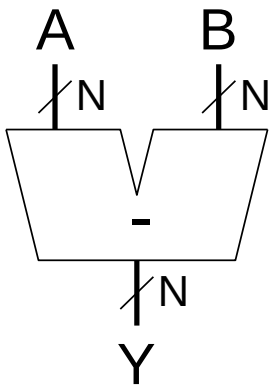
Lógica Combinacional en VHDL (III)

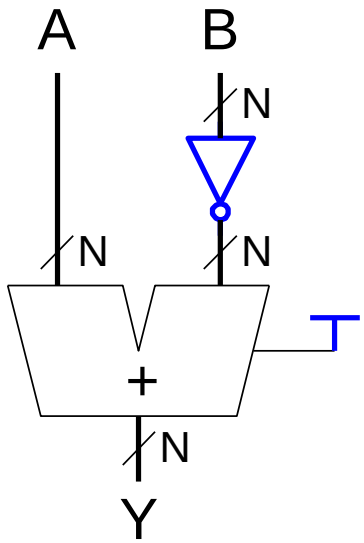
UNIVERSIDAD NEBRIJA

Restador

Symbol

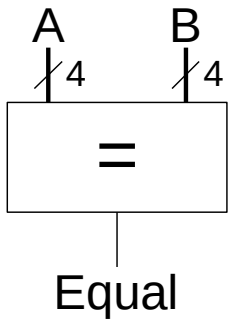


Implementation

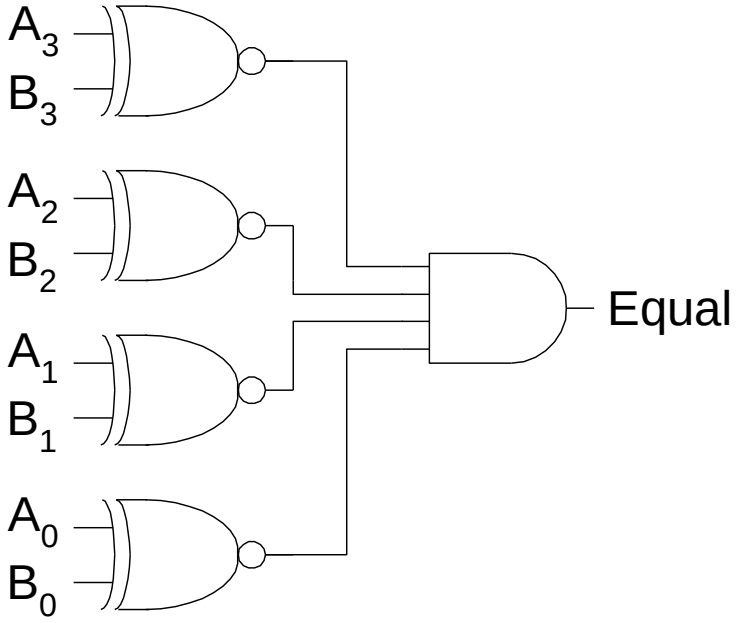


Comparador: Igualdad

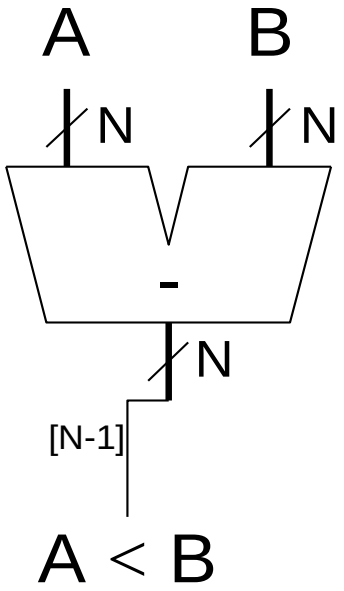
Symbol



Implementation



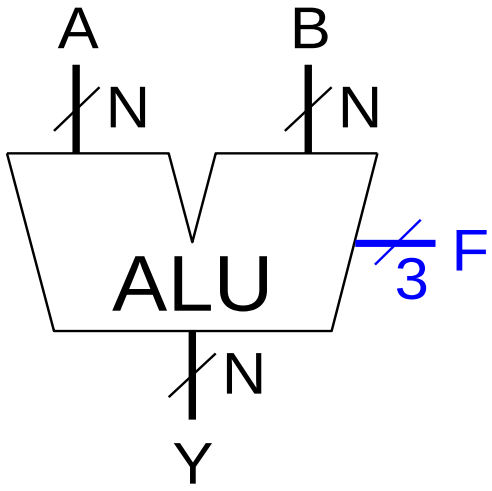
Comparator: Menor que



Implementación VHDL

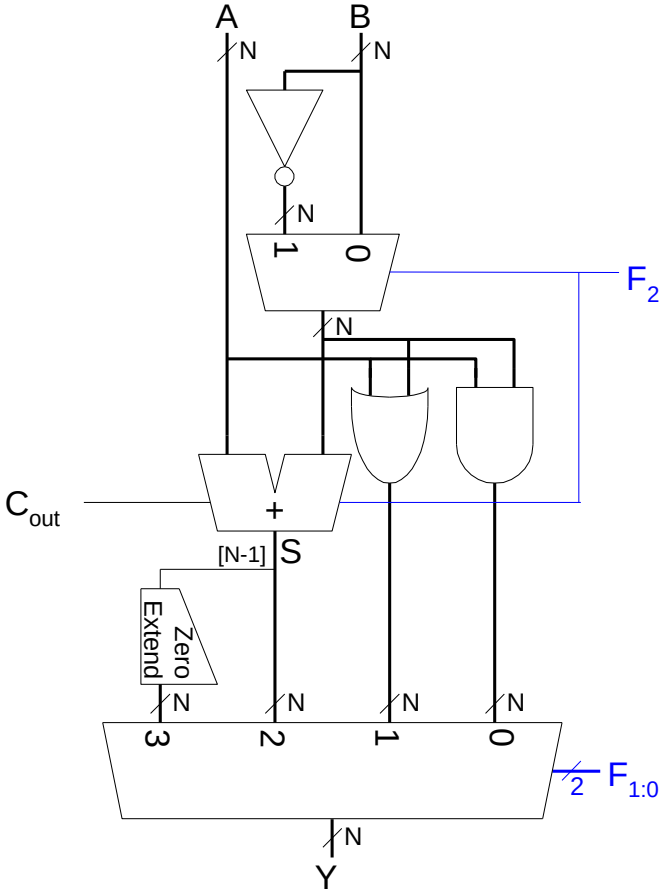
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity comparator is
  generic(N: integer := 8);
  port(a, b: in  STD_LOGIC_VECTOR(N-1 downto 0);
       eq, neq, lt, lte, gt, gte: out STD_LOGIC);
end comparator;

architecture synth of comparator is
begin
  eq  <= '1' when (a = b)  else '0';
  neq <= '1' when (a /= b) else '0';
  lt  <= '1' when (a < b)  else '0';
  lte <= '1' when (a <= b) else '0';
  gt  <= '1' when (a > b)  else '0';
  gte <= '1' when (a >= b) else '0';
end synth;
```

Arithmetic Logic Unit (ALU)



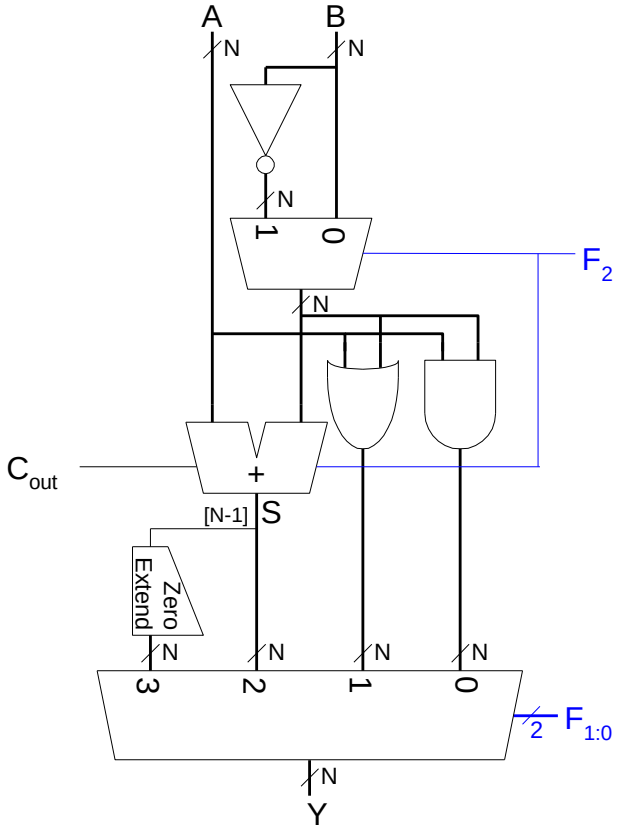
$F_{2:0}$	Function
000	A and B
001	A or B
010	A + B
011	no usado
100	A and B’
101	A or B’
110	A - B
111	SLT

Diseño de la ALU



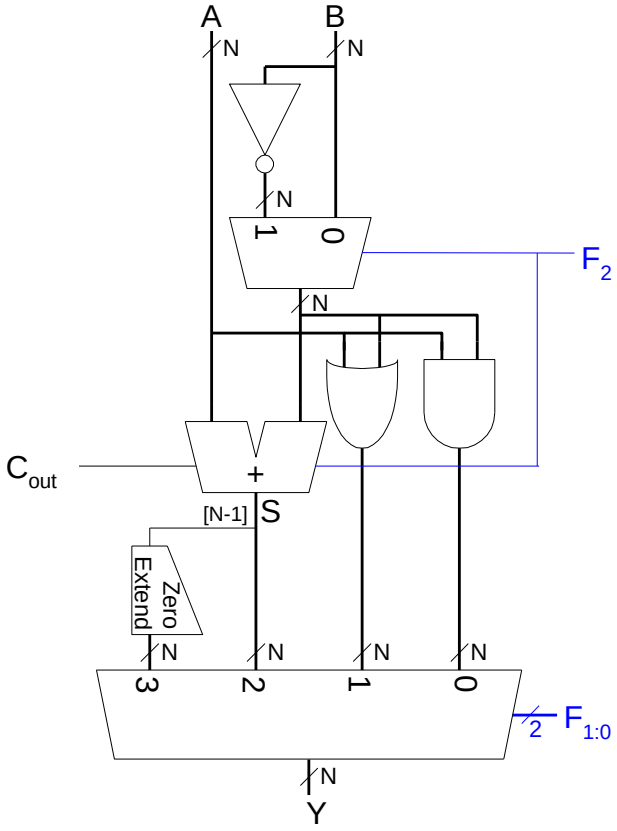
$F_{2:0}$	Function
000	A and B
001	A or B
010	A + B
011	no usado
100	A and B’
101	A or B’
110	A - B
111	SLT

Ejemplo Set Less Than (SLT)



- Configura una ALU de 32-bit para la operación SLT : $A = 25$ and $B = 32$

Set Less Than (SLT) Example



- Configura una ALU de 32-bit para la operación SLT : A = 25 and B = 32
  - ***A*** **<** ***B***, hace que Y sea la representación de 1 con 32-bits (0x00000001)
  - $\boldsymbol{F_{2:0}}$ **= 111**
    - $\boldsymbol{F_2}$ **= 1** (sumador funciona como restador), por tanto 25 - 32 = -7
    - -7 en complement a 2 tiene un 1 en su bit más significativo bit (S_{31} = 1)
    - $\boldsymbol{F_{1:0}}$ **= 11** el multiplexor selecciona $Y = S_{31}$ (zero extended) = 0x00000001.

Shifters: Desplazamiento

- **Logical shifter:** desplaza el valor a la izquierda o derecho y rellena los espacios vacíos con ceros.
  - Ex: 11001 >> 2 =
  - Ex: 11001 << 2 =

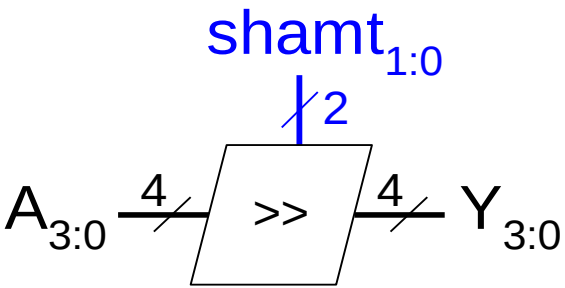
- **Arithmetic shifter:** igual que el lógico, pero en el desplazamiento a la derecha, en vez de rellenar con ceros, rellena con el valor del antiguo bit más (most significant bit - msb).
  - Ex: 11001 >>> 2 =
  - Ex: 11001 <<< 2 =

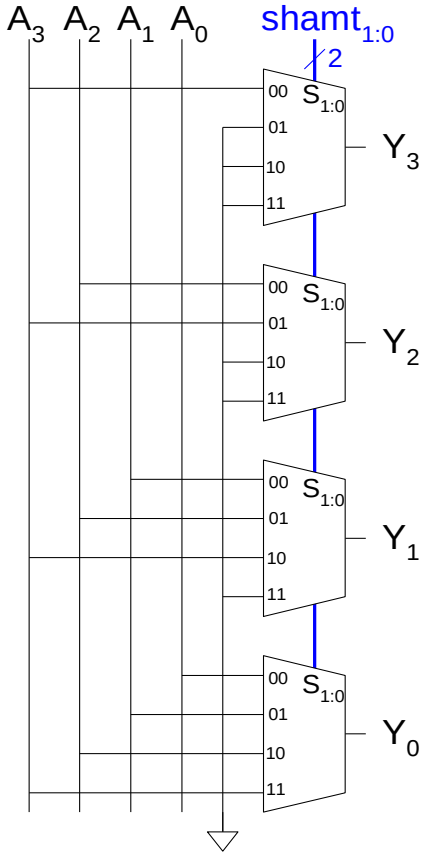
- **Rotator:** rota los bits en círculo, de forma que esos bits salen por un extremo y entran por el otro.
  - Ex: 11001 ROR 2 =
  - Ex: 11001 ROL 2 =

Shifters: Desplazamiento

- **Logical shifter:**
  - Ex: 11001 >> 2 = 00110
  - Ex: 11001 << 2 = 00100
- **Arithmetic shifter:**
  - Ex: 11001 >>> 2 = **11**110
  - Ex: 11001 <<< 2 = 00100
- **Rotator:**
  - Ex: 11001 ROR 2 = 01110
  - Ex: 11001 ROL 2 = 00111

Diseño del shifter





Shifters como Multiplicadores, Divisores

- $A << N = A \times 2^N$
  - **Ejemplo:** 00001 << 2 = 00100 $(1 \times 2^2 = 4)$
  - **Ejemplo:** 11101 << 2 = 10100 $(-3 \times 2^2 = -12)$
- $A >>> N = A \div 2^N$
  - **Ejemplo:** 01000 >>> 2 = 00010 $(8 \div 2^2 = 2)$
  - **Ejemplo:** 10000 >>> 2 = 11100 $(-16 \div 2^2 = -4)$

Multiplicadores

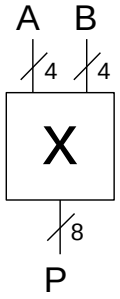
- **Productos parciales** se forman multiplicando un dígito del multiplicador por el multiplicando
- Las sumas parciales **desplazadas** se suman para formar el resultado final.

Decimal		**Binary**
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
		0101
		+ 0000
9660	result	0100011

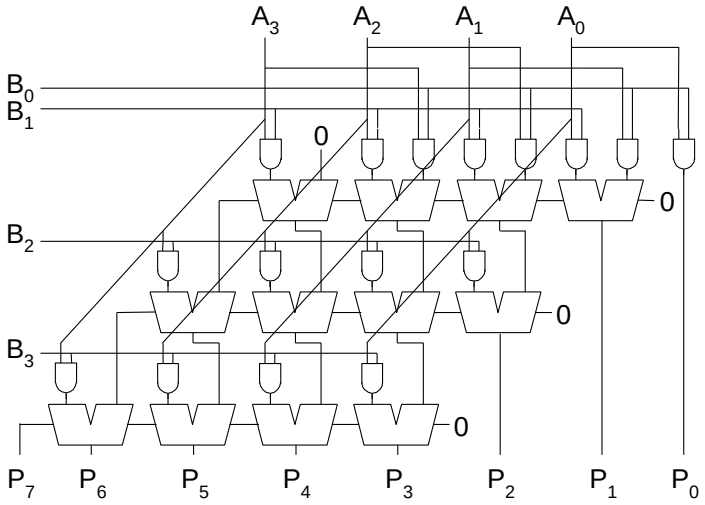
230 x 42 = 9660

5 x 7 = 35

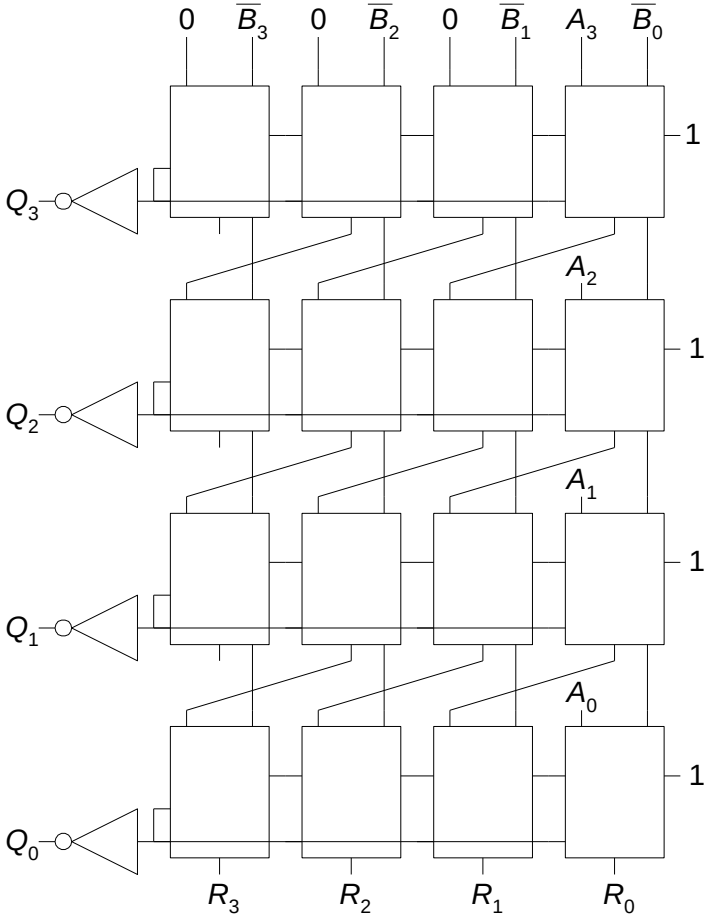
Multiplicador 4 x 4



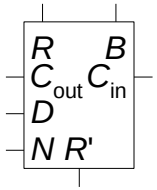
			A_3	A_2	A_1	A_0		
	x		B_3	B_2	B_1	B_0		
			A_3B_0	A_2B_0	A_1B_0	A_0B_0		
		A_3B_1	A_2B_1	A_1B_1	A_0B_1			
	A_3B_2	A_2B_2	A_1B_2	A_0B_2				
+	A_3B_3	A_2B_3	A_1B_3	A_0B_3				
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	

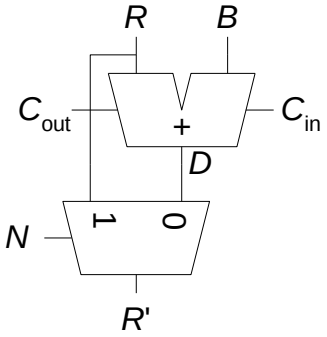


Divisor 4 x 4



Legend





A/B = Q + R/B

Algorithm:

R′ = 0

for *i* = N-1 to 0

R = {R′ << 1. A_i}

D = R - B

if D < 0, Q_i=0, R′ =R

else Q_i=1, R′ =D

R′ =R