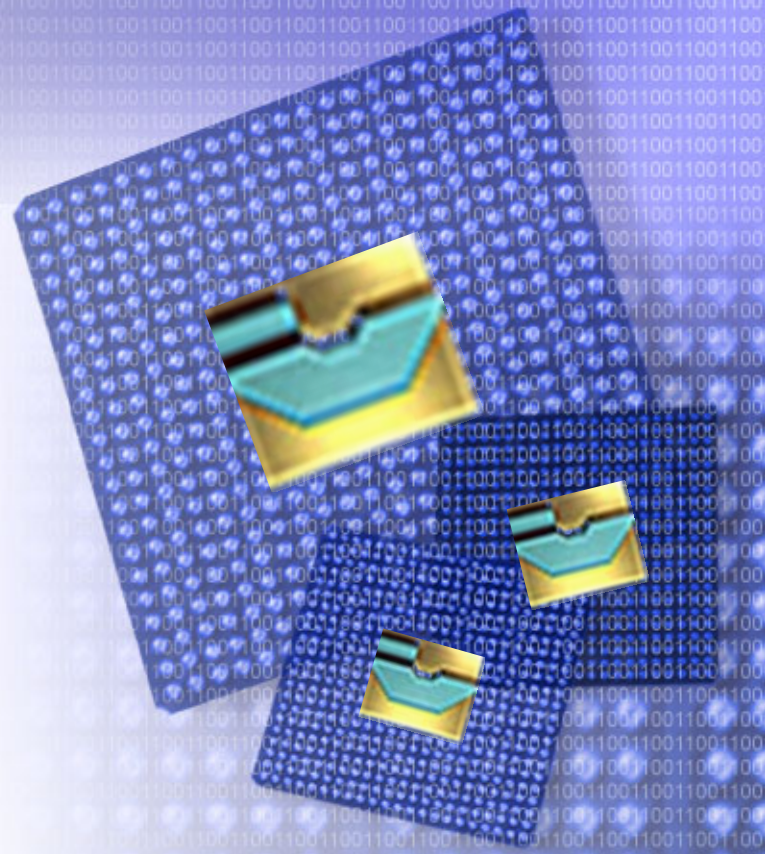
Universitat d'Alacant
Universidad de Alicante

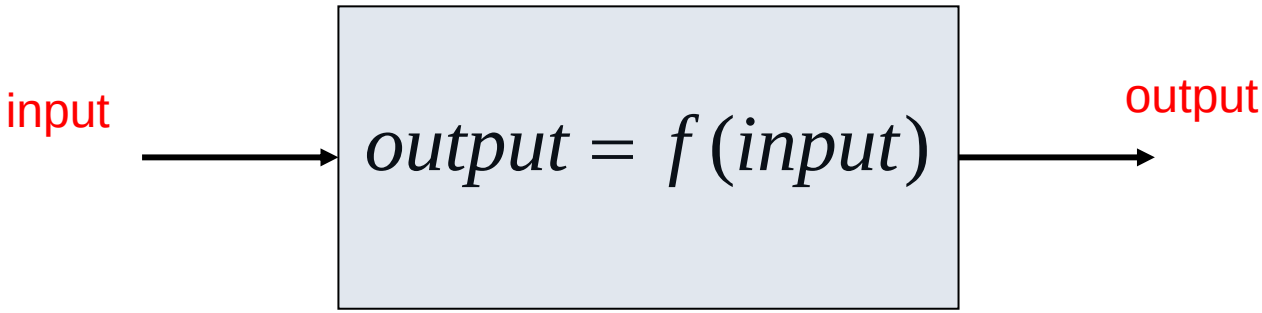


VHDL. Lenguaje de descripción hardware

Modelado combinacional

Modelado combinacional

- El modelado de sistemas combinacionales es muy sencillo, puesto que en todo momento las salidas **sólo dependen** del estado actual de las entradas, es decir, el *circuito no tiene memoria*.



Modelado combinacional

- Un circuito combinacional puede tener multitud de entradas y salidas, por lo tanto:
  - Necesitar miles o millones de términos canónicos para describirlo matemáticamente.
  - Tablas de verdad que resultan inmanejables.
- La clave para el desarrollo de estos sistemas es el pensamiento estructurado
  - Un sistema complejo se concibe como un conjunto de subsistemas más pequeños, cada uno de los cuales tiene una descripción más sencilla.
- Estos bloque combinacionales constituyen las estructuras básicas con las que se construyen los sistemas procesadores
  - Multiplexores, decodificadores, comparadores, sumadores, etc…

Descripción VHDL de sistemas combinacionales (1/3)

Para describir circuitos combinacionales utilizaremos sentencias de asignación concurrentes y procesos.

- **Sentencias de asignación concurrentes.** Van fuera del proceso. Esto se debe a que en una declaración concurrente no importa el orden en el que se escriban las señales, ya que el resultado para una determinada función sería el mismo. Se utilizarán tres tipos de sentencias de asignación concurrentes:

Descripción VHDL de sistemas combinacionales (2/3)

- **Sentencias de asignación simples, por ejemplo**:
  salida1 <= A and B;

- **Sentencias de asignación condicionales, por ejemplo**:
  salida2 <= A when S = ‘1’ else B;

- **Sentencias de selección, por ejemplo:**
  with S select
  salida3 <= ‘1’ when “00”;
  ‘0’ when others;

- **Siempre hay que evitar la realimentación combinacional**, ya que esto modela elementos de memoria asíncronos. Es decir, hay que evitar que una misma señal aparezca en el lado izquierdo y en el lado derecho de la sentencia de asignación:
  Y <= Y nand X;

Descripción VHDL de sistemas combinacionales (3/3)

- **Procesos.** Se debe respetar las siguientes reglas:
  - Si una señal es leída en el interior de un proceso, es decir, aparece en el lado derecho de una sentencia de asignación o en una condición, debe aparecer en su lista de sensibilidad.
  - Si a una señal se le asigna un valor de forma condicional (con sentencias if o case) nos debemos asegurar que no existen condiciones para las cuales el valor de la señal no se ha definido en el código.
  - En VHDL las señales mantienen su valor hasta que se les asigna uno nuevo; por lo tanto, si para una señal dada, su valor está indefinido, el compilador sintetiza un latch (biestable) que almacena la señal.

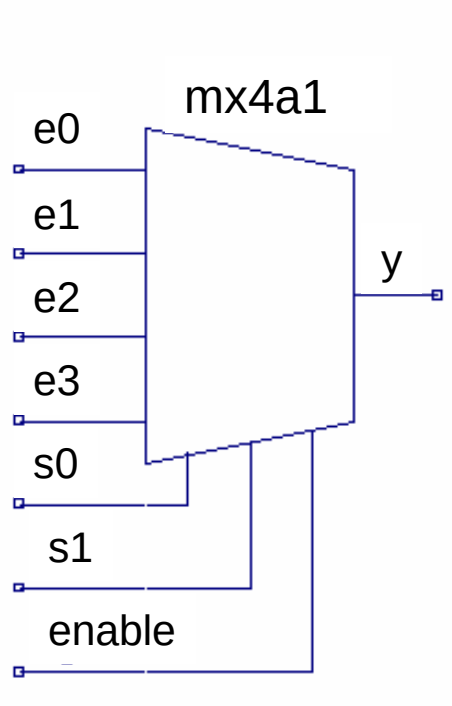
Multiplexores

- Circuitos que tienen *n* entradas de datos y *una* salida de datos, por lo tanto necesitan *m* entradas de selección, donde

  m es

$$n = 2^m \Rightarrow m = \frac{\log n}{\log 2}$$

  Habitualmente también poseen *una entrada de habilitación*



Multiplexores

- La función *f* que describe el comportamiento de un multiplexor de dos entradas (I_0 , I_1) , una salida y una entrada de selección (S), se puede describir mediante la tabla 1.
- Aplicando Karnaugh se obtiene:

$$f = \bar{S}I_0 + SI_1$$

tabla 1

S	I_1	I_0	f
0	0	**0**	**0**
0	0	**1**	**1**
0	1	**0**	**0**
0	1	**1**	**1**
1	**0**	0	**0**
1	**0**	1	**0**
1	**1**	0	**1**
1	**1**	1	**1**

Multiplexores

- La función *f* que describe el comportamiento de un multiplexor también se puede expresar con la tabla 2, con una única entrada, de selección.
- Con esta tabla y aplicando el teorema de expansión

tabla 2

S	f
0	I_0
1	I_1

$$f(b_1, b_2, ..., b_n) = b_1 \bullet f(1, b_2, ..., b_n) + \overline{b}_1 \bullet f(0, b_2, ..., b_n)$$

$$f(b_1, b_2, ..., b_n) = [b_1 + f(0, b_2, ..., b_n)] \bullet [\overline{b}_1 + f(1, b_2, ..., b_n)]$$

$$f(S) = \overline{S} \bullet F(0) + S \bullet F(1)$$

F(1) es la salida del multiplexor cuando S=1, es decir $F(1)=I_1$
F(0) es la salida del multiplexor cuando S=0, es decir $F(0)=I_0$
por lo tanto obtenemos:

$$f = \overline{S}I_0 + SI_1$$

Multiplexores

- Obtener las ecuaciones de un multiplexor de cuatro entradas, una salida y dos entradas de selección

$$f(S_1,S_0) = \overline{S}_1 \bullet f(0,S_0) + S_1 \bullet f(1,S_0)$$

tabla 2

S_1	S_0	f
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- Aplicando de nuevo el teorema de expansión a las funciones $f(1,S_0)$ y $f(0,S_0)$

$$f(1,S_0) = \overline{S}_0 \bullet f(1,0) + S_0 \bullet f(1,1)$$
$$f(0,S_0) = \overline{S}_0 \bullet f(0,0) + S_0 \bullet f(0,1)$$

$$f(S_1,S_0) = \overline{S}_1 \bullet f(0,S_0) + S_1 \bullet f(1,S_0) =$$
$$= \overline{S}_1 \bullet \left[\overline{S}_0 \bullet f(0,0) + S_0 \bullet f(0,1)\right] + S_1 \bullet \left[\overline{S}_0 \bullet f(1,0) + S_0 \bullet f(1,1)\right] =$$
$$= \overline{S}_1 \bullet \overline{S}_0 \bullet f(0,0) + \overline{S}_1 \bullet S_0 \bullet f(0,1) + S_1 \bullet \overline{S}_0 \bullet f(1,0) + S_1 \bullet S_0 \bullet f(1,1) =$$
$$= \overline{S}_1 \bullet \overline{S}_0 \bullet I_0 + \overline{S}_1 \bullet S_0 \bullet I_1 + S_1 \bullet \overline{S}_0 \bullet I_2 + S_1 \bullet S_0 \bullet I_3$$

Multiplexores (2/3)

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

entity mx4a1 is
    Port ( e0              : in std_logic;
           e1              : in std_logic;
           e2              : in std_logic;
           e3              : in std_logic;
           enable          : in std_logic;
           s               : in std_logic_vector(1 downto 0);
           y               : out std_logic
         );
end mx4a1;
```

Multiplexores (3/3)

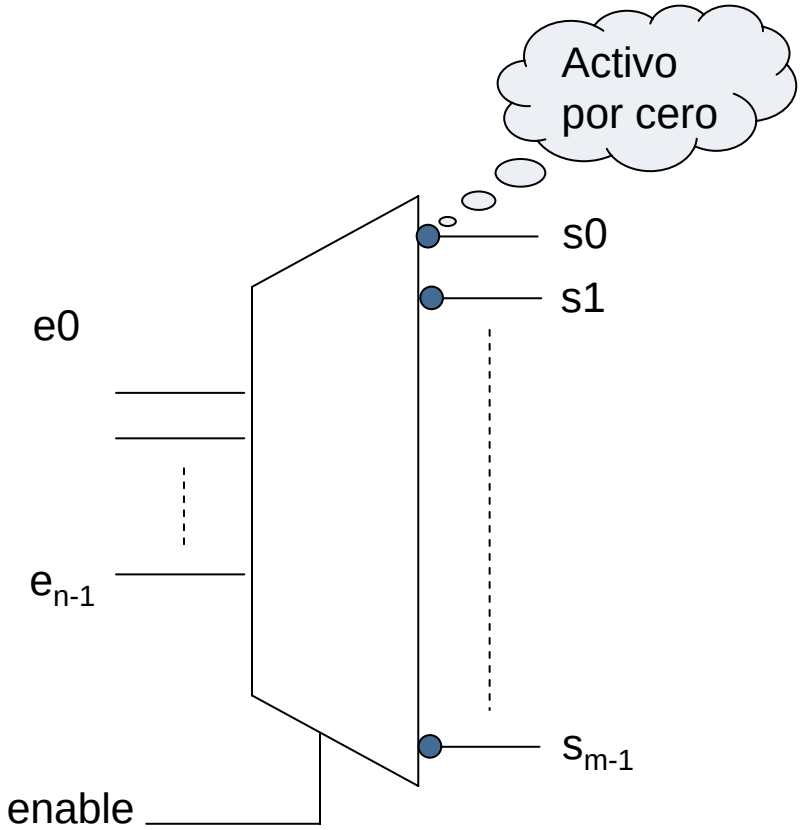
```
   architecture Comportamiento of mx4a1 is
begin
    multiplexor: process(e0,e1,e2,e3,s)
    begin
        if enable ='0' then
                y <= (others=>'Z');
        else
                case s is
                        when "00" => y <= e0;
                        when "01" => y <= e1;
                        when "10" => y <= e2;
                        when others  => y <= e3;
                end case;
        end if;
    end process multiplexor;
end Comportamiento;
```

Decodificadores (1/5)

- Es un circuito lógico con *n* entradas y *m* salidas, donde *m* es

$$m = 2^n$$

- Podemos encontrar decodificadores con *entrada* de *habilitación*, activos por cero o activos por uno



Decodificadores (2/5)

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

entity deco3a8 is

    Port ( e0 : in std_logic;
           e1 : in std_logic;
           e2 : in std_logic;
           s0 : out std_logic;
           s1 : out std_logic;
           s2 : out std_logic;
           s3 : out std_logic;
           s4 : out std_logic;
           s5 : out std_logic;
           s6 : out std_logic;
           s7 : out std_logic;
         enable : in std_logic);
end deco3a8;
```

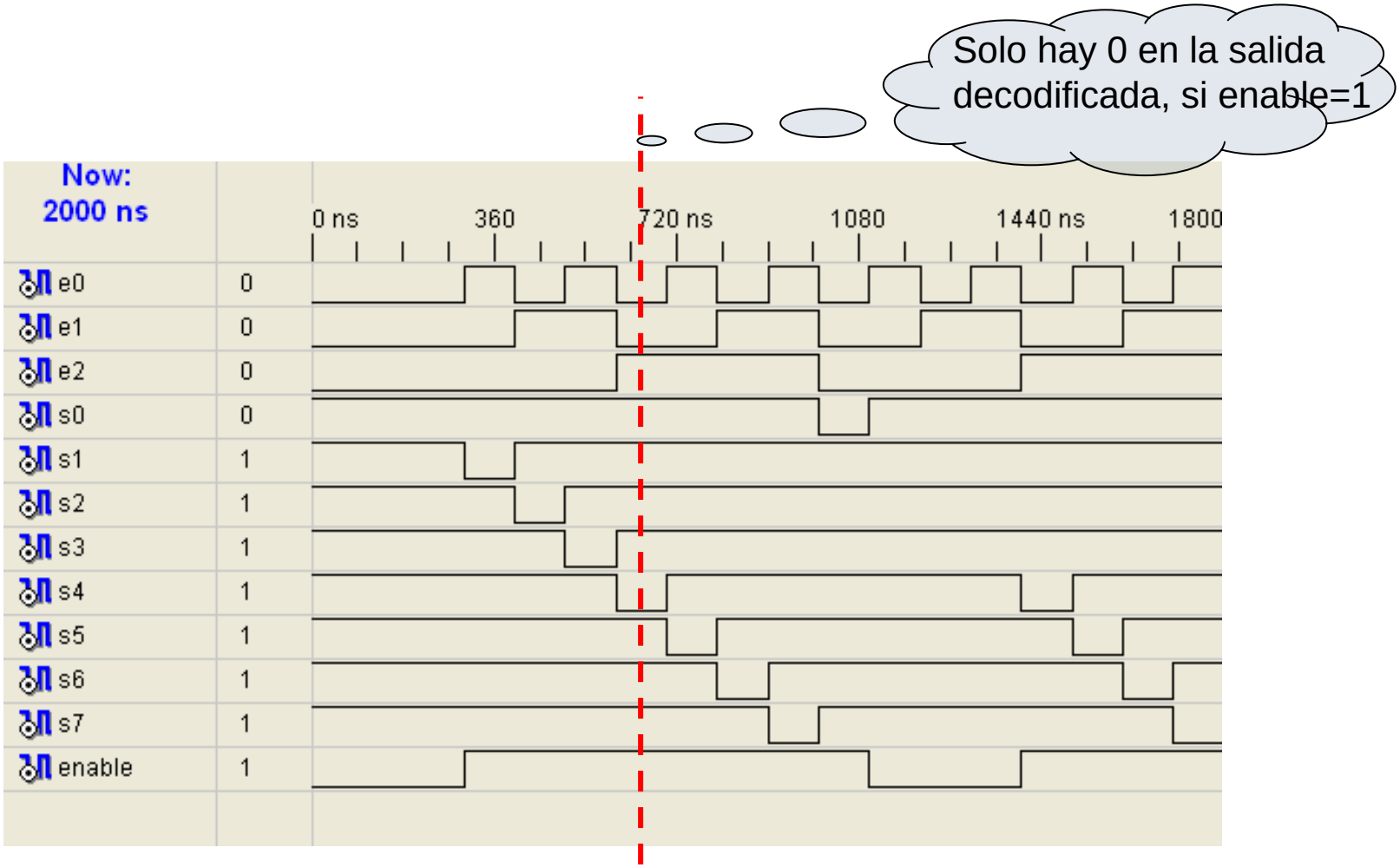
Decodificadores (3/5)

```
architecture Comportamiento of deco3a8 is
begin
PROCESS(enable, e0, e1, e2)
          variable  cod_ent: std_logic_vector(2 DOWNTO 0);
begin
          if enable='0' then
                    s7<= '1'; s6<= '1'; s5<= '1'; s4<= '1';
                    s3<= '1'; s2<= '1'; s1<= '1'; s0<= '1';
           else
                    cod_ent:= e2&e1&e0;
                    CASE cod_ent IS
                    WHEN "000" =>
                              S7<= '1'; S6<= '1'; S5<= '1'; S4<= '1';
                              S3<= '1'; S2<= '1'; S1<= '1'; S0<= '0';
                    WHEN "001" =>
                              S7<= '1'; S6<= '1'; S5<= '1'; S4<= '1';
                              S3<= '1'; S2<= '1'; S1<= '0'; S0<= '1';
                    WHEN "010" =>
                              S7<= '1'; S6<= '1'; S5<= '1'; S4<= '1';
                              S3<= '1'; S2<= '0'; S1<= '1'; S0<= '1';
                    WHEN "011" =>
                              S7<= '1'; S6<= '1'; S5<= '1'; S4<= '1';
                              S3<= '0'; S2<= '1'; S1<= '1'; S0<= '1';
                    WHEN "100" =>
                              S7<= '1'; S6<= '1'; S5<= '1'; S4<= '0';
                              S3<= '1'; S2<= '1'; S1<= '1'; S0<= '1';
```

Decodificadores (4/5)

```
        WHEN "101" =>
                S7<= '1'; S6<= '1'; S5<= '0'; S4<= '1';
                S3<= '1'; S2<= '1'; S1<= '1'; S0<= '1';
        WHEN "110" =>
                S7<= '1'; S6<= '0'; S5<= '1'; S4<= '1';
                S3<= '1'; S2<= '1'; S1<= '1'; S0<= '1';
        WHEN "111" =>
                S7<= '0'; S6<= '1'; S5<= '1'; S4<= '1';
                S3<= '1'; S2<= '1'; S1<= '1'; S0<= '1';
        WHEN OTHERS =>
                S7<= '1'; S6<= '1'; S5<= '1'; S4<= '1';
                S3<= '1'; S2<= '1'; S1<= '1'; S0<= '1';
        END CASE;
    END IF;
END PROCESS;
end Comportamiento;
```

Decodificadores (5/5)

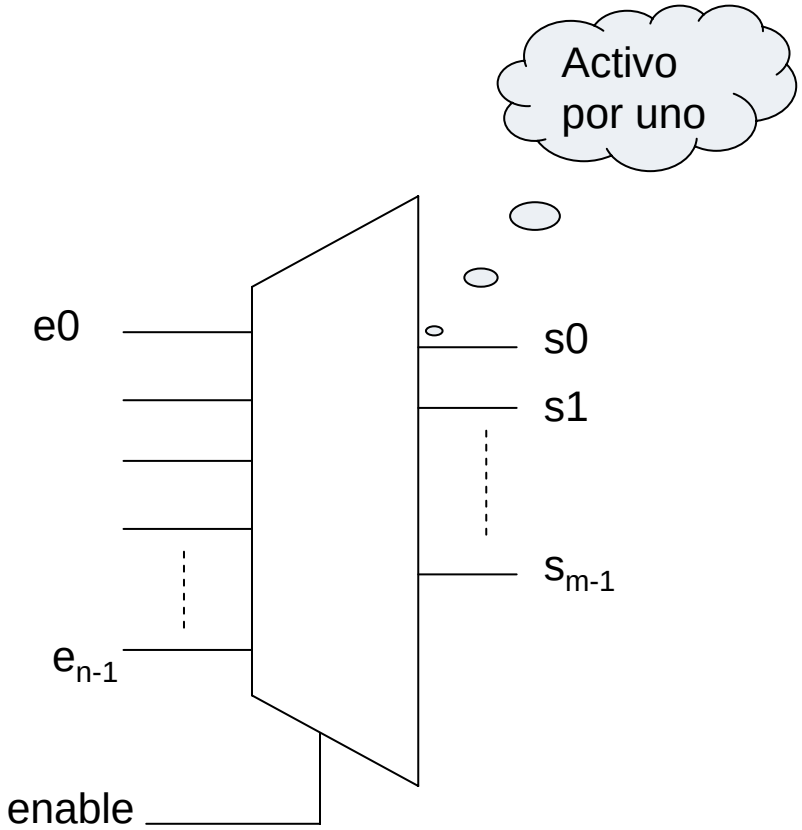


Codificadores (1/4)

- Es un circuito lógico con *n* entradas y *m* salidas, donde *m* es

$$n = 2^m \Rightarrow m = \frac{\log n}{\log 2}$$

- Habitualmente también poseen *una entrada de habilitación*, pueden ser activos por *cero* o por *uno*



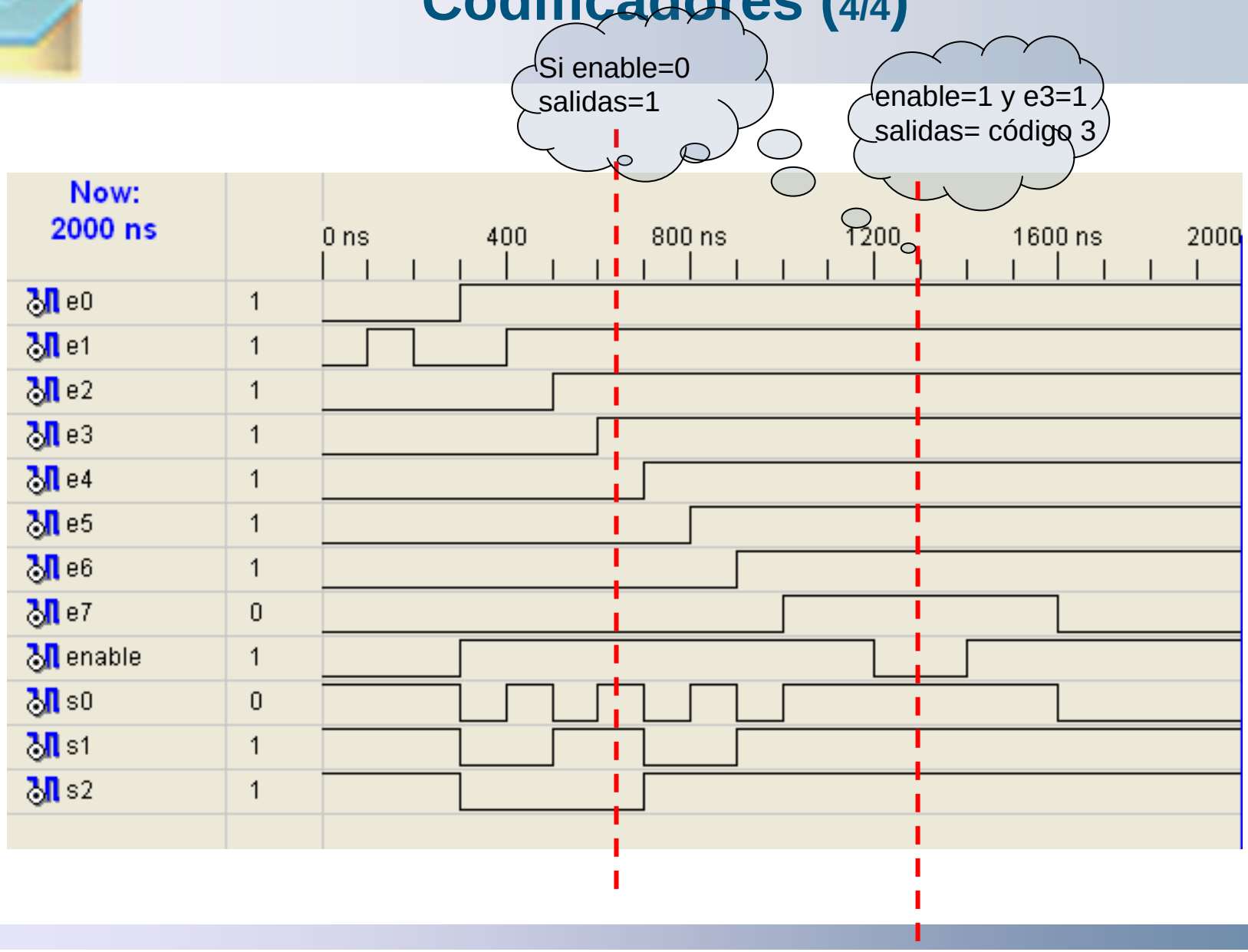
Codificadores (2/4)

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

entity CODIFICADOR is
    Port ( e0 : in std_logic;
           e1 : in std_logic;
           e2 : in std_logic;
           e3 : in std_logic;
           e4 : in std_logic;
           e5 : in std_logic;
           e6 : in std_logic;
           e7 : in std_logic;
           enable : in std_logic;
           s0 : out std_logic;
           s1 : out std_logic;
           s2 : out std_logic);
end CODIFICADOR;
```

Codificadores (3/4)

```
architecture Behavioral of CODIFICADOR is
begin
 process(e0,e1,e2,e3,e4,e5,e6,e7,enable)
 begin
          if enable = '0' then
                    s0 <= '1'; s1 <= '1'; s2 <= '1';
          else
-- con una estructura IF_THEN_ELSE se establece la prioridad de la codificación
                    if    e7 ='1' then
                              s2 <= '1'; s1 <= '1'; s0 <= '1';
                              elsif     e6 ='1' then
                                        s2 <= '1'; s1 <= '1'; s0 <= '0';
                              elsif     e5 ='1' then
                                        s2 <= '1'; s1 <= '0'; s0 <= '1';
                              elsif     e4 ='1' then
                                        s2 <= '1'; s1 <= '0'; s0 <= '0';
                              elsif     e3 ='1' then
                                        s2 <= '0'; s1 <= '1'; s0 <= '1';
                              elsif     e2 ='1' then
                                        s2 <= '0'; s1 <= '1'; s0 <= '0';
                              elsif     e1 ='1' then
                                        s2 <= '0'; s1 <= '0'; s0 <= '1';
                              elsif     e0 ='1' then
                                        s2 <= '0'; s1 <= '0'; s0 <= '0';
                    end if;
           end if;
 end process;
end Behavioral;
```

Codificadores (4/4)

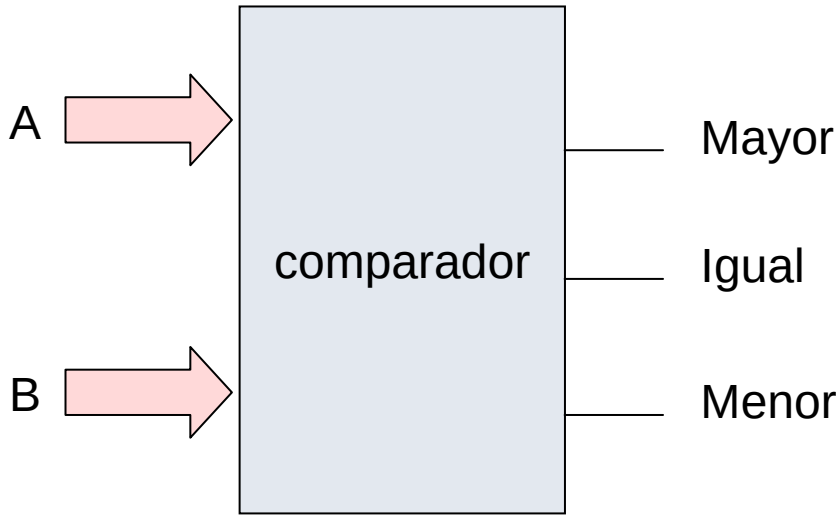


Comparadores (1/4)

- La comparación de la igualdad de dos palabras binarias es una operación comúnmente utilizada en sistemas de cómputo e interfaces de dispositivos. A un circuito que compara dos palabras binarias e indica si son iguales se le conoce como comparador.

  Algunos comparadores también indican una relación aritmética (mayor, menor o igual) entre las palabras.

  Estos dispositivos se denominan comparadores de magnitud.



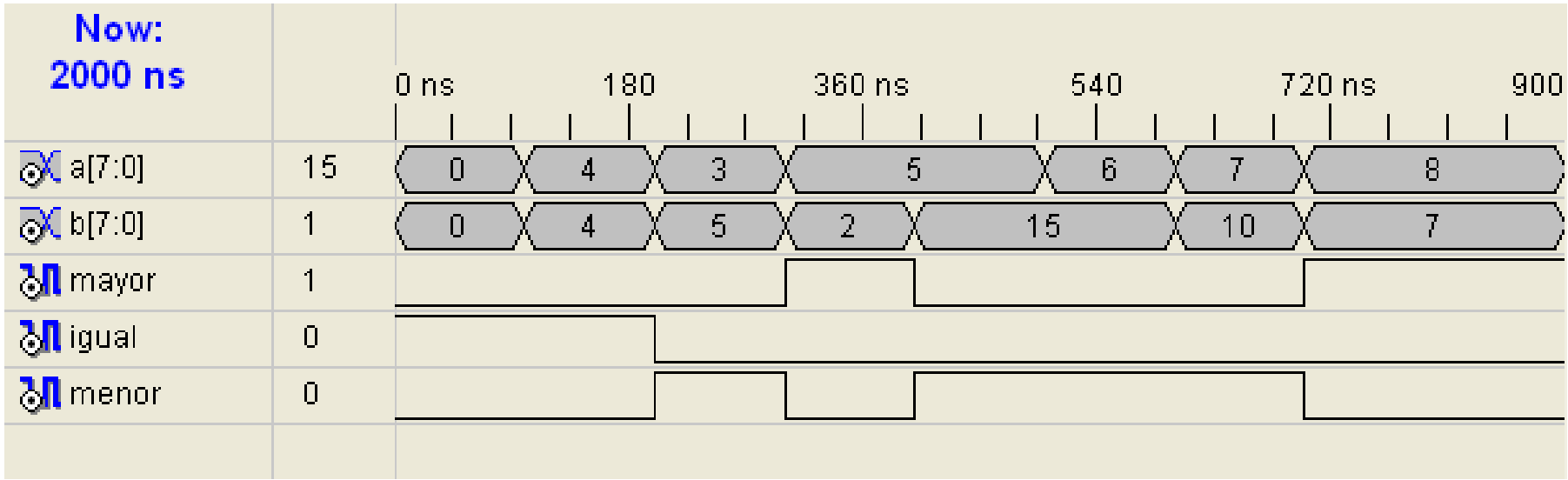
Comparadores (2/4)

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

entity comparador is
        generic (Nbits:integer:=8);
    Port ( a : in std_logic_vector(Nbits-1 downto 0);
           b : in std_logic_vector(Nbits-1 downto 0);
           mayor : out std_logic;
           igual : out std_logic;
           menor : out std_logic);
end comparador;
```

Comparadores (3/4)

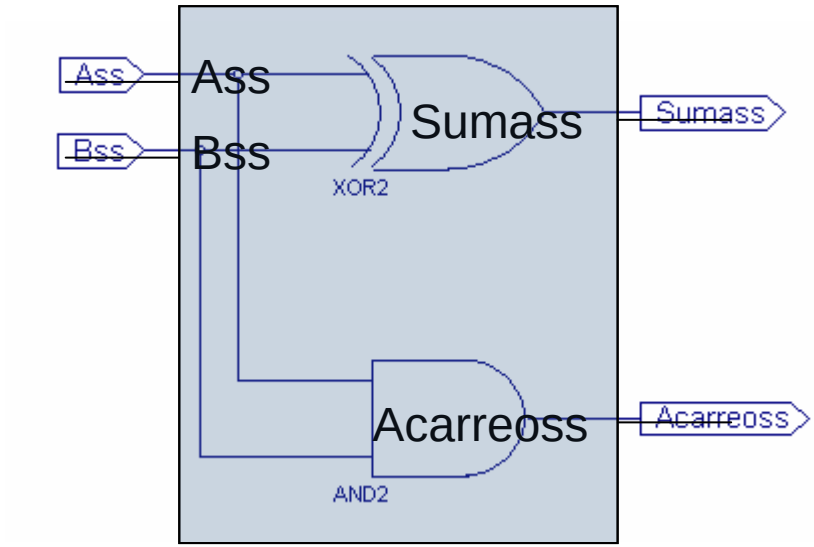
```
architecture Behavioral of comparador is
begin
  process(a,b)
  begin
        if (a=b) then
                igual <='1';
        else
                igual <='0';
        end if;
        if (a>b) then
                mayor <='1';
        else
                mayor <='0';
        end if;
        if (a<b) then
                menor <='1';
        else
                menor <='0';
        end if;
end process;
end Behavioral;
```

Comparadores (4/4)



Sumadores

Este ejemplo permite aprender la jerarquía de los diseños



A	B	Suma	Acarreo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

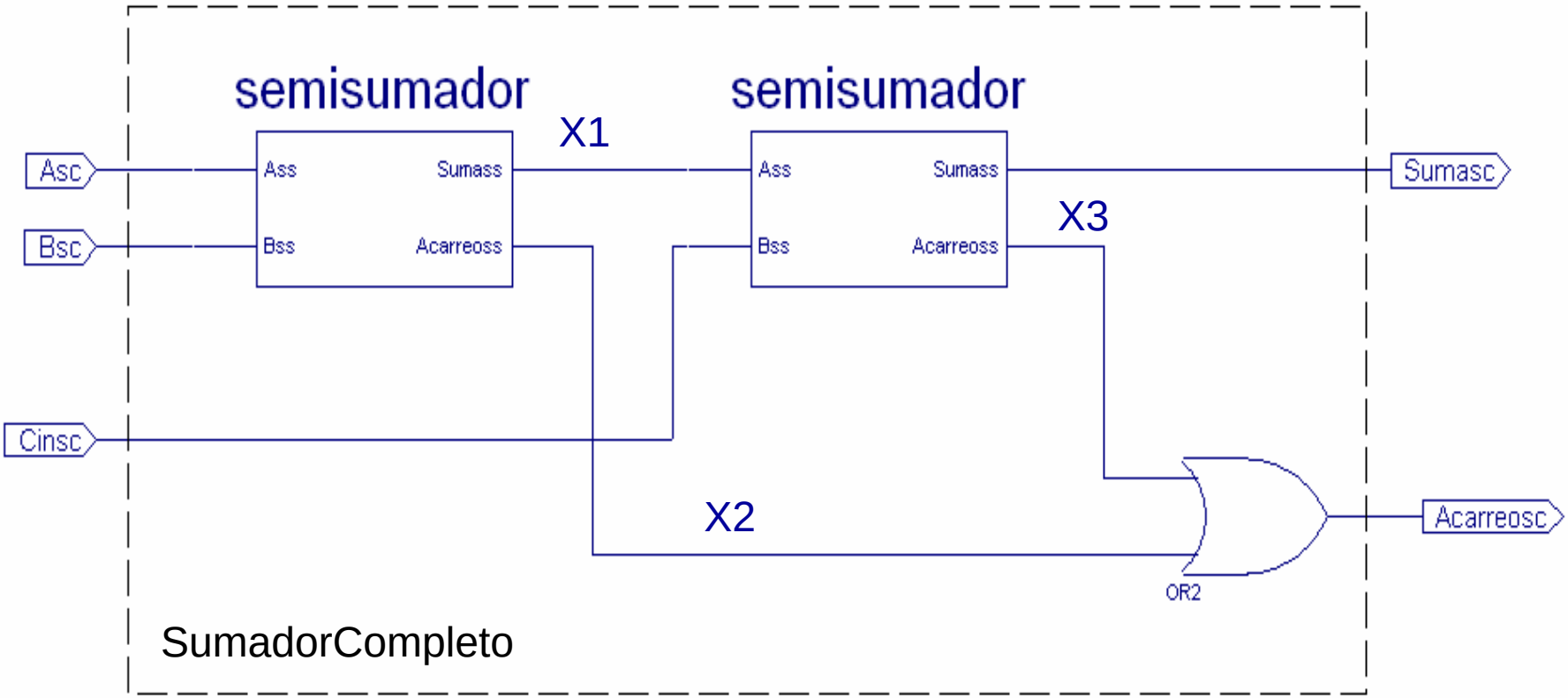
Sumadores

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SemiSumador IS
PORT ( Ass, Bss             :IN std_logic;
       Sumass, Acarreoss    : OUT std_logic);
END SemiSumador ;

ARCHITECTURE SemiSumador OF SemiSumador IS
BEGIN
       Sumass        <= Ass xor Bss;
       Acarreoss     <= Ass and Bss;
END SemiSumador ;
```

Sumadores

SumadorCompleto

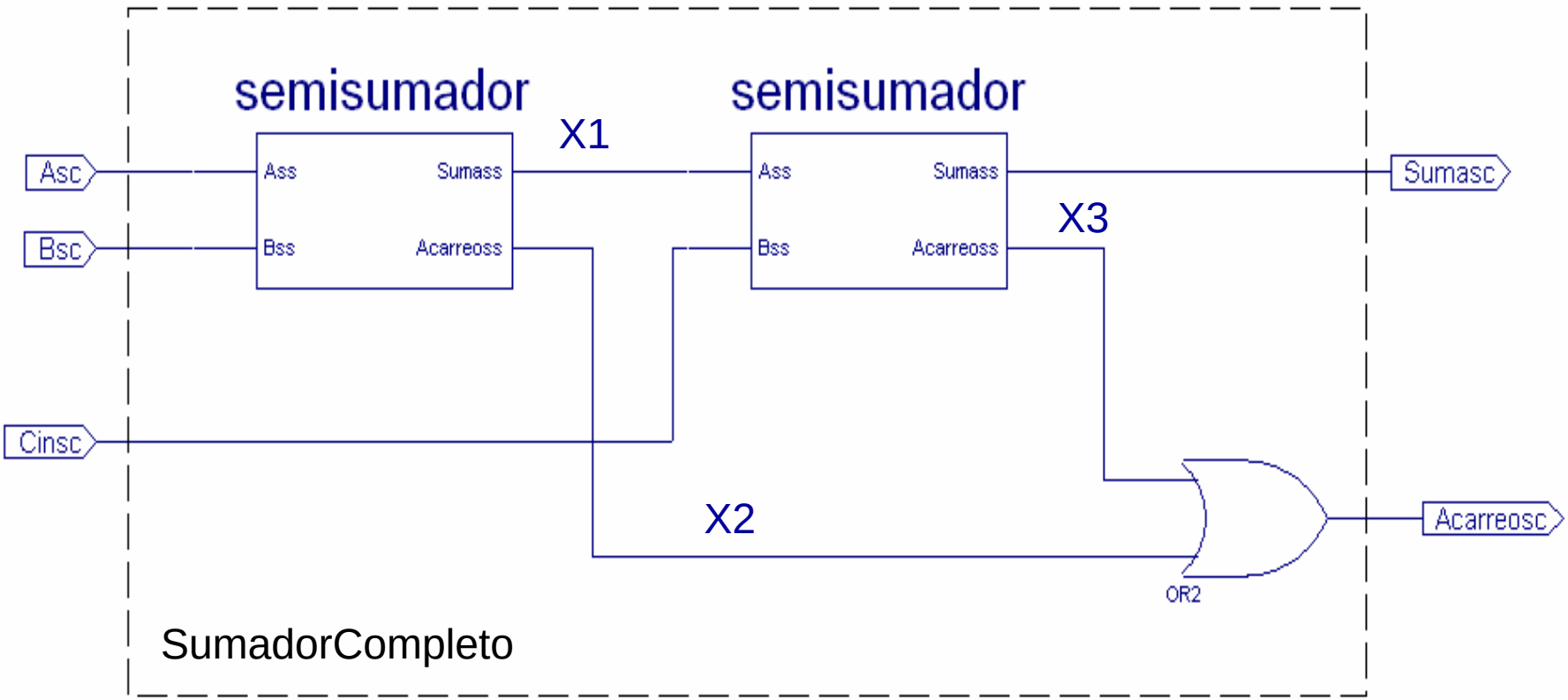
A	B	Cin	Suma	Acarreo
0	0	0	0	0



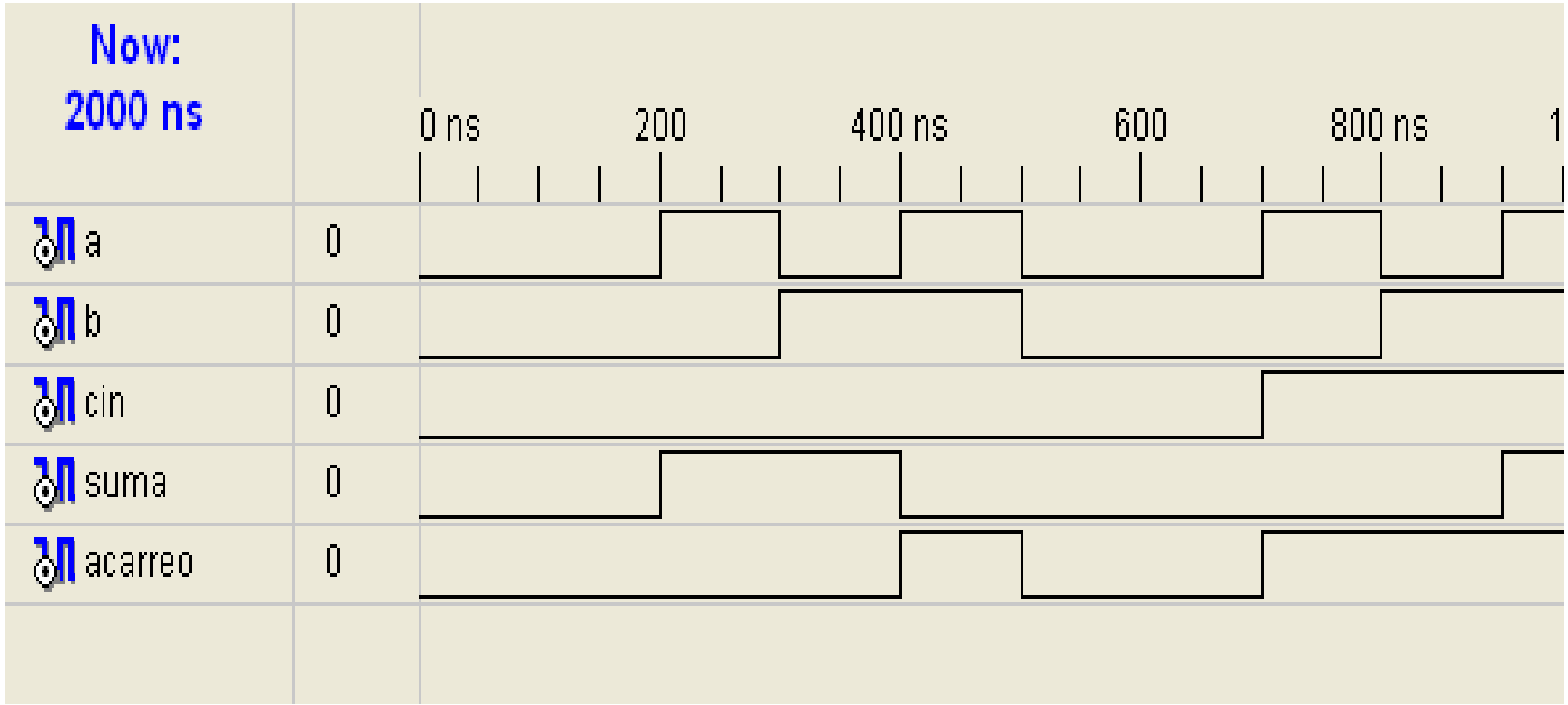
Sumadores

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-------------------------------------------------------
-- Declaración de la entidad y arquitectura del sumador.
-------------------------------------------------------
ENTITY SumadorCompleto IS
PORT ( Asc, Bsc, Cinsc          : IN std_logic;
       Sumasc, Acarreosc        : OUT std_logic);
END SumadorCompleto ;

ARCHITECTURE SumadorCompleto OF SumadorCompleto IS

-- Se declara el uso de componentes tipo SemiSumador.
COMPONENT SemiSumador
--Se indica los puertos que tiene el componente.

PORT( Ass, Bss                  : IN std_logic;
      Sumass, Acarreoss         : OUT std_logic);
END COMPONENT;
```

Sumadores

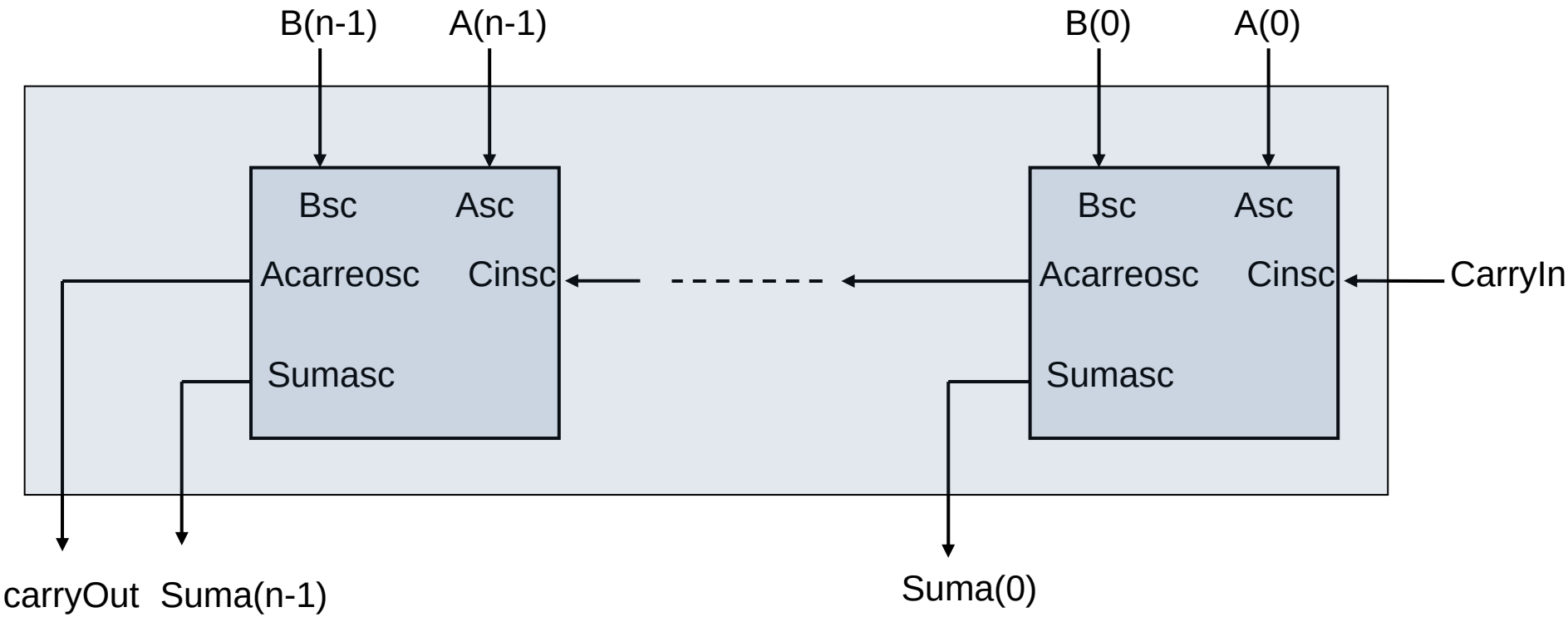


Sumador Completo simulación



Sumador de *n* bits

- La mejor forma de construir un sumador serie de *n* bits consiste en aprovechar la posibilidad de replicación de estructuras que posibilita el lenguaje VHDL mediante la instrucción GENERATE.



Sumador de *n* bits

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

ENTITY SumadorNbits IS
  GENERIC(Nbits: integer:=8);
  PORT ( A, B          :IN std_logic_vector(Nbits-1 DOWNTO 0);
       CarryIn         :IN std_logic;
       Suma            :OUT std_logic_vector(Nbits-1 DOWNTO 0);
       CarryOut        :OUT std_logic);
END SumadorNbits;

ARCHITECTURE SumadorNbits OF SumadorNbits IS
  -- Se declara el uso de componentes tipo SumadorCompleto
  COMPONENT SumadorCompleto
    --Se indica los puertos que tiene el componente.
      PORT ( A, B, Cin       : IN std_logic;
             Suma, Acarreo   : OUT std_logic);
  END COMPONENT;
-- Señal auxiliar para la propagación de acarreos
  SIGNAL CarryTemp            : std_logic_vector(Nbits DOWNTO 0);
```

Sumador de *n* bits

```
--Comienzo de la arquitectura
BEGIN
-- Se inicializa el acarreo.
       CarryTemp(0)<=CarryIn;
-- Se genera el conjunto de SumadoresCompletos

   GeneraSC: FOR i IN 0 TO Nbits-1 GENERATE
   SC: sumadorcompleto PORT MAP (A=>A(i), B=>B(i),
                                 Cin=>CarryTemp(i),
                                 Suma=>Suma(i),
                                 Acarreo=>CarryTemp(i+1));
   END GENERATE;

-- Se genera el acarreo de salida.
       CarryOut<=CarryTemp(Nbits-1);

END SumadorNbits;
```

Sumador de *n* bits simulación