

PRÁCTICA 1. INTRODUCCIÓN AL VHDL

dtic





PRÁCTICA 1. INTRODUCCIÓN AL VHDL

Índice

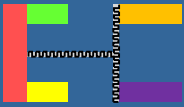
- ⊙ Introducción
- ⊙ Conceptos básicos
- ⊙ Unidades básicas de diseño
- ⊙ Modelado de sistemas I
- ⊙ Simulación de un ejemplo sencillo
- ⊙ Modelado de sistemas II
- ⊙ Propuesta de ejercicio práctico
- ⊙ Elementos del lenguaje
- ⊙ Modelado secuencial
- ⊙ Práctica a implementar





Introducción

- ⊙ VHDL es un lenguaje para describir hardware digital utilizado por la industria en todo el mundo.
- ⊙ **VHDL**: es un acrónimo de: **V**HSIC **H**ardware **D**escription **L**anguage (VHSIC = **V**ery **H**igh **S**peed **I**ntegrated **C**ircuits)
- ⊙ Utilización:
 - ⊙ Descripción hardware
 - ⊙ Simulación
 - ⊙ Síntesis
- ⊙ Beneficios prácticos:
 - ⊙ Es un mecanismo para el diseño digital y para la documentación de diseños reutilizables.
 - ⊙ Diseños reutilizables.



ESTRUCTURA DE UN DISEÑO VHDL

Conceptos básicos

```
LIBRARY IEEE;  
USE IEEE.std_logic_1164.ALL;
```

Declaración de puertos

```
ENTITY Nombre_entidad IS  
[GENERIC (    )];  
PORT (        );  
END Nombre_entidad;
```

Nombre de la entidad

Parte declarativa de la arquitectura

```
ARCHITECTURE Nombre_arquitectura OF Nombre_entidad IS
```

Definición de las señales a usar.
Definición de los tipos y subtipos a utilizar

Cuerpo de la arquitectura

```
BEGIN
```

Se modela el comportamiento del circuito con asignaciones, instanciaciones y procesos

```
END Nombre_arquitectura;
```

Nombre de la arquitectura



DECLARACIÓN DE LAS BIBLIOTECAS

Conceptos básicos

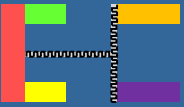
```
LIBRARY nombre_biblioteca;
```

```
USE nombre_biblioteca.nombre_paquete.parte_paquete;
```

```
LIBRARY IEEE;  
USE IEEE.std_logic_1164.ALL;
```

Declaración de biblioteca

Usa todas las definiciones del paquete std_logic_1164



BIBLIOTECAS COMUNMENTE UTILIZADAS

Conceptos básicos

⊙ ieee

- ⊙ Especifica sistemas lógicos de múltiples niveles e incluye los tipos de datos `STD_LOGIC` y `STD_LOGIC_VECTOR`.

} Necesita declararse explícitamente

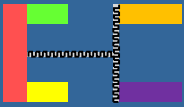
⊙ Std

- ⊙ Especifica tipos de datos predefinidos (`BIT`, `BOOLEAN`, `INTEGER`, `REAL`, `SIGNED`, `UNSIGNED`, etc.), operaciones aritméticas, funciones de conversión básica de tipos, funciones básicas de texto e/s, etc.

} Visibles por defecto

⊙ Work

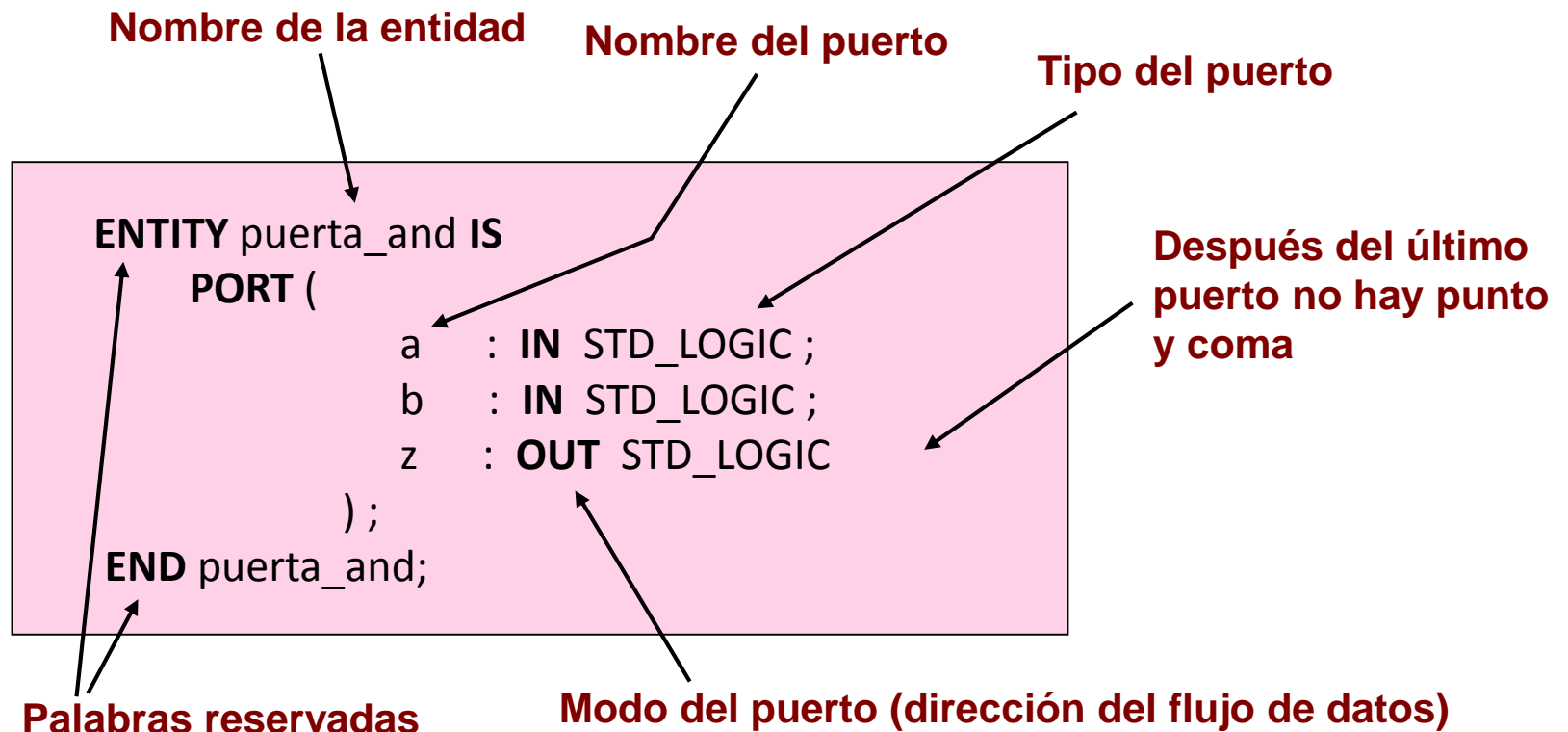
- ⊙ Diseños creados por el usuario después de la compilación.

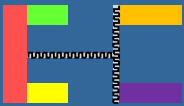


DECLARACIÓN DE LA ENTIDAD

Unidades
básicas de
diseño

- Entidad (ENTITY): Describe la interfaz del componente declarando las entradas y salidas al mismo. Es la caja negra visible para su interconexión.

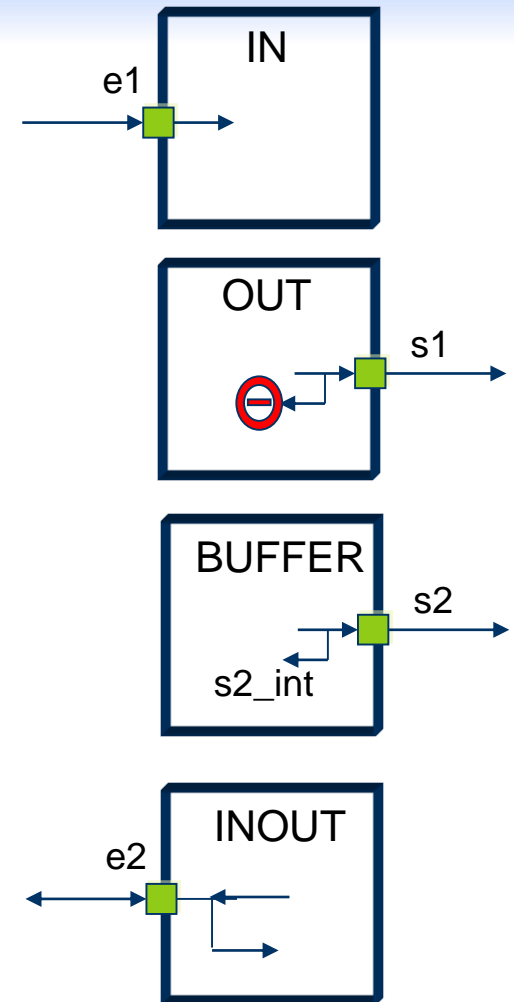


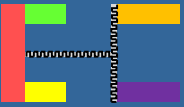


MODOS DE UN PUERTO

Unidades básicas de diseño

- Indican la dirección y si el puerto puede leerse y escribirse dentro de la entidad.
- IN:** Señal que entra en la entidad y no sale. Puede ser leída pero no escrita.
- OUT:** Señal que sale fuera de la entidad no usada internamente. No puede ser leída dentro de la entidad.
- BUFFER:** Señal que sale de la entidad y también es realimentada dentro de la entidad.
- INOUT:** Señal bidireccional, señal de entrada/salida de la entidad.

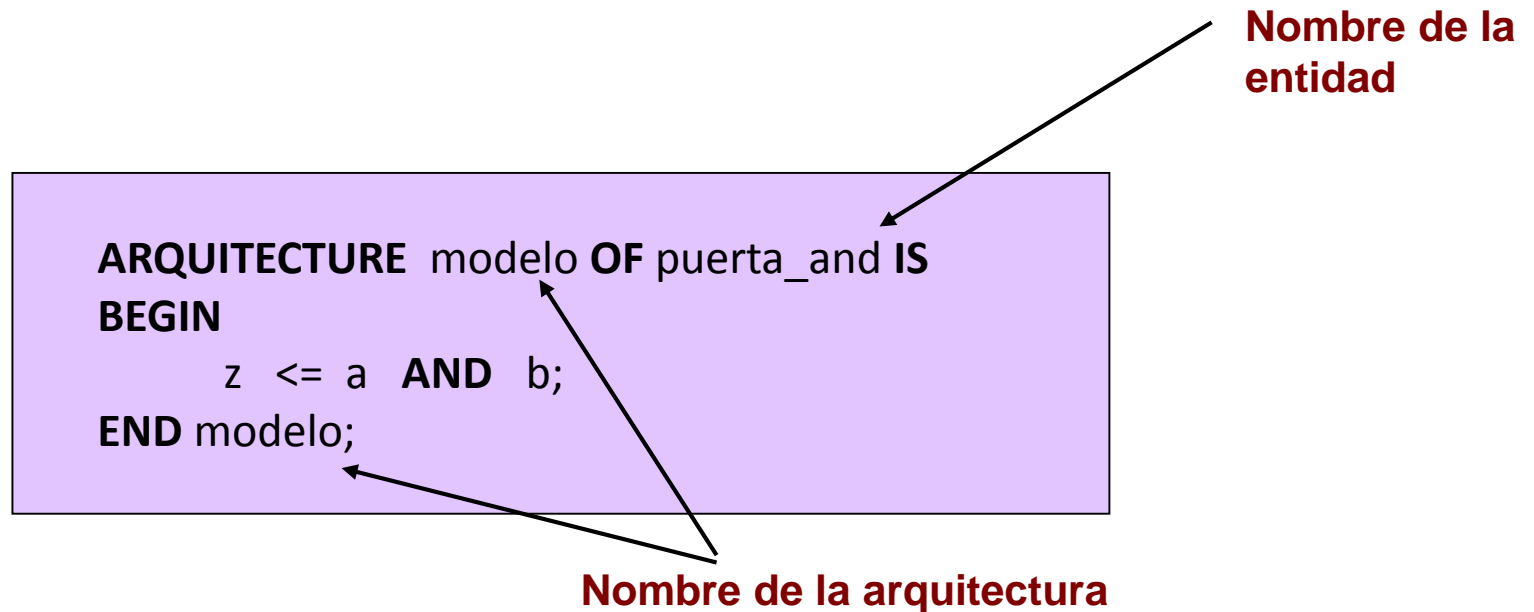


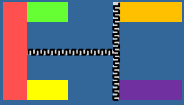


ARQUITECTURA

Unidades básicas de diseño

- ⊙ La entidad describe los puertos del módulo.
- ⊙ Arquitectura (ARCHITECTURE): Corresponde al cuerpo del elemento que se describe. Contiene el código de la descripción funcional del diseño (es decir, qué hace el elemento).



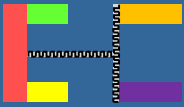


ESTILOS DESCRIPTIVOS

Modelado
de
sistemas I

- ◎ Soporta distintos niveles de descripción de la arquitectura:
 - ◎ **Estructural.** Define explícitamente componentes y la conexión entre ellos. Equivale textualmente a dibujar un esquema.
 - ◎ **Flujo de datos (Dataflow).** Asigna expresiones a señales. (Especifica las ecuaciones lógicas),
 - ◎ **Comportamiento (Behavioral).** Se escribe un algoritmo que describe el comportamiento de un circuito. No proporciona al sintetizador información de cómo será el circuito, siendo éste el que lo determina. El proceso (process) es la parte fundamental de este tipo de descripción.



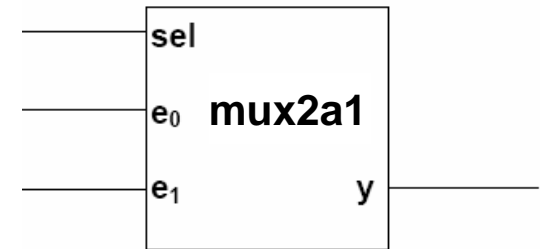


ESTILOS DESCRIPTIVOS - EJEMPLO

Modelado
de
sistemas I

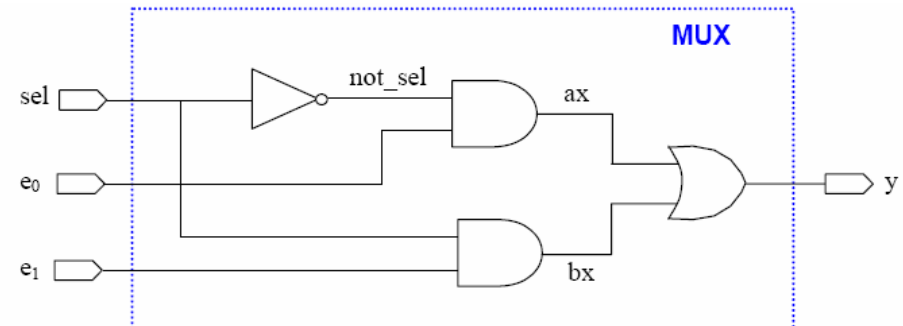
- Declaración de la entidad Multiplexor 2 a 1.

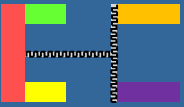
```
ENTITY mux2a1 IS  
  PORT ( e0, e1, sel : IN STD_LOGIC;  
          y           : OUT STD_LOGIC  
        );  
END mux2a1;
```



- Estilo flujo de datos

```
ARCHITECTURE flujo_datos OF mux2a1 IS  
  SIGNAL not_sel, ax, bx : STD_LOGIC;  
  BEGIN  
    not_sel <= NOT sel;  
    ax <= e0 AND not_sel;  
    bx <= e1 AND sel;  
    y <= ax OR bx;  
  END flujo_datos;
```



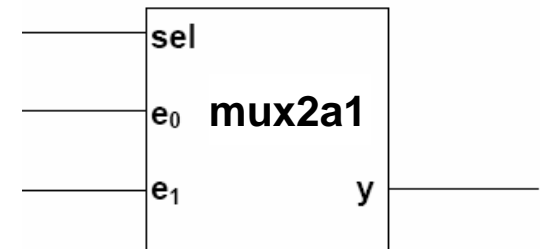


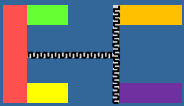
ESTILOS DESCRIPTIVOS - EJEMPLO

Modelado
de
sistemas I

🎯 Estilo Comportamiento

```
ARCHITECTURE comportamiento OF mux2a1 IS  
BEGIN  
    PROCESS (e0,e1,sel)  
    BEGIN  
        IF (sel='0') THEN  
            y<= e0;  
        ELSE  
            y<= e1;  
        END IF;  
    END PROCESS;  
END comportamiento;
```





ESTILOS DESCRIPTIVOS - EJEMPLO

Modelado
de
sistemas I

Estilo Estructural

ARCHITECTURE estructural **OF** mux2a1 **IS**

COMPONENT inv

PORT (e : **IN** STD_LOGIC; y : **OUT** STD_LOGIC);

END COMPONENT;

COMPONENT and2

PORT (e1, e2 : **IN** STD_LOGIC; y : **OUT** STD_LOGIC);

END COMPONENT;

COMPONENT or2

PORT (e1, e2 : **IN** STD_LOGIC; y : **OUT** STD_LOGIC);

END COMPONENT;

SIGNAL ax, bx, not_sel : STD_LOGIC

BEGIN

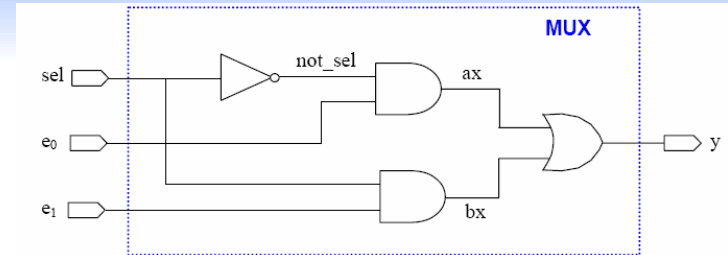
U0 : inv **PORT MAP** (e => sel, sal => not_sel);

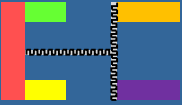
U1 : and2 **PORT MAP** (e1 => e0, e2 => not_sel, sal => ax);

U2 : and2 **PORT MAP** (e1 => e1, e2 => sel, sal => bx);

U3 : or2 **PORT MAP** (e1 => ax, e2 => bx, sal => y);

END estructural;

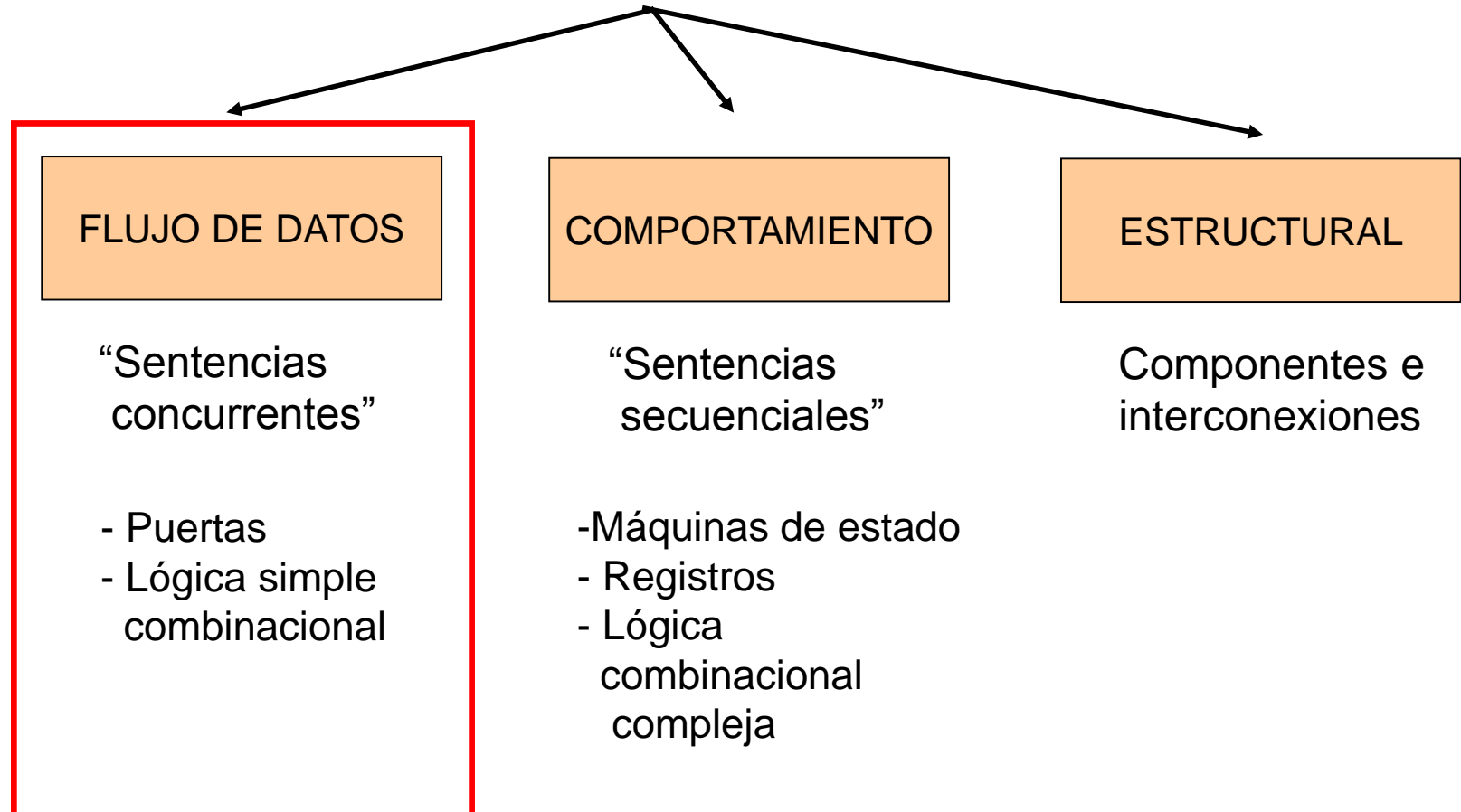


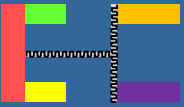


MODELOS DESCRIPTIVOS

Modelado
de
sistemas I

Estilos de diseño VHDL





DESCRIPCIÓN POR FLUJO DE DATOS

Modelado
de
sistemas I

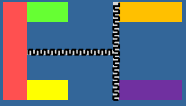
- Describe como se mueven los datos a través del sistema y de varios pasos de procesamiento.
- Utiliza sentencias concurrentes.
- Las sentencias concurrentes se evalúan al mismo tiempo; el orden de las sentencias no importan.
- Esto no es cierto para sentencias secuenciales (o de comportamiento)

Este orden

$\left\{ \begin{array}{l} A_sal \leq AXOR B; \\ Resultado \leq A_sal XOR C; \end{array} \right.$

Es lo mismo que

$\left\{ \begin{array}{l} Resultado \leq A_sal XOR C; \\ A_sal \leq AXOR B; \end{array} \right.$



SENTENCIAS CONCURRENTES

- Asignación simple

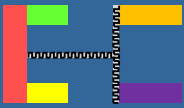
```
suma <= ope1 xor ope2;
```

- Para SEÑALES y PUERTOS, el operador es <=, el lado izquierdo es el destino y el lado derecho la fuente.

- OJO:** fuente y destino tienen que ser del mismo tipo

- Para VARIABLES el operador es :=

```
acarreo_intermedio := ope1 and ope2;
```

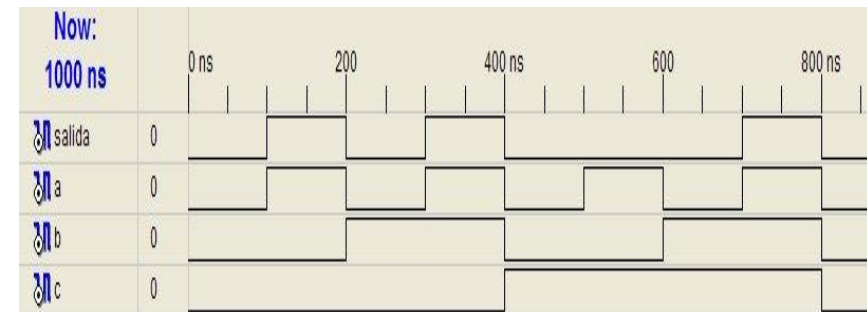
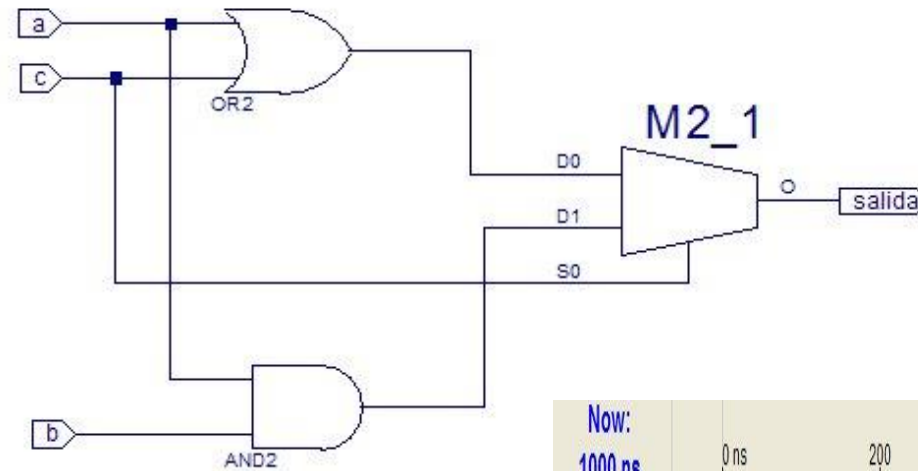


SENTENCIAS CONCURRENTES. WHEN-ELSE

Modelado
de
sistemas I

- Permite realizar asignaciones condicionales

```
salida <= a and b WHEN c = '1' ELSE a or c;
```

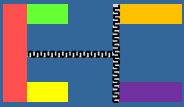




WHEN - ELSE

Modelado
de
sistemas I

```
[Etiqueta]: Señal <=  Valor_1 WHEN Condición_1 ELSE  
                        Valor_2 WHEN Condición_2 ELSE  
                        ...  
                        Valor_N-1 WHEN Condición_N-1 ELSE  
                        Valor_N;
```



EJEMPLO. BUFFER TRI-ESTADO

Modelado
de
sistemas I

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

```
ENTITY tri_estado IS
```

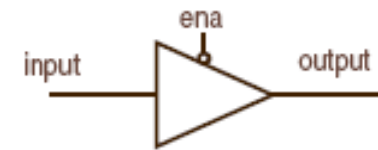
```
    PORT (   ena:   IN STD_LOGIC;  
            input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);  
            output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)  
        );
```

```
END tri_estado;
```

```
ARCHITECTURE tri_estado_flujo OF tri_estado IS  
BEGIN
```

```
    output <= input WHEN (ena = '0') ELSE  
        (OTHERS => 'Z');
```

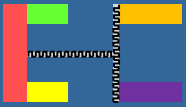
```
END tri_estado_flujo;
```



Bus de 8 bits.
El bit de más a la derecha es el LSB



OTHERS significa todos los bits no especificados directamente,
en este caso todos los bits



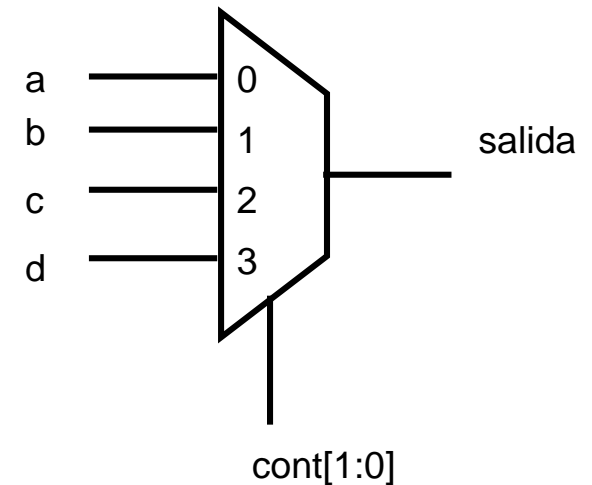
SENTENCIAS CONCURRENTES. WITH-SELECT

Modelado
de
sistemas I

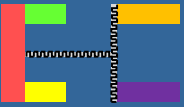
- ⊙ Permite realizar asignaciones selectivas

WITH cont **SELECT**

```
salida <= a WHEN "00",  
          b WHEN "01",  
          c WHEN "10",  
          d WHEN OTHERS;
```



- ⊙ Por su ejecución en paralelo (balanceada) es similar a un CASE
- ⊙ Se pueden dar problemas si no se pone el último **WHEN** OTHERS

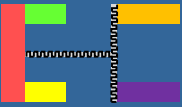


WITH - SELECT

Modelado
de
sistemas I

[Etiqueta]: **WITH** expresión **SELECT**

```
Señal <= valor_1 WHEN resultado_1 [,
        valor_2 WHEN resultado_2 ] [,
        ... ] [,
        valor_N WHEN resultado_N] [,
        valor_cuando_otros WHEN OTHERS];
```



EJEMPLO. WITH - SELECT

Modelado
de
sistemas I

```
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL;  
  
ENTITY circuito IS  
    PORT ( A, B    : IN STD_LOGIC;  
           enable : IN STD_LOGIC;  
           salida  : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)  
    );  
END circuito;
```

enable	A	B	salida
0	0	0	ZZ
0	0	1	ZZ
0	1	0	ZZ
0	1	1	ZZ
1	0	0	00
1	0	1	01
1	1	0	01
1	1	1	10

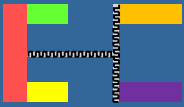
```
ARCHITECTURE con_with_select OF circuito IS  
BEGIN  
    WITH ( enable & A & B ) SELECT
```

```
        salida <= "00" WHEN "100",  
                "01" WHEN "101", -- Puede ser también "01" WHEN "101" | "110"  
                "01" WHEN "110", -- Este se quitaría  
                "10" WHEN "111",  
                "ZZ" WHEN OTHERS;
```

```
END tcon_with_select;
```

**Símbolo para asignar el mismo
valor a varios resultados**





UN EJEMPLO SENCILLO

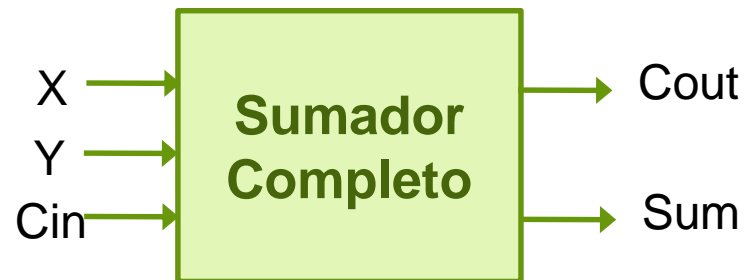
Simulación
de un
ejemplo
sencillo

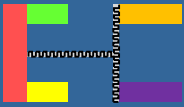
- Sumador completo mediante descripción de flujo de datos
- Entidad: define la interfaz del sistema

ENTITY Sumador_completo **IS**

```
PORT (X, Y, Cin : IN  STD_LOGIC;           - - Entradas
        Sum, Cout : OUT STD_LOGIC);         - - Salidas
```

END Sumador_completo;





UN EJEMPLO SENCILLO

Simulación
de un
ejemplo
sencillo

- El sistema puede describirse internamente de muchas formas

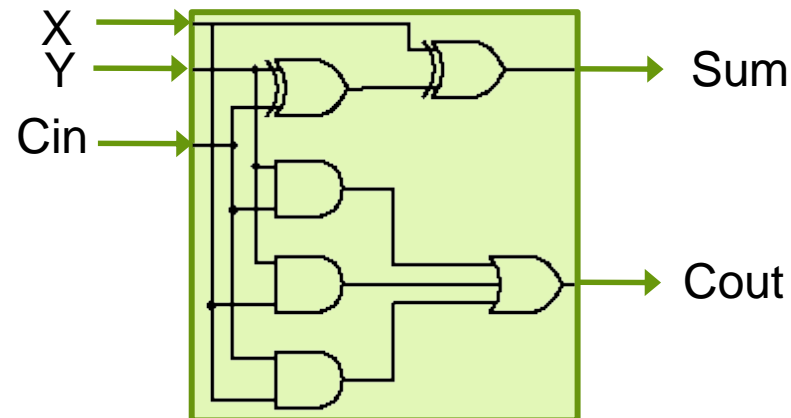
ARCHITECTURE Ecuaciones **OF** Sumador_completo **IS**

BEGIN - - Asignaciones concurrentes

Sum <= X **XOR** Y **XOR** Cin;

Cout <= (X **AND** Y) **OR** (X **AND** Cin) **OR** (Y **AND** Cin);

END Ecuaciones;



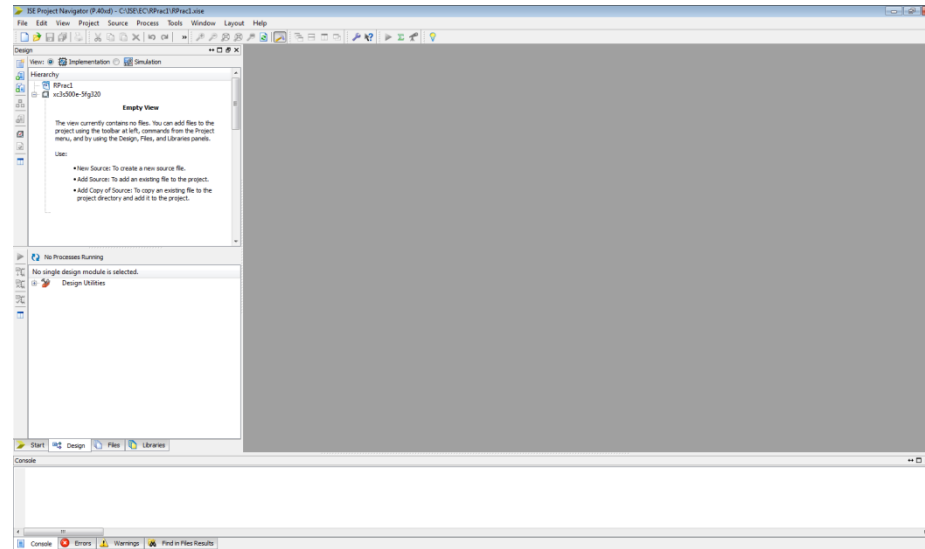


ARRANQUE DEL ENTORNO

Simulación
de un
ejemplo
sencillo

- 🎯 Arranque del entorno Xilinx ISE 14.7. Desde el menú Inicio de Windows se accede a la aplicación Project Navigator:

Programas -> Xilinx Design Tools -> ISE Design Suite 14.7 -> ISE Design Tools -> Project Navigator



- 🎯 Comenzaremos creando un nuevo proyecto desde el menú *File*, que iniciará un asistente en el que donde deberemos indicar un nombre y el lugar donde queremos que ubique este proyecto y los archivos que se generen.



ARRANQUE DEL ENTORNO

Simulación
de un
ejemplo
sencillo

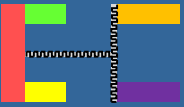
Nombre del Proyecto

Ubicación

Tipo de Diseño

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Isim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

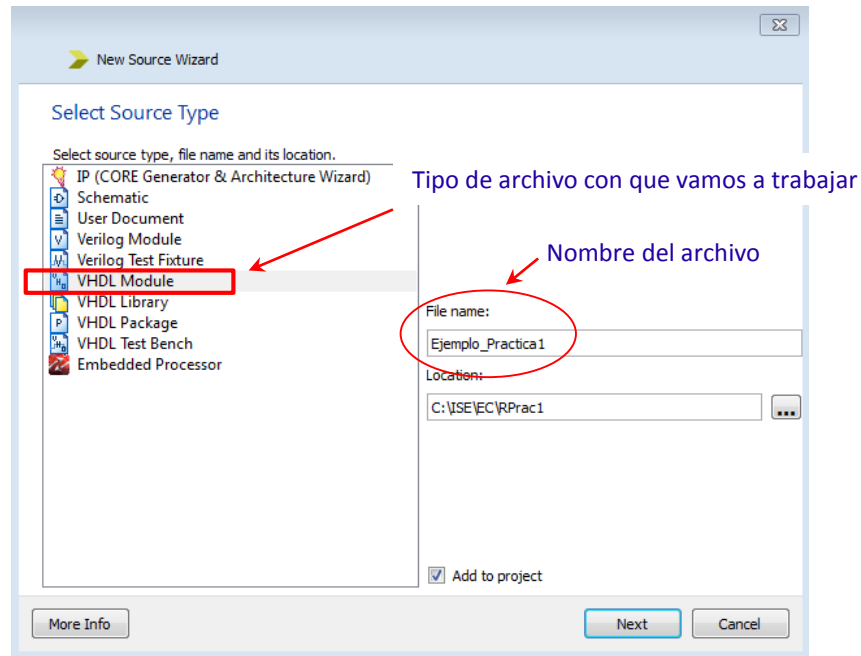
- Tras pulsar *Next*, obtenemos el cuadro correspondiente a los ajustes del dispositivo sobre el que se quiere trabajar. Los valores por defecto son los adecuados, pero debemos asegurarnos que en **Simulator** aparece *Isim* y que en **Prefered Language** tenemos *VHDL*.
- Pulsamos nuevamente *Next* y nos aparecerá un cuadro resumen de las opciones que hemos elegido, y tras pulsar *Finish* podremos empezar a trabajar en nuestro entorno.

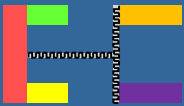


CREACIÓN DEL ESPACIO DE TRABAJO

Simulación
de un
ejemplo
sencillo

- Empezaremos por crear el fichero fuente que contendrá nuestro diseño. Para ello, en la opción de menú *Project*, seleccionaremos *New Source*. Nos aparecerá un nuevo cuadro de en el que deberemos seleccionar "VHDL Module" y darle el nombre que queramos.





GENERACIÓN DE UN FICHERO VHDL

Simulación
de un
ejemplo
sencillo

- Tras pulsar *Next* nos aparecerá un asistente que nos ayudará a crear nuestra ENTIDAD (ENTITY, la caja negra en la que describíamos las entradas y salidas que forman parte de nuestro circuito).

Define Module

Specify ports for module.

Entity name: Sumador_Completo

Architecture name: Ecuaciones

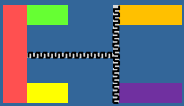
Port Name	Direction	Bus	MSB	LSB
X	in	<input type="checkbox"/>		
Y	in	<input type="checkbox"/>		
Cin	in	<input type="checkbox"/>		
Sum	out	<input type="checkbox"/>		
Cout	out	<input type="checkbox"/>		

More Info Next Cancel

Nombres de la entidad y de la arquitectura que le asignamos

Nombres de las conexiones e indicación de si se trata de entradas o salidas

- Si lo completamos y pulsamos *Next* nos aparecerá un resumen en el que se nos mostrarán estos mismos datos y pulsando *Finish*, nos aparecerá la pantalla principal del *Project Navigator* con nuestro proyecto inicializado.



GENERACIÓN DE UN FICHERO VHDL

Simulación
de un
ejemplo
sencillo

ISE Project Navigator (P.40xd) - C:\ISE\EC\RPrac1\RPrac1.xise - [Ejemplo_Practical1.vhd*]

File Edit View Project Source Process Tools Window Layout Help

Design View: Implementation Simulation

Hierarchy

- RPrac1 ← Nombre del proyecto
 - xc3s500e-5fg320
 - Sumador_Completo - Ecuaciones (Ejemplo_Practical1.vhd) ← Nombre de la entidad y la arquitectura, junto con el nombre del archivo VHDL creado

No Processes Running

Processes: Sumador_Completo - Ecuaciones

- Design Summary/Reports
- Design Utilities
- User Constraints
- Synthesize - XST
- Implement Design
- Generate Programming File
- Configure Target Device
- Analyze Design Using ChipScope

```
1  -- Company: DTIC / UA
2  -- Engineer: F. Javier Brotons
3  --
4  --
5  -- Create Date: 10:13:28 11/06/2014
6  -- Design Name: Sumador Completo
7  -- Module Name: Sumador_Completo - Ecuaciones
8  -- Project Name: Práctica 1
9  -- Revision: 1.0
10
11
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14
15 entity Sumador_Completo is
16     Port ( X : in STD_LOGIC;
17           Y : in STD_LOGIC;
18           Cin : in STD_LOGIC;
19           Sum : out STD_LOGIC;
20           Cout : out STD_LOGIC);
21 end Sumador_Completo;
22
23 architecture Ecuaciones of Sumador_Completo is
24 begin
25
26
27
28 end Ecuaciones;
29
```

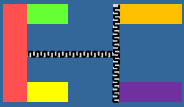
Entidad creada por el asistente

Definición de la arquitectura creada por el asistente, que debemos completar

Console

```
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/ISE/EC/RPrac1/Ejemplo_Practical1.vhd" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
```

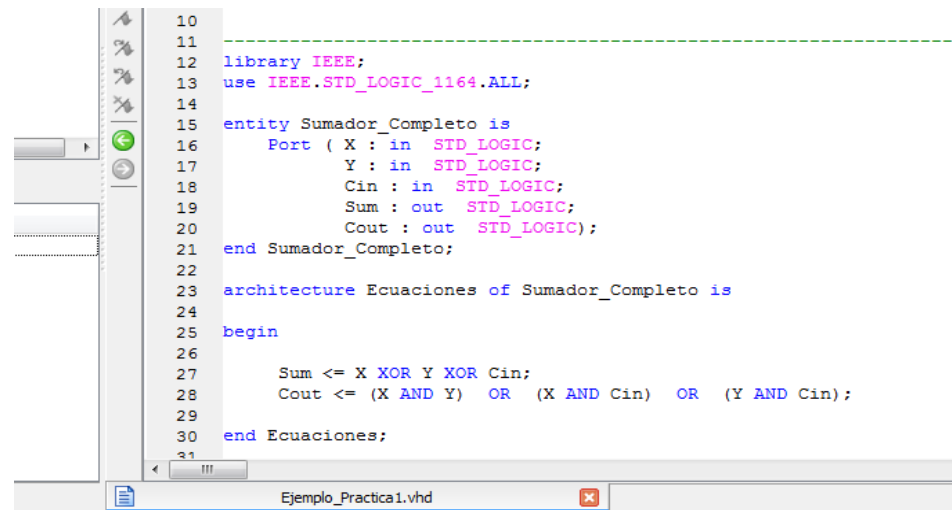
Ln 15 Col 1 | VHDL



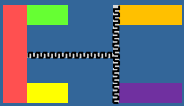
COMPLETANDO EL PROYECTO

Simulación
de un
ejemplo
sencillo

- El asistente es una ayuda, pero no tenemos por qué utilizarlo. Podemos crear nuestros archivos VHDL en cualquier editor de texto previamente y luego incorporarlos a nuestro proyecto a través de la opción disponible menú desplegable (*Project -> Add Source*).
- Como podemos comprobar la entidad, en este caso, está perfectamente generada y no necesita que añadamos ni modifiquemos nada, pero la descripción de la arquitectura está en vacía y debemos completarla.



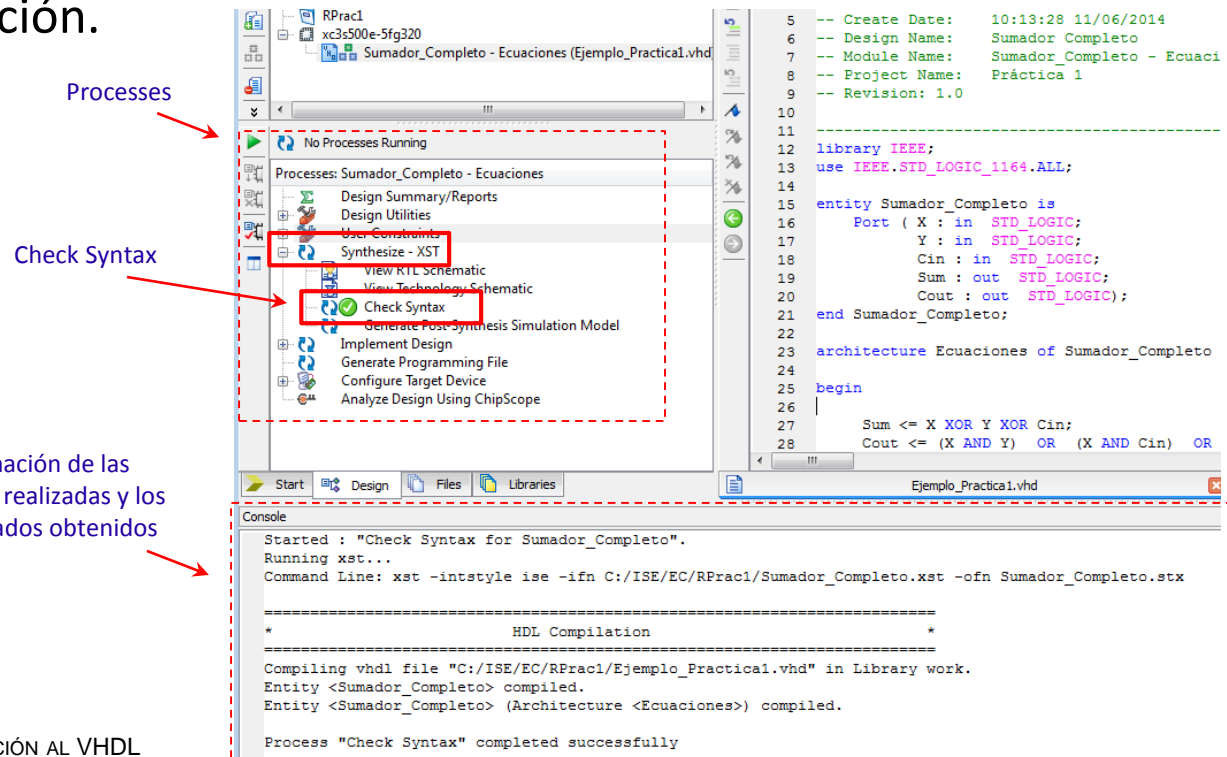
```
10
11
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14
15 entity Sumador_Completo is
16     Port ( X : in  STD_LOGIC;
17           Y : in  STD_LOGIC;
18           Cin : in  STD_LOGIC;
19           Sum : out STD_LOGIC;
20           Cout : out STD_LOGIC);
21 end Sumador_Completo;
22
23 architecture Ecuaciones of Sumador_Completo is
24
25 begin
26
27     Sum <= X XOR Y XOR Cin;
28     Cout <= (X AND Y) OR (X AND Cin) OR (Y AND Cin);
29
30 end Ecuaciones;
31
```

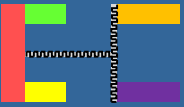


COMPROBAR SINTAXIS Y COMPILACIÓN

Simulación
de un
ejemplo
sencillo

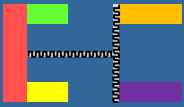
- Una vez completado, comprobaremos si todo es correcto y compilaremos el diseño. Para ello, en el cuadro correspondiente a los *Processes* (tareas que es capaz de realizar el programa) abriremos el desplegable de *Synthesize - XST* y seleccionaremos *Check Syntax*. Si todo está bien estaremos en disposición de continuar con la simulación.





Simulación de un ejemplo sencillo

- ⦿ Si la compilación se ha realizado correctamente, debemos comprobar que nuestro diseño es correcto y que su comportamiento es el esperado . Para ello debemos generar un archivo de test (*testbench*).
- ⦿ El testbench también consistirá en una descripción VHDL, en el que la entidad carece de puertos de entrada y salida. Su arquitectura siempre es de tipo estructural e incluye la entidad que se quiere probar (*uut : unit under test*) y las señales necesarias para ello.
- ⦿ A esta unidad se le aplican un conjunto de vectores de prueba, de los cuales se conoce la salida.
- ⦿ Una vez simulada, se compara la salida que ofrece la *uut* con la salida esperada, y si no son iguales es que ha habido algún error de diseño.
- ⦿ En nuestro caso, realizar el testbench es sencillo y exhaustivo, ya que la entidad implementa es un simple sumador completo.

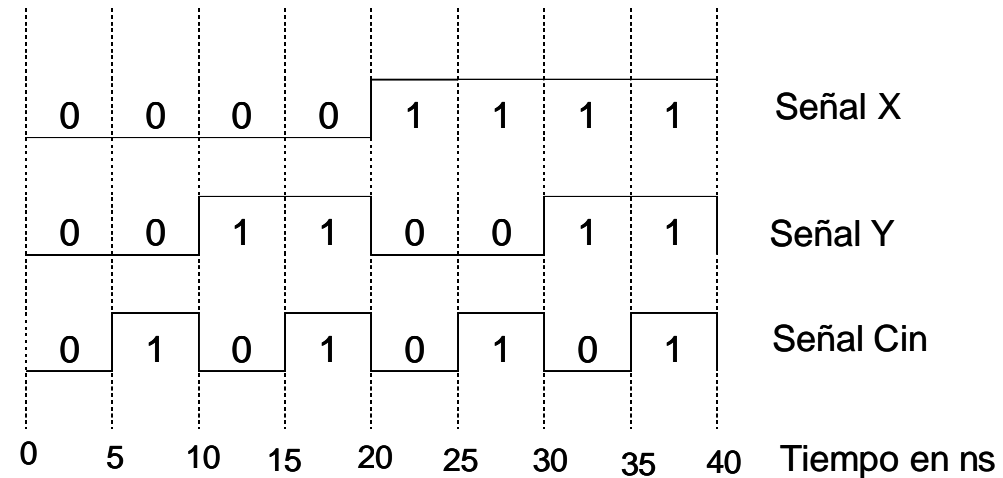


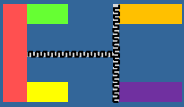
TEST-ESTÍMULOS DE ENTRADA

Simulación
de un
ejemplo
sencillo

- El test debe contener todas las posibles combinaciones de los valores de entrada (si las dimensiones del problema lo permiten):

Entradas			Salidas	
X	Y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

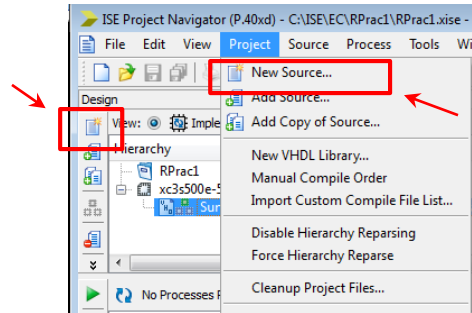




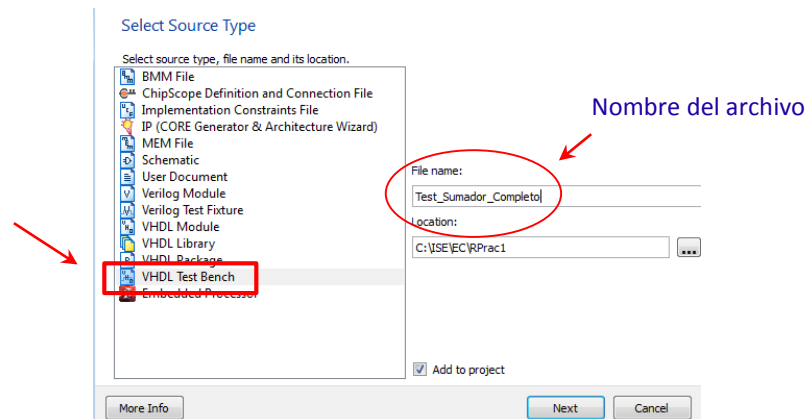
GENERACIÓN DEL ARCHIVO DE TEST

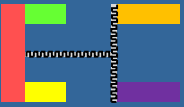
Simulación
de un
ejemplo
sencillo

- Para generar nuestro archivo de test seleccionaremos la opción disponible menú desplegable (*Project -> New Source*), o bien mediante el icono superior situado a la izquierda del cuadro correspondiente al Diseño.



- Nos aparecerá el cuadro de selección del tipo de archivo y ahora elegiremos VHDL Test Bench, le daremos un nombre y pulsaremos *Next*.



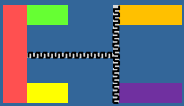


GENERACIÓN DEL ARCHIVO DE TEST

Simulación
de un
ejemplo
sencillo

- ⦿ Nos aparece un nuevo cuadro de diálogo en el que nos pedirá que seleccionemos el archivo fuente al que queremos asociarlo. En este caso sólo aparecerá “Sumador_Completo” puesto que es el único diseño que hemos creado hasta ahora.
- ⦿ Volveremos a pulsar Next y nos aparecerá el cuadro resumen del archivo que se va a generar tras pulsar *Finish*.
- ⦿ El archivo que se ha creado incluye la mayoría de los elementos necesarios para la simulación, si bien también incluye la estructura para la definición de una señal de reloj que, puesto que no nos es necesaria, deberemos suprimir.
- ⦿ Veamos las diferentes partes del archivo generado.





ARCHIVO DE TEST

Simulación
de un
ejemplo
sencillo

```
-- Company: DTIC /UA
-- Engineer: F. Javier Brotons
--
-- Create Date: 12:11:42 11/06/2014
-- Design Name:
-- Module Name: C:/ISE/EC/RPrac1/Test_Sumador_Completo.vhd
-- Project Name: RPrac1
-- Revision 0.01 - File Created
```

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

```
ENTITY Test_Sumador_Completo IS
END Test_Sumador_Completo;
```

Declaración de la Entidad (Vacía, sin puertos de entrada y salida)

```
ARCHITECTURE behavior OF Test_Sumador_Completo IS

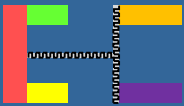
    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Sumador_Completo
    PORT(
        X : IN  std_logic;
        Y : IN  std_logic;
        Cin : IN  std_logic;
        Sum : OUT std_logic;
        Cout : OUT std_logic
    );
END COMPONENT;
```

Arquitectura, que incluye el elemento a simular (COMPONENT) con sus pines de entrada y salida

```
--Inputs
```

Definición de las señales que conectaremos a los



ARCHIVO DE TEST

Simulación de un ejemplo sencillo

```
--Inputs
signal X : std_logic := '0';
signal Y : std_logic := '0';
signal Cin : std_logic := '0';
```

```
--Outputs
signal Sum : std_logic;
signal Cout : std_logic;
```

```
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name
```

```
constant <clock>_period : time := 10 ns;
```

BEGIN

```
-- Instantiate the Unit Under Test (UUT)
uut: Sumador_Completo PORT MAP (
    X => X,
    Y => Y,
    Cin => Cin,
    Sum => Sum,
    Cout => Cout
);
```

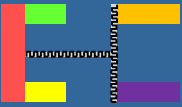
```
-- Clock process definitions
<clock>_process :process
begin
    <clock> <= '0';
    wait for <clock>_period/2;
    <clock> <= '1';
    wait for <clock>_period/2;
end process;
```

Definición de las señales que conectaremos a los pines de entrada y salida del elemento que queremos comprobar. Las entradas están inicializadas por defecto a 0.

Definición de un posible periodo para un reloj. Puesto que no tenemos reloj **ELIMINAREMOS** esta línea

Conexión de las señales a los pines de entrada y salida del elemento a comprobar.

Definición de un reloj. Puesto que no tenemos reloj **ELIMINAREMOS** todo este proceso



ARCHIVO DE TEST

Simulación de un ejemplo sencillo

```
-- Stimulus process
stim_proc: process
begin
    -- hold reset state for 100 ns.
    wait for 100 ns;

    wait for <clock>_period*10;

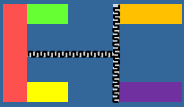
    -- insert stimulus here

    wait;
end process;

END;
```

Ahora debemos dar los valores a las señales, que serán los valores que queden conectados a los puertos de entrada. Puesto que no lo vamos a hacer en forma de proceso, **ELIMINAREMOS** todo este bloque y lo sustituiremos por los que tenemos más abajo

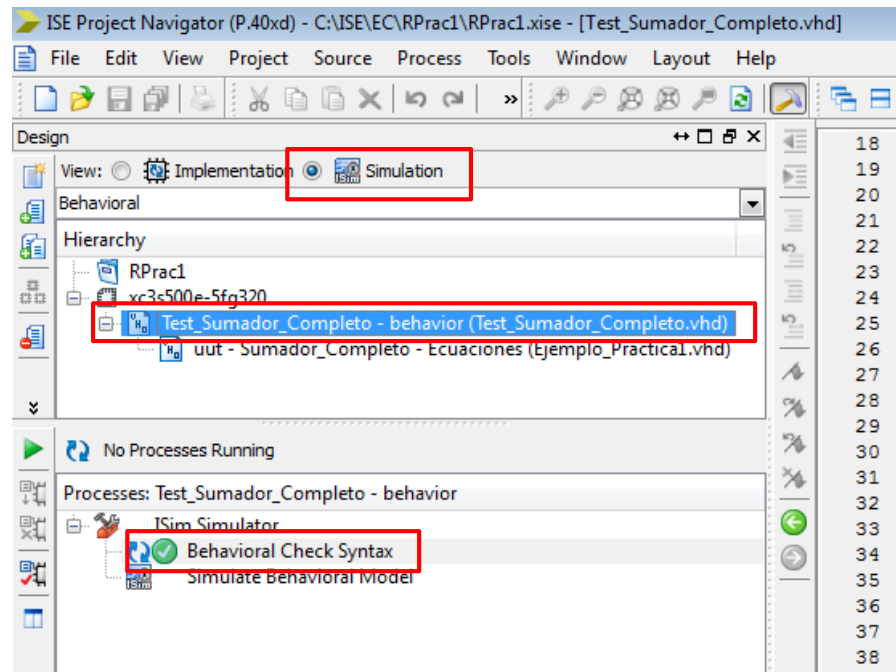
```
-- Valor de las señales de entrada
Cin <= '0', '1' after 100 ns, '0' after 200 ns, '1' after 300 ns, '0' after 400 ns,
'1' after 500 ns, '0' after 600 ns, '1' after 700 ns;
Y <= '0', '1' after 200 ns, '0' after 400 ns, '1' after 600 ns;
X <= '0', '1' after 400 ns;
```

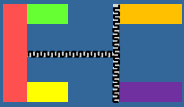


CONDICIONES DE SIMULACIÓN

Simulación
de un
ejemplo
sencillo

- ⦿ A continuación comprobaremos que hemos escrito correctamente nuestro test:
 - ⦿ En la ventana correspondiente diseño (*Design*), marcaremos ahora la opción *View Simulation*.
 - ⦿ Seleccionaremos nuestro archivo de test (En este caso solo está el archivo *Test_Sumador_Completo.vhd*).
 - ⦿ En la ventana *Processes*, dentro de *ISim Simulator*, seleccionaremos *Behavioral Check Syntax*.

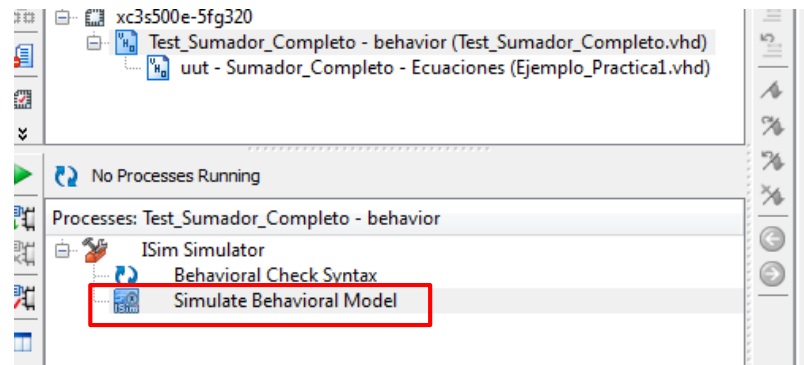




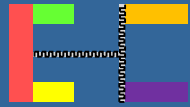
ACTIVACIÓN DEL SIMULADOR

Simulación
de un
ejemplo
sencillo

- Si todo ha ido bien, estaremos en disposición de simular nuestro diseño.
- Para ello solo tendremos que elegir la opción que sigue : *Simulate Behavioral Model*

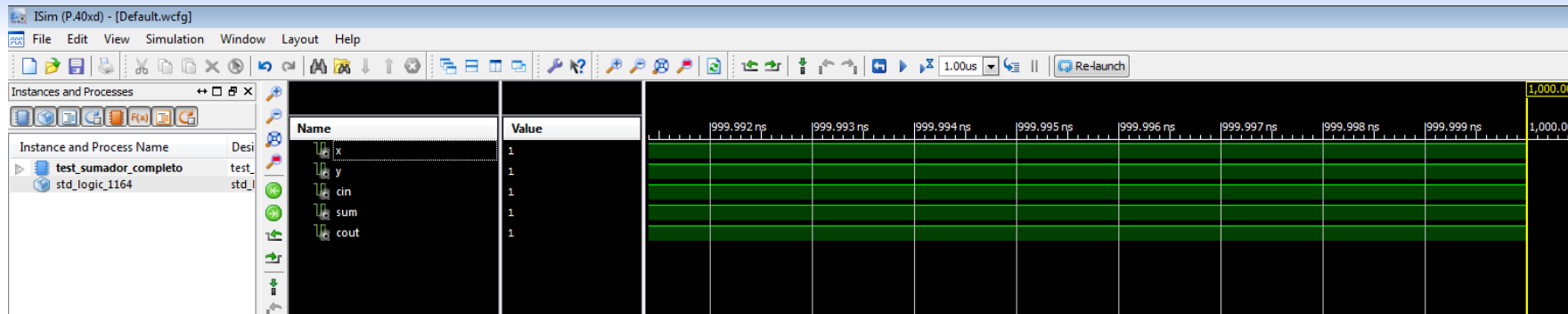


- Se abrirá un nuevo programa: ISim, que nos mostrará los resultados del test que acabamos de diseñar con las señales que hayamos definido (en este caso X, Y, cin, cout, sum).
- Por defecto simula 1 μ s con una resolución de 1ps, y veremos la pantalla siguiente:

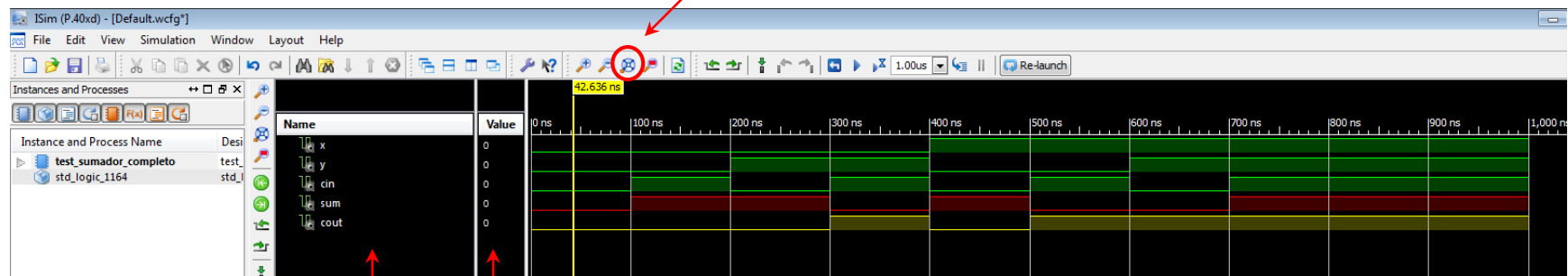


EJECUCIÓN DEL SIMULADOR

Simulación
de un
ejemplo
sencillo

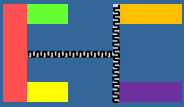


- Estos valores de defecto no son adecuados para nuestra simulación, pero basta con que seleccionemos *full view* para visualizar los resultados que estamos obteniendo de forma apropiada:



Señales

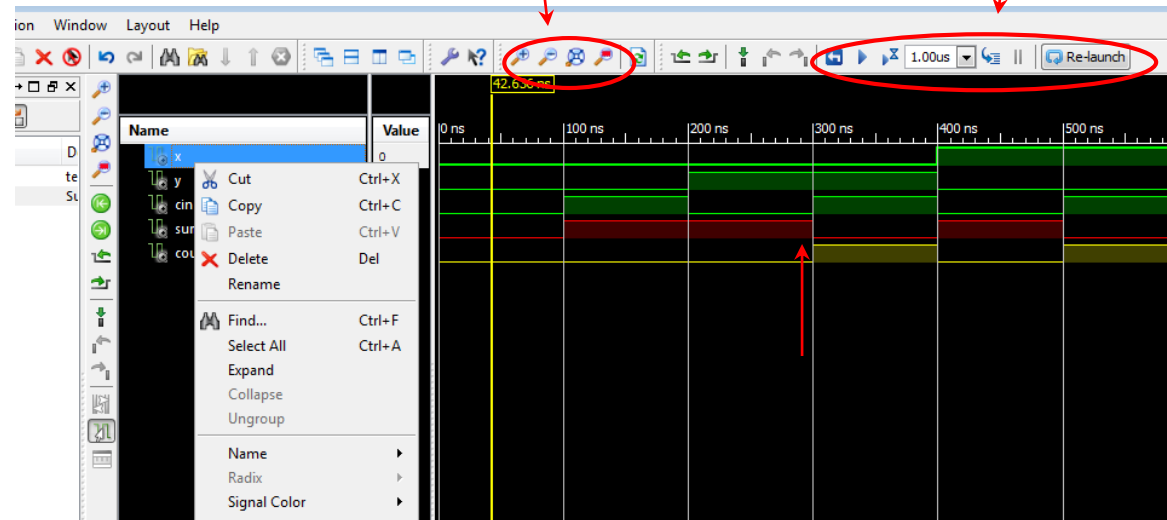
Valor de la señal en la posición del cursor

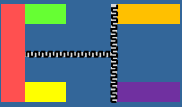


VISUALIZACIÓN DE LA SIMULACIÓN

Simulación
de un
ejemplo
sencillo

- Además de las opciones de zoom, la barra superior de opciones nos ofrece multiples posibilidades como incrementar el tiempo de simulación sobre el existente, modificar el tiempo de simulación o incluso reiniciarlo y ejecutarla paso a paso.
- Seleccionando una señal, mediante el botón derecho del ratón podremos acceder a opciones particulares como carbiarle el nombre, el color, la base de representación, agrupar varias, etc.

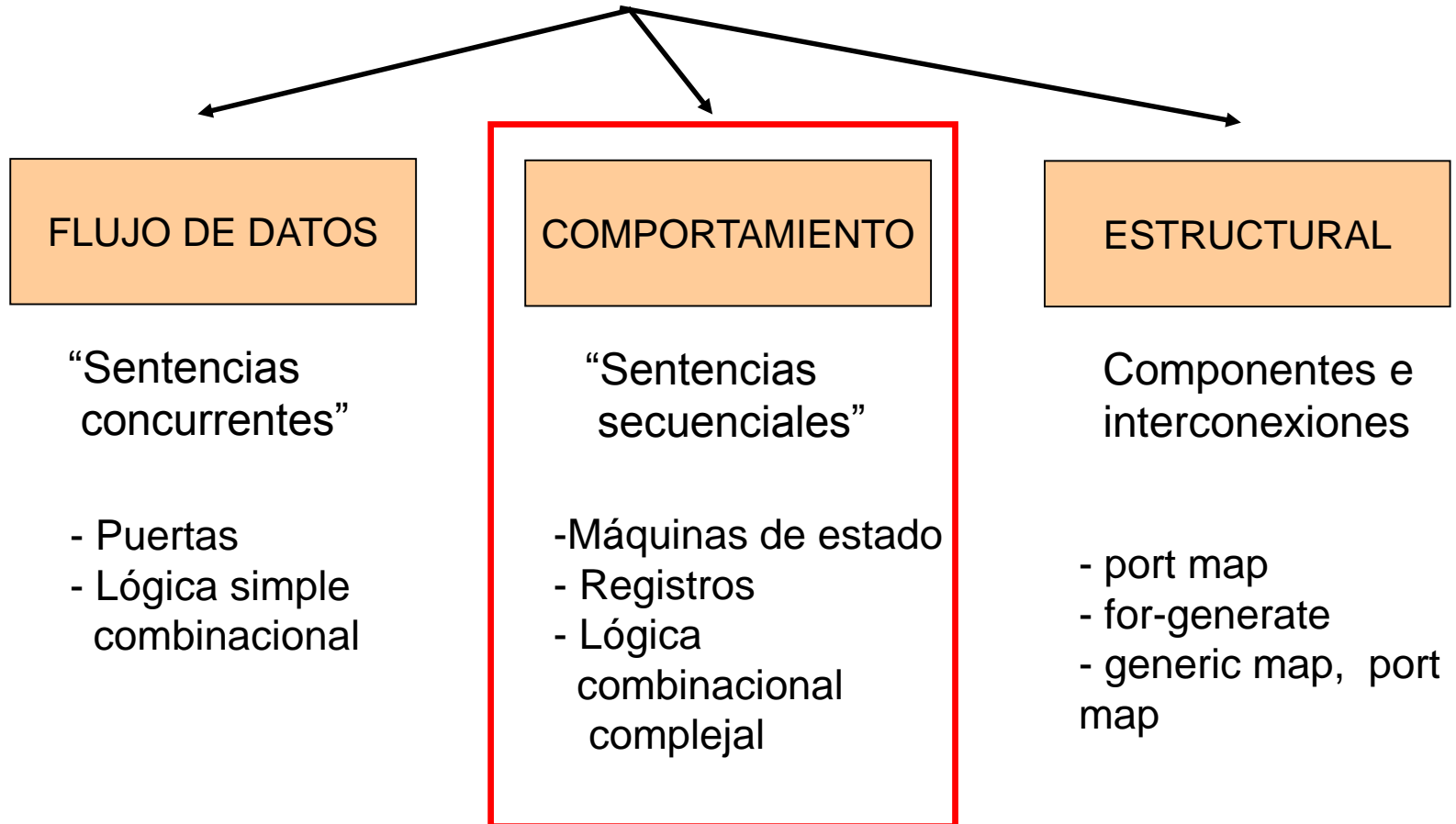


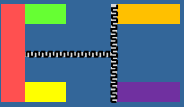


MODELOS DESCRIPTIVOS

Modelado
de
sistemas II

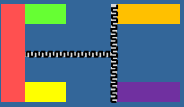
Estilos de diseño VHDL





- ⊙ El process es el elemento de diseño principal.
- ⊙ Es una secuencia de instrucciones denominadas sentencias secuenciales.

```
[Etiqueta]: PROCESS (lista de sensibilidad)
    ...
    parte declarativa (variables, procedimientos, tipos, etc=)
    ...
    BEGIN
        ...
        Sentencias que describen el comportamiento
        ...
    END PROCES [Etiqueta];
```

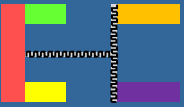


CARACTERÍSTICAS DEL PROCESS

Modelado
de
sistemas II

- ⊙ Los procesos se “disparan” (su código se ejecuta) cuando cambia alguna de las señales de su lista de sensibilidad.
- ⊙ Un proceso sin lista de sensibilidad es válido pero se activa con cualquier evento.
- ⊙ Las instrucciones dentro del proceso se ejecutan secuencialmente sin avanzar el tiempo durante su ejecución.
- ⊙ Las señales cambian su valor cuando avanza el tiempo.
- ⊙ Una arquitectura puede tener más de un proceso. Todos los procesos se ejecutarán en paralelo.
- ⊙ Cuando se ejecuta la última sentencia el control se pasa al inicio del proceso.





SENTENCIAS SECUENCIALES - IF

⊙ Sintaxis

```
IF Condición_1 THEN
    Sentencias secuenciales
ELSIF Condición_2 THEN
    Sentencias secuenciales
ELSE Condición_3 THEN
    Sentencias secuenciales
END IF;
```

- ⊙ `else` y `elsif` son opcionales.
- ⊙ Es el tipo de sentencia más comunmente utilizada.



EJEMPLO- IF

Modelado
de
sistemas II

Selector : **PROCESS** (reset, clock)

BEGIN

IF reset = 1 **THEN**

f <= (others => '0');

ELSIF (Clock = '1' **AND** Clock'EVENT) **THEN**

IF Sel = "00" **THEN**

f <= x1;

ELSIF Sel = "10" **THEN**

f <= x2;

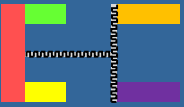
ELSE

f <= x3;

END IF;

END IF;

END PROCESS;



SENTENCIAS SECUENCIALES - CASE

⊙ Sintaxis

```
CASE expresión IS
    WHEN caso1 =>
        secuencia sentencias1;
    WHEN caso2 =>
        secuencia sentencias2;
    ...
    WHEN cason =>
        secuencia sentenciasn;
    WHEN OTHERS =>
        resto de casos;
END CASE;
```

- ⊙ Los casos tienen que cubrir todos los posibles valores de la expresión.
- ⊙ Se utiliza **others** para considerar el resto de casos.

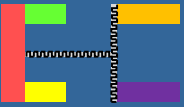


EJEMPLO- CASE

```
ope_ALU : PROCESS ( a, b, ope)
BEGIN
    CASE ope IS
        WHEN "00" =>
            resultado <= a + b ;
        WHEN "01" =>
            resultado <= a - b ;
        WHEN "10" =>
            resultado <= a AND b ;
        WHEN "11" =>
            resultado <= a OR b ;
        WHEN OTHERS =>
            resultado <= ( OTHERS => 'Z' ) ;

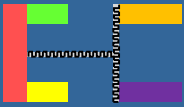
    END CASE;
END PROCESS;
```





- ⊙ Son características que se pueden asociar a cualquier elemento VHDL.
 - ⊙ Atributos definidos por el usuario
 - ⊙ Predefinidos para señales: dan información sobre las señales o definen nuevas señales implícitas derivadas.
- ⊙ Algunos atributos para señales:

S'DELAYED (t)	Retarda la señal S t unidades de tiempo
S'STABLE(t)	Verdadero cuando no ha habido eventos en S durante t.
S'QUIET(t)	Verdadero cuando no ha habido transacciones en S durante t
S'EVENT	Verdadero cuando se ha producido un evento en S
S'ACTIVE	Verdadero cuando se ha producido una transacción en S
S'LAST_ACTIVE	Tiempo transcurrido desde la última transacción en S
S'LAST_VALUE	Valor de S antes de que ocurriera el último evento



EJEMPLOS DE ATRIBUTOS

⊙ Algunos ejemplos de atributos para señales:

(clock='1') **AND** (clock'**STABLE**)

Señal clock = '1'

(clock='1') **AND** (**NOT** clock'**EVENT**)

Señal clock = '1'

(clock='1') **AND** (**NOT** clock'**STABLE**)

Flanco de subida de la señal clock

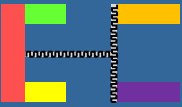
(clock='1') **AND** (clock'**EVENT**)

Flanco de subida de la señal clock

(clock='0') **AND** (clock'**EVENT**)

Flanco de bajada de la señal clock

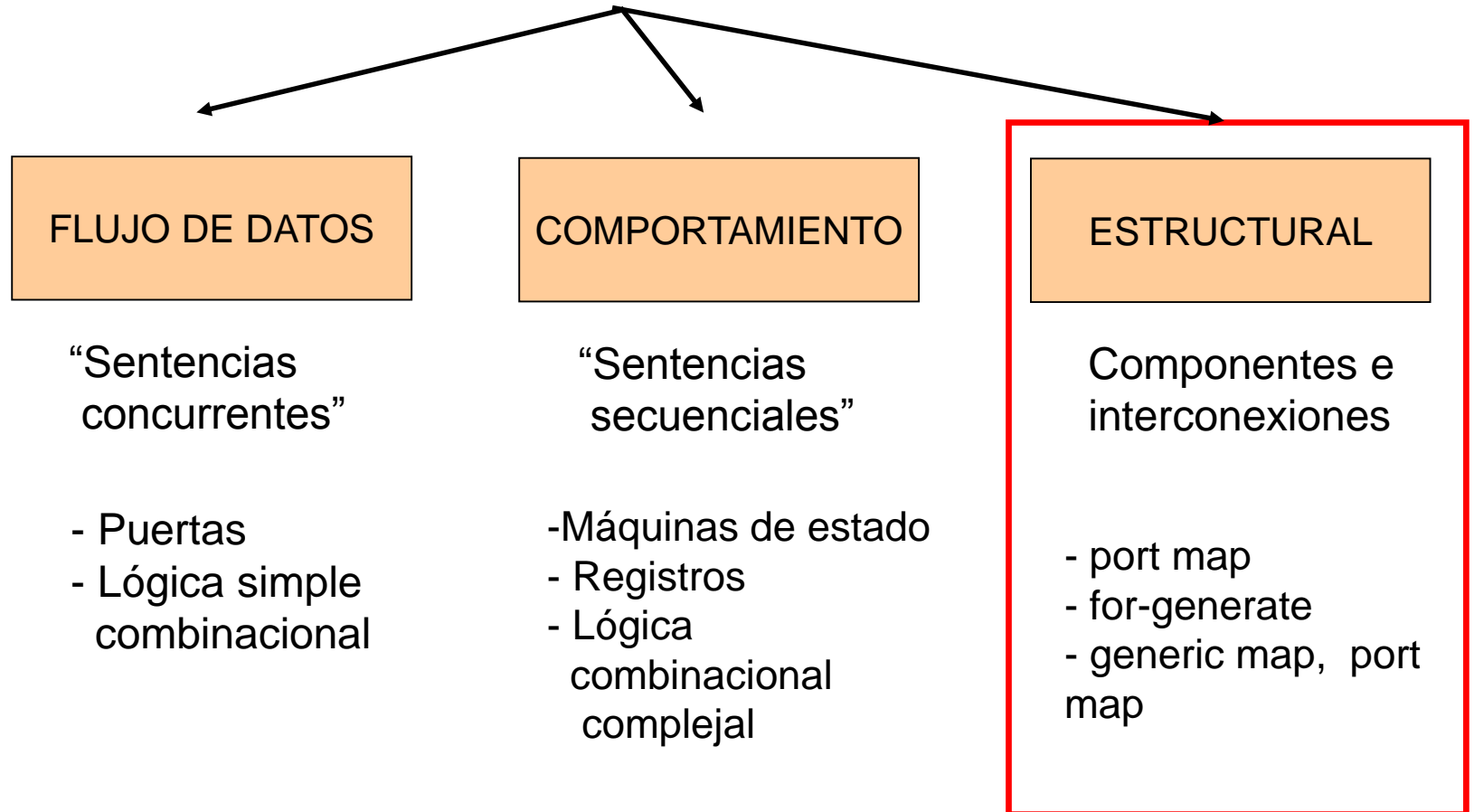
⊙ Para detectar flancos de señales tipo **STD_LOGIC_VECTOR** es mejor utilizar las funciones **rising_edge (S)** y **falling_edge (S)**.



MODELOS DESCRIPTIVOS

Modelado
de
sistemas II

Estilos de diseño VHDL





- ⊙ Un **COMPONENT** es una instanciación de una entidad (junto con una arquitectura)
- ⊙ Permite al diseñador reutilizar piezas comunes de códigos (permite diseño jerárquico)
- ⊙ Dos formas de declarar componentes:
 - ⊙ Declaración de componentes explícitos. (Los componentes se declaran en el código principal, en la parte declarativa de la arquitectura)
 - ⊙ Paquetes de declaración de componentes. (los componentes se declaran en un paquete)
- ⊙ La declaración de componentes le dice al compilador los puertos del componente que se van a instanciar.





COMPONENTE. DECLARACIÓN

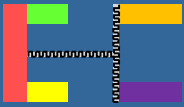
Modelado
de
sistemas II

⦿ Declaración:

```
COMPONENT nombre_componente IS
```

```
    PORT (locales);
```

```
END COMPONENT;
```



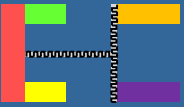
COMPONENTE. LLAMADA

- ⊙ Dos tipos de llamadas:

Etiqueta: nombre_componente **PORT MAP** (lista_puertos);

Etiqueta: nombre_componente **GENERIC MAP** (lista_parámetros)
PORT MAP (lista_puertos);

- ⊙ Mediante **lista_parámetros** se pasan parámetros genéricos al componente.
- ⊙ La conexión de los puertos puede ser:
 - ⊙ Posicional;
 - ⊙ Por asignación; ***nombre_puerto_comp => señal_o_puerto_asig***



EJEMPLO DECLARACIÓN Y LLAMADA

Modelado
de
sistemas II

COMPONENT dec2a4

PORT (w : **IN** STD_LOGIC_VECTOR(1 DOWNT0 0);

En : **IN** STD_LOGIC;

y : **OUT** STD_LOGIC_VECTOR (0 TO 3);

END COMPONENT;

U4: dec2a4 **PORT MAP** (w => q,

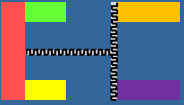
En => ena,

y => z);

← **Conectividad por
asignación**

U4: dec2a4 **PORT MAP** (q, ena, z);

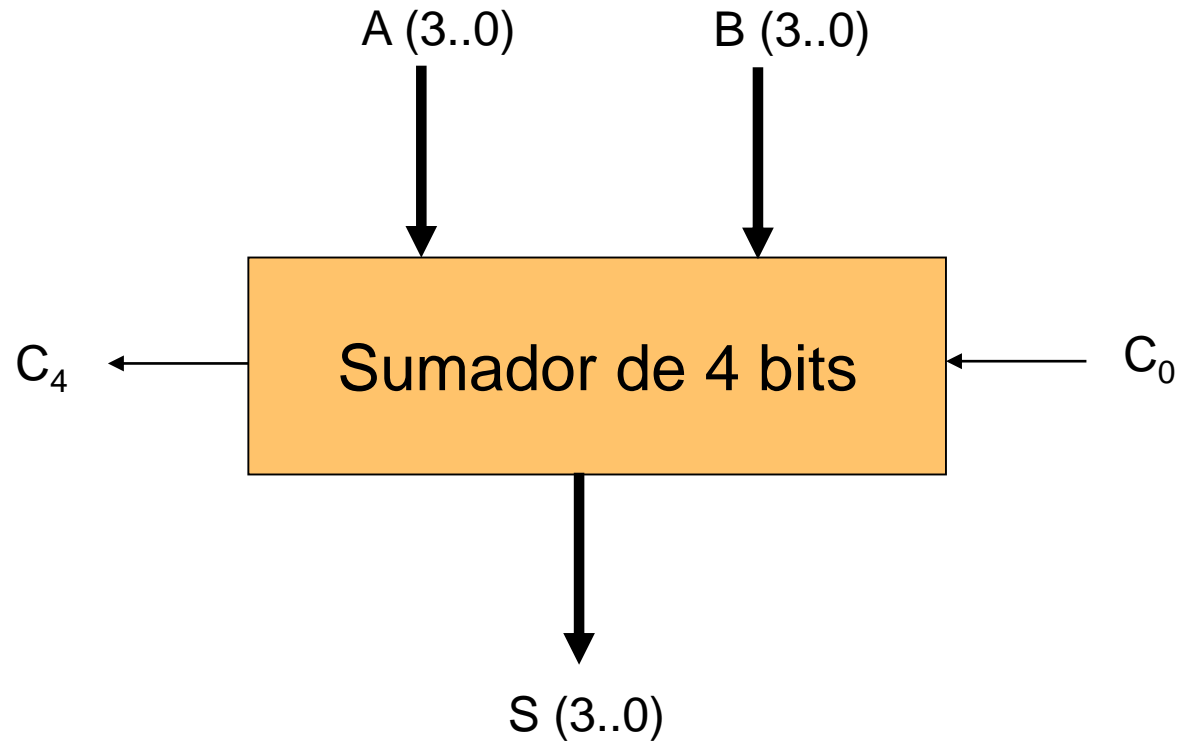
← **Conectividad
posicional**

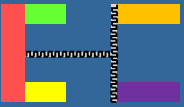


EJEMPLO PRÁCTICO. SUMADOR 4 BITS

Propuesta
de ejercicio
práctico

- Se propone describir y simular en Xilinx ISE el sumador binario con propagación de acarreo de 4 bits de la figura.





EJEMPLO PRÁCTICO. SUMADOR 4 BITS

Propuesta de ejercicio práctico

🎯 Descripción de la entidad

-- Sum4bits_ent.vhd

LIBRARY ieee;

USE ieee.std_logic_1164.**ALL**;

ENTITY sumador_4bits **IS**

PORT (A : **IN** STD_LOGIC_VECTOR (3 **DOWNTO** 0);

 B : **IN** STD_LOGIC_VECTOR (3 **DOWNTO** 0);

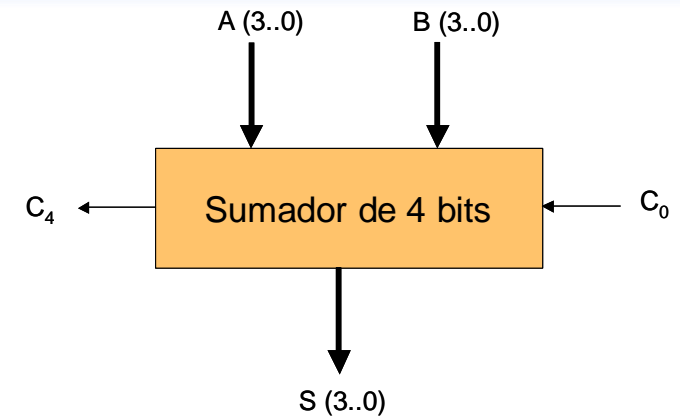
 C0: **IN** STD_LOGIC;

 C4: **OUT** STD_LOGIC;

 S : **OUT** STD_LOGIC_VECTOR (3 **DOWNTO** 0)

);

END sumador_4bits;

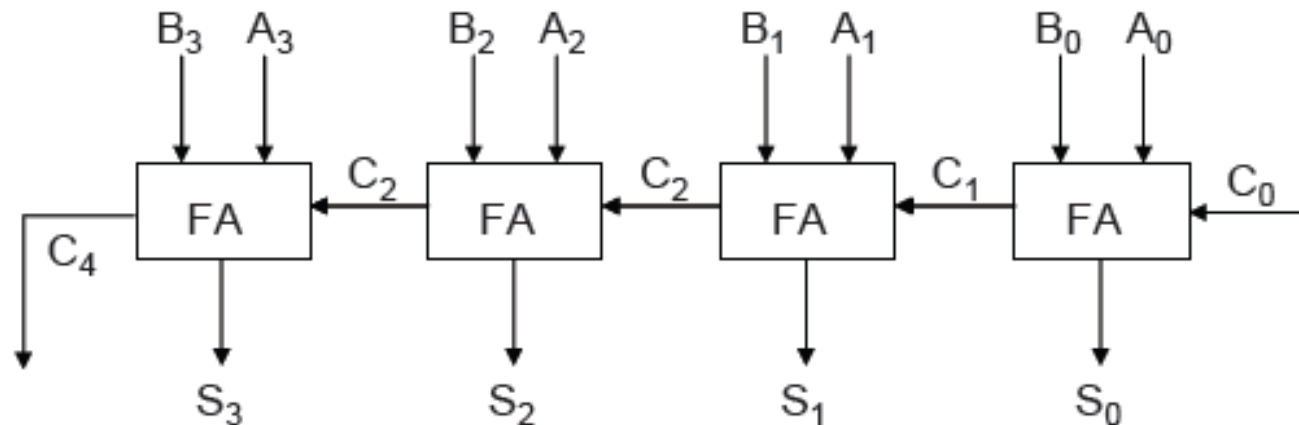


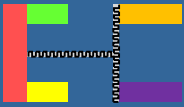


EJEMPLO PRÁCTICO. SUMADOR 4 BITS

Propuesta
de ejercicio
práctico

- El sumador con propagación de acarreo estará construido a partir de sumadores completos





EJEMPLO PRÁCTICO. SUMADOR 4 BITS

Propuesta de ejercicio práctico

- Se partirá de la descripción ya realizada y simulada del sumador completo.

-- Sum4bits_arc1.vhd

ARCHITECTURE circuito **OF** sumador_4bits **IS**

COMPONENT Sumador_completo

PORT (X , Y, Cin : **IN** STD_LOGIC;
Sum, Cout: **OUT** STD_LOGIC);

END COMPONENT;

SIGNAL C1, C2, C3: STD_LOGIC;

BEGIN

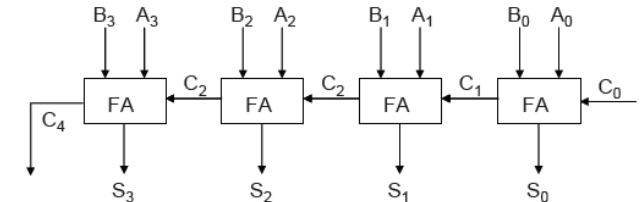
Sum1: Sumador_completo **PORT MAP** (A(0), B(0), C0, S(0), C1);

Sum2: Sumador_completo **PORT MAP** (A(1), B(1), C1, S(1), C2);

Sum3: Sumador_completo **PORT MAP** (A(2), B(2), C2, S(2), C3);

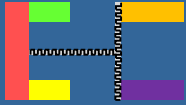
Sum4: Sumador_completo **PORT MAP** (A(3), B(3), C3, S(3), C4);

END circuito;



Declaramos el component en la zona de declaración de arquitectura

Señales de interconexión de los sumadores completos



TEST DEL SUMADOR 4 BITS

Propuesta
de ejercicio
práctico

```
-- Sum4bits_tb.vhd
```

```
LIBRARY ieee;
```

```
USE ieee.STD_LOGIC_1164.ALL;
```

```
ENTITY test_sumador_4 IS END test_sumador_4;
```

```
ARCHITECTURE test_flujo OF test_sumador_4 IS
```

```
-- PARTE DECLARATIVA
```

```
-- Declaración de componente que se va a testear
```

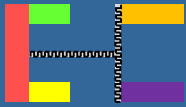
```
COMPONENT Sumador_4bits PORT ( A : IN STD_LOGIC_VECTOR (3 DOWNTO 0);  
                                B : IN STD_LOGIC_VECTOR (3 DOWNTO 0);  
                                C0 : IN STD_LOGIC;  
                                C4 : OUT STD_LOGIC;  
                                S : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
```

```
END COMPONENT;
```

```
-- Configuración de la arquitectura. NO OBLIGATORIO
```

```
FOR C1: Sumador_4bits USE ENTITY WORK.Sumador_4bits;
```





TEST DEL SUMADOR 4 BITS

Propuesta de ejercicio práctico

-- Declaración de las señales

```
SIGNAL Dato_A, Dato_B : STD_LOGIC_VECTOR (3 DOWNTO 0);  
SIGNAL Cin : STD_LOGIC;
```

-- CUERPO DE LA ARQUITECTURA

BEGIN

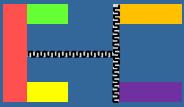
-- Instanciación del componente a testear y conexión de puertos

```
C1: Sumador_4bits PORT MAP ( A  => Dato_A,  
                             B  => Dato_B,  
                             C0 => Cin);
```

-- Valor de las señales de entrada

```
Cin  <= '0', '1' AFTER 20 ns, '0' AFTER 40 ns, '1' AFTER 60 ns, '0' AFTER 80 ns;  
Dato_A <= "0000", "1111" AFTER 10 ns, "1101" AFTER 20 ns, "1100" AFTER 30 ns,  
         "0111" AFTER 40 ns, "0101" AFTER 50 ns;  
Dato_B <= "1001", "0110" AFTER 10 ns, "0111" AFTER 20 ns, "0100" AFTER 30 ns,  
         "1000" AFTER 40ns, "1001" AFTER 50ns;
```

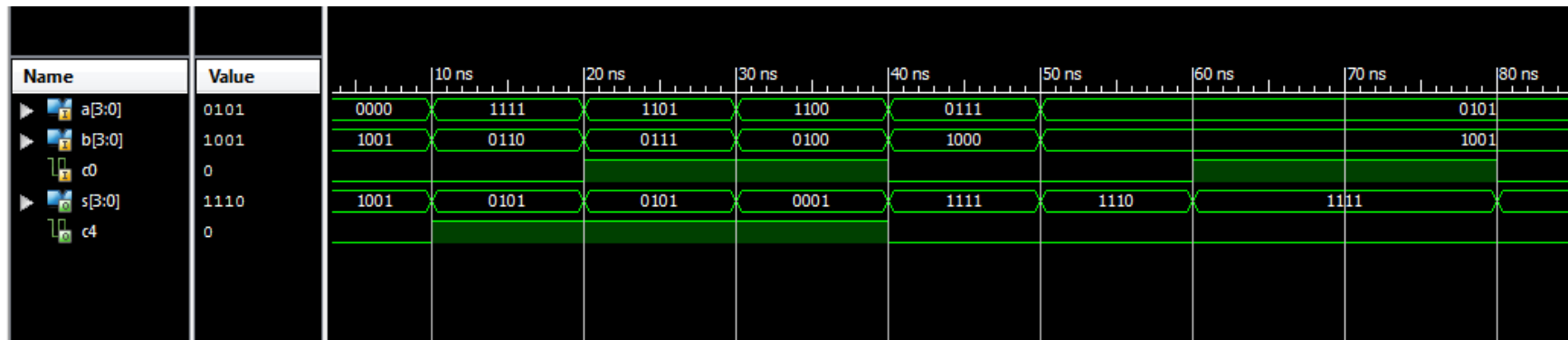
END test_flujo;



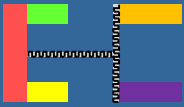
SIMULACIÓN DEL SUMADOR 4 BITS

Propuesta de ejercicio práctico

- El resultado de la simulación lo obtendremos mediante ISim:
- Desplazando el cursor por la ventana verificaremos el resultado correcto de la simulación.



- En la ventana izquierda de la pantalla de simulación tenemos los elementos que componen nuestro circuito.
- Si nos situamos sobre cualquiera de ellos, mediante el botón derecho del ratón podremos añadir a la nuestra simulación las formas de onda de las entradas y salidas que forman parte de él (*Add wave to window*).



SUMADOR 4 BITS MODIFICADO

Propuesta
de ejercicio
práctico

- Se puede simplificar el código del sumador utilizando la sentencia FOR-GENERATE.

-- sum4bits_arc2.vhd

ARCHITECTURE circuito **OF** sumador_4bits **IS**

COMPONENT Sumador_completo

PORT (X , Y, Cin : **IN** STD_LOGIC;

Sum, Cout: **OUT** STD_LOGIC);

END COMPONENT;

SIGNAL C: STD_LOGIC_VECTOR (4 **DOWNTO** 0);

BEGIN

C(0) <= C0;

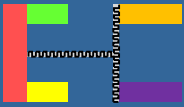
Sum: **FOR** i **IN** 0 **TO** 3 **GENERATE**

Sum1: Sumador_completo **PORT MAP** (A(i), B(i), C(i), S(i), C(i+1));

END GENERATE;

C4 <= C(4);

END circuito;



FUNDAMENTOS DEL VHDL

Elementos del lenguaje

⊙ Invariancias del VHDL:

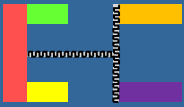
- ⊙ Invariante a mayúsculas, es decir, dos expresiones iguales conteniendo mayúsculas y minúsculas son idénticas.

Salida <= A and B; \equiv SALIDa <= a AND b;

- ⊙ Invariante a los espacios, es decir, dos expresiones iguales conteniendo más o menos espacios son idénticas.

Salida <= A and B; \equiv Salida <= A and B;

- ⊙ Los comentarios van detrás de dos rayas “- -” y conviene que sean claros para que las descripciones puedan ser fácilmente utilizadas por otras personas o por ti mismo.



FUNDAMENTOS DEL VHDL

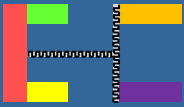
Elementos del lenguaje

- ⊙ VHDL es relativamente laxo con la utilización de paréntesis, una buena idea es utilizar los paréntesis de manera que una persona la pueda entender con facilidad.

```
IF x='0' AND y='0' OR z='1' THEN
....
END IF;
```

```
IF (((x='0') AND (y='0')) OR (z='1')) THEN
.....
END IF;
```

- ⊙ Cada asignación termina con “;”
- ⊙ Cada “**if**” tiene el correspondiente “**then**” y termina con el correspondiente “**end if**”. Si se necesita “**else if**” se utilizará “**elsif**”
- ⊙ Cada “**case**” termina con el correspondiente “**end case**”



FUNDAMENTOS DEL VHDL

Elementos del lenguaje

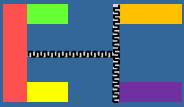
- ⦿ **Identificadores.** Son las palabras que se utilizan para identificar a las funciones, señales, puertos, variables, etc..
- ⦿ El identificador debe dar suficiente información para su uso .
- ⦿ El identificador puede ser tan largo como se quiera, pero un nombre demasiado largo es complicado de utilizar, y demasiado corto quizá proporcione poca información.
- ⦿ El identificador puede contener cualquier combinación de las letras (A-Z y a-z) números (0-9) y el sub-guión (“_”)
- ⦿ El identificador debe empezar por un carácter alfabético.
- ⦿ El identificador no puede termina con el sub-guión (“_”)





- ⊙ **Operadores y símbolos especiales.**
 - ⊙ Aritméticos: +, -, /, *
 - ⊙ Lógicos: not, and, or, nand, nor, xor.
 - ⊙ Relación: =, /=, <=, >=, <, >.
 - ⊙ Concatenación: &.
 - ⊙ Otros: (,), :, ;.
 - ⊙ Los comentarios comienzan con –
 - ⊙ Las sentencias terminan con ;.





FUNDAMENTOS DEL VHDL

Elementos del lenguaje

⊙ Constantes:

- ⊙ Mantienen el valor durante toda la ejecución. Se declaran antes del **BEGIN**.

Ejemplo: **CONSTANT** retardo :**TIME** := 15 ns;

⊙ Variables:

- ⊙ Solo pueden declararse y utilizarse dentro de los procesos.
- ⊙ La asignación se realiza con :=
- ⊙ Las asignaciones se realizan inmediatamente

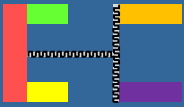
Ejemplo: **VARIABLE** estado: **BIT** := 0 ns; -- Declaración
 estado := 1; -- Asignación





- ⦿ **Señales:**
 - ⦿ Representan conexiones físicas.
 - ⦿ Se declaran en sentencias concurrentes y pueden aparecer también en procesos.
 - ⦿ Las asignaciones no son instantáneas, se actualizan en el siguiente paso de simulación.
 - ⦿ Se utilizan la comunicar procesos e interconectar componentes.
 - ⦿ Para asignar valores a una señal se utiliza `<=`





FUNDAMENTOS DEL VHDL

Elementos del lenguaje

- ⊙ **Datos.** VHDL tiene tipos predefinidos y permite definir nuevos tipos

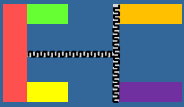
TYPE estado **IS** (inicio, arriba, abajo, stop);

- ⊙ Tipos básicos predefinidos (Tipos IEEE-1076)

```
TYPE BIT IS ('0', '1');  
TYPE BOOLEAN IS (FALSE, TRUE);  
TYPE TIME IS RANGE 0 TO 1E20  
    UNITS  
        Fs;  
        Ps=100fs;  
    END UNITS;
```

- ⊙ En el paquete IEEE_standard_logic1164:

```
TYPE STD_LOGIC is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```



FUNDAMENTOS DEL VHDL

Elementos del lenguaje

⊙ Datos compuestos.

- ⊙ Vectores: Objetos del mismo tipo ordenados por índices:

Ejemplo: **TYPE** dato_byte **IS ARRAY** (7 **DOWNTO** 0) **OF** BIT;

- ⊙ Registros

Ejemplo: **TYPE** Fecha **IS RECORD**

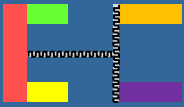
dia : INTEGER RANGE 1 TO 31;

mes: INTEGER RANGE 1 TO 12;

año: INTEGER RANGE 1900 TO 2025;

END RECORD;

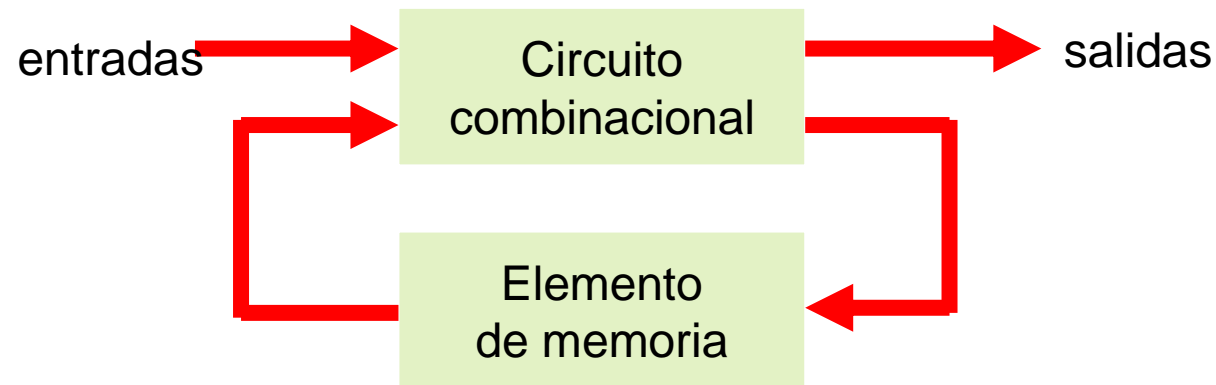
VARIABLE hoy, ayer: fecha;



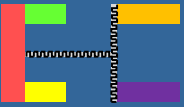
MODELADO SECUENCIAL

Modelado secuencial

- Los sistemas secuenciales son aquellos en los cuales la salida es función de las entradas y del estado actual en el que se encuentra.



- Básicamente existen dos tipos de sistemas secuenciales:
 - Síncronos.** Su comportamiento está sincronizado con el reloj del sistema.
 - Asíncronos.** Su funcionamiento depende del orden y momento en el que se aplican las señales de entrada.



BIESTABLE D

Modelado secuencial

- Biestable D activo por flanco de subida con señal reset asíncrona.

```
LIBRARY ieee;  
USE ieee.STD_LOGIC_1164.ALL;
```

```
ENTITY BiestableD IS  
    PORT ( D, CLK : IN std_logic;  
          rstH : IN std_logic;  
          Q : OUT std_logic);  
END BiestableD ;
```

```
ARCHITECTURE BiestableDarq OF BiestableD IS  
BEGIN
```

```
    PROCESS (CLK, rstH)  
    BEGIN
```

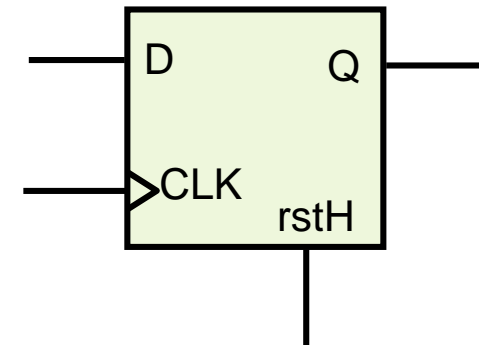
```
        IF (rstH = '1') THEN  
            Q <= '0';
```

```
        ELSIF (CLK'event and CLK = '1') THEN  
            Q <= D;
```

```
        END IF;
```

```
    END PROCESS;
```

```
END BiestableDarq;
```



Señal reset asíncrona

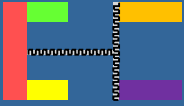
Biestable disparado
por flanco de subida



Práctica 1

Práctica a implementar

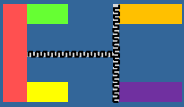
- ④ Realizar el siguiente trabajo:
 1. Implementar el circuito sumador de 4 bits descrito anteriormente. Ejecuta el test y comprueba que los resultados son los esperados.
 2. Implementar el biestable D anteriormente. Diseñar y ejecutar un test para que se comprueben todas las opciones del biestable.
 3. Implementar el siguiente circuito combinacional. Diseñar y ejecutar un test para que se comprueben todas las opciones del circuito.
 4. Implementar el siguiente circuito secuencial. Diseñar y ejecutar un test para que se comprueben todas las opciones del circuito.



Práctica a implementar

Práctica a implementar

- ④ La **práctica se entregará en un archivo zip que contendrá:**
 - ④ El archivo de la memoria documental con el siguiente formato:
 - Página 1: Nombre de la asignatura, título de la práctica, número de la práctica, nombre alumno, email alumno, D.N.I. del alumno, grupo teoría, fecha.
 - Página 2: índice de la práctica.
 - Página 3: listado de los archivos que entrega.
 - Página 4 y sucesivas el resto de la práctica (descripción de la práctica, qué hace, cómo lo hace, problemas surgidos, volcados de pantalla de la simulación, ...).
 - Las páginas deben tener número de página.
 - ④ Los archivos asociados a la implementación para que el/la profesor/a pueda ejecutarlos.

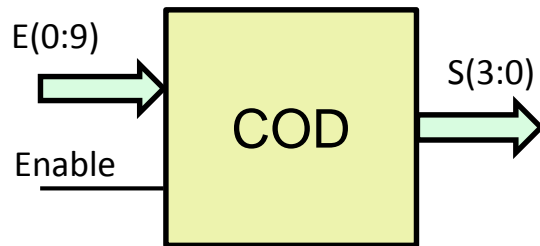


CIRCUITO CODIFICADOR PRIORITARIO

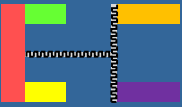
Práctica a implementar

CIRCUITO COMBINACIONAL

- ⦿ Diseña un circuito codificador prioritario de 10 a 4 líneas (codificador decimal). El circuito deberá tener las entradas activas a nivel bajo y proporcionar en todo momento la combinación de salida correspondiente a la entrada activada. En el caso de no estar ninguna de las entradas activadas, todas las salidas pasarán a *alta impedancia* (Z). Apoyándote en su tabla de verdad, escribe un testbench que nos permita comprobar su funcionamiento.



<i>Enable</i>	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	S3	S2	S1	S0
0	X	X	X	X	X	X	X	X	X	X	Z	Z	Z	Z
1	0	X	X	X	X	X	X	X	X	X	1	0	0	1
1	1	0	X	X	X	X	X	X	X	X	1	0	0	0
1	1	1	0	X	X	X	X	X	X	X	0	1	1	1
1	1	1	1	0	X	X	X	X	X	X	0	1	1	0
1	1	1	1	1	0	X	X	X	X	X	0	1	0	1
1	1	1	1	1	1	0	X	X	X	X	0	1	0	0
1	1	1	1	1	1	1	0	X	X	X	0	0	1	1
1	1	1	1	1	1	1	1	0	X	X	0	0	1	0
1	1	1	1	1	1	1	1	1	0	X	0	0	0	1
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	Z	Z	Z	Z



Práctica a implementar

CIRCUITO SECUENCIAL

- El código que tenemos a continuación se corresponde con el de un contador de 8 bits activo por flanco de subida y dotado de una señal de reset asíncrono. Incorpóralo a un proyecto y escribe un testbench que nos permita comprobar el correcto funcionamiento de dicho contador.

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.ALL;
USE ieee.STD_LOGIC_UNSIGNED.ALL;

ENTITY contador IS
    GENERIC (Nbits:INTEGER :=8);
    PORT (
        CLK : IN STD_LOGIC;
        rst  : IN STD_LOGIC;
        Q    : INOUT STD_LOGIC_VECTOR (Nbits-1 DOWNT0 0));
END contador ;

ARCHITECTURE ContaNBits OF contador IS
    BEGIN
        PROCESS (CLK, rst)
        BEGIN
            IF rst='1' THEN Q <= (OTHERS =>'0');
            ELSIF CLK='1' AND CLK'EVENT THEN Q <= Q + '1';
            END IF;
        END PROCESS;
    END ContaNBits;
```

