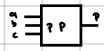
* Usado para construir circuitos que contienen operadores booleanos, input y output
* Un output siempre se basa en la salida
  * La entrada es una función de su input
* Varios outputs
  * Cada salida tiene su fórmula

* Parity generator
  * Crea la paridad de bits (par, impar) para añadir a una palabra.

Par:



a	b	c	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$P = a'b'c + a'bc' + ab'c' + abc$

$P = a'(b'c + bc') + a(b'c' + bc)$

$P = a'(b \oplus c) + a(b \oplus c)'$

$P = a \oplus (b \oplus c)$

* Parity checker
  * Verifica correctamente la paridad:

a	b	c	p	OK
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

$OK = a'b'c'p' + a'bcp + a'bc'p + a'bcp' + ab'c'p + ab'cp' + abc'p' + abcp$

$OK = a'b'(c'p' + cp) + a'b(c'p + cp') + ab'(c'p + cp') + ab(c'p' + cp)$

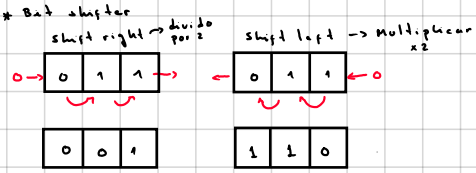
$OK = a'b'(c \oplus p)' + a'b(c \oplus p) + ab'(c \oplus p) + ab(c \oplus p)'$

$OK = (a'b' + ab)(c \oplus p)' + (a'b + ab')(c \oplus p)$

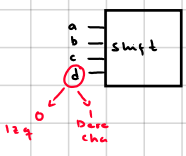
$OK = (a \oplus b)'(c \oplus p)' + (a \oplus b)(c \oplus p)$

$OK = ((a \oplus b) \oplus (c \oplus p))$

* Bit shifter



Palabra 3 bits:



d	a	b	c	x	y	z
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			

- Half adder
  - Adding two binary digits together with carry

Cin = carry entrada

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$c = ab$

$s = a\bar{b} + \bar{a}b$

$= a \oplus b$

Recordar:
0 + 0 = 0
0 + 1 = 0
1 + 0 = 0
1 + 1 = 10
1 + 0 + 0 = 10
1 + 0 + 1 = 10
1 + 1 + 0 = 10
1 + 1 + 1 = 11

Cin	a	b	c	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

$c = c_{in}'ab + c_{in}a\bar{b} + c_{in}\bar{a}b + c_{in}ab$

$= c_{in}'ab + c_{in}(a\bar{b} + \bar{a}b) + c_{in}ab$

$= (c_{in}' + c_{in})ab + c_{in}(a\bar{b} + \bar{a}b)$

$= 1ab + c_{in}(a \oplus b)$

$= ab + c_{in}(a \oplus b)$

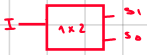
$s = c_{in}'\bar{a}b + c_{in}'a\bar{b} + c_{in}a\bar{b} + c_{in}ab$

$= c_{in}'(\bar{a}b + a\bar{b}) + c_{in}(a\bar{b} + ab)$

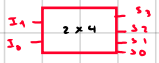
$= c_{in}(a \oplus b) + c_{in}(a \oplus b)'$

$= c_{in} \oplus (a \oplus b)$

Decoder:
- Uses the inputs and the respective values to select one specific output line
- Unique output line asserted, set to 1, other to 0
- Defined by number of outputs (2 to n Decoder)



I	s_1	s_0
0	0	1
1	1	0



n x 2^n

I_1	I_0	s_3	s_2	s_1	s_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

3 to 8:

I_2	I_1	I_0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$s_7 = I_2 I_1 I_0$

$s_6 = I_2 I_1 I_0'$

$s_5 = I_2 I_1' I_0$

$s_4 = I_2 I_1' I_0'$

$s_3 = I_2' I_1 I_0$

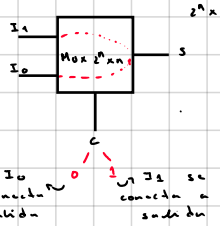
$s_2 = I_2' I_1 I_0'$

$s_1 = I_2' I_1' I_0$

$s_0 = I_2' I_1' I_0'$

- Multiplexer (Mux)
  - Selects binary info and forwards it to output line

$2^n x n$ -> Entradas x control

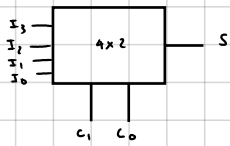


c	I_1	I_0	s
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$s = c'I_1'I_0 + c'I_1I_0 + cI_1I_0' + cI_1I_0$

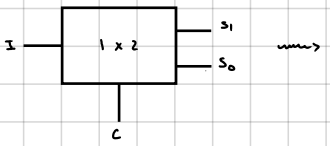
$= (c'I_1' + cI_1)I_0 + I_1(c'I_0 + cI_0')$

$= (c \oplus I_1)'I_0 + I(c \oplus I_0)$



~> Tabla muy grande

- Demultiplexor
  - "Inverso del Mux" -> n x 2^n



C	I	S_1	S_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	0

2 x 4

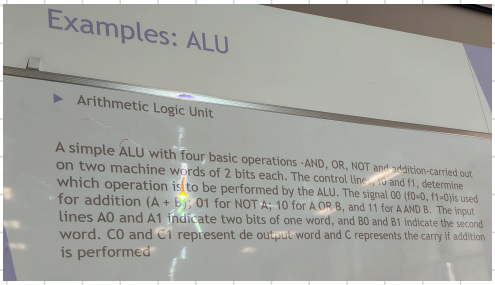
I_2	I_1	I_0	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	0	0
1	1	1	1	0	0	0

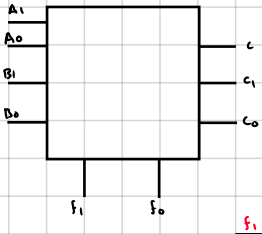
$S_0 = C_1' C_0' I$

$S_1 = C_1' C_0 I$

$S_2 = C_1 C_0' I$

$S_3 = C_1 C_0 I$





Sum	A+B	0	0
NOT	A	1	0
A OR	B	0	1
A AND	B	1	1

f_1	f_0	A_1	A_0	B_1	B_0	C	C_1	C_0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1
0	0	0	0	1	0	0	1	0
0	0	0	0	1	1	0	1	1
0	0	0	1	0	0	0	0	1
0	0	0	1	0	1	0	1	0
0	0	0	1	1	0	0	1	1
0	0	0	1	1	1	1	0	0
0	0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	1	1
0	0	1	0	1	0	1	0	0
0	0	1	0	1	1	1	0	1
0	0	1	1	0	0	0	1	1
0	0	1	1	0	1	1	0	0
0	0	1	1	1	0	1	0	1
0	0	1	1	1	1	1	1	0

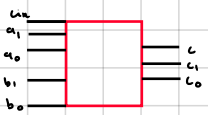
f_1	f_0	A_1	A_0	B_1	B_0	C	C_1	C_0
0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	1
0	1	0	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	0	0	1
0	1	0	1	1	0	0	1	1
0	1	0	1	1	1	0	1	1
0	1	1	0	0	0	0	1	0
0	1	1	0	0	1	0	1	1
0	1	1	0	1	0	0	1	0
0	1	1	0	1	1	0	1	1
0	1	1	1	0	0	0	1	1
0	1	1	1	0	1	0	1	1
0	1	1	1	1	0	0	1	1
0	1	1	1	1	1	0	1	1

Composición de circuitos:

- Componemos funciones en pocas palabras

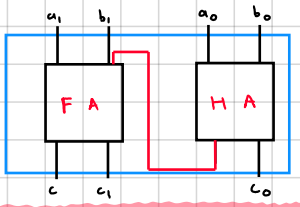
Ej: Circuito que sume dos palabras de 16 bits con carry

Sum 2:

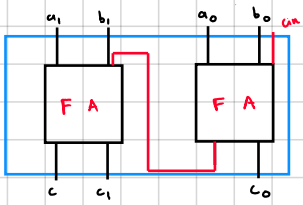


- Construir 1 más carry

$$\begin{array}{cc} a_1 & a_0 \\ + \; b_1 & b_0 \\ \hline & a_0 + b_0 \bmod 2 \end{array}$$



⟹ Problema: Si se hacen muchas composiciones, se vuelve muy lento.



⟹ Mucho mejor

* Más eficiente: 16 Full adder, 4-4-4, 2-2-2-2-2-2-2-2 se puede.

Operación: Unidad aritmética lógica por composición / Circuito por función