Arquitectura de Computadoras Avanzada

Juan Meza

		E	G	L
	xy	$x = y$	$x > y$	$x < y$
$\bar{x}\bar{y}$	00	1	0	0
$\bar{x}y$	01	0	0	1
$x\bar{y}$	10	0	1	0
xy	11	1	0	0

$$E = \bar{x}\bar{y} + xy = \overline{(x \oplus y)}$$

$$G = x\bar{y}$$

$$L = \bar{x}y$$

El comparador de 2 bits

Ahora buscaremos diseñar una función de lógica combinacional que determine si el número x_1x_0 de 2 bits es igual que, mayor que o menor que el número y_1y_0 de 2 bits:

Los números de 2 bits x_1x_0 y y_1y_0 son cuando $x_1 = y_1$ y $x_0 = y_0$ por tanto, las funciones de igualdad de 1 bit $\overline{(x_1 \oplus y_1)}$ y $\overline{(x_0 \oplus y_0)}$ se pueden combinar con la función AND para producir la siguiente función de igualdad de 2 bits:

$$E = \overline{(x_1 \oplus y_1)} \cdot \overline{(x_0 \oplus y_0)}$$

La función mayor que, se deduce al notar que $x_1x_0 > y_1y_0$ cuando:

$$G = x_1\overline{y_1} + \overline{(x_1 \oplus y_1)}x_0\overline{y_0}$$

La función de menor que, se deduce al notar que $x_1x_0 < y_1y_0$ cuando

$$L = \overline{x_1}y_1 + \overline{(x_1 \oplus y_1)}\overline{x_0}y_0$$

El comparador de 4 bits

Un comparador de 4 bits puede diseñarse por extensión directa de las formulas de un comparador de 2 bits. Las formulas resultantes para el comparador de 4 bits son:

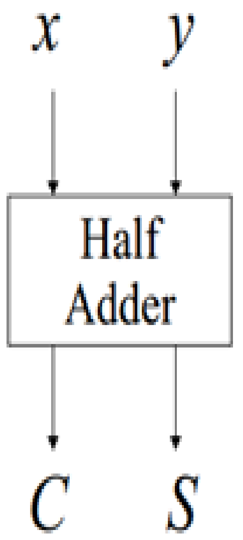
$$E = \overline{(x_3 \oplus y_3)} \cdot \overline{(x_2 \oplus y_2)} \cdot \overline{(x_1 \oplus y_1)} \cdot \overline{(x_0 \oplus y_0)}$$

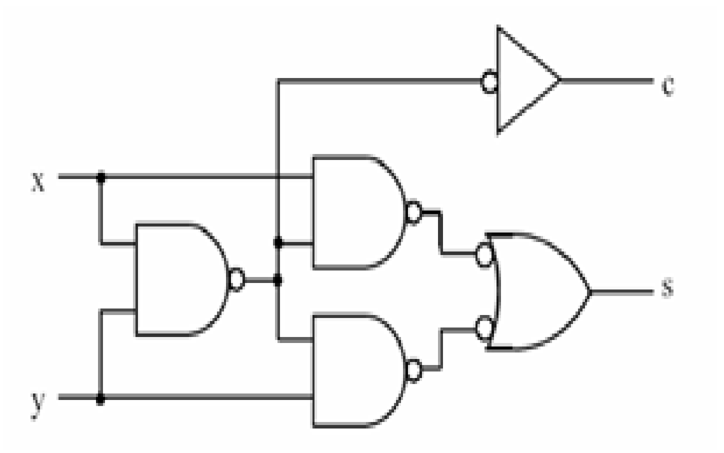
$$G = x_3\overline{y_3} + \overline{(x_3 \oplus y_3)} \cdot x_2\overline{y_2} + \overline{(x_3 \oplus y_3)} \cdot \overline{(x_2 \oplus y_2)} \cdot x_1\overline{y_1} + \overline{(x_3 \oplus y_3)} \cdot \overline{(x_2 \oplus y_2)} \cdot \overline{(x_1 \oplus y_1)} \cdot x_0\overline{y_0}$$

$$L = \overline{x_3}y_3 + \overline{(x_3 \oplus y_3)} \cdot \overline{x_2}y_2 + \overline{(x_3 \oplus y_3)} \cdot \overline{(x_2 \oplus y_2)} \cdot \overline{x_1}y_1 + \overline{(x_3 \oplus y_3)} \cdot \overline{(x_2 \oplus y_2)} \cdot \overline{(x_1 \oplus y_1)} \cdot \overline{x_0}y_0$$

Medio Sumador

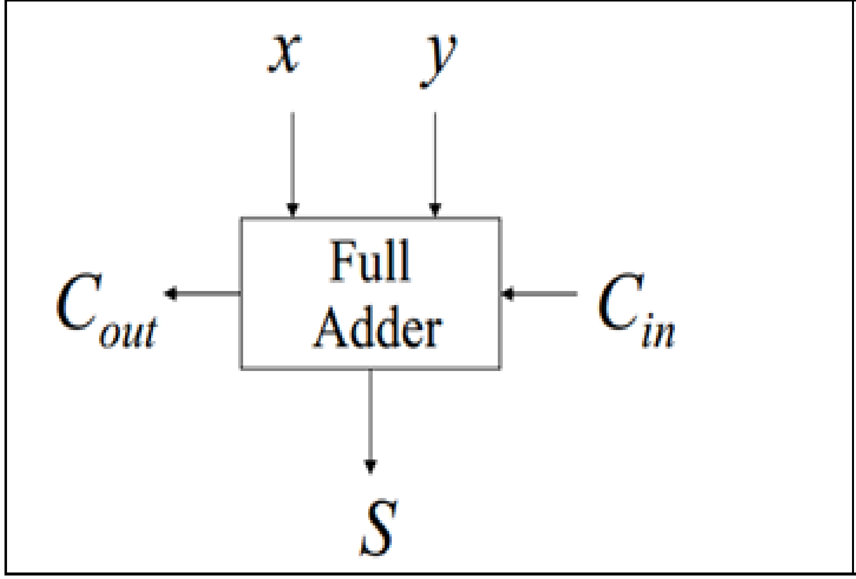
Entradas: x, y, Salidas: $s = x \oplus y$, $c = y\,x$

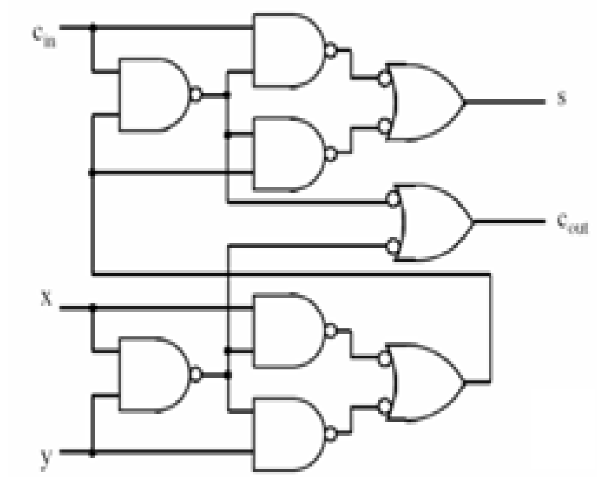


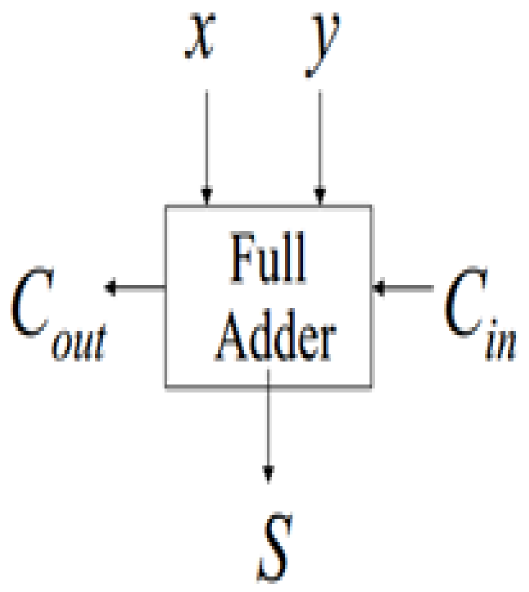


Sumador completo

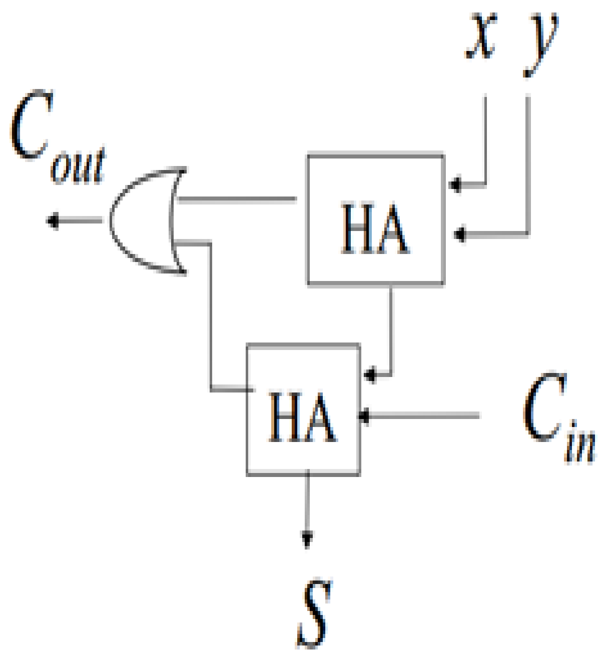
Entradas: x, y, c_{in}, Salidas: $s = x \oplus y \oplus c_{in}$ y $c_{out} = y\,x + x c_{in} + y\,c_{in}$







x
y
Full Adder
C_{out}
C_{in}
S



x
y
C_{out}
HA
HA
C_{in}
S

$$c_1 = x_0y_0 + c_0(x_0 \oplus y_0)$$

$$c_2 = x_1y_1 + \left(x_0y_0 + c_0(x_0 \oplus y_0)\right)(x_1 \oplus y_1)$$

$$c_3 = x_2y_2 + \left(x_1y_1 + \left(x_0y_0 + c_0(x_0 \oplus y_0)\right)(x_1 \oplus y_1)\right)(x_2 \oplus y_2)$$

$$c_4 = x_3y_3 + \left(x_2y_2 + \left(x_1y_1 + \left(x_0y_0 + c_0(x_0 \oplus y_0)\right)(x_1 \oplus y_1)\right)(x_2 \oplus y_2)\right)(x_3 \oplus y_3)$$

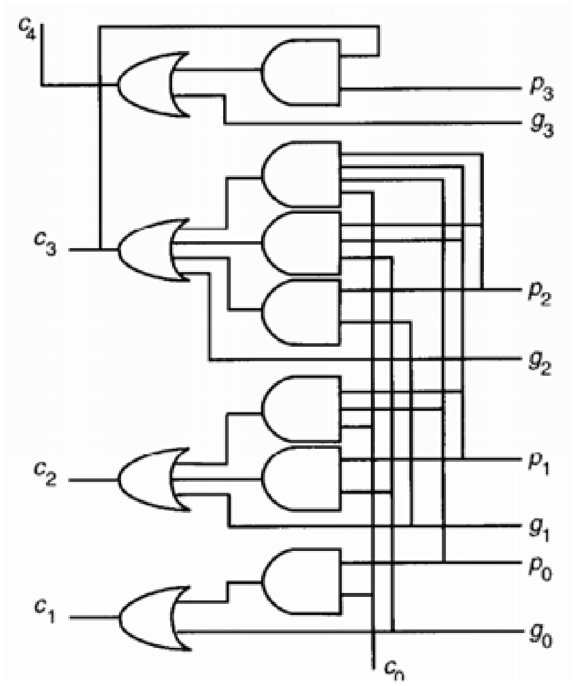
Con esto evitamos que el sumador anterior necesite del acarreo del bloque anterior

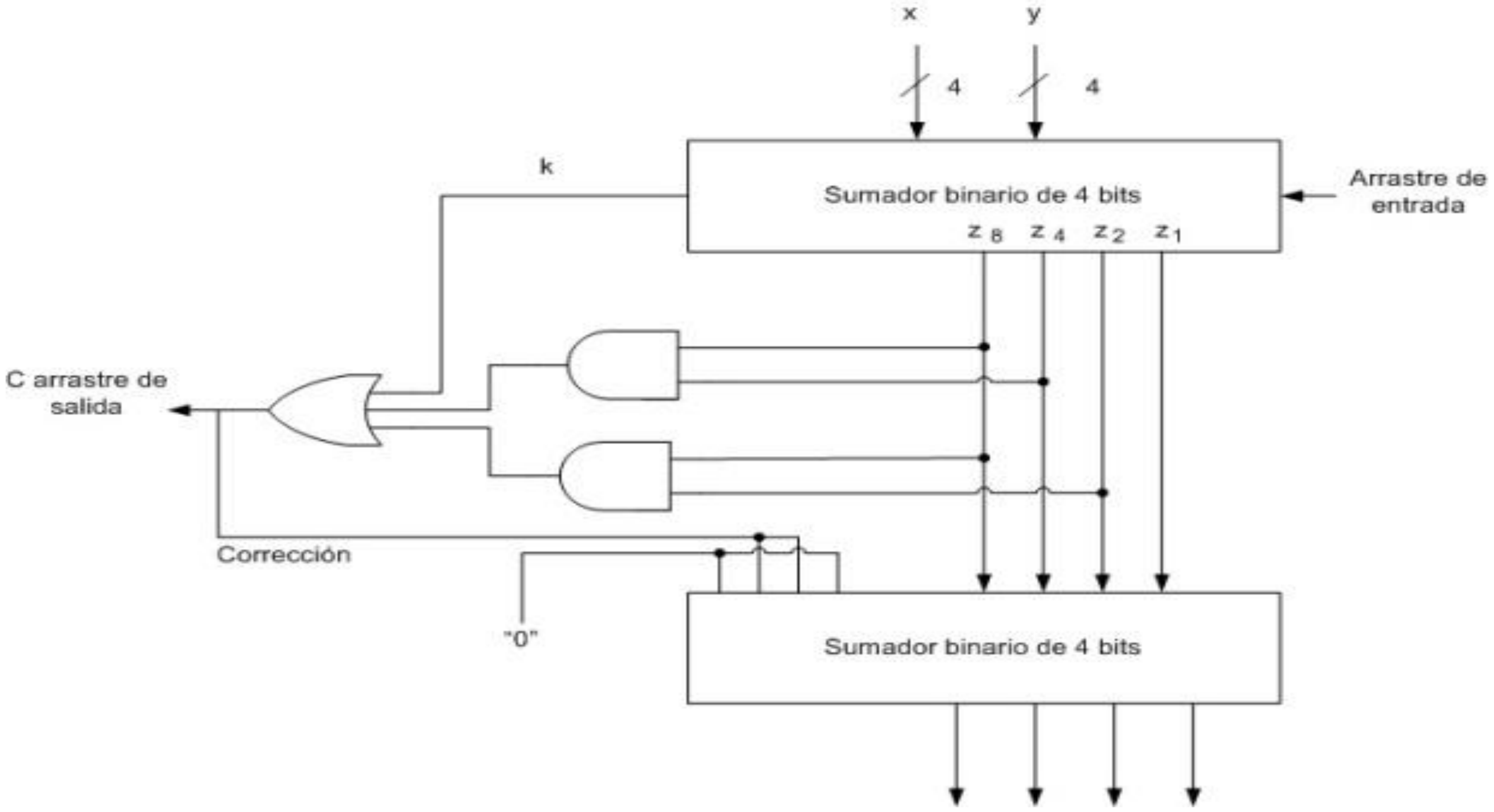
$$c_1 = G_0 + c_0P_0$$

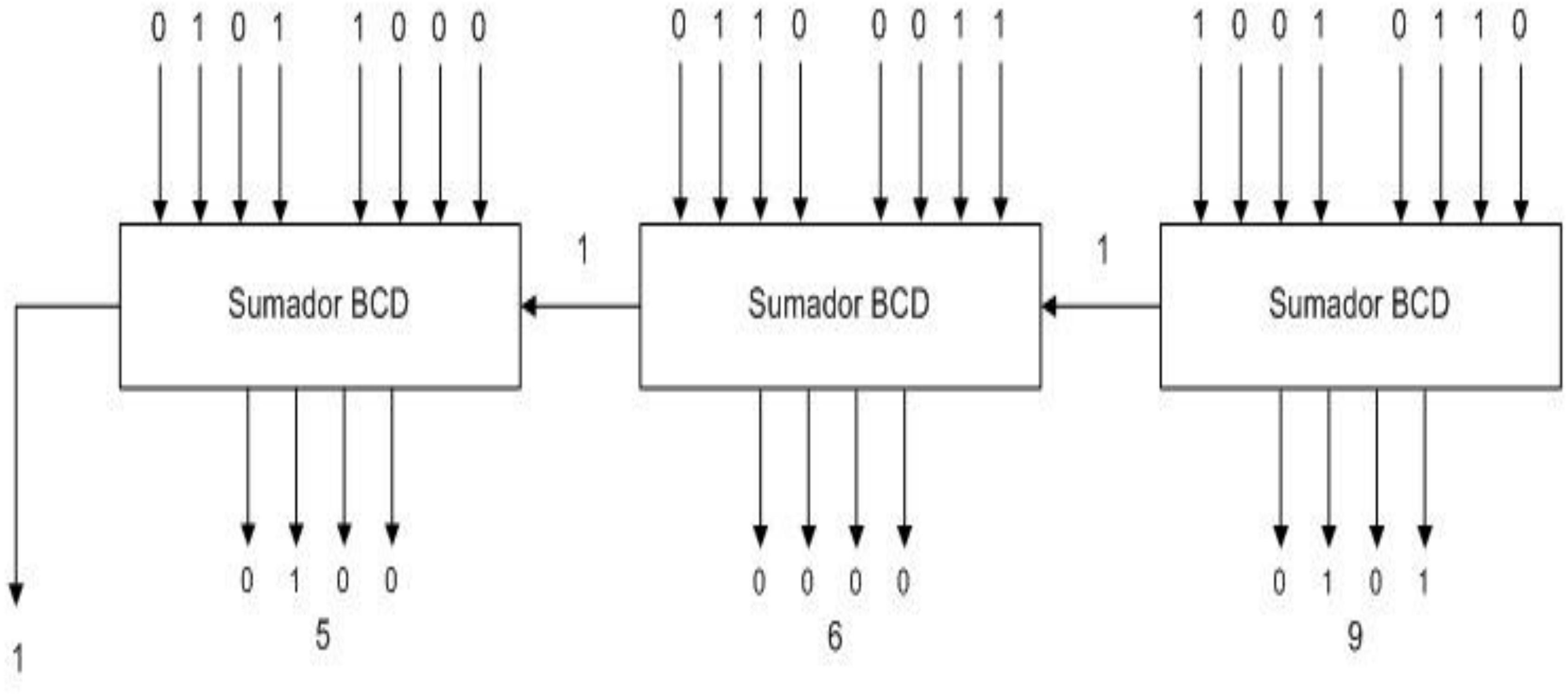
$$c_2 = G_1 + (G_0 + c_0P_0)P_1$$

$$c_3 = G_2 + (G_1 + (G_0 + c_0P_0)P_1)P_2$$

$$c_4 = G_3 + (G_2 + (G_1 + (G_0 + c_0P_0)P_1)P_2)P_3$$




x
y
4
4
k
Sumador binario de 4 bits
Arrastre de entrada
z 8
z 4
z 2
z 1
C arrastre de salida
Corrección
"0"
Sumador binario de 4 bits


0 1 0 1 1 0 0 0
0 1 1 0 0 0 1 1
1 0 0 1 0 1 1 0
Sumador BCD
Sumador BCD
Sumador BCD
1
1
1
0 1 0 0
5
0 0 0 0
6
0 1 0 1
9

Sumador 1 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity suma1 is
    port( cin, x, y : in std_logic;
            s, cout : out std_logic);
end suma1;
architecture xxx of suma1 is
    begin
          s <= cin xor x xor y;
          cout <= (x and y) or (cin and (x xor y));
end xxx;
```

Sumador con Function

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity suma1 is
        port(      x,y,Cin: in std_logic;
                                    s, Cout: out std_logic);
end suma1 ;
architecture archfull_add of suma1 is
function acarreo(a,b,c: std_logic) return std_logic is
        begin
                return ((a and b) or (c and (a xor b)));
end acarreo;
function suma(a,b,c: std_logic) return std_logic is
        begin
                return (a xor b xor c);
end suma;
begin
        s <= suma(x,y,Cin);
        Cout <= acarreo(x,y,Cin);
end architecture;
```

Sumador con Procedure

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity suma1 is
        port(     x,y,Cin: in std_logic;
                                s, Cout: out std_logic);
end suma1 ;
architecture archfull_add of suma1 is
procedure suma(variable x,y,cin :in std_logic;
                                        variable s,cout  : out std_logic) is
begin
        s:= x xor y xor cin;
        cout := (x and y) or (cin and (x xor cin));
end procedure;
begin
        process(x,y,cin)
                variable xx,yy,ccin,ss,ccout: std_logic;
begin
        xx:=x; yy:=y; ccin:=cin;
        suma(xx,yy,ccin,ss,ccout);
        s<=ss;cout<=ccout;
end process;
end architecture;
```

Sumador 1 bit: Verilog

```
module suma1(x, y, cin, s, cout);
   input x;
   input y;
   input cin;
   output s;
   output cout;
   assign s = x ^ y ^ cin;
   assign cout = (x & y) | (cin & (x ^ y));
endmodule
```

Sumador de 4 bits

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity suma4x is
    port( cin            : in std_logic;
            x, y : in std_logic_vector(3 downto 0);
            s             : out std_logic_vector(3 downto 0);
            cout         : out std_logic);
end suma4x;
architecture xxx of suma4x is
signal c : std_logic_vector(0 to 4);
    begin
            c(0) <= cin;
            s(0) <= c(0) xor x(0) xor y(0);
            c(1) <= (x(0) and y(0)) or (c(0) and (x(0) xor y(0)));
            s(1) <= c(1) xor x(1) xor y(1);
            c(2) <= (x(1) and y(1)) or (c(1) and (x(1) xor y(1)));
            s(2) <= c(2) xor x(2) xor y(2);
            c(3) <= (x(2) and y(2)) or (c(2) and (x(2) xor y(2)));
            s(3) <= c(3) xor x(3) xor y(3);
            c(4) <= (x(3) and y(3)) or (c(3) and (x(3) xor y(3)));
            cout <= c(4);
end xxx;
```

Sumador llamando componente

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity suma4 is
	port (Cin : in std_logic;
			x,y : in std_logic_vector(3 downto 0);
		s   : out std_logic_vector(3 downto 0);
			Cout : out std_logic);
end suma4;
architecture hola of suma4 is
component suma1 is
	port( cin, x, y : in std_logic;
			s, cout		: out std_logic);
end component ;
signal c : std_logic_vector(0 to 4);
	begin
	c(0)<= Cin;
	u0 : suma1 port map (c(0),x(0),y(0),s(0),c(1));
	 u1 : suma1 port map (c(1),x(1),y(1),s(1),c(2));
	u2 : suma1 port map (c(2),x(2),y(2),s(2),c(3));
	u3 : suma1 port map (c(3),x(3),y(3),s(3),c(4));
	Cout<= C(4);
end hola;
```

Sumador usando GENERATE

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY suma4  IS
      PORT(  Cin              : IN STD_LOGIC;
         x, y   :IN STD_LOGIC_VECTOR ( 3 DOWNTO 0 );
                              s                    : OUT STD_LOGIC_VECTOR ( 3 DOWNTO 0 );
                              Cout          : OUT STD_LOGIC);
END suma4;
ARCHITECTURE arq_suma OF suma4 IS
      SIGNAL c : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
      BEGIN
            c(0) <= Cin;
            CICLO : FOR I IN 0 TO 3 GENERATE
                  s(I) <= c(I) xor x(I) xor y(I);
                  c(I+1) <= (x(I) and y(I)) or (c(I) and (x(I) xor y(I)));
            END GENERATE;
            Cout <= c(4);
END arq_suma;
```

Sumador usando x'range GENERATE

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY suma4  IS
      PORT(  Cin            : IN STD_LOGIC;
         x, y   :IN STD_LOGIC_VECTOR ( 3 DOWNTO 0 );
                        s                       : OUT STD_LOGIC_VECTOR ( 3 DOWNTO 0 );
                        Cout         : OUT STD_LOGIC);
END suma4;
ARCHITECTURE arq_suma OF suma4 IS
      SIGNAL c : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
      BEGIN
            c(0) <= Cin;
            CICLO : FOR I IN x'range GENERATE
                        s(I) <= c(I) xor x(I) xor y(I);
                        c(I+1) <= (x(I) and y(I)) or (c(I) and (x(I) xor y(I)));
            END GENERATE;
            Cout <= c(4);
END arq_suma;
```

Sumador usando FOR-LOOP

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY suma4 IS
      PORT (   Cin  : IN STD_LOGIC;
        x, y : IN STD_LOGIC_VECTOR ( 3 DOWNTO 0 );
                              s    : OUT STD_LOGIC_VECTOR ( 3 DOWNTO 0 );
                              Cout : OUT STD_LOGIC);
END suma4;
ARCHITECTURE arq_suma OF suma4 IS
signal c : std_logic_VECTOR(4 DOWNTO 0);
      BEGIN
              PROCESS ( x, y, Cin )
                              BEGIN
                              c(0) <= Cin;
                              FOR I IN 0 TO 3 LOOP
                                          s(I) <= c(I) XOR x(I) XOR y(I);
                                          c(I+1) <= (x(I) and y(I)) or (c(I) and (x(I) xor y(I)));
                END LOOP;
              END PROCESS;
              Cout <= c(4);
END arq_suma;
```

```
module suma4(x, y, cin, s, cout);
input [3:0] x;
input [3:0] y;
input cin;
output [3:0] s;
output cout;
    wire c1,c2,c3;
    suma1 u0 (x[0], y[0], cin, s[0], c1);
    suma1 u1 (x[1], y[1], c1, s[1], c2);
    suma1 u2 (x[2], y[2], c2, s[2], c3);
    suma1 u3 (x[3], y[3], c3, s[3], cout);
endmodule
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity suma4 is
    port(x, y: in std_logic_vector (3 downto 0);
                      Suma: out std_logic_vector (4 downto 0));
end suma4;
architecture arqsum of suma4 is
begin
    Suma <= x + y;
end arqsum;
```

Sumador de BCD

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity suma_bcd is
        port (Arrastre_en : in std_logic;
                        x,y : in std_logic_vector(3 downto 0);
                s   : out std_logic_vector(3 downto 0);
                        Arrastre_sa : out std_logic);
end suma_bcd;
architecture behav of suma_bcd is
signal k,C,nada,temp1, temp2, temp3 : std_logic;
signal z,xx : std_logic_vector(3 downto 0);
component suma4 is
        port (Cin : in std_logic;
                        x,y : in std_logic_vector(3 downto 0);
                s   : out std_logic_vector(3 downto 0);
                        Cout : out std_logic);
end component;
begin
        U0 : suma4 port map(Arrastre_en, x,y, z, k);
        U1 : suma4 port map(C, xx,z, s, nada);
        C <= '0';
        xx(0) <= '0';
        xx(3) <= '0';
        xx(1) <= temp3;
        xx(2) <= temp3;
        temp1 <= z(3) and z(2);
        temp2 <= z(3) and z(1);
        temp3 <= k or temp1 or temp2;
        Arrastre_sa <= temp3;
end behav;
```

Sumador con acarreo adelantado

```
module sumador_cla_4bits (x, y, cin, s, cout);
input [3:0] x, y;
input cin;
output [3:0] s;
output cout;
 wire p0,g0, p1,g1, p2,g2, p3,g3;
wire c4, c3, c2, c1;
assign p0 = x[0] ^ y[0], p1 = x[1] ^ y[1], p2 = x[2] ^ y[2], p3 = x[3] ^ y[3];
assign g0 = x[0] & y[0], g1 = x[1] & y[1], g2 = x[2] & y[2], g3 = x[3] & y[3];
assign c1 = g0 | (p0 & cin), c2 = g1 | (p1 & g0) | (p1 & p0 & cin), c3 = g2 | (p2 & g1) |
      (p2 & p1 & g0) | (p2 & p1 & p0 & cin),
 c4 = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0) | (p3 & p2 & p1 & p0 &
      cin);
assign s[0] = p0 ^ cin, s[1] = p1 ^ c1, s[2] = p2 ^ c2, s[3] = p3 ^ c3;
assign cout = c4;
endmodule
```

restador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity suma4_resta is
        port( w  : in std_logic;
                x, y  : in std_logic_vector( 3 downto 0);
                s     : out std_logic_vector( 3 downto 0);
                    cout  : out std_logic);
end suma4x_resta;
architecture xxx of suma4_resta IS
component suma4 is
        port(    cin     : in std_logic;
                    x, y    :  in std_logic_vector( 3 downto 0);
                    s       : out std_logic_vector( 3 downto 0);
                    cout    : out std_logic);
end component ;
signal xx : std_logic_vector(3 downto 0);
begin
        xx(0) <= w xor x(0);
        xx(1) <= w xor x(1);
        xx(2) <= w xor x(2);
        xx(3) <= w xor x(3);
        u0 : suma4 port map(w, xx, y , s, cout);
end xxx;
```

```
module suma4_resta (w,x,y,s,cout);
  input w;
  input [3:0] x;
  input [3:0] y;
  output [4:0] s;
  output cout;
  reg  [4:0] s;
  reg  cout;
  wire [3:0] xx;
  assign xx[0] = x[0] ^ w;
  assign xx[1] = x[1] ^ w;
  assign xx[2] = x[2] ^ w;
  assign xx[3] = x[3] ^ w;
  sumador_cla_4bits i_sumador_cla_4bits (
                .x (xx[3:0]),
                .y (y[3:0]),
                .cin (w),
                .s (s[4:0]),
                .cout (cout)
       );
Endmodule
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
 entity SumaResta is
     port (              A : in  STD_LOGIC_VECTOR (3 downto 0);
                         B : in  STD_LOGIC_VECTOR (3 downto 0);
                         Sel : in  STD_LOGIC;
                         Salida : out  STD_LOGIC_VECTOR (3 downto 0));
end SumaResta;
 architecture Behavioral of SumaResta is
begin
     Salida <= ( A - B ) when Sel = '1' else ( A + B );
end Behavioral;
```