
Clase 2:

Descripción de la funcionalidad

2.1 Introducción/Repaso

2.2 Concepto de concurrencia

2.3 Procesos

2.4 Diseño de circuitos combinacionales

2.4.1. Sentencias concurrentes

2.4.2. Descripción mediante procesos

Clase 2:

Descripción de la funcionalidad

2.1 Introducción/Repaso

2.2 Concepto de concurrencia

2.3 Procesos

2.4 Diseño de circuitos combinacionales

2.4.1. Sentencias concurrentes

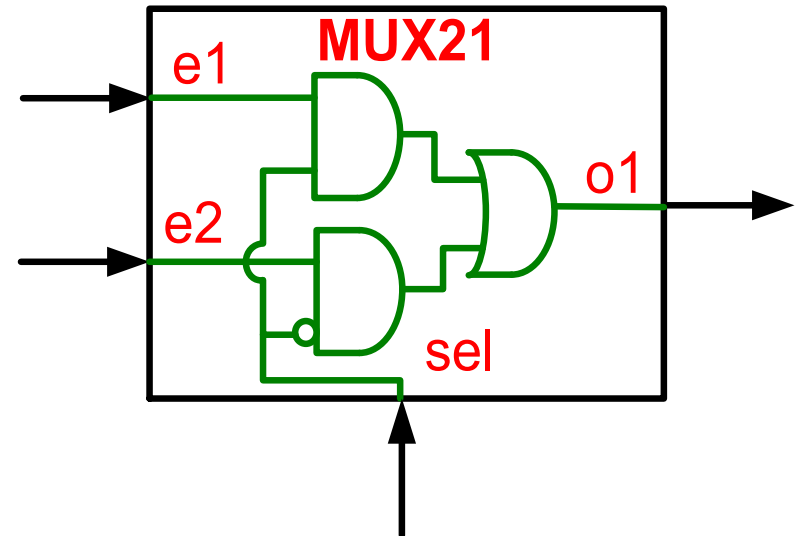
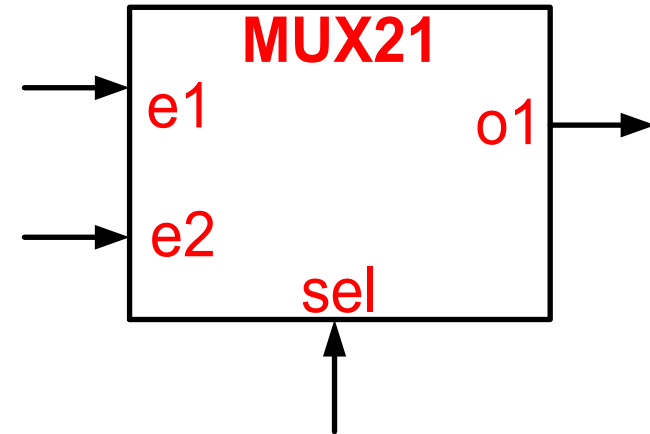
2.4.2. Descripción mediante procesos

Introducción

-- Multiplexor de dos entradas

```
entity mux21 is  
port ( e1, e2: IN std_logic;  
        sel: IN std_logic;  
        o1: OUT std_logic);  
end mux21;
```

```
architecture A of mux21 is  
begin  
process(e1,e2, sel)  
begin  
if (sel= '0' ) then  
o1 <= E1;  
else  
o1 <= e2;  
endif;  
end process;  
end A;
```



Introducción. Formas de describir una ARCHITECTURE

- ◆ Dos formas distintas :

1. Órdenes concurrentes: Ejecución de órdenes en paralelo.
2. Utilizando **process**: Dentro de un process las órdenes se ejecutan de forma secuencial.

Es posible combinar ambas en una misma ARCHITECTURE

Clase 2:

Descripción de la funcionalidad

2.1 Introducción

2.2 *Concepto de concurrencia*

2.3 Procesos

2.4 Diseño de circuitos combinacionales

2.4.1. Sentencias concurrentes

2.4.2. Descripción mediante procesos

Concepto de concurrencia

ARCHITECTURE

Sentencias concurrentes: Órdenes independientes que se ejecutan en paralelo.

```
A<=C and D or G;  
F <= A when E='0' else 'H';
```

PROCESS: Conjunto de órdenes que se ejecutan de forma secuencial.

```
process (e1, e2, sel)  
begin  
    if (sel='0') then  
        o1 <= E1;  
    else  
        o1 <= e2;  
    endif;  
end process;
```

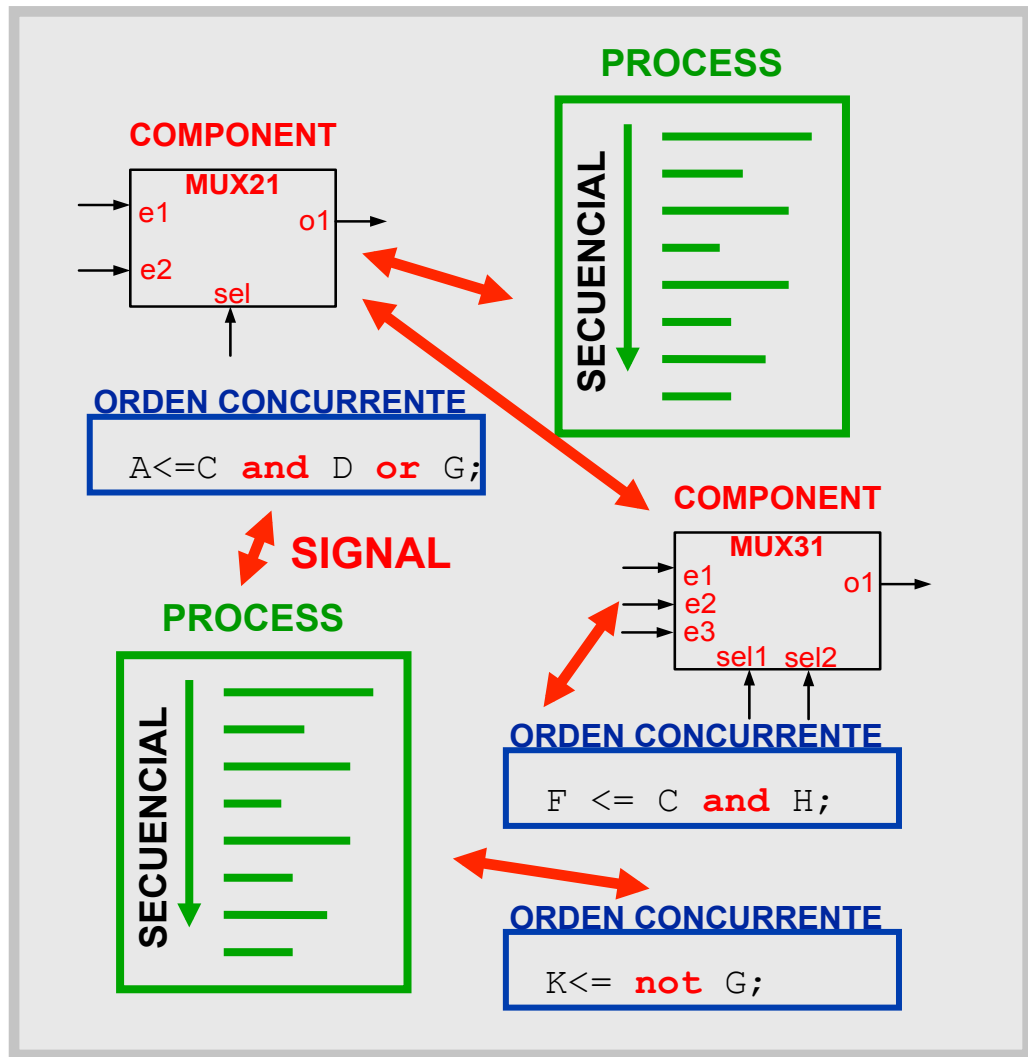
INSTANCIAS A COMPONENTES: Cada componente será un bloque que funcionará en paralelo

```
cont1: contador  
GENERIC MAP (N=>7)  
PORT MAP ( ..... )
```

- ◆ Por naturaleza, todos los elementos de un circuito funcionan de forma concurrente.
- ◆ **Todas las sentencias fuera de un *process* se ejecutan de forma paralela.**
- ◆ El orden de las sentencias no es importante.
- ◆ Un *componente* se entiende como otra sentencia concurrente.
- ◆ Un *process* se comporta externamente como una sentencia concurrente.

Concepto de concurrencia

ARCHITECTURE



- ◆ Todas las órdenes dentro de un process se ejecutan secuencialmente, pero externamente se comporta como una sola orden concurrente.
- ◆ Un *componente* se entiende como otra sentencia concurrente.
- ◆ Los diferentes elementos se comunican utilizando **signals**

Clase 2:

Descripción de la funcionalidad

2.1 Introducción

2.2 Concepto de concurrencia

2.3 Procesos

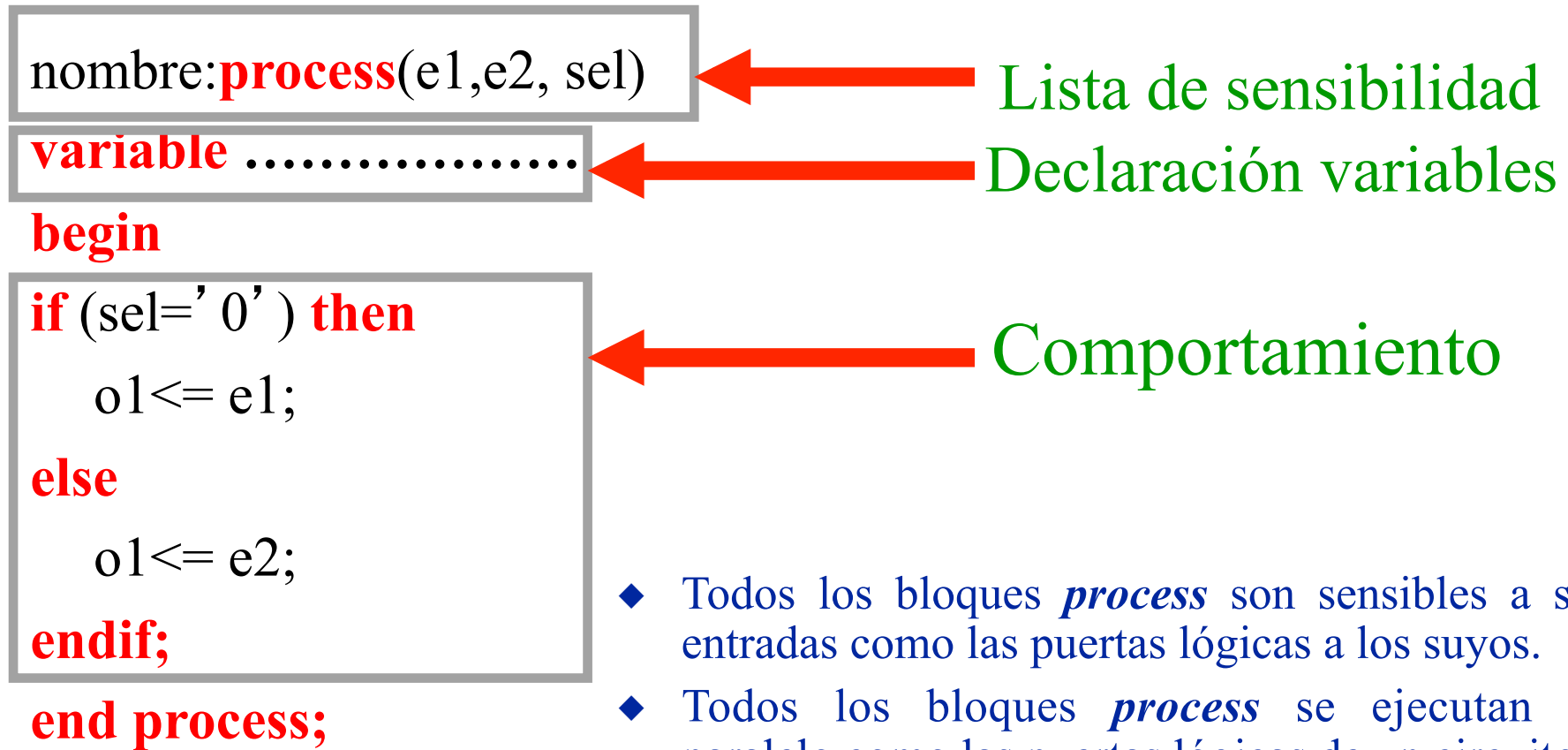
2.4 Diseño de circuitos combinacionales

2.4.1. Sentencias concurrentes

2.4.2. Descripción mediante procesos

Proceso

El bloque `process` como unidad básica de ejecución.



Proceso. Lista de sensibilidad

- ◆ El proceso se ejecuta en su totalidad cuando cambia algunas señal de la lista de sensibilidad

```
proc1: process (a,b,c)
begin
  x <= a and b and c;
end process;
```

```
proc2: process (a,b)
begin
  x <= a and b and c;
end process;
```

- ◆ Desde el punto de vista de simulación, ambos procesos son muy diferentes.
- ◆ El *proc2* no es sintetizable (faltará un latch para c que es sensible a ambos flancos de a y b), por tanto:
 - El programa de síntesis añadirá la señal c a la lista de sensibilidad para poder asociar un circuito al código.
 - El programa de síntesis produce un mensaje de error.

Proceso. Lista de sensibilidad

CONCLUSIÓN: En un código VHDL para síntesis de un proceso combinacional, en la lista de sensibilidad deben incluirse todas las señales que afecten a la evaluación del proceso.

- Un proceso sin lista de sensibilidad se estaría ejecutando siempre, por tanto el simulador nunca avanzaría.
- Si una señal de la lista de sensibilidad cambia el proceso es evaluado sucesivamente hasta que todas las señales sensibles se estabilicen.

Proceso. Asignación de señales

- ◆ Todas las expresiones de un proceso se resuelven con el **valor actual** de las señales.
- ◆ Las asignaciones se hacen efectivas cuando se **termine** la evaluación del proceso.
- ◆ Todas las señales serán actualizadas con el **último valor** asignado dentro del proceso.
- ◆ El proceso vuelve a evaluarse si se ha modificado alguna señal de su lista sensible.
- ◆ Todos los procesos cuya lista sensible ha sido alterada y todas las órdenes concurrentes se ejecutan hasta que se alcanza un estado de equilibrio (no hay más cambios).

Proceso. Asignación de señales (ejemplo 1)

◆ Diferentes opciones para implementar el mux2

```
process(e1,e2, sel)
begin
  if (sel=' 0' ) then
    o1<= e1;
  else
    o1<= e2;
  endif;
end process;
```

```
process(e1,e2, sel)
begin
  o1<= e1;
  if (sel=' 1' ) then
    o1<= e2;
  endif;
end process;
```

```
process(e1,e2, sel)
begin
  if (sel=' 1' ) then
    o1<= e2;
  endif;
  o1<= e1;
end process;
```

— **INCORRECTO**

O1 siempre valdrá e1

Clase 2:

Descripción de la funcionalidad

2.1 Introducción

2.2 Concepto de concurrencia

2.3 Procesos

2.4 *Diseño de circuitos combinacionales*

2.4.1. Sentencias concurrentes

2.4.2. Descripción mediante procesos

Clase 2:

Descripción de la funcionalidad

2.1 Introducción

2.2 Concepto de concurrencia

2.3 Procesos

2.4 Diseño de circuitos combinacionales

2.4.1. Sentencias concurrentes

2.4.2. Descripción mediante procesos

Sentencias concurrentes. Asignación simple

```
señal_destino <= señal_origen;
```

- ◆ Deben ser del mismo tipo (integer, std_logic, etc)
- ◆ señal_destino puede ser:
 - Señal
 - Puerto: OUT, INOUT (y *BUFFER*)
- ◆ señal_origen puede ser:
 - Señal
 - Puerto: IN, INOUT (y *BUFFER*)

PUEDEN USARSE DENTRO O FUERA DE UN PROCESS

Sentencias concurrentes. Ecuaciones lógicas

Operador	Ejemplo
NOT	<code>a <= NOT b;</code>
AND, NAND	<code>a <= b AND c;</code>
OR, NOR, XOR	<code>a <= b OR c AND d;</code>

PUEDEN USARSE DENTRO O FUERA DE UN PROCESS

Sentencias concurrentes. Ejemplo

```
library IEEE;  
    use IEEE.std_logic_1164.all;  
entity mux21 is  
port ( e1, e2:      IN std_logic;  
      sel:      IN std_logic;  
      o1:      OUT std_logic);  
end mux21;  
architecture A of mux21 is  
begin  
    o1 <= e1 and not(sel) or e2 and sel;  
end A;
```

Sentencias concurrentes. WHEN ELSE

```
Señal Destino <= Expresión1 WHEN Condición ELSE Expresión2;
```

```
Señal Destino <= Expresión1 WHEN Condición1 ELSE  
    Expresión2 WHEN Condición2 ELSE  
    Expresión3 WHEN Condición3 ELSE  
    .....  
    ExpresiónN WHEN CondiciónN  
ELSE Expresión;
```

NO PUEDEN USARSE DENTRO DE UN PROCESS

Sentencias concurrentes. WHEN ELSE

```
library IEEE;  
    use IEEE.std_logic_1164.all;  
entity mux21 is  
port ( e1, e2:      IN std_logic;  
      sel:        IN std_logic;  
      o1:        OUT std_logic);  
end mux21;  
architecture A of mux21 is  
begin  
    o1 <= e1 when sel='0' else e2;  
end A;
```

Sentencias concurrentes. WITH ... SELECT ...

```
WITH expresión_selección SELECT  
Señal Destino <=  
    Expresión1 WHEN Valor_Selección1,  
    Expresión2 WHEN Valor_Selección2,  
    ...  
    ExpresiónN WHEN OTHERS;
```

```
WITH sel SELECT  
o1 <=  
    e1 WHEN '0',  
    e2 WHEN OTHERS;
```

NO PUEDEN USARSE DENTRO DE UN PROCESS

Clase 2:

Descripción de la funcionalidad

2.1 Introducción

2.2 Concepto de concurrencia

2.3 Procesos

2.4 Diseño de circuitos combinacionales

2.4.1. Sentencias concurrentes

2.4.2. *Descripción mediante procesos*

IF THEN ELSIF ELSE

```
IF condición 1 THEN  
    Consecuencia 1;  
ELSIF condición 2 THEN  
    Consecuencia 2;  
ELSIF ... THEN  
    ...  
ELSE Consecuencia;  
END IF;
```

NO PUEDEN USARSE FUERA DE UN PROCESS

IF THEN ELSIF ELSE

```
architecture A of mux21 is
begin
  process (e1, e2, sel)
  begin
    if (sel='0') then
      o1 <= e1;
    else
      o1 <= e2;
    endif;
  end process;
end A;
```


CASE WHEN

```
CASE objeto IS  
    WHEN caso1 =>  
        Código1;  
    WHEN caso2 =>  
        Código2;  
    WHEN caso3 =>  
        Código34;  
    WHEN OTHERS =>  
        Código;  
END CASE;
```

NO PUEDEN USARSE FUERA DE UN PROCESS

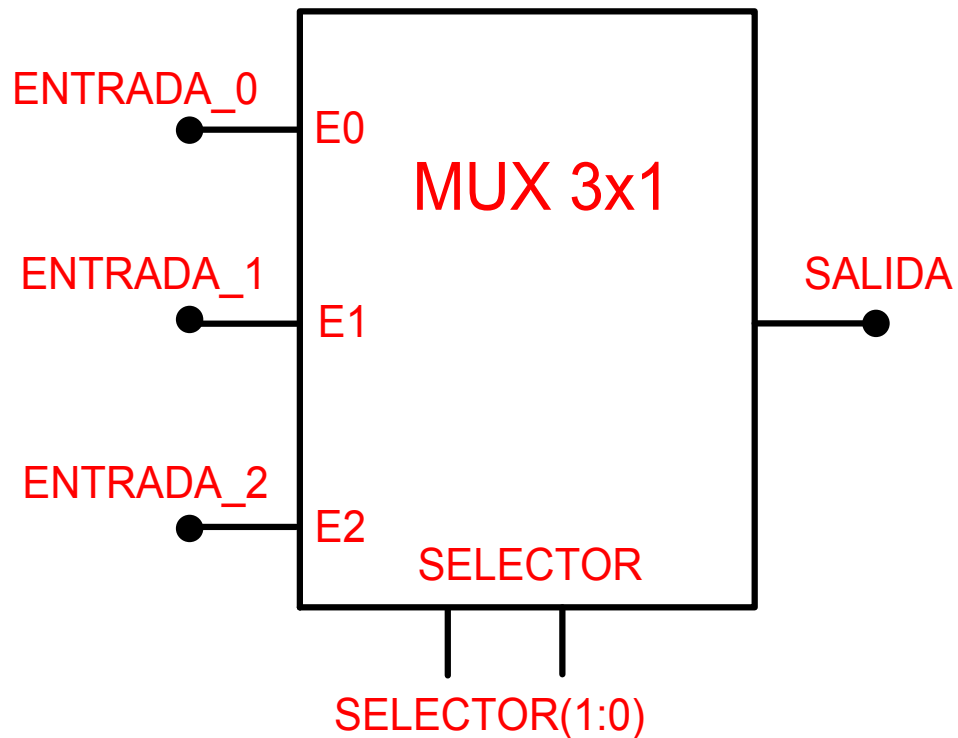
CASE WHEN

```
architecture A of mux21 is
begin
  process (e1, e2, sel)
  begin
    case sel is
      when '0' =>
        o1<=e1;
      when others =>
        o1<=e2;
    end case;
  end process;
end A;
```

WHEN OTHERS es obligatorio si no están cubiertos todos los casos

Generación de LATCHES

Ej. Multiplexor 3x1



¿Qué ocurre si una
s e n t e n c i a
condicional se deja
incompleta?

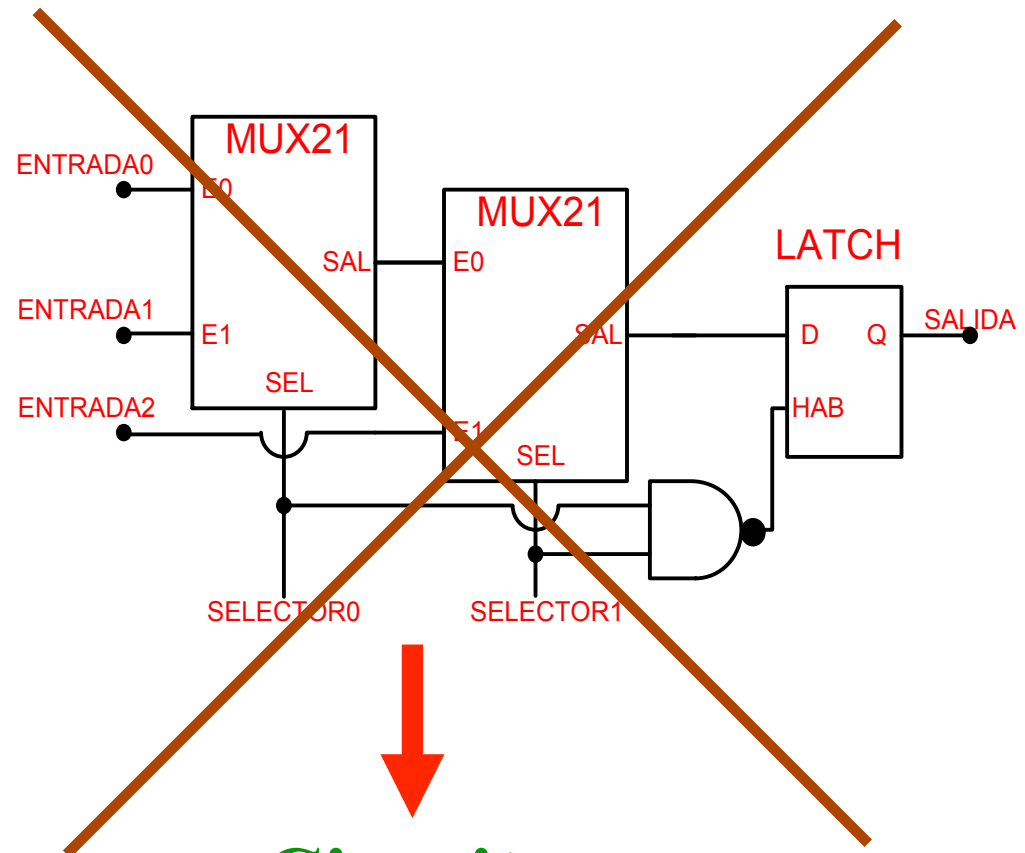
Se genera un **LATCH**

Sentencias condicionales incompletas.

- ◆ No se define la condición (*selector*="11")

Programación

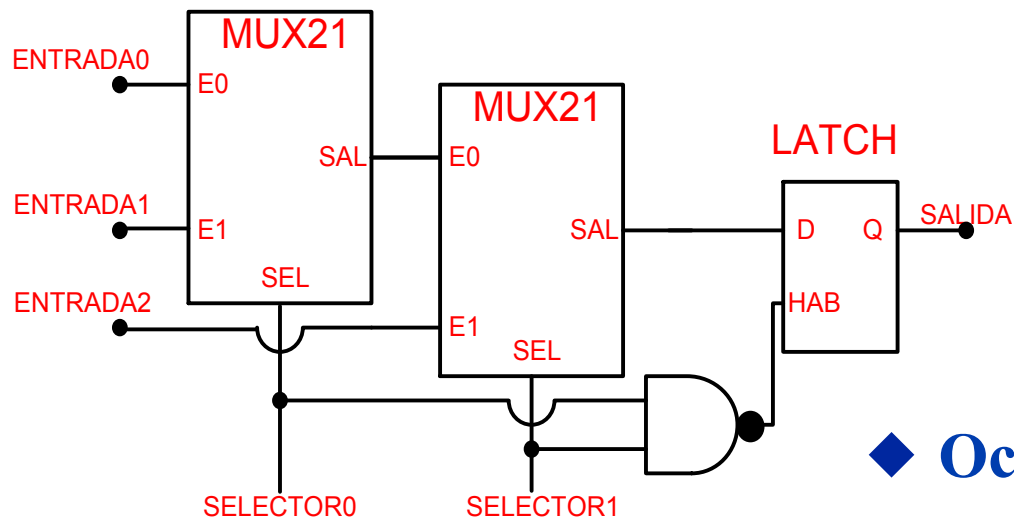
```
if (selector="00") then  
    salida<=entrada_0;  
elsif (selector="01") then  
    salida<=entrada_1;  
elsif (selector="10") then  
    salida<=entrada_2;  
endif;
```



**Circuito
secuencial**

Sentencias condicionales incompletas.

Circuito con un latch



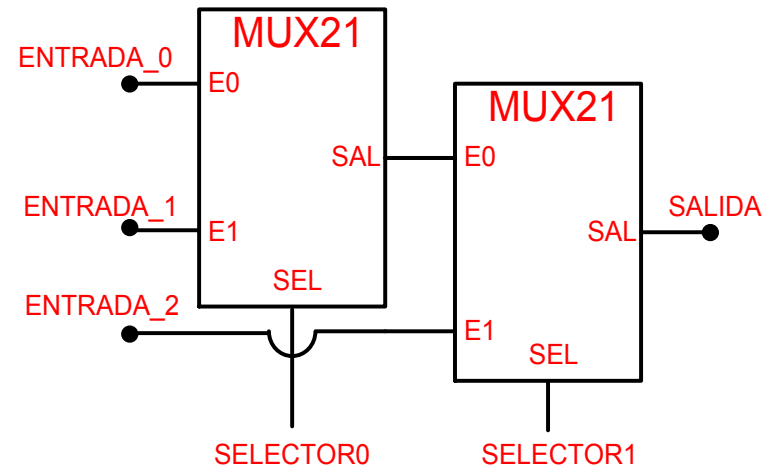
- ◆ Ocupa más área
- ◆ No realiza la función lógica:
 - ¡¡Es sensible a pulsos !!

Sentencias condicionales incompletas.

◆ Código correcto

Descripción

```
if (selector="00") then
  salida<=entrada_0;
elsif (selector="01") then
  salida<=entrada_1;
elsif (selector="10") then
  salida<=entrada_2;
else
  salida<=entrada_2;
end if;
```



**Circuito
combinacional**

Circuitos combinacionales y secuenciales.

Conclusiones

- ◆ Los circuitos combinacionales deben estar completamente especificados.
- ◆ Se genera una celda de memoria siempre que:
 - No se especifica el valor de la salida para todos los posibles valores de las entradas
- ◆ Un latch es un circuito muy sensible a pulsos y timing:
 - Sólo debe ser utilizado en casos concretos.

Driver de una pantalla de 7 LEDs

Ejemplo diseño combinacional

```
— LIBRARY IEEE;  
— USE IEEE.std_logic_1164.all;  
— ENTITY bit2led IS  
—   PORT (  
—     bits: IN STD_LOGIC_VECTOR (3 downto 0);  
—     led: OUT STD_LOGIC_VECTOR (6 downto 0)  
—   );  
— END bit2led;  
— ARCHITECTURE bit2led_arch OF bit2led IS  
— BEGIN  
—   -- contenido de la arquitectura  
  
— END bit2led_arch;
```


Driver de una pantalla de 7 LEDs

Ejemplo diseño combinacional

```
async_bit: PROCESS(bits)
```

```
BEGIN
```

```
CASE bits IS
```

```
  WHEN "0001" =>
```

```
    LED <= "0000110";  --1
```

```
  WHEN "0010" =>
```

```
    LED <= "1011011";  --2
```

```
  WHEN "0011" =>
```

```
    LED <= "1001111";  --3
```

```
  WHEN "0100" =>
```

```
    LED <= "1100110";  --4
```

```
  WHEN "0101" =>
```

```
    LED <= "1101101";  --5
```

```
  WHEN "0110" =>
```

```
    LED <= "1111101";  --6
```

```
  WHEN "0111" =>
```

```
    LED <= "0000111";  --7
```

```
  WHEN "1000" =>
```

```
    LED <= "1111111";  --8
```

```
  WHEN "1001" =>
```

```
    LED <= "1101111";  --9
```

```
  WHEN "1010" =>
```

```
    LED <= "1110111";  --A
```

```
  WHEN "1011" =>
```

```
    LED <= "1111100";  --b
```

```
  WHEN "1100" =>
```

```
    LED <= "0111001";  --C
```

```
  WHEN "1101" =>
```

```
    LED <= "1011110";  --d
```

```
  WHEN "1110" =>
```

```
    LED <= "1111001";  --E
```

```
  WHEN "1111" =>
```

```
    LED <= "1110001";  --F
```

```
  WHEN OTHERS =>
```

```
    LED <= "0111111";  --0
```

```
  END CASE;
```

```
END PROCESS;
```



Tema 4:

Descripción de la funcionalidad

2.1 Introducción/Repaso

2.2 Concepto de concurrencia

2.3 Procesos

2.4 Diseño de circuitos combinacionales

2.4.1. Sentencias concurrentes

2.4.2. Descripción mediante procesos

2.5 Interconexión de componentes

La sección ARCHITECTURE. Componente

- ◆ Componente: entidad de un orden inferior de jerarquía
- ◆ Sintáxis:

```
component nombre
  generic (definición de generics);
  port (definición de puertos);
end component;
```

Declaración del componente.

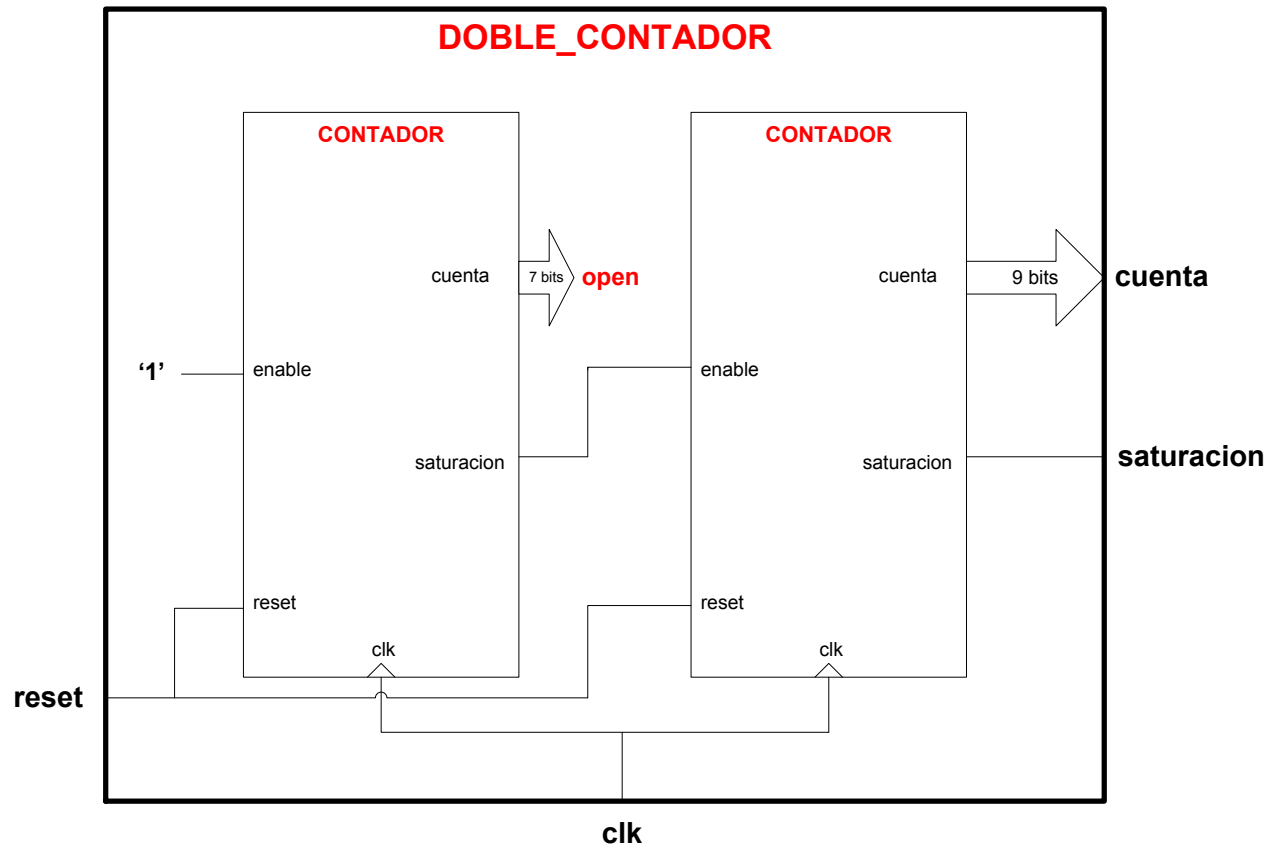
*En la zona de
declaraciones de
señales y
componentes*

```
nombre_instancia : nombre_componente
  GENERIC MAP (.....)
  PORT MAP (.....);
```

Utilización del componente.

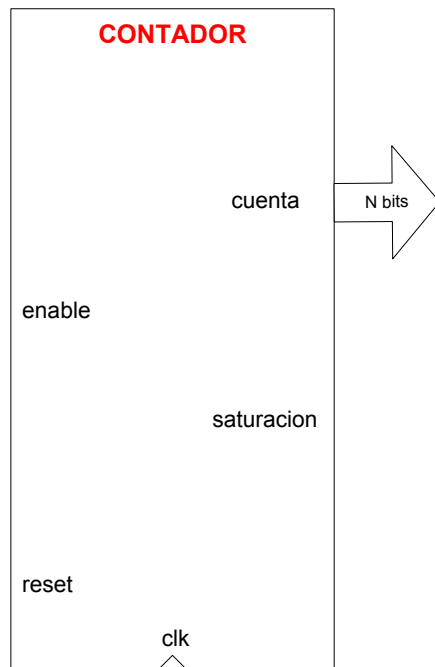
*Dentro del cuerpo
de la arquitectura
(BEGIN)*

La sección ARCHITECTURE. Descripción de un esquema. Ejemplo



Objetivo

La sección ARCHITECTURE. Descripción de un esquema. Ejemplo

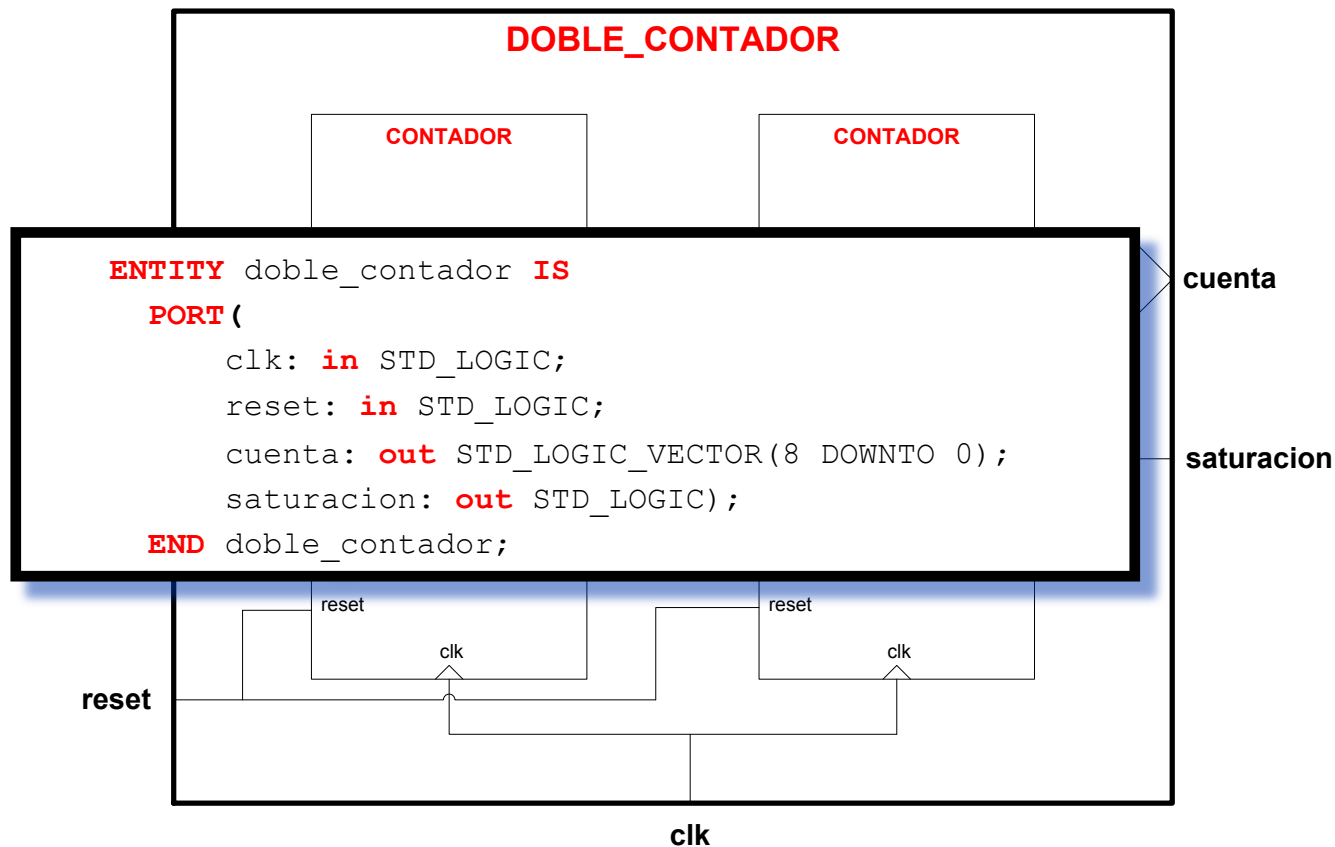


```
ENTITY contador IS
    GENERIC (N:integer:=10);
    PORT (
        clk: in STD_LOGIC;
        reset: in STD_LOGIC;
        enable: in STD_LOGIC;
        cuenta: out STD_LOGIC_VECTOR(N-1 DOWNT0 0);
        saturacion: out STD_LOGIC);
END contador;

ARCHITECTURE contador_arch OF contador IS
BEGIN
    . . . . .
END contador;
```

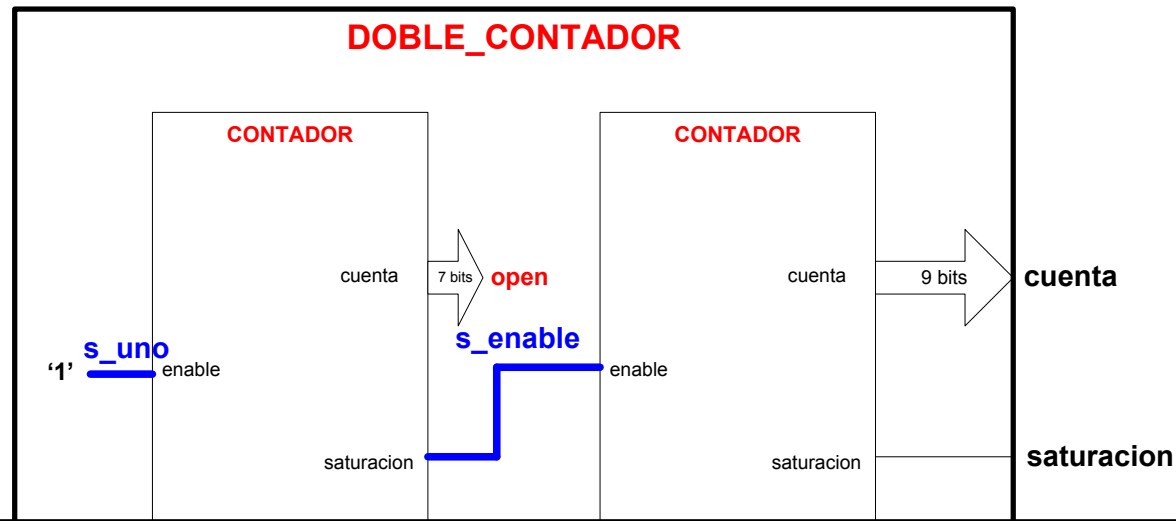
Previamente debemos haber diseñado el nivel de jerarquía inferior

La sección ARCHITECTURE. Descripción de un esquema. Ejemplo



PASO 1: Definir la entidad de DOBLE_CONTADOR

La sección ARCHITECTURE. Descripción de un esquema. Ejemplo

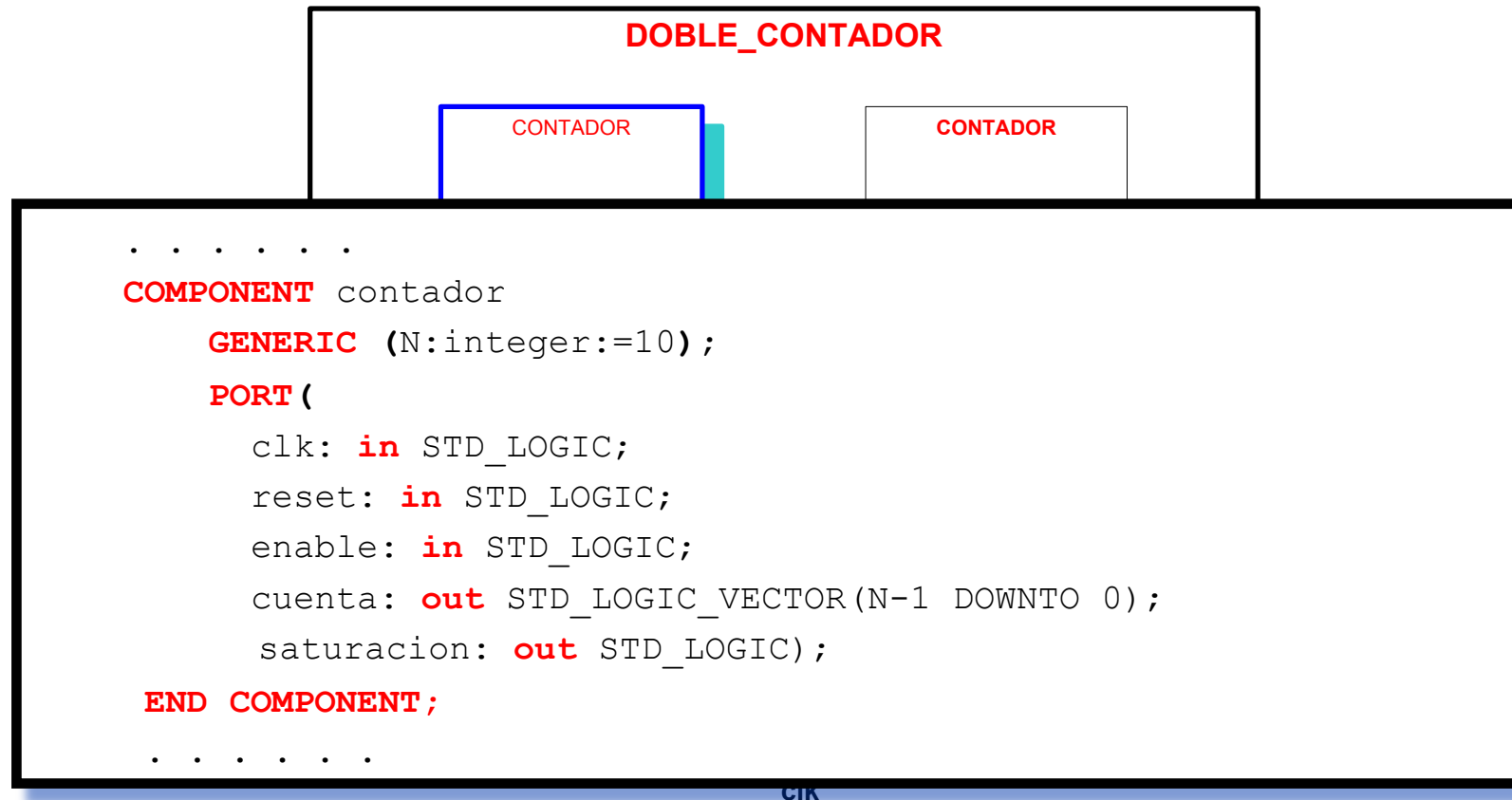


```
ARCHITECTURE doble_contador_arch OF doble_contador IS  
SIGNAL s_enable, s_uno: STD_LOGIC;  
. . . . .
```

clk

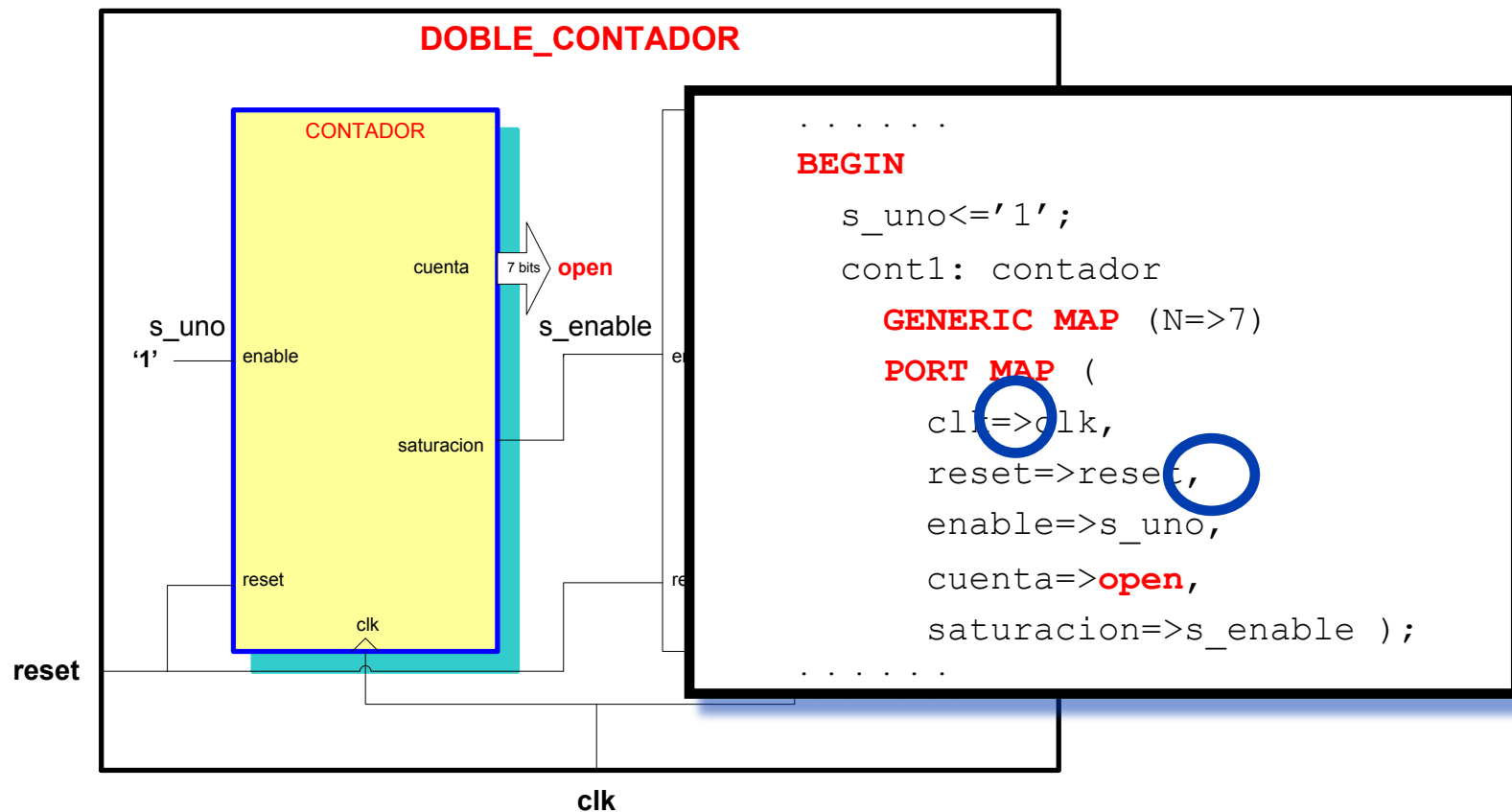
PASO 2: Declarar las señales necesarias (cables que no salen al exterior)

La sección ARCHITECTURE. Descripción de un esquema. Ejemplo



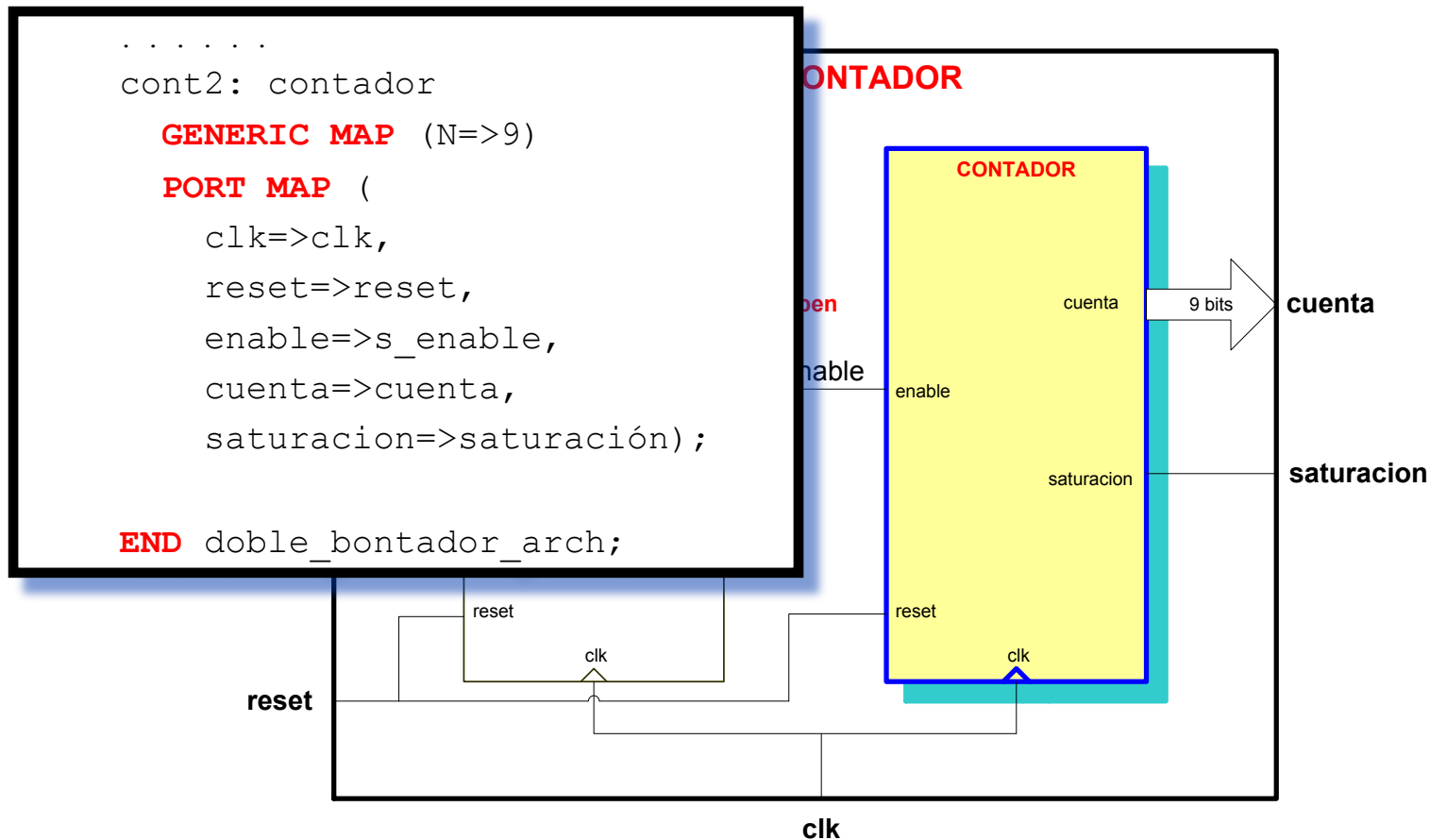
PASO 3: Declarar los componentes necesarios (entidades del nivel de jerarquía inferior)

La sección ARCHITECTURE. Descripción de un esquema. Ejemplo



PASO 4: Realizar la instancia (llamada) al primer contador. (dentro del BEGIN)

La sección ARCHITECTURE. Descripción de un esquema. Ejemplo



PASO 4: Realizar la instancia (llamada) al segundo contador.

DEBERES

- ◆ La próxima clase será la más importante de VHDL. **REPASAR LO VISTO HASTA AHORA ANTES DE LA CLASE.**
- ◆ Instalar el software de la asignatura.
- ◆ Usaremos el ISE, de Xilinx
- ◆ Página de descarga:

<http://www.xilinx.com/support/download/index.htm>

Actualización Historial Favoritos Ventana Ayuda

Xilinx: Downloads

http://www.xilinx.com/support/download/index.htm

Apple Yahoo! Google Maps YouTube Wikipedia News (353) Popular

Version	Device Pack - 14.4 Product Update	
2012.4 - 14.4	All Platforms (TAR/GZ - 322.40 MB) MD5 Sum Value: c8a13ea70a053bae90ca0c314fc4b1c3	Download Type: Product Update Last Updated: 02/04/2013 Answers: 54044
2012.3 - 14.3	Important Information The 14.4 Device Pack is an update and must be installed onto an existing 14.4 full product installation.	
2012.2 - 14.2		
14.1		
13.4		
13.3		
13.2	Multi-File Download: Vivado and ISE Design - 2012.4 Full Product Installation	
13.1	All Platforms - Split Installer Base Image - File 1/4 (TAR/GZ - 2.68 GB) MD5 Sum Value: 56835ce1a119f7e91be333af764b9d2e	Download Includes: Vivado Design Suite (All Editions) Vivado High Level Synthesis (HLS) Documentation Navigator ISE WebPACK (Free) ISE Design Suite (All Editions) System Generator for DSP Platform Studio and Embedded Development Kit (EDK) Software Development Kit (SDK) Lab Tools: Standalone Installation
12.4	Install Data A - File 2/4 (ZIP - 1.93 GB) MD5 Sum Value: 9d519d01927715c4d564e95d6535fd82	
12.3	Install Data B - File 3/4 (ZIP - 1.88 GB) MD5 Sum Value: 1edb704c27632a3ea1b5bb484713576d	
12.2	Install Data C - File 4/4 (ZIP - 1.84 GB) MD5 Sum Value: 8bf42fc385e3ff6f6e4bc7f5feeed8a1	
12.1	Important Information NEW! Download Smaller Files: If you have trouble downloading large files, try the new multiple file download above. We've split the Vivado/ISE Design Suite Installer into four smaller pieces. All four files must be downloaded prior to installation. After downloading, to install: 1. Un-tar the Split Installer Base Image into a temporary directory of your choosing 2. Copy the three Install Data files into this temporary directory 3. Run xsetup and use the default settings on the Select Download Location Directory dialogue	
11.5		
11.4		
11.3		
11.2		
11.1		
10.1 SP3	Vivado and ISE Design Suites - 2012.4 Full Product Installation	
10.1	All Platforms (TAR/GZ - 8.27 GB) MD5 Sum Value: f76dc701ed763826b1e79ef26a4ef170	Download Includes: Vivado High Level Synthesis (HLS) Documentation Navigator ISE WebPACK (Free) ISE Design Suite (All Editions) System Generator for DSP Software Development Kit (SDK) Lab Tools: Standalone Installation Vivado Design Suite (All Editions)
Archive	Full Installer for Windows (TAR/GZ - 6.20 GB) MD5 Sum Value: e2d01b68f101265d0d4f4bd62943c431	
	Full Installer for Linux (TAR/GZ - 6.44 GB) MD5 Sum Value: a8051942d71db3b42ce426d7ba3f374b	

Se ha producido un error al abrir la página. Para más detalles, seleccione Ventana > Actividad.