

Lenguaje VHDL

Código para representar sistemas
digitales en VHDL

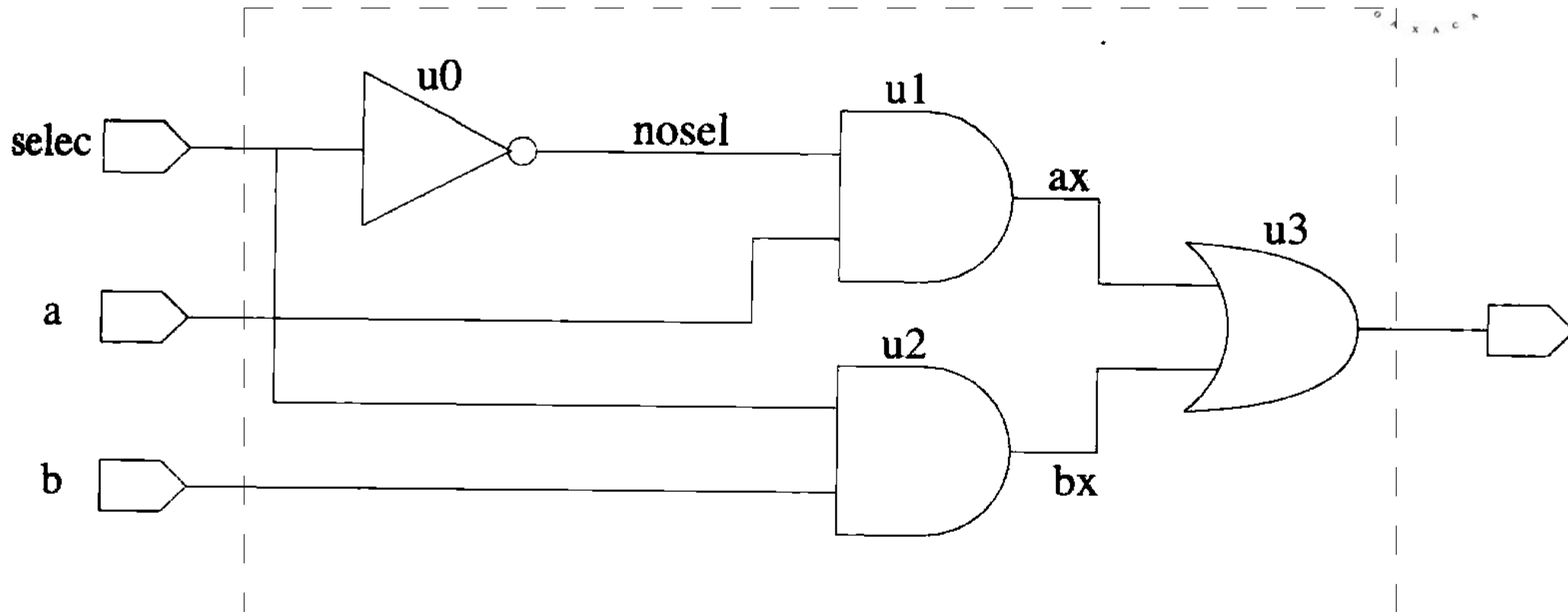
Las secciones fundamentales que forman el código en VHDL son: librería (LIBRARY), entidad (ENTITY) y arquitectura (ARCHITECTURE).

LIBRARY: contiene un conjunto de librerías que contienen la definición de datos, estructuras, etc.

ENTITY: Especifica las entradas/salidas del circuito.

ARCHITECTURE: Existen dos formas de describir un circuito, estructural y por su comportamiento. Esta forma a su vez se divide en flujo de datos y algorítmica.

ENTIDAD

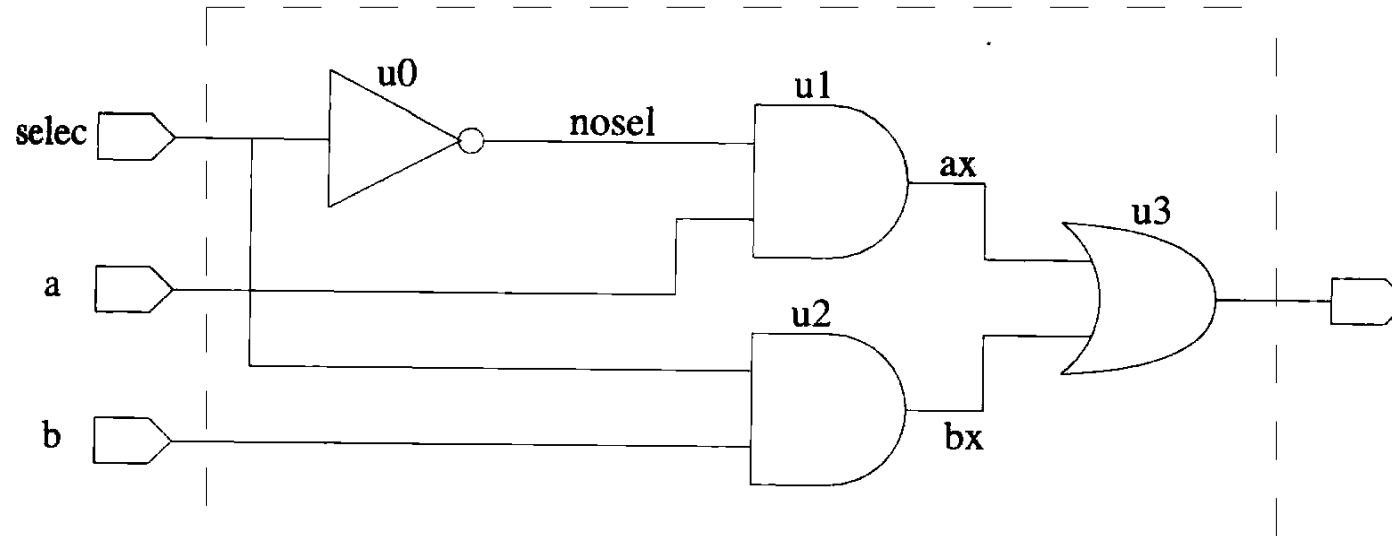


ENTITY MUX IS

PORT (A:	IN	BIT;
B:	IN	BIT;
SELEC:	IN	BIT;
SALIDA:	OUT	BIT);

END MUX

Arquitectura (estructural)



ARCHITECTURE ESTRUCTURAL OF MUX IS

SIGNAL AX,BX,NOSEL: BIT;

BEGIN

U0: ENTITY INV

U1: ENTITY AND2

U2: ENTITY AND2

U3: ENTITY OR2

END ESTRUCTURAL;

PORT MAP (E=>SELEC, Y=>NOSEL);

PORT MAP (E1=>A, E2=>NOSEL, Y=>AX);

PORT MAP (B, SELEC, BX);

PORT MAP (E1=>AX, E2=>BX,Y=>SALIDA);

Descripción estructural completa



ENTITY MUX IS

```
PORT ( A:      IN      BIT;  
       B:      IN      BIT;  
       SELEC:   IN      BIT;  
       SALIDA:  OUT     BIT);
```

END MUX;

ARCHITECTURE ESTRUCTURAL OF MUX IS

SIGNAL AX,BX,NOSEL: BIT;

BEGIN

```
U0: ENTITY    INV    PORT MAP (E=>SELEC, Y=>NOSEL);  
U1: ENTITY    AND2   PORT MAP (E1=>A, E2=>NOSEL, Y=>AX);  
U2: ENTITY    AND2   PORT MAP (B, SELEC, BX);  
U3: ENTITY    OR2    PORT MAP (E1=>AX, E2=>BX,Y=>SALIDA);
```

END ESTRUCTURAL;

Descripción completa comportamental-algorítmica



```
ENTITY MUX IS
  PORT (    A:      IN    BIT;
           B:      IN    BIT;
           SELEC:  IN    BIT;
           SALIDA: OUT   BIT);
END MUX;
```

architecture Comportamental of MUX is

```
BEGIN
  PROCESS (A,B,selec)
  begin
    if (selec='0') then
      salida<=a;
    else
      salida<=b;
    end if;
  end process;
end Comportamental;
```

Descripción completa comportamental-flujo de datos



```
ENTITY MUX IS
  PORT ( A:      IN      BIT;
         B:      IN      BIT;
         SELEC:  IN      BIT;
         SALIDA: OUT     BIT);
END MUX;
```

architecture Flujo1 of MUX is
signal nosel,ax,bx: bit;

```
BEGIN
  Nosel <= not selec;
  Ax<=    a  AND  nosel;
  Bx<=    b  AND  selec;
  Salida<= ax OR  bx;
end Flujo1;
```

Descripción completa comportamental-flujo de datos (transferencia entre registros)



```
ENTITY MUX IS
  PORT ( A:      IN      BIT;
         B:      IN      BIT;
         SELEC:  IN      BIT;
         SALIDA: OUT     BIT);
END MUX;
```

```
architecture Flujo2 of MUX is
BEGIN
  salida<=a when selec='0' ELSE b;
end Flujo2;
```


Entidad

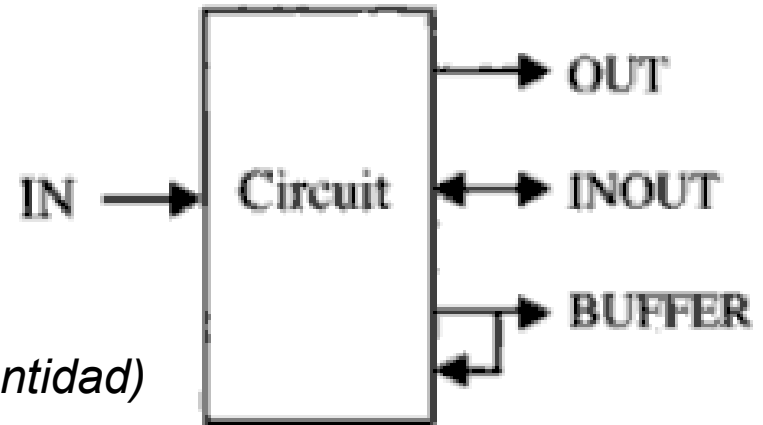


Una entidad está compuesta por tres elementos:

1. Puertos de entrada-salida. Cada una de las señales de entrada y salida en una entidad, son llamadas puertos. Los puertos deben poseer nombre, modo y tipo de datos.

2. Modos.

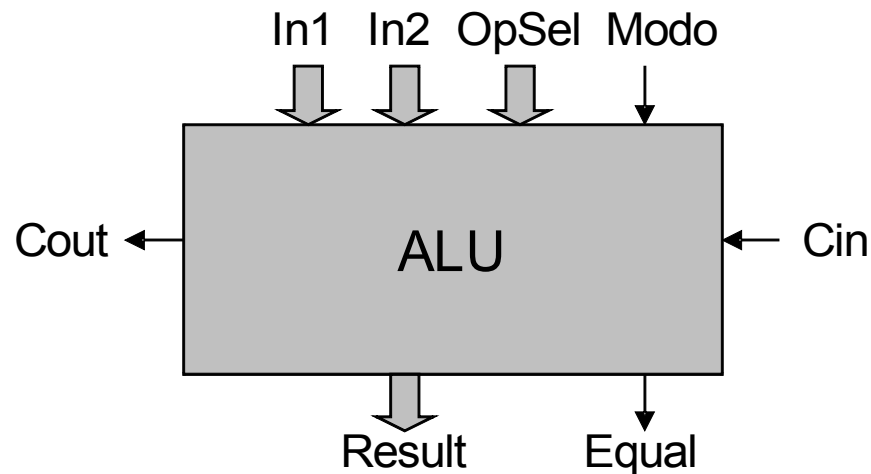
- (a) Modo in
- (b) Modo out
- (c) Modo inout (*puerto bidireccional con Retroalimentación dentro o fuera de la entidad*)
- (d) Modo buffer (se comporta como salida pero permite realizar retroalimentaciones internas)



Tipos de datos.

- (a) std_logic.
- (b) Boolean.
- (c) std_logic_vector (vectores de bits).
- (d) Integer.

Ejercicio: Declare la entidad de la ALU de cuatro bits mostrada.



Entity ALU is

```
Port (
  In1 :    in    std_logic_vector (3 downto 0);    -- Primer Operando
  In2 :    in    std_logic_vector (3 downto 0);    -- Segundo Operando
  OpSel :   in    std_logic_vector(0 to 2);        -- Selector de Operación
  Mode :   in    std_logic;                        -- Modo Aritmético/Lógico
  Cin :    in    std_logic;                        -- Acarreo de entrada
  Result : out    std_logic_vector (3 downto 0);    -- Resultado
  Cout :   out    std_logic;                       -- Acarreo de salida
  Equal :  out    std_logic;                       -- 1 cuando In1 = In2
);
```

End Entity ALU;



Arquitectura



Una arquitectura (*architecture*) se define como la estructura que describe el funcionamiento de una entidad.

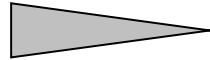
Las arquitecturas se clasifican en:

1. Estructural.
2. Funcional (comportamental).
3. Flujo de datos.

Comparación de los tipos de Arquitectura

¿QUÉ HACE?

Descripción
Funcional



¿CÓMO ESTÁ
COMPUERTA?

Descripción
Estructural



La descripción estructural es mucho más fácil de sintetizar que la descripción funcional por que se refiere a componentes físicos concretos.

Sin embargo es más difícil de desarrollar por que requiere de mayor experiencia del diseñador para hacer los diseños más eficientes.

1212

Descripción funcional (comportamental) comparador



La descripción funcional expone la forma en que trabaja el sistema; se consideran las entradas y salidas sin importar como esté organizado en su interior.



entity comp is

Port (

a : in STD_LOGIC_VECTOR (1 downto 0);

b : in STD_LOGIC_VECTOR (1 downto 0);

c : out STD_LOGIC);

end comp;

architecture funcional of comp is
begin

process(a,b)

begin

if a = b then

c <='1';

else

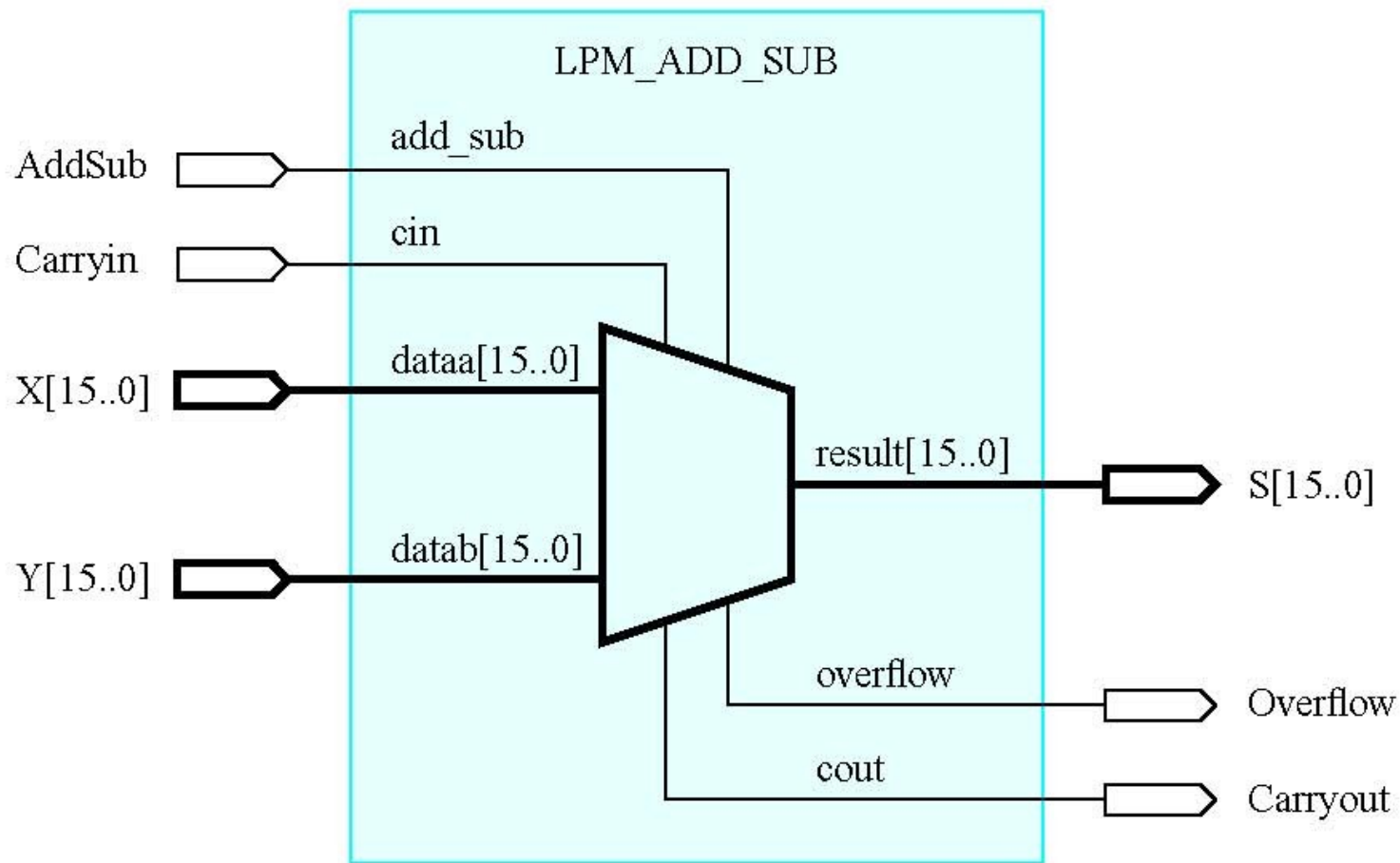
c <='0';

end if;

end process;

end funcional;

Ejercicio: Escriba la declaración de entidad para el siguiente circuito.



Tipos de declaraciones



En VHDL existen dos tipos de declaraciones: **secuenciales y concurrentes**.

Las secuenciales requieren (*if-then-else*) o (*case*), dentro de la instrucción *process*.

La declaración concurrente se obtiene con *when-e/se* o con ecuaciones booleanas.

Ejercicio: Escriba las declaraciones de las arquitecturas secuencial y concurrente para un comparador de dos bits.

Secuencial

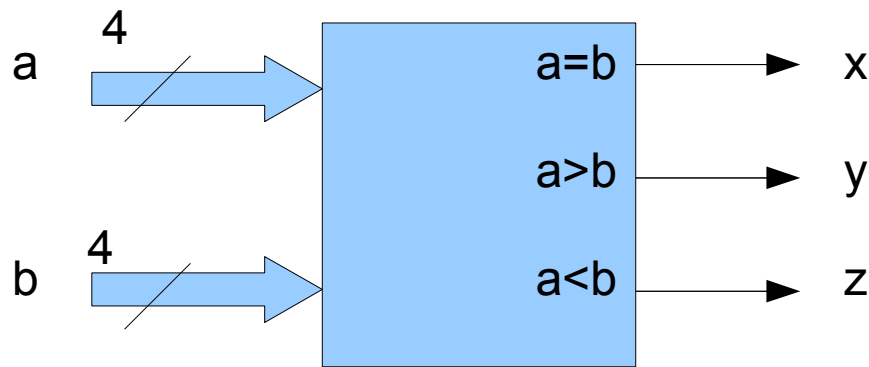
```
architecture funcional of comp is
begin
    process(a,b)
    begin
        if a = b then
            c <='1';
        else
            c <='0';
        end if;
    end process;
end funcional;
```

Concurrente

```
entity comp_concurrente is
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
          b : in  STD_LOGIC_VECTOR (1 downto 0);
          c : out STD_LOGIC);
end comp_concurrente;

architecture concurrente of comp is
begin
    c<='1' when (a=b) else '0';
end concurrente;
```


Comparador de magnitud de 4 bits (secuencial)



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

```
entity compa is
  Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
        b : in  STD_LOGIC_VECTOR (3 downto 0);
        x : out STD_LOGIC;
        y : out STD_LOGIC;
        z : out STD_LOGIC);
end compa;
```

architecture Behavioral of compa is
begin

```
  process(a,b) begin
```

```
    if a=b then
      x<='1';
    elsif a>b then
      y<='1';
    else
      z<='1';
    end if;
```

```
  end process;
```

```
end Behavioral;
```

Declaraciones concurrentes



En una declaración concurrente no importa el orden en que se escriban las señales, ya que el resultado para determinada función sería el mismo.

En VHDL existen tres tipos de declaraciones concurrentes:

- 1. Declaraciones condicionales asignadas a una señal (*when-else*).**
- 2. Declaraciones concurrentes asignadas a señales.**
- 3. Selección de una señal (*with-select-when*).**

Declaraciones condicionales asignadas a una señal (*when-else*).



La declaración **when-else** se utiliza para asignar valores a una señal, determinando así la ejecución de una condición propia del diseño.

Ejemplo:

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

entity tabla is

```
Port ( a,b,c : in STD_LOGIC;  
      f : out STD_LOGIC);
```

end tabla;

architecture Behavioral of tabla is

begin

```
f <= '1' when (a='0' and b='0' and c='0') else  
      '1' when (a='0' and b='1' and c='1') else  
      '1' when (a='1' and b='1' and c='0') else  
      '1' when (a='1' and b='1' and c='1') else  
      '0';
```

end Behavioral;

Operadores lógicos



Los operadores lógicos presentan el siguiente orden y prioridad:

1. Expresiones entre paréntesis
2. Complementos
3. Funcion AND
4. Función OR

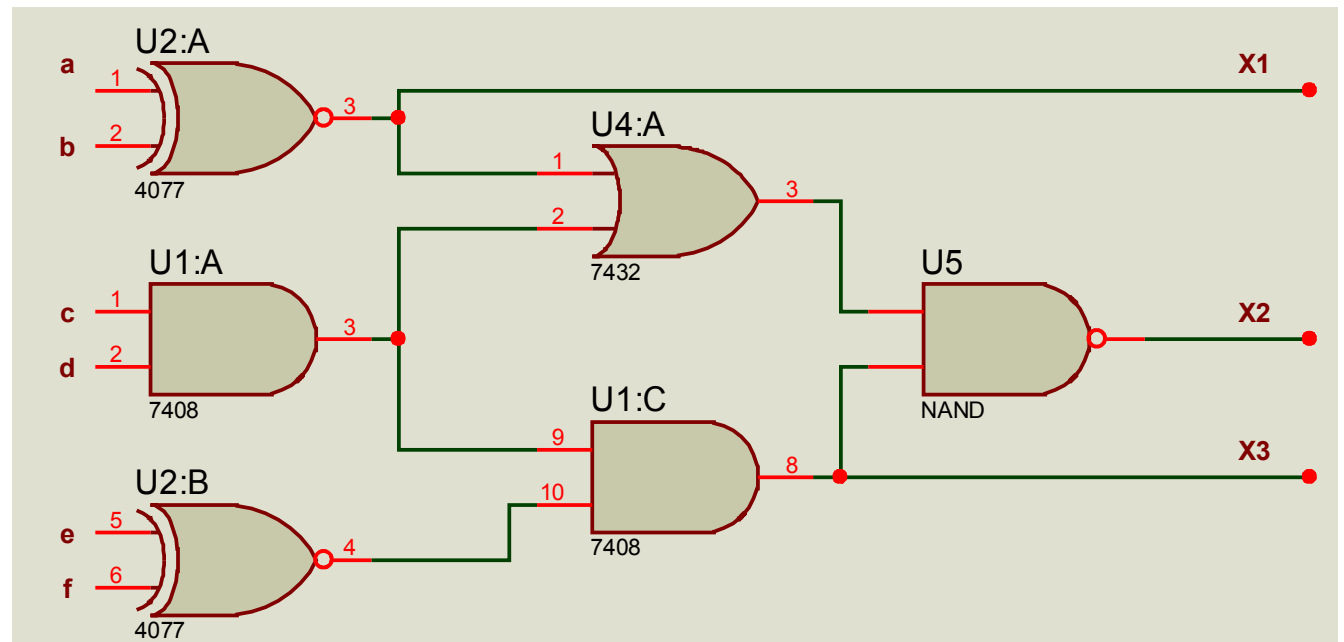
Ecuación	En VHDL
$Q = a + x \cdot y$	$q = a \text{ or } (x \text{ and } y)$
$Y = \overline{a + b \cdot c + d}$	$y = \text{not}(a \text{ or } (b \text{ and } c) \text{ or } d)$

Declaraciones concurrentes asignadas a señales



Las funciones de salida se describen mediante la ecuación booleana que describe el comportamiento de cada una de las compuertas.

Ejemplo:



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED;
```

entity logic is

```
port(
  a,b,c,d,e,f:   in STD_LOGIC;
  x1,x2,x3:      out STD_LOGIC);
end logic;
```

architecture booleana of logic is
begin

```
x1<= a xnor b;
x2<= (((c and d) or (a xnor b))nand
      ((c and d) and(e xnor f)));
x3<=(c and d) and (e xnor f);
end booleana;
```

Selección de una señal (with-select-when)



Esta declaración se utiliza para asignar un valor a una señal con base en el valor de otra señal previamente seleccionada.

a(0)	a(1)	C
0	0	1
0	1	0
1	0	1
1	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity arq is
    port(
        a : in STD_LOGIC_VECTOR(1 downto 0);
        C : out STD_LOGIC);
end arq;
```

```
architecture arq of arq is
begin
    with a select
        c <= '1' when "00",
        '0' when "01",
        '1' when "10",
        '0' when others;
end arq;
```

Ejercicio

Diseñe un circuito combinacional que detecte números primos de 4 bits. Realice la tabla de verdad y elabore un programa que describa su función. Utilice instrucciones *del tipo with_ select-when*.

architecture detecta of primos is
begin

```
entity primos is
  port(
    X : in STD_LOGIC_VECTOR(3 downto 0 );
    F : out STD_LOGIC
  );
end primos;
```

```
with X select
  F<= '1' when "0001",
      '1' when "0010",
      '1' when "0011",
      '1' when "0101",
      '1' when "0111",
      '1' when "1011",
      '1' when "1110",
      '0' when others;
```


Programación mediante declaraciones secuenciales

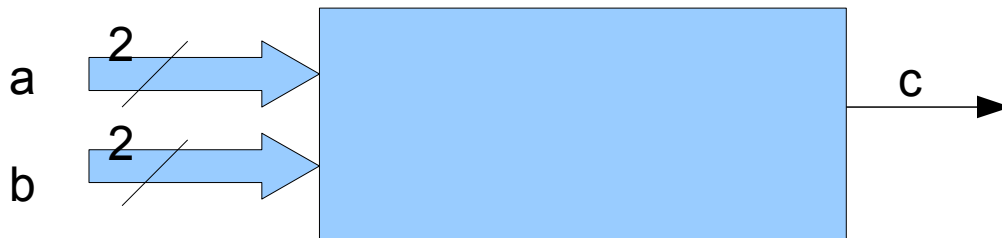


A diferencia de una declaración concurrente, una secuencial debe ejecutarse en el orden en que aparece y formar parte de un proceso (*process*).

- Declaración **if-then-else**

```
if la condición es cierta then
    realiza la operación 1;
else
    realiza la operación 2;
end if;
```

Ejemplo de un comparador



entity comp is

Port (

a : in STD_LOGIC_VECTOR (1 downto 0);

b : in STD_LOGIC_VECTOR (1 downto 0);

c : out STD_LOGIC);

end comp;

architecture funcional of comp is
begin

process(a,b)

begin

if a = b then

c <='1';

else

c <='0';

end if;

end process;

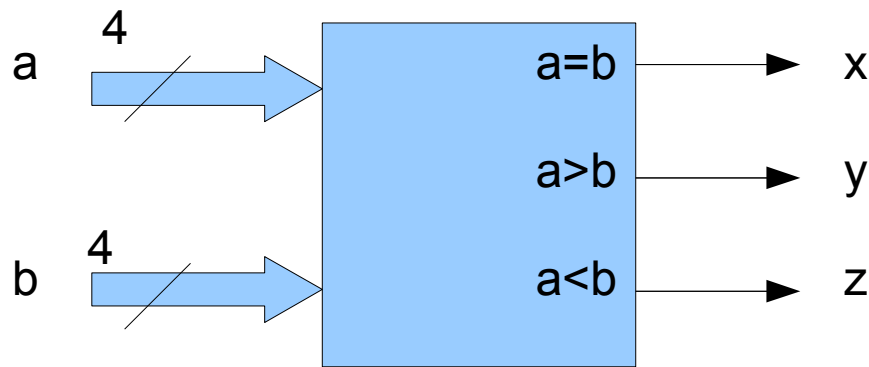
end funcional;

Para más condiciones (>2)



```
if la condición 1 se cumple then  
    realiza operación 1;  
elsif la condición 2 se cumple then  
    realiza operación 2;  
else  
    realiza operación 3;  
end if;
```

Comparador de magnitud de 4 bits (secuencial)



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

```
entity compa is
  Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
        b : in  STD_LOGIC_VECTOR (3 downto 0);
        x : out STD_LOGIC;
        y : out STD_LOGIC;
        z : out STD_LOGIC);
end compa;
```

architecture Behavioral of compa is
begin

```
  process(a,b) begin
```

```
    if a=b then
      x<='1';
    elsif a>b then
      y<='1';
    else
      z<='1';
    end if;
```

```
  end process;
```

```
end Behavioral;
```

Ejercicio



- Diseñe un comparador de dos números A y B, cada uno formado por dos bits, la salida del comparador también es de dos bits y está representada por la variable Z de tal forma que sí:

$A=B$ entonces $Z=11$

$A<B$ entonces $Z=01$

$A>B$ entonces $Z=10$

Operaciones relacionales



- Se usan para evaluar la igualdad, desigualdad o la magnitud en una expresión. Los operadores de igualdad y desigualdad se definen en todos los tipos de datos. Los operadores de magnitud (<, <=, > y >=) son aplicados en tipos escalares.

Operador	Significado
=	Igual
/=	Diferente
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual

Tipos de datos : Escalares



Se definen mediante la palabra reservada **RANGE**.

- Enteros.
 - **TYPE** byte **IS RANGE** 0 TO 255;
 - **TYPE** index **IS RANGE** 7 **DOWNT** 1;
 - **TYPE** integer **IS RANGE** -2147483648 **TO** 2147483648;--
- Reales
 - **TYPE** nivel **IS RANGE** 0.0 **TO** 5;
 - **TYPE** real **IS RANGE** -1.0E38 **TO** 1.0E38; --predefinido
- Físicos
- Enumerados

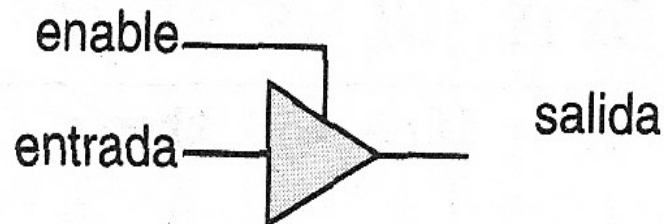
Tipos lógicos estándar



Tipo	Significado
'U'	Valor no inicializado
'X'	Valor fuerte desconocido
'0'	0 fuerte
'1'	1 fuerte
'Z'	Alta impedancia
'W'	Valor débil desconocido
'L'	0 débil
'H'	1 débil
'.'	No importa (don't care)

Buffer tri-estado

- Estos tienen diversas aplicaciones ya sea como salidas de sistemas (modo buffer) o como parte integral de un circuito.



architecture secuencial of \buffer\ is
begin

```
process (entrada, enable) begin
    if (enable='1')then
        salida<=entrada;
    else
        salida<='Z';
    end if;
end process;
```

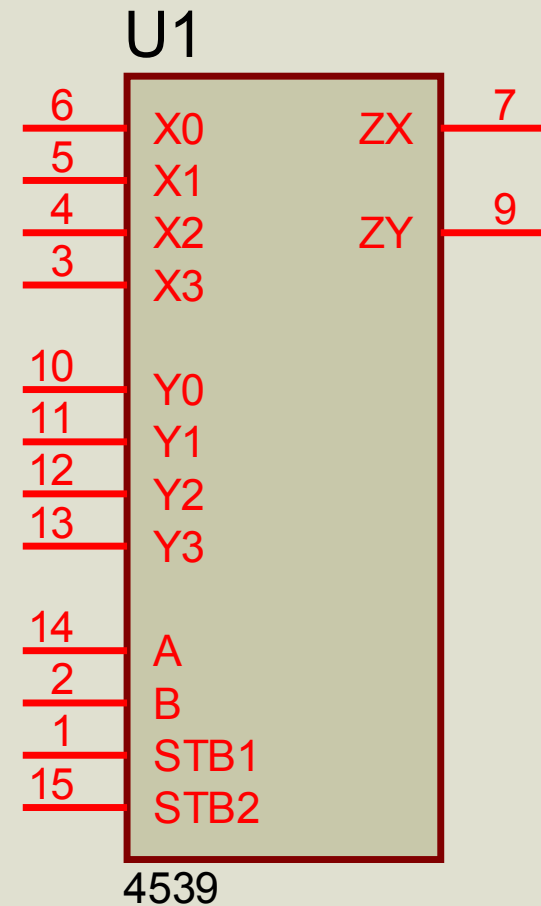
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity \buffer\ is
    port(
        entrada,enable : in STD_LOGIC;
        salida :          out STD_LOGIC
    );
end \buffer\;
```

end secuencial;

Multiplexores

- Emplee *with-select-when* para diseñar el circuito (no considere las señales STB1 y STB2).
- Diseñe el circuito empleando ecuaciones booleanas.



Multiplexor



```
entity mux2 is
  port(
    a,b,c,d,sel : in STD_LOGIC_VECTOR(1 downto 0);
    z :          out STD_LOGIC_VECTOR(1 downto 0)
  );
end mux2;
```

```
architecture mux2 of mux2 is
begin
```

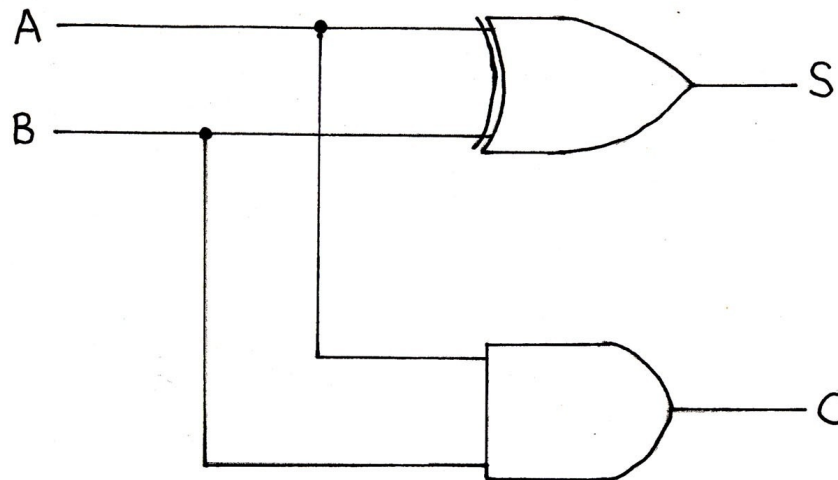
```
  with sel select
    z<= a when "00",
        b when "01",
        c when "10",
        d when others;
end mux2;
```

Medio sumador

A and B are the inputs
S represents the output *sum* and
C represents the output *carry*.

The relevant truth table for this circuit is:

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Implemente el circuito en VHDL.

Medio sumador

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity medsum is
    port(
        a,b : in STD_LOGIC;
        s,c : out STD_LOGIC);
end medsum;

--}} End of automatically maintained section

architecture medsum of medsum is
begin

    s<=a xor b;
    c<=a and b;

end medsum;
```

Tarea para entregar el 12/Nov



- ◆ Multiplexor de 4 canales a 1
- ◆ Decodificador de BCD a 7 segmentos
- ◆ Registro con carga paralela, con habilitación externa y reset asíncrono.
- ◆ Sumador completo de 1 bit

Operadores y tipos de datos



Operator	Description	Data type of operands	Data type of result
<code>a ** b</code>	exponentiation	integer	integer
<code>a * b</code>	multiplication	<i>integer type for constants and array boundaries, not synthesis</i>	
<code>a / b</code>	division		
<code>a + b</code>	addition		
<code>a - b</code>	subtraction		
<code>a & b</code>	concatenation	1-D array, element	1-D array
<code>a = b</code>	equal to	any	boolean
<code>a /= b</code>	not equal to	scalar or 1-D array	boolean
<code>a < b</code>	less than		
<code>a <= b</code>	less than or equal to		
<code>a > b</code>	greater than		
<code>a >= b</code>	greater than or equal to		
<code>not a</code>	negation	boolean, std_logic, std_logic_vector	same as operand
<code>a and b</code>	and		
<code>a or b</code>	or		
<code>a xor b</code>	xor		

Sobrecarga de operadores (síntesis)



Table 3.2 Overloaded operators and data types in the IEEE numeric_std package

Overloaded operator	Description	Data type of operands	Data type of result
a * b	arithmetic operation	unsigned, natural	unsigned
a + b		signed, integer	signed
a - b			
a = b	relational operation		
a /= b			
a < b		unsigned, natural	boolean
a <= b		signed, integer	boolean
a > b			
a >= b			

Sobre carga de operadores



```
u4 <= u2 + u1;  -- ok, both operands unsigned
u5 <= u2 + 1;    -- ok, operands unsigned and natural
```

```
s5 <= s2 + s1;  -- not ok, + undefined over the types
s6 <= s2 + 1;   -- not ok, + undefined over the types
```

```
s5 <= std_logic_vector(unsigned(s2) + unsigned(s1)); -- ok
s6 <= std_logic_vector(unsigned(s2) + 1);             -- ok
```

Conversiones entre tipo de datos



Table 3.3 Type conversions between `std_logic_vector` and numeric data types

Data type of a	To data type	Conversion function/type casting
unsigned, signed	<code>std_logic_vector</code>	<code>std_logic_vector(a)</code>
signed, <code>std_logic_vector</code>	unsigned	<code>unsigned(a)</code>
unsigned, <code>std_logic_vector</code>	signed	<code>signed(a)</code>
unsigned, signed	integer	<code>to_integer(a)</code>
natural	unsigned	<code>to_unsigned(a, size)</code>
integer	signed	<code>to_signed(a, size)</code>

Conversiones



```
signal s1,s2,s3,s4,s5,s6 :std_logic_vector(3 downto 0);
```

```
signal u1,u2,u3,u4,u5,u6,u7 :unsigned(3 downto 0);
```

```
begin
```

```
u1<=s1;
```

```
u2<=5
```

```
s2<=u3;
```

```
s3<=5;
```

{ !! error tipos de datos diferentes }

```
u1<=unsigned(s1);
```

```
u2<=to_unsigned(5,4);
```

```
s2<=std_logic_vector(u3);
```

```
s3<=std_logic_vector(to_unsigned(5,4));
```

{ correcto !!! }

Concatenación &



```
signal a1: std_logic;  
signal a4: std_logic_vector(3 downto 0);  
signal b8, c8, d8: std_logic_vector(7 downto 0);
```

```
. . .  
b8 <= a4 & a4;  
c8 <= a1 & a1 & a4 & "00";  
d8 <= b8(3 downto 0) & c8(3 downto 0);
```

Corrimiento con el operador &

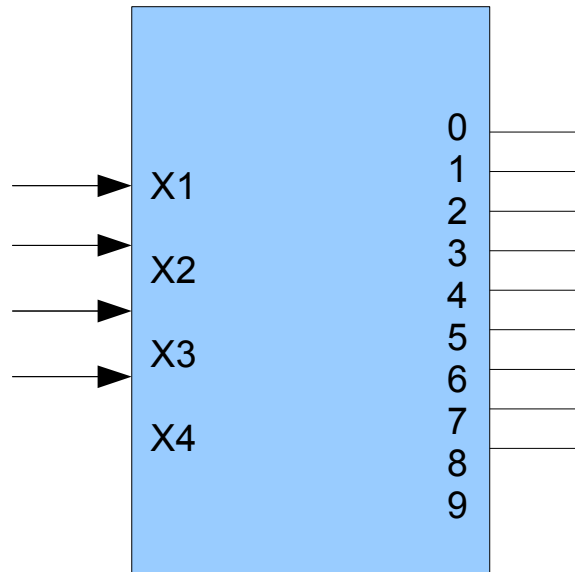


Algunas veces no es posible sintetizar un corrimiento y se emplea el operador concatenación (&).

```
signal a: std_logic_vector(7 downto 0);
signal rot, shl, sha: std_logic_vector(7 downto 0);
. . .
-- rotate a to right 3 bits
rot <= a(2 downto 0) & a(8 downto 3);
-- shift a to right 3 bits and insert 0 (logic shift)
shl <= "000" & a(8 downto 3);
-- shift a to right 3 bits and insert MSB
-- (arithmetic shift)
sha <= a(8) & a(8) & a(8) & a(8 downto 3);
```

Estructuras básicas mediante declaraciones secuenciales

Decodificador BCD/DEC



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity dec_bcd_dec is
```

```
port(
```

```
    x      : in  STD_LOGIC_VECTOR(3 downto 0);
```

```
    s0,s1,s2,s3,s4,s5,s6,s7,s8,s9      : out STD_LOGIC);
```

```
end dec_bcd_dec;
```

architecture secuencial of dec_bcd_dec is
begin

```
    process (x) begin
```

```
        if x="0000" then
```

```
            s0<='1';
```

```
        elsif x="0001" then
```

```
            s1<='1';
```

```
        elsif x="0010" then
```

```
            s2<='1';
```

```
        elsif x="0011" then
```

```
            s3<='1';
```

```
        elsif x="0100" then
```

```
            s4<='1';
```

```
        elsif x="0101" then
```

```
            s5<='1';
```

```
        elsif x="0110" then
```

```
            s6<='1';
```

```
        elsif x="0111" then
```

```
            s7<='1';
```

```
        elsif x="1000" then
```

```
            s8<='1';
```

```
        else
```

```
            s9<='1';
```

```
        end if;
```

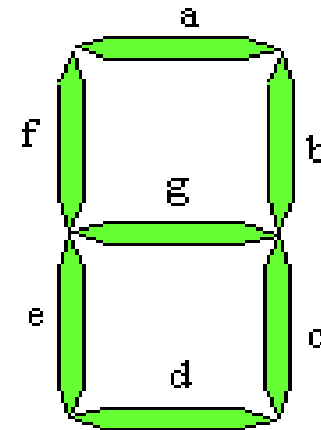
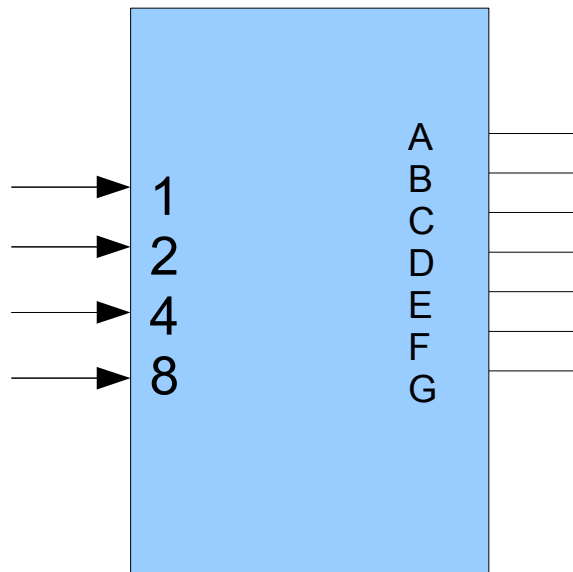
```
    end process;
end secuencial;
```

Ejercicio

(3 quiz = 1 punto de examen)



Diseñe un decodificador de BCD a 7 segmentos.

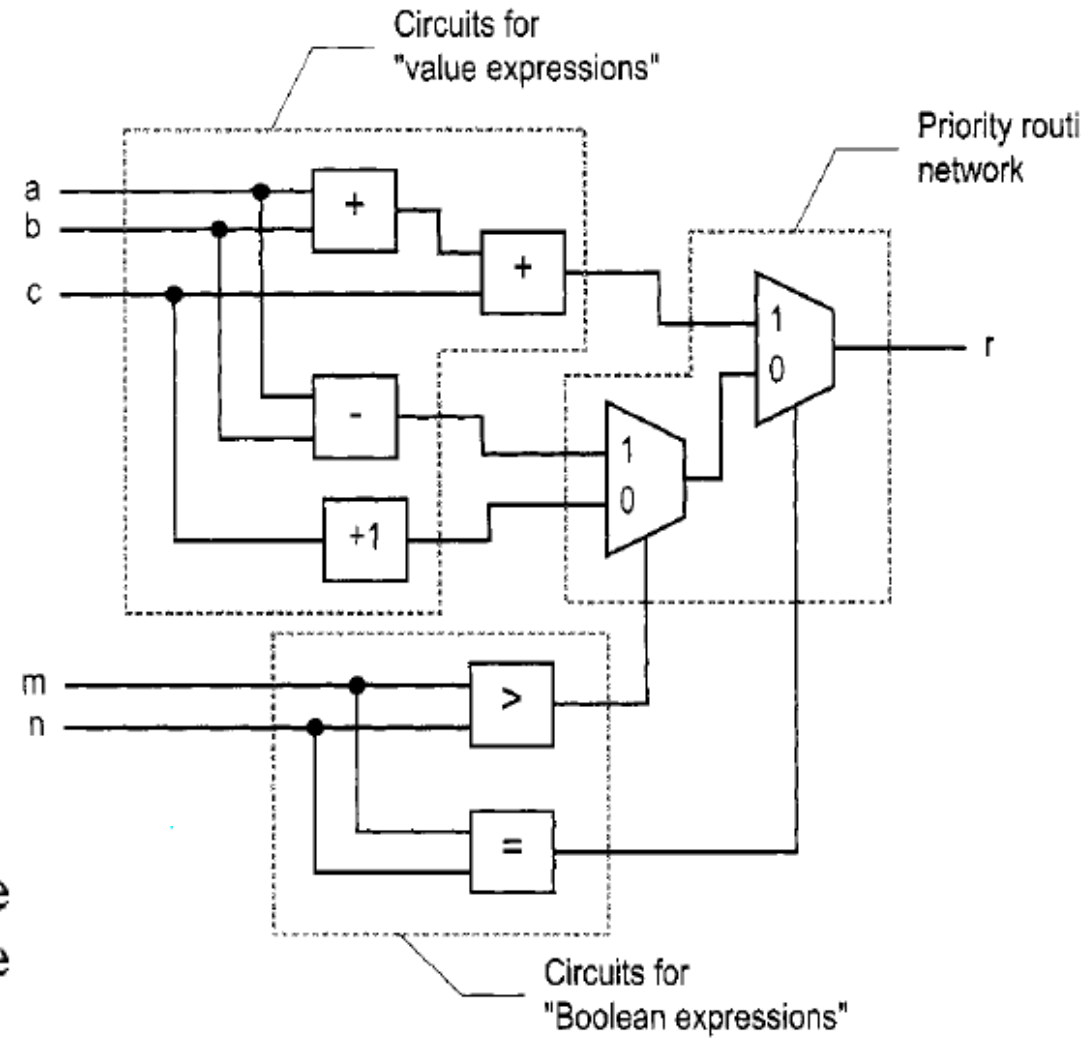


Empleando CASE

architecture secuencial of bcd7seg is
begin

```
process (A) begin
  case A is
    when "0000" => d<="0000001";
    when "0001" => d<="1001111";
    when "0010" => d<="0010010";
    when "0011" => d<="0000110";
    when "0100" => d<="1001100";
    when "0101" => d<="0100100";
    when "0110" => d<="0100000";
    when "0111" => d<="0001110";
    when "1000" => d<="0000000";
    when "1001" => d<="0000100";
    when others => d<="1111111";
  end case;
end process;
end secuencial;
```


Declaraciones condicionales asignadas a una señal



```

r <= a + b + c when m = n else
a - b      when m > n else
c + 1;
    
```

Ejemplo: Codificador de prioridad



input r	output pcode
1 — — —	100
0 1 — —	011
0 0 1 —	010
0 0 0 1	001
0 0 0 0	000

Quiz: Diseñe el codificador empleando *when-else* (declaración condicional a señales).

Implementación (1)



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity prio_encoder is
    port(
        r      : in STD_LOGIC_VECTOR(4 downto 1);
        pcode   : out STD_LOGIC_VECTOR(2 downto 0)
    );
end prio_encoder;

architecture arq_when of prio_encoder is
begin

    pcode<="100" when r(4)='1' else
           "011" when r(3)='1' else
           "010" when r(2)='1' else
           "001" when r(1)='1' else
           "000";

end arq_when;
```

Implementación (2)



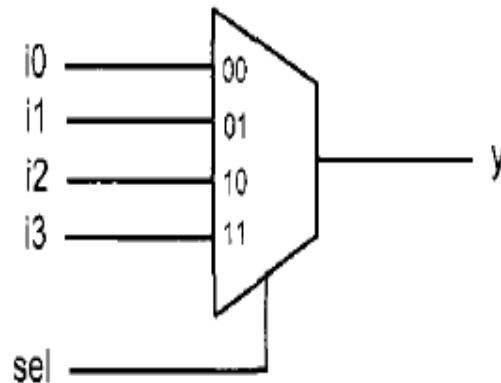
architecture arq_when of prio_encoder is

begin

```
with r select
pcode<= "100" when "1---",
        "011" when "01--",
        "001" when "0001",
        "000" when others;
end arq_when;
```

Diseñe el siguiente circuito empleando *with-select-when*

en	input		output
	a(1)	a(0)	y
0	—	—	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000



sel	y
00	i0
01	i1
10	i2
11	i3

Implementación

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decoder is
    port(
        a  : in STD_LOGIC_VECTOR(1 downto 0);
        en : in STD_LOGIC;
        y  : out STD_LOGIC_VECTOR(3 downto 0)
    );
end decoder;

architecture arq_when of decoder is
    signal s:std_logic_vector(2 downto 0 );
begin

    s<=en&a;
    with s select
        y<= "0000" when "0--",
            "0001" when "100",
            "0001" when "101",
            "0010" when "110",
            "1000" when others;
end arq_when;
```

Comparación concurrente vs secuencial



Existen declaraciones equivalentes en ambas arquitecturas, sin embargo, un *if* o *case* permite cualquier número y tipo de declaraciones secuenciales es sus ramas y así es más flexible y versátil. Un uso disciplinado puede hacer el código más descriptivo y aún hacer un circuito más eficiente.

Ejemplos

```
large  <= a when a > b else  
        b;
```

```
small  <= b when a > b else  
        a;
```

```
process (a, b)  
begin  
    if a > b then  
        large <= a;  
        small <= b;  
    else  
        large <= b;  
        small <= a;  
    end if;  
end;
```


Código intuitivo



```
max <= a when ((a > b) and (a > c)) else  
c when (a > b) else  
b when (b > c) else  
c;
```

```
process (a, b, c)  
begin  
  if (a > b) then  
    if (a > c) then  
      max <= a;  
    else  
      max <= c;  
    end if;  
  else  
    if (b > c) then  
      max <= b;  
    else  
      max <= c;  
    end if;  
  end if;  
end process;
```

Código intuitivo



```
signal ac_max, bc_max: std_logic;  
.  
.  
.  
  
ac_max <= a when (a > c) else  
        c;  
bc_max <= b when (b > c) else  
        c;  
max <= ac_max when (a > b) else  
        bc_max;
```

Memoria no asignada



Para prevenir este error, deberán seguirse las siguientes reglas:

1. Incluye todas las señales de entrada en la lista sensible.
2. Incluir el *e/se* en el *if*.
3. Asignar un valor a cada señal en cada rama.

Memoria no asignada



```
process(a)           -- b missing from sensitivity list
begin
  if (a > b) then      -- eq not assigned in this branch
    gt <= '1';
  elsif (a = b) then   -- gt not assigned in this branch
    eq <= '1';
  end if;              -- else branch is omitted
end process;
```

```
process(a,b)
begin
  if (a > b) then
    gt <= '1';
    eq <= '0';
  elsif (a = b) then
    gt <= '0';
    eq <= '1';
  else
    gt <= '0';
    eq <= '0';
  end if;
end process;
```

```
process(a,b)
begin
  gt <= '0';           -- assign default value
  eq <= '0';
  if (a > b) then
    gt <= '1';
  elsif (a = b) then
    eq <= '1';
  end if;
end process;
```

Constantes



Uso de constantes: sumador de 4 bits



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity sumador_c_carry is
    port(
        a : in STD_LOGIC_VECTOR(3 downto 0);
        b : in STD_LOGIC_VECTOR(3 downto 0);
        sum : out STD_LOGIC_VECTOR(3 downto 0);
        cout : out STD_LOGIC);
end sumador_c_carry;

architecture concurrente of sumador_c_carry is
    signal a_ext,b_ext,sum_ext : unsigned(4 downto 0);
begin
    a_ext<=unsigned('0'&a);
    b_ext<=unsigned('0'&b);
    sum_ext<=a_ext+b_ext;
    sum<=std_logic_vector(sum_ext(3 downto 0));
    cout<=sum_ext(4);
end concurrente;
```



Usando constantes para el código



Una mejor opción es definir el tamaño del sumador como una constante, de esta manera modificando este valor el código sirve para cualquier tamaño de sumador.

```
entity sumador_cc_var is
```

```
    port(
```

```
        a :    in STD_LOGIC_VECTOR(7 downto 0);
        b :    in STD_LOGIC_VECTOR(7 downto 0);
        sum :   out STD_LOGIC_VECTOR(7 downto 0);
        cout :  out STD_LOGIC;
```

```
end sumador_cc_var;
```

```
architecture concurrente of sumador_cc_var is
```

```
    constant N:integer:=8;
```

```
    signal a_ext,b_ext,sum_ext : unsigned(N downto 0);
```

```
    begin
```

```
        a_ext<=unsigned('0'&a);
```

```
        b_ext<=unsigned('0'&b);
```

```
        sum_ext<=a_ext + b_ext;
```

```
        sum<=std_logic_vector(sum_ext(N-1 downto 0));
```

```
        cout<=sum_ext(N);
```

```
    end concurrente;
```

Las constantes permiten tener un código más fácil de entender y mantener.

Generic

Se emplea para pasar información dentro de una entidad y componente. Ya que el *generic* no puede ser modificado dentro de la arquitectura, este funciona como una constante.

```
entity entity_name is
  generic(
    generic_name: data_type := default_values;
    generic_name: data_type := default_values;
    . . .
    generic_name: data_type := default_values
  )
  port(
    port_name: mode data_type;
    . . .
  );
end entity_name;
```


Sumador con *generic*

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

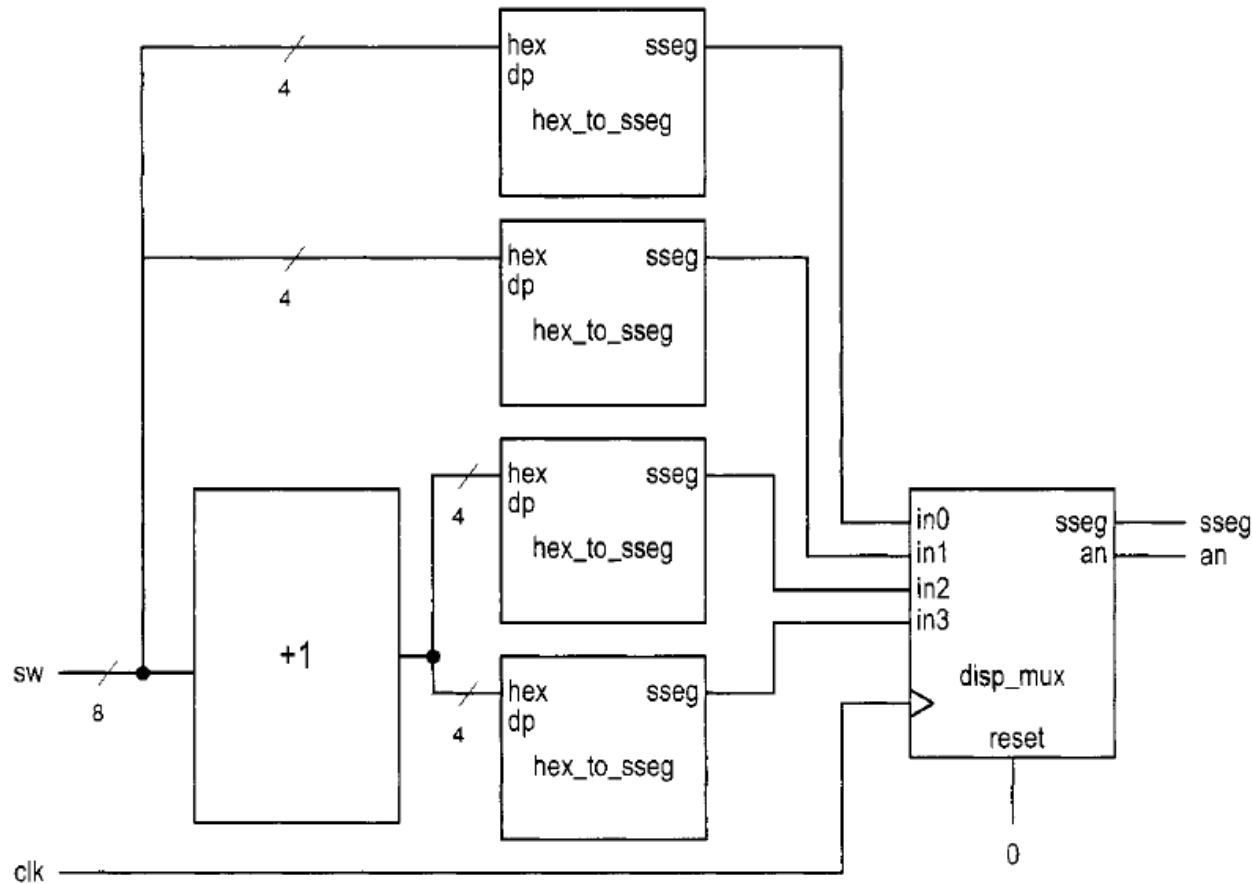
entity sum_var2 is
    generic (N:integer:=4);
    port(
        b : in  STD_LOGIC_VECTOR(N-1 downto 0);
        a : in  STD_LOGIC_VECTOR(N-1 downto 0);
        suma : out STD_LOGIC_VECTOR(N-1 downto 0);
        cout : out STD_LOGIC);
end sum_var2;

architecture gen of sum_var2 is
    signal a_ext,b_ext,sum_ext : unsigned(N downto 0);
begin
    a_ext<=unsigned('0'&a);
    b_ext<=unsigned('0'&b);
    sum_ext<=a_ext + b_ext;
    sumA<=std_logic_vector(sum_ext(N-1 downto 0));
    cout<=sum_ext(N);
end gen;
```

Recomendaciones

- Use los tipos de datos *std_logic* y *std_logic_vector*, en la declaración de entidad de puerto y para las señales internas que envuelven operaciones no aritméticas.
- Use el paquete *numeric_std* y sus tipos de datos *unsigned* y *signed* para señales internas que envuelven operaciones aritméticas.
- Use la conversión de datos de la tabla para convertir señales y expresiones.
- Usar el tipo entero y operadores aritméticos para expresiones constantes y arreglos pero no para síntesis (no usar como tipo de datos para una señal).
- Usar el tipo enumerado para los estados en una FSM.

Ejemplo : Genere el código en VHDL para el siguiente circuito



Considere que ya existen los submódulos VHDL de `hex_to_sseg` y `disp_mux`.

Código de hex_to_sseg



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

entity hex_to_sseg is
  port(
    hex : in STD_LOGIC_VECTOR(3 downto 0);
    dp  : in std_logic;
    sseg: out STD_LOGIC_vector(7 downto 0));
end hex_to_sseg;
```

architecture secuencial of hex_to_sseg is
begin

```
process (hex) begin
  case hex is
    when "0000" => sseg(6 downto 0) <= "1000000";
    when "0001" => sseg(6 downto 0) <= "1111001";
    when "0010" => sseg(6 downto 0) <= "0100100";
    when "0011" => sseg(6 downto 0) <= "0110000";
    when "0100" => sseg(6 downto 0) <= "0011001";
    when "0101" => sseg(6 downto 0) <= "0010010";
    when "0110" => sseg(6 downto 0) <= "0000010";
    when "0111" => sseg(6 downto 0) <= "1111000";
    when "1000" => sseg(6 downto 0) <= "0000000";
    when "1001" => sseg(6 downto 0) <= "0010000";
    when "1010" => sseg(6 downto 0) <= "0001000";--a
    when "1011" => sseg(6 downto 0) <= "0000011";--b
    when "1100" => sseg(6 downto 0) <= "1000110";--c

    when "1101" => sseg(6 downto 0) <= "0100001";--d

    when "1110" => sseg(6 downto 0) <= "0000100";--e
    when others => sseg(6 downto 0) <= "0001110";--f
  end case;
  sseg(7) <= dp;
end process;
end secuencial;
```

Código disp_mux (1/2)



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

entity disp_mux is
port (
    clk, reset:           in std_logic;
    in3, in2, in1, in0: in std_logic_vector(7 downto 0);
    an:                   out std_logic_vector (3 downto 0);
    sseg :                 out std_logic_vector (7 downto 0));
end disp_mux;
```

Código disp_mux (2/2)



```
architecture arch of disp_mux is
    constant N: integer :=18;
    signal q_reg , q_next : unsigned(N-1 downto
0);
    signal                                sel: std_logic_vector
(1 downto 0);
    begin
        process (clk, reset)    begin
            if (reset='1') then
                q_reg <=(others=>'0') ;
            elsif (clk'event and clk='1') then
                q_reg <= q_next;
            end if ;
        end process ;

        q_next<=q_reg+1;

        sel<=std_logic_vector(q_reg(N-1 downto N-2));
```

```
        process (sel , in0, in1 , in2, in3)
        begin
            case sel is
                when "00" =>
                    an <= "1110";
                    sseg <= in0;
                when "01" =>
                    an <= "1101";
                    sseg <= in1;
                when "10" =>
                    an<="1011";
                    sseg<=in2;
                when others =>
                    an<="0111";
                    sseg<=in3;
            end case;
        end process;
    end arch;
```

Entidad



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

entity hex_to_sseg_test is
  port (
    clk: in    std_logic;
    sw : in     std_logic_vector(7 downto 0);
    an : out std_logic_vector(3 downto 0);
    sseg:  out std_logic_vector(7 downto 0));
end hex_to_sseg_test;
```

Arquitectura



architecture Behavioral of hex_to_sseg_test is
signal inc: std_logic_vector(7 downto 0);
signal led3,led2,led1,led0:std_logic_vector(7 downto 0);

```
begin
    inc<=sw+1;
    sseg_unit_0:entity work.hex_to_sseg
        port map(hex=>sw(3 downto 0),dp=>'0',sseg=>led0);
    sseg_unit_1:entity work.hex_to_sseg
        port map(hex=>sw(7 downto 4),dp=>'0',sseg=>led1);
    sseg_unit_2:entity work.hex_to_sseg
        port map(hex=>inc(3 downto 0),dp=>'1',sseg=>led2);
    sseg_unit_3:entity work.hex_to_sseg
        port map(hex=>inc(7 downto 4),dp=>'1',sseg=>led3);

    disp_unit:entity work.disp_mux

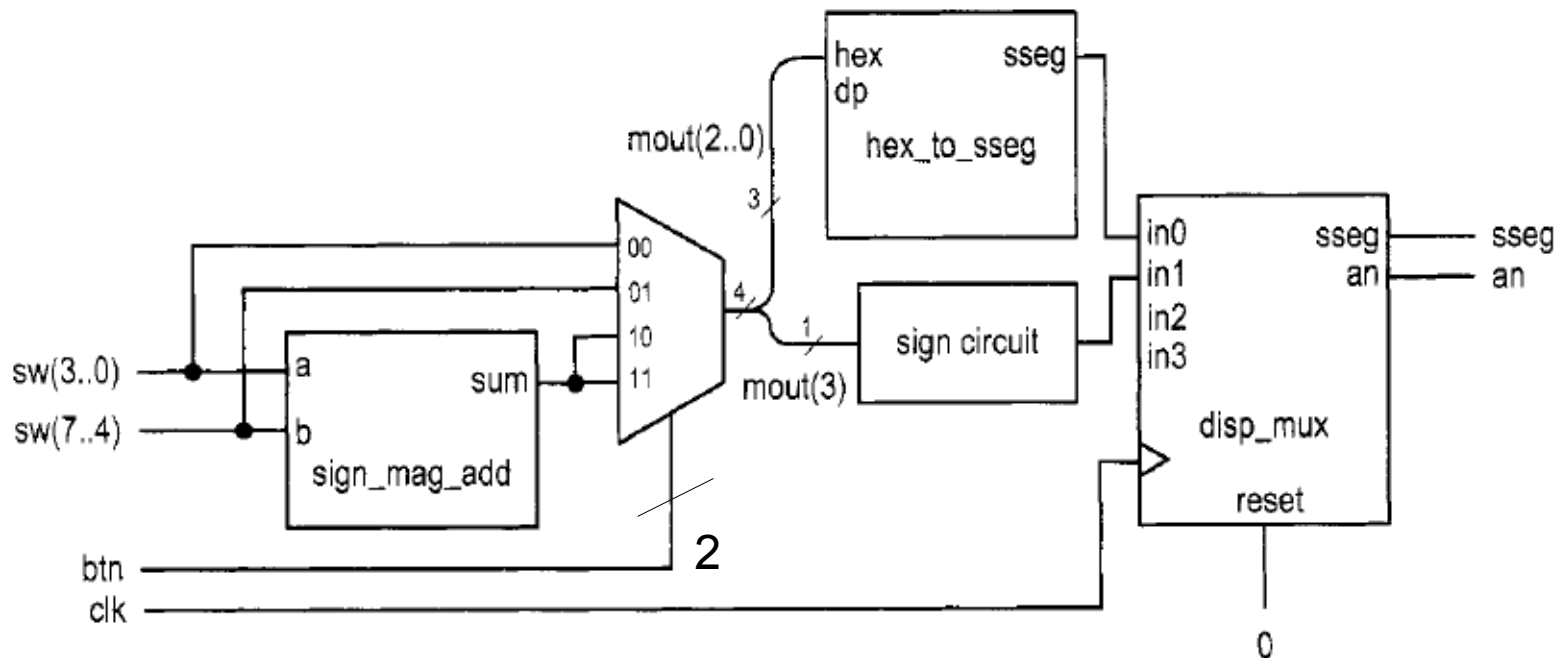
    port map(
        clk => clk, reset=>'0',
        in0=>led0, in1 => led1, in2=>led2, in3=>led3,
        an=>an, sseg=>sseg) ;
end Behavioral;
```


Ejercicio



- Implemente los circuitos de la página 68 y 75.

Sumador con signo: (por equipo)



Considere que todos los componentes que forman este dispositivo ya fueron creados.

Números de punto flotante



- Versión completa



- Versión simplificada



$$(-1)^s * \text{exp} * f$$

Sumador de punto flotante



		sort	align	add/sub	normalize
eg. 1	+0.54E3	-0.87E4	-0.87E4	-0.87E4	-0.87E4
	<u>-0.87E4</u>	<u>+0.54E3</u>	<u>+0.05E4</u>	<u>+0.05E4</u>	<u>+0.05E4</u>
				-0.82E4	-0.82E4
eg. 2	+0.54E3	-0.55E3	-0.55E3	-0.55E3	-0.55E3
	<u>-0.55E3</u>	<u>+0.54E3</u>	<u>+0.54E3</u>	<u>+0.54E3</u>	<u>+0.54E3</u>
				-0.01E3	-0.10E2
eg. 3	+0.54E0	-0.55E0	-0.55E0	-0.55E0	-0.55E0
	<u>-0.55E0</u>	<u>+0.54E0</u>	<u>+0.54E0</u>	<u>+0.54E0</u>	<u>+0.54E0</u>
				-0.01E0	-0.00E0
eg. 4	+0.56E3	+0.56E3	+0.56E3	+0.56E3	+0.56E3
	<u>+0.52E3</u>	<u>+0.52E3</u>	<u>+0.52E3</u>	<u>+0.52E3</u>	<u>+0.52E3</u>
				+1.07E3	+0.10E4

Sumador fp



$$(-1)^s * \text{exp} * f$$

- El exponente y mantisa tienen formato *unsigned*.
- La representación es normalizada, el MSB debe ser uno. Si la magnitud de la operación es menor que la magnitud no cero normalizada, está deberá ser convertida a cero.

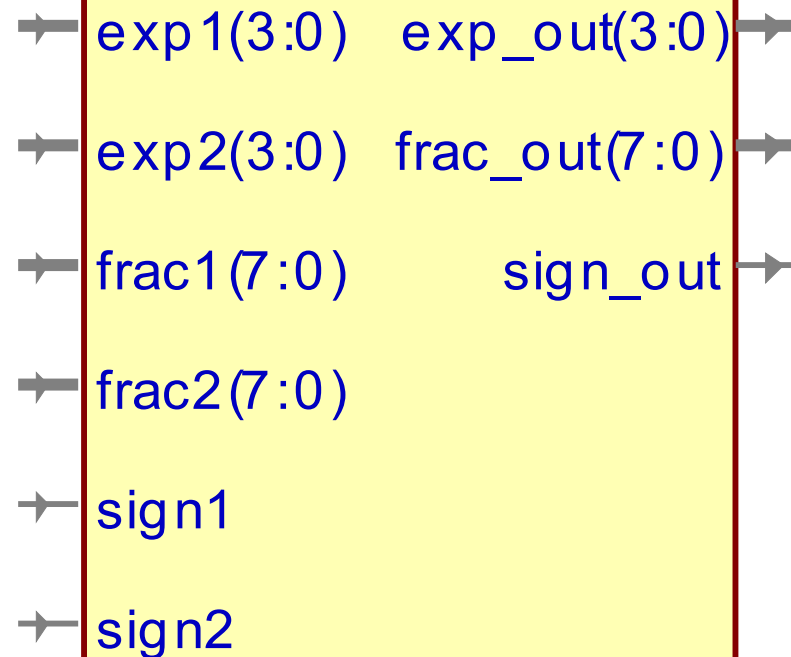
Pasos para sumar números de punto flotante



1. Ordenamiento. Seleccionar el número con la magnitud mayor.
2. Alineamiento. Cambiar el exponente del número menor al del número mayor, empleando corrimientos.
3. Suma o resta. Sumar si los números tienen el mismo signo, y restar en caso contrario.
4. Normalizar. Ajustar el resultado a un formato normalizado. Tres procedimientos pueden ser necesarios:
 - Después de una resta pueden surgir ceros después del punto.
 - Después de una resta el resultado puede ser tan pequeño que deba ser convertido a cero.
 - Si se suman puede surgir un acarreo.

Entidad

U5



fp_adder

Sumador de punto flotante 1/2



architecture arch of fp_adder is

```
signal signb, signs: std_logic;
signal expb, exps, expn: unsigned(3 downto 0);
signal fracb, fracs, fracn: unsigned(7 downto 0);
signal sum_norm: unsigned(7 downto 0);
signal exp_diff: unsigned(3 downto 0);
signal sum: unsigned(8 downto 0); --one extra for carry
signal lead0: unsigned(2 downto 0);
```

begin

-- 1st stage: sort to find the larger number

process (sign1, sign2, exp1, exp2, frac1, frac2)

begin

if (exp1 & frac1) > (exp2 & frac2) then

signb <= sign1;

signs <= sign2;

expb <= unsigned(exp1);

exps <= unsigned(exp2);

fracb <= unsigned(frac1);

fracs <= unsigned(frac2);

else

signb <= sign2;

signs <= sign1;

expb <= unsigned(exp2);

exps <= unsigned(exp1);

fracb <= unsigned(frac2);

fracs <= unsigned(frac1);

end if;

-- 2nd stage: align smaller number

exp_diff <= expb - exps;

with exp_diff select

fracn <=

fracs when "0000",
"0" & fracs(7 downto 1) when "0001",
"00" & fracs(7 downto 2) when "0010",
"000" & fracs(7 downto 2) when "0011",
"0000" & fracs(7 downto 4) when "0100",
"00000" & fracs(7 downto 5) when "0101",
"000000" & fracs(7 downto 6) when "0110",
"0000000" & fracs(7) when "0111",
"00000000"

Sumador de punto flotante 2/2



-- 3rd stage: add/subtract

```
sum <= ('0' & fracb) + ('0' & fracb) when signb=signs else
      ('0' & fracb) - ('0' & fracb);
```

-- 4th stage: normalize

-- count leading 0s

```
lead0 <= "000" when (sum(7)='1') else
      "001" when (sum(6)='1') else
      "010" when (sum(5)='1') else
      "111" when (sum(4)='1') else
      "100" when (sum(3)='1') else
      "101" when (sum(2)='1') else
      "110" when (sum(1)='1') else
      "111";
```

-- shift significand according to leading 0

with lead0 select

```
sum_norm <=
  sum(7 downto 0)      when "000",
  sum(6 downto 0) & '0' when "001",
  sum(5 downto 0) & "00" when "010",
  sum(4 downto 0) & "000" when "011",
  sum(3 downto 0) & "0000" when "100",
  sum(2 downto 0) & "00000" when "101",
  sum(1 downto 0) & "000000" when "110",
  sum(0) & "0000000" when others;
```

-- normalize with special conditions

process(sum,sum_norm,expb,lead0)

begin

if sum(8)='1' then -- w/ carry out; shift frac to right

expn <= expb + 1;

fracn <= sum(8 downto 1);

elsif (lead0 > expb) then -- too small to normalize;

expn <= (others=>'0'); -- set to 0

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

else

expn <= expb - lead0;

fracn <= sum_norm;

end if;

end process;

-- form output

sign_out <= signb;

exp_out <= std_logic_vector(expn);

frac_out <= std_logic_vector(fracn);

end arch;

Circuito de prueba



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

```
entity fp_adder_test is
  port(
    clk: in std_logic;
    sw: in std_logic_vector(7 downto 0);
    btn: in std_logic_vector(3 downto 0);
    an: out std_logic_vector(3 downto 0);
    sseg: out std_logic_vector(7 downto 0)
  );
end fp_adder_test;
```

```
architecture arch of fp_adder_test is
  signal sign1, sign2: std_logic;
  signal exp1, exp2: std_logic_vector(3 downto 0);
  signal frac1, frac2: std_logic_vector(7 downto 0);
  signal sign_out: std_logic;
  signal exp_out: std_logic_vector(3 downto 0);
  signal frac_out: std_logic_vector(7 downto 0);
  signal led3, led2, led1, led0:
    std_logic_vector(7 downto 0);
begin
  -- set up the fp adder input signals
  sign1 <= '0';
  exp1 <= "1000";
  frac1 <= '1' & sw(1) & sw(0) & "10101";
  sign2 <= sw(7);
  exp2 <= "1000";
  frac2 <= '1' & sw(6 downto 0);
```

```
-- instantiate fp adder
fp_add_unit: entity work.fp_adder
  port map(
    sign1=>sign1, sign2=>sign2, exp1=>exp1, exp2=>exp2,
    frac1=>frac1, frac2=>frac2,
    sign_out=>sign_out, exp_out=>exp_out,
    frac_out=>frac_out
  );

-- instantiate three instances of hex decoders
-- exponent
sseg_unit_0: entity work.hex_to_sseg
  port map(hex=>exp_out, dp=>'0', sseg=>led0);
-- 4 LSBs of fraction
sseg_unit_1: entity work.hex_to_sseg
  port map(hex=>frac_out(3 downto 0),
    dp=>'1', sseg=>led1);
-- 4 MSBs of fraction
sseg_unit_2: entity work.hex_to_sseg
  port map(hex=>frac_out(7 downto 4),
    dp=>'0', sseg=>led2);
-- sign
led3 <= "11111110" when sign_out='1' else -- middle bar
  "11111111"; -- blank

-- instantiate 7-seg LED display time-multiplexing module
disp_unit: entity work.disp_mux
  port map(
    clk=>clk, reset=>'0',
    in0=>led0, in1=>led1, in2=>led2, in3=>led3,
    an=>an, sseg=>sseg
  );
end arch;
```



Práctica

Comprobar el diseño del sumador de punto flotante



- Existen errores en el código fuente, corregirlos.
- Verificar que los resultados obtenidos en los displays de siete segmentos son los correctos.