHDL

Contenidos

- Metodologías de Diseño Electrónico. Introducción al VHDL.
- Diseño de componentes combinacionales estándar.
- Diseño de componentes secuenciales estándar.
- Diseño de memorias y máquinas de estados.

Objetivos del bloque temático

- Desarrollar la habilidad de implementar sistemas microelectrónicos complejos, utilizando técnicas de codiseño hardware/software, herramientas de diseño electrónico y lenguajes (VHDL, ensamblador, C,...) más adecuados disponibles.



Tema 1 y 2:

Metodologías de Diseño Electrónico

Motivación del tema

- Conocimientos previos:
 - Álgebra booleana, simplificación de funciones lógicas
 - Circuitos combinacionales y secuenciales básicos
 - Diagramas de esquemas
- Conocimientos desarrollados:
 - Características de las herramientas de diseño electrónico
 - Justificación del uso de HLDs para diseño de circuitos VLSI
 - Introducción al diseño con VHDL

Objetivos

- Analizar las metodologías de diseño electrónico
- Comprender los niveles de abstracción al describir un sistema electrónico, ubicando en ellos el lenguaje VHDL
- Comprender el sistema de modelado del hardware que implementa el VHDL, mediante *procesos concurrentes*
- Analizar la sintaxis básica del lenguaje, que capacite al alumno para la definición de circuitos sencillos
- Describir, sintetizar y simular los componentes de un sistema a nivel RTL, combinacionales y secuenciales



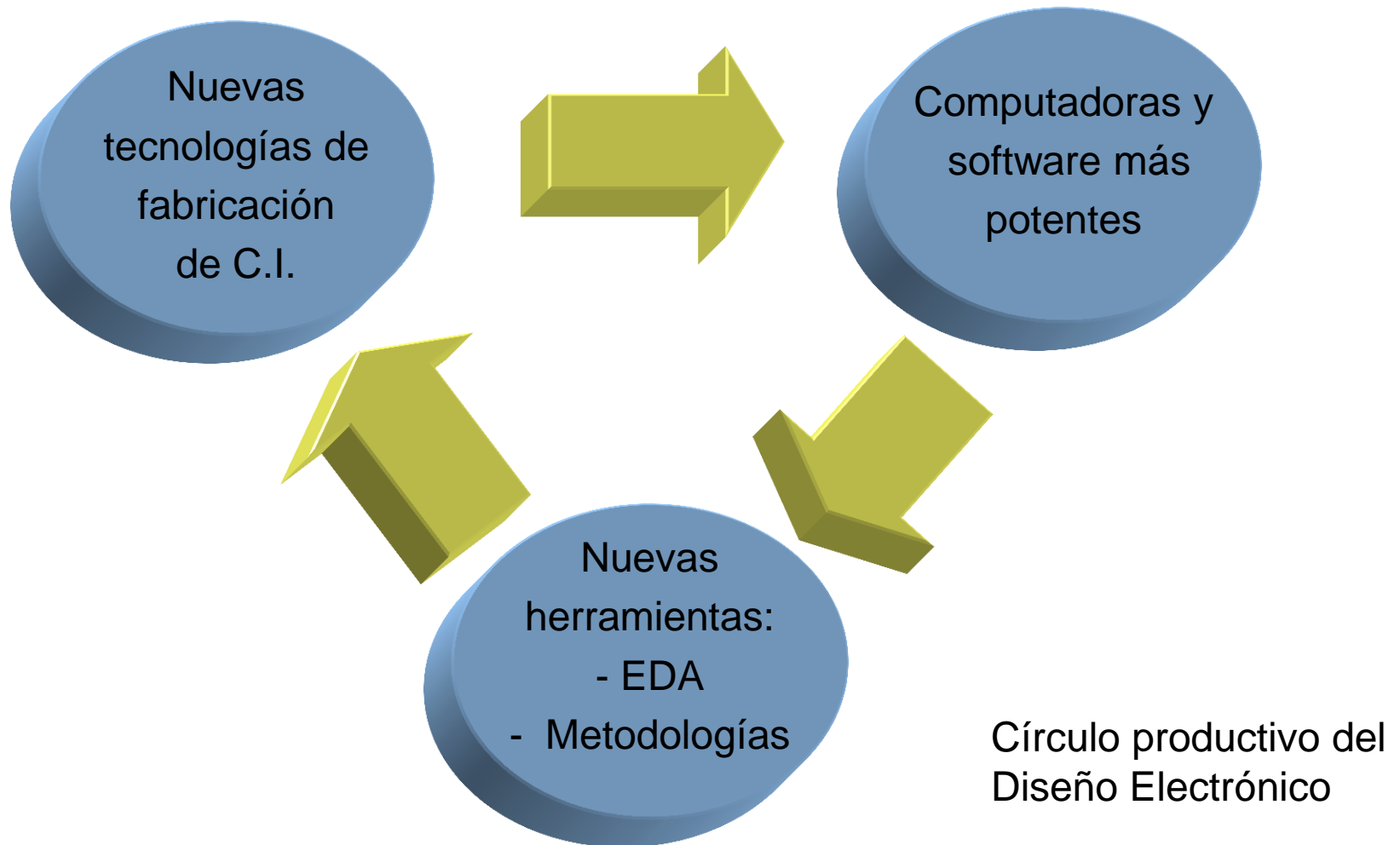
Desarrollo de contenidos

Metodologías de Diseño Electrónico

Sumario

- Introducción
- Niveles de abstracción/descripción de circuitos
- Metodologías de Diseño Electrónico
- Conceptos básicos de VHDL
- Resumen

Introducción



Sumario

- Introducción
- Niveles de abstracción/descripción de circuitos
- Metodologías de Diseño Electrónico
- Conceptos básicos de VHDL
- Resumen

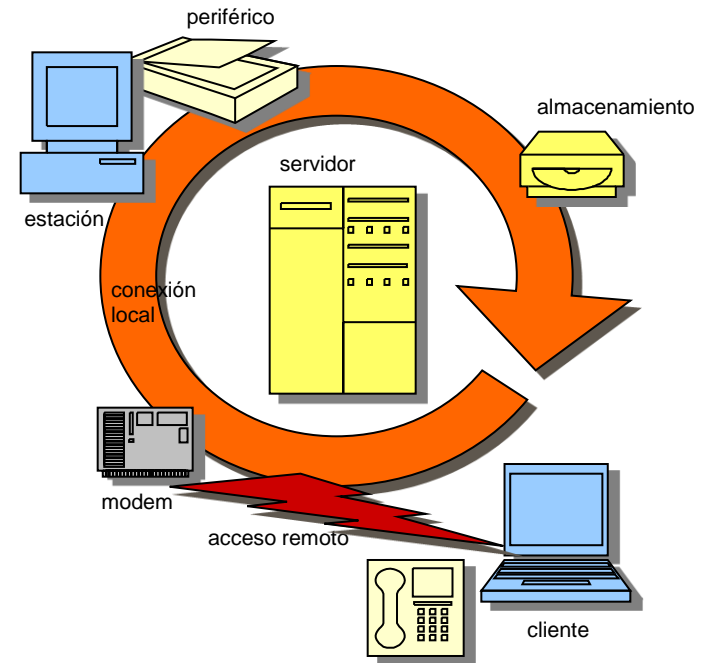
Niveles de abstracción/descripción de circuitos

- Niveles:
 - sistema
 - chip
 - transferencia de registros
 - puertas lógicas
 - circuito eléctrico
 - físico

Niveles de abstracción/descripción de circuitos

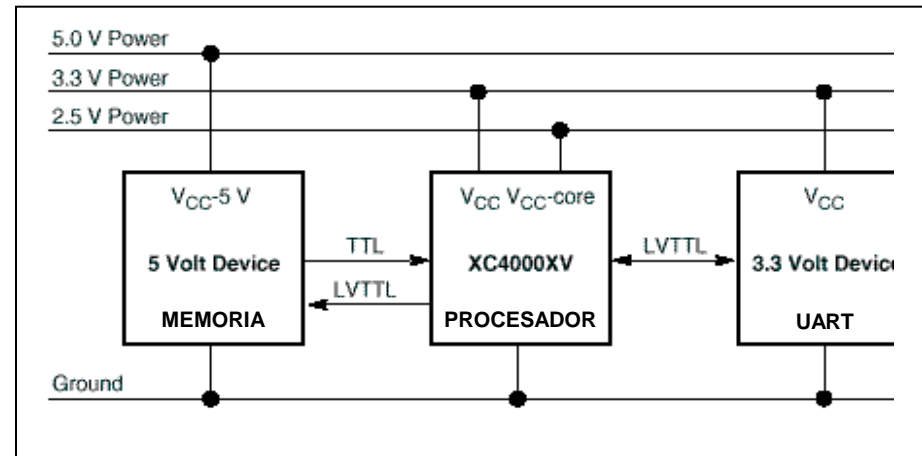
■ Niveles:

- sistema
- chip
- transferencia de registros
- puertas lógicas
- circuito eléctrico
- físico



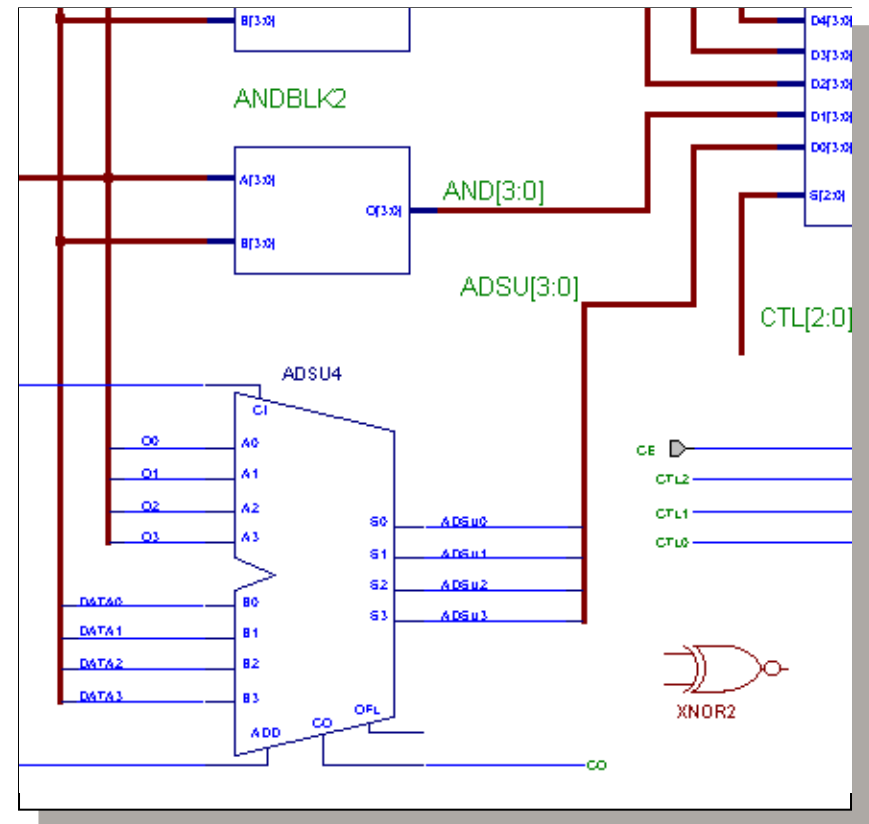
Niveles de abstracción/descripción de circuitos

- Niveles:
 - sistema
 - chip
 - transferencia de registros
 - puertas lógicas
 - circuito eléctrico
 - físico



Niveles de abstracción/descripción de circuitos

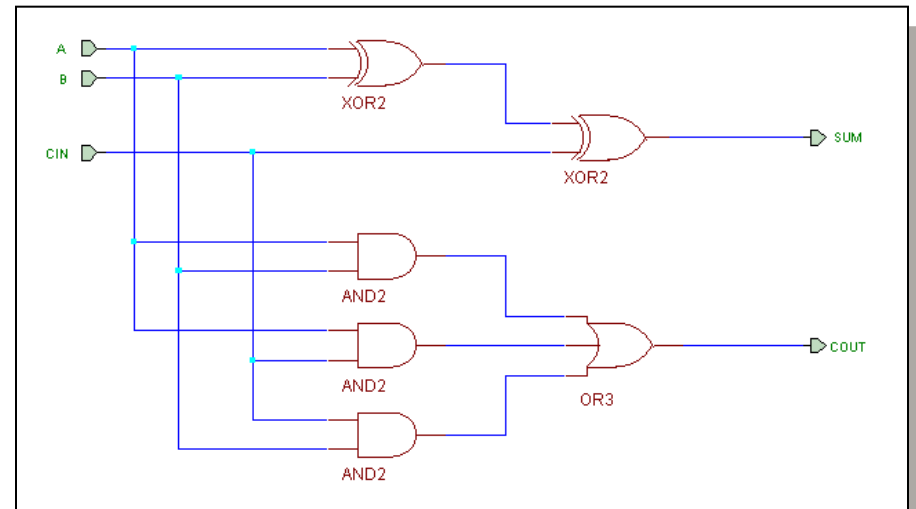
- Niveles:
 - sistema
 - chip
 - transferencia de registros
 - puertas lógicas
 - circuito eléctrico
 - físico



Generado con Xilinx® Foundation® 3.1i

Niveles de abstracción/descripción de circuitos

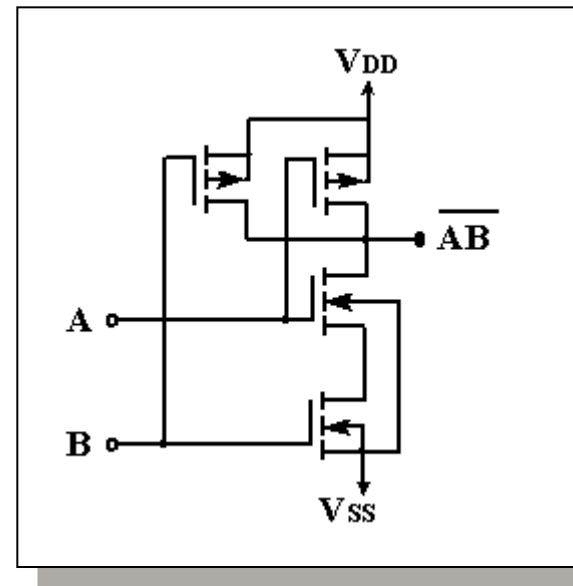
- Niveles:
 - sistema
 - chip
 - transferencia de registros
 - puertas lógicas
 - circuito eléctrico
 - físico



Generado con Xilinx® Foundation® 3.1i

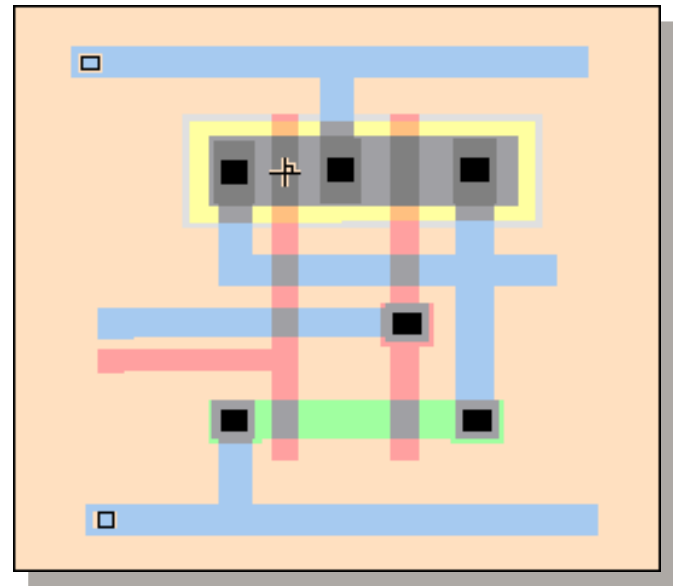
Niveles de abstracción/descripción de circuitos

- Niveles:
 - sistema
 - chip
 - transferencia de registros
 - puertas lógicas
 - circuito eléctrico
 - físico



Niveles de abstracción/descripción de circuitos

- Niveles:
 - sistema
 - chip
 - transferencia de registros
 - puertas lógicas
 - circuito eléctrico
 - físico



Sumario

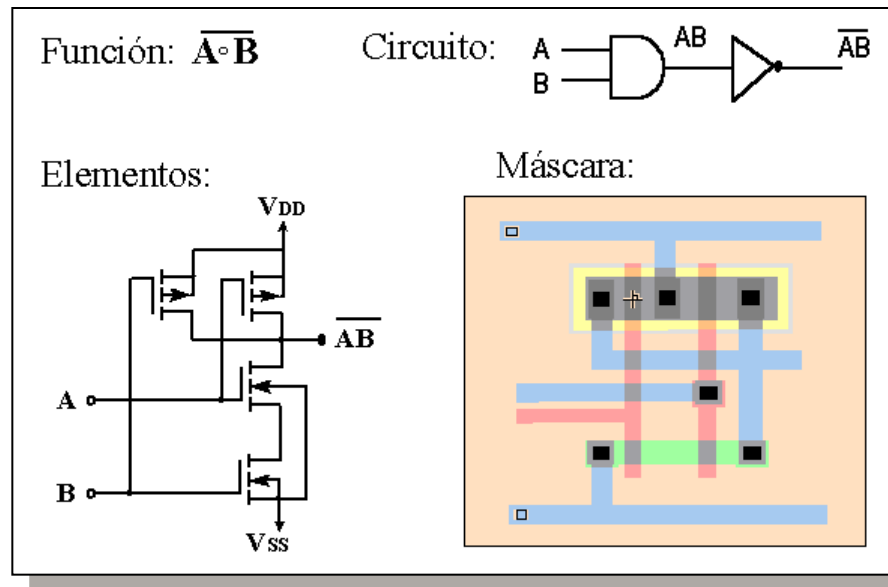
- Introducción
- Niveles de abstracción/descripción de circuitos
- Metodologías de Diseño Electrónico
- Conceptos básicos de VHDL
- Resumen

Metodologías de diseño electrónico

- Perspectiva histórica tradicional:
 - Diseño con transistores
 - Diseño con ecuaciones booleanas
 - Diseño con esquemáticos
 - Diseño con Lenguajes de Descripción Hardware (HDLs)

Diseño a nivel de transistor

- Diseño a medida (*full-custom*)
- Herramientas de diseño físico:
 - Simulación de circuitos eléctricos (SPICE)
 - Diseño de patrones geométricos (*layout*)
 - Verificación de Reglas de Diseño (DRC)



Diseño con ecuaciones booleanas

- Diseño con bibliotecas de *primitivas* (ASICs, PLDs)
- Diseño lógico (a nivel puerta)
- Generación de ecuaciones booleanas
- Técnicas de minimización
- Utilidad: decodificadores, multiplexores, etc.
- Ejemplo:

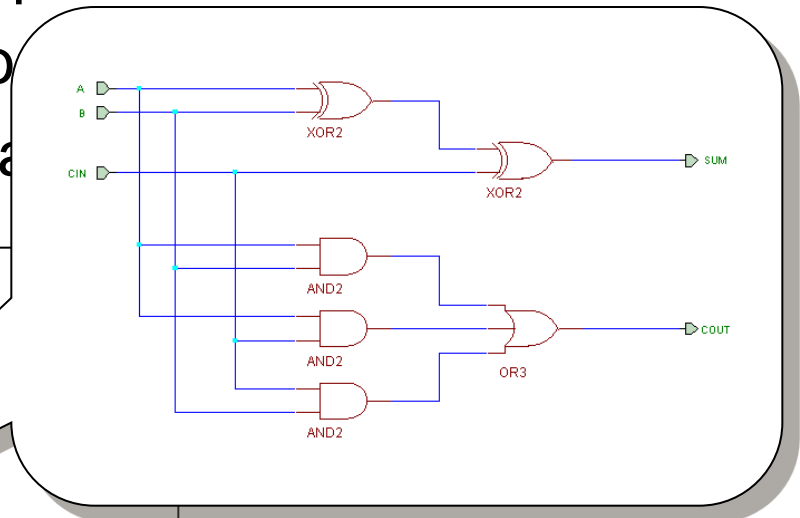
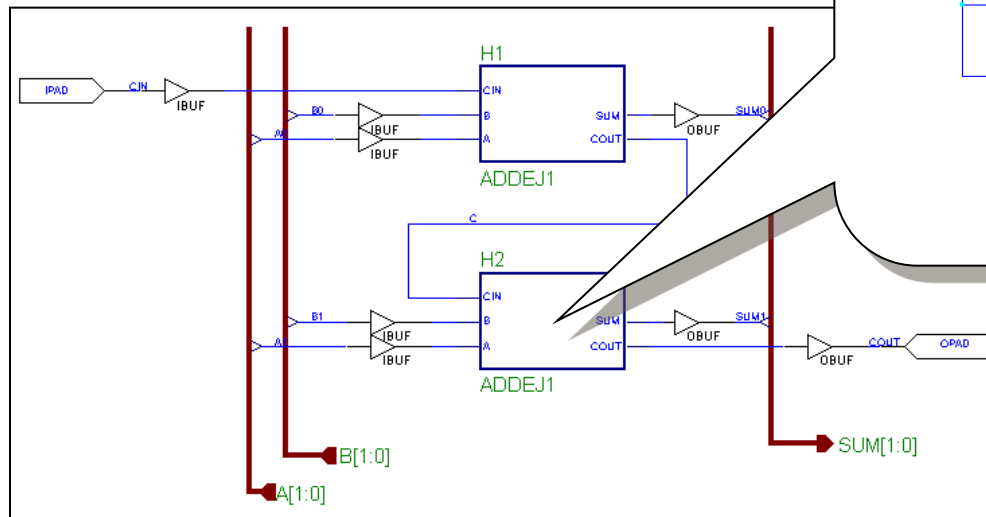
```
sum := (a$b) $cin;  
  
cout := (a&b) # (a&cin) # (b&cin) ;
```

Diseño con esquemáticos

- Concepto de jerarquía
- Bloques funcionales: interfaz + arquitectura
- Utilización de librerías de bloques
- Niveles de anidación ilimitados
- Metodología “de abajo hacia arriba” (*bottom-up*)

Diseño con esquemáticos

- Concepto de jerarquía
- Bloques funcionales: interfaz + arquitectura
- Utilización de librerías de bloques
- Niveles de anidación ilimitados
- Metodología “de abajo hacia arriba”



Generado con Xilinx® Foundation® 3.1i

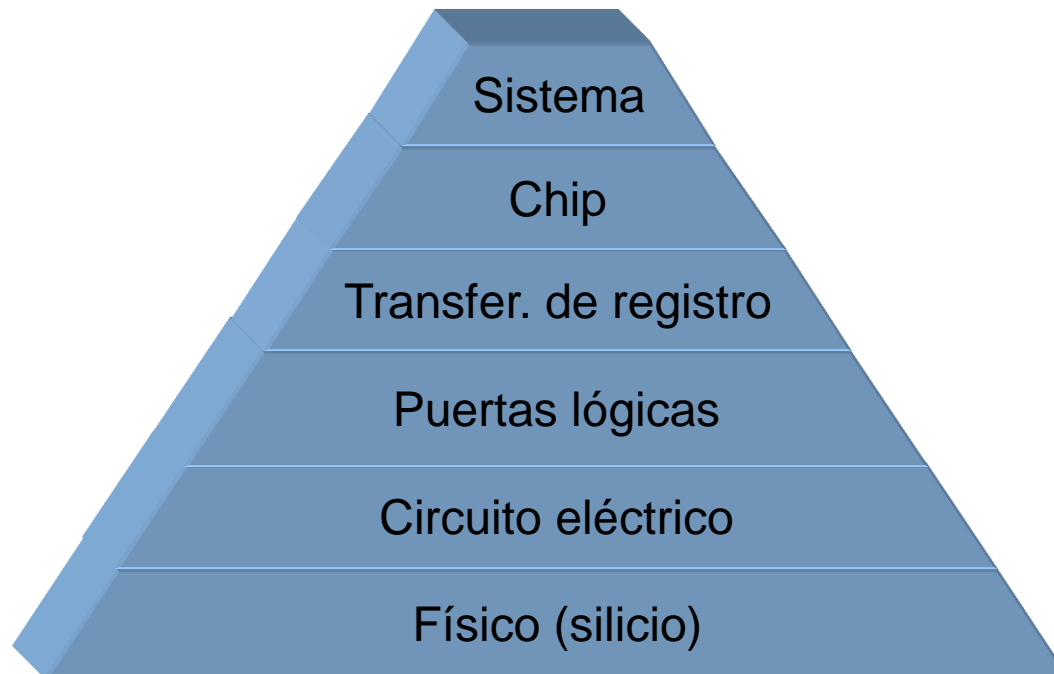
Diseño con HDLs

- HDL = *Hardware Description Language*
- Descripción textual del circuito (no gráfica)
- Adecuado para diseños complejos
- Especificación estructural o comportamental
- Metodología “de arriba hacia abajo” (*top-down*)
- Independencia de la tecnología hasta la síntesis
- Ejemplos: VHDL, Verilog, Abel, SystemC, Handel-C...

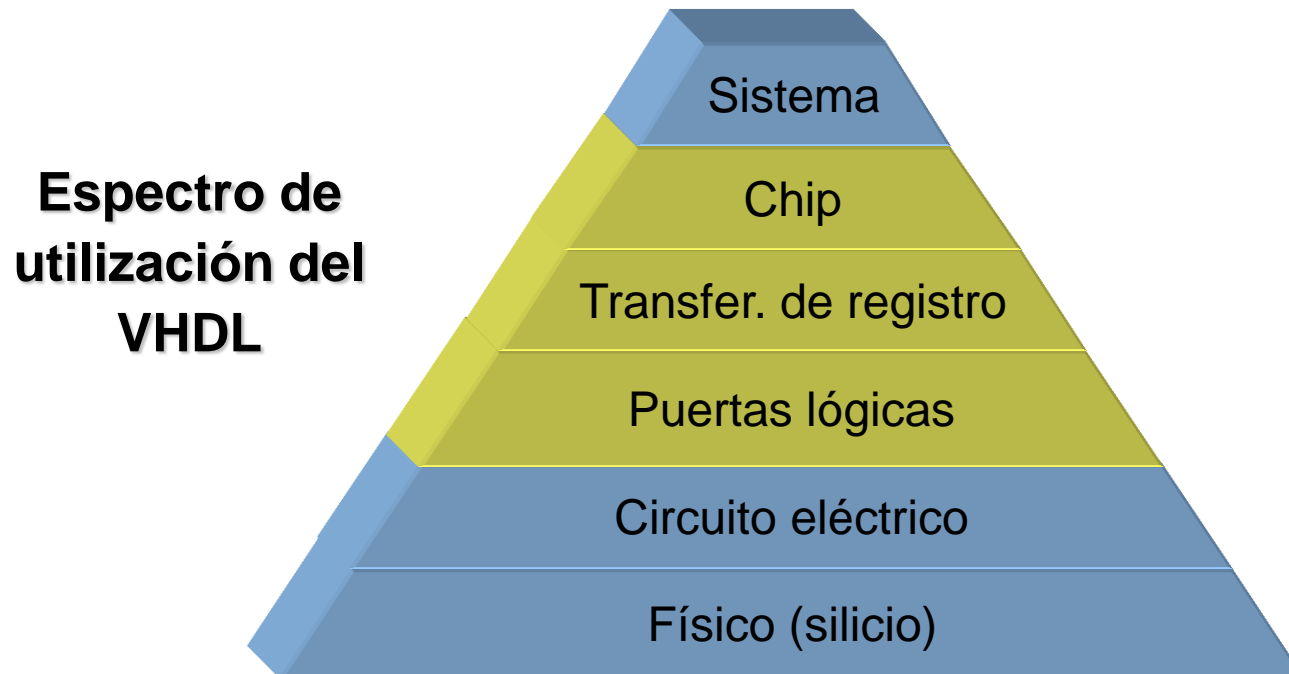
Sumario

- Introducción
- Niveles de abstracción/descripción de circuitos
Metodologías de Diseño Electrónico
- Conceptos básicos de VHDL
- Resumen

VHDL: ámbito de aplicación



VHDL: ámbito de aplicación



VHDL: sintaxis general

- VHDL es insensible a mayúsculas/minúsculas
- Utiliza “;” para terminar una sentencia
- Utiliza “--” para indicar un comentario
- Altamente tipado: poca conversión de tipos automática
- Sentencias/sintaxis similares a otros lenguajes:
 - Asignación a señal
 - Control de flujo condicional
 - Selección
 - Bucles
 - Procesos

VHDL: sintaxis general

- VHDL es insensible a mayúsculas/minúsculas
 - Utiliza “;” para terminar una sentencia
 - Utiliza “--” para comentarios
 - Altamente sensible a la puntuación
 - Sentencias
 - Asignación
 - Control de flujo
 - Selección
 - Bucles
 - Procesos
- Convention

 - **Keywords** in upper case
 - BEGIN, END, ENTITY, ARCHITECTURE, LOOP, ...
 - **Variables** in lower case
 - i, j, k, clock, ...
 - **Types** in lower case
 - integer, std_logic, std_logic_vector

This is just a convention – you can choose your own!

VHDL: sintaxis general

- VHDL es insensible a mayúsculas/minúsculas
- Utiliza “;” para terminar una sentencia
- Utiliza “--” para indicar un comentario
- Altamente tipado: poca conversión de tipos automática
- Sentencias/sintaxis similares a otros lenguajes:
 - Asignación a señal
 - Control de flujo condicional
 - Selección
 - Bucles
 - Procesos

```
--comentario: 3 ejemplos
--de sentencias de asignación

suma <= (a AND b) AND cin;

reg <= "00101" XOR dato;

salida <= datoA + datoB;
```

VHDL: sintaxis general

- VHDL es insensible a mayúsculas/minúsculas
- Utiliza “;” para terminar una sentencia
- Utiliza “--” para indicar un comentario
- Altamente tipado: poca conversión de tipos automática
- Sentencias/sintaxis similares a otros lenguajes:
 - Asignación a señal
 - Control de flujo condicional
 - Selección
 - Bucles
 - Procesos

```
IF (control='1') THEN
    salida <= datoA AND datoB;
ELSE
    salida <= datoA OR datoB;
END IF;
```

VHDL: sintaxis general

- VHDL es insensible a mayúsculas/minúsculas
- Utiliza “;” para terminar una sentencia
- Utiliza “--” para indicar un comentario
- Altamente tipado: poca conversión de tipos automática
- Sentencias/sintaxis similares a otros lenguajes:
 - Asignación a señal
 - Control de flujo condicional
 - Selección
 - Bucles
 - Procesos

```
CASE sel IS
    WHEN "00" =>
        sal <= a AND b;
    WHEN "01" =>
        sal <= a OR b;
    WHEN OTHERS =>
        sal <= "0000";
END CASE;
```


VHDL: sintaxis general

- VHDL es insensible a mayúsculas/minúsculas
- Utiliza “;” para terminar una sentencia
- Utiliza “--” para indicar un comentario
- Altamente tipado: poca conversión de tipos automática
- Sentencias/sintaxis similares a otros lenguajes:
 - Asignación a señal
 - Control de flujo condicional
 - Selección
 - Bucles
 - Procesos

```
FOR i IN 1 TO 8 LOOP  
    salida(9-i) <= entrada(i);  
END LOOP;
```

VHDL: sintaxis general

- VHDL es insensible a mayúsculas/minúsculas
- Utiliza “;” para terminar una sentencia
- Utiliza “--” para indicar un comentario
- Altamente tipado: poca conversión de tipos automática
- Sentencias/sintaxis similares a otros lenguajes:
 - Asignación a señal
 - Control de flujo condicional
 - Selección
 - Bucles
 - Procesos

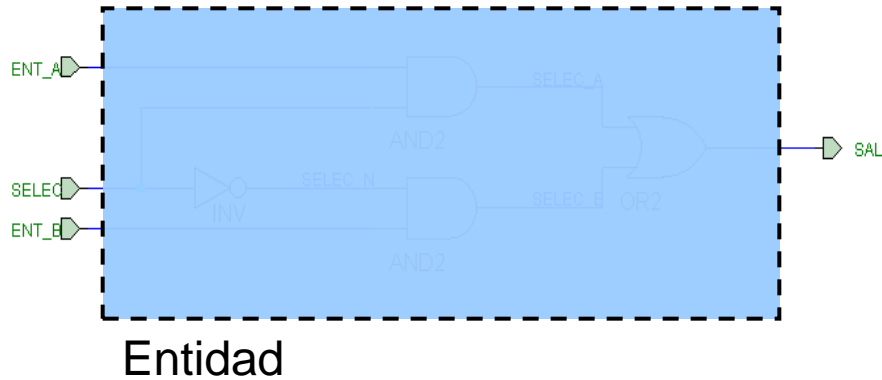
```
PROCESS (señal1, señal2)
    --declaracion de variables
BEGIN
    --cuerpo del proceso
    ...
    --sentencias secuenciales;

END PROCESS;
```

Ejemplo: multiplexor (I)

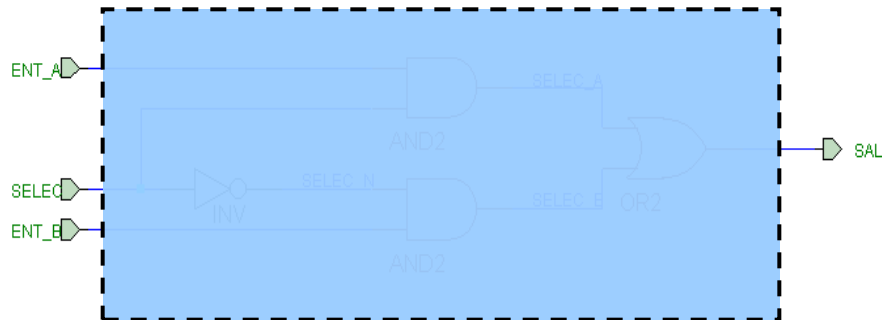
Ejemplo: multiplexor (I)

- Circuito lógico:



Ejemplo: multiplexor (I)

- Circuito lógico:

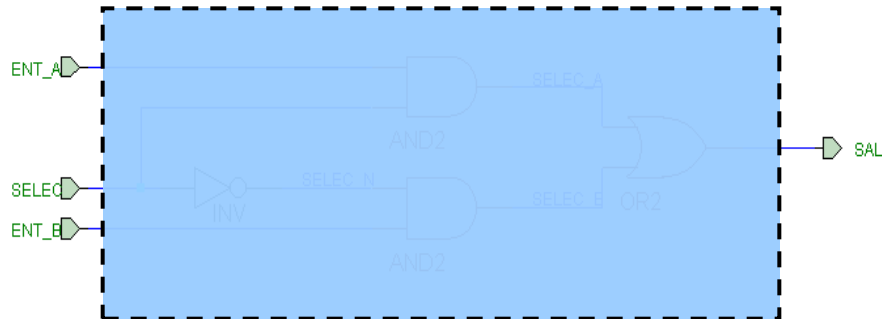


Entidad

Es la interfaz del circuito (módulo) con el exterior. Define (principalmente):
Entradas, salidas, tipos de señales y tamaño.

Ejemplo: multiplexor (I)

■ Circuito lógico:



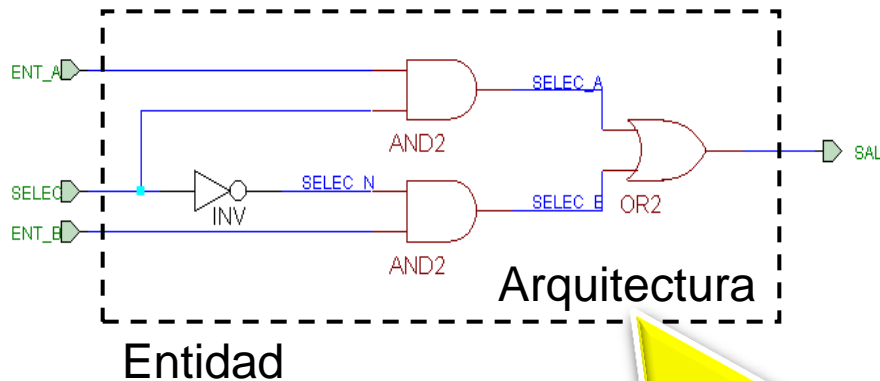
Entidad

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

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT (ent_a: IN std_logic;
          ent_b: IN std_logic;
          selec: IN std_logic;
          sal  : OUT std_logic);
END mux2to1;
```

Ejemplo: multiplexor (I)

■ Circuito lógico:



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

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT (ent_a: IN std_logic;
          ent_b: IN std_logic;
          selec: IN std_logic;
          sal  : OUT std_logic);
END mux2to1;
```

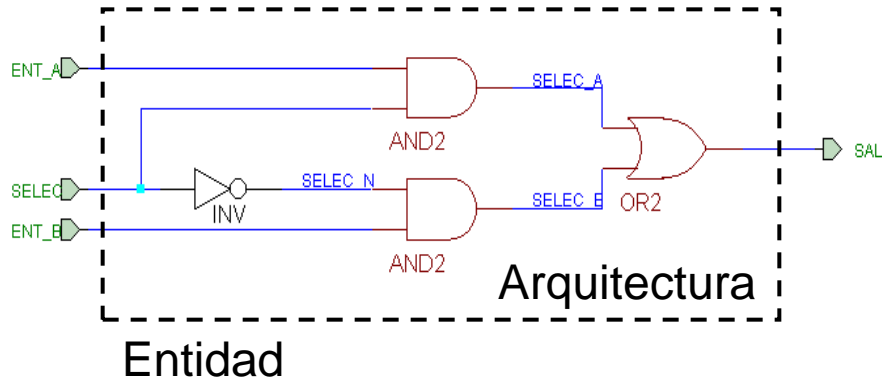
Define la implementación del circuito (módulo). Define (principalmente):

Su comportamiento y/o su estructura.

Es posible utilizar diferentes “estilos” descriptivos...

Ejemplo: multiplexor (I)

■ Circuito lógico:

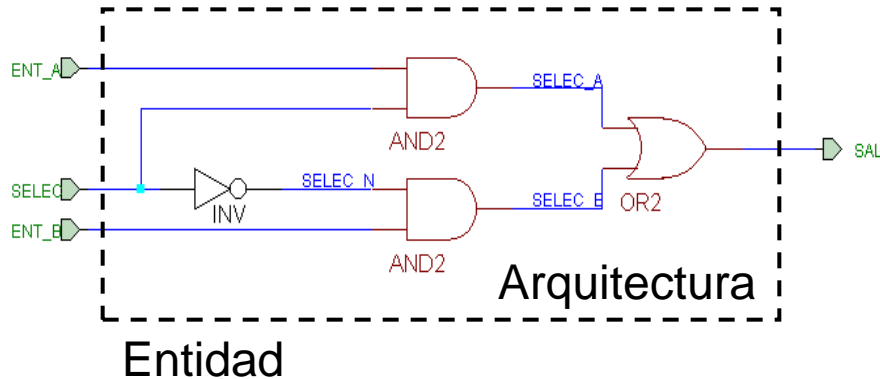


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT (ent_a: IN std_logic;
          ent_b: IN std_logic;
          selec: IN std_logic;
          sal  : OUT std_logic);
END mux2to1;
```


Ejemplo: multiplexor (I)

■ Circuito lógico:



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

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT    (ent_a: IN std_logic;
             ent_b: IN std_logic;
             selec: IN std_logic;
             sal  : OUT std_logic);
END mux2to1;
```

```
ARCHITECTURE flujo_datos OF mux2to1 IS

    SIGNAL selec_n, selec_a, selec_b: std_logic;

BEGIN

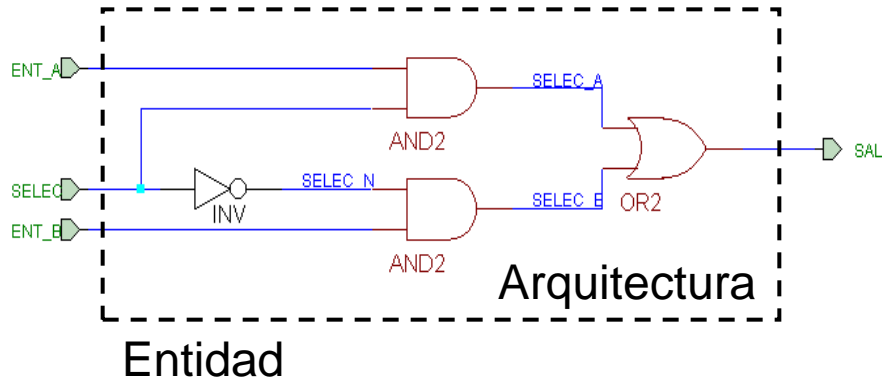
    selec_n <= NOT(selec);
    selec_a <= selec AND ent_a;
    selec_b <= selec_n AND ent_b;
    sal <= selec_a OR selec_b;

END flujo_datos;
```

Descripción de flujo de datos ("Comportamental")

Ejemplo: multiplexor (II)

■ Circuito lógico:

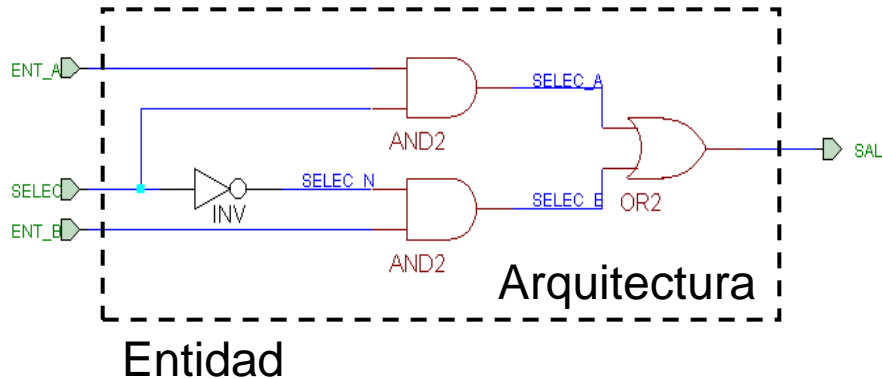


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT (ent_a: IN std_logic;
          ent_b: IN std_logic;
          selec: IN std_logic;
          sal  : OUT std_logic);
END mux2to1;
```

Ejemplo: multiplexor (II)

■ Circuito lógico:



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

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT (ent_a: IN std_logic;
          ent_b: IN std_logic;
          selec: IN std_logic;
          sal  : OUT std_logic);
END mux2to1;
```

```
ARQUITECTURE comportamental OF mux2to1 IS
BEGIN
```

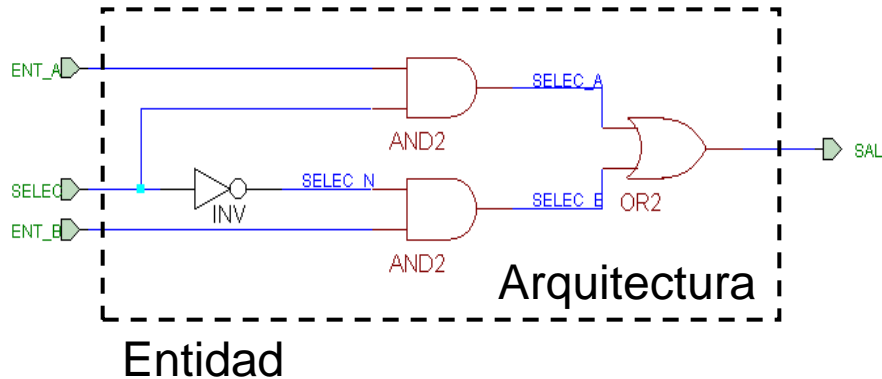
```
    PROCESS (ent_a, ent_b, selec)
    BEGIN
        IF (selec='1') THEN
            sal<=ent_a;
        ELSE
            sal<=ent_b;
        END IF;
    END PROCESS;
```

```
END comportamental;
```

Descripción algorítmica
("Comportamental")

Ejemplo: multiplexor (III)

■ Circuito lógico:

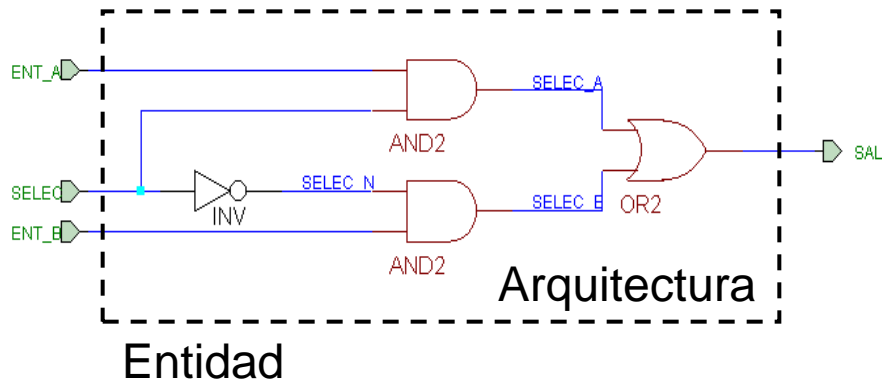


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT (ent_a: IN std_logic;
          ent_b: IN std_logic;
          selec: IN std_logic;
          sal  : OUT std_logic);
END mux2to1;
```

Ejemplo: multiplexor (III)

■ Circuito lógico:



Descripción estructural

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

--Declaración de la entidad del circuito
ENTITY mux2to1 IS
    PORT    (ent_a: IN std_logic;
             ent_b: IN std_logic;
             selec: IN std_logic;
             sal  : OUT std_logic);
END mux2to1;
```

```
ARCHITECTURE estructural OF mux2to1 IS
    SIGNAL selec_n, selec_a, selec_b: std_logic;
    COMPONENT puerta_and IS
        PORT    (e1: IN std_logic; e2: IN std_logic;
                 s: OUT std_logic);
    END COMPONENT;
    COMPONENT puerta_or IS
        PORT    (e1: IN std_logic; e2: IN std_logic;
                 s: OUT std_logic);
    END COMPONENT;
    COMPONENT puerta_inv IS
        PORT    (e: IN std_logic;
                 s: OUT std_logic);
    END COMPONENT;
BEGIN

    I_1: puerta_inv PORT MAP(selec, selec_n);
    I_2: puerta_and PORT MAP(ent_a, selec, selec_a);
    I_3: puerta_and PORT MAP(ent_b,selec_n, selec_b);
    I_4: puerta_or PORT MAP(selec_a, selec_b, sal);

END estructural;
```

Sumario

- Introducción
- Niveles de abstracción/descripción de circuitos
Metodologías de Diseño Electrónico
- Conceptos básicos de VHDL
- Resumen

Resumen

- Existen diferentes niveles de descripción de circuitos
- Las herramientas de CAD electrónico utilizan diferentes metodologías de diseño según los niveles sobre los que actúan.
- Los HDLs son la mejor alternativa para diseño digital VLSI: mayor capacidad descriptiva, legibilidad, reusabilidad, e independencia de la tecnología.
- Las descripciones en VHDL se basan en la definición de **una entidad** y **una o varias arquitecturas** asociadas.
- La funcionalidad de la arquitectura se especifica mediante **uno o varios procesos concurrentes**.

Resumen

- La descripción de la **arquitectura** puede realizarse utilizando un estilo comportamental (de flujo de datos, o algorítmico) o estructural.
- **Importante:** **Varios estilos** descriptivos pueden **mezclarse** en una misma arquitectura!!!
- **Ejercicio 1** (dificultad *): completar el modelo estructural del multiplexor propuesto en el ejemplo.
- **Ejercicio 2** (dificultad **): modificar los tres modelos propuestos para el multiplexor para el caso de entradas de varios bits de tamaño. Hint: usar el tipo de datos `std_logic_vector (n-1 downto 0)`.



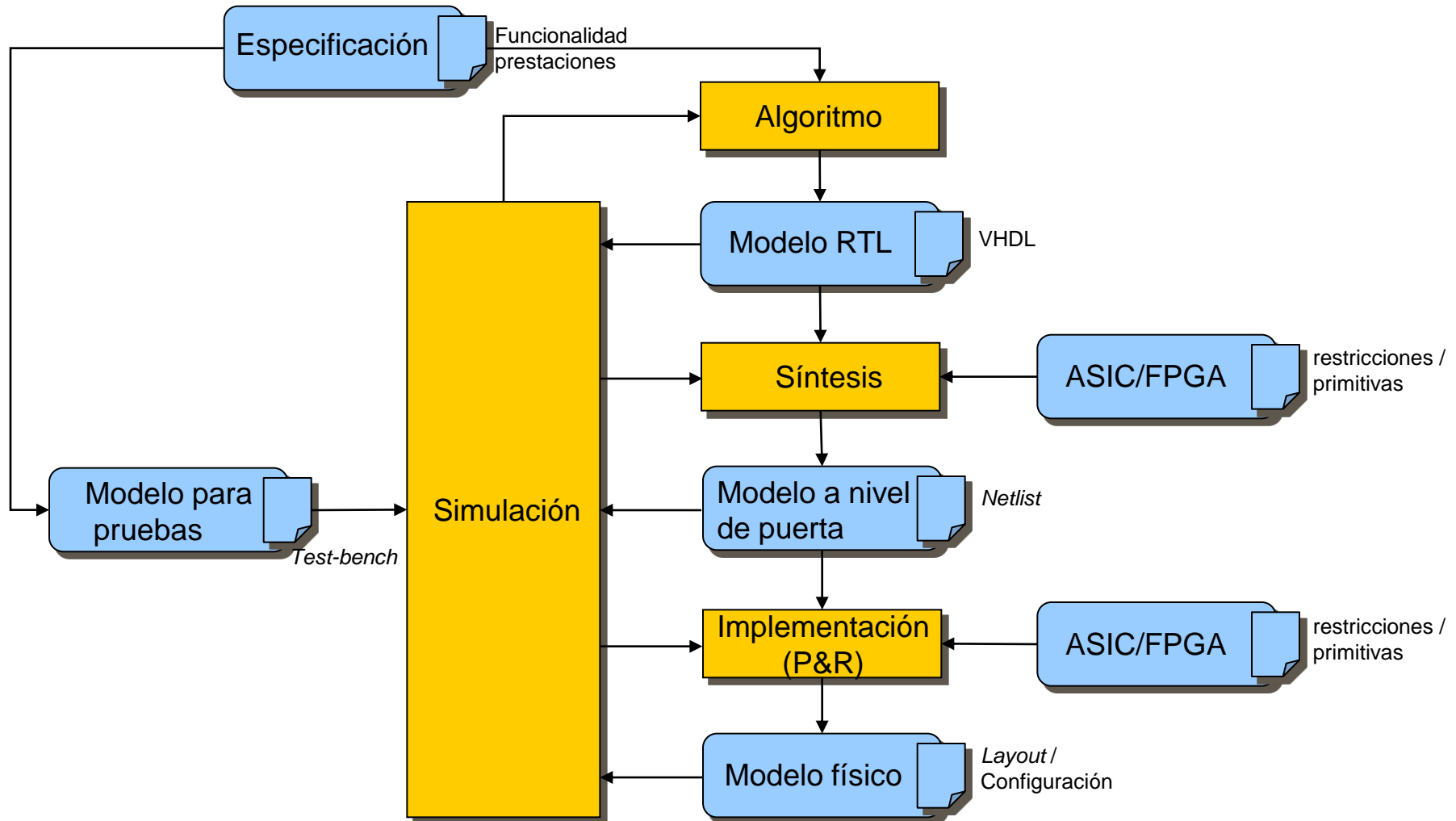
Tema 3:

Introducción al software ISE 6.2i de Xilinx Inc.

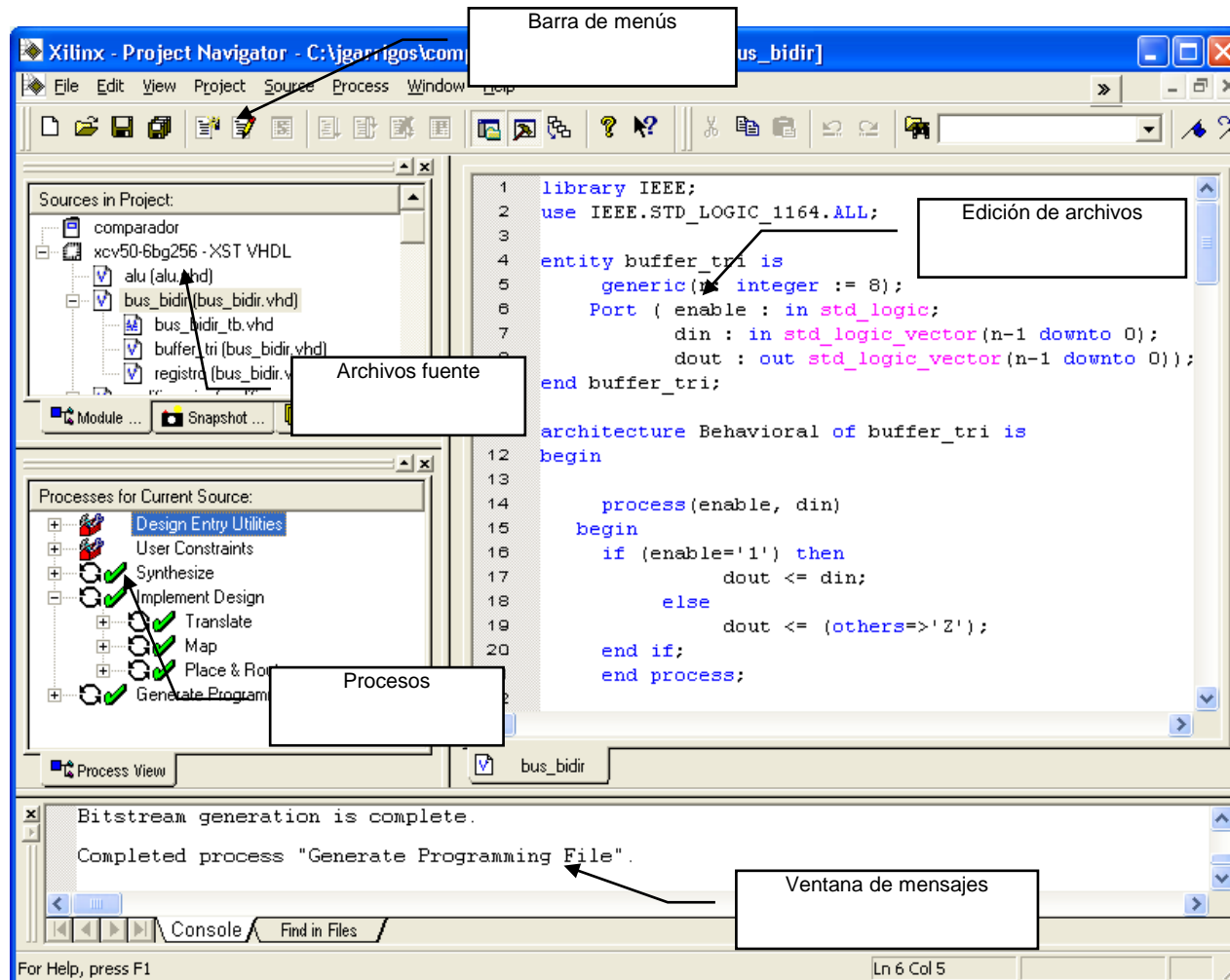
Sumario

- Introducción
- Metodología de diseño
- Aspecto general del ISE6.2i
- Detalle de los procesos
- Parámetros de síntesis con XST
- Parámetros de implementación
- Procesos de Simulación
- Simulación con ModelSim 5.7

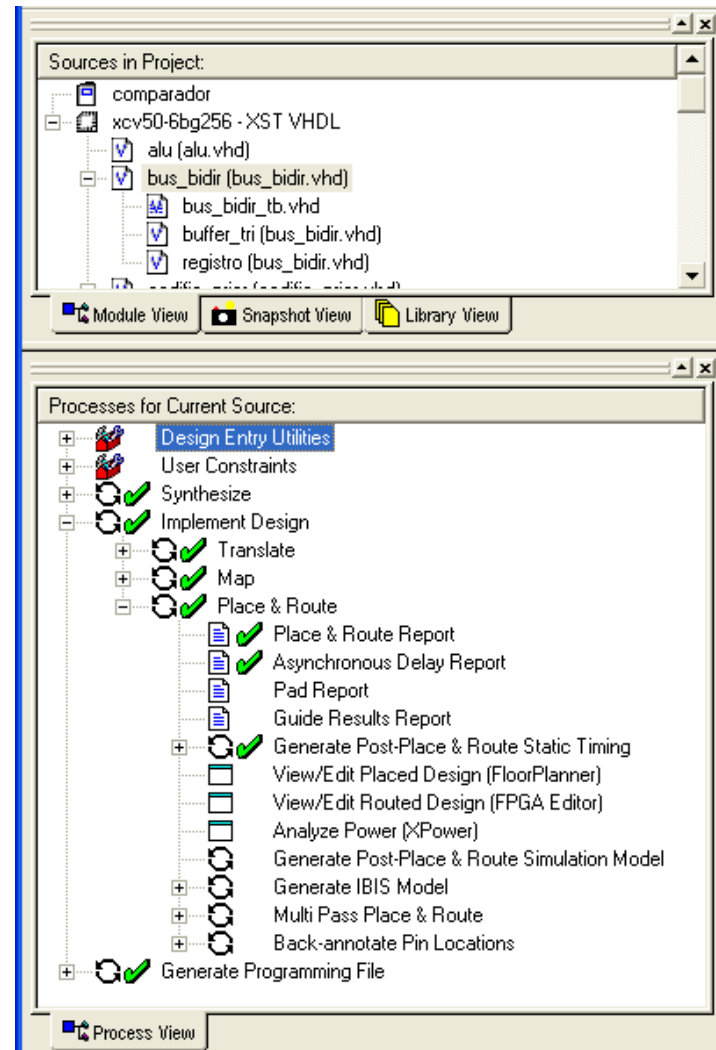
VHDL: metodología de diseño



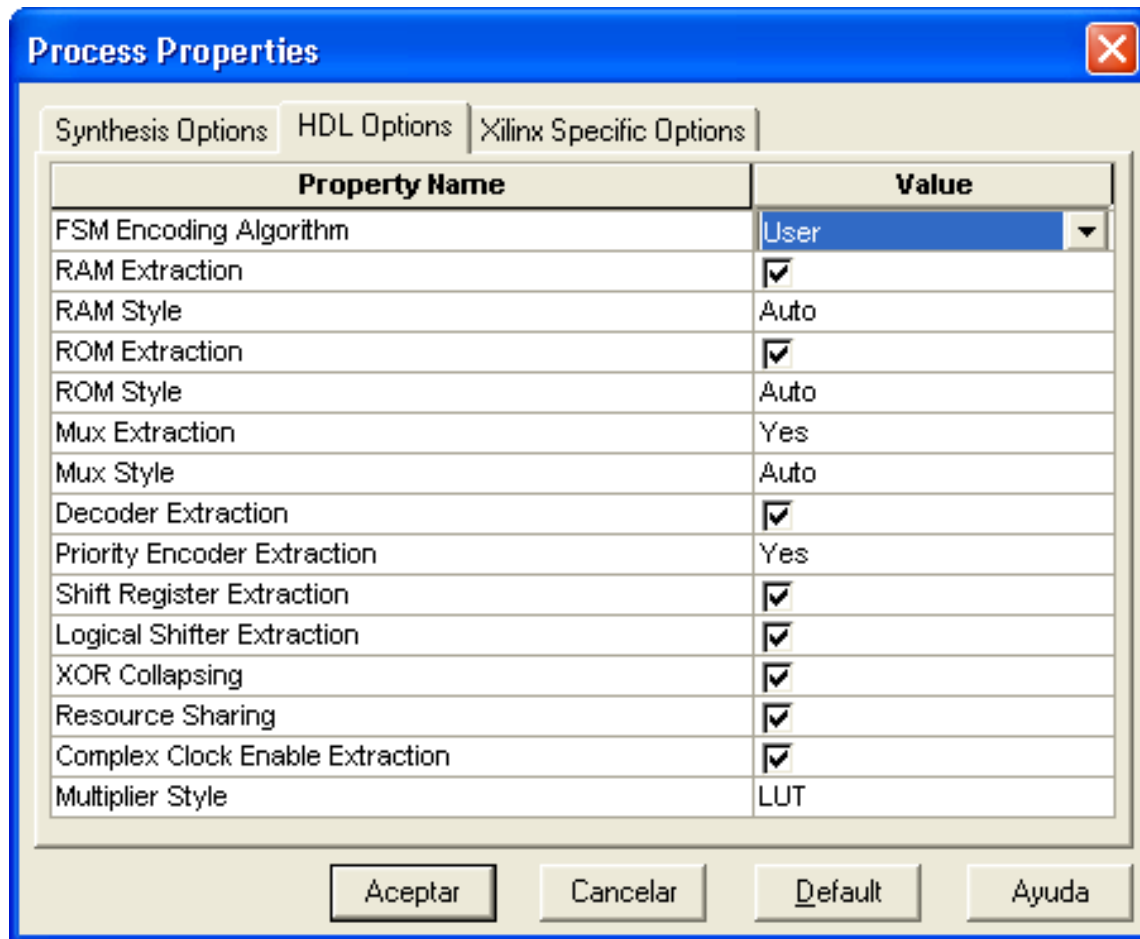
Aspecto general del ISE6.2i



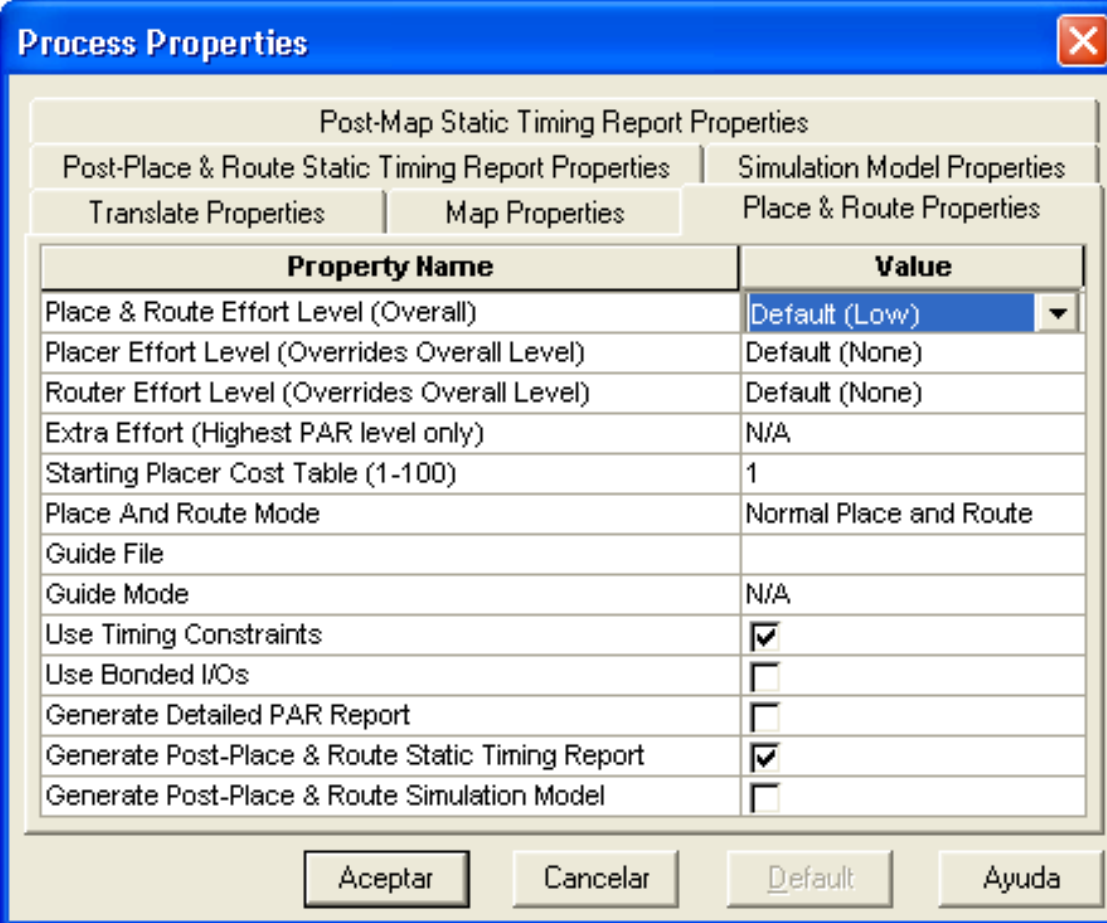
Detalle de los procesos



Parámetros de síntesis con XST



Parámetros de implementación

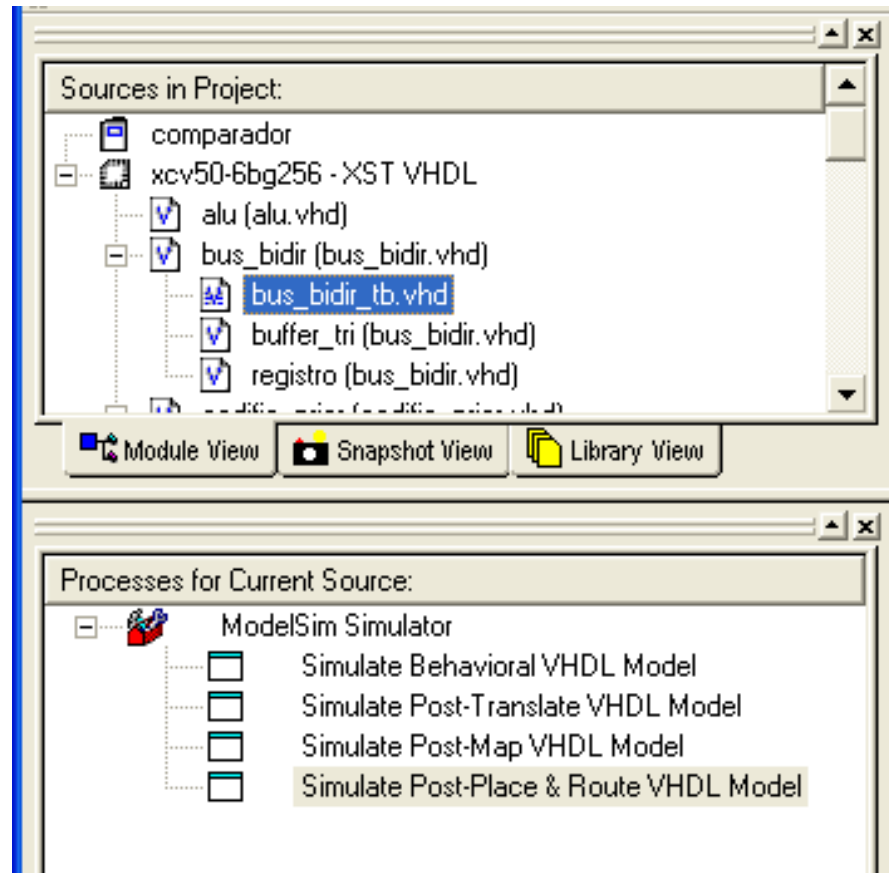


The image shows a 'Process Properties' dialog box with a blue title bar and a close button (X) in the top right corner. The dialog is divided into several tabs. The 'Post-Map Static Timing Report Properties' tab is selected and active. Below the tabs, there is a table with two columns: 'Property Name' and 'Value'. The table contains the following rows:

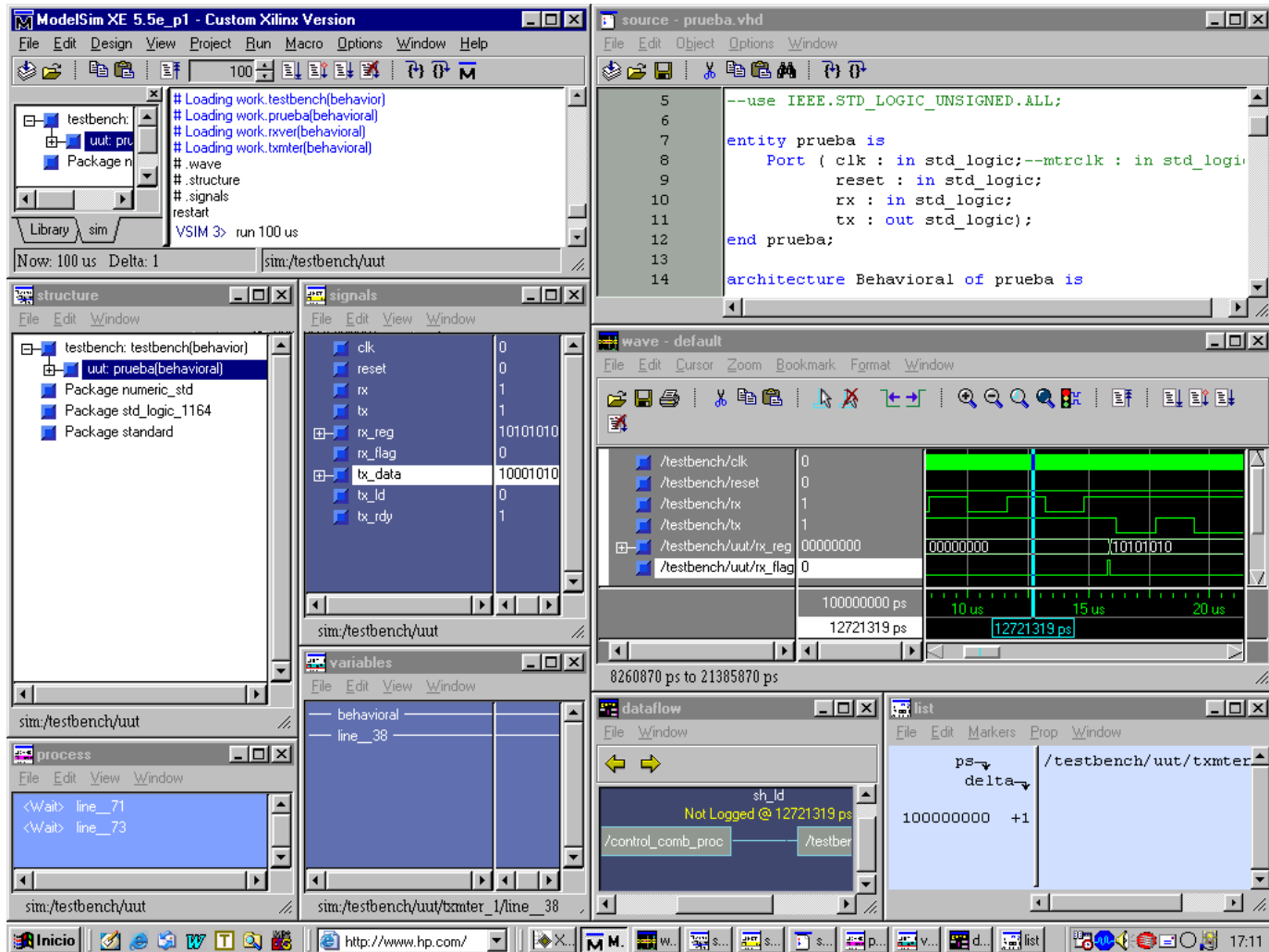
Property Name	Value
Place & Route Effort Level (Overall)	Default (Low) ▼
Placer Effort Level (Overrides Overall Level)	Default (None)
Router Effort Level (Overrides Overall Level)	Default (None)
Extra Effort (Highest PAR level only)	N/A
Starting Placer Cost Table (1-100)	1
Place And Route Mode	Normal Place and Route
Guide File	
Guide Mode	N/A
Use Timing Constraints	<input checked="" type="checkbox"/>
Use Bonded I/Os	<input type="checkbox"/>
Generate Detailed PAR Report	<input type="checkbox"/>
Generate Post-Place & Route Static Timing Report	<input checked="" type="checkbox"/>
Generate Post-Place & Route Simulation Model	<input type="checkbox"/>

At the bottom of the dialog, there are four buttons: 'Aceptar', 'Cancelar', 'Default', and 'Ayuda'.

Procesos de Simulación



Simulación con ModelSim 5.7





Tema 4:

Descripción RTL de componentes combinacionales

Titulación: Ingeniero de Telecomunicación
Asignatura: Arquitectura y Tecnología de Computadores



Desarrollo de contenidos

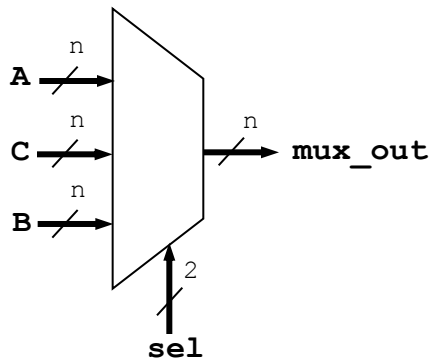
Descripción RTL de componentes combinacionales

Sumario

- Introducción
- Multiplexores
- Codificadores/decodificadores
- Codificadores con prioridad
- Comparadores
- *Buffers* triestado
- Desplazadores y rotadores
- Unidades Aritmético-Lógicas

Multiplexores

■ Ejemplo 4.1: Multiplexor de 3 entradas de tamaño arbitrario



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY multiplexor IS
    GENERIC(n : positive := 8);
    PORT (sel : IN std_logic_vector(1 DOWNTO 0);
          A   : IN std_logic_vector(n-1 DOWNTO 0);
          C   : IN std_logic_vector(n-1 DOWNTO 0);
          B   : IN std_logic_vector(n-1 DOWNTO 0);
          mux_out: OUT std_logic_vector(n-1 DOWNTO 0));
END multiplexor;

ARCHITECTURE Behavioral OF multiplexor IS

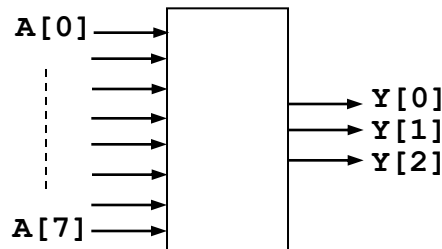
BEGIN

    PROCESS (sel, A, B, C)
    BEGIN
        CASE sel IS
            WHEN "00" =>
                MUX_OUT <= A;
            WHEN "01" =>
                MUX_OUT <= B;
            WHEN others =>
                MUX_OUT <= C;
        END CASE;
    END PROCESS;

END Behavioral;
```

Codificadores/decodificadores

■ Ejemplo 4.2: Codificador de 8 a 3



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY cod_8_3 IS
    PORT ( A : IN std_logic_vector(7 DOWNTO 0);
          Y : OUT std_logic_vector(2 DOWNTO 0));
END cod_8_3;

ARCHITECTURE Behavioral OF cod_8_3 IS

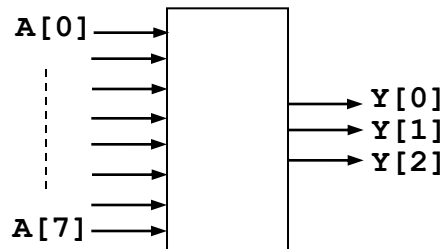
BEGIN

    PROCESS (A)
    BEGIN
        CASE A IS
            WHEN "00000001" => Y <= "000";
            WHEN "00000010" => Y <= "001";
            WHEN "00000100" => Y <= "010";
            WHEN "00001000" => Y <= "011";
            WHEN "00010000" => Y <= "100";
            WHEN "00100000" => Y <= "101";
            WHEN "01000000" => Y <= "110";
            WHEN OTHERS      => Y <= "111";
        END CASE;
    END PROCESS;

END Behavioral;
```

Codificadores/decodificadores

■ Ejemplo 4.4: Codificador de 8 a 3 con indiferencias



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY cod_8_3 IS
    PORT ( A : IN std_logic_vector(7 DOWNT0 0);
          Y : OUT std_logic_vector(2 DOWNT0 0));
END cod_8_3;

ARCHITECTURE Behavioral OF cod_8_3 IS

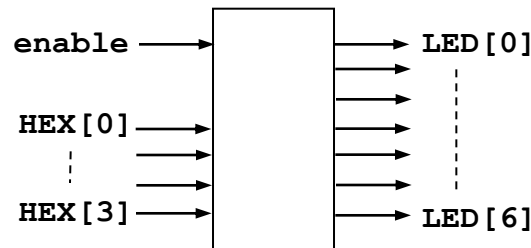
BEGIN

    PROCESS(A)
    BEGIN
        CASE A IS
            WHEN "00000001" => Y <= "000";
            WHEN "00000010" => Y <= "001";
            WHEN "00000100" => Y <= "010";
            WHEN "00001000" => Y <= "011";
            WHEN "00010000" => Y <= "100";
            WHEN "00100000" => Y <= "101";
            WHEN "01000000" => Y <= "110";
            WHEN "10000000" => Y <= "111";
            WHEN OTHERS      => Y <= "---";
        END CASE;
    END PROCESS;

END Behavioral;
```

Codificadores/decodificadores

- **Ejemplo 4.3:**
Convertidor de
código hexadecimal
a 7 segmentos



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY hex2led IS
    PORT ( enable : IN std_logic;
          HEX     : IN std_logic_vector (3 DOWNT0 0);
          LED     : OUT std_logic_vector (6 DOWNT0 0));
END hex2led;

ARCHITECTURE Behavioral OF hex2led IS

BEGIN

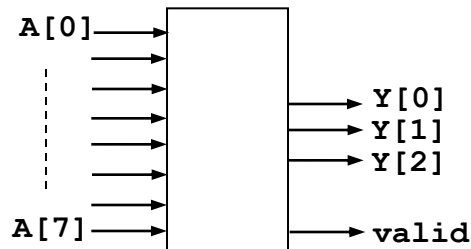
PROCESS (enable, HEX)
BEGIN
    IF (enable = '1') THEN
        LED <= "1111111"; -- apaga los 7 segmentos
    ELSE
        CASE HEX IS --nº segmento:6543210
            WHEN "0001" => LED <= "1111001"; --1
            WHEN "0010" => LED <= "0100100"; --2
            WHEN "0011" => LED <= "0110000"; --3
            WHEN "0100" => LED <= "0011001"; --4
            WHEN "0101" => LED <= "0010010"; --5
            WHEN "0110" => LED <= "0000010"; --6
            WHEN "0111" => LED <= "1111000"; --7
            WHEN "1000" => LED <= "0000000"; --8
            WHEN "1001" => LED <= "0010000"; --9
            WHEN "1010" => LED <= "0001000"; --A
            WHEN "1011" => LED <= "0000011"; --B
            WHEN "1100" => LED <= "1000110"; --C
            WHEN "1101" => LED <= "0100001"; --D
            WHEN "1110" => LED <= "0000110"; --E
            WHEN "1111" => LED <= "0001110"; --F
            WHEN OTHERS => LED <= "1000000"; --0
        END CASE;
    END IF;
END PROCESS;

END BEHAVIORAL;
```

```
--      0
--      ---
-- 5 |   | 1
-- --- <- 6
-- 4 |   | 2
-- ---
--      3
```


Codificadores con prioridad

■ Ejemplo 4.5: Codificador con prioridad de 8 a 3



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY codific_prior IS
    PORT ( A      : IN std_logic_vector(7 DOWNTO 0);
          valid   : OUT std_logic;
          Y       : OUT std_logic_vector(2 DOWNTO 0));
END codific_prior;

ARCHITECTURE Behavioral OF codific_prior IS

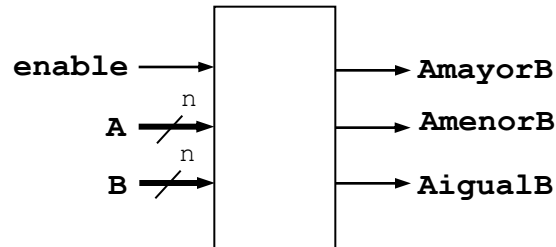
BEGIN

    PROCESS (A)
    BEGIN
        valid <= '1';
        IF (A(7)='1') THEN
            Y <= "111";
        ELSIF (A(6)='1') THEN
            Y <= "110";
        ELSIF (A(5)='1') THEN
            Y <= "101";
        ELSIF (A(4)='1') THEN
            Y <= "100";
        ELSIF (A(3)='1') THEN
            Y <= "011";
        ELSIF (A(2)='1') THEN
            Y <= "010";
        ELSIF (A(1)='1') THEN
            Y <= "001";
        ELSIF (A(0)='1') THEN
            Y <= "000";
        ELSE
            Y <= "---";
            valid <= '0';
        END IF;
    END PROCESS;

END Behavioral;
```

Comparadores

■ Ejemplo 4.6: Comparador de n bits con entrada de habilitación



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY comparador IS
    GENERIC(n          : positive := 8);
    PORT ( enable      : IN std_logic;
          A           : IN std_logic_vector(n-1 DOWNT0 0);
          B           : IN std_logic_vector(n-1 DOWNT0 0);
          AmenorB      : OUT std_logic;
          AmayorB       : OUT std_logic;
          AigualB       : OUT std_logic);
END comparador;

ARCHITECTURE Behavioral OF comparador IS

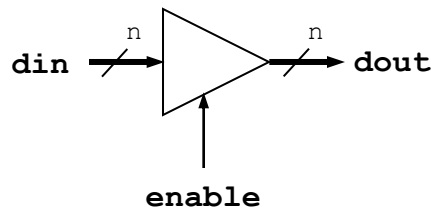
BEGIN

    PROCESS (enable, A, B)
    BEGIN
        IF (enable = '0') THEN
            AmenorB <= '0'; AmayorB <= '0'; AigualB <= '0';
        ELSE
            IF ( A < B ) THEN
                AmenorB <= '1';
            ELSE
                AmenorB <= '0';
            END IF;
            IF ( A > B ) THEN
                AmayorB <= '1';
            ELSE
                AmayorB <= '0';
            END IF;
            IF ( A = B ) THEN
                AigualB <= '1';
            ELSE
                AigualB <= '0';
            END IF;
        END IF;
    END PROCESS;

END Behavioral;
```

Buffers triestado

■ Ejemplo 4.7: *Buffer* triestado de tamaño arbitrario



```
-- buffer triestado de tamaño arbitrario
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY triestado IS
    GENERIC(n : positive := 8);
    PORT(enable: IN std_logic;
          din   : IN std_logic_vector(n-1 DOWNT0 0);
          dout  : OUT std_logic_vector(n-1 DOWNT0 0));
END triestado;

ARCHITECTURE Behavioral OF triestado IS

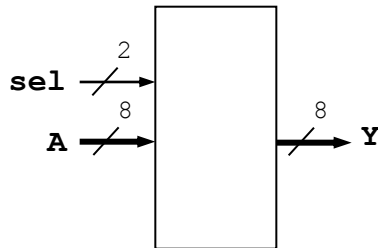
BEGIN

    PROCESS (enable, din)
    BEGIN
        IF (enable='1') THEN
            dout <= din;
        ELSE
            dout <= (OTHERS => 'Z');
        END IF;
    END PROCESS;

END Behavioral;
```

Desplazadores y rotadores

■ Ejemplo 4.9: Desplazador lógico a la izquierda



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY desplazador IS
    PORT ( A      : IN std_logic_vector(7 DOWNTO 0);
          sel     : IN std_logic_vector(1 DOWNTO 0);
          Y       : OUT std_logic_vector(7 DOWNTO 0));
END desplazador;

ARCHITECTURE Behavioral OF desplazador IS

BEGIN

    PROCESS (A, sel)
    BEGIN
        CASE sel IS
            WHEN "00" =>
                Y <= A;
            WHEN "01" =>
                Y(7 DOWNTO 1) <= A(6 DOWNTO 0);
                Y(0) <= '0';
            WHEN "10" =>
                Y(7 DOWNTO 2) <= A(5 DOWNTO 0);
                Y(1 DOWNTO 0) <= (OTHERS=>'0');
            WHEN OTHERS =>
                Y(7 DOWNTO 3) <= A(4 DOWNTO 0);
                Y(2 DOWNTO 0) <= (OTHERS=>'0');
        END CASE;
    END PROCESS;

END Behavioral;
```

Desplazadores y rotadores

■ Ejemplo 4.9: Desplazador lógico a la izquierda

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY desplazador IS
    PORT ( A      : IN std_logic_vector(7 DOWNTO 0);
          sel     : IN std_logic_vector(1 DOWNTO 0);
          Y       : OUT std_logic_vector(7 DOWNTO 0));
END desplazador;

ARCHITECTURE Behavioral OF desplazador IS

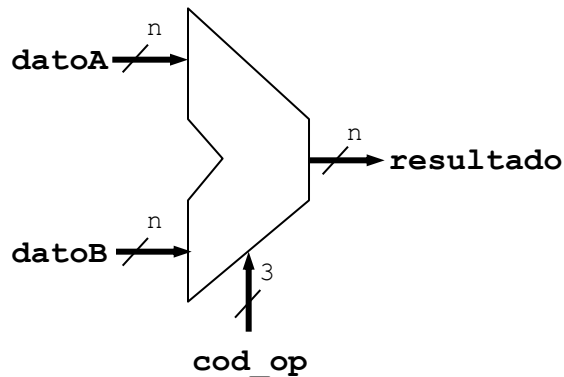
BEGIN

    PROCESS (A, sel)
    BEGIN
        CASE sel IS
```

Operación	Sintaxis 1	Sintaxis 2*
Desplazam. Lógico a la Izqda.	<code>Y <= A(5 DOWNTO 0) & "00";</code>	<code>Y <= A sll 2</code>
Desplazam. Lógico a la Drcha.	<code>Y <= "00" & A(7 DOWNTO 2);</code>	<code>Y <= A srl 2</code>
Desplazam. Aritmético a la Izqda.	<code>Y <= A(5 DOWNTO 0) & A(0) & A(0);</code>	<code>Y <= A sla 2</code>
Desplazam. Aritmético a la Drcha.	<code>Y <= A(7) & A(7) & A(7 DOWNTO 2);</code>	<code>Y <= A sra 2</code>
Rotar a la Izada.	<code>Y <= A(5 DOWNTO 0) & A(7 DOWNTO 6);</code>	<code>Y <= A rol 2</code>
Rotar a la Drcha.	<code>Y <= A(1 DOWNTO 0) & A(7 DOWNTO 2);</code>	<code>Y <= A sor 2</code>
*En realidad, esta sintaxis sólo permite operar sobre tipos de datos definidos como <i>bit_vector</i> , y no sobre <i>std_logic_vector</i> , como veremos más adelante. No obstante, la biblioteca <i>numeric_std</i> sobrecarga estos operadores para que también puedan utilizarse con sus tipos de datos, <i>signed</i> y <i>unsigned</i> .		

Unidades aritmético-lógicas

■ Ejemplo 4.10: Unidad aritmético-lógica trivial



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;

ENTITY alu IS
    GENERIC ( n      : positive := 8 );
    PORT ( cod_op    : IN std_logic_vector(2 DOWNTO 0);
          datoA      : IN std_logic_vector(n-1 DOWNTO 0);
          datoB      : IN std_logic_vector(n-1 DOWNTO 0);
          resultado  : OUT std_logic_vector(n-1 DOWNTO 0));
END alu;

ARCHITECTURE Behavioral OF alu IS
BEGIN

    PROCESS (cod_op, datoA, datoB)
    BEGIN
        CASE cod_op IS
            --operaciones logicas
            WHEN "000" => resultado <= datoA;
            WHEN "001" => resultado <= datoA AND datoB;
            WHEN "010" => resultado <= datoA OR datoB;
            WHEN "011" => resultado <= datoA XOR datoB;
            --operaciones aritmeticas
            WHEN "100" => resultado <= std_logic_vector(signed(datoA) + 1);
            WHEN "101" => resultado <= std_logic_vector(signed(datoA) + signed(datoB));
            WHEN "110" => resultado <= std_logic_vector(signed(datoA) - signed(datoB));
            WHEN OTHERS => resultado <= std_logic_vector(signed(datoA) - 1);

        END CASE;
    END PROCESS;

END Behavioral;
```



Tema 9:

VHDL para simulación

Titulación: Ingeniero de Telecomunicación
Asignatura: Arquitectura y Tecnología de Computadores



Desarrollo de contenidos

VHDL para simulación

Sumario

- Introducción
- Tipos de simulaciones
- Tipos de simuladores
- Diseño de bancos de prueba (*testbenches*)

Tipos de simulaciones

1. Simulación funcional

- Simulación pre-síntesis. Sirve para verificar que la sintaxis y la funcionalidad del modelo VHDL introducido es la esperada. Retardos de propagación nulos.
- En Xilinx → *Behavioral Simulation*

2. Simulación con retardos unitarios

- Simulación post-síntesis. Contiene el mapeado en primitivas (CLBs, IOBs) del sistema.
- En Xilinx → *Post-Translate Simulation* (retardos unitarios)
- En Xilinx → *Post-Map Simulation* (incluye retardos en las primitivas: *partial backannotated timing*)

3. Simulación temporal

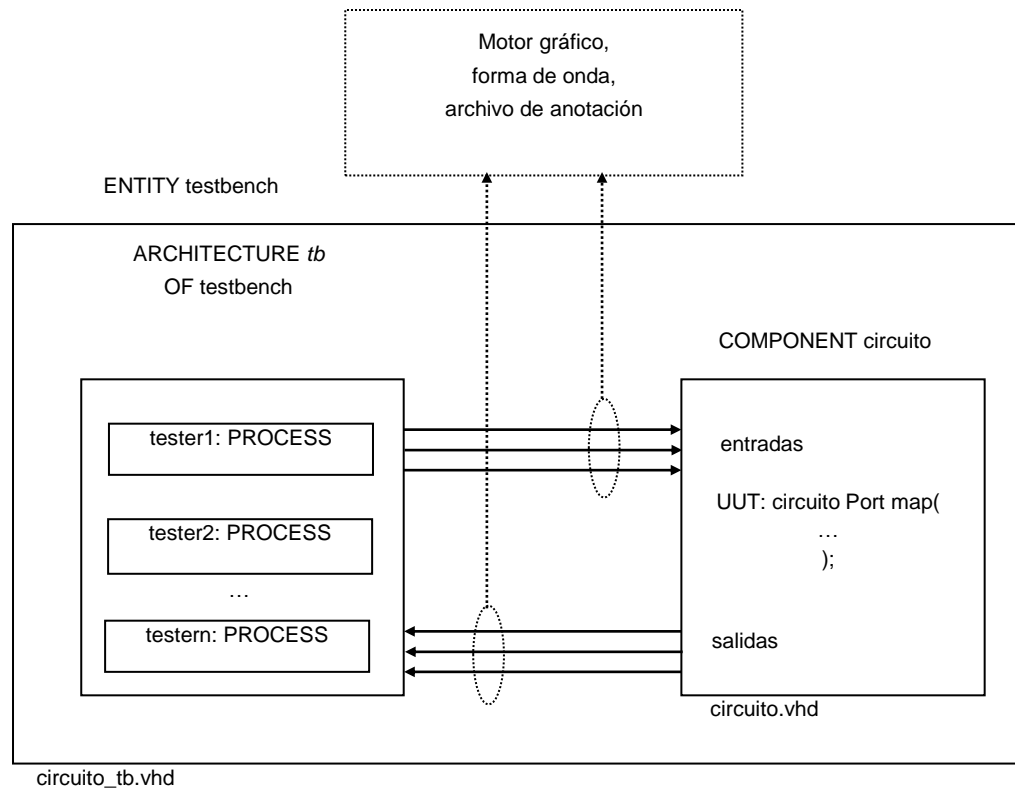
- Simulación post-place&route. Contiene el mapeado en primitivas con retardos en ellas y en el rutado.
- En Xilinx → *Post-Place&route Simulation*

Tipos de simuladores

- **Simuladores basados en ciclos**
 - Se evalúan las entradas de los biestables cuando se activa la señal de reloj.
 - Se ignoran los retardos de propagación: no se simula la propagación de los valores a través de la lógica combinacional.
- **Simuladores dirigidos por eventos**
 - Solo evalúan la lógica cuando se produce un cambio en sus entradas, y consideran únicamente la lógica afectada por la propagación de los eventos.
 - Tipos: de código compilado (rápido y caro) o de código interpretado (más barato y lento).
- Los simuladores dirigidos por eventos son más exactos y (actualmente) más rápidos que los basados en ciclos.

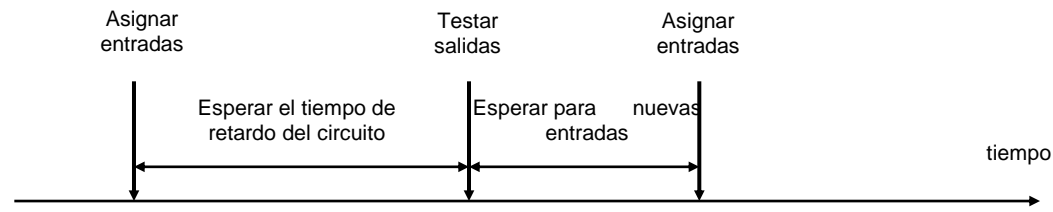
Diseño del banco de pruebas (*testbench*)

- Arquitectura del *testbench*:

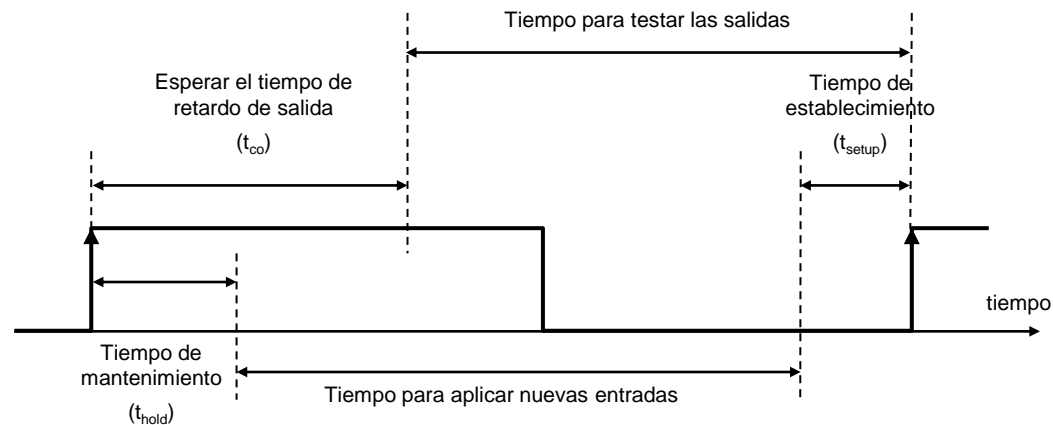


Diseño del banco de pruebas (*testbench*)

- Temporización de la simulación:



a) Sistemas combinacionales



b) Sistemas secuenciales

Diseño del banco de pruebas (*testbench*)

■ Ejemplo 9.1: Comparador de magnitud: Técnicas básicas

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

    COMPONENT comparador
        PORT (enable : IN std_logic;
              A : IN std_logic_vector(7 downto 0);
              B : IN std_logic_vector(7 downto 0);
              AmenorB : OUT std_logic;
              AmayorB : OUT std_logic;
              AigualB : OUT std_logic );
    END COMPONENT;

    SIGNAL enable : std_logic;
    SIGNAL A : std_logic_vector(7 downto 0);
    SIGNAL B : std_logic_vector(7 downto 0);
    SIGNAL AmenorB : std_logic;
    SIGNAL AmayorB : std_logic;
    SIGNAL AigualB : std_logic;

    ...
```

```
...
BEGIN

    uut: comparador PORT MAP (enable => enable,
                              A => A,
                              B => B,
                              AmenorB => AmenorB,
                              AmayorB => AmayorB,
                              AigualB => AigualB );

    -- proceso para el test:
    tb : PROCESS
        BEGIN
            enable <= '0';
            A <= "10000000";
            B <= "10000001";
            wait for 100 ns;
            enable <= '1';
            wait for 100 ns;
            A <= "11000000";
            B <= "11000001";
            wait for 100 ns;
            A <= "10000010";
            B <= "10000001";
            wait for 100 ns;
            enable <= '0';
            wait for 100 ns;
            A <= "00001111";
            B <= "01011100";
            wait for 100 ns;
            enable <= '1';
            A <= "01000000";
            B <= "01000000";
            wait for 100 ns;
            A <= "11000000";
            B <= "01000000";
            wait; -- espera indefinidamente
        END PROCESS;

    END;
```

Diseño del banco de pruebas (*testbench*)

■ Ejemplo 9.2: Comparador de magnitud: Verificación de valores

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

    COMPONENT comparador
        PORT (enable : IN std_logic;
              A : IN std_logic_vector(7 downto 0);
              B : IN std_logic_vector(7 downto 0);
              AmenorB : OUT std_logic;
              AmayorB : OUT std_logic;
              AigualB : OUT std_logic);
    END COMPONENT;

    SIGNAL enable : std_logic := '1';
    SIGNAL A : std_logic_vector(7 downto 0);
    SIGNAL B : std_logic_vector(7 downto 0);
    SIGNAL AmenorB : std_logic;
    SIGNAL AmayorB : std_logic;
    SIGNAL AigualB : std_logic;
    ...
```

```
...
BEGIN

    uut: comparador PORT MAP (enable => enable,
                              A => A,
                              B => B,
                              AmenorB => AmenorB,
                              AmayorB => AmayorB,
                              AigualB => AigualB );

    --asignacion de valores basada en tiempo absoluto
    enable <= '0','1' after 100 ns,'0' after 400 ns,'1' after 600 ns;

    -- proceso para test (asignacion basada en tiempo relativo):
    tb : PROCESS
    BEGIN
        --enable <='0'; --ya no es necesario, especificado aparte
        A <= "10000000";
        B <= "10000001";
        wait for 100 ns;
        assert (AmenorB='0' and AmayorB='0' and AigualB='0')
        report "Error a los 100 ns."
        severity error;

        --enable <= '1';
        wait for 100 ns;
        assert (AmenorB='1' and AmayorB='0' and AigualB='0')
        report "Error a los 200 ns."
        severity error;

        A <= "11000000";
        B <= "11000001";
        wait for 100 ns;
        assert (AmenorB='1' and AmayorB='0' and AigualB='0')
        report "Error a los 300 ns."
        severity error;

        ...
    END;
```

Diseño del banco de pruebas (*testbench*)

■ Ejemplo 9.3: Contador: descripción de señales de reloj

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS
    COMPONENT contador
        PORT (clk : IN std_logic;
              reset : IN std_logic;
              ce : IN std_logic;
              load : IN std_logic;
              dir : IN std_logic;
              din : IN std_logic_vector(7 downto 0);
              count : OUT std_logic_vector(7 downto 0) );
    END COMPONENT;

    SIGNAL clk : std_logic := '0';
    SIGNAL reset : std_logic := '1';
    SIGNAL ce : std_logic;
    SIGNAL load : std_logic;
    SIGNAL dir : std_logic;
    SIGNAL din : std_logic_vector(7 downto 0);
    SIGNAL count : std_logic_vector(7 downto 0);
    CONSTANT periodo: time := 100 ns;
    ...
```

```
...
BEGIN
```

```
    uut: contador PORT MAP(clk => clk,
                           reset => reset,
                           ce => ce,
                           load => load,
                           dir => dir,
                           din => din,
                           count => count );
```

```
    clk <= not clk after periodo/2; --reloj periodico
```

```
    tb : PROCESS
    BEGIN
```

```
        din <= X"07";           -- Asignamos valores iniciales,
        dir <= '1';             -- pero la salida se mantiene a 0
        ce <= '0';              -- porque reset esta activo al inicio.
        load <= '0';
        wait for 2*periodo;      -- Esperamos 2 period con reset activo.
        reset <= '0';           -- Desactivamos el reset y
        ce <= '1';              -- habilitamos la cuenta.
        wait for 15*periodo;     -- Contamos hasta 15.
        load <= '1';            -- Habilitamos la carga, lo que
        wait for periodo;       -- carga el valor inicial 7.
        load <= '0';            -- Deshabilitamos la carga.
        wait for 7*periodo;      -- Contamos desde 7 hasta 14.
        reset <= '1';           -- Reset a 0
        wait for 2*periodo;      -- durante 2 periodos.
        reset <= '0';           -- Desactivamos el reset y
        wait for 7*periodo;      -- contamos hasta 7.
        ce <= '0';              -- Mantenemos la cuenta en 7
        wait for 4*periodo;      -- durante 4 periodos.
        ce <= '1';              -- Habilitamos la cuenta
        wait for 7*periodo;      -- y contamos hasta 13.
        reset <= '1';           -- Reset a 0 del contador.
        wait; -- espera indefinidamente
```

```
    END PROCESS;
```

```
END;
```


Diseño del banco de pruebas (*testbench*)

■ Ejemplo 9.6: hex2disp: generación de otros estímulos repetitivos

```
--relojes asimetricos, duracion limitada
clk_gen: PROCESS
BEGIN
    FOR n IN 1 TO 5000 LOOP
        clk <='0';
        WAIT FOR 70 ns;
        clk <='1';
        WAIT FOR 30 ns;
    END LOOP;
END PROCESS;
```

```
--relojes asimetricos, duracion ilimitada
clk_gen: PROCESS
BEGIN
    WHILE true LOOP
        clk <='0';
        WAIT FOR 70 ns;
        clk <='1';
        WAIT FOR 30 ns;
    END LOOP;
END PROCESS;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

    COMPONENT hex2led
    PORT (HEX : IN std_logic_vector(3 downto 0);
          enable : IN std_logic;
          LED : OUT std_logic_vector(6 downto 0) );
    END COMPONENT;

    SIGNAL HEX : std_logic_vector(3 downto 0);
    SIGNAL enable : std_logic;
    SIGNAL LED : std_logic_vector(6 downto 0);

BEGIN

    uut: hex2led PORT MAP(HEX => HEX,
                          enable => enable,
                          LED => LED);

    enable <= '1', '0' after 100 ns; -- generamos pulso de habilitac

    -- proceso principal de test
    tb : PROCESS
    BEGIN
        wait for 100 ns; --esperamos que el circuito este habilitado
        for i in 0 to 15 loop
            HEX <= std_logic_vector(to_unsigned(i,4));
            wait for 100 ns;
        end loop;
        wait; -- espera indefinidamente
    END PROCESS;

END;
```



Tema 5:

Descripción RTL de componentes secuenciales

Titulación: Ingeniero de Telecomunicación
Asignatura: Arquitectura y Tecnología de Computadores



Desarrollo de contenidos

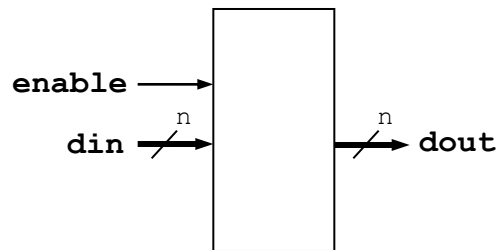
Descripción RTL de componentes secuenciales

Sumario

- Introducción
- *Latches*
- Flip-flops
- Registros de desplazamiento
- Contadores

Latches

- **Ejemplo 5.1:**
Descripción de
latches tipo D
sincronizados
(*gated D latch*)



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY milatch IS
    GENERIC( n      : positive := 8);
    PORT (enable : IN std_logic;
          din    : IN std_logic_vector(n-1 DOWNTO 0);
          dout   : OUT std_logic_vector(n-1 DOWNTO 0));
END milatch;

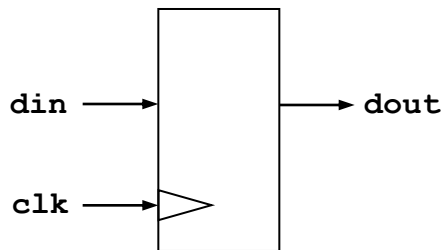
ARCHITECTURE Behavioral OF milatch IS
BEGIN

    PROCESS (enable, din)
    BEGIN
        IF (enable = '1') THEN
            dout <= din;
        END IF;
    END PROCESS;

END Behavioral;
```

Flip-flops

■ Ejemplo 5.2: Descripción de un *flip-flop* tipo D



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY biestableD IS
    PORT ( clk : IN std_logic;
          din : IN std_logic;
          dout : OUT std_logic);
END biestableD;

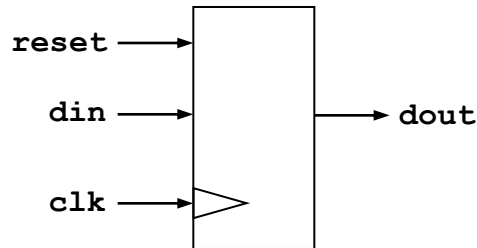
ARCHITECTURE Behavioral OF biestableD IS
BEGIN

    PROCESS (clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            dout <= din;
        END IF;
    END PROCESS;

END Behavioral;
```

Flip-flops

- **Ejemplo 5.3:**
Descripción de un *flip-flop* tipo D con *reset* síncrono



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY biestabled IS
    PORT ( clk   : IN std_logic;
          reset : IN std_logic;
          din    : IN std_logic;
          dout   : OUT std_logic);
END biestabled;

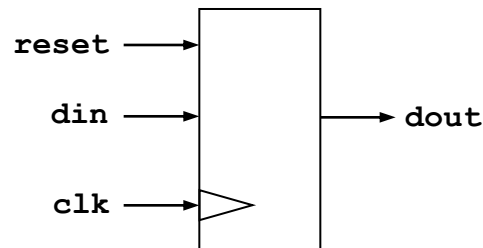
ARCHITECTURE Behavioral OF biestabled IS
BEGIN

    PROCESS (clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            IF (reset = '1') THEN
                dout <= '0';
            ELSE
                dout <= din;
            END IF;
        END IF;
    END PROCESS;

END Behavioral;
```

Flip-flops

- **Ejemplo 5.4:**
Descripción de un *flip-flop* tipo D con *reset* asíncrono



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY biestableD IS
    PORT ( clk : IN std_logic;
          reset: IN std_logic;
          din  : IN std_logic;
          dout : OUT std_logic);
END biestableD;

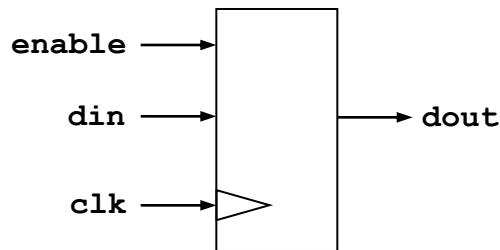
ARCHITECTURE Behavioral OF biestableD IS
BEGIN

    PROCESS (clk, reset)
    BEGIN
        IF (reset = '1') THEN
            dout <= '0';
        ELSIF (clk'EVENT AND clk = '1') THEN
            dout <= din;
        END IF;
    END PROCESS;

END Behavioral;
```


Flip-flops

- **Ejemplo 5.5:**
Descripción de un *flip-flop* tipo D con habilitación de reloj



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY biestableD IS
    PORT ( clk      : IN std_logic;
          enable    : IN std_logic;
          din       : IN std_logic;
          dout      : OUT std_logic);
END biestableD;

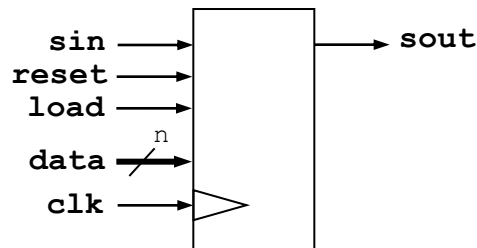
ARCHITECTURE Behavioral OF biestableD IS
BEGIN

    PROCESS (clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            IF (enable = '1') THEN
                dout <= din;
            END IF;
        END IF;
    END PROCESS;

END Behavioral;
```

Registros de desplazamiento

- **Ejemplo 5.8:**
Registro de desplazamiento con reset asíncrono, carga síncrona, y entrada/salida serie



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY reg_desplaza IS
    GENERIC (n : positive := 8);
    PORT ( clk : IN std_logic;
          reset : IN std_logic;
          load : IN std_logic;
          data : IN std_logic_vector(n-1 DOWNT0 0);
          sin : IN std_logic;
          sout : OUT std_logic);
END reg_desplaza;

ARCHITECTURE Behavioral OF reg_desplaza IS
    SIGNAL reg: std_logic_vector (n-1 DOWNT0 0);
BEGIN

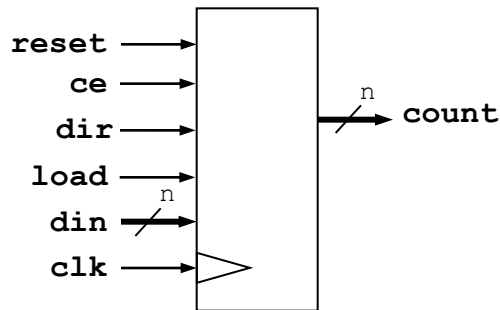
    PROCESS (reset, clk)
    BEGIN
        IF (reset='1') THEN
            reg <= (OTHERS=>'0');
        ELSIF (clk'EVENT AND clk='1') THEN
            IF (load='1') THEN
                reg <= data;
            ELSE
                reg <= reg(n-2 DOWNT0 0) & sin; --desplaza a izquierdas
            END IF;
        END IF;
    END PROCESS;

    sout <= reg(n-1);

END Behavioral;
```

Contadores

- **Ejemplo 5.9:**
Contador síncrono,
reset asíncrono,
entradas de carga y
habilitación síncronas



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY contador IS
    GENERIC( n          : positive := 8);
    PORT (   clk, reset, ce, load, dir : IN std_logic;
            din   : IN std_logic_vector(n-1 DOWNTO 0);
            count  : OUT std_logic_vector(n-1 DOWNTO 0));
END contador;

ARCHITECTURE Behavioral OF contador IS
    SIGNAL count_aux: unsigned (n-1 DOWNTO 0);
BEGIN

    PROCESS (clk, reset)
    BEGIN
        IF reset='1' THEN
            count_aux <= (OTHERS => '0');
        ELSIF clk='1' AND clk'EVENT THEN
            IF load='1' THEN
                count_aux <= unsigned(din);
            ELSE
                IF ce='1' THEN
                    IF dir='1' THEN
                        count_aux <= count_aux + 1;
                    ELSE
                        count_aux <= count_aux - 1;
                    END IF;
                END IF;
            END IF;
        END IF;
    END PROCESS;

    count <= std_logic_vector(count_aux);
END Behavioral;
```



Tema 8:

Síntesis de alto nivel

Titulación: Ingeniero de Telecomunicación
Asignatura: Arquitectura y Tecnología de Computadores



Desarrollo de contenidos

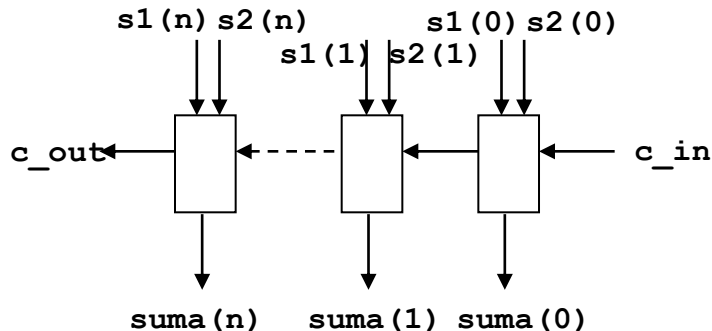
Síntesis de alto nivel

Sumario

- Introducción
- Reutilización del código: sentencia GENERATE
- Bibliotecas de uso común: numeric_std
- Inferencia de memoria ROM
- Inferencia de memoria RAM
- Inferencia de Máquinas de Estado

Sentencia GENERATE

- **Ejemplo 6.11:**
Sumador genérico de
n bits.



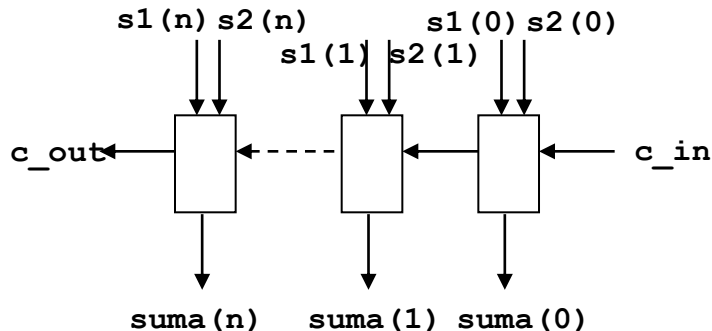
```
-- Sumador completo de 1 bit
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY sum_completo IS
    PORT (dato1 : IN STD_LOGIC;      -- bit 1er sumando
          dato2 : IN STD_LOGIC;      -- bit 2º sumando
          carry_in : IN STD_LOGIC;    -- acarreo de entrada
          suma : OUT STD_LOGIC;       -- salida sumada
          carry_out : OUT STD_LOGIC   -- salida de acarreo
    );
END sum_completo;

ARCHITECTURE sum_completo_arch OF sum_completo IS
BEGIN
    suma <= dato1 XOR dato2 XOR carry_in;
    carry_out <= (dato1 AND dato2)
                OR (dato1 AND carry_in)
                OR (dato2 AND carry_in);
END sum_completo_arch;
```

Sentencia GENERATE

■ Ejemplo 6.11: Sumador genérico de n bits.



```
-- Sumador de n-bits, a partir de un sumador completo de 1 bit.
-- GENERIC para definir tamaño sumador, luego ahora es obligatorio
-- GENERATE para automatizar instanciación del array de sumador
ENTITY sumador IS
    GENERIC (size : POSITIVE := 8);
    PORT
        (sum1 : IN STD_LOGIC_VECTOR (size-1 DOWNT0 0); -- 1er sumando
         sum2 : IN STD_LOGIC_VECTOR (size-1 DOWNT0 0); -- 2º sumando
         c_in : IN STD_LOGIC; -- acarreo de entrada
         suma : OUT STD_LOGIC_VECTOR (size-1 DOWNT0 0); -- salida suma
         c_out : OUT STD_LOGIC -- acarreo de salida);
END sumador;

ARCHITECTURE sumador_arch OF sumador IS
    COMPONENT sum_completo IS
        PORT (dato1 : IN STD_LOGIC; -- bit primer sumando
              dato2 : IN STD_LOGIC; -- bit segundo sumando
              carry_in : IN STD_LOGIC; -- bit de acarreo de entrada
              suma : OUT STD_LOGIC; -- bit de la salida sumada
              carry_out : OUT STD_LOGIC -- bit de acarreo de salida
              );
    END COMPONENT;

    -- señal c necesaria para trasladar los acarreos entre sumadores
    SIGNAL c : STD_LOGIC_VECTOR (size DOWNT0 0);

BEGIN

    c(0) <= c_in;

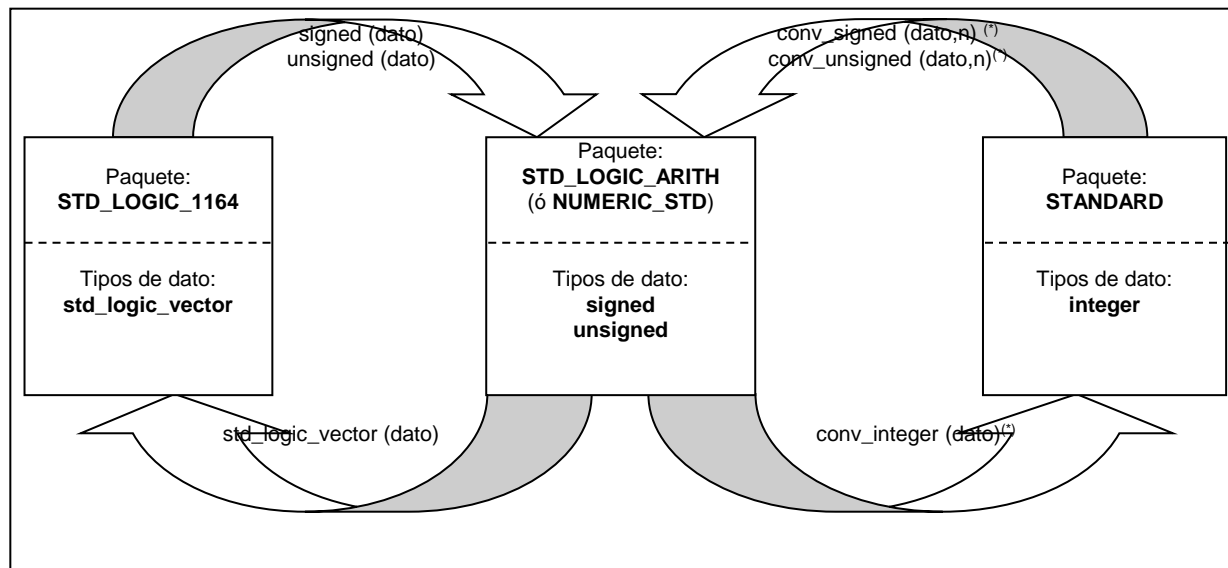
    gen_sumador: FOR i IN 0 TO size-1 GENERATE -- generamos size-1 sumadores
        sum_array: sum_completo PORT MAP
            (dato1 => sum1(i),
             dato2 => sum2(i),
             carry_in => c(i),
             suma => suma(i),
             carry_out => c(i+1)); -- asociación explícita de puertos
    END GENERATE;

    c_out <= c(size);

END sumador_arch;
```

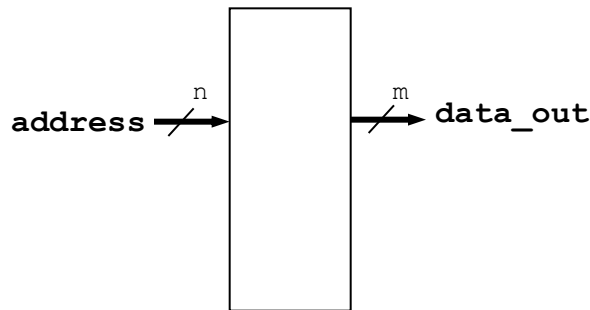

Biblioteca numeric_std

- Resumen de la conversión entre tipos de datos numéricos y vectores de bits:



Memoria ROM

■ Ejemplo 8.1: ROM de lectura asíncrona



```
--Infiere memoria ROM de lectura asincrona
--(por ello en Xilinx se implementara, por defecto, como memoria
distribuida)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity rom_asinc is
    Generic (data_width : integer := 8;
            address_width : integer := 5;
            mem_depth : integer := 32);
    Port(address : in STD_LOGIC_VECTOR(address_width-1 downto 0);
          data_out : out STD_LOGIC_VECTOR(data_width-1 downto 0) );
end rom_asinc;

architecture Behavioral of rom_asinc is
    type rom_type is array (mem_depth-1 downto 0) of
        STD_LOGIC_VECTOR (data_width-1 downto 0);
    constant rom : rom_type :=(
        "10001010", "11110000", "10101010", "00001111", -- 3
        "01010101", "00000000", "11111111", "11001100", -- 7
        "01010101", "00000000", "11111111", "11001100", -- 11
        "01010101", "00000000", "11111111", "11001100", -- 15
        "10001010", "11110000", "10101010", "00001111", -- 19
        "01010101", "00000000", "11111111", "11001100", -- 23
        "01010101", "00000000", "11111111", "11001100", -- 27
        "01010101", "00000000", "11111111", "11001100" ); --31

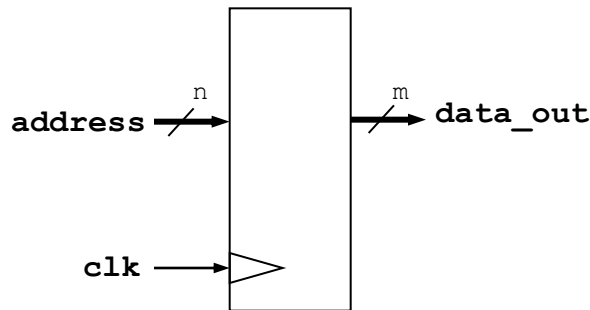
    begin

        data_out <= rom(to_integer(unsigned(address)));

    end Behavioral;
```

Memoria ROM

■ Ejemplo 8.2: ROM de lectura síncrona



```
--Infiere memoria ROM de lectura síncrona
--(por ello en Xilinx se implementara, por defecto, como block
RAM)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity rom_sync is
    Generic (data_width  : integer := 8;
            address_width : integer := 5;
            mem_depth     : integer := 32);
    Port(address: in STD_LOGIC_VECTOR(address_width-1 downto 0);
          clk    : in STD_LOGIC;
          data_out : out STD_LOGIC_VECTOR(data_width-1 downto 0) );
end rom_sync;

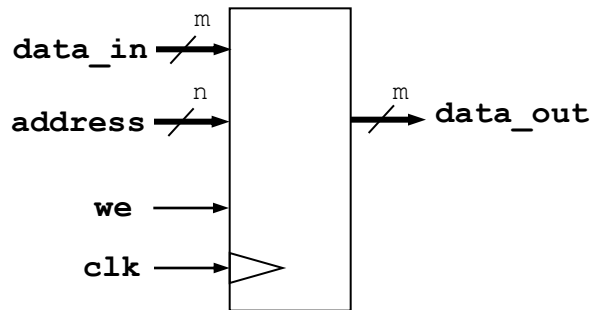
architecture Behavioral of rom_sync is
    type rom_type is array (mem_depth-1 downto 0) of
        STD_LOGIC_VECTOR (data_width-1 downto 0);
    constant rom : rom_type :=(
        "10001010", "11110000", "10101010", "00001111", -- 3
        "01010101", "00000000", "11111111", "11001100", -- 7
        "01010101", "00000000", "11111111", "11001100", -- 11
        "01010101", "00000000", "11111111", "11001100", -- 15
        "10001010", "11110000", "10101010", "00001111", -- 19
        "01010101", "00000000", "11111111", "11001100", -- 23
        "01010101", "00000000", "11111111", "11001100", -- 27
        "01010101", "00000000", "11111111", "11001100" ); --31
    begin

        rd: process(clk)
        begin
            if (rising_edge(clk)) then
                data_out <= rom(to_integer(unsigned(address)));
            end if;
        end process;

    end Behavioral;
```

Memoria RAM

■ Ejemplo 8.3: RAM de escritura síncrona y lectura asíncrona



```
--Infiere RAM de escritura síncrona pero lectura asíncrona
--(por ello en Xilinx se implementara, por defecto, como
distributed RAM)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ram_asinc_read is
    Generic (data_width : integer := 8;
            address_width : integer := 5;
            mem_depth : integer := 32);
    Port (data_in : in STD_LOGIC_VECTOR(data_width-1 downto 0);
          we, clk : in STD_LOGIC;
          address : in STD_LOGIC_VECTOR(address_width-1 downto 0);
          data_out : out STD_LOGIC_VECTOR(data_width-1 downto 0) );
end ram_asinc_read;

architecture Behavioral of ram_asinc_read is
    type mem_type is array (mem_depth-1 downto 0) of
        STD_LOGIC_VECTOR (data_width-1 downto 0);
    signal mem : mem_type;
    begin

        wr: process(clk)
        begin
            if (rising_edge(clk)) then
                if (we = '1') then
                    mem(to_integer(unsigned(address))) <= data_in;
                end if;
            end if;
        end process;

        data_out <= mem(to_integer(unsigned(address)));

    end Behavioral;
```

Máquinas de Estado

■ Ejemplo 8.8: FSM utilizando dos procesos

Se pretende diseñar el algoritmo de control de un robot explorador para una misión en Marte. El robot irá tomando muestras de la superficie del planeta mientras se desplaza. Para controlar su movimiento, el frontal del robot posee un sensor (x) que detecta obstáculos en la trayectoria actual cuando están a menos de una distancia determinada, y dos salidas (zi y zd) que controlan el movimiento de sendas orugas.

En ausencia de obstáculos ($x='0'$), el robot debe avanzar en la dirección en que se encuentre orientado ($zizd='00'$). Cada vez que detecte un obstáculo ($x='1'$), el robot deberá girar para evitarlo y continuar la exploración. En cada obstáculo el giro será a derechas ($zizd='01'$) o izquierdas ($zizd='10'$) de forma alternativa, con el fin de evitar un posible movimiento en círculos y abarcar una mayor zona de exploración.

Diagrama de Estados

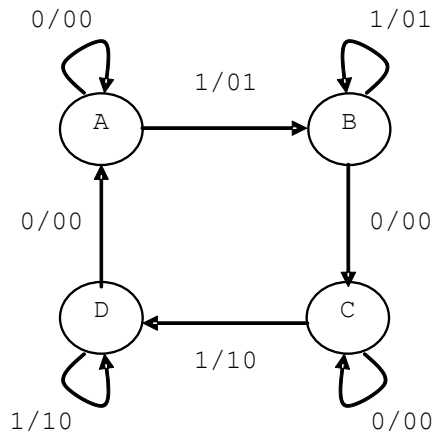


Tabla de Estados

	0	1
A	A/00	B/01
B	C/00	B/01
C	C/00	D/10
D	A/00	D/10

Tabla de Transiciones

	0	1
00	00/00	01/01
01	11/00	01/01
11	11/00	10/10
10	00/00	10/10

Máquinas de Estado

■ Ejemplo 8.8: FSM utilizando dos procesos

```
--dos procesos, salidas combinacionales
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ROBOT IS
    PORT ( CLK,RESET,xin : IN std_logic;
          zd,zi : OUT std_logic);
END;

ARCHITECTURE BEHAVIORAL OF ROBOT IS
    TYPE type_sreg IS (STATE0,STATE1,STATE2,STATE3);
    SIGNAL sreg, next_sreg : type_sreg;
    BEGIN

        PROCESS (CLK, RESET)
            BEGIN
                IF ( RESET='1' ) THEN
                    sreg <= STATE0;
                ELSIF CLK='1' AND CLK'event THEN
                    sreg <= next_sreg;
                END IF;
            END PROCESS;

        ...
```

```
...
        PROCESS (sreg, xin)
            BEGIN
                CASE sreg IS
                    WHEN STATE0 =>
                        IF ( xin='1' ) THEN
                            next_sreg<=STATE1;
                            zi<='0'; zd<='1';
                        ELSE
                            next_sreg<=STATE0;
                            zi<='0'; zd<='0';
                        END IF;
                    WHEN STATE1 =>
                        IF ( xin='0' ) THEN
                            next_sreg<=STATE2;
                            zi<='0'; zd<='0';
                        ELSE
                            next_sreg<=STATE1;
                            zi<='0'; zd<='1';
                        END IF;
                    WHEN STATE2 =>
                        IF ( xin='1' ) THEN
                            next_sreg<=STATE3;
                            zi<='1'; zd<='0';
                        ELSE
                            next_sreg<=STATE2;
                            zi<='0'; zd<='0';
                        END IF;
                    WHEN STATE3 =>
                        IF ( xin='0' ) THEN
                            next_sreg<=STATE0;
                            zi<='0'; zd<='0';
                        ELSE
                            next_sreg<=STATE3;
                            zi<='1'; zd<='0';
                        END IF;
                END CASE;
            END PROCESS;
        END BEHAVIORAL;
```