Celdas estándar

29 de agosto de 2024

Teniendo en cuenta que los transistores NMOS "Encienden" cuando V_{I_i} ='1', y los PMOS cuando V_{I_i} ='0', se diseñan los denominados circuitos CMOS compuestos de transistores NMOS y PMOS conectados de manera complementaria.

Para ello se utiliza un grupo de transistores como *Pull up network (PUN)* que conecta F a V_{DD} cuando $F(I_1, I_2, \ldots, I_n) = 1$ y otro grupo *Pull down network (PDN)* que conecta F a V_{SS} cuando $F(I_1, I_2, \ldots, I_n) = 0$, tal como se ilustra en la figura 1.

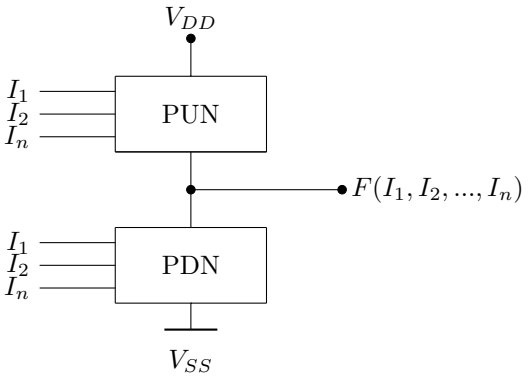

Figura 1: Redes pull-up y pull-down en CMOS

- **Observación:** Noise Margins, punto de conmutación, retardo, y potencia consumida dependen de la entrada que se trate, ya que las capacidades y resistencia de salida no son iguales para las redes PON y PUN. Además, V_T de cada transistor se ve modificado por $V_T = V_{TO} + \gamma\sqrt{|-2\phi_F + V_{SB}|} - \sqrt{|2\phi_F|}$

A continuación se muestran los distintos bloques básicos CMOS con los cuales se puede lograr cualquier función lógica al combinarlos en redes pull-up y pull-down.

1. PMOS Serie (NOR)

En la figura 2 se puede ver un PMOS en configuración serie, el mismo conduce cuando todos los transistores están encendidos (ON) es decir:

$$V_{I1} = V_{I2} = \ldots = V_{In} = 0$$

Esto equivale a la función lógica NOR:

$$F = \underbrace{\overline{I_1 + I_2 + \ldots + I_N}}_{\text{NOR}}$$

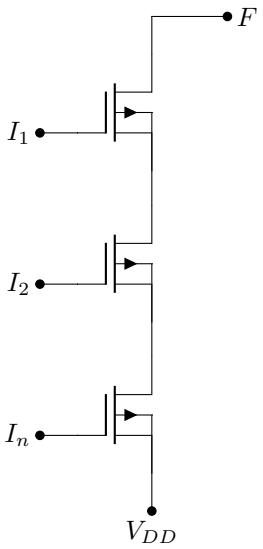

Figura 2: PMOS serie (NOR)

2. PMOS Paralelo (NAND)

En la figura 3 se puede ver un PMOS en configuración paralelo, el mismo conduce cuando algún transistor está encencido, es decir algún transistor tiene su entrada en $I_n = 0$. Esto equivale a la función lógica NAND:

$$F = \underbrace{\overline{I_1 \cdot I_2 \cdot \ldots \cdot I_N}}_{\text{NAND}}$$

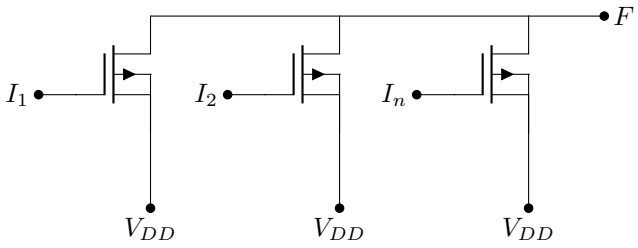

Figura 3: PMOS paralelo (NAND)

3. NMOS Serie (NAND)

El mismo se representa en la figura 4 y conduce cuando todos los transistores están encendidos: $V_{I1} = V_{I2} = \ldots = V_{In} = 1$ equivalente a la función lógica NAND:

$$F = \underbrace{\overline{I_1 \cdot I_2 \cdot \ldots \cdot I_N}}_{\text{NAND}}$$

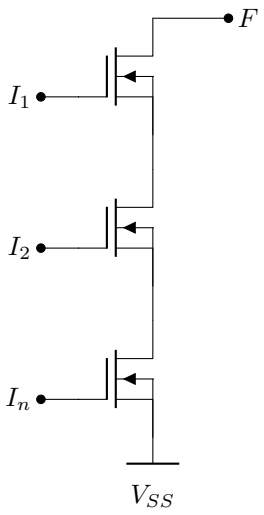

Figura 4: NMOS serie (NAND)

4. NMOS Paralelo (NOR)

Por último en la figura 5 se ve que el mismo responde a la misma ecuación que el PMOS serie, esto es cuando cualquier transistor está encendido ($I_N = 1$) la salida es 0.

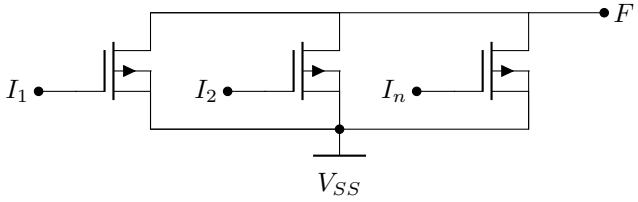

Figura 5: NMOS paralelo (NOR)

$$F = \underbrace{\overline{I_1 + I_2 + \ldots + I_N}}_{\text{NOR}}$$

Finalmente en la figura 6 se ven las dos funciones lógica, NAND y NOR en su configuración complementaria.

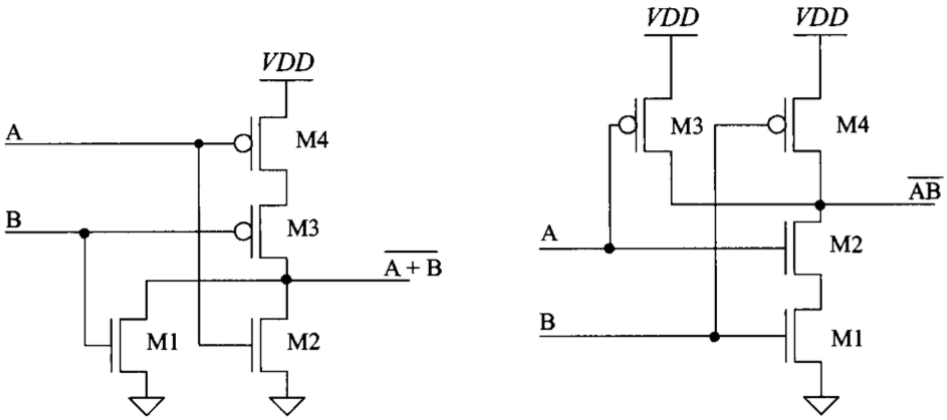

Figura 6: NOR y NAND en configuración CMOS

5. Buffer

Consiste de dos inversores en cascada de forma tal que la función lógica sea la identidad.

XXXXXXXXXXXXXXXXXXX FIGURA XXXXXXXXXXXXXXXXXXX

6. Buffer 3-State

En la Figura 7 se ve un buffer 3-State.

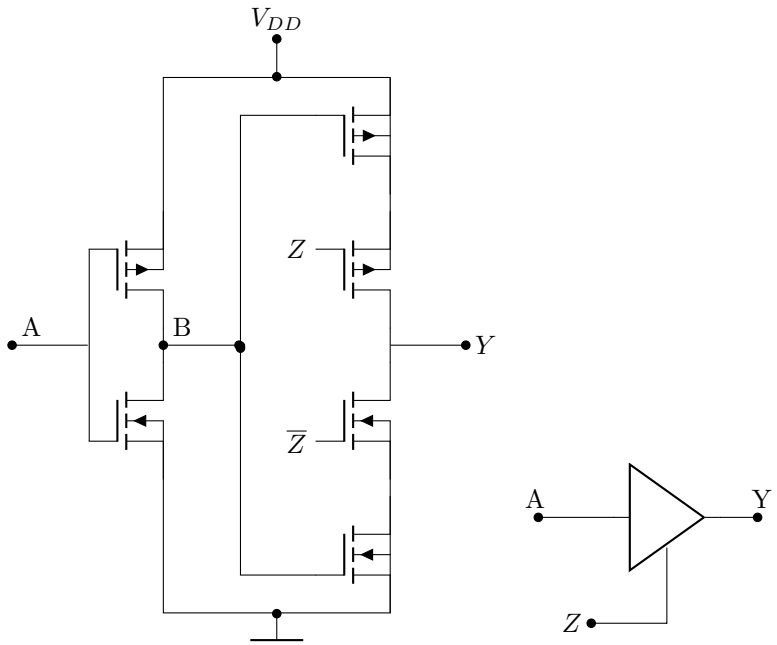

Figura 7: Buffer 3 state circuito y símbolos utilizados

7. Transmission gate

Estos circuitos se contruyen en base a las propiedades complementarias de los transistores NMOS y PMOS:

- **Dispositivos NMOS:** aportan un '0' "fuerte" pero un '1' "debil".
- **Dispositivos PMOS:** aportan un '1' "fuerte" pero un '0' "debil".

Luego, la idea es utilizar el dispositivo NMOS como Pull-Down y el PMOS como Pull-Up, conectándolos en paralelo.

Los transmission gates se comportan como switches bidireccionales, controlados por las señales de control, C y $\overline{C}$, que son complementarias.

- $C = 1$ En este caso, ambos mosfet están encendidos, permitiendo el paso de señal a través de la compuerta $=> A = B$.
  - **Entrada** $V_i = V_{DD}$

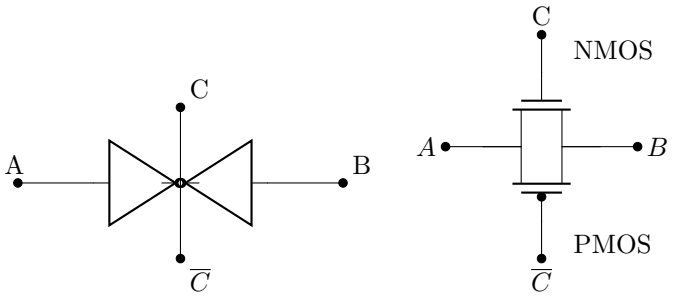

Figura 8: Transmission Gate, Simbolo y Circuito

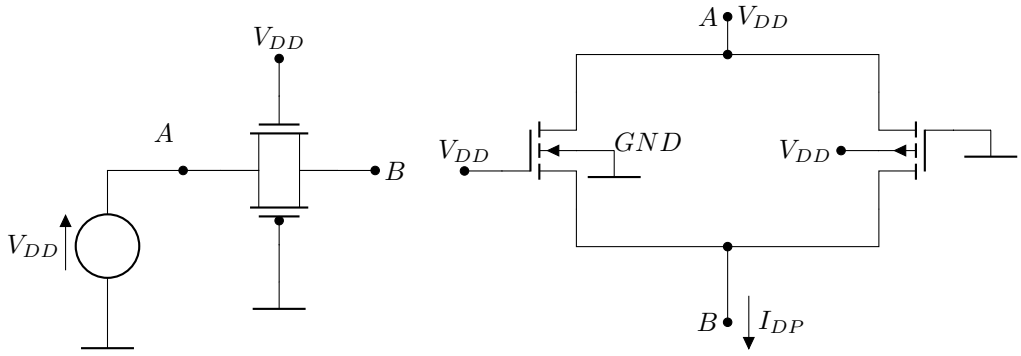

Figura 9: Transmission Gate con $V_i = V_{DD}$, $I_{D_P} = 0$

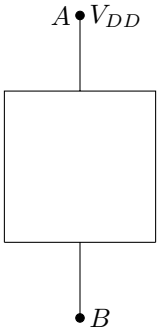

Figura 10: Circuito Equivalente

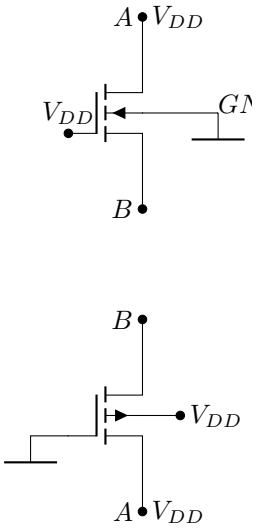

Figura 11: Dispositivos

$V_{GS} > V_{T_N} \Rightarrow V_B = V_{DD} - V_{T_N} < V_{DD}$
$V_{T_N} = V_{TO_N} + \gamma\sqrt{|-2\Phi_F + V_B|} - \sqrt{|2\Phi_F|}$
Luego:
$V_B = V_{DD} - VT_{ON} + \gamma\sqrt{|-2\Phi_F + V_B|} - \sqrt{|2\Phi_F|}$
$V_{SG} = V_{DD} > V_T$
$V_B = V_{DD}$

- **Entrada** $V_i = 0$

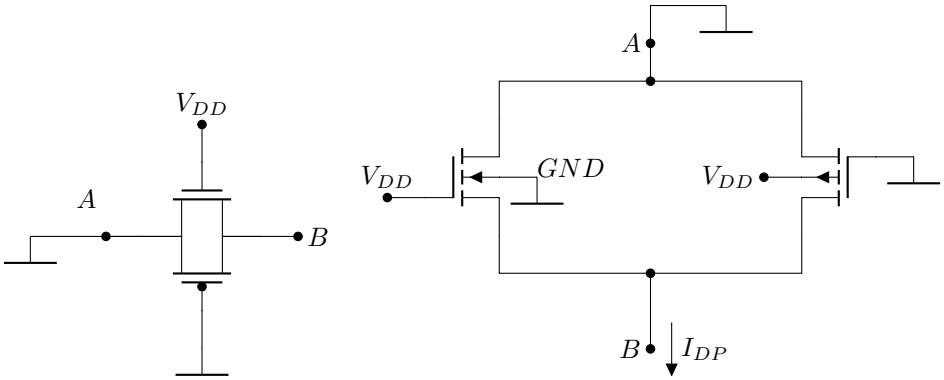

Figura 12: Transmission Gate con $V_i = 0$, $I_{D_P} = 0$

$V_{GS} = V_{DD} > V_T$
$V_B = 0$
$V_{SG} > V_{T_P} \Rightarrow V_B = V_{T_P} > 0$
$V_{T_P} = V_{TO_P} + \gamma\sqrt{|-2\Phi_F + (V_{DD} - V_B)|} - \sqrt{|2\Phi_F|}$
Luego:
$V_B = V_{TO_P} + \gamma\sqrt{|-2\Phi_F + (V_{DD} - V_B)|} - \sqrt{|2\Phi_F|}$

- $C = 0$ Pone a ambos transistores en corte, creando así un circuito abierto entre A y B.

Como se detalla más adelante, con los transmission gates se pueden crear compuertas complejas y muy eficientes.

8. Multiplexor basado en TG

El primer circuito a analizar es el multiplexor de dos entradas de datos y una de selección.

La función lógica de este componente es:

$$Y = A \cdot \overline{S} + B \cdot S$$

Con compuertas tradicionales se implementa como sigue:

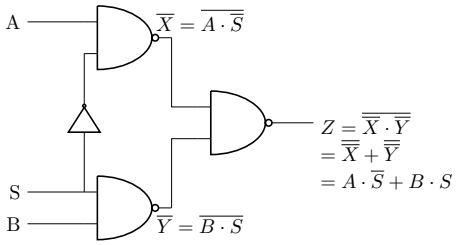

Figura 13: Multiplexor estandar

En total se necesitan 14 transistores (4 por cada NAND más 2 por el Inversor).

En cambio, si se implementa con transmission gate, se requieren 12 transistores (4 de los dos transmission gates, 8 para el resto de las compurtas).

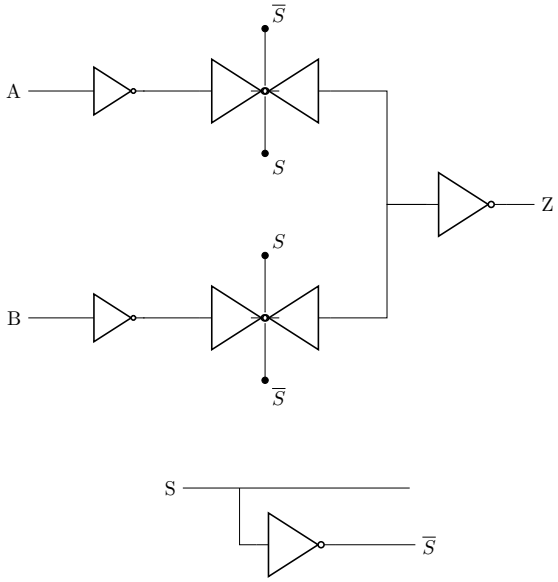

Figura 14: Multiplexor basado en transmission gates

Para mayor cantidad de entradas y lineas de seleccion se ahorran muchos transistores.

9. XOR Basada en TG

La función booleana XOR esta representada por:

$$A \oplus B = \overline{A} \cdot B + \overline{A} \cdot B$$

Vemos que para un MUX:

$Z = \overline{A} \cdot S + B \cdot S \Rightarrow$ si $B = \overline{A}$ y $S = B$, la función lógica del MUX se transforma en XOR.

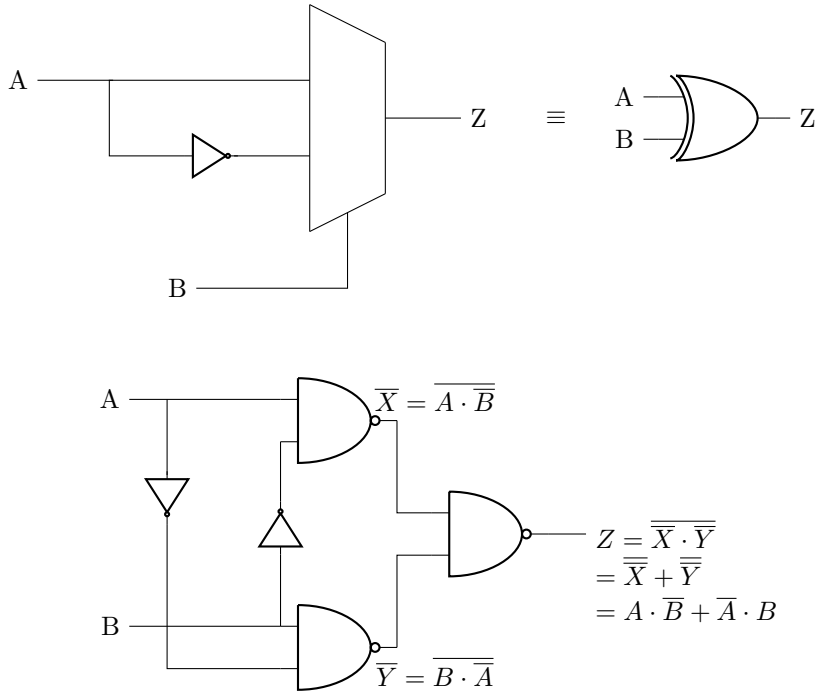

Figura 15: XOR a partir de un MUX

En este caso, para armar la compuerta XOR se necesitan 16 transistores.

Usando *Transmition Gates* (TG),

Observamos nuevamente que se reduce la cantidad de transistores necesarios para implementar la compuerta, es decir se requieren 10 transistores contra los 16 indicados anteriormente.

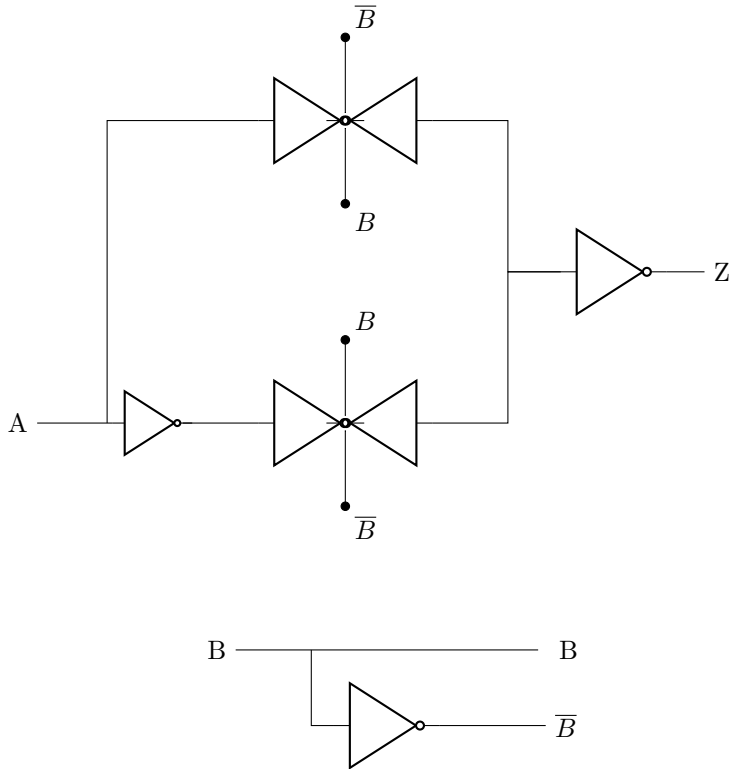

Figura 16: XOR basado en TG

10. Otra implementación de XOR/XNOR

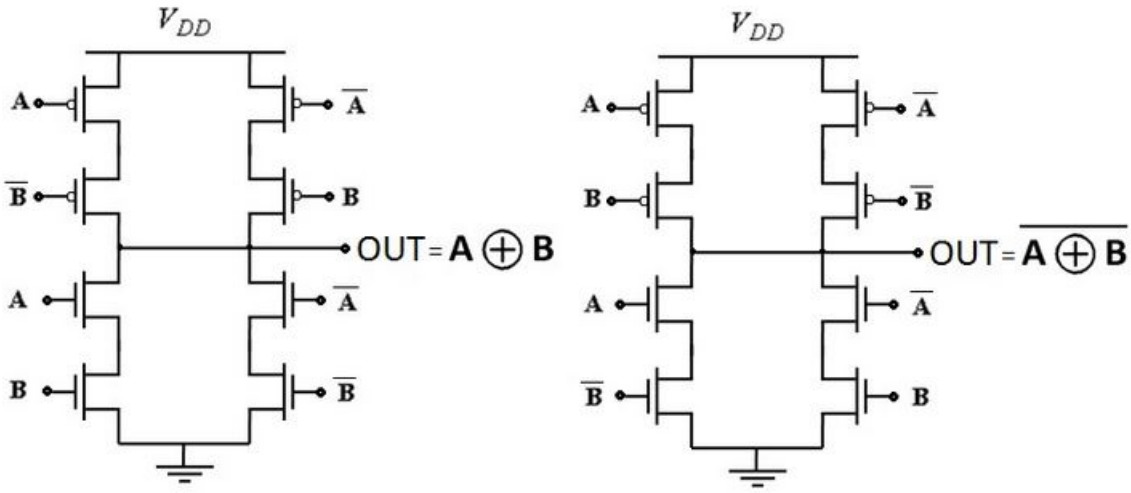

Figura 17: XOR/XNOR

11. Sumadores (Adders)

Otro componente importante en el diseño digital es el Sumador. Si pensamos su contrucción utilizando compuertas tradicionales, vemos que estos dispositivos consumen gran cantidad de transistores, por lo cual, para lograr un diseño eficiente tendremos una serie de problemáticas entre las que se encuentran:

- Area grande
- Consumo elevado de potencia.
- Retardo de propagacion del Carry Elevado.

Analicemos la función lógica que representa a un Adder. La tabla de verdad es:

A su vez, los mapas de karnaugh para las salidas S y C_{out} son:

Si desarrollamos, obtendremos:

$S = A\overline{B}\ \overline{C_{in}} + \overline{A}\ \overline{B}C_{in} + ABC_{in} + \overline{A}B\overline{C}_{in}$

$= A(\overline{B}\ \overline{C_{in}} + BC_{in}) + \overline{A}(BC_{in} + B\overline{C}_{in})$

$= \overline{A}(B \oplus C_{in}) + A(\overline{B}\ \overline{C}_{in} + BC_{in})$

A	B	C_{in}	C_{out}	S
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

Cuadro 1: Tabla de Verdad del Sumador

C_{out}		B C_i			
		00	01	11	10
A	0			1	
	1		1	1	1

Cuadro 2: Mapa karnaugh C_{out}

S		B C_i			
		00	01	11	10
A	0		1		1
	1	1		1	

Cuadro 3: Mapa karnaugh S

$= \overline{A}(B \oplus C_{in}) + A(\overline{B}\ \overline{C}_{in} + BC_{in} + B\overline{B} + C_{in}\overline{C}_{in})$

$= \overline{A}(B \oplus C_{in}) + A(B\ \overline{C}_i) + C_{in}\overline{B}$

$= \overline{A}(B \oplus C_{in}) + A(\overline{\overline{B}C_{in}})\overline{\overline{C_{in}}B}$

$= \overline{A}(B \oplus C_{in}) + A(\overline{\overline{B}C_{in} + \overline{C_{in}}B})$

$= \overline{A}(B \oplus C_{in}) + A\overline{(B \oplus C_{in})}$

$= \overline{A}(B \oplus C_{in}) + A\overline{(B \oplus C_{in})}$

Luego,

- $S = A \oplus B \oplus C_{in}$
- $C_{out} = AC_{in} + AB + BC_{in}$

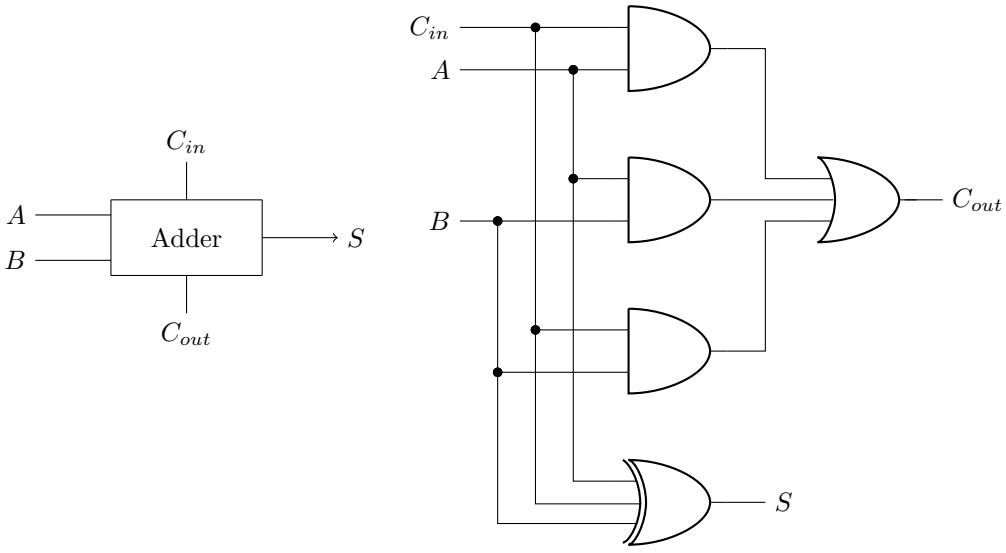

Figura 18: Adder

Veamos como podemos optimizar el resultado anterior.

Sean:

- $G = AB$ (Generate)
- $D = \overline{A + B}$ (Delete)
- $P = A \oplus B$ (Propagate)

Entonces:

$C_{out} = G + PC_{in} = AB + (A \oplus B)C_{in} + AB + (\overline{A}B + \overline{B}A)C_{in} = AB + \overline{A}BC_{in} + \overline{B}AC_{in}$

$S = P \oplus C_{in}$

En principio, la ecuacón de C_{out} parece distinta a la ecuacion original $C_0 = AC_i + AB + BC_i$.

Observemos la tabla de verdad:

A	B	C_{in}	$AB + \overline{A}BC_{in} + \overline{B}AC_{in}$	C_{out}
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Cuadro 4: Tabla de Verdad Carry Out para ambas ecuaciones

Identica tabla de verdad, por lo tanto son la misma funcion logica.

Verificamos que $C_{out} = G + PC_{in}$, entonces:

- Si $P = 1$, entonces $A \neq B = G = 0$, entonces $C_0 = C_i$
- Si $P = 0$, entonces $C_0 = G$

Luego:

$C_{out} = G\overline{P} + PC_{in} = AB\ \overline{A \oplus B} + A \oplus BC_{in}$

$= AB(\overline{\overline{A}B + \overline{B}A}) + (\overline{A}B + \overline{B}A)C_{in}$

$= AB\ \overline{\overline{A}B}\ \overline{\overline{B}A}$

$= AB(A + \overline{B})(B + \overline{A}) + \overline{A}BC_{in} + \overline{B}AC_{in}$

$= ABA + AB\overline{B} + ABB + AB\overline{A} + \overline{A}BC_{in} + \overline{B}AC_{in}$

$= AB + 0 + AB + 0 + \overline{A}BC_{in} + \overline{B}ACin$

dando como resultado:

$C_{out} = AB + \overline{A}BC_{in} + \overline{B}AC_{in}$

Obtenemos hasta ahora:

- $C_{out} = G\overline{P} + PC_{in}$
- $S = P \oplus C_{in}$
- $G = AB$
- $P = A \oplus B$

Vemos que si $P = 0 \Rightarrow C_{out} = G$, pero si $P = 0 = A \oplus B$, entonces $A = B \Rightarrow G = A = B$

Finalmente:

- $C_{out} = A\overline{P} + PC_{in}$
- $S = P \oplus C_{in}$
- $P = A \oplus B$

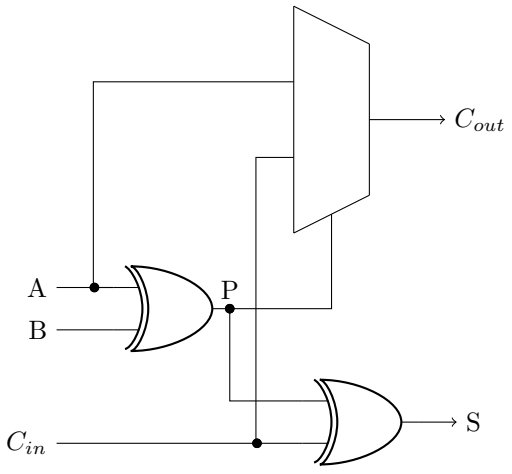

Figura 19: Sumador de dos bits optimizado

El Mux y las XOR se implementan con Transmission gates, como vimos en las figuras 14 y 16 dando como resultado el siguiente circuito óptimo para un sumador de dos bits.

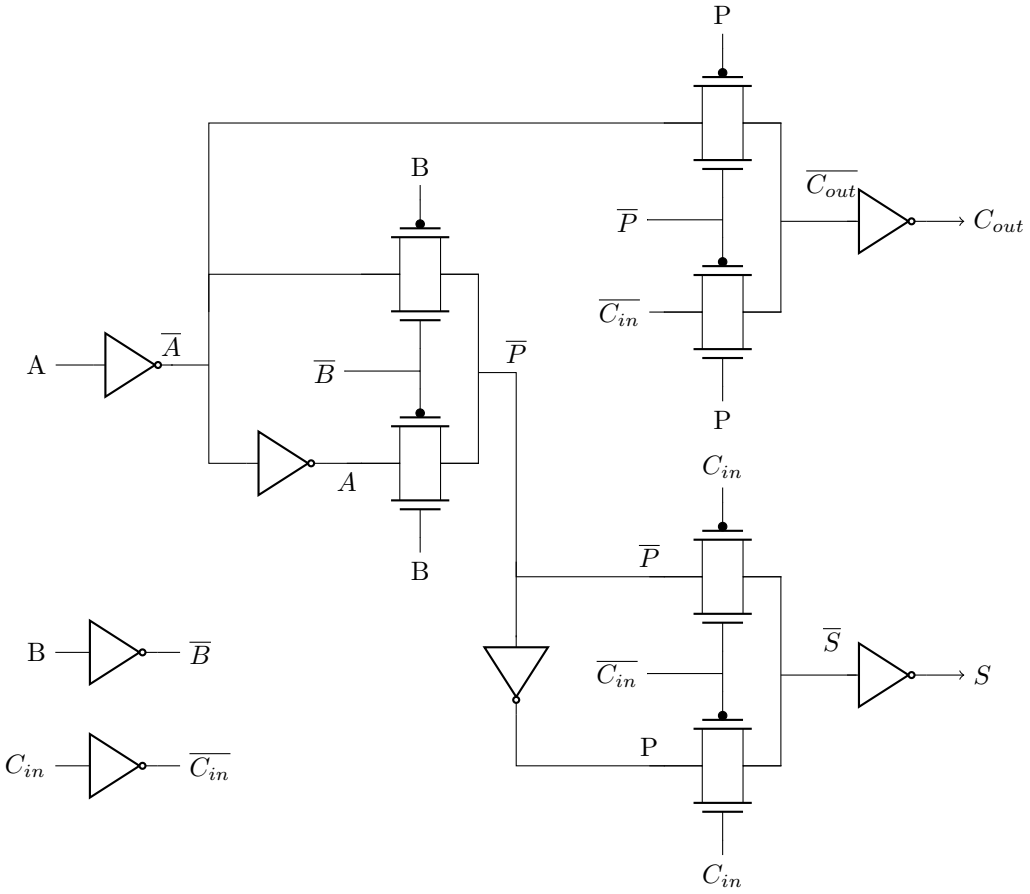

Figura 20: circuito Sumador de dos bits optimizado

12. Manchester Carry Chain

Para aumentar la perfomance del sumador mostrado en la sección anterior, se utiliza el circuito denominado Manchester Carry para el cálculo del carry.

La mejora que impone este componente está basada en la observación que el dalay del sumador está dominado por el camino de propagación del carry.

A partir de las siguientes definiciones armamos la tabla de verdad del circuito lógico.

- $G = AB$

- $D = \overline{A + B}$
- $P = A \oplus B$

AB	$G = AB$	A+B	$D = \overline{A + B}$	$P = A \oplus B$
00	0	0	1	0
01	0	1	0	1
10	0	1	0	1
11	1	1	0	0

Cuadro 5: Tabla de Verdad Manchester Carry

- $P = 1$
  - $C_{out} = C_{in}$, $D = 0$, $G = 0$, T_2 abierto, T_1 abierto
- $P = 0$
  - $G = 1$ *es decir* $A = B = 1 \Rightarrow C_{out} = 1,\ D = 0, \overline{C}_{out} = 0$
  - $G = 0$ *es decir* $A \neq B \Rightarrow C_{out} = 0,\ D = 1, \overline{C}_{out} = 1$

Finalmente, el circuito sumador con Manchester Carry es:

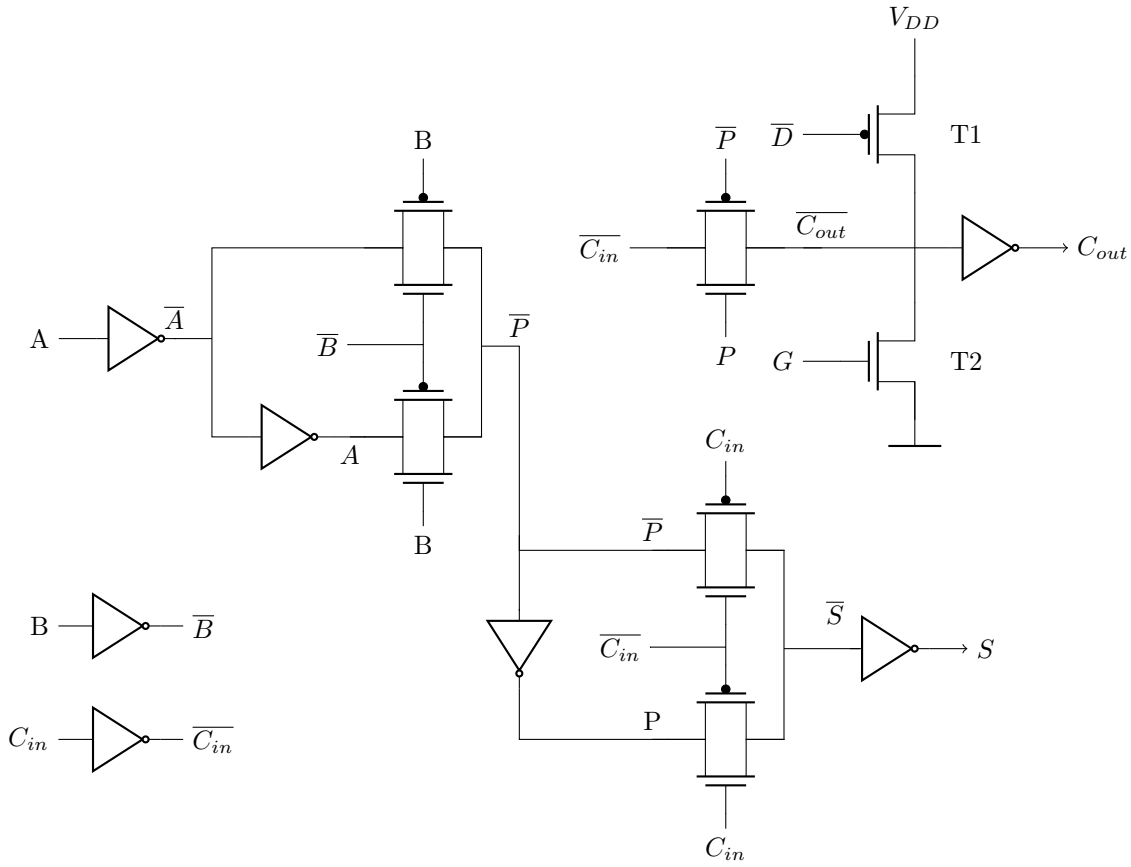

Figura 21: circuito Sumador de dos bits optimizado

13. Preguntas para pensar

1. Hay algún limitante en la cantidad de entradas que se puede poner a una compuerta CMOS?

2. Por qué el buffer CMOS se construye con dos inversores en cascada y no con simplemente con dos transisores (NMOS arriba, PMOS debajo) de forma tal que la función sea no inversora?

3. Qué sucede si al multiplexor basado en TG se le quitan los inversores?

4. Cuál es la resistencia equivalente de salida de una NOR y una NAND? Depende del valor lógico del resto de entradas al momento que una está conmutando?

5. Cómo varía el tiempo de propagación, rise time, fall time para una NAND/NOR cuando dos entradas conmutan a la vez de forma tal de que el valor lógico de la salida conmuta? Y si una de las entradas ya está activa y sólo conmuta la otra?