2. Sumadores

Los sumadores son cirtuitos muy utilizados en muchos tipos de sistemas digitales en los que se procesan datos numéricos. Para comprender su diseño y funcionamiento se parte del diseño de un semisumador.

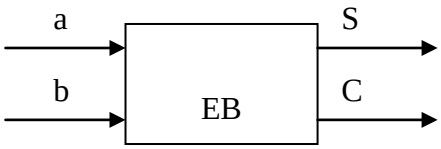
2.1. Diseño

Semi-sumador (half-adder)

Se debe realizar la suma de dos números de un único bit.

Diagrama *Tabla de verdad*

Diagrama

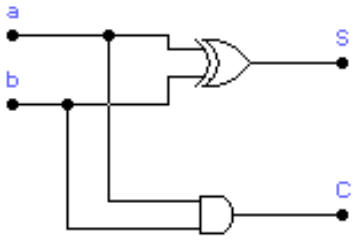


Donde a y b son los bits a sumar, S el resultado de la suma y C el acarreo generado.

Tabla de verdad

a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Circuito



Sumador completo (full-adder)

El diseño anterior es poco útil ya que habitualmente los números a sumar tienen más de un bit. Si recordamos el procedimiento de suma, esta debe realizarse bit a bit tal y como se muestra en la Figura 83.

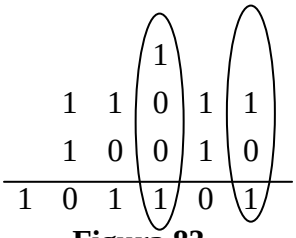
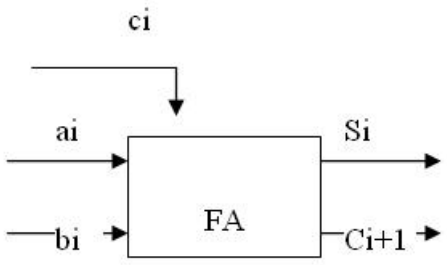


Figura 83

Se observa que en la operacion de suma para cada pareja de bits, debe incluirse el acarreo producido por la suma de los bits anteriores. Así pues, es necesario disponer de un circuito que realice la suma incluido el acarreo anterior. Este tipo de circuito se denomina Sumador Completo (Full Adder).

Diagrama



Donde ai y bi son los bits a sumar y ci el acarreo de proveniente del resultado de una suma anterior, S el resultado de la suma y C el acarreo.

Tabla de Verdad

ci	ai	bi	Si	Ci+1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

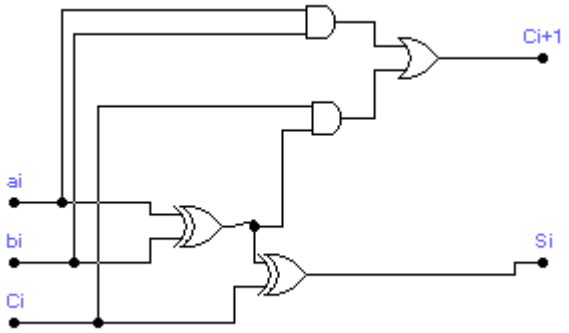
Circuito

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i + c_i (a_i + b_i)$$

o

$$c_{i+1} = a_i b_i + c_i (a_i \oplus b_i)$$



2.2. Sumador binario de acarreo en cascada (ripple-carry adder)

Conectando los Sumadores Completos necesarios se logra el circuito que realiza la suma de dos números binarios de n bits (Figura 84). La salida del acarreo de cada sumador completo se conecta a la entrada de acarreo de la siguiente etapa de orden inmediatamente superior. Así, la suma y el acarreo de salida de cualquier etapa no se puede generar hasta que esté disponible el acarreo de la etapa anterior. Por lo tanto, el resultado de la operación no podrá "leerse" hasta que se hayan generado todos los acarreos de todas las etapas. Este tiempo de retardo será $n*t_{FA}$ donde n es el número de bits de cada operando y t_{FA} el tiempo de retardo de cada etapa de Sumador Completo.

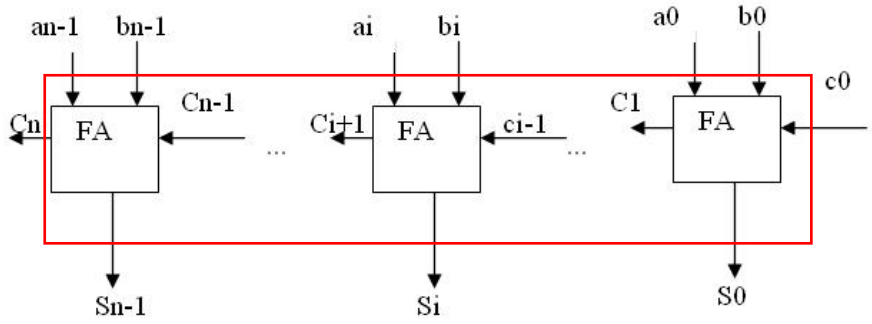


Figura 84

2.3. Sumador rápido – sumador con acarreo anticipado (fast-adder)

Existe un método que permite acelerar el poceso de adición eliminando ese retardo del acarreo serie. Es el denominado acarreo anticipado y consiste en modificar la estructura del circuito, es decir, modificar las funciones obtenidas que son las presentadas en Ecuación 4:

$$s_i = a_i \oplus b_i \oplus c_i \qquad c_{i+1} = a_i b_i + c_i (a_i + b_i)$$

Ecuación 4

El objetivo será lograr una expresión para el acarreo de cada etapa que dependa de los datos disponibles que son a_i, b_i y c_0. Para ello, se reescriben las ecuacioens de Ecuación 4 tal y comose muestra en Ecuación 5.

$$\begin{aligned} c_{i+1} &= a_i b_i + c_i a_i + c_i b_i \\ c_{i+1} &= a_i b_i + (a_i + b_i) c_i \\ \boxed{c_{i+1} &= g_i + p_i c_i} \end{aligned} \qquad \begin{aligned} g_i &= a_i b_i \\ p_i &= a_i + b_i \end{aligned} \Rightarrow \qquad \boxed{S_i = p_i \oplus g_i \oplus c_i}$$

Ecuación 5

Donde g_i representa la generación de acarreo y p_i la propagación de acarreo. Analicemos el significado de estos términos.

Si gi+1 es 1, entonces existirá acarreo ya que Ci+1 será 1 independientemente del valor de ci.

Si en alguna de las etapas el carry es 1, se propagará a la sigueitne etapa si pi es 1 también. En caso contrario, no habrá propagación de carry.

Veamos un ejemplo para el caso de dos números de dos bits:

$$\begin{array}{cccc} & c_1 & c_0 & \\ & x_1 & x_0 & \\ & y_1 & y_0 & \\ \hline c_2 & s_1 & s_0 & \end{array}$$

a) Diseño mediante sumador serie:

$$s_i = a_i \oplus b_i \oplus c_i \qquad c_{i+1} = a_i b_i + c_i (a_i + b_i)$$

i= 0	i= 1
$s_0 = x_0 \oplus y_0 \oplus c_0$ $c_1 = x_0 y_0 + c_0 (x_0 + y_0)$	$s_1 = x_1 \oplus y_1 \oplus c_1$ $c_2 = x_1 y_1 + c_1 (x_1 + y_1)$

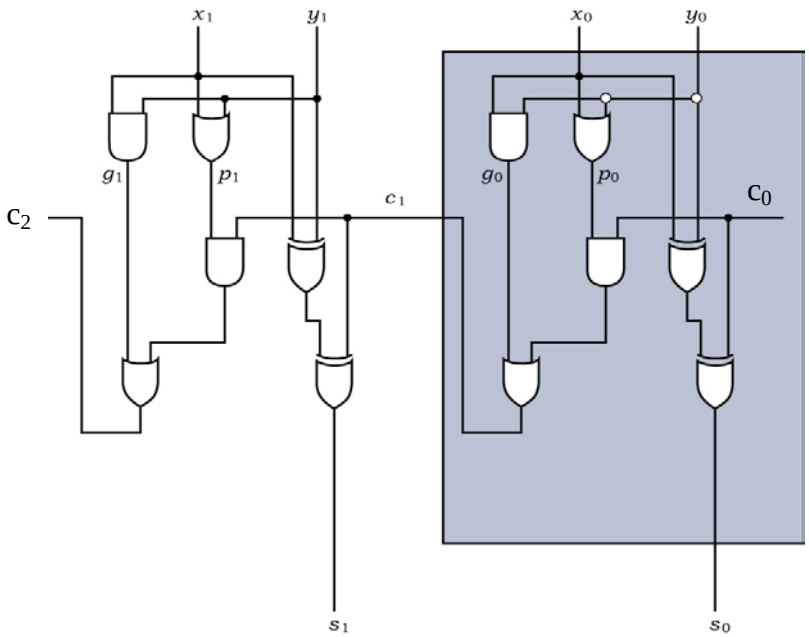


Figura 85. Primeras dos etapas del sumador con acarreo en cascada

b) Diseño mediante sumador rápido:

$$s_i = p_i \oplus g_i \qquad g_i = a_i b_i$$
$$c_{i+1} = g_i + p_i c_i \qquad p_i = a_i + b_i$$

i= 0		i= 1	
$s_0 = p_0 \oplus g_0 \oplus c_0$	$g_0 = x_0 y_0$	$s_1 = p_1 \oplus g_1 \oplus c_1$	
$c_1 = g_0 + p_0 c_0$	$p_0 = x_0 + y_0$	$c_2 = g_1 + p_1 c_1$	$g_1 = x_1 y_1$
		$c_2 = g_1 + p_1 (g_0 + p_0 c_0)$	$p_1 = x_1 + y_1$
		$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$	

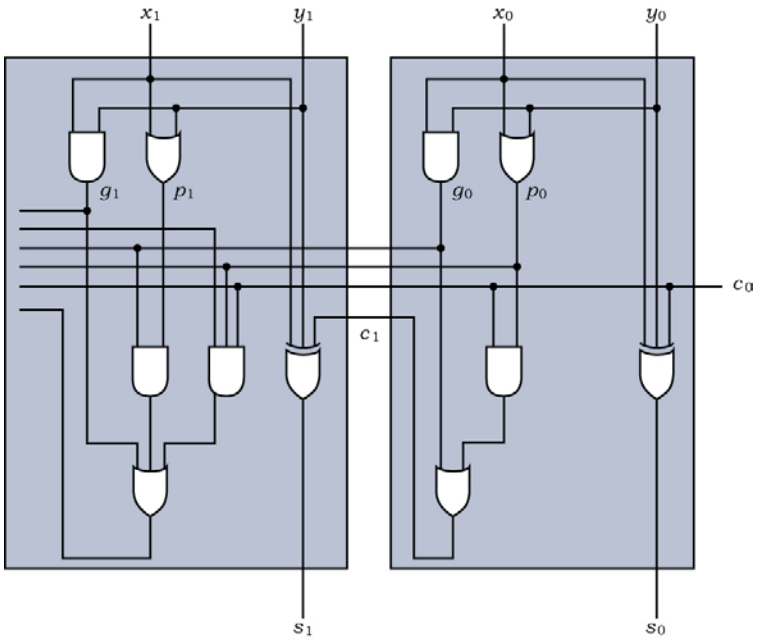


Figura 86. Primeras Dos etapas del sumador rápido

A primera vista podría concluirse que el diseño del sumador rápido es mejor, ya que el aumento de etapas no supone un aumento del los tiempos de retardo, tal y como puede observarse en la Figura 87. En ella, se ha implementado el circuito que realiza la suma de dos números de cuatro bits, es decir, el circuito que realiza la operación representada en la Ecuación 6.

$$\begin{array}{cccc} c_3 & c_2 & c_1 & c_0 \\ x_3 & x_2 & x_1 & x_0 \\ y_3 & y_2 & y_1 & y_0 \\ \hline c_4 & s_3 & s_2 & s_1 & s_0 \end{array}$$

Ecuación 6

Cabe destacar que en el circuito de la Figura 87 se han utilizado sumadores completos en lugar de las puertas exor utilizadas en la Figura 86. Esto es debido a que la salida Si del sumador completo coincide con la salida de la función exor para tres entradas.

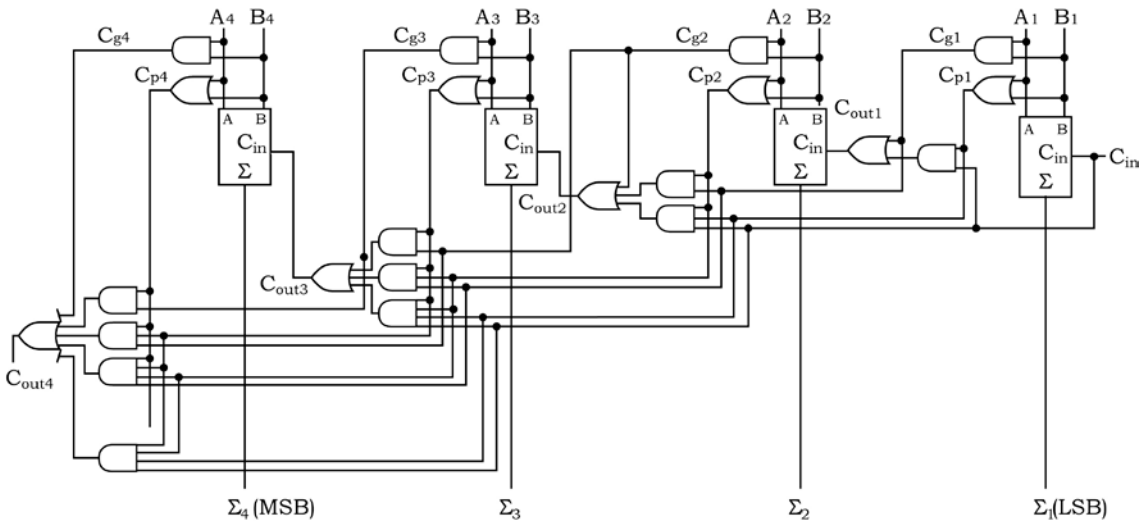


Figura 87. Sumador de 4 etapas con acarreo anticipado

Si analizamos este tipo de circuitos, se observa que todos los acarreos se obtienen tras un retardo de tres puertas. Efectivamante, este hecho se mantiene independientemente del valor de *n* siempre y cuando se utilicen puertas AND con un número mayor de entradas. Por lo tanto, el problema surge para valores mayores de *n*, por ejemplo *n*=7. En este caso, se requeriría una puerta AND de 9 entradas. Como este tipo de puertas no existe, habría que dividir y utilizar puertas de menor número de entradas (por ejemplo 4), lo cual supondría un aumento de los retardos por puerta. Asi pues, se puede concluir que con lo susmadore srápidos los tiempos de retardo no se reducen considerablemnte para cualquier valor de *n*.

2.4. Circuitos comerciales: 7483 y 74283

Dos ejemplos de sumadores de dos números de 4 bits son el 74LS83 y 74LS283. En la Figura 88 se muestra el patillaje. Los parámetros temporales pueden consultarse en sus hojas de características.

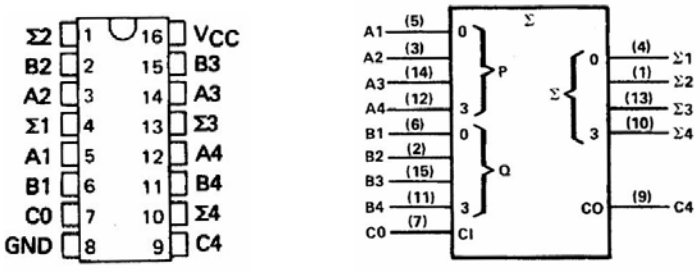


Figura 88

En la Figura 89 se muestra el diagrama lógico correspondiente al circuito integrado 74LS283. Comparándolo con la Figura 87 podría decirse que se trata de otra función, sin embargo, únicamente se han modificado las funciones lógicas según la Ecuación 7, es decir, se ha variado la estructura pero no la funcionalidad.

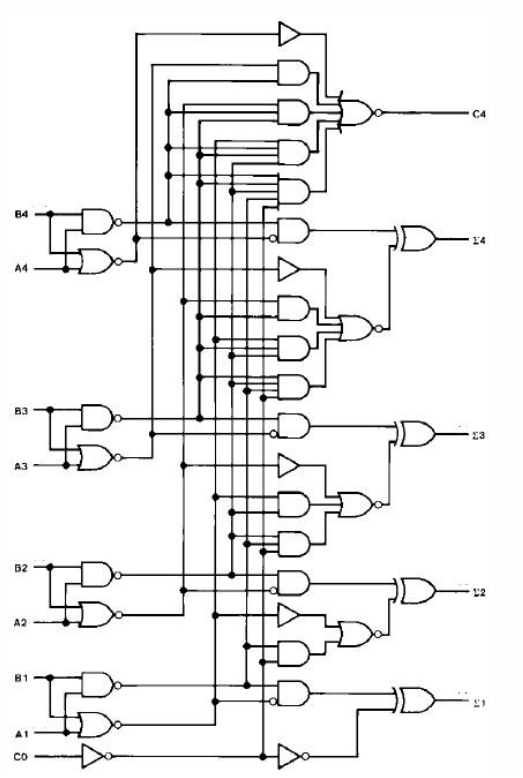


Figura 89

$$s_i = a_i \oplus b_i \oplus c_i = \left(\overline{a_i} b_i + a_i \overline{b_i}\right) \oplus c_i = \left((a+b).\left(\overline{a}+\overline{b}\right)\right) \oplus c_i = \left(\left(\overline{\overline{a+b}}\right).\left(\overline{a.b}\right)\right) \oplus c_i$$

Ecuación 7

2.5. Aplicación

Un ejemplo de aplicación es un sistema de recuento de votos impelementado a partir de sumadores de dos números de 4 bits y sumadores completos. Si observamos la Tabla de Verdad del sumador completo (Tabla 25) y tomamos la salida como la representación de un número binario de dos bits $C_{i+1}S_i$, entonces ese número representa la cantidad de 1s en las entradas.

ci	ai	bi	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabla 25

Finalmente, basta con sumar las salidas de los Sumadores Completos para obtener la cantidad de 1s totales. El circutio completo, se representa en la Figura 90. El circuito contará los botos favorable y los votos en contra una vez configurados adecuadamante los pulsadores.

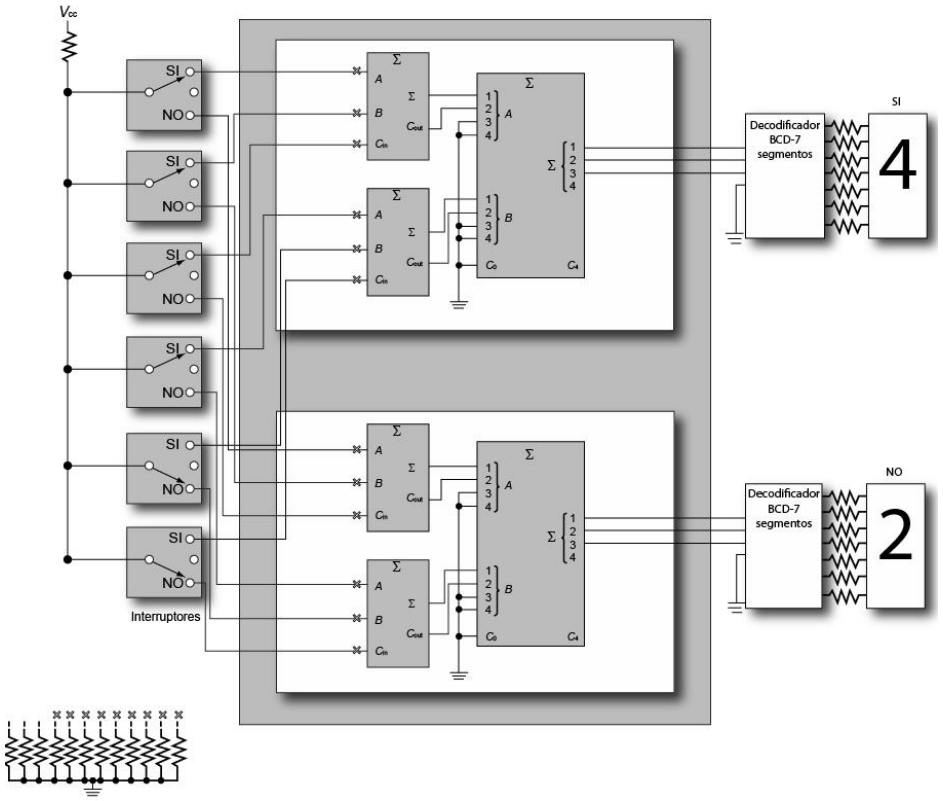


Figura 90

2.6. Descripción VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity sumador is port(
a,b: in std_logic_vector(3 downto 0);
cin: in std_logic;
sum: out std_logic_vector(4 downto 0)
);
end sumador;
1   architecture archisumador of sumador is
2   begin
3   process (a,b,cin)
4   variable aux:std_logic_vector(4 downto 0);
5   begin
6   aux:=('0' & a) + ('0' & b);
7   if cin='1' then aux:=aux+1;
8   elsif cin='0' then null;
9   end if;
10  sum<=aux;
11  end process;
12  end archisumador;
```