

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





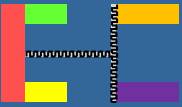
Introducción

- 🎯 VHDL es un lenguaje para describir hardware digital utilizado por la industria en todo el mundo.
- 🎯 **VHDL**: es un acrónimo de:

VHSIC **H**ardware **D**escription **L**anguage



VHSIC = **V**ery **H**igh **S**peed **I**ntegrated **C**ircuits



BREVE HISTORIA

Introducción

- ⦿ **1980:** El departamento de defensa de EEUU funda el proyecto para crear un HDL estándar dentro del programa VHSIC
- ⦿ **1981:** Woods Hole Workshop, reunión inicial entre el Gobierno, Universidades e Industria
- ⦿ **1983:** Se concedió a Intermetrics, IBM y Texas Instruments el contrato para desarrollar VHDL
- ⦿ **1985:** Versión 7.2 de dominio público
- ⦿ **1987:** El IEEE lo ratifica como su estándar 1076 (VHDL-87)
- ⦿ **1993:** El lenguaje VHDL fue revisado y ampliado, pasando a ser estándar 1076 '93 (VHDL-93)
- ⦿ **2000:** Última modificación de VHDL



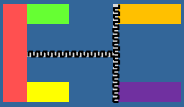
CARACTERÍSTICAS PRINCIPALES

Conceptos básicos

- ④ Utilización:
 - ④ Descripción hardware
 - ④ Simulación
 - ④ Síntesis

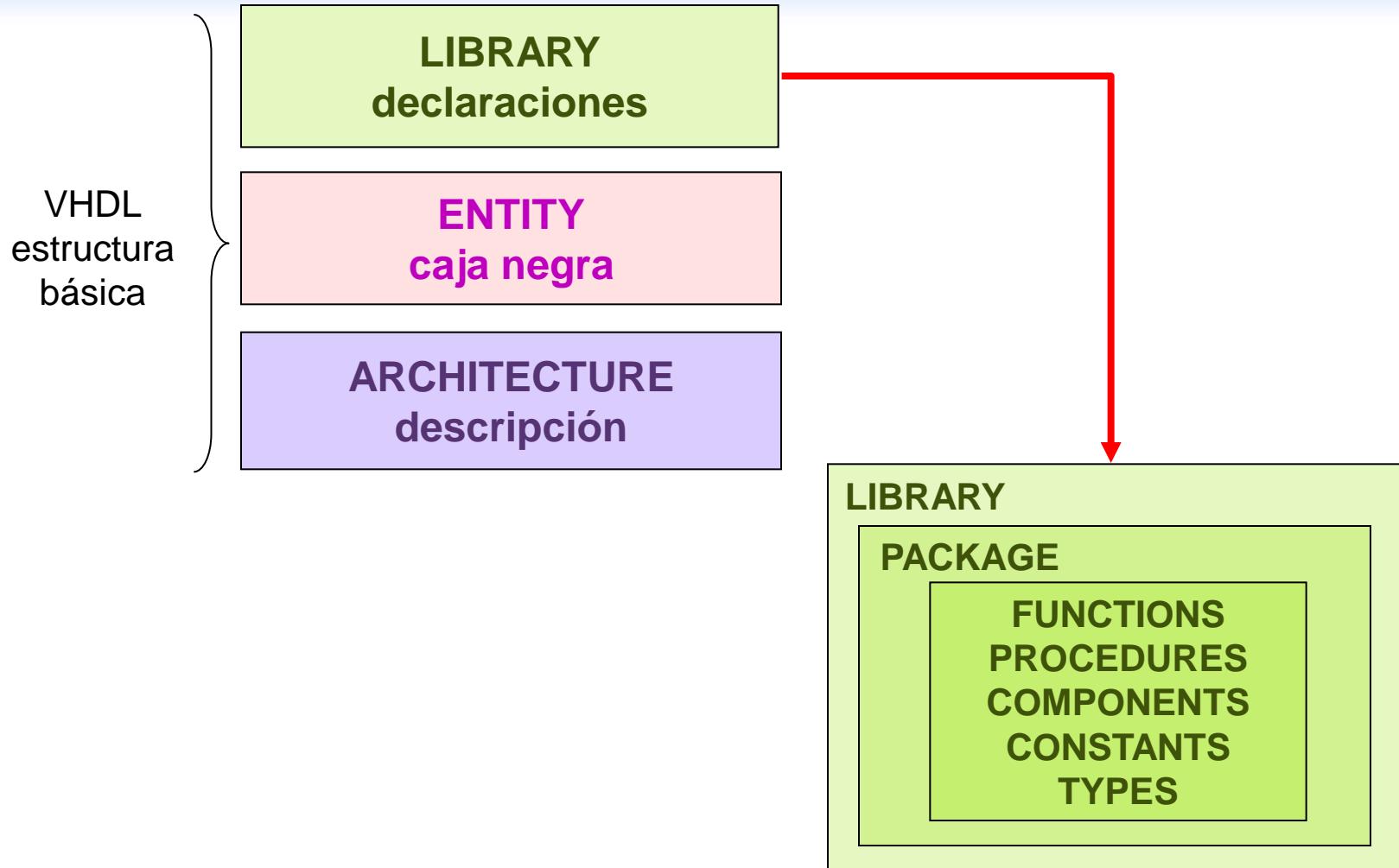
- ④ Beneficios prácticos:
 - ④ Es un mecanismo para el diseño digital y para la documentación de diseños reutilizables.
 - ④ Diseños reutilizables.

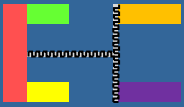




ESTRUCTURA DE UN DISEÑO EN VHDL

Conceptos
básicos





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

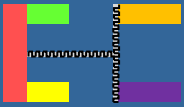
Cuerpo de la arquitectura

```
BEGIN
```

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

```
END Nombre_arquitectura;
```

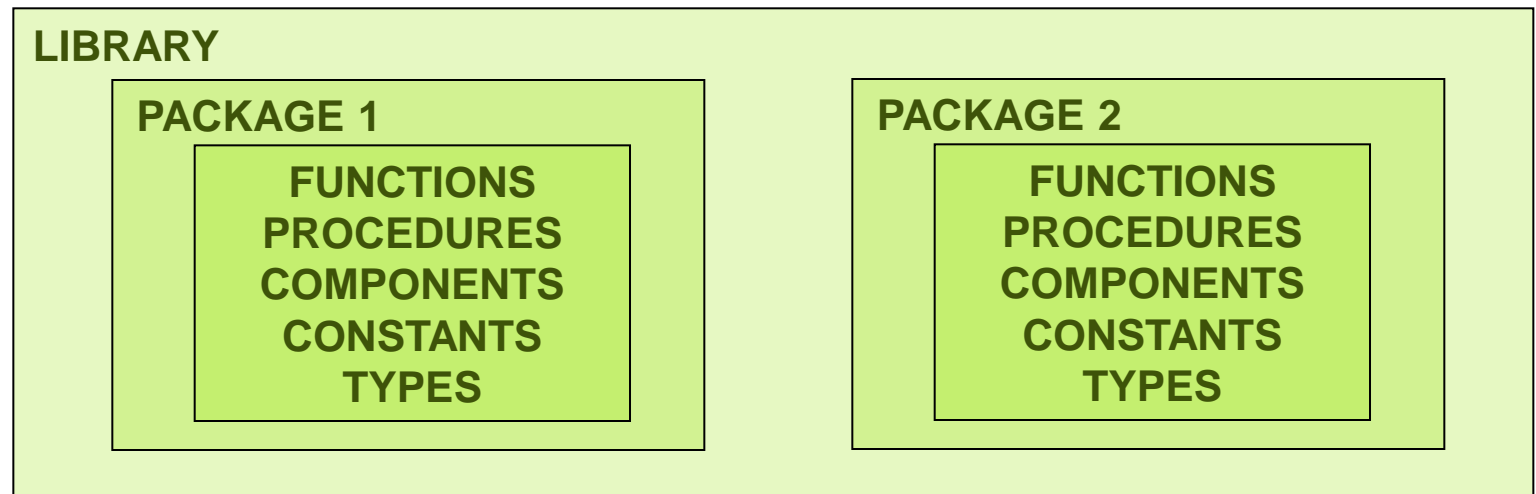
Nombre de la arquitectura



BIBLIOTECAS (LIBRARY)

Conceptos básicos

- Almacena los elementos de diseño (tipo de datos, operadores, componentes, objetos, funciones,...) organizados en Paquetes (Packages).
- Los Packages han de hacerse “visibles” para poder ser utilizados.





DECLARACIÓN DE LAS BIBLIOTECAS

Conceptos básicos

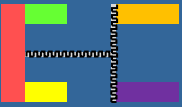
```
LIBRARY nombre_biblioteca;
```

```
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

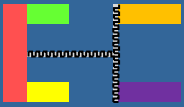
⊙ Std

- ⊙ 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

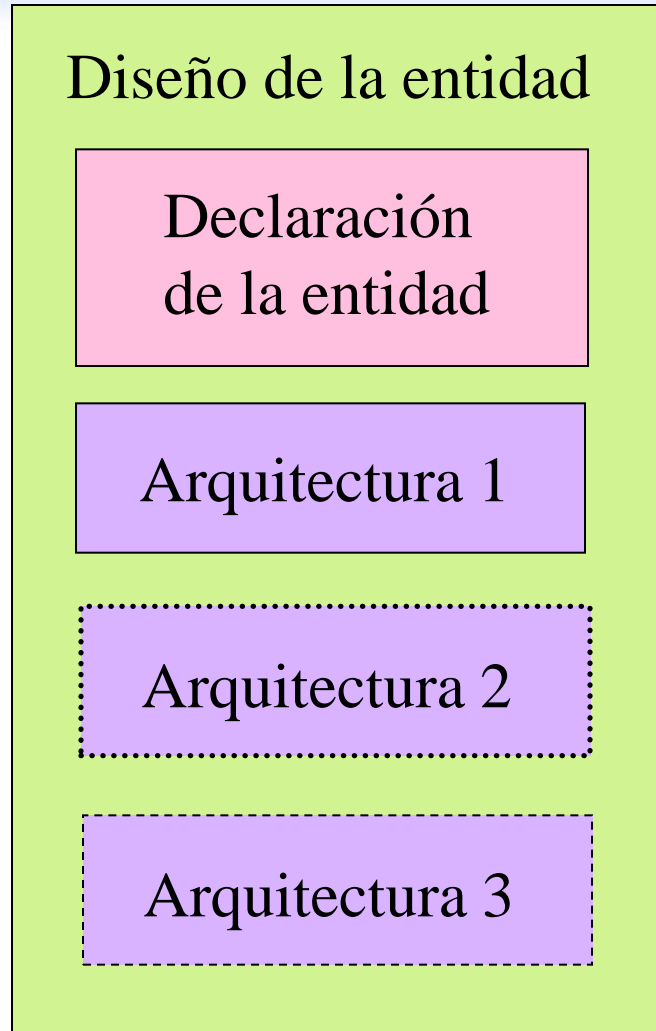
⊙ Work

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

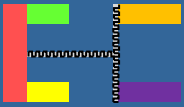


DISEÑO DE LA ENTIDAD

Unidades
básicas de
diseño



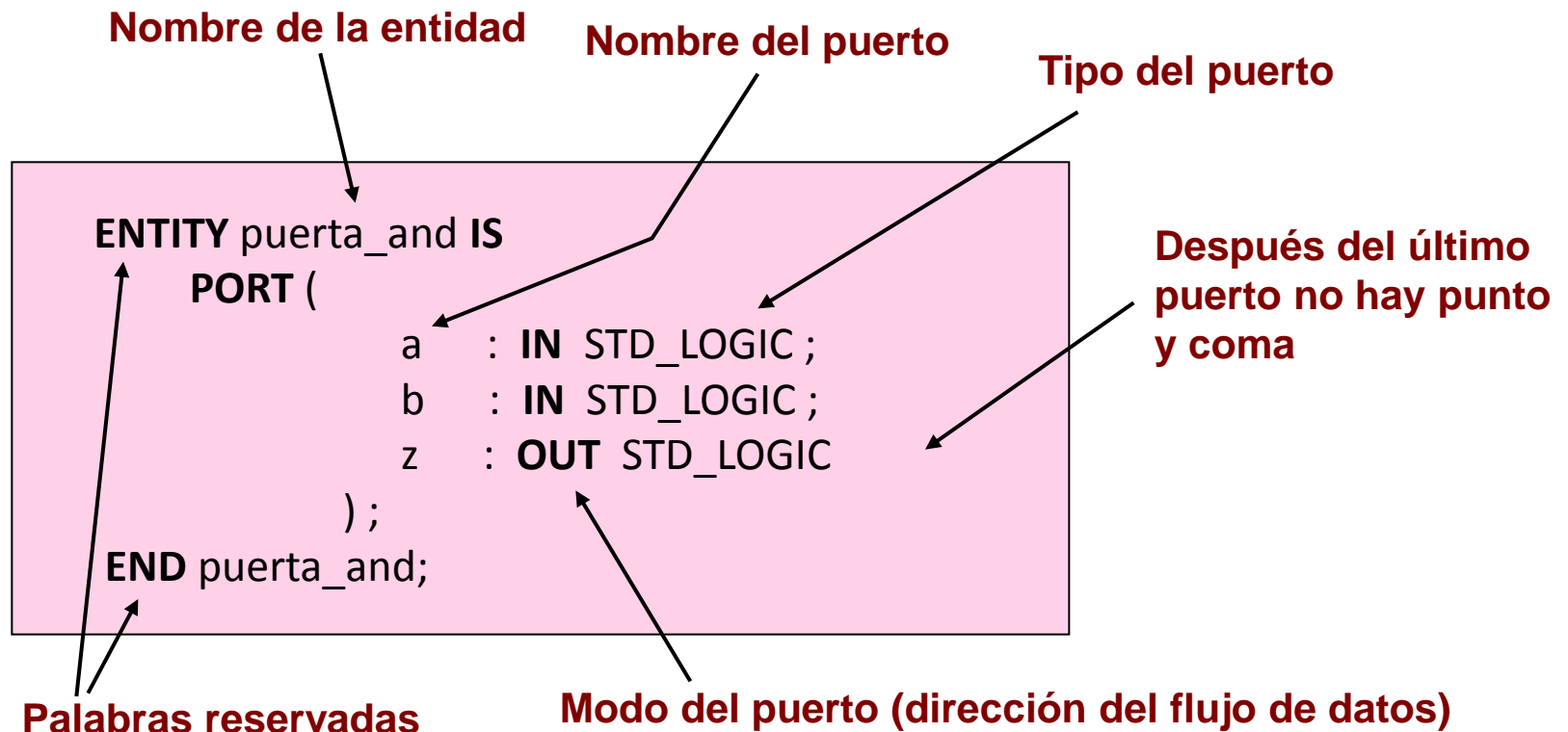
- ⊙ El diseño de la entidad es el bloque constructivo de diseño más básico
- ⊙ Una entidad puede tener varias arquitecturas distintas.

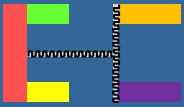


DECLARACIÓN DE LA ENTIDAD

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.





SINTAXIS SIMPLIFICADA

Unidades
básicas de
diseño

```
ENTITY Nombre_entidad IS
```

```
    [GENERIC(.....)];
```

```
    PORT (
```

```
        nombre_puerto : modo_puerto tipo_señal ;
```

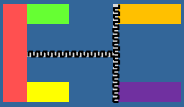
```
        nombre_puerto : modo_puerto tipo_señal;
```

```
        .....
```

```
        nombre_puerto : modo_puerto tipo_señal
```

```
    );
```

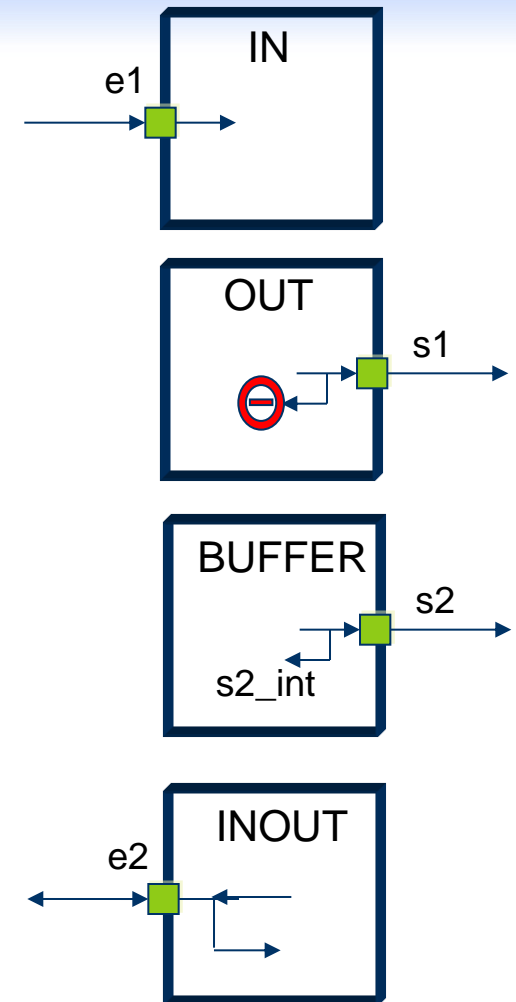
```
END Nombre_entidad;
```

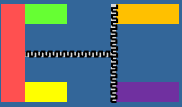


MODOS DE UN PUERTO

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.

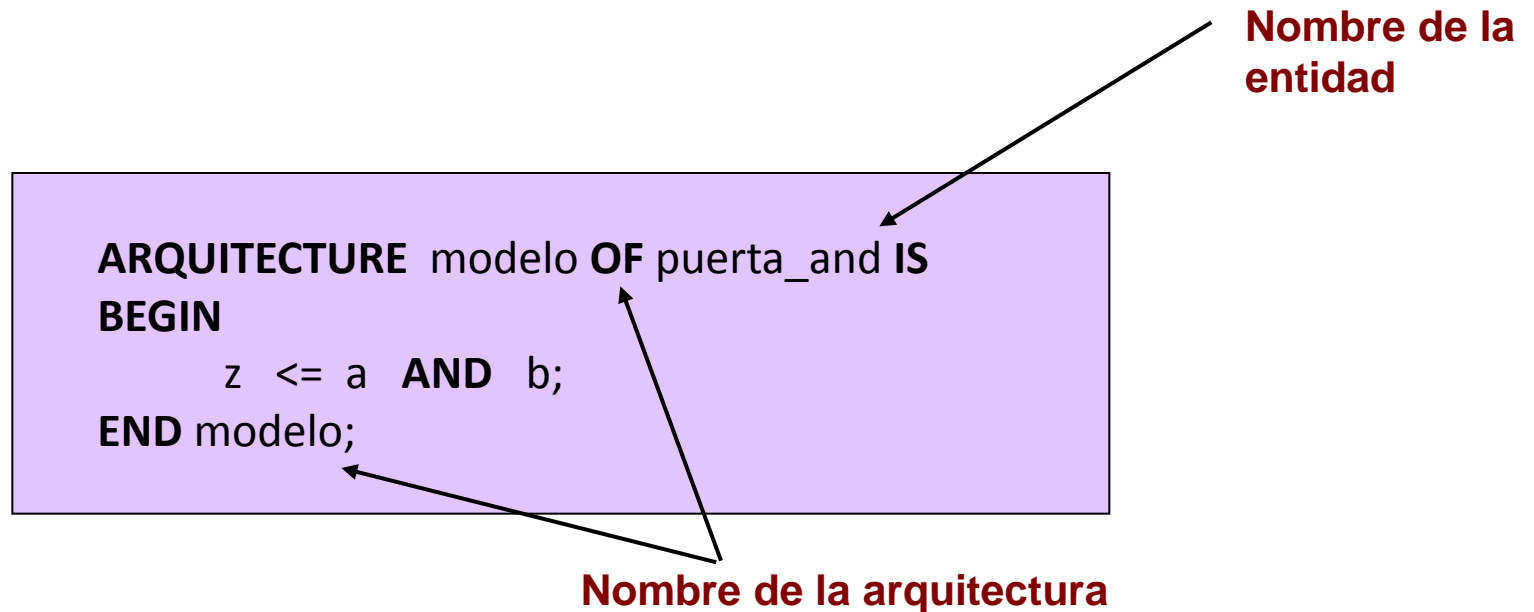


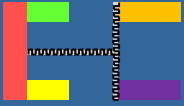


ARQUITECTURA

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).





SINTAXIS SIMPLIFICADA

Unidades
básicas de
diseño

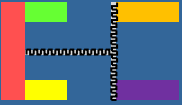
ARQUITECTURE nombre_arquitectura **OF** Nombre_entidad **IS**

[*declaraciones*]

BEGIN

código

END nombre_arquitectura;

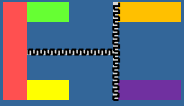


CARACTERÍSTICAS PRINCIPALES

Unidades básicas de diseño

- ⊙ Bibliotecas (library):
 - ⊙ Almacena los elementos de diseño: tipo de datos, operadores, componentes, objetos, funciones,...Estos elementos se organizan en Paquetes (Packages).
 - ⊙ Packages: son unidades de almacenamiento de elementos. Han de hacerse “visibles” para poder ser utilizados. (Bibliotecas std y work visibles por defecto)
- ⊙ Entidades (Entity)
 - ⊙ Es el modelo de interfaz de un circuito con el exterior mediante terminales de entrada y salida.
- ⊙ Architecture (Architecture)
 - ⊙ Describe el comportamiento del circuito.



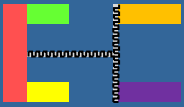


ESTILOS DESCRIPTIVOS

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.



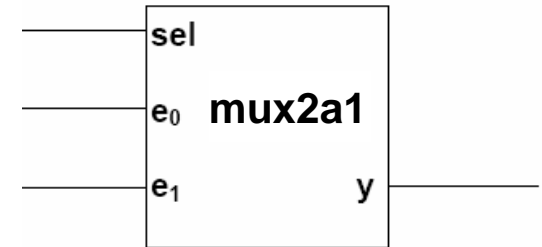


ESTILOS DESCRIPTIVOS - EJEMPLO

Modelado
de
sistemas I

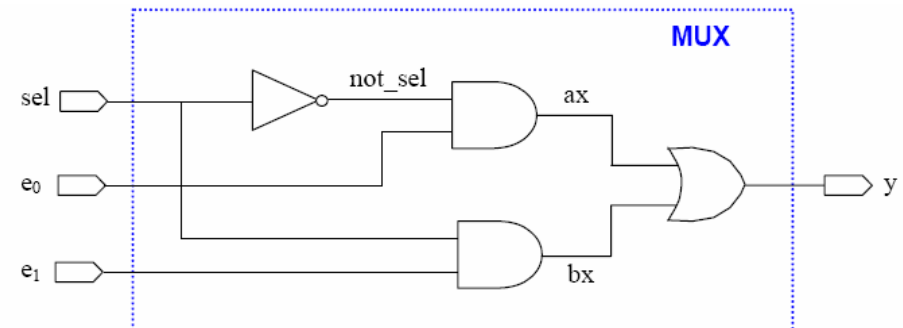
- Declaración de la entidad Multiplexor 2 a 1.

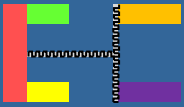
```
ENTITY mux2a1 IS  
  PORT ( e0, e1, sel : IN STD_LOGIC;  
          y           : OUT STD_LOGIC  
        );  
END mux2a1;
```



- 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;

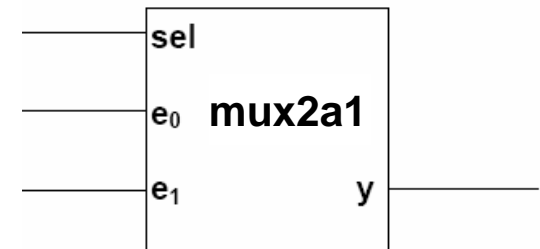
ELSE

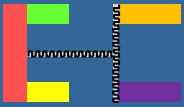
y<= e1;

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

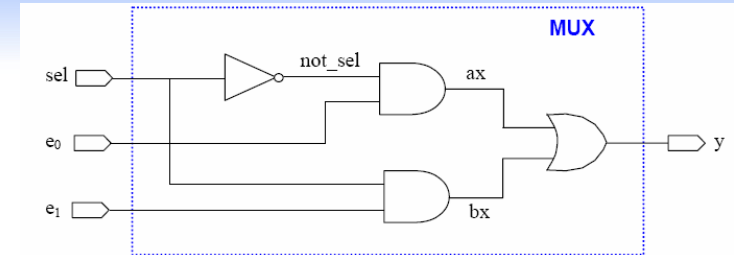
U0 : inv **PORT MAP** (e =>sel, sal=>not_sel);

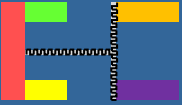
U1 : and2 **PORT MAP** (e1=>e0, e2=>not_sel, sal=>ax);

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

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

END estructural;

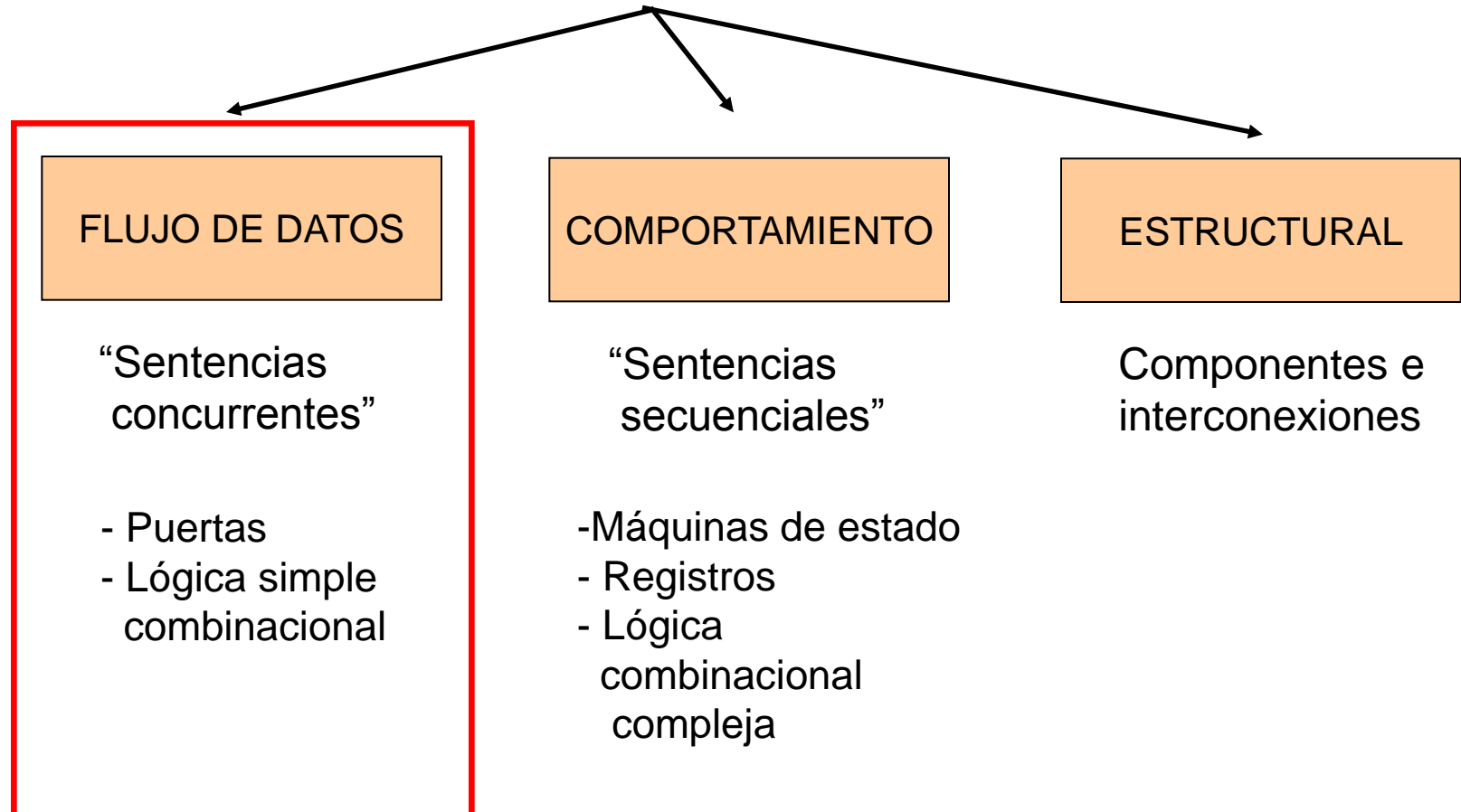


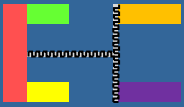


MODELOS DESCRIPTIVOS

Modelado
de
sistemas I

Estilos de diseño VHDL





DESCRIPCIÓN POR FLUJO DE DATOS

Modelado
de
sistemas I

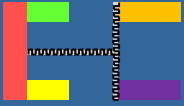
- Describe como se mueven los datos a través del sistema y de varios pasos de procesamiento.
- Utiliza sentencias concurrentes.
- Las sentencias concurrentes se evalúan al mismo tiempo; el orden de las sentencias no importan.
- Esto no es cierto para sentencias secuenciales (o de comportamiento)

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

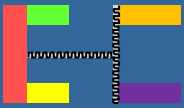
```
suma <= ope1 xor ope2;
```

- 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;
```

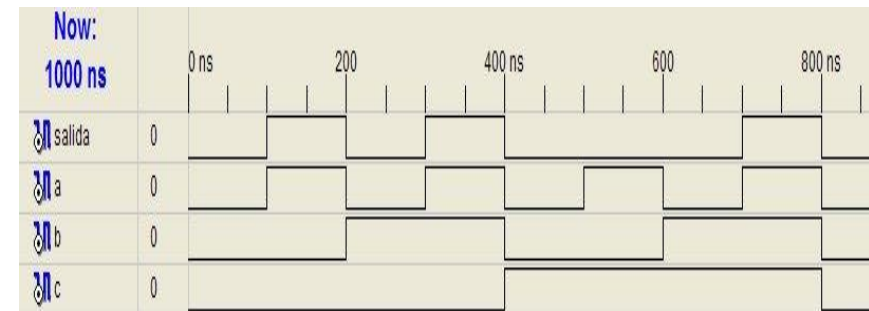
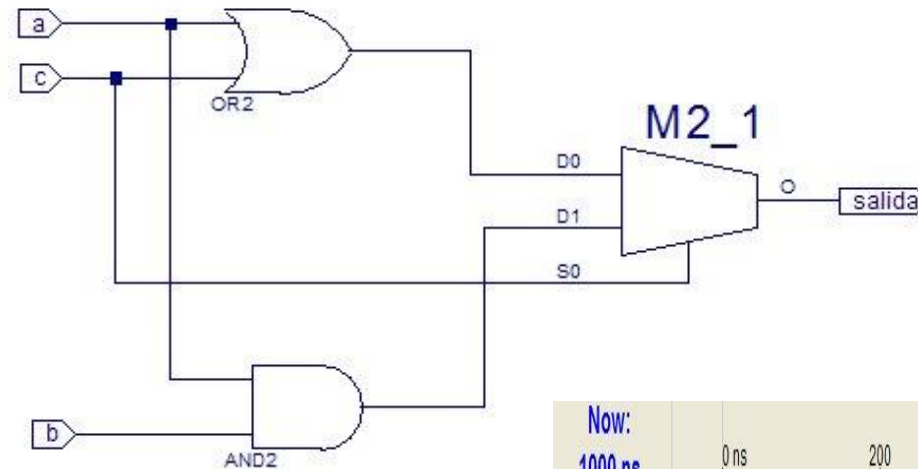



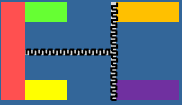
SENTENCIAS CONCURRENTES. WHEN-ELSE

Modelado
de
sistemas I

- Permite realizar asignaciones condicionales

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



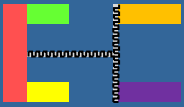


WHEN - ELSE

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
```

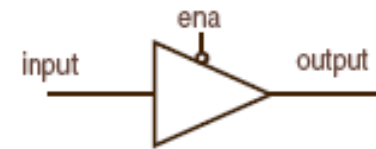
```
    PORT (   ena:   IN STD_LOGIC;  
            input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);  
            output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)  
        );
```

```
END tri_estado;
```

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

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

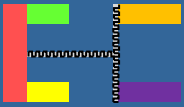
```
END tri_estado_flujo;
```



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



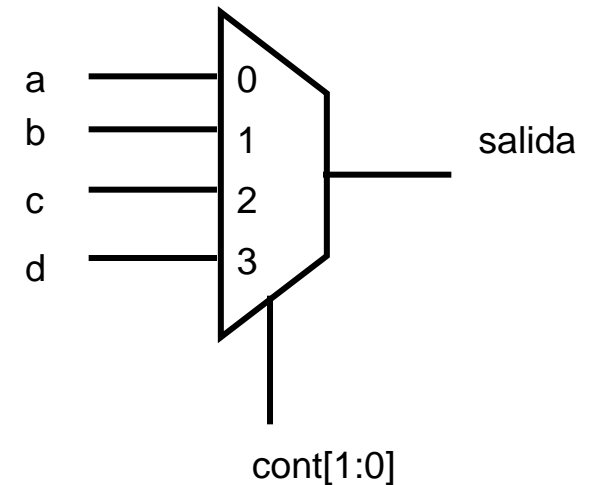
SENTENCIAS CONCURRENTES. WITH-SELECT

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

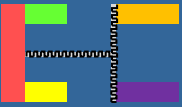


WITH - SELECT

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

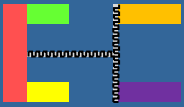
```
ARCHITECTURE con_with_select OF circuito IS  
BEGIN  
    WITH ( enable & A & B ) SELECT
```

```
    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**





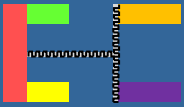
SENTENCIAS FOR-GENERATE

Modelado
de
sistemas I

- ⊙ Permite reducir el tamaño del código de aquellas estructuras que se repiten considerablemente.

```
[Etiqueta]: FOR identificador IN rango GENERATE  
            BEGIN  
                (Sentencias concurrentes)  
            END GENERATE [Etiqueta];
```

```
[Etiqueta]: IF condición GENERATE  
            BEGIN  
                (Sentencias concurrentes)  
            END GENERATE [Etiqueta];
```



EJEMPLO FOR-GENERATE

Modelado
de
sistemas I

....

SIGNAL x : BIT_VECTOR (7 DOWNT0 0);

SIGNAL y : BIT_VECTOR (15 DOWNT0 0);

SIGNAL z : BIT_VECTOR (7 DOWNT0 0);

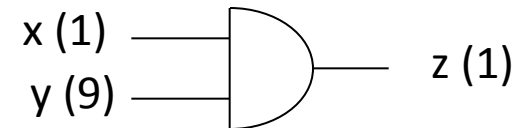
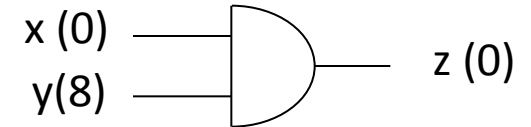
...

G1: FOR i **IN** x'RANGE **GENERATE**

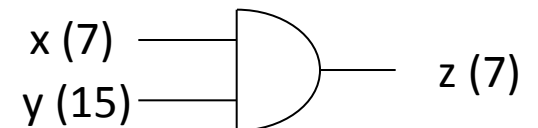
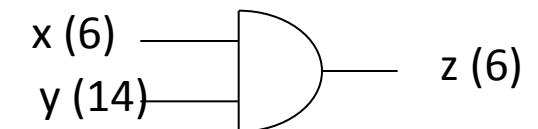
z(i) <= x(i) **AND** y(i+8);

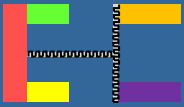
END GENERATE;

....



...





UN EJEMPLO SENCILLO

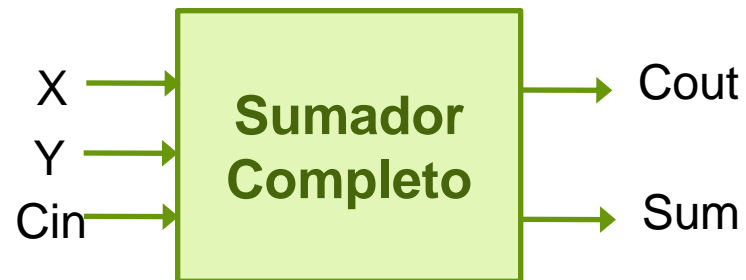
Simulación
de un
ejemplo
sencillo

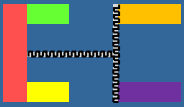
- Sumador completo mediante descripción de flujo de datos
- Entidad: define la interfaz del sistema

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

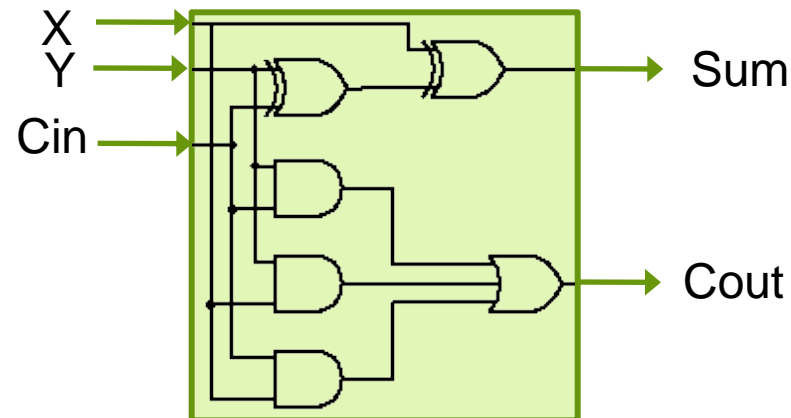
ARCHITECTURE Ecuaciones **OF** Sumador_completo **IS**

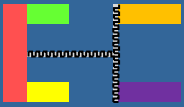
BEGIN - - Asignaciones concurrentes

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

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

END Ecuaciones;





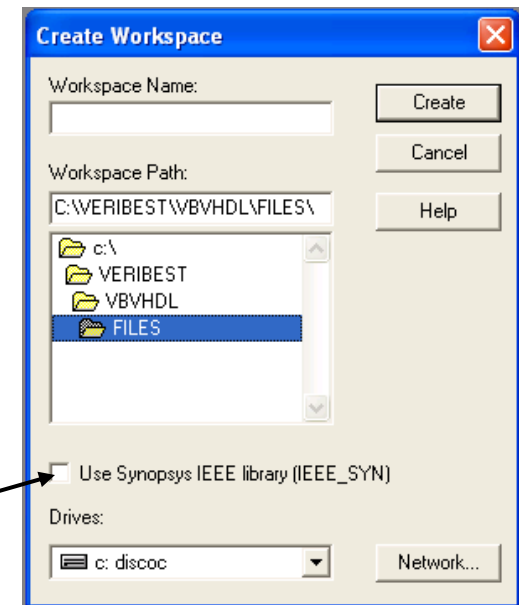
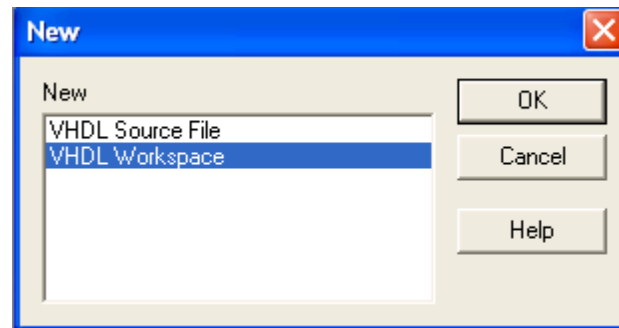
ARRANQUE DEL SIMULADOR

Simulación
de un
ejemplo
sencillo

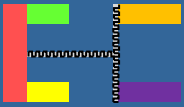
- Arranque del simulador Veribest. Desde el menú Inicio de Windows se accede a la aplicación Veribest VHDL Simulator:

Programas -> Veribest VB99.0 -> Veribest VHDL Simulator -> VeriBest VHDL

- Se comienza creando un espacio de trabajo desde el menú *File*. La herramienta sugiere ubicarlo en la carpeta *files*, aunque el usuario puede decidir cualquier otro sitio.



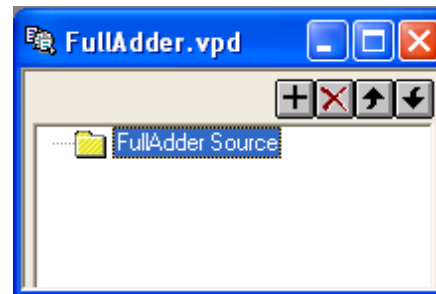
Se activa cuando se van a utilizar librerías IEEE (unsigned, ...)



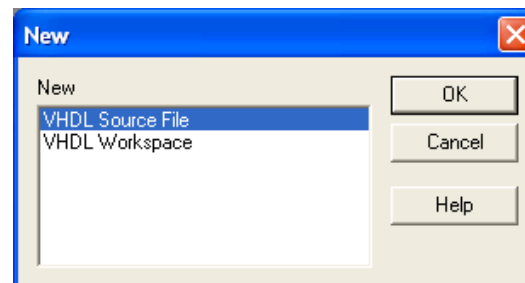
CREACIÓN DEL ESPACIO DE TRABAJO

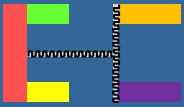
Simulación
de un
ejemplo
sencillo

- ⦿ Aparece una ventana con el nombre del espacio de trabajo que hemos creado y dentro una carpeta que es donde se van a guardar todos los ficheros que escribamos dentro del espacio de trabajo.



- ⦿ Se crea un archivo de VHDL para introducir el diseño desde el menú *File -> New*. En la ventana que aparece se selecciona *VHDL Source File*.





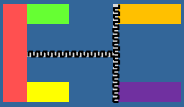
EDICIÓN DE UN FICHERO VHDL

Simulación
de un
ejemplo
sencillo

- ⦿ Aparece una ventana con una hoja vacía del editor en la que escribiremos el código VHDL. Lo salvaremos con el nombre que queramos y preferiblemente en el mismo lugar donde se haya creado el espacio de trabajo.

```
-----  
--  
-- Definición de la entidad para      --  
-- el sumador completo                --  
--  
-----  
  
LIBRARY ieee;  
USE ieee.STD_LOGIC_1164.all;  
  
ENTITY Sumador_completo IS  
    PORT (X, Y, Cin : IN STD_LOGIC;  
          Sum, Cout : OUT STD_LOGIC);  
  
END Sumador_completo;  
|  
  
"C:\VeriBest\vbvhdl\files\Sum_C_ent.vhd" saved.  Ln 18, Col 1  NUM
```

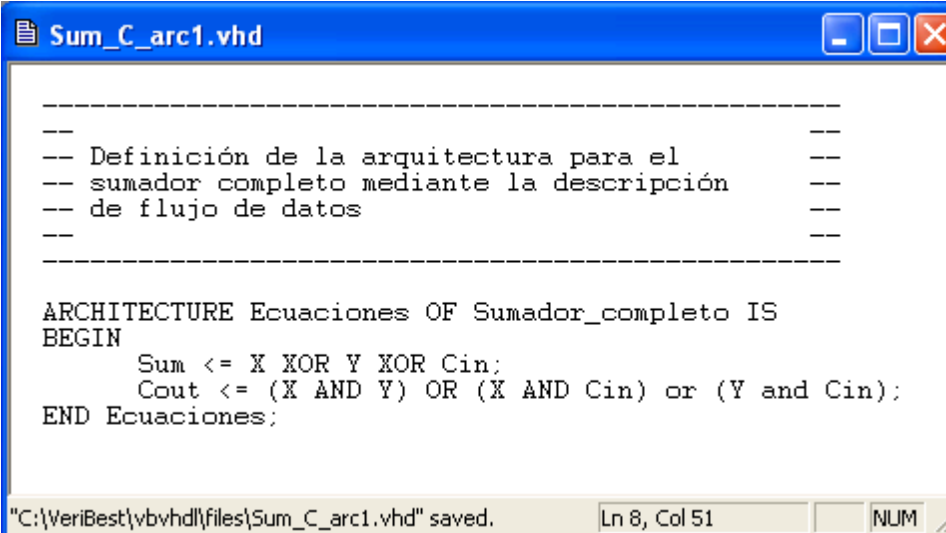
- ⦿ Nuestro diseño lo escribiremos en dos ficheros separados, uno para la entidad y otro para la arquitectura.



EDICIÓN DE UN FICHERO VHDL

Simulación
de un
ejemplo
sencillo

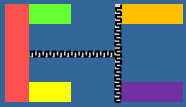
- Repetimos el proceso y creamos un nuevo fichero con la arquitectura del sumador completo.



```
-----  
--  
-- Definición de la arquitectura para el  
-- sumador completo mediante la descripción  
-- de flujo de datos  
--  
-----  
  
ARCHITECTURE Ecuaciones OF Sumador_completo IS  
BEGIN  
    Sum <= X XOR Y XOR Cin;  
    Cout <= (X AND Y) OR (X AND Cin) or (Y and Cin);  
END Ecuaciones;
```

"C:\VeriBest\vbvhdl\files\Sum_C_arc1.vhd" saved. Ln 8, Col 51 NUM

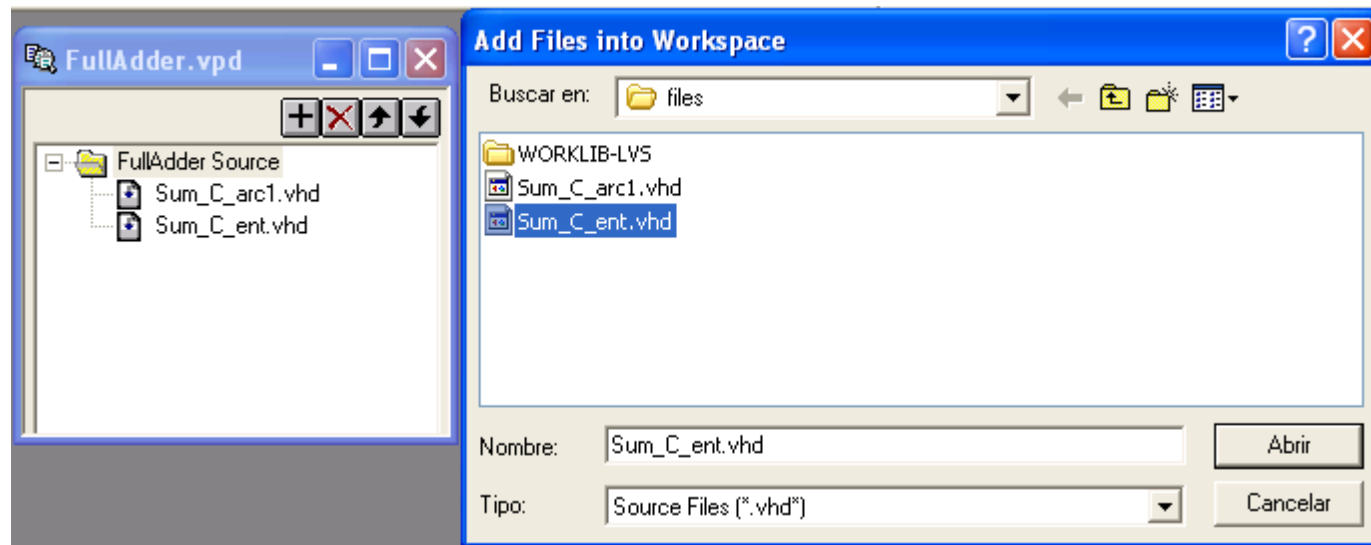
- Aunque el compilador no distingue entre mayúsculas y minúsculas, se ha optado por escribir las palabras reservadas en mayúsculas para una mejor visualización .

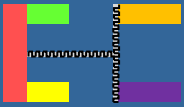


PREPARAR COMPILACIÓN

Simulación
de un
ejemplo
sencillo

- ⦿ A continuación incorporación los archivos VHDL que se desee compilar al espacio de trabajo. Para ello, tras activar la ventana del entorno de trabajo, pulsamos el botón **+** o bien ejecutamos la opción *Add Files into Workspace* del menú *Workspace*. Aparece una ventana donde seleccionaremos sucesivamente los ficheros deseados (nombres y rutas de 8 caracteres y sin espacios).

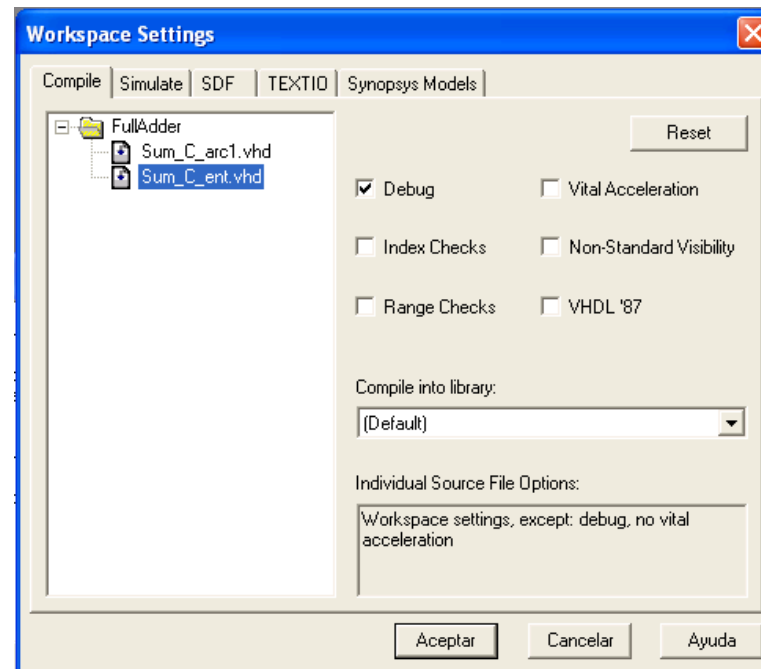


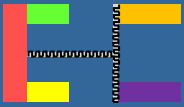


PREPARAR COMPILACIÓN

Simulación
de un
ejemplo
sencillo

- Establecemos las condiciones de compilación seleccionando la opción *Settings* dentro del menú *Workspace*. En la nueva ventana que se abre seleccionamos en el submenú *Compile* la opción *Debug* de todas las posibles para cada fichero tras abrir la carpeta que aparece.

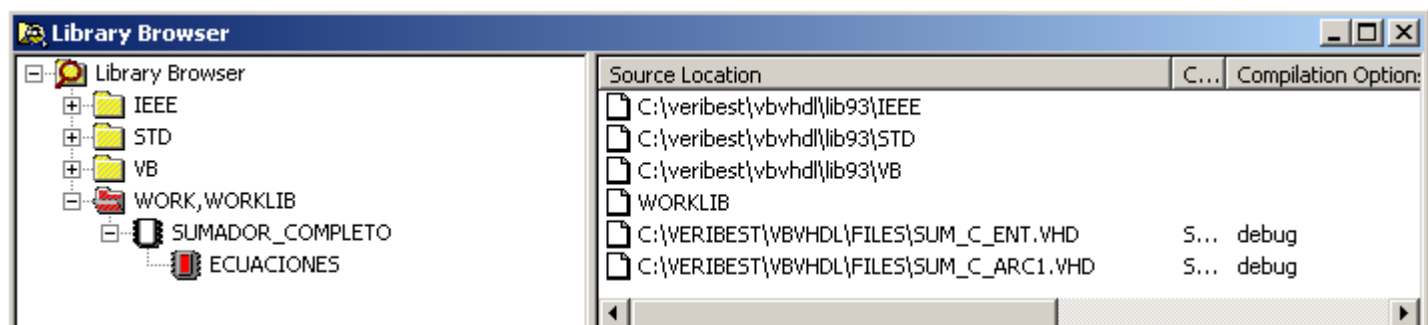


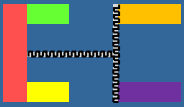


COMPILACIÓN

Simulación
de un
ejemplo
sencillo

- 🎯 Lanzamos la compilación de los archivos en un orden lógico de jerarquía (especificado por el orden en la lista o al compilar cada archivo por separado). Los mensajes aparecen en la ventana inferior (*General, Build, Simulate*):
 - 🎯 Desde el menú *Workspace* o seleccionando el fichero deseado y activando el botón (📁 o 📄).
 - 🎯 Las unidades compiladas y su estructura se puede ver abriendo el *Library Browser* (menú *Library*) y abriendo después la biblioteca *Work*.



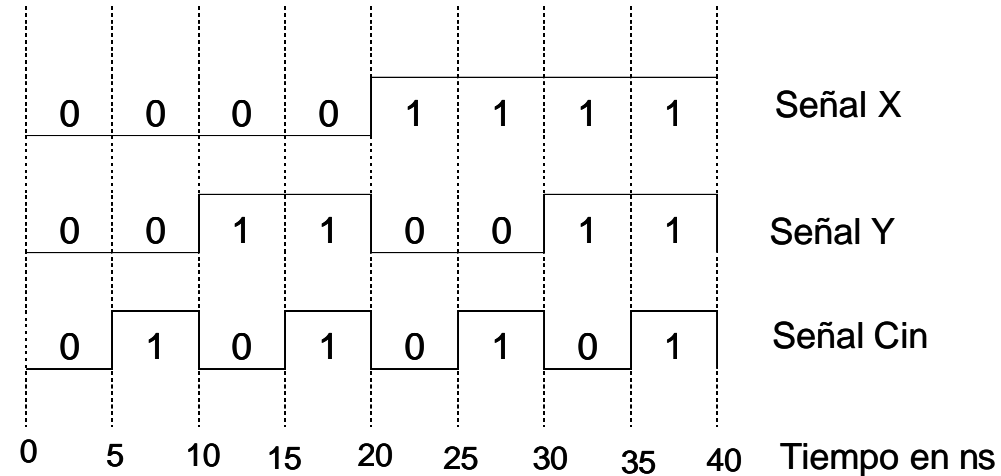


TEST-ESTÍMULOS DE ENTRADA

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



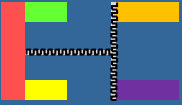


FICHERO DE TEST

Simulación
de un
ejemplo
sencillo

- ⦿ Para testear un componente es necesario escribir una entidad y una arquitectura.
- ⦿ Las entidades para test no tienen puertos.
- ⦿ Las arquitecturas son de tipo estructural. En la parte declarativa hay que incluir el componente a testear, la arquitectura y declaración de señales.
- ⦿ Una vez escrito se guardará el archivo con nombre `sumador_tb.vhd` y se procederá a compilarlo y guardarlo en el workspace como se ha explicado con anterioridad

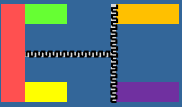




TEST DEL SUMADOR COMPLETO

Simulación
de un
ejemplo
sencillo

```
LIBRARY ieee;  
USE ieee.STD_LOGIC_1164.ALL;  
  
ENTITY test_sumador IS END test_sumador;  
  
ARCHITECTURE test_flujo OF test_sumador IS  
  
-- PARTE DECLARATIVA  
  
-- Declaración de componente que se va a testear  
  
    COMPONENT Sumador_completo PORT ( X, Y, Cin      : IN STD_LOGIC;  
                                         Sum, Cout   : OUT STD_LOGIC);  
  
    END COMPONENT;  
  
-- Configuración de la arquitectura. NO OBLIGATORIO  
-- Por defecto se tomaría la última arquitectura compilada  
  
    FOR C1: Sumador_completo USE ENTITY WORK.Sumador_completo;
```



TEST DEL SUMADOR COMPLETO

Simulación
de un
ejemplo
sencillo

-- Declaración de las señales

```
SIGNAL X, Y, Cin, SUM, Cout : STD_LOGIC;
```

-- CUERPO DE LA ARQUITECTURA

```
BEGIN
```

-- Instanciación del componente a testear y conexión de puertos

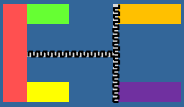
```
C1: Sumador_completo PORT MAP ( X  => X,  
                                   Y   => Y,  
                                   Cin => Cin,
```

```
SUM => SUM,  
Cout => Cout);
```

-- Valor de las señales de entrada

```
Cin <= '0', '1' AFTER 5 ns, '0' AFTER 10 ns, '1' AFTER 15 ns, '0' AFTER 20 ns,  
      '1' AFTER 25 ns, '0' AFTER 30 ns, '1' AFTER 35 ns;  
Y    <= '0', '1' AFTER 10 ns, '0' AFTER 20 ns, '1' AFTER 30 ns;  
X    <= '0', '1' AFTER 20 ns;
```

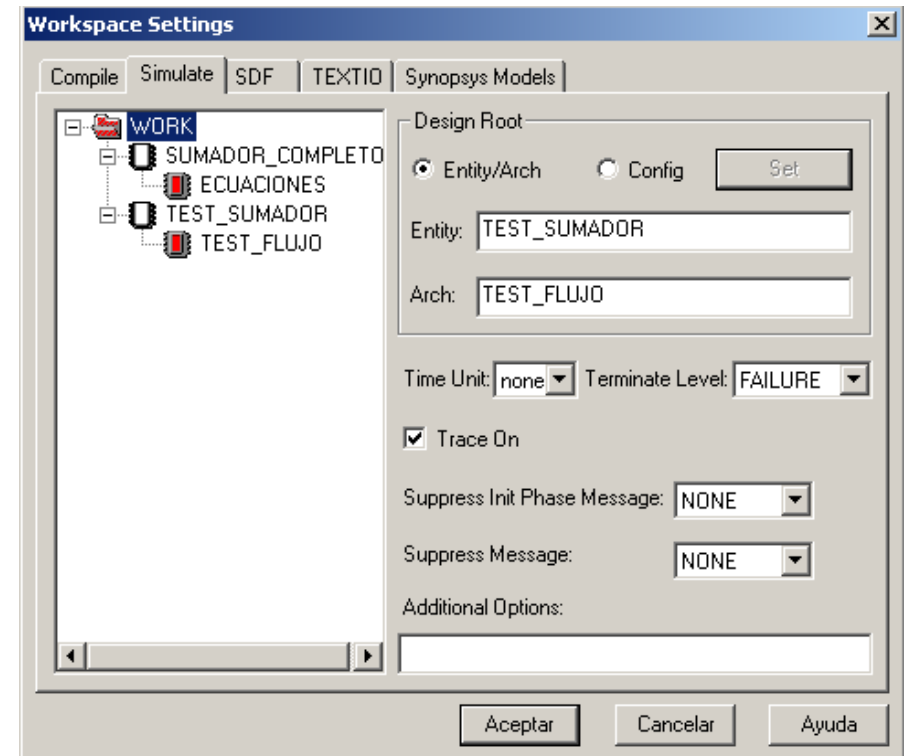
```
END test_flujo;
```

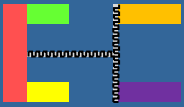


CONDICIONES DE SIMULACIÓN

Simulación
de un
ejemplo
sencillo

- Se selecciona la opción *Settings* del menú *Workspace*. En la ventana de diálogo que aparece se selecciona *Simulate* y se abre la carpeta *WORK*.
- Con la opción *SET* se asigna a las casillas *Entity* y *Arch* la entidad y arquitectura de la unidad de diseño que se quiere simular.
- Se activa la opción *Trace On* para visualizar las formas de onda resultantes en la simulación.

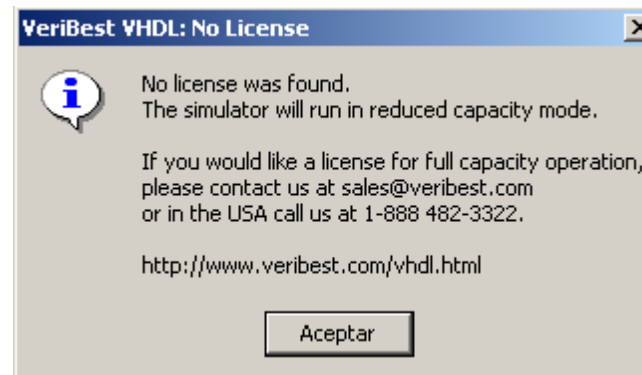


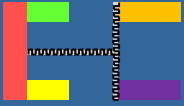


ACTIVACIÓN DEL SIMULADOR

Simulación
de un
ejemplo
sencillo



- Se activa la simulación con la opción *Execute Simulator* del menú *Workspace* o mediante el botón correspondiente en la barra de herramientas.
- Se acepta el mensaje que informa que no existe licencia para ejecutar código VHDL de más de un determinado número de líneas.

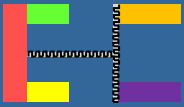




EJECUCIÓN DEL SIMULADOR



Simulación
de un
ejemplo
sencillo

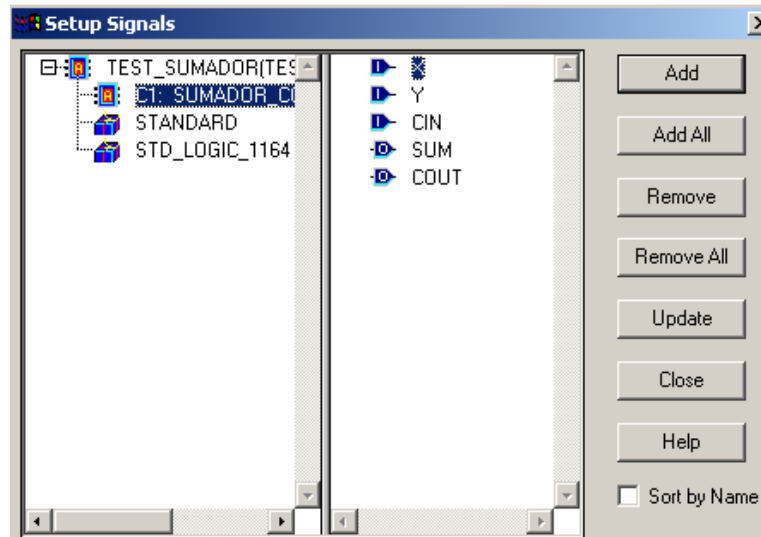
- ⦿ La ejecución del simulador se puede realizar desplegando el menú *Simulate* y activando *Run* o mediante el botón *Play* () de la barra de herramientas.
- ⦿ Si se desea desactivar el simulador se selecciona la opción *Quit* del menú *Simulate* o se pulsa sobre el botón de parada de la barra de herramientas ()

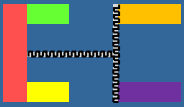


SEÑALES A VISUALIZAR

Simulación
de un
ejemplo
sencillo

- Las ondas resultantes se pueden visualizar seleccionando la opción *New Waveform Window* del menú *Tools* o pulsando el botón correspondiente de la barra de herramientas ().
- En la nueva ventana, al activar el botón  aparece la ventana de *Setup signals* con la arquitectura del test y sus señales asociadas. Se seleccionan las señales que se desean representar mediante el botón *Add* o *Add All*.

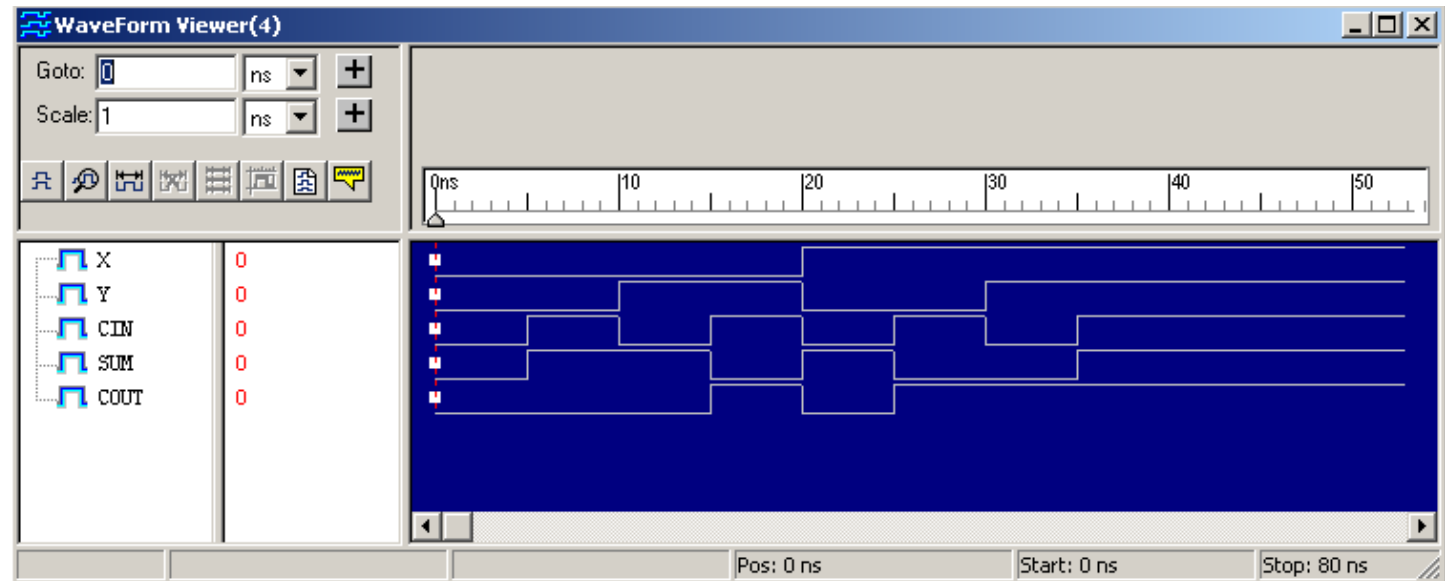




VISUALIZACIÓN DE LA SIMULACIÓN

Simulación
de un
ejemplo
sencillo

- Las señales seleccionadas se incorporan en la ventana *WaveForm Viewer*.
- Se permiten distintas opciones (Ampliaciones o reducciones, zooms parciales, visualización de valores y selección de tipo de información a mostrar, desplazamiento de cursor por la regleta graduada ...)

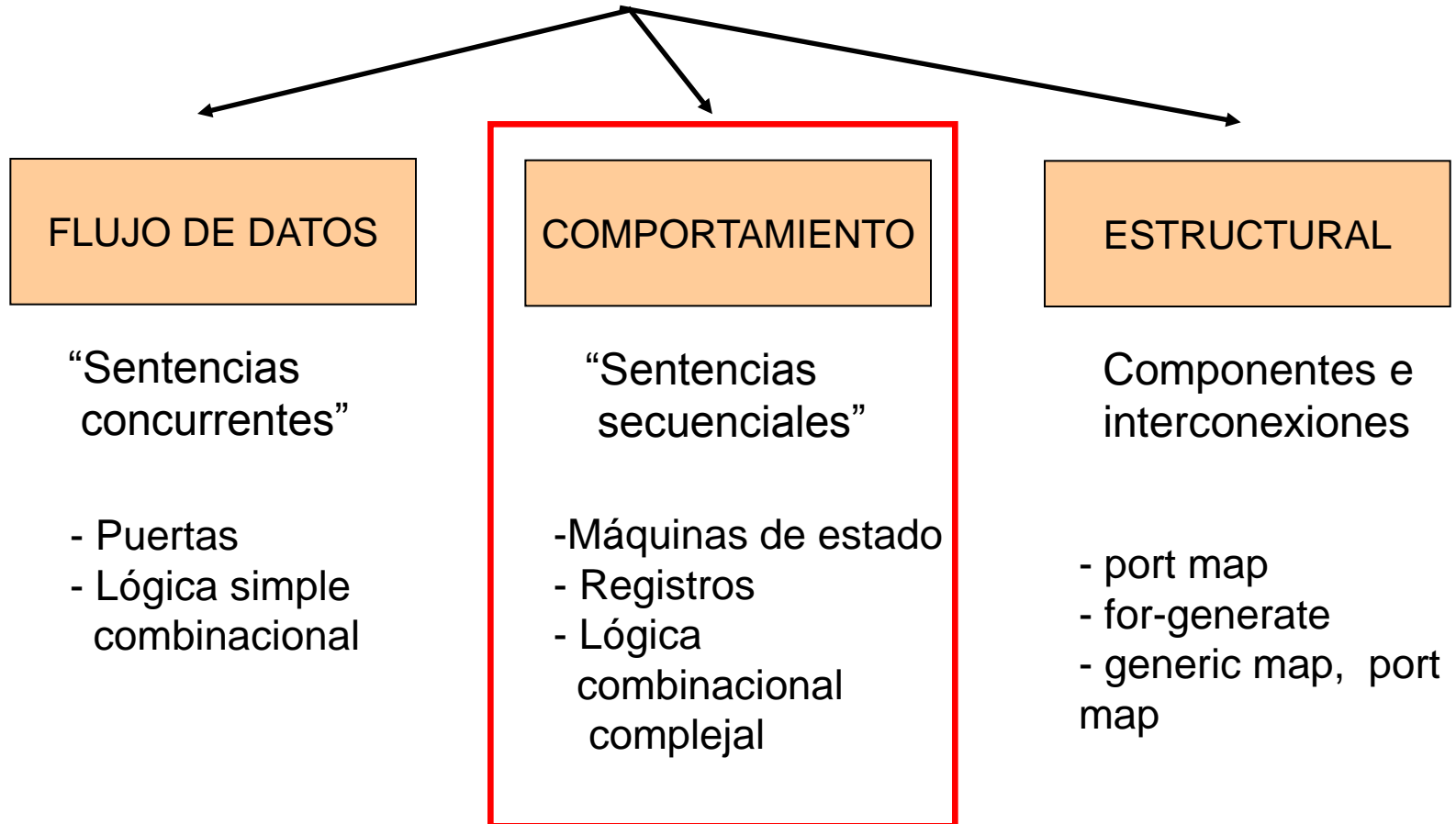


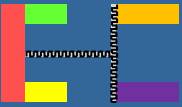


MODELOS DESCRIPTIVOS

Modelado
de
sistemas II

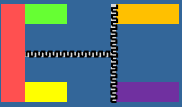
Estilos de diseño VHDL





- ⊙ 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.



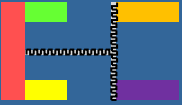


SENTENCIAS SECUENCIALES - IF

⊙ 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;

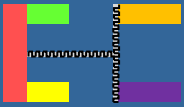
ELSE

f <= x3;

END IF;

END IF;

END PROCESS;

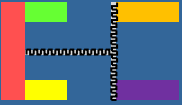


SENTENCIAS SECUENCIALES - CASE

⊙ 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.



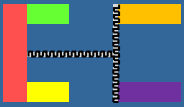
EJEMPLO- CASE

Modelado
de
sistemas II

```
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;
```





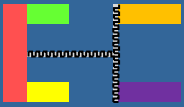
SENTENCIAS SECUENCIALES - FOR LOOP

⦿ Sintaxis

```
[etiqueta:] FOR índice IN rango'inf TO rango'sup LOOP  
          secuencia de sentencias  
END LOOP [etiqueta];
```

⦿ Ejemplo

```
bucle: FOR i IN 0 TO 5 LOOP  
      x(i) <= enable AND w(i+2);  
      y(0, i) <= w(i);  
END LOOP bucle;
```



SENTENCIAS SECUENCIALES - WHILE LOOP

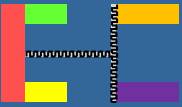
Modelado
de
sistemas II

- ⊙ El bucle se repite mientras la condición que haya después del while sea verdadera.
- ⊙ Sintaxis

```
[etiqueta:] WHILE condición LOOP  
                secuencia de sentencias  
            END LOOP [etiqueta];
```

- ⊙ Ejemplo

```
lazo1: WHILE (i<10) LOOP  
        WAIT UNTIL clk'EVENT AND clk='1';  
    END LOOP lazo1;
```



SENTENCIAS SECUENCIALES - LOOP

- ⊙ El bucle se repite infinitamente.
- ⊙ Sintaxis

```
[etiqueta:] LOOP  
                secuencia de sentencias  
                END LOOP [etiqueta];
```

- ⊙ Ejemplo

```
lazo1: LOOP  
      a <= c AND b;  
      END LOOP lazo1;
```

- ⊙ La única sentencia que deja romper el bucle es **EXIT**

SENTENCIAS EXIT Y NEXT

Exit:

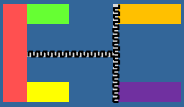
- Termina inmediatamente el bucle. Si hay varios bucles anidados, sale de donde se encuentre la instrucción o bien del bucle que se especifica en la etiqueta.

```
EXIT [etiqueta] [when condición];
```

Next

- Se salta el resto de sentencias del bucle y comienza con la siguiente iteración.

```
NEXT [etiqueta] [when condición];
```



SENTENCIAS WAIT

Modelado
de
sistemas II

- Espera a que ocurra una condición

WAIT UNTIL condición; (Ej: **WAIT UNTIL** e1='1' **AND** e2='1';)

- Espera a que cambie una señal de una lista de señales

WAIT ON lista de señales; (Ej: **WAIT ON** e1, e2, clk;)

- Espera un cierto tiempo

WAIT tiempo; (Ej: **WAIT** 25 ns;)

- Espera indefinidamente

WAIT ;

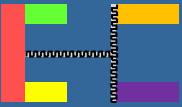




SENTENCIAS NULL

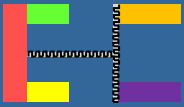
- ⦿ No realiza ninguna función. Pasa la ejecución a la siguiente sentencia secuencial.
- ⦿ Útil en sentencias CASE cuando no se quiere realizar ninguna acción para alguno de los casos.

[etiqueta]: **NULL**;



- ⊙ 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

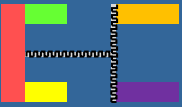
(clock='1') **AND** (clock'**EVENT**)

Flanco de subida de la señal clock

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

Flanco de bajada de la señal clock

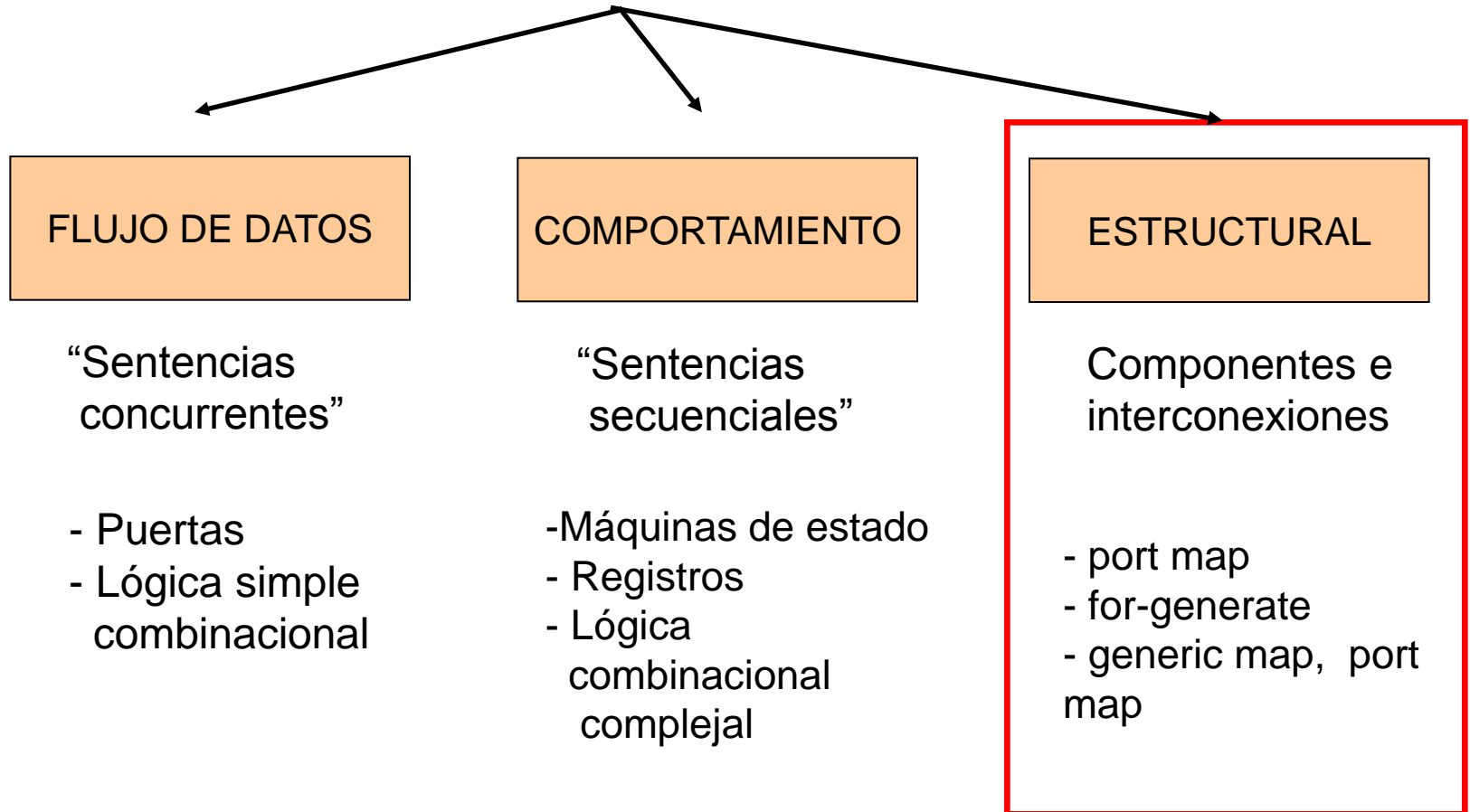
- ⊙ Para detectar flancos de señales tipo **STD_LOGIC_VECTOR** es mejor utilizar las funciones **rising_edge (S)** y **falling_edge (S)**.

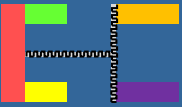


MODELOS DESCRIPTIVOS

Modelado
de
sistemas II

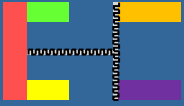
Estilos de diseño VHDL





- ⊙ 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

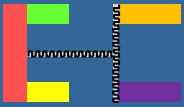
Modelado
de
sistemas II

⊙ Declaración:

```
COMPONENT nombre_componente IS
```

```
    PORT (locales);
```

```
END COMPONENT;
```



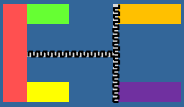
COMPONENTE. LLAMADA

- ⊙ 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);

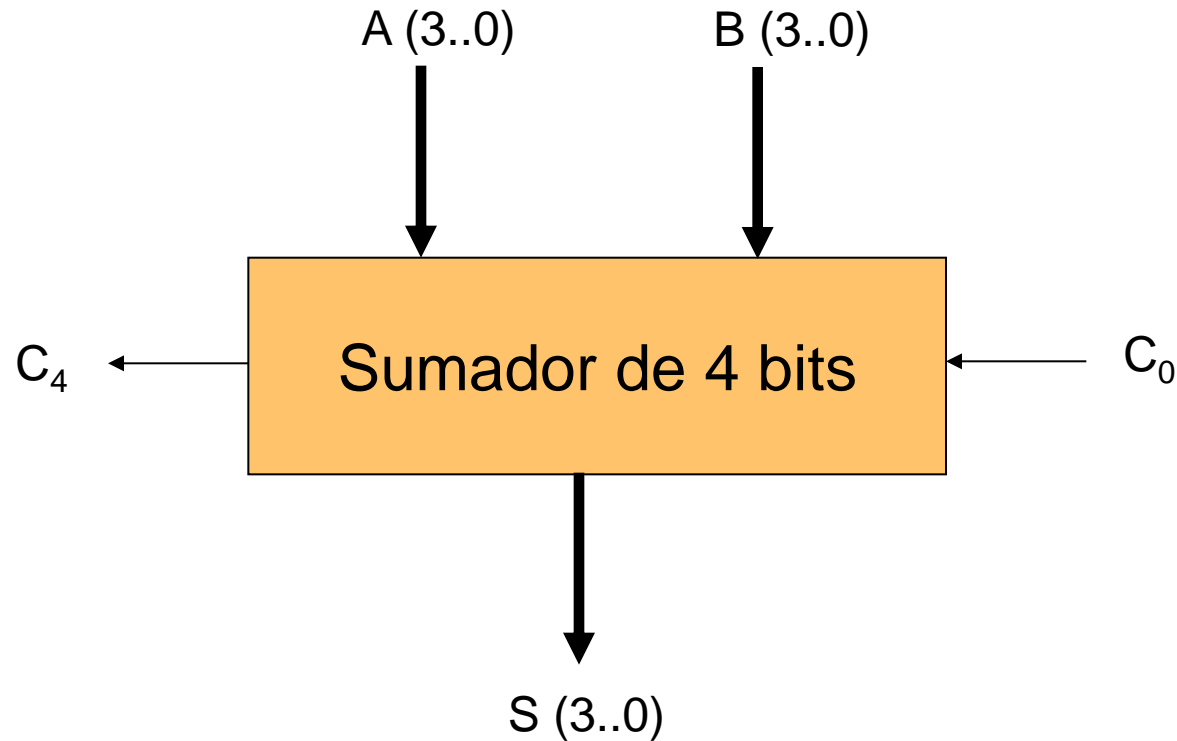
← **Conectividad
posicional**

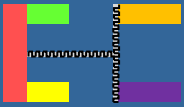


EJEMPLO PRÁCTICO. SUMADOR 4 BITS

Propuesta
de ejercicio
práctico

- Se propone describir y simular en Veribest 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

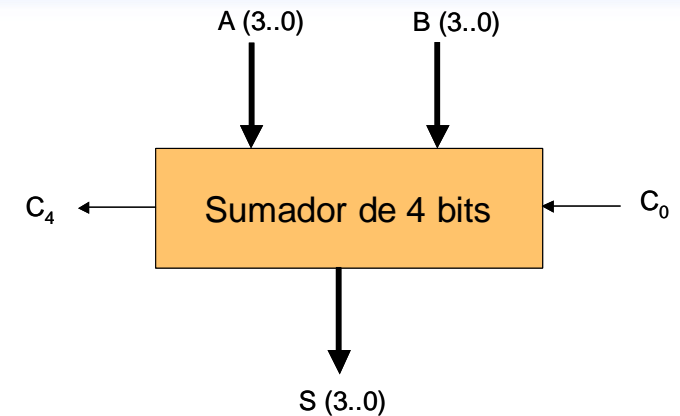
LIBRARY ieee;

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;

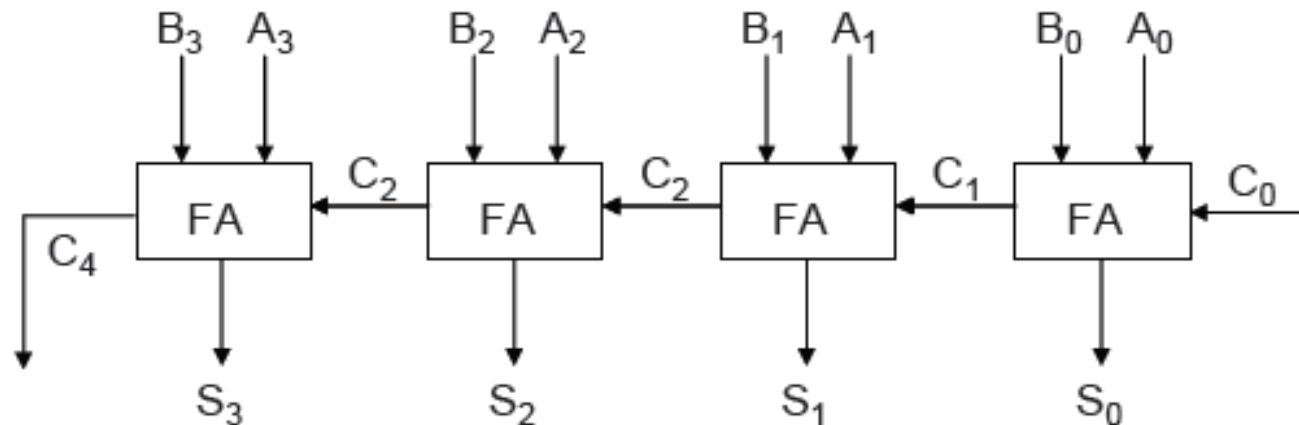


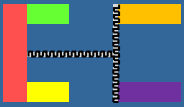


EJEMPLO PRÁCTICO. SUMADOR 4 BITS

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

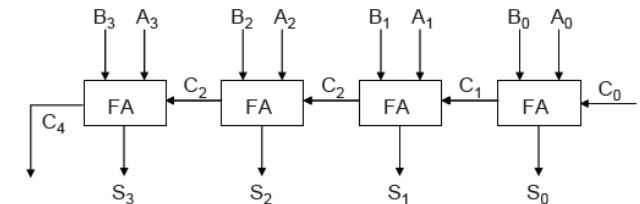
Sum1: Sumador_completo **PORT MAP** (A(0), B(0), C0, S(0), C1);

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);

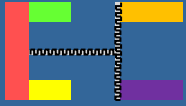
Sum4: Sumador_completo **PORT MAP** (A(3), B(3), C3, S(3), C4);

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
```

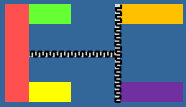
```
COMPONENT Sumador_4bits PORT ( A : IN STD_LOGIC_VECTOR (3 DOWNTO 0);  
                                B : IN STD_LOGIC_VECTOR (3 DOWNTO 0);  
                                C0 : IN STD_LOGIC;  
                                C4 : OUT STD_LOGIC;  
                                S : OUT STD_LOGIC_VECTOR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

```
END COMPONENT;
```

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

```
-- Por defecto se tomaría la última arquitectura compilada
```

```
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

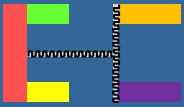
-- Instanciación del componente a testear y conexión de puertos

```
    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;
```

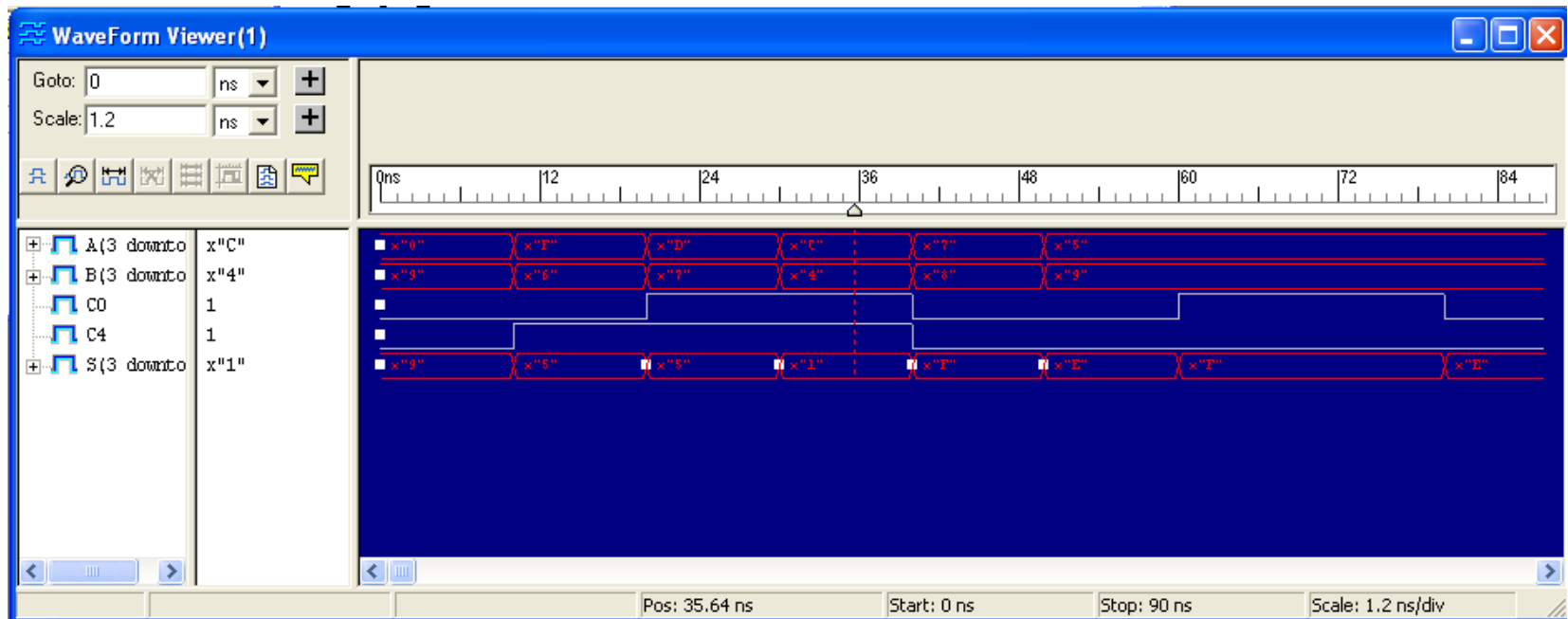
END test_flujo;

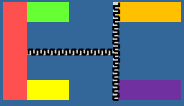


SIMULACIÓN DEL SUMADOR 4 BITS

Propuesta
de ejercicio
práctico

- El resultado de la simulación se podrá observar en la WaveForm Window.
- Desplazando el cursor por la ventana verificaremos el resultado correcto de la simulación.





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;
```

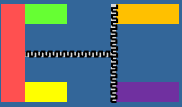
```
        Sum: FOR i IN 0 TO 3 GENERATE
```

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

```
        END GENERATE;
```

```
        C4 <= C(4);
```

```
    END circuito;
```



- ⦿ Son unidades de diseño donde se declaran y definen tipos, objetos, funciones, componentes...., para ser utilizados en cualquier descripción y modelo.

```
PACKAGE nombre_paquete IS
    -- zona de declaraciones
    -- tipos, subtipos, constantes, atributos...
END PACKAGE nombre_paquete;
[ PACKAGE BODY nombre_paquete IS
    -- descripción de FUNCTION y PROCEDURE
END PACKAGE nombre_paquete; ]
```

- ⦿ Primera parte obligatoria. La segunda solo si se declaran subprogramas (FUNCTION y PROCEDURE).





EJEMPLO DE PAQUETE

Elementos del lenguaje

🎯 Ejemplo de declaración de paquete

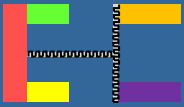
```
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL;
```

```
PACKAGE mipaquete IS  
    TYPE color IS (rojo,verde,ambar);  
    COMPONENT inversor PORT (a: IN STD_LOGIC; B: OUT STD_LOGIC);  
    END COMPONENT;  
    FUNCTION flanco_pos (SIGNAL s: STD_LOGIC) RETURN BOOLEAN;  
END mipaquete ;
```

🎯 Ejemplo de cuerpo de paquete

```
PACKAGE BODY mipaquete IS  
    FUNCTION flanco_pos (SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS  
        BEGIN  
            RETURN (s'EVENT AND s='1');  
        END flanco_pos;  
END mipaquete ;
```





FUNDAMENTOS DEL VHDL

Elementos del lenguaje

⊙ Invariancias del VHDL:

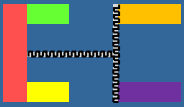
- ⊙ Invariante a mayúsculas, es decir, dos expresiones iguales conteniendo mayúsculas y minúsculas son idénticas.

`Salida <= A and B; ≡ SALIDa <= a AND b;`

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

`Salida <= A and B; ≡ 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

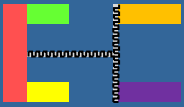
Elementos del lenguaje

- ⊙ 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**”
- ⊙ Cada “**loop**” termina con el correspondiente “**end loop**”



FUNDAMENTOS DEL VHDL

Elementos del lenguaje

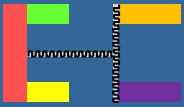
- ⊙ **Identificadores.** Son las palabras que se utilizan para identificar a las funciones, señales, puertos, variables, etc..
- ⊙ El identificador debe dar suficiente información para su uso .
- ⊙ El identificador puede ser tan largo como se quiera, pero un nombre demasiado largo es complicado de utilizar, 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**.

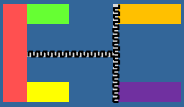
Ejemplo: **CONSTANT** retardo :**TIME** := 15 ns;

⊙ Variables:

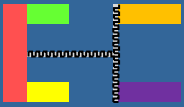
- ⊙ Solo pueden declararse y utilizarse dentro de los procesos.
- ⊙ La asignación se realiza con :=
- ⊙ Las asignaciones se realizan inmediatamente

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





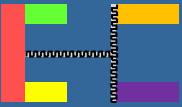
- ⊙ **Señales:**
 - ⊙ Representan conexiones físicas.
 - ⊙ Se declaran en sentencias concurrentes y pueden aparecer también en procesos.
 - ⊙ Las asignaciones no son instantáneas, se actualizan en el siguiente paso de simulación (sentencia **WAIT** o si ha terminado una sentencia concurrente).
 - ⊙ Se utilizan al comunicar procesos e interconectar componentes.
 - ⊙ Si no se especifica un retardo (**AFTER**) se aplica un retardo tipo delta (δ).
 - ⊙ Para asignar valores a una señal se utiliza `<=`



- ◎ **Asignación de señales:**
 - ◎ El valor a asignar se pasa a una cola de eventos y se actualiza al avanzar el tiempo de simulación.

- ◎ Existen tres modelos:
 - ◎ **Transporte**. Describe el comportamiento de una línea de transmisión ideal.
 - ◎ **Inercial**. Describe los circuitos reales.
 - ◎ **Delta**. Es el retardo por defecto





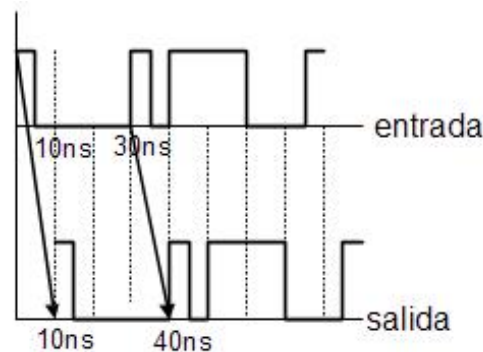
FUNDAMENTOS DEL VHDL

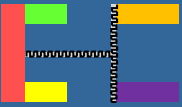
Elementos del lenguaje

- ⊙ **Transporte.** Permite modelar dispositivos que presentan un comportamiento en frecuencia casi infinito:
 - ⊙ La entrada se propaga a la salida sin ninguna alteración. No importa lo pequeña o lo grande que sea la duración de la entrada.
 - ⊙ Este comportamiento es típico de las líneas de transmisión.

```
nom_señal <= TRANSPORT [expresión] AFTER tiempo;
```

```
salida <= TRANSPORT entrada AFTER 10ns;
```





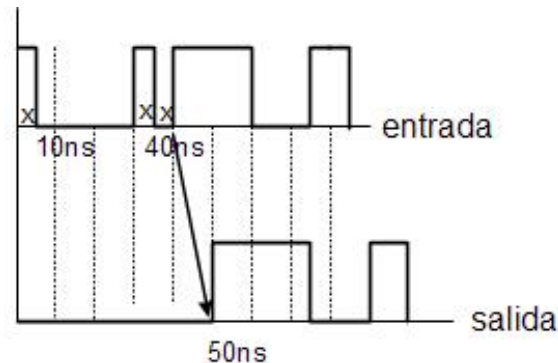
FUNDAMENTOS DEL VHDL

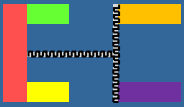
Elementos del lenguaje

- ⊙ **Inercial.** Permite modelar el comportamiento temporal de la conmutación de los circuitos:
 - ⊙ La entrada debe tener un valor estable durante tiempo especificado antes de que se propague a la salida, sino la entrada se ignora.

```
nom_signal_out <= [[REJECT tiempo] INERTIAL] [expresión] nom_signal  
AFTER tiempo;
```

salida <= **REJECT** 7ns **INERTIAL** entrada **AFTER** 10ns;





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**

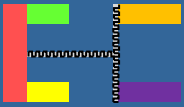
dia : INTEGER RANGE 1 TO 31;

mes: INTEGER RANGE 1 TO 12;

año: INTEGER RANGE 1900 TO 2025;

END RECORD;

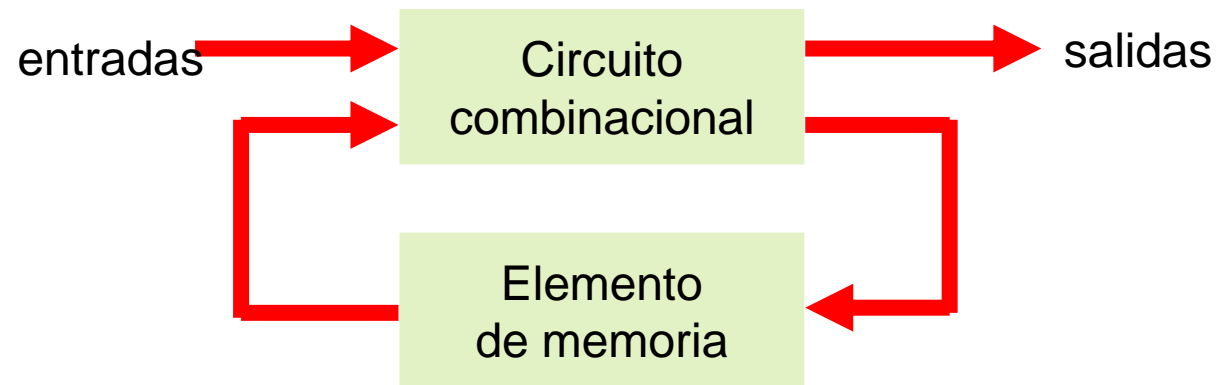
VARIABLE hoy, ayer: fecha;



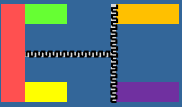
MODELADO SECUENCIAL

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

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

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

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

```
ARCHITECTURE BiastableDarq OF BiastableD IS  
BEGIN
```

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

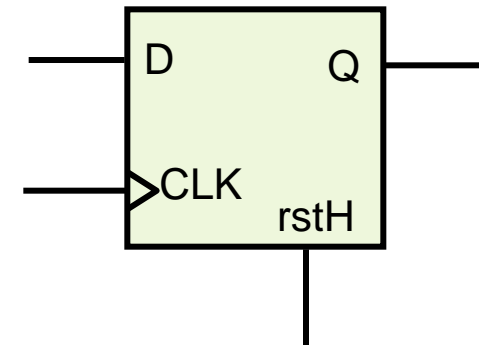
```
        IF (rstH = '1') THEN  
            Q <= '0';
```

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

```
        END IF;
```

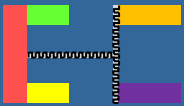
```
    END PROCESS;
```

```
END BiastableDarq;
```



Señal reset asíncrona

Biastable disparado
por flanco de subida



CONTADOR DE 8 BITS

Trabajo propuesto

- Simular el siguiente contador de 8 bits

```
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');
```

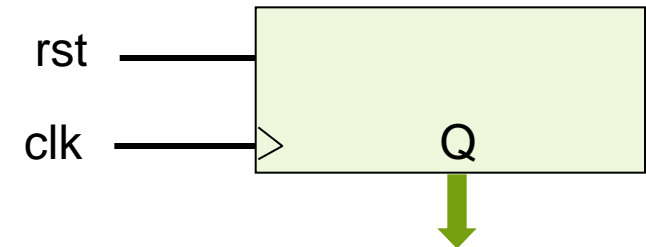
```
        ELSIF CLK='1' AND CLK'EVENT THEN
```

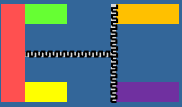
```
            Q <= Q + 1;
```

```
        END IF;
```

```
    END PROCESS;
```

```
END ContaNBits;
```





CONTADOR DE 8 BITS

Modelado secuencial

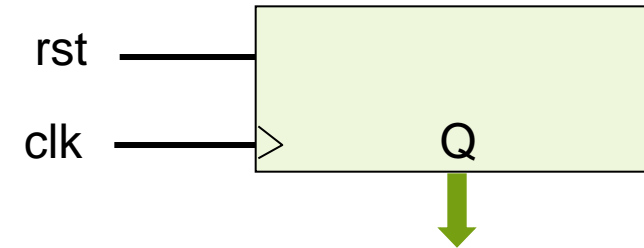
Con el siguiente testbench:

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

ENTITY test_contador IS END test_contador ;

ARCHITECTURE test_1 OF test_contador IS
  COMPONENT 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 COMPONENT;

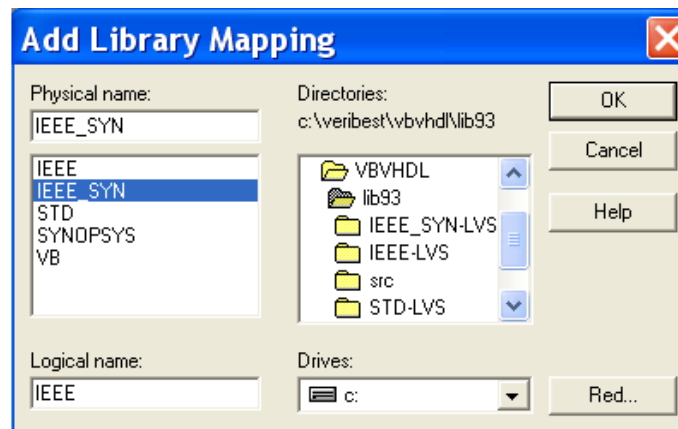
  FOR C1: contador USE ENTITY WORK.contador;
--Declaración de las señales
  SIGNAL CLK, rst : STD_LOGIC :='0';
  SIGNAL Q : STD_LOGIC_VECTOR (7 DOWNT0 0);
--Cuerpo de la Arquitectura
  BEGIN
    C1: contador PORT MAP (CLK => CLK, rst => rst, Q =>Q);
--Asignación de Valores
    rst <= '1', '0' AFTER 80 ns;
    PROCESS (CLK)
      BEGIN
        CLK <= NOT CLK AFTER 25ns;
      END PROCESS;
  END test_1;
```

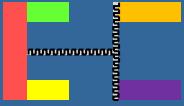


Forma sencilla
de simular
el reloj

Anexo CONTADOR DE 8 BITS

- ⦿ Para utilizar algunas bibliotecas como la arith o unsigned hay que añadir una biblioteca especial para esas características.
- ⦿ Pulsar sobre la opción *Add Library Mapping* del menú *Library* y buscar la biblioteca IEEE_SYN.
- ⦿ Este nombre IEEE_SYN se incorpora al nombre físico (*Physical name*) y se le asigna el nombre lógico IEEE para evitar tener que retocar el código.

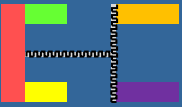




Práctica a implementar

Práctica a implementar

- ⊙ La realización de las prácticas implica la entrega del trabajo propuesto que se detalla a continuación.
- ⊙ Cada una de las prácticas se entregará en un archivo comprimido que deberá contener:
 - ⊙ El archivo de la memoria documental (en la que se indica cómo se ha hecho la práctica, cómo se ha desarrollado, el código que ha implementado y volcados de pantalla demostrando que funciona).
 - ⊙ Los archivos asociados a la implementación para que el profesor pueda ejecutarlos.



Ejercicios a Realizar

Realización práctica 1

- ⊙ Realiza paso a paso el ejemplo correspondiente a la implementación de un sumador completo (pags. 33 a 50). Ejecuta el test y comprueba que los resultados son los esperados.
- ⊙ Teniendo en cuenta que nos apoyaremos en el ejercicio anterior, implementa ahora el ejemplo correspondiente al sumador de 4 bits (pags. 71 a 77). Comprueba que el test nos proporciona los resultados correctos. Modifícalo añadiendo **tres valores más** para los Datos A y B.
- ⊙ Haz lo mismo para el ejemplo de circuito secuencial que representa al contador de 8 bits (pags. 94 a 96).
- ⊙ Escribe un testbench para el ejemplo del biestable D (pag. 93) y comprueba que funciona correctamente. Modifica la descripción original y añádele una entrada asíncrona de PRESET activa a nivel alto. Incorpórala también al testbench.