LABORATORIO DE CIRCUITOS DIGITALES
(2005-II)
SEGUNDA CLASE DE VHDL

- ✓ *TIPOS y MODOS DE DATOS*
- ✓ *DESCRIPCIÓN CONCURRENTE*
  - Sentencias de asignación: with – select, when - else
- ✓ *DESCRIPCIÓN COMPORTAMENTAL*
  - Procesos asíncronos (decoder, mux, demux, etc.)

Instructor: Daniel Llamocca

✓ *TIPOS Y MODOS DE DATO*

- **Tipo:** Hay muchas formas de definir tipos de datos, pero la IEEE lo ha estandarizado en un paquete llamado *std_logic_1164*. El cual contiene los siguientes tipos:
  - std_logic, std_logic_vector, std_logic_2d
    - El tipo std_logic define 9 posibles estados:
      - 'U' → Sin inicializar
      - 'X' → Forzado a desconocido
      - '0' → 0 forzado
      - '1' → 1 forzado
      - 'Z' → Alta impedancia
      - 'W' → Desconocido débil
      - 'L' → Cero débil
      - 'H' → Uno débil
      - '-' → No importa
- También existen:
  - integer
  - array
  - definidos por el usuario

✓ Uso de std_logic_vector

- En el ejemplo se aprecia cómo se define una señal de tipo *std_logic_vector* y cómo es que se trabaja con este tipo de datos:

```
library ieee;
use ieee.std_logic_1164.all;

entity muestra is
 port ( A: in std_logic_vector (3 downto 0);
        -- A: |A3|A2|A1|A0|
        y: out std_logic);
end muestra;

architecture est of muestra is
begin
 -- El circuito representa una compuerta AND
 -- de 4 entradas: A(3), A(2), A(1), A(0)
 y <= A(3) and A(2) and A(1) and A(0);
end est;
```

✓ *TIPOS Y MODOS DE DATO*

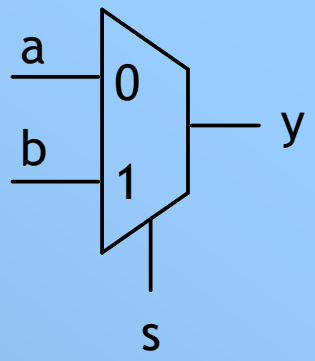
- **Modo:**
- Indica la característica física de las entradas o salidas de un circuito. En VHDL existen los siguientes modos:

  - IN → Define a un pin como entrada de un circuito
  - OUT → Define a un pin como salida de un circuito.
  - INOUT → Define un pin bidireccional. Útil cuando se trabaja con buses bidireccionales
  - BUFFER → Define a un pin como salida de un circuito, con la particularidad que esta salida puede realimentarse al circuito.

✓ *DESCRIPCIÓN CONCURRENTE*

- En esta descripción, los circuitos se especifican en forma estructural. El orden de las sentencias es irrelevante, pues éstas representan circuitos que funcionan todos a la vez. Con esta descripción, sólo pueden construirse circuitos combinacionales.
- La Descripción Horizontal (Clase VHDL 1) es un tipo básico de descripción concurrente.
- La Descripción Estructural (Clase VHDL 3) es una generalización de la descripción concurrente (el código VHDL es inherentemente concurrente).
- Se presenta ahora 2 sentencias de asignación concurrentes (with – select y when – else), que son más versátiles que las sencillas asignaciones de la descripción horizontal.

- **SENTENCIAS DE ASIGNACIÓN CONCURRENTE:**
  - **Asignación seleccionada de señal (WITH – SELECT):** Permite que se le asigne a una señal uno de varios valores en base a un criterio de selección.
  - **1er ejemplo:** MUX 2 a 1



with s select: 's' especificará el criterio de selección.

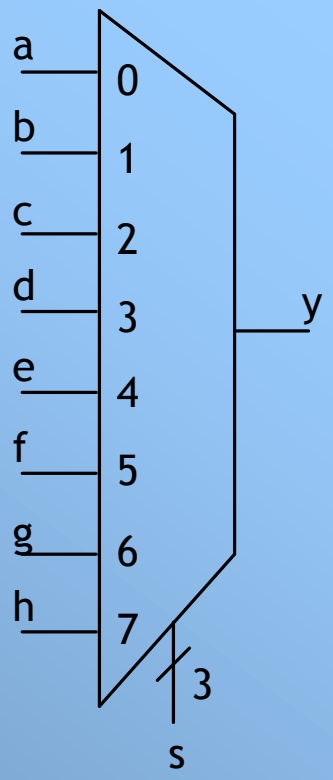
when indica el valor asignado a 'y' para cada valor de 's'.

En el último caso se usa **when others** (se requiere que se incluyan todos los casos posibles de 's', que son 9 por ser del tipo *std_logic*)

```
library ieee;
use ieee.std_logic_1164.all;

entity mux21_con is
 port ( a, b, s: in std_logic;
        y: out std_logic);
end mux21_con;

architecture est of mux21_con is
begin
 with s select
     y <= a when '0',
          b when others;
end est;
```

- Asignación seleccionada de señal (WITH - SELECT):
- 2do ejemplo:

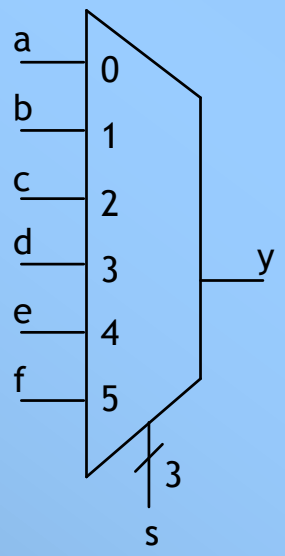
MUX 8 a 1




```
library ieee;
use ieee.std_logic_1164.all;

entity mux81_con is
 port ( a,b,c,d,e,f,g,h: in std_logic;
        s: in std_logic_vector (2 downto 0);
        y: out std_logic);
end mux81_con;

architecture est of mux81_con is
begin
 with s select
  y <= a when "000", -- notar ',' en vez de ';'
       b when "001",
       c when "010",
       d when "011",
       e when "100",
       f when "101",
       g when "110",
       h when others;
end est;
```

- **Asignación seleccionada de señal (WITH - SELECT):**
- **2do ejemplo:**

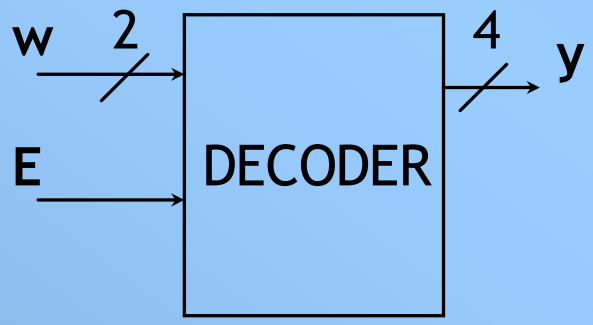
MUX 6 a 1



- Note el uso del valor ‘ - ’ **(condición de no importa)**
- Esto permite que el compilador sintetice mejor el circuito.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux61_con is
 port (a,b,c,d,e,f: in std_logic;
       s: in std_logic_vector (2 downto 0);
       y: out std_logic);
end mux61_con;

architecture est of mux61_con is
begin
 with s select
  y <= a when "000",
       b when "001",
       c when "010",
       d when "011",
       e when "100",
       f when "101",
       '-' when others;
end est;
```

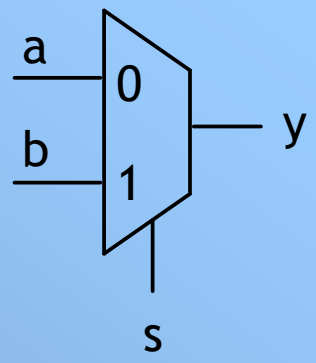
- **Asignación seleccionada de señal (WITH - SELECT):**
- **3er ejemplo:** Decodificador 2 a 4 con habilitador




```
library ieee;
use ieee.std_logic_1164.all;

entity dec2a4_con is
 port (w: in std_logic_vector (1 downto 0);
       E: in std_logic;
       y: out std_logic_vector (3 downto 0));
end dec2a4_con;

architecture est of dec2a4_con is
signal Ew: std_logic_vector (2 downto 0);
begin
 Ew <= E & w -- concatenación
 with Ew select
  y <= "0001" when "100",
       "0010" when "101",
       "0100" when "110",
       "1000" when "111",
       "0000" when others;
end est;
```

- Note el uso del operador '&' que sirve para concatenar señales. La señal 'Ew' contiene 3 bits.

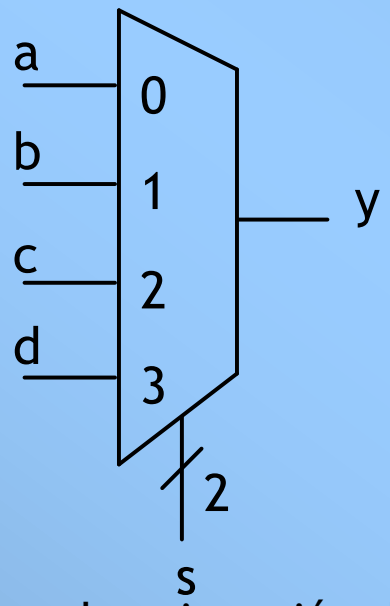
- **SENTENCIAS DE ASIGNACIÓN CONCURRENTE:**
  - **Asignación condicionada de señal (WHEN - ELSE):** En forma similar a la asignación seleccionada, esta asignación permite que una señal tome uno de varios valores en base a una condición. La sintaxis, sin embargo, es diferente y permite describir ciertos circuitos en forma más compacta.
  - **1er ejemplo:** MUX 2 a 1



- Si la condición del when es falsa, después de else se asigna un valor a 'y' ('b'). Este asignación de 'b' podría estar condicionada por otra cláusula when-else como se verá en el sgte. ejemplo.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux21_cond is
 port ( a, b, s: in std_logic;
        y: out std_logic);
end mux21_cond;

architecture est of mux21_cond is
begin
   y <= a when s = '0' else b;
end est;
```

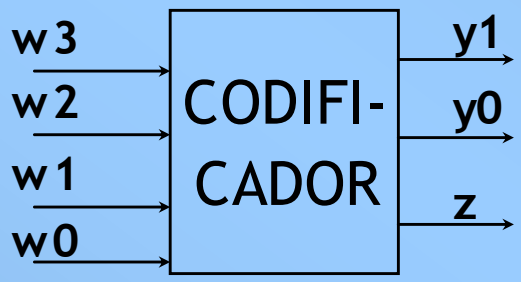
- **Asignación condicionada de señal (WHEN - ELSE):**
- **2do ejemplo:** MUX 4 a 1




```
library ieee;
use ieee.std_logic_1164.all;

entity mux41_cond is
 port (a, b, c, d: in std_logic;
       s: in std_logic_vector (1 downto 0);
       y: out std_logic);
end mux41_cond;

architecture est of mux41_cond is
begin
   y <= a when s = "00" else
        b when s = "01" else
        c when s = "10" else
        d;
end est;
```

- Note que la asignación de 'b' está condicionada por otra cláusula when-else. Igual con la asignación de 'c'. Sólo la asignación de 'd' no está condicionada. En realidad, no hay límite en el número de condiciones anidadas.

- **Asignación condicionada de señal (WHEN - ELSE):**
- **3er ejemplo:** Codificador 4 a 2 con prioridad

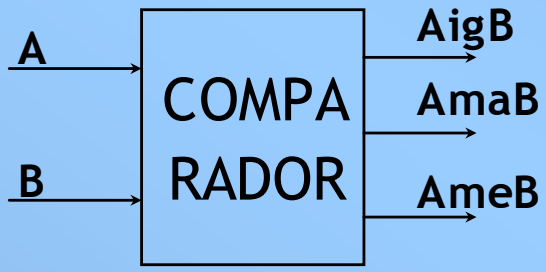


w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	0	0	0
1	x	x	x	1	1	1
0	1	x	x	1	0	1
0	0	1	x	0	1	1
0	0	0	1	0	0	1

- when-else tiene un nivel de prioridad, por eso es muy sencillo describir este circuito. Con with-select sería más tedioso pues hay que indicar todos los casos.

```
library ieee;
use ieee.std_logic_1164.all;

entity cod_pri_cond is
 port (w: in std_logic_vector (3 downto 0);
       y: out std_logic_vector (1 downto 0);
       z: out std_logic);
end cod_pri_cond;

architecture est of cod_pri_cond is
begin
   y <= "11" when w(3) = '1' else
        "10" when w(2) = '1' else
        "01" when w(1) = '1' else
        "00";
   z <= '0' when w = "0000" else '1';
end est;
```

- **Asignación condicionada de señal (WHEN - ELSE):**
- **4to ejemplo:** Comparador de 4 bits.




```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --#s sin signo

entity compara is
 port (A,B: in std_logic_vector (3 downto 0);
       AigB, AmaB, AmeB: out std_logic);
end compara;

architecture est of compara is
begin
   AigB <= '1' when A = B else '0';
   AmaB <= '1' when A > B else '0';
   AmeB <= '1' when A < B else '0';
end est;
```

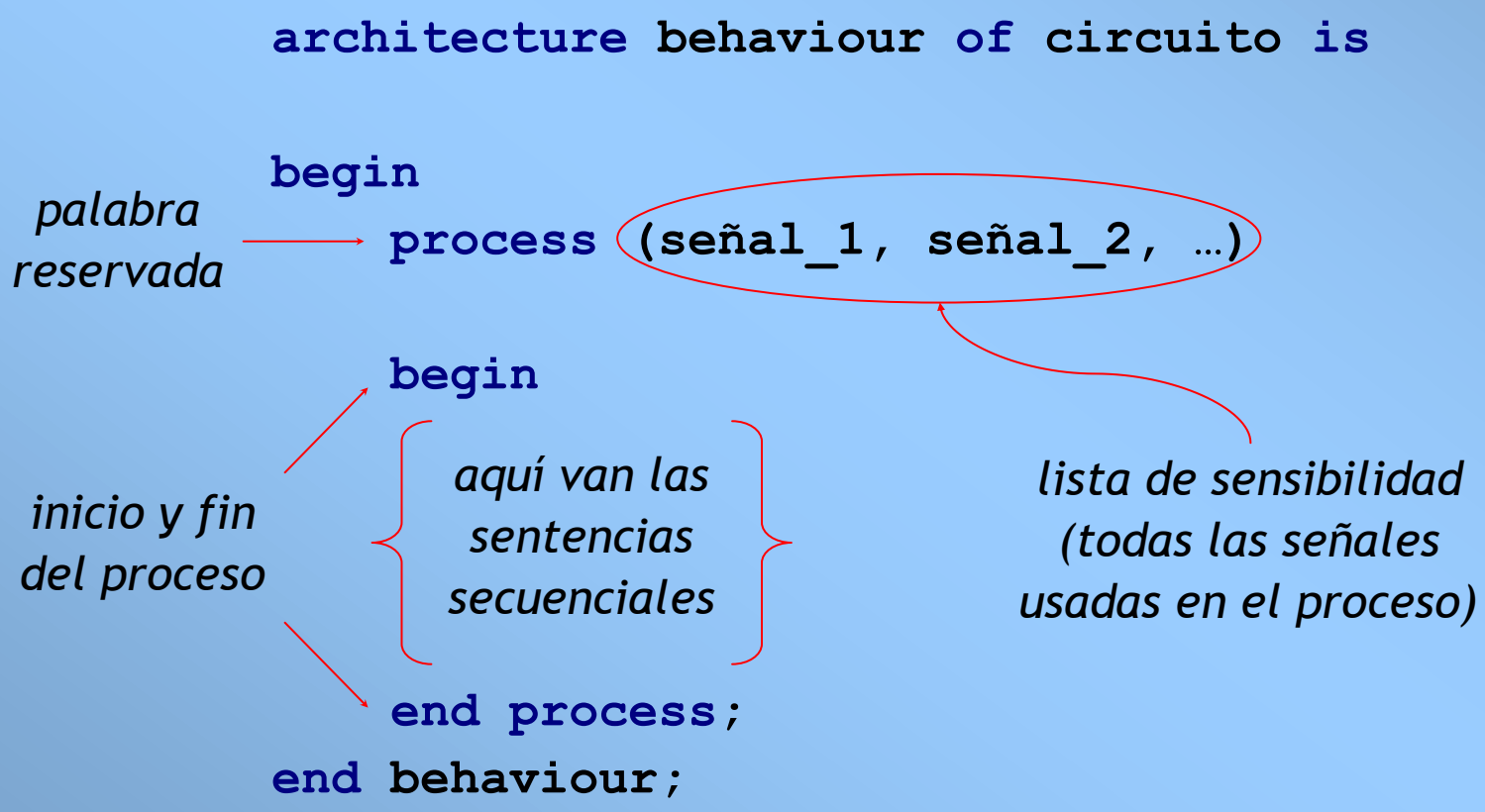
- Note el uso de los operadores '=, >, <' para comparar números.
- Ojo que se debe indicar con qué tipo de números se está trabajando (ver 3ª línea)

✓ *DESCRIPCIÓN COMPORTAMENTAL (ó SECUENCIAL)*

- En este tipo de descripción el comportamiento del circuito se especifica a través de una serie de sentencias que se ejecutan una tras otra; el orden de las sentencias es muy importante. Esto característica se aprovecha para **implementar circuitos secuenciales**. Las sentencias secuenciales deben estar dentro de un bloque del código VHDL llamado ‘proceso’ (PROCESS).
- El código secuencial no se limita a describir circuitos secuenciales, **también se pueden describir circuitos combinacionales**.
- En la presente sesión se usará esta descripción secuencial para implementar circuitos combinacionales, esto es se trabajará con los **procesos asíncronos**.

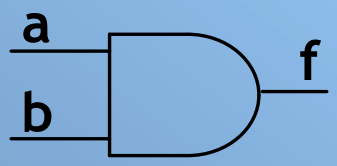
- **Procesos Asíncronos** (Implementación de circuitos combinacionales por medio de la descripción secuencial)

  Se muestra el esquema de una descripción secuencial. Note que la sentencia 'process' es la que denota al bloque secuencial:



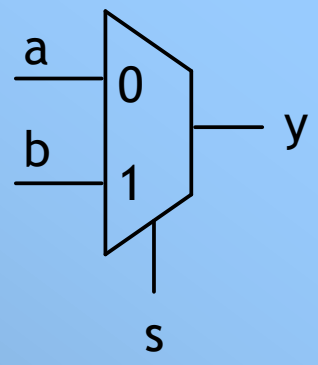
- **SENTENCIAS SECUENCIALES:**
  - **Sentencia IF:** Condicional Simple
  - **1er ejemplo:** Se muestra como ejemplo a una compuerta 'AND'. La lista de sensibilidad la componen 'a' y 'b' (las entradas de la compuerta). La sentencia IF acepta los operadores 'and', 'or', 'nor', 'nand', 'not', 'xor', 'xnor'.

```
library ieee;
use ieee.std_logic_1164.all;

entity and_sec is
 port (a, b : in std_logic;
          f : out std_logic);
end and_sec;
```




```
architecture bhv of and_sec is

begin
 process (a, b)
 begin
  if (a = '1') and (b = '1') then
     f <= '1';
  else
     f <= '0';
  end if;
 end process;
end bhv;
```

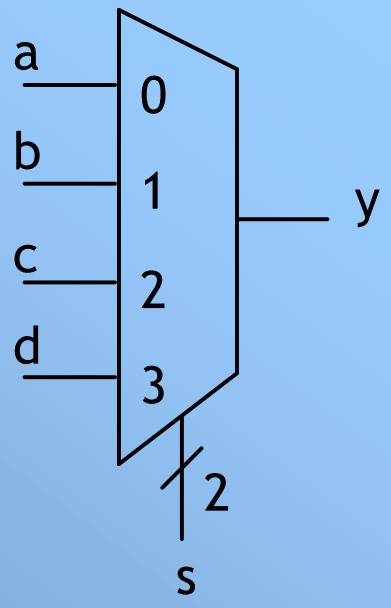
- **Sentencia IF:**
- **2do ejemplo:** Multiplexor de 2 entradas y una salida.




```
library ieee;
use ieee.std_logic_1164.all;

entity mux21_sec is
 port (a, b, s: in std_logic;
       y : out std_logic);
end mux21_sec;

architecture bhv of mux21_sec is
begin
  process (a, b, s)
  begin
    if s = '0' then
       y <= a;
    else
       y <= b;
    end if;
  end process;
end bhv;
```

- **Sentencia IF:**
- **3er ejemplo:** Multiplexor de 4 entradas y una salida.

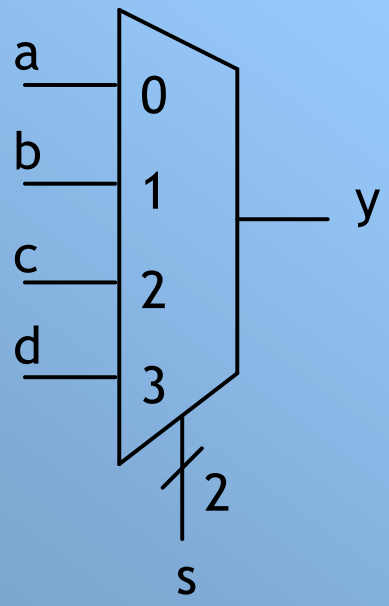
  Primero se muestra la descripción horizontal:




```
library ieee;
use ieee.std_logic_1164.all;

entity mux41_co is
 port ( a, b, c, d: in std_logic;
        s: in std_logic_vector (1 downto 0);
        y: out std_logic);
end mux41_co;

architecture est of mux41_co is
begin
 y <= (a and not s(1) and not s(0)) or
      (b and not s(1) and s(0)) or
      (c and s(1) and not s(0)) or
      (d and s(1) and s(0));
end est;
```

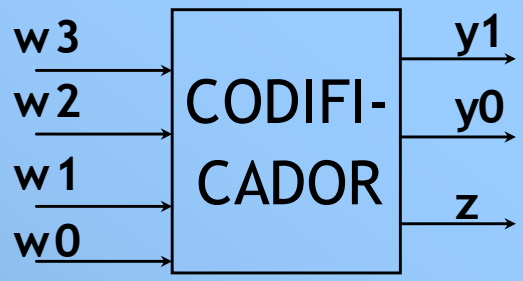
- **Sentencia IF:**
- **3er ejemplo:** Multiplexor de 4 entradas y una salida.

Descripción secuencial: Note que el código es mas fácil de leer. Para un mux de 8 entradas, la descripción horizontal ya es muy complicada.




```
library ieee;
use ieee.std_logic_1164.all;

entity mux41_sec is
 port (a, b, c, d : in std_logic;
       s: in std_logic_vector (1 downto 0);
       y : out std_logic);
end mux41_sec;

architecture bhv of mux41_sec is
begin
  process (a, b, c, d, s)
  begin
    if s = "00" then y <= a;
    elsif s = "01" then y <= b;
    elsif s = "10" then y <= c;
    else y <= d;
    end if;
  end process;
end bhv;
```

- **Sentencia IF:**
- **4to ejemplo:** Codificador 4 a 2 con prioridad.



w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	0	0	0
1	x	x	x	1	1	1
0	1	x	x	1	0	1
0	0	1	x	0	1	1
0	0	0	1	0	0	1

```
library ieee;
use ieee.std_logic_1164.all;

entity cod_pri_a is
 port (w: in std_logic_vector (3 downto 0);
       y: out std_logic_vector (1 downto 0);
       z : out std_logic);
end cod_pri_a;

architecture bhv of cod_pri_a is
begin
  process (w)
  begin
    if w(3) = '1' then y <= "11";
    elsif w(2) = '1' then y <= "10";
    elsif w(1) = '1' then y <= "01";
    else y <= "00";
    end if;
    if w = "0000" then
      z <= '0';
    else z <= '1';
    end if;
  end process;
end bhv;
```

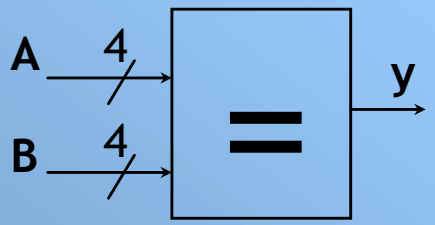
- **Sentencia IF:**
- **4to ejemplo:** Codificador 4 a 2 con prioridad (Otra forma).

- Recordar que las sentencias se ejecutan una después de otra:
- La 1ª sentencia asigna y <= “00” por defecto. La segunda cambia a ‘y’ si es que w(1) = ‘1’. La tercera y la cuarta sobreescriben al ‘y’ anterior si se cumplen las respectivas condiciones.
- ‘z’ empieza con ‘1’ , pero si la condición se cumple cambia a ‘0’.

```
library ieee;
use ieee.std_logic_1164.all;

entity cod_pri_b is
 port (w: in std_logic_vector (3 downto 0);
       y: out std_logic_vector (1 downto 0);
       z : out std_logic);
end cod_pri_b;

architecture bhv of cod_pri_b is
begin
  process (w)
  begin
    y <= "00";
    if w(1) = '1' then y <= "01"; end if;
    if w(2) = '1' then y <= "10"; end if;
    if w(3) = '1' then y <= "11"; end if;
    z <= '1';
    if w = "0000" then z <= '0'; end if;
  end process;
end bhv;
```

- **Sentencia IF:**
- **5to ejemplo:** Comparador de 4 bits

Descripción secuencial: Código más fácil de hacer. Los 4 bits se comparan de una sola vez. La descripción horizontal hubiera sido extensa.




```
library ieee;
use ieee.std_logic_1164.all;

entity comp_4 is
 port (a, b : in std_logic_vector (3 downto 0);
       y : out std_logic);
end comp_4;

architecture bhv of comp_4 is
begin
  process (a, b)
  begin
    if a = b then
       y <= '1';
    else
       y <= '0';
    end if;
  end process;
end bhv;
```

- **Sentencia IF:**
- **Ejemplo de un mal diseño**: Se usa el comparador de 4 bits, pero esta vez no se utiliza 'else' en el 'if'.

¡Atención!

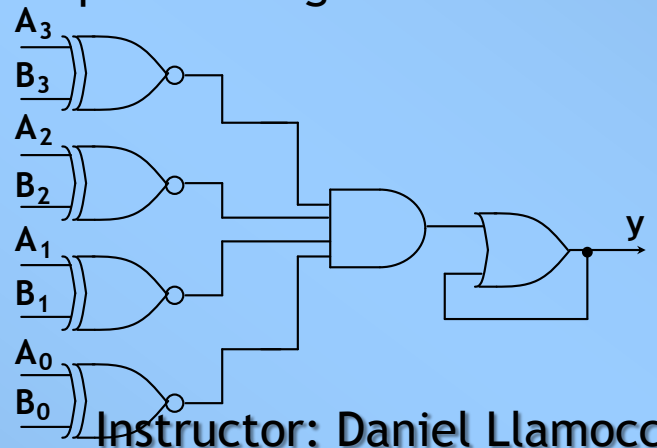
Si a ≠ b ➔ y = ?

Al no especificarse, la semántica del VHDL hace que se retenga el último valor de 'y'.

∴ Si a = b ➔ y = '1' para siempre. Se dice que la salida tiene una **memoria implícita** ya que 'recuerda' el valor de 'y'. Esto genera un comparador defectuoso.

```
library ieee;
use ieee.std_logic_1164.all;

entity comp_4 is
 port (
   a, b : in std_logic_vector (3 downto 0);
      y : out std_logic);
end comp_4;

architecture bhv of comp_4 is
begin
  process (a, b)
  begin
    if a = b then
       y <= '1';
    end if;
  end process;
end bhv;
```

El circuito sintetizado quedaría algo así:



- **REGLAS PARA UN BUEN DISEÑO COMBINACIONAL:**
- **Regla 1:** TODAS las señales de entrada en el PROCESS deben aparecer en la lista de sensibilidad.
- **Regla 2**: TODAS las combinaciones posibles de señales de entrada/salida deben incluirse en el código. Esto es, debe poderse obtener toda la tabla de verdad del circuito al observar el código.

```
architecture bhv of comp_4 is
begin
  process (a, b)
  begin
    if a = b then
       y <= '1';
    else
       y <= '0';
    end if;
  end process;
end bhv;
```

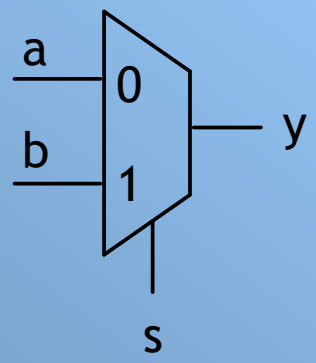
```
architecture bhv of comp_4 is
begin
  process (a, b)
  begin
    if a = b then
       y <= '1';
    end if;
  end process;
end bhv;
```

Tabla de verdad incompleta.
No se especifica cuándo y = '0'

- **SENTENCIAS SECUENCIALES:**
  - **Sentencia CASE:**
  - **1er ejemplo:** MUX 2 a 1

    El CASE se usa en casos de decisión múltiple para los que el IF se vuelve complejo.

    La sentencia CASE debe incluir un '**when**' para todos los posibles valores de la señal de selección. En el último caso se usa '**when others**' (esto se hace aún se halla descrito todas las combinaciones de '0' y '1' ya que el std_logic tiene 9 estados).



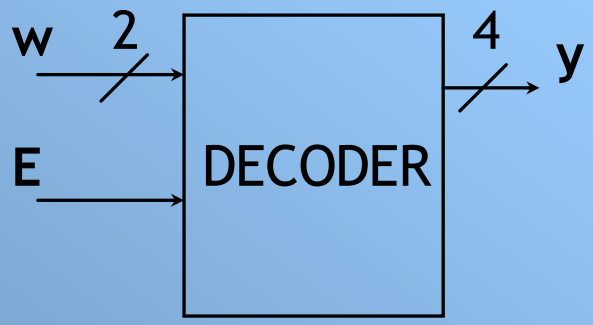

```
library ieee;
use ieee.std_logic_1164.all;

entity mux21_cas is
 port (a, b, s : in std_logic;
       y : out std_logic);
end mux21_cas;

architecture bhv of mux21_cas is
begin
  process (a, b, s)
  begin
    case s is
      when '0' =>
        y <= a;
      when others =>
        y <= b;
    end case;
  end process;
end bhv;
```

- **Sentencia CASE:**
- **2do ejemplo:** Decodificador binario a gray

B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

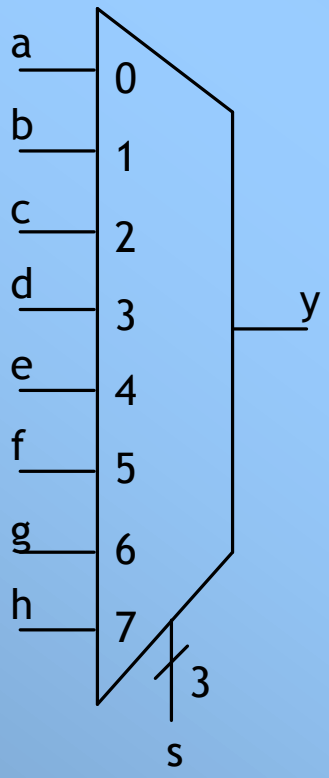
```
library ieee;
use ieee.std_logic_1164.all;

entity dec_bag is
 port( B: in std_logic_vector(2 downto 0);
       G: out std_logic_vector(2 downto 0));
end dec_bag;

architecture bhv of dec_bag is
begin
 process (B)
 begin
  case B is
    when  "000" => G <= "000";
    when  "001" => G <= "001";
    when  "010" => G <= "011";
    when  "011" => G <= "010";
    when  "100" => G <= "110";
    when  "101" => G <= "111";
    when  "110" => G <= "110";
    when others => G <= "100";
  end case;
 end process;
end bhv;
```

- **Sentencia CASE:**
- **3er ejemplo:** Decodificador de 2 a 4 con habilitador
- Observe cómo se combina IF con CASE para formar este decodificador con habilitador. Si no existiera 'else', la salida tendría una memoria implícita (LATCH).




```
library ieee;
use ieee.std_logic_1164.all;

entity dec2a4 is
 port(w: in std_logic_vector (1 downto 0);
      y: out std_logic_vector (3 downto 0);
      E: in std_logic);
end dec2a4;

architecture bhv of dec2a4 is
begin
  process (w, E)
  begin
    if E = '1' then
      case w is
        when "00" => y <= "0001";
        when "01" => y <= "0010";
        when "10" => y <= "0100";
        when others => y <= "1000";
      end case;
    else y <= "0000";
    end if;
  end process;
end bhv;
```

- **Sentencia CASE:**
- **4to ejemplo:**
  Multiplexor de 8 a 1




```
library ieee;
use ieee.std_logic_1164.all;

entity mux81_cas is
 port(a,b,c,d,e,f,g,h: in std_logic;
      s: in std_logic_vector (2 downto 0);
      y: out std_logic);
end mux81_cas;

architecture bhv of mux81_cas is
begin
  process (a,b,c,d,e,f,g,h,s)
  begin
      case s is
        when "000" => y <= a;
        when "001" => y <= b;
        when "010" => y <= c;
        when "011" => y <= d;
        when "100" => y <= e;
        when "101" => y <= f;
        when "110" => y <= g;
        when others => y <= h;
      end case;
  end process;
end bhv;
```

- **Sentencia CASE:**
- **5to ejemplo:** Multiplexor de 7 a 1
- Note el uso de la sentencia **y <= ‘ - ’ (condición de no importa)**
- Esto permite que el compilador sintetice mejor el circuito.
- Si se hubiera hecho when others => **y <= g;** el compilador tendría que asignar a ‘g’ para los casos “110” y “111”.

```
when "110" => y <= g;
when "111" => y <= g;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity mux71_cas is
 port(a,b,c,d,e,f,g,h: in std_logic;
      s: in std_logic_vector (2 downto 0);
      y: out std_logic);
end mux71_cas;

architecture bhv of mux71_cas is
begin
  process (a,b,c,d,e,f,g,s)
  begin
    case s is
      when "000" => y <= a;
      when "001" => y <= b;
      when "010" => y <= c;
      when "011" => y <= d;
      when "100" => y <= e;
      when "101" => y <= f;
      -- when others => y <= g;
      when "110" => y <= g;
      when others => y <= '-';
    end case;
  end process;
end bhv;
```

- **Sentencia CASE:**
- **6to ejemplo:** Decodificador de 7 segmentos
- También se usa el valor '-' (no importa) para que se reduzca el circuito, ya que la decodificación va sólo de "0000" a "1001".
- **Note que con CASE se evita la salida con memoria implícita, debido a que la sentencia when others siempre captura los casos restantes.**

```
library ieee;
use ieee.std_logic_1164.all;

entity dec_7seg is
 port(bcd: in std_logic_vector(3 downto 0);
      leds: out std_logic_vector(0 to 6));
end dec_7seg;

architecture bhv of dec_7seg is
begin
  process (bcd)
  begin
   case bcd is                    --abcdefg
     when "0000" => leds <= "1111110";
     when "0001" => leds <= "0110000";
     when "0010" => leds <= "1101101";
     when "0011" => leds <= "1111001";
     when "0100" => leds <= "0110011";
     when "0101" => leds <= "1011011";
     when "0110" => leds <= "1011111";
     when "0111" => leds <= "1110000";
     when "1000" => leds <= "1111111";
     when "1001" => leds <= "1111011";
     when others => leds <= "-------";
   end case;
  end process;
end bhv;
```