Els circuits lògics combinacionals

Montse Peiron Guàrdia
Fermín Sánchez Carracedo

1. Índex

Introducció

Un computador és una màquina construïda a partir de dispositius electrònics bàsics, interconnectats adequadament. Podem dir que aquests dispositius són les "peces" o "maons" amb els quals es construeix un computador.

L'objectiu de l'assignatura *Introducció als Computadors* és entendre com està format un computador a nivell electrònic. Per a poder entendre-ho, és necessari conèixer a fons els dispositius electrònics bàsics. Aquest és l'objectiu d'aquest mòdul i els següents.

Els dispositius electrònics més elementals són les *portes lògiques* i els *blocs lògics*, que formen els *circuits lògics*. Un circuit lògic es pot veure com un conjunt de dispositius que manipulen d'una manera determinada els senyals electrònics que els arriben (els *senyals d'entrada*), i generen com a resultat un altre conjunt de senyals (els *senyals de sortida*).

Hi ha dos grans tipus de circuits lògics:

- els **circuits combinacionals**, que es caracteritzen perquè el valor dels senyals de sortida en un moment determinat depèn del valor dels senyals d'entrada en aquest mateix moment

- els **circuits seqüencials**, en els quals el valor dels senyals de sortida en un moment determinat depèn dels valors que han arribat pels senyals d'entrada des de la posada en funcionament del circuit i fins en aquest mateix moment (tenen, per tant, capacitat de memòria).

En aquest mòdul s'estudien els circuits lògics combinacionals. Els circuits seqüencials s'estudiaran al mòdul "Els circuits lògics seqüencials".

Objectius

L'objectiu fonamental d'aquest mòdul és conèixer a fons els circuits lògics combinacionals, és a dir, saber com estan formats i ser capaços d'utilitzar-los amb agilitat, fins al punt d'estar-hi totalment familiaritzats.

Per a arribar a aquest punt caldrà haver satisfet els objectius següents:

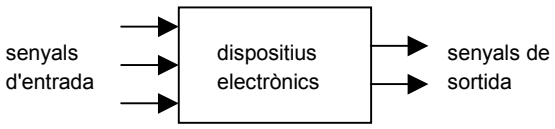
- entendre l'àlgebra de Boole i les diferents maneres d'expressar funcions lògiques
- conèixer les diferents portes lògiques, veure com es poden utilitzar per a sintetitzar funcions lògiques i ser capaços de fer-ho. Entendre per què és desitjable minimitzar el nombre de portes i de nivells de portes dels circuits, i saber fer-ho
- conèixer la funcionalitat dels diferents blocs combinacionals, i ser capaços d'utilitzar-los en el disseny de circuits

En definitiva, després de l'estudi d'aquest mòdul hem de ser capaços de construir fàcilment un circuit qualsevol usant els diferents dispositius que s'hauran conegut, així com d'entendre la funcionalitat de qualsevol circuit donat.

1. Fonaments de l'electrònica digital

1.1. Circuits, senyals, funcions lògiques

Entenem per **circuit** un sistema format per un cert nombre de **senyals d'entrada** (cada senyal correspon a un cable), un conjunt de **dispositius electrònics** que realitzen operacions sobre els senyals d'entrada (els manipulen electrònicament), i que generen un cert nombre de **senyals de sortida**. Els senyals de sortida, doncs, es poden veure com a funcions dels senyals d'entrada, i es pot dir que els dispositius electrònics *computen* aquestes funcions.

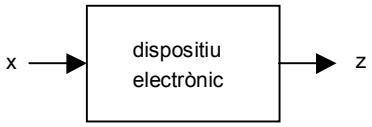


En els circuits que formen els sistemes computadors, els cables es poden trobar en dos valors de tensió (voltatge): tensió alta o tensió baixa. Aquest dos valors de tensió s'identifiquen normalment pels símbols "1" i "0" respectivament, de manera que es diu que un senyal val 0 (quan en el cable corresponent hi ha tensió baixa) o val 1 (quan en el cable hi ha tensió alta, també anomenada tensió d'alimentació). Als senyals que poden prendre els valors 0 o 1 se'ls anomena **senyals lògics** o **binaris**. Un **circuit lògic** és aquell en el qual els senyals d'entrada i de sortida són lògics. Les funcions que computa un circuit lògic són **funcions lògiques**.

En un principi, la tensió alta o tensió d'alimentació era de 5 volts. En els últims anys ha passat a ser de 3'3 o inclús 2'5 volts, mentre que la baixa és de 0 volts.

Com que els senyals només poden prendre dos valors, direm que l'un és el contrari de l'altre. Així dons, podem afirmar que quan un senyal no val 1, llavors segur que val 0, i viceversa.

Prenem un circuit que tingui només un senyal d'entrada (que anomenem *x*), un dispositiu electrònic i un senyal de sortida (que anomenem *z*).



Donat que tant *x* com *z* només poden valdre 0 o 1, només existeixen 4 dispositius electrònics diferents que puguin interconnectar *x* i *z*:

- un dispositiu que faci que la sortida *z* valgui sempre 0 (aquest dispositiu consistiria en connectar la sortida a una font de tensió de 0 volts)

- un dispositiu que faci que la sortida valgui sempre el mateix que l'entrada *x* (aquest dispositiu consistiria en connectar directament la sortida amb l'entrada)

- un dispositiu que faci que la sortida valgui sempre el contrari del que val l'entrada (aquest dispositiu consistiria en un inversor del nivell de tensió).

- un dispositiu que faci que la sortida valgui sempre 1 (aquest dispositiu consistiria en connectar la sortida directament a la tensió d'alimentació).

En altres paraules, podem dir que només existeixen 4 funcions lògiques que tinguin una sola variable d'entrada, tal com es mostra a la figura 1:

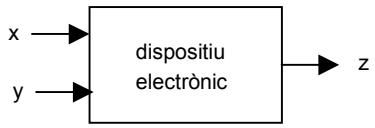
Figura 1

funció lògica	**descripció**	valor de la funció quan **x = 0**	valor de la funció quan **x = 1**
$z = f_0(x) = 0$	funció constant 0	0	0
$z = f_1(x) = x$	funció identitat	0	1
$z = f_2(x) = x'$	funció contrari	1	0
$z = f_3(x) = 1$	funció constant 1	1	1

Introduïm aquí ...

...la nomenclatura x' per referir-nos a la funció "contrari de x". Més endavant la definirem amb més propietat.

Prenem ara un circuit que tingui dos senyals d'entrada (que anomenem *x* i *y*), un dispositiu electrònic i un senyal de sortida (que anomenem *z*).



En aquest cas, existeixen 16 dispositius electrònics diferents que puguin interconnectar les entrades amb la sortida, que correponen a les funcions lògiques que es mostren a la figura 2.

Figura 2

funció lògica	valor de la funció quan x = 0, y = 0	x = 0, y = 1	x = 1, y = 0	x = 1, y = 1
$z = g_0(x,y)$	0	0	0	0
$z = g_1(x,y)$	0	0	0	1
$z = g_2(x,y)$	0	0	1	0
$z = g_3(x,y)$	0	0	1	1
$z = g_4(x,y)$	0	1	0	0
$z = g_5(x,y)$	0	1	0	1
$z = g_6(x,y)$	0	1	1	0
$z = g_7(x,y)$	0	1	1	1
$z = g_8(x,y)$	1	0	0	0
$z = g_9(x,y)$	1	0	0	1
$z = g_{10}(x,y)$	1	0	1	0
$z = g_{11}(x,y)$	1	0	1	1
$z = g_{12}(x,y)$	1	1	0	0
$z = g_{13}(x,y)$	1	1	0	1
$z = g_{14}(x,y)$	1	1	1	0
$z = g_{15}(x,y)$	1	1	1	1

Així doncs, si tenim en compte els circuits amb 1 o 2 entrades, podem arribar a dissenyar fins a 4 + 16 = 20 dispositius electrònics diferents. Ara bé, a la pràctica només se'n construeixen un subconjunt, concretament els que corresponen a les funcions f_2, g_1 i g_7 (i algunes altres que veurem més endavant), perquè a partir d'aquestes funcions es poden construir totes les altres, tal com veurem en l'apartat 1.2.

A continuació veurem que existeix una correspondència entre els elements d'un circuit lògic i la lògica intuïtiva (d'aquí deriva la denominació de circuits "lògics"). La lògica té 5 components bàsiques:

- els valors "fals" i "cert"
- les conjuncions "i" i "o"
- la partícula de negació "no"

Per exemple, siguin les dues frases següents:

frase_A: "en Joan estudia química"
frase_B: "la Carme estudia piano".

Cadascuna d'aquestes frases pot ser certa o falsa. A partir d'aquestes i de les conjuncions i la negació, construïm ara les tres frases següents:

frase_I: "en Joan estudia química **i** la Carme estudia piano"
frase_O: "en Joan estudia química **o** la Carme estudia piano"
frase_NO: "en Joan **no** estudia química"

Segons la lògica,

- frase_I és certa només si són certes frase_A i frase_B simultàniament.
- frase_O és certa si ho és frase_A o bé frase_B, o bé si ho són totes dues.
- frase_NO és certa només si frase_A és falsa.

Normalment, quan utilitzem la conjunció "o" en llenguatge natural ho fem de forma exclusiva. És a dir, si diem "X o Y" ens referim a què o bé és cert X o bé és cert Y, però no totes dues coses alhora. La "o" lògica, en canvi, inclou també la possibilitat que les dues afirmacions siguin certes.

La correspondència de la lògica amb els elements d'un circuit lògic és la següent:

lògica	circuits lògics
fals	0
cert	1
conjunció "i"	funció g_1
conjunció "o"	funció g_7
partícula "no"	funció f_2

En efecte, siguin dos senyals lògics *x* i *y*. Analitzant les taules de les figures 1 i 2, podem veure que

- $g_1(x,y)$ val 1 només si valen 1 totes dues variables *x* i *y* simultàniament
- $g_7(x,y)$ val 1 si *x* val 1 o bé *y* val 1 o bé valen 1 totes dues
- $f_2(x)$ val 1 només si *x* val 0

És a dir, es compleix el mateix que en l'exemple de les frases. Per això, la funció g_1 s'anomena AND (la conjunció "i" en anglès), la funció g_7 s'anomena OR (la conjunció "o" en anglès) i la funció f_2 s'anomena NOT ("no" en anglès).

Així doncs, els circuits lògics es construeixen a partir del mateix fonament que la lògica. En l'apartat següent s'estudia aquest fonament.

1.2. Àlgebra de Boole

Una **àlgebra de Boole** és una entitat matemàtica formada per un conjunt que conté 2 elements, unes operacions bàsiques sobre aquests elements, i una llista d'axiomes que defineixen les propietats que compleixen les operacions.

La lògica va ser formalitzada matemàticament per George Boole el 1854, quan va definir el que avui es coneix per *àlgebra de Boole*. De fet, aquí presentem un cas particular d'àlgebra de Boole, anomenat *àlgebra de Boole binària*; en general, el conjunt d'una àlgebra de Boole pot tenir més de 2 elements.

El 1938, Claude Shannon va definir el comportament dels circuits lògics sobre el fonament de l'àlgebra de Boole, creant l'*àlgebra de commutació*.

Els dos **elements** d'una àlgebra de Boole es poden anomenar "fals" i "cert" o, més usualment, "0" i "1". Així, una variable booleana o **variable lògica** pot prendre els valors 0 i 1.

Les **operacions booleanes bàsiques** són:

- la **negació** o **complementació** o **NOT,** que correspon a la partícula "no" i es representa amb una cometa simple ('). Així, l'expressió x' denota la negació de la variable *x*, i es llegeix "no x".
- el **producte lògic** o **AND**, que correspon a la conjunció "i" de la lògica i es representa pel símbol "·". Així, si *x* i *y* són variables lògiques, l'expressió $x \cdot y$ denota el seu producte lògic, i es llegeix "x i y".
- la **suma lògica** o **OR**, que correspon a la conjunció "o" i es representa pel símbol "+". Així, l'expressió $x+y$ denota la suma lògica de les variables *x* i *y*, i es llegeix "x o y".

L'operació de negació es pot representar també d'altres maneres. Per exemple, $!x$ o, més usualment, $\overline{x}$.

Aquestes operacions booleanes bàsiques es poden definir escrivint el resultat que donen per cada possible combinació de valors de les variables d'entrada, tal com es mostra a la figura 3. A les taules d'aquesta figura hi ha a l'esquerra de la ratlla vertical totes les combinacions possibles de les variables, i a la dreta el resultat de l'operació per a cada combinació. Podem comprovar que corresponen a les funcions f_2, g_1 i g_7 que hem vist a l'apartat anterior (figures 1 i 2).

Figura 3

x	x'
0	1
1	0

Operació NOT

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

Operació AND

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

Operació OR

És important...

...no confondre els operadors " · " i "+" amb les operacions "producte enter" i "suma entera" a les quals estem acostumats. El significat dels símbols vindrà donat pel context en el qual ens trobem. Així, si estem treballant amb enters, 1 + 1 = 2 ("u més u igual a dos"), mentre que si estem en un context booleà, 1 + 1 = 1 ("cert o cert igual a cert", tal com es pot veure a la taula de l'operació OR).

Els **axiomes** que descriuen el comportament de les operacions booleanes són els següents:

Siguin *x*, *y* i *z* variables lògiques. Llavors es compleix que

a) propietat commutativa:

$$x + y = y + x$$
$$x \cdot y = y \cdot x$$

b) propietat associativa:

$$x + (y + z) = (x + y) + z$$
$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

La formulació que aquí es presenta dels axiomes booleans va ser introduïda per Huntington el 1904, a partir de la descripció original de Boole.

c) propietat distributiva:

$$x \cdot (y + z) = x \cdot y + x \cdot z$$
$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

Observem que la segona igualtat de la propietat distributiva no es compleix en el context de l'aritmètica entera.

d) elements neutres:

$$x + 0 = x$$
$$x \cdot 1 = x$$

e) complementació: per qualsevol x, es compleix que

$$x + x' = 1$$
$$x \cdot x' = 0$$

A partir d'aquests axiomes, es poden demostrar una sèrie de lleis o teoremes molt útils per a treballar amb expressions algebraiques booleanes.

Teoremes de l'àlgebra de Boole

Si *x*, *y* i *z* són variables lògiques, es compleixen les lleis següents:

1) Principi de dualitat

Tota identitat deduïda a partir dels axiomes segueix sent certa si les operacions + i · i els elements 0 i 1 s'intercanvien en tota l'expressió.

Aquesta llei es pot comprovar per exemple examinant les parelles d'expressions que apareixen en la definició dels axiomes.

2) Llei d'idempotència

$$x + x = x$$
$$x \cdot x = x$$

3) Llei d'absorció

$$x + x \cdot y = x$$
$$x \cdot (x+y) = x$$

4) Llei de dominància

$$x + 1 = 1$$
$$x \cdot 0 = 0$$

5) Llei d'involució

$$(x')' = x$$

6) Lleis de De Morgan

$$(x + y)' = x' \cdot y'$$
$$(x \cdot y)' = x' + y'$$

Un bon exercici per a veure que es compleixen aquestes lleis és pensar en la seva correspondència amb la lògica.

En les expressions algebraiques, utilitzem els parèntesis de la mateixa manera en què hi estem acostumats en aritmètica entera; per exemple, x + y·z és el mateix que x + (y·z) i és diferent que (x + y)·z.
Per negar una expressió sencera la posarem entre parèntesis, i per tant x + y' és diferent de (x + y)'.

1.3. Representació de funcions lògiques

Les funcions lògiques es poden expressar de diverses maneres. Les que usarem nosaltres són les **expressions algebraiques** i les **taules de veritat**.

1.3.1. Expressions algebraiques

Les **expressions algebraiques** estan formades per variables lògiques, els elements 0 i 1, els operadors producte (·), suma (+) i negació ('), i els símbols "(", ")" i "=".

Mitjançant aquests elements es pot expressar qualsevol funció lògica. Per exemple, la funció g_4 de la figura 2 es pot expressar així:

$$g_4(x,y) = x' \cdot y$$

L'expressió *x'·y* val 1 (és certa) només si valen 1 (són certes) les subexpressions *x'* i *y* simultàniament. L'expressió *x'* val 1 només si *x* val 0. En la descripció de la funció g_4 (figura 2) es pot comprovar que només val 1 en el cas en què x = 0 i y = 1.

Una mateixa funció lògica es pot expressar mitjançant infinites expressions algebraiques equivalents.

Per saber si dues expressions algebraiques són equivalents (és a dir, expressen la mateixa funció) podem analitzar si una es pot derivar de l'altra usant els axiomes i lleis de l'àlgebra de Boole.

Per exemple, la primera llei de De Morgan ens diu que les funcions lògiques f i g són la mateixa:

$$f(x,y) = (x + y)'$$
$$g(x,y) = x' \cdot y'$$

A la inversa, a partir de l'expressió algebraica d'una funció podem trobar altres expressions equivalents aplicant-li els axiomes i lleis de l'àlgebra de Boole.

Per exemple, sigui la funció

$$f(x,y,z) = x \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y \cdot z$$

Aplicant l'axioma c) (propietat distributiva) obtenim la següent expressió equivalent:

$$f(x,y,z) = x \cdot y' \cdot z + (x' + x) \cdot y \cdot z$$

Per l'axioma d) (de complementació) i després el b) (d'elements neutres) obtenim

$$f(x,y,z) = x \cdot y' \cdot z + 1 \cdot y \cdot z = x \cdot y' \cdot z + y \cdot z$$

Per la propietat distributiva,

$$f(x,y,z) = (x \cdot y' + y) \cdot z$$

I així podríem seguir trobant infinites expressions equivalents per la mateixa funció.

Igual com fem habitualment quan treballem amb equacions, podem ometre el signe "·" en els productes lògics, per tal de simplificar l'escriptura de les expressions algebraiques: si escrivim seguits els noms de diverses variables, sobreentendrem que se'n fa el producte lògic. Així, les dues expressions següents són igualment vàlides:

$$x_1' \cdot x_0 + x_2 \cdot x_1' \cdot x_0'$$
$$x_1' x_0 + x_2 x_1' x_0'$$

En aritmètica entera...

...és impossible escriure la taula de veritat d'una funció, ja que les variables d'entrada poden prendre infinits valors possibles, i per tant la llista hauria de ser infinita. En àlgebra de Boole és possible gràcies a què les variables poden prendre només 2 valors.

1.3.2. Taules de veritat

Una **taula de veritat** expressa una funció lògica especificant el valor que té la funció per cada possible combinació de valors de les variables d'entrada.

Concretament, a l'esquerra de la taula hi ha una llista amb totes les combinacions de valors possibles de les variables d'entrada, i a la dreta el valor de la funció per cadascuna de les combinacions. Per exemple, les taules que havíem vist a la figura 3 eren, de fet, les taules de veritat de les funcions NOT, AND i OR.

Si una funció té n variables d'entrada, i donat que una variable pot prendre només dos valors, 0 o 1, les entrades poden prendre 2^n combinacions de valors diferents. Per tant, la seva taula de veritat tindrà a l'esquerra n columnes (una per cada variable) i 2^n files (una per cada combinació possible). A la dreta hi haurà una columna amb els valors de la funció.

Les files les escriurem sempre en *ordre lexicogràfic*: és a dir, si interpretem les diferents combinacions com a números naturals, escriurem primer la combinació corresponent al número 0 (formada per només 0s), després la corresponent al número 1 i successivament en ordre creixent fins a la corresponent al número 2^n-1 (formada per només 1s).

La figura 4 mostra l'estructura de les taules de veritat per a funcions d'1, 2 i 3 variables d'entrada.

Figura 4

Estructura de la taula de veritat d'una funció de

1 variable

a	f
0	
1	

2 variables

a	b	f
0	0	
0	1	
1	0	
1	1	

3 variables

a	b	c	f
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

No obstant...

...les combinacions es podrien escriure en qualsevol ordre. Per exemple, les dues taules següents expressen la mateixa funció:

x	y	f
0	0	0
0	1	1
1	0	1
1	1	0

x	y	f
0	1	1
1	1	0
1	0	1
0	0	0

Per exemple, la funció g_5 de la figura 2 té la taula de veritat següent:

x	y	g_5
0	0	0
0	1	1
1	0	0
1	1	1

L'ordre en què posem les columnes de les variables és significatiu. Per exemple, podíem haver escrit la taula de veritat de la funció g_5 de la manera següent:

y	x	g_5
0	0	0
0	1	0
1	0	1
1	1	1

Com a norma, posarem les variables a les columnes en ordre alfabètic d'esquerra a dreta. Si les variables són *a*, *b*, *c* i *d,* posarem *a* a la columna de més a l'esquerra i *d* a la de més a la dreta.

És força usual anomenar les variables d'una funció amb una mateixa lletra i diferents subíndexos; per exemple, x_2, x_1, x_0. En aquest cas, posarem a la columna de més a l'esquerra la variable de subíndex més alt, i a la de més a la dreta la de subíndex més baix.

Direm que la variable que posem a la columna de més a l'esquerra en una taula de veritat és la variable **de més pes**, i la que posem a la columna de més a la dreta la **de menys pes**.

Una vegada fixat l'ordre de les columnes i les files, la taula de veritat d'una funció és **única**.

Podem expressar el comportament de diverses funcions que tenen les mateixes variables d'entrada en una única taula de veritat. En aquest cas, a la dreta de la línia vertical hi haurà tantes columnes com funcions. No establirem cap convenció per a ordenar les columnes corresponents a les funcions (es poden posar en qualsevol ordre). A continuació es mostra un exemple.

x_2	x_1	x_0	f_0	f_1	f_2
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	0	1	1

1.3.3. Correspondència entre expressions algebraiques i taules de veritat

És important saber passar amb agilitat d'una expressió algebraica d'una funció a la seva taula de veritat, i viceversa.

Per a obtenir la taula de veritat a partir d'una expressió algebraica, podem fer-ho de dues maneres:

a) avaluar l'expressió per cada combinació de les variables, i escriure al lloc corresponent de la taula el resultat de l'avaluació.

Per exemple, sigui la funció següent:

$$f(a,b,c) = a + a'b'c + ac'$$

Avaluant l'expressió per al cas de la combinació [ab c] = [0 0 0], obtenim

$$0 + 1\cdot1\cdot0 + 0\cdot1 = 0 + 0 + 0 = 0$$

Per tant, a la primera fila de la taula hi anirà un 0. Per la combinació [a b c] = [0 0 1] obtenim

$$0 + 1\cdot1\cdot1 + 0\cdot1 = 0 + 1 + 0 = 1$$

Per tant, a la segona fila hi anirà un 1. I així succesivament fins a haver avaluat la funció per totes les combinacions.

b) analitzant l'expressió "a vista", deduir per quins valors de les variables la funció val 1 o 0, i anar omplint la taula.

Prenem la mateixa funció de l'exemple anterior. Veiem que sempre que a = 1, la funció val 1 (per la llei 2). Per tant, podem posar un 1 en totes les files en les quals a = 1 (les quatre últimes).

A continuació estudiem els casos en què a = 0 (els que ens falten per omplir). L'expressió de la funció ens queda

$$f(0,b,c) = 0 + 0'\cdot b'\cdot c + 0\cdot c' = b'\cdot c$$

Aquesta expressió val 1 només en el cas [b c] = [0 1]. Per tant, posarem un 1 a la fila corresponent a la combinació [0 0 1]. Per les combinacions que encara ens queden per omplir, la funció val 0.

Observem que el pas d'expressió a taula de veritat ens pot resultar molt útil a l'hora de determinar si dues expressions algebraiques són o no equivalents. Podem obtenir la taula de veritat a partir de cadascuna de les expressions, i si les taules resultants són iguals podem concloure que les dues expressions corresponen a una mateixa funció.

El pas de taula de veritat a expressió algebraica es pot fer "a vista", si tenim prou experiència. Però ens resultarà més còmode fer-ho coneixent les expressions en suma de mintermes, que veurem a l'apartat següent.

1.3.4. Expressions en suma de mintermes

Ja hem vist que existeixen infinites expressions algebraiques equivalents per a una funció qualsevol. Ara bé, tota funció lògica es pot expressar amb una expressió en suma de productes. És a dir, una expressió de la forma

$$A + B + C$$

on A, B i C són termes producte, és a dir, tenen la forma

$$X\cdot Y\cdot Z$$

on X, Y i Z són variables lògiques, bé negades bé sense negar. Per exemple, les dues expressions següents corresponen a la mateixa funció (podeu demostrar-ho), però només la segona té forma de suma de productes:

$$f(x,y,z) = (x+yz')'z$$
$$f(x,y,z) = x'y'z + x'yz$$

A partir de la taula de veritat d'una funció, és molt senzill obtenir-ne una expressió en suma de productes. Vegem les taules de veritat de les 4 funcions següents:

x_2	x_1	x_0	f_0	f_1	f_2	F
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	1	1
1	1	1	0	0	0	0

Examinant f_0, f_1 i f_2 veiem que tenen la particularitat de valdre 1 només per una de les combinacions de les variables d'entrada, de manera que podem expressar-les fàcilment amb un sol terme producte, en el qual **hi apareixen totes les variables**, negades o sense negar. Les variables que valen 0 en la combinació que fa que la funció valgui 1 apareixeran negades al terme producte; les que valen 1 apareixeran sense negar. Així, obtenim els termes producte següents per a les funcions anteriors:

$$f_0(x_2,x_1,x_0) = x_2'x_1'x_0$$
$$f_1(x_2,x_1,x_0) = x_2x_1'x_0'$$
$$f_2(x_2,x_1,x_0) = x_2x_1x_0'$$

S'anomena *terme mínim* o **minterme** el terme producte (únic) que expressa una funció que només val 1 per una sola combinació de les variables d'entrada.

No ens resulta tan senzill obtenir "a vista" una expressió per a la funció F, ja que val 1 per diverses combinacions de les variables. Ara bé, sí que podem veure fàcilment que F val 1 si o bé f_0 o bé f_1 o bé f_2 valen 1. És a dir,

$$F = f_0 + f_1 + f_2$$

Per tant, obtenim que

$$F(x_2,x_1,x_0) = x_2'x_1'x_0 + x_2x_1'x_0' + x_2x_1x_0'$$

que és una expressió algebraica d'F en forma de suma de productes.

Així doncs, a partir de la taula de veritat d'una funció podem obtenir-ne una expressió en suma de productes fent la suma lògica dels mintermes que la formen. Per això les expressions que obtenim d'aquesta manera s'anomenen **sumes de mintermes**.

Tota funció lògica es pot expressar també en forma de producte de sumes, però en aquest curs no en parlarem.

Les expressions en suma de mintermes o producte de sumes s'anomenen ***formes canòniques*** d'una funció. Els mintermes s'anomenen *termes canònics*.

Al capítol següent (apartat 2.3) veurem que a partir d'una expressió en suma de mintermes es pot obtenir una altra expressió en suma de productes més senzilla.

1.4. Altres funcions comunes

A l'apartat 1.1 hem vist que hi ha 4 funcions lògiques d'una variable i 16 de dues variables. Ens hem fixat especialment en les funcions f_2, g_1 i g_7, perquè corresponen a les operacions bàsiques de l'àlgebra de Boole, i les hem anomenat respectivament *negació* (NOT), *producte lògic* (AND) i *suma lògica* (OR). Després hem vist que qualsevol funció lògica es pot construir a partir d'aquestes tres.

D'entre les funcions de dues variables, n'hi ha algunes altres que reben també un nom propi, perquè s'usen de manera comuna. Són les següents:

Funció O-exclusiva o XOR

Aquesta funció val 1 quan alguna de les dues variables d'entrada val 1, però no quan valen 1 totes dues alhora; per això rep el nom de "O-exclusiva". És la funció g_6 de la figura 2.

Es representa pel símbol "⊕", i té aquesta taula de veritat:

a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

$a \oplus b = a'b + ab'$

La funció XOR de més de dues variables es calcula així:

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

Per tant, dóna 0 si un nombre parell de les variables valen 1, i dóna 1 si un nombre senar de les variables valen 1. Fixem-nos que en el cas de dues variables, aquest fet es tradueix en el següent:

a ⊕ b = 0 => *a* i *b* tenen el mateix valor
a ⊕ b = 1 => *a* i *b* tenen valors diferents

Per tant, la funció XOR ens permet saber si dues variables són iguals o no.

Altres propietats de la XOR:

$0 \oplus a = a$
$1 \oplus a = a'$

Així, si fem la XOR d'una variable amb un 0 la variable queda inalterada, mentre que si fem la XOR amb un 1 a la sortida hi apareix la variable negada

Funció NAND

És la negació de l'AND, és a dir:

$$a \text{ NAND } b = (a \cdot b)'$$

Per tant, val 1 sempre que no es compleixi que les dues variables d'entrada valen 1. És, doncs, la funció g_{14} de la figura 2. Aquesta és la seva taula de veritat:

a	b	(a·b)'
0	0	1
0	1	1
1	0	1
1	1	0

Funció NOR

És la negació de l'OR, és a dir:

$$a \text{ NOR } b = (a+b)'$$

Per tant, val 1 només quan cap de les dues variables d'entrada val 1. És la funció g_8 de la figura 2. Aquesta és la seva taula de veritat:

a	b	(a+b)'
0	0	1
0	1	0
1	0	0
1	1	0

1.5. Funcions incompletament especificades

Hi ha funcions per a les quals algunes combinacions de les variables no es donaran mai. Per exemple, suposem una funció $f(x_2,x_1,x_0)$ en la qual les variables corresponen a senyals connectats a 3 aparells de mesura del so en una sala, de manera que

$x_2 = 1$ si el so supera els 100 dB, $x_2 = 0$ altrament
$x_1 = 1$ si el so supera els 200 dB, $x_1 = 0$ altrament
$x_0 = 1$ si el so supera els 300 dB, $x_0 = 0$ altrament

La combinació $[x_2\, x_1\, x_0] = [0\ 1\ 0]$ no es donarà mai, ja que és impossible que el so estigui per sota dels 100 dB i per sobre dels 200 dB simultàniament. Tampoc no es donaran mai les combinacions [0 0 1], [0 1 1] i [1 0 1].

Les combinacions que no es donaran mai es diuen **combinacions no importa** o **combinacions *don't care***.

A la taula de veritat d'una funció, escriurem "x" a les files corresponents a les combinacions no importa. Per exemple, suposem que la funció anterior ha de valdre 1 si el so està entre (100 dB, 200 dB] o si el so supera els 300 dB. La seva taula de veritat és la següent:

x_2	x_1	x_0	f
0	0	0	0
0	0	1	x
0	1	0	x
0	1	1	x
1	0	0	1
1	0	1	x
1	1	0	0
1	1	1	1

Al capítol següent veurem com obtenir expressions algebraiques de funcions incompletament especificades.

2. Implementació de circuits lògics combinacionals

2.1. Portes lògiques. Síntesi i anàlisi

A les figures 1 i 2 havíem vist que es poden construir fins a 20 dispositius electrònics diferents per a funcions d'1 i 2 variables d'entrada. Ara bé, en la pràctica només es construeixen els que computen les funcions NOT, AND, OR, XOR, NAND i NOR.

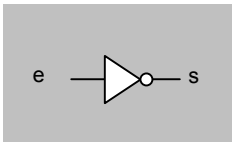
Els dispositius electrònics que computen funcions lògiques s'anomenen **portes lògiques.** Són dispositius que estan connectats a un cert nombre de senyals d'entrada i un senyal de sortida.

Les portes lògiques estan formades internament per diferents combinacions de ***transistors***, que són els dispositius electrònics més elementals.

A continuació presentem el símbol gràfic amb el qual es representa cada porta lògica. En totes les figures que apareixen a continuació, els senyals d'entrada queden a l'esquerra de les portes i el senyal de sortida queda a la dreta.

Porta NOT o inversor

Aquesta porta es representa gràficament amb aquest símbol:

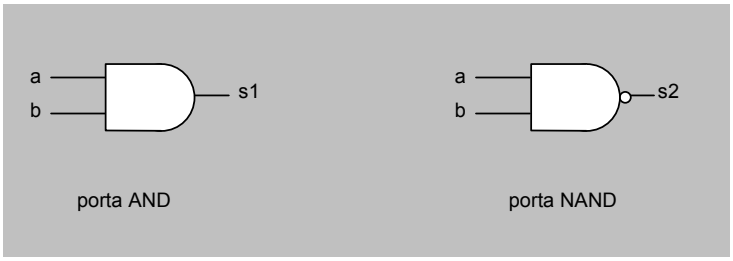


El funcionament és el següent: si a l'entrada *e* hi ha un 0 (tensió baixa), llavors a la sortida *s* hi haurà un 1 (tensió alta). I a la inversa, si hi ha un 1 al punt *e*, llavors hi haurà un 0 al punt *s*.

Per tant, la porta NOT implementa físicament la funció de negació: s = e'.

Portes AND i NAND

Es representen gràficament amb aquests símbols:

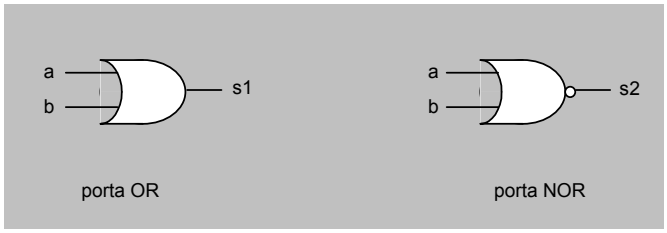


La porta AND implementa la funció lògica AND, és a dir: la sortida *s1* val 1 només si les dues entrades *a* i *b* valen 1. Per tant, s1 = a·b.

La porta NAND implementa la funció lògica NAND. Al punt *s2* hi ha un 1 sempre que a alguna de les dues entrades *a* o *b* hi hagi un 0. És a dir, s2 = (a·b)'.

Portes OR i NOR

Es representen gràficament amb aquests símbols:



La porta OR implementa la funció lògica OR, és a dir: a la sortida *s1* hi ha un 1 si qualsevol de les dues entrades està a 1. Per tant, s1 = a + b.

La porta NOR implementa la funció lògica NOR. Al punt *s2* hi trobarem un 1 només quan a totes dues entrades hi hagi un 0, és a dir: s2 = (a + b)'.

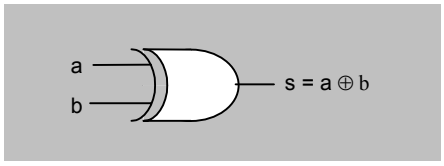
En un circuit...

... el cercle simbolitza una negació, en senyals de sortida (com en les portes NOT, NAND i NOR).

També s'usa un cercle per negar senyals d'entrada, però en aquest curs no farem servir aquesta possibilitat.

Porta XOR

Implementa la funció lògica XOR, és a dir: la sortida *s* val 1 si alguna de les dues entrades val 1, però no si valen 1 totes dues alhora. Es representa gràficament amb aquest símbol:



Totes les portes (llevat de la NOT) poden tenir més de dos senyals d'entrada. El seu funcionament és el següent:

- En una porta AND d'n entrades, la sortida val 1 només quan totes n entrades valen 1.
- En una porta OR d'n entrades, la sortida val 1 quan una o més d'una de les n entrades valen 1.
- En una porta XOR d'n entrades, la sortida val 1 quan hi ha un nombre senar d'entrades a 1. El zero es considera un nombre parell, i per tant si totes les entrades valen 0 la sortida també val 0.

Síntesi i anàlisi

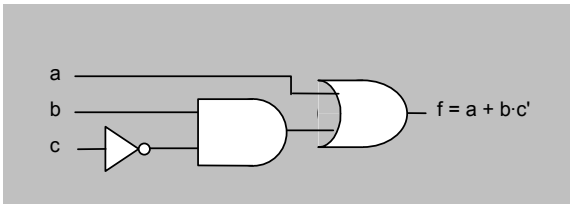
Qualsevol funció lògica es pot implementar usant aquestes portes, és a dir, es pot construir un circuit que es comporti com la funció.

El procés d'obtenir el circuit que implementa una funció a partir d'una expressió algebraica s'anomena **síntesi**.

El procés d'obtenir una expressió d'una funció a partir del circuit que la implementa s'anomena **anàlisi**.

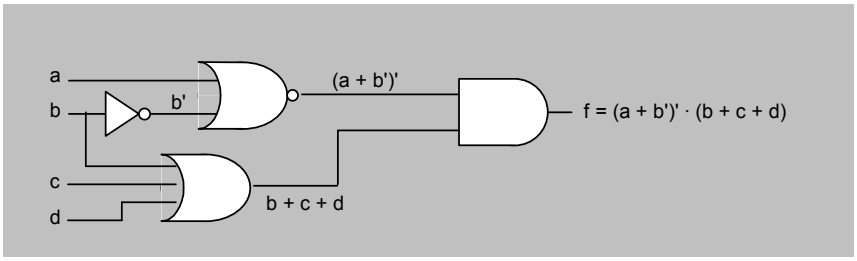
Per a sintetitzar (o implementar) una funció a partir de la seva expressió algebraica és suficient amb substituir cada operador de la funció per la porta lògica adequada. Per exemple, un terme producte de tres variables s'implementarà amb una porta AND de tres entrades. Les portes han d'estar interconnectades entre sí i amb les entrades segons ho indiqui l'expressió.

Per exemple, el circuit que implementa la funció f(a,b,c) = a + bc' és el següent:

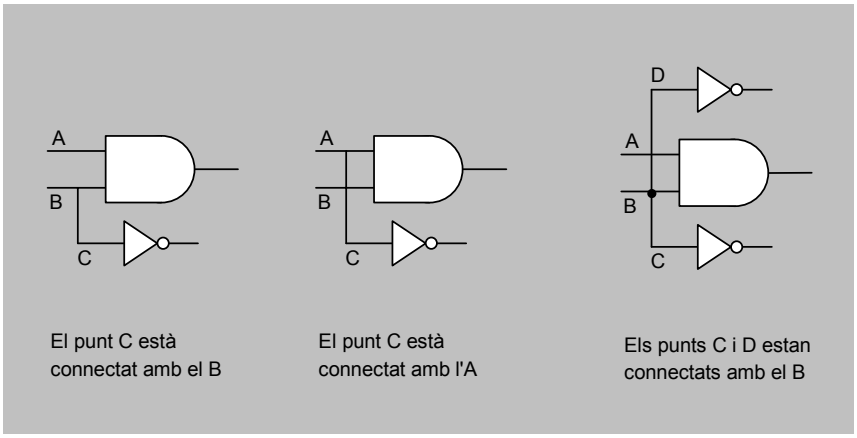


Per a analitzar un circuit, cal anar escrivint les expressions que corresponen a la sortida de cada porta, començant des de les entrades del circuit, fins a obtenir l'expressió corresponent a la línia de sortida del circuit.

Per exemple, el circuit següent implementa la funció f(a,b,c,d) = (a+b')'(b+c+d).



En un circuit, si una línia comença sobre una altra línia perpendicular, s'entén que les línies estan connectades (i per tant que tenen el mateix valor lògic). Si dues línies perpendiculars es creuen, s'entén que no estan connectades. Si damunt d'una línia s'hi posa un punt, s'entén que totes les linies que el toquen estan connectades. A continuació es mostren alguns exemples.



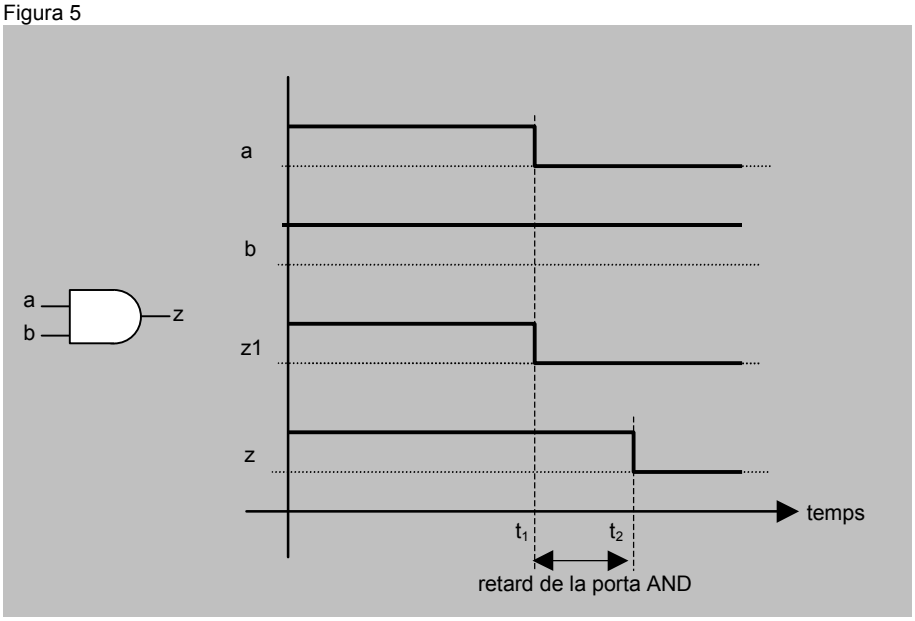
2.2. Disseny de circuits a dos nivells

2.2.1. Retards. Cronogrames. Nivells de portes

Les portes lògiques no responen instantàniament a les variacions en els senyals d'entrada, sinó que tenen un cert **retard**. La millor manera d'entendre aquest concepte és mirant la figura 5, en la qual hi ha un circuit senzill (només una porta AND) i un *cronograma* del seu funcionament.

Un **cronograma** és una representació gràfica de l'evolució dels senyals d'un circuit al llarg del temps.

Figura 5



En un cronograma...

...les línies de punts hortizontals representen el valor lògic 0 per a cada senyal. Les línies contínues gruixudes representen el valor en què es troba cada senyal en cada moment (0 si la línia contínua és sobre la línia de punts, 1 si està més amunt).

En vertical es poden assenyalar amb línies discontínues instants determinats de temps (com ara t_1 i t_2 en el cas de la Figura 5).

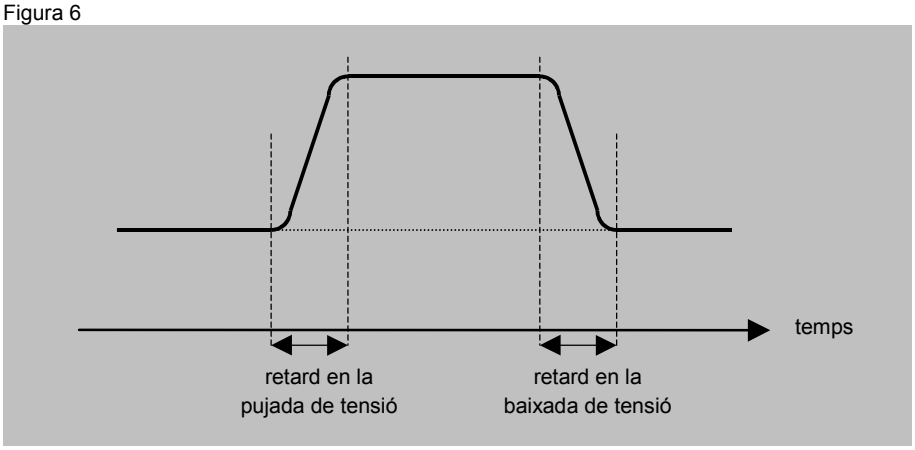
Suposem que a les entrades *a* i *b* hi tenim connectats dos interruptors que ens permeten en cada moment posar aquests senyals a 0 o a 1. En l'exemple de la figura

5, hem suposat que inicialment tots dos senyals estan a 1, i que en l'instant t_1 posem el senyal *a* a 0. En aquest moment, el senyal *z* s'hauria de posar a 0, perquè 0·1 = 0. Aquesta hipotètica situació és la que es mostra en el cronograma amb el senyal z1. No obstant, en la realitat *z* no es posa a 0 fins a l'instant t_2, degut a què els dispositius electrònics interns de la porta AND triguen un cert temps en reaccionar. Direm que la porta AND té un retard de $t_2 - t_1$.

Cada porta té un retard diferent, que depèn de la tecnologia que s'hagi usat per a construir-la. Els retards són molt petits, de l'ordre de nanosegons, però cal tenir-los en compte a l'hora de construir físicament un circuit.

De fet, la transició entre diferents nivells de tensió d'un senyal tampoc no és instantània, sinó que es produeix tal com es mostra a la figura 6.

Figura 6



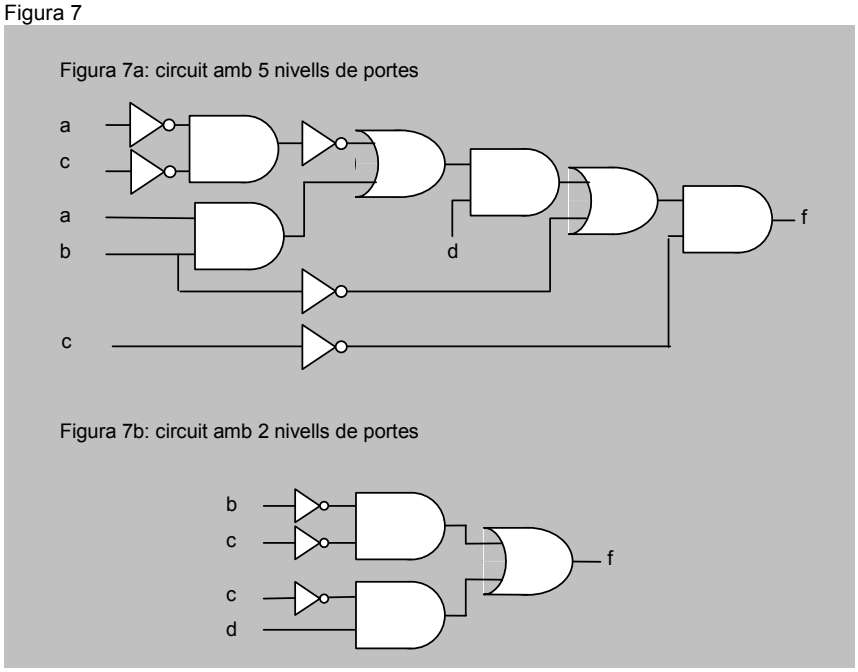
Un dels objectius dels enginyers electrònics que construeixen circuits és que el temps de resposta del circuit sigui el més petit possible. Donat que cada porta té un cert retard, un circuit serà en general més ràpid com menys *nivells de portes* hi hagi entre les entrades i les sortides.

El temps de resposta d'una porta depèn, entre altres coses, del número d'entrades de la porta.

El nombre de **nivells de portes** d'un circuit és el màxim nombre de portes que un senyal ha de travessar consecutivament per generar el senyal de sortida.
En comptabilitzar el nombre de nivells de portes d'un circuit **no es tenen en compte les portes NOT** (per raons que no veurem en aquest curs).

Per exemple, la figura 7 mostra el nombre de nivells de portes de dos circuits diferents (podeu comprovar que la funció que implementen és la mateixa). Un bon enginyer escolliria el circuit de la figura 7b per a implementar aquesta funció, ja que és més ràpid.

Figura 7

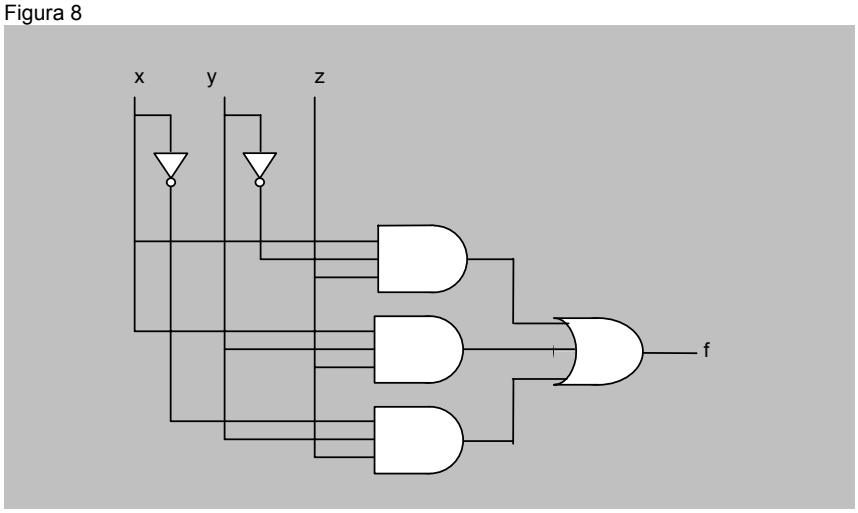


2.2.2. Síntesi a dos nivells

No existeix una fórmula universal per a trobar l'expressió d'una funció que donarà lloc al circuit més ràpid possible. No obstant, coneixem una manera de garantir que un circuit no tindrà més de 2 nivells de portes: partir de l'expressió de la funció en suma de mintermes. En efecte, el circuit corresponent a una suma de mintermes té, a més de les portes NOT, un primer nivell de portes AND (que computen els diferents mintermes) i un segon nivell en el qual hi haurà una única porta OR, amb tantes entrades com mintermes tingui l'expressió. Per exemple, la figura 8 mostra el circuit que s'obté en sintetitzar aquesta funció:

$$f = xy'z + xyz + x'yz$$

Figura 8



Els circuits que s'obtenen a partir de les sumes de mintermes s'anomenen **circuits a 2 nivells**. Anomenarem **síntesi a 2 nivells** el procés d'obtenir-los.

És possible que una funció es pugui implementar amb un circuit que tingui menys de 2 nivells. No obstant, no existeix una manera de trobar-lo sistemàticament. Així doncs, les expressions de les funcions en suma de mintermes ens permeten construir els circuits més ràpids possibles que podem obtenir de manera sistemàtica.

D'altra banda, a partir de l'expressió d'una funció en suma de mintermes en pot resultar un circuit amb menys de 2 nivells: si hi ha un sol minterme no hi haurà el nivell OR.

Tenir en compte els retards de les diferents portes i de les transicions entre nivells de tensió és fonamental a l'hora de construir circuits. No obstant, en aquest curs considerarem que els circuits són ideals, de manera que no es tindran mai en compte els retards, és a dir, s'assumirà sempre que són 0.

2.3. Minimització de funcions

Ja hem vist que un dels objectius a aconseguir en construir un circuit és que sigui el més ràpid possible. La síntesi a 2 nivells ens permet aconseguir de forma sistemàtica un grau de rapidesa acceptat.

També és desitjable que un circuit sigui petit i barat. Aquestes dues característiques estan relacionades amb el nombre de portes del circuit: com menys n'hi hagi, més petit i barat serà.

2.3.1. Simplificació d'expressions

Al capítol anterior s'ha vist com obtenir l'expressió d'una funció en suma de mintermes. A vegades, aquestes expressions es poden simplificar, la qual cosa ens permetrà implementar la funció amb un circuit més petit.

En concret, les simplificacions que ens seran útils corresponen als casos en què **dos o més mintermes es poden reduir a un sol terme producte,** en el qual hi apareixeran menys variables.

Per exemple, sigui la funció següent:

$$f(x,y,z) = x'yz' + x'yz + ...$$

Aquests dos mintermes ens diuen que la funció val 1 si x = 0 i y = 1 i z = 0 o bé si x = 0 i y = 1 i z = 1. Però aquesta informació es pot resumir dient que la funció val 1 sempre que x = 0 i y = 1, *independentment de quant valgui z*. Això es pot expressar algebraicament treient factor comú en els dos mintermes:

$$f(x,y,z) = x'yz' + x'yz + ...$$
$$= x'y(z'+z) + ... = x'y \cdot 1 + ... = x'y + ...$$

Prenem ara una funció que val 1 per les següents combinacions [x y z] (entre altres):

[1 0 0]
[1 0 1]
[1 1 0]
[1 1 1]

Aquesta funció val 1 sempre que x = 1, independentment de quant valguin *y* i *z*. Ho podem expressar algebraicament així:

$$f(x,y,z) = xy'z' + xy'z + xyz' + xyz + ... =$$
$$xy'(z'+z) + xy(z'+z) + ... =$$
$$xy' + xy + ... = x(y'+y) + ... = x + ...$$

Així doncs, veiem que podem obtenir un sol terme producte en els casos següents:

- si en dos mintermes totes les variables es mantenen constants llevat d'una; llavors podem treure factor comú eliminant la variable que canvia. Al terme producte resultant hi apareixeran n-1 variables, essent n el nombre de variables de la funció
- si en quatre mintermes totes les variables es mantenen constants llevat de dues; llavors podem treure factor comú dues vegades eliminant les dues variables que canvien. Al terme producte resultant hi apareixeran n-2 variables
- en general, si en 2^m mintermes totes les variables es mantenen constants llevat d'm; llavors podem treure factor comú m vegades eliminant les m variables que canvien. Al terme producte resultant hi apareixeran n-m variables.

També podem detectar altres casos en què es pot treure factor comú en una expressió algebraica. Per exemple, $f(x,y,z) = xy' + xz = x\cdot(y'+z)$. Però aquests casos no ens interessen, perquè l'expressió resultant no és una suma de productes.

El fet d'obtenir l'expressió en suma de productes més simplificada possible d'una funció, treient factor comú en els casos que s'acaben de descriure, s'anomena **minimitzar la funció**.

A partir de l'expressió minimitzada d'una funció podrem construir circuits més petits i barats que els que obteníem de l'expressió en suma de mintermes: potser caldran menys portes NOT, algunes de les portes AND seran de menys entrades, tindran menys portes AND i la porta OR serà de menys entrades.

La figura 9 mostra els circuits corresponents a l'expressió en suma de mintermes (figura 9a) i a l'expressió minimitzada (figura 9b) d'aquesta funció:

$$f(x_2,x_1,x_0) = x_2'x_1x_0 + x_2x_1'x_0' + x_2x_1x_0' =$$
$$= x_2'x_1x_0 + x_2x_0'(x_1'+x_1) = x_2'x_1x_0 + x_2x_0'$$

Es pot veure que el circuit de la figura 9a té 3 portes NOT, 3 portes AND de tres entrades i una porta OR de tres entrades. El circuit simplificat, en canvi, té només dues portes NOT, dues portes AND (una de dues entrades i l'altra de tres) i una porta OR de dues entrades.

Figura 9

Figura 9a

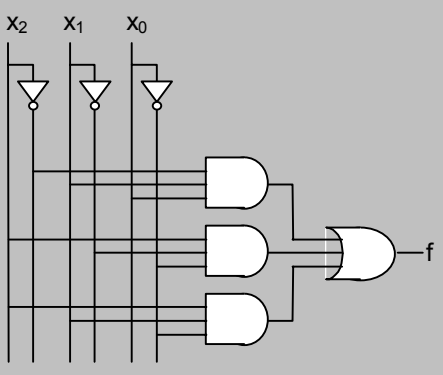
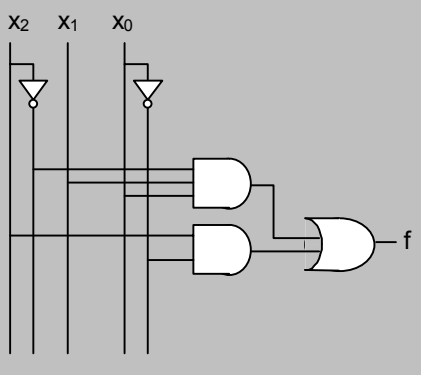


Figura 9b



2.3.2. Síntesi mínima a 2 nivells. Mètode de Karnaugh

Mirant la taula de veritat de la funció podem detectar els casos en què l'expressió en suma de mintermes es pot minimitzar. Ara bé, en alguns casos ho podem veure fàcilment, en altres és més dificil. Per exemple, la figura 10a mostra la taula de veritat de la funció

$$f(x_2,x_1,x_0) = x_2'x_1$$

Només mirant la taula resulta fàcil obtenir aquesta expressió, perquè les combinacions per les quals la funció val 1 (els mintermes) estan en files consecutives.

La funció de la figura 10b és $f(x_2,x_1,x_0) = x_1'x_0$, perquè val 1 sempre que $x_1 = 0$ i $x_0 = 1$, independentment de quant valgui x_2. Però en aquest cas no és tan senzill de veure-ho a simple vista, perquè els mintermes no estan en files consecutives.

Figura 10

Figura 10a

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Figura 10b

x_2	x_1	x_0	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

El **mètode de Karnaugh** proporciona una mecànica senzilla per a detectar visualment els casos en què es pot minimitzar una expressió en suma de mintermes. Per tant, d'entre tots els circuits a 2 nivells que implementen una funció, permet obtenir fàcilment el més petit de tots.

El mètode de Karnaugh consta de 4 passos:

1) Traslladar la taula de veritat de la funció a una estructura que s'anomena *mapa de Karnaugh*
2) Detectar visualment els casos en què es pot treure factor comú
3) Deduir els termes producte més simples possible
4) Obtenir l'expressió mínima de la funció fent la suma lògica dels termes producte.

Vegem en detall com es duu a terme cadascun d'aquests passos.

1) Construcció del mapa de Karnaugh

El **mapa de Karnaugh** és una transcripció de la taula de veritat d'una funció a una estructura formada per caselles en la qual cada casella correspon a una combinació de les variables (i per tant a una fila de la taula de veritat).

Es diu que dues caselles del mapa són **adjacents** si corresponen a combinacions en les quals només canvia el valor d'una variable.

La figura 11 mostra l'estructura del mapa de Karnaugh per a funcions de 2, 3 i 4 variables. Per exemple, la casella que està ombrejada amb gris clar correspon a la combinació [a b c d] = [1 1 0 1].

També és possible aplicar el mètode de Karnaugh a funcions de més de 4 variables, però en aquest curs no s'estudiarà. De fet, per a aquests casos existeixen altres mètodes més adequats, que no estudiarem en aquest curs.

Figura 11

Estructura dels mapes de Karnaugh per a funcions de

2 variables

a / b	0	1
0		
1		

3 variables

a b / c	00	01	11	10
0				
1				

4 variables

a b / c d	00	01	11	10
00				
01				
11				
10				

Fixem-nos que a les capçaleres de les files i les columnes dels mapes de Karnaugh les combinacions no estan en ordre lexicogràfic.

D'aquesta manera es compleix que les caselles adjacents queden disposades en el mapa de la manera següent:

- dues caselles que tenen una aresta en comú són adjacents
- en els mapes de 3 i 4 variables, les caselles de la columna de més a la dreta són adjacents amb les de la columna de més a l'esquerra
- en els mapes de 4 variables, les caselles de la fila superior són adjacents amb les de la fila inferior

Una vegada dibuixat el mapa, posarem dins de cada casella el valor de la funció per la combinació corresponent de variables, a partir de la taula de veritat. A la figura 12 es pot veure un exemple.

Figura 12

x_3	x_2	x_1	x_0	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$x_1 x_0$ \ $x_3 x_2$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	1	0
10	1	0	1	1

De fet, n'hi ha prou amb omplir les caselles per les quals la funció val 1, i així és com ho farem habitualment.

2) Detecció dels casos en què es pot treure factor comú

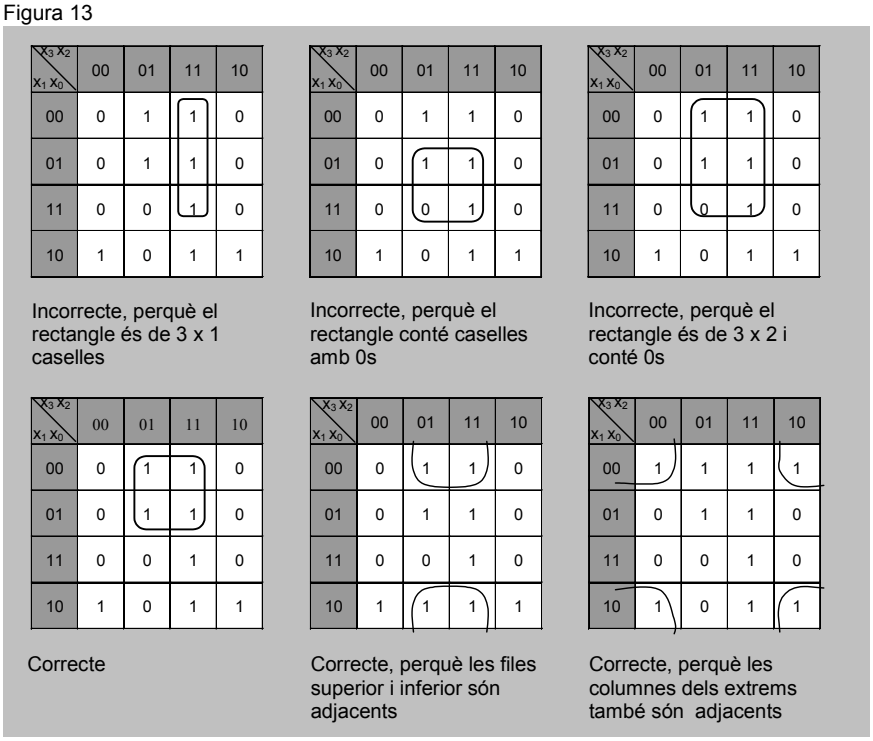
Suposem que dues caselles adjacents contenen 1s; corresponen, doncs, a dos mintermes de la funció. Però entre elles només canvia el valor d'una variable (per la definició d'adjacència), i per tant corresponen a un cas en què es pot treure factor comú dels dos mintermes (eliminant la variable que canvia de valor).

Suposem ara que quatre caselles adjacents contenen 1s. Entre elles només canvia el valor de 2 variables, i per tant corresponen a un cas en què es pot treure factor comú dues vegades.

Així, el segon pas del mètode de Karnaugh consisteix en agrupar amb rectangles els 1s que estiguin en caselles adjacents, formant grups de 1, 2, 4, 8 o 16 1s. Els costats d'aquests rectangles han de ser d'un nombre de caselles potència de 2, i al seu interior només hi pot haver 1s.

A la figura 13 es mostren diversos exemples de rectangles correctes i incorrectes.

Figura 13

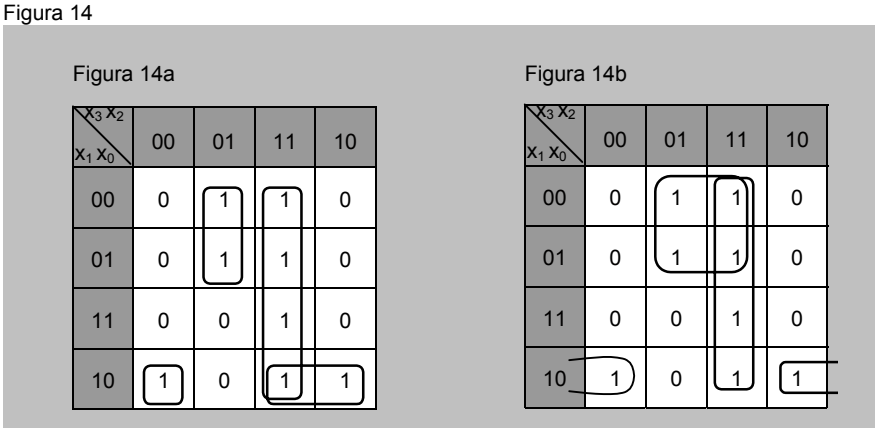


La manera d'agrupar els 1s d'un mapa no és única. En fer les agrupacions, cal tenir en compte el següent:

- tots els 1s han de formar part d'algun grup
- els grups han de ser com més grans millor (per tal que a cada terme hi aparegui el menor nombre possible de variables)
- com menys grups hi hagi millor (per tal d'obtenir el menor nombre possible de termes producte)
- un mateix 1 pot formar part de més d'un grup, si això ajuda a satisfer els dos objectius anteriors

A la figura 14 es mostren dues maneres d'agrupar els 1s del mapa de l'exemple anterior. Totes dues són correctes, però la de la figura 14b és millor. Sempre cal procurar trobar l'agrupació òptima.

Figura 14



3) Deducció dels termes producte

Prenem el mapa de la figura 14b. El grup dels 4 1s de la tercera columna correspon a les combinacions $[x_3\, x_2\, x_1\, x_0]$ =

[1 1 0 0]
[1 1 0 1]
[1 1 1 0]
[1 1 1 1]

L'expressió en suma de mintermes que s'obtindria d'aquests quatre 1s és $x_3x_2x_1'x_0' + x_3x_2x_1'x_0 + x_3x_2x_1x_0' + x_3x_2x_1x_0$. Traient factor comú obtenim x_3x_2, perquè el valor de la funció és independent d'x_1 i x_0. Mirant el mapa, podem veure que les variables x_3 i x_2 no canvien de valor dintre del rectangle, mentre que x_1 i x_0 prenen totes les combinacions possibles.

Així doncs, de cada grup n'obtindrem un terme producte, de la manera següent:

- només hi apareixen les variables el valor de les quals és constant per totes les caselles que formen el grup
- si en totes les caselles del grup una variable val 1, la variable apareix al terme producte sense negar
- si en totes les caselles del grup una variable val 0, la variable apareix al terme producte negada

Així, en el mapa de la figura 14b, de l'altre grup de 4 1s n'obtenim el terme

$$x_2x_1'$$

Del grup de 2 1s n'obtenim el terme

$$x_2'x_1x_0'$$

Observem que a partir d'un rectangle de 2^m 1s obtenim un terme producte amb n-m variables, sent n el número de variables de la funció. Per exemple, en el terme $x_2'x_1x_0'$, que s'obté a partir d'un rectangle de dos 1s (m=1, n-m=3), hi ha 3 variables. Mentre que el terme x_2x_1' té només 2 variables perquè s'obté d'un rectangle amb quatre 1s (m=2, n-m=2) .

4) Obtenció de l'expressió mínima de la funció

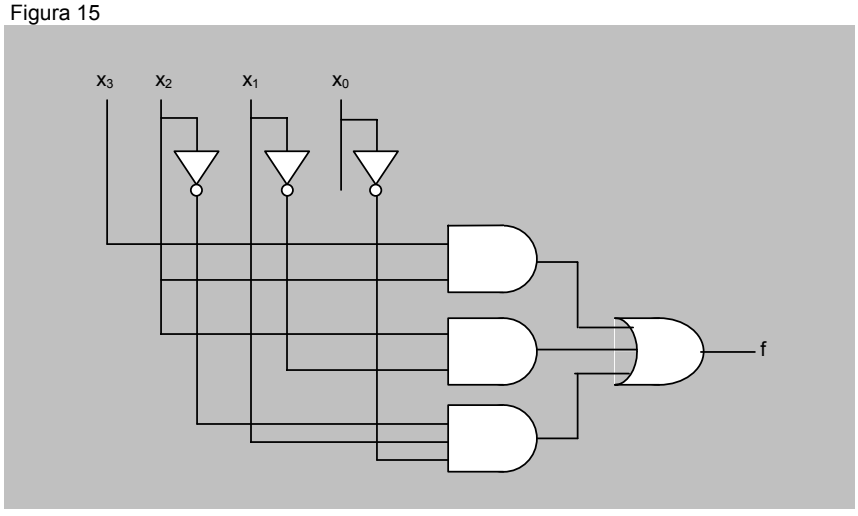
Ja sabem que una funció es pot expressar fent la suma lògica de tots els seus mintermes. Els termes producte obtinguts en el pas anterior "resumeixen" els mintermes d'una funció. Per tant, podem expressar la funció fent la suma lògica d'aquests termes producte.

En l'exemple de la figura 14b, l'expressió minimitzada de la funció és

$$f(x_3,x_2,x_1,x_0) = x_3x_2 + x_2x_1' + x_2'x_1x_0'$$

La figura 15 mostra el circuit a 2 nivells que implementa aquesta funció a partir de l'expressió minimitzada. Es pot veure que és més petit i barat que el que obtindríem a partir de l'expressió en suma de productes original, ja que aquest tindria 4 portes NOT, 8 portes AND de 3 entrades cadascuna i una porta OR de 8 entrades. De tots els circuits a 2 nivells que implementen aquesta funció, el de la figura 15 és el més petit i barat; es diu que és el **circuit mínim a 2 nivells**.

Figura 15



2.3.3. Minimització de funcions incompletament especificades

Tal com hem vist en l'apartat 1.5, el valor de les funcions incompletament especificades és indiferent per algunes combinacions de les variables (les combinacions "no importa"). És a dir, per les combinacions "no importa" tant podem suposar que la funció val 1 com que val 0.

A la taula de veritat de la funció posem "x" a les files corresponents a les combinacions "no importa". Al mapa de Karnaugh també posarem "x" a les caselles corresponents. Donat que el valor de la funció en aquests casos és indiferent, suposarem que les "x" són un "1" o un "0" segons més ens convingui, tenint en compte que sempre hem de procurar obtenir el menor nombre possible d'agrupacions i que les agrupacions siguin el més grans possibles.

Prenem la funció d'exemple que hem vist a l'apartat 1.5. La figura 16 mostra la seva taula de veritat i el mapa de Karnaugh, amb les agrupacions de caselles.

Figura 16

x_2	x_1	x_0	f
0	0	0	0
0	0	1	x
0	1	0	x
0	1	1	x
1	0	0	1
1	0	1	x
1	1	0	0
1	1	1	1

x_2 x_1 / x_0	00	01	11	10
0		x		1
1	x	x	1	x

Ens interessa suposar que les "x" de la fila inferior valen 1, perquè així obtenim un grup de 4 1s i un grup de 2 1s. Per tant, l'expressió mínima de la funció és

$$f = x_0 + x_2x_1'$$

Fixem-nos que la "x" de la casella $[x_2\ x_1\ x_0] = [0\ 1\ 0]$ l'hem pres com a 0, perquè si no obtindríem un grup addicional innecessàriament (els grups han de cobrir tots els 1s, però no cal que cobreixin totes les "x").

3. Blocs combinacionals

Un **bloc combinacional** és un circuit lògic combinacional amb una funcionalitat determinada. Està construït a partir de portes, com els circuits que hem vist fins ara.

Fins aquest moment, les "peces" que hem fet servir per a sintetitzar circuits han estat portes lògiques. Després d'estudiar aquest capítol, podrem fer servir també els blocs combinacionals com a peces per a dissenyar circuits més complexos, com per exemple un computador, que no es poden pensar a nivell de portes però sí a nivell de blocs.

3.1. Multiplexor. Multiplexor de busos

Imaginem que en una ciutat hi ha 3 carrers que conflueixen en un altre carrer d'un únic carril. Farà falta un urbà o algun tipus de senyalització per controlar que en cada moment circulin cap al carrer de sortida els cotxes provinents d'un únic carrer confluent.

Multiplexor

Un **multiplexor** és un bloc que fa la funció d'"urbà" en circuits electrònics. Té un cert nombre de senyals d'entrada que "competeixen" per connectar-se a un únic senyal de sortida, i uns senyals de control que serveixen per a determinar quin dels senyals d'entrada es connecta en cada moment amb la sortida.

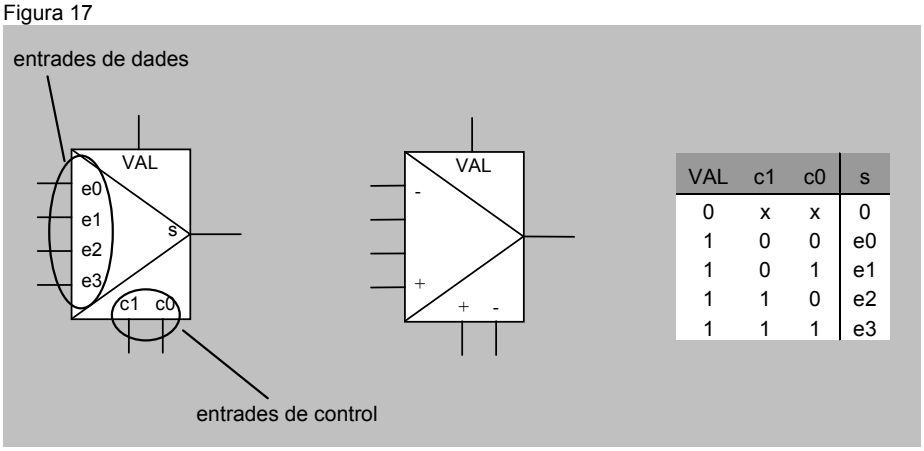
Més concretament, les entrades i sortides d'un multiplexor són les següents:

- 2^m entrades *de dades*, identificades per la lletra *e* i numerades des de 0 fins a 2^m-1. Direm que l'entrada de dades numerada amb el 0 és la *de menys pes*, i la numerada amb el 2^m-1 la *de més pes*.
- una sortida de dades, *s*
- m entrades *de control* o *de selecció*, identificades per la lletra *c* i numerades des de 0 fins a m-1. Direm que l'entrada de control numerada amb el 0 és la *de menys pes*, i la numerada amb el m-1 la *de més pes*.
- una entrada *de validació*, que anomenem *VAL*.

La figura 17 mostra dues possibles representacions gràfiques d'un multiplexor de 4 entrades de dades (m = 2). La diferència entre elles és que en la primera hi posem els noms de les entrades, mentre que en la segona només indiquem amb els signes "+" i "-" quines són les de més i menys pes (la resta s'assumeix que estan ordenades en ordre de pes). Totes dues representacions són igualment vàlides.

Per a especificar el tamany d'un multiplexor direm quantes entrades de dades té, o bé quantes entrades de control té. Per exemple, un "multiplexor de 8 entrades de dades" és el mateix que un "multiplexor de 3 entrades de control".

Figura 17



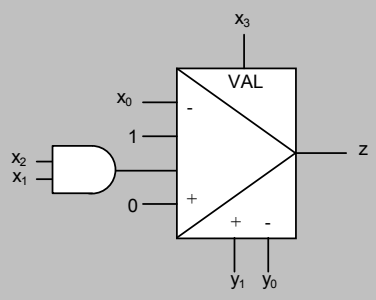
VAL	c1	c0	s
0	x	x	0
1	0	0	e0
1	0	1	e1
1	1	0	e2
1	1	1	e3

Si en un multiplexor no hi dibuixem l'entrada de validació, assumirem que aquesta està a 1.

La figura 17 també mostra la taula de veritat que descriu el funcionament del multiplexor, que és el següent:

- Quan l'entrada de validació val 0, la sortida del multiplexor es posa a 0 (i per tant el valor de les entrades de dades és indiferent, tal com reflecteixen les "x" a la taula de veritat).
- Quan l'entrada de validació val 1, llavors les entrades de control determinen quina de les entrades de dades es connecta amb la sortida, de la manera següent: s'interpreten les variables connectades a les entrades de control (c_1 i c_0 a l'exemple) com un número codificat en binari amb m bits (l'entrada de més pes correspon al bit de més pes); si el número codificat és *i*, l'entrada de dades que es connecta amb la sortida és la numerada amb el número *i*.

La figura següent mostra un exemple d'ús d'un multiplexor, i la taula de veritat que descriu el funcionament del circuit.



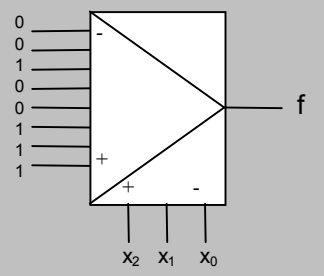
x_3	y_1	y_0	z
0	x	x	0
1	0	0	x_0
1	0	1	1
1	1	0	x_2 x_1
1	1	1	0

Una possible aplicació dels multiplexors és la implementació de funcions lògiques. La figura 18 mostra la implementació d'una funció amb un multiplexor que té tantes entrades de control com variables té la funció. La sortida del multiplexor valdrà 1 si

les variables connectades als senyals de control construeixen les combinacions 2, 5, 6 o 7, que corresponen als casos en què la funció f val 1.

Figura 18

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Multiplexor de busos

Un **mot d'n bits** és una agrupació d'n bits, usualment amb un significat semàntic conjunt (per exemple, un número codificat en binari amb n bits).

Per convenció, usarem lletres majúscules per donar nom a un mot o variable d'n bits. Cadascun dels bits que el formen l'identificarem per la mateix lletra en minúscula, junt amb un subíndex per a identificar el seu pes. Per exemple,

$$A = a_7a_6a_5a_4a_3a_2a_1a_0 = a_{7..0}$$

Per referir-nos a un subconjunt dels bits d'un mot escriurem els subíndexos corresponents. Per exemple, $a_{4..1}$.

Un **bus** és una agrupació de senyals lògics, o, dit d'una altra manera, un senyal que té més d'un bit. Així, un mot d'n bits circularà per un bus d'n bits.

Els busos es representen gràficament tal com es mostra a la figura 19a. A la línia que representa el senyal hi posem un ratlleta transversal acompanyada d'un número que indica el nombre de bits del bus.

A vegades interessa disgregar els bits que formen un bus, o agregar-hi bits addicionals. Les figures 19b, 19c i 19d mostren alguns exemples de com ho representarem gràficament.

Figura 19

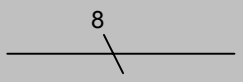


Figura 19a: bus de 8 bits

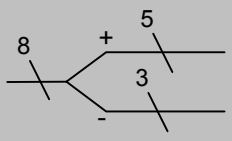


Figura 19b: el bus de 8 bits es disgrega en dos busos, un format pels 5 bits de més pes i un altre format pels 3 bits de menys pes.

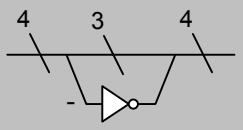


Figura 19c: la línia inferior correspon al bit de menys pes del bus de 4 bits. Aquest bit es nega i després s'agrega de nou al bus original

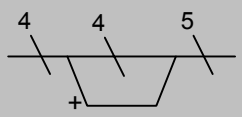
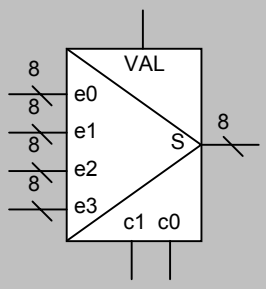


Figura 19d: la línia inferior correspon a una replicació del bit de més pes del bus de 4 bits, perquè el bus original segueix sent de 4 bits. Aquest nou bit s'agrega posteriorment al bus original, quedant finalment un bus de 5 bits.

Un **multiplexor de busos** funciona igual que un multiplexor de bits, però en aquest cas les entrades de dades i la sortida són busos d'un determinat nombre de bits. La figura 20 mostra la representació gràfica d'un multiplexor de busos de 8 bits de 4 entrades de dades.

Figura 20



VAL	c_1	c_0	S
0	x	x	0
1	0	0	e0
1	0	1	e1
1	1	0	e2
1	1	1	e3

3.2. Codificadors i descodificadors

La funció d'un **codificador** és generar la codificació binària d'un número donat.

Els codificadors disposen dels senyals següents:

- una entrada de validació, *VAL*, que funciona igual que en el cas dels multiplexors: si val 0 totes les sortides valen 0 (quan no dibuixem l'entrada de validació, assumirem que val 1)
- 2^m entrades de dades (d'un bit), identificades per la lletra *e* i numerades de 0 a 2^m-1 (la de número més alt és la de més pes)
- m sortides de dades (d'un bit), identificades per la lletra *s* i numerades de 0 a m-1, que s'interpreten com si formessin un número codificat en binari amb m bits (la sortida de més pes correspon al bit de més pes).

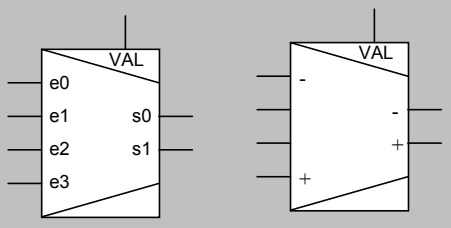
El funcionament d'un codificador és el següent:

Quan l'entrada de validació val 1, si l'entrada de més pes d'entre les que estan a 1 és la numerada amb el número *i*, llavors les sortides codifiquen en binari el número *i*.

La figura 21 mostra la representació gràfica d'un codificador 4-2 i la taula de veritat que explica el seu funcionament (observem que, com en el cas dels multiplexors, podem fer servir els símbols "+" i "-" en lloc dels noms de les entrades i sortides).

Per a especificar el tamany d'un codificador, direm "codificador 2^m-m"; per exemple, un "codificador 4-2" és un codificador de 4 entrades i 2 sortides.

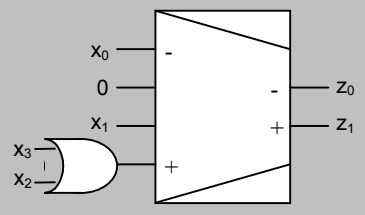
Figura 21



VAL	e3	e2	e1	e0	s1	s0
0	x	x	x	x	0	0
1	0	0	0	0	0	0
1	0	0	0	1	0	0
1	0	0	1	x	0	1
1	0	1	x	x	1	0
1	1	x	x	x	1	1

Fixem-nos que les sortides del codificador valen 0 tant si no hi ha cap entrada a1 com si està a 1 l'entrada de menys pes (i també quan l'entrada VAL és 0).

La figura següent mostra un exemple d'ús d'un codificador, i la taula de veritat que descriu el funcionament del circuit.



x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	x	1	0
0	1	x	x	1	1
1	0	x	x	1	1
1	1	x	x	1	1

Recordem que si en un codificador no hi dibuixem l'entrada de validació, assumirem que aquesta està a 1.

Els **descodificadors** fan la funció inversa als codificadors: donada una combinació binària present a l'entrada, indiquen a quin número decimal correspon.

Els descodificadors disposen dels senyals següents:

- una entrada de validació
- m entrades de dades, identificades per la lletra *e* i numerades de 0 a m-1, que s'interpreten com si formessin un número codificat en binari (l'entrada de més pes correspon al bit de més pes)
- 2^m sortides, identificades per la lletra *s* i numerades de 0 a 2^m-1, de les quals només una val 1 en cada moment (si l'entrada de validació està a 0, llavors totes les sortides valen 0).

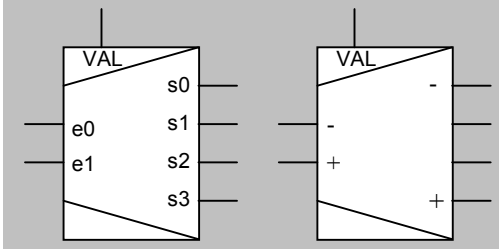
El funcionament d'un descodificador és el següent:

Quan l'entrada de validació val 1, si les entrades codifiquen en binari el número *i*, llavors es posa a 1 la sortida numerada com a *i*.

La figura 22 mostra la representació gràfica d'un descodificador 2-4 i la taula de veritat que descriu el seu funcionament.

Per a identificar el tamany dels descodificadors usarem la mateixa convenció que en el cas dels codificadors; per exemple, "descodificador 3-8".

Figura 22



VAL	e1	e0	s3	s2	s1	s0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Els descodificadors també es poden fer servir per a implementar funcions lògiques. Si la funció té n variables, usarem un descodificador n-2^n, i connectarem les variables a les entrades en ordre de pes. D'aquesta manera, la sortida *i* del descodificador es posarà a 1 quan les variables, interpretades com a bits d'un número en binari, codifiquin el número *i*. Per exemple, si connectem les variables [x_1 x_0] a les entrades [e1 e0] del descodificador de la figura 22, la sortida s2 valdrà 1 quan [x_1 x_0] = [1 0].

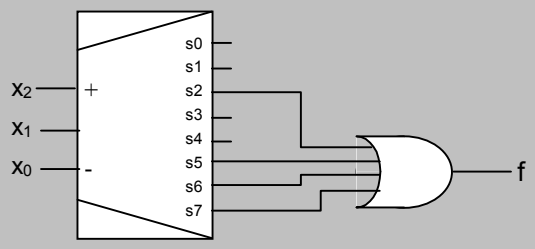
Per tant, per a implementar la funció serà suficient amb connectar a una porta OR les sortides corresponents a les combinacions que fan que la funció valgui 1. Quan alguna d'aquestes combinacions estigui present a l'entrada del descodificador, la sortida corresponent es posarà a 1 i per tant de la porta OR en sortirà un 1. Quan les variables construeixin una combinació que fa que la funció valgui 0, totes les entrades de la porta OR valdran 0 i per tant també la seva sortida.

La figura 23 en mostra un exemple. Podem comprovar que la sortida de la porta OR valdrà 1 quan valgui 1 la sortida s2 del descodificador, o la s5 o la s6 o la s7. És a

dir, quan les variables d'entrada construeixin alguna de les combinacions que fan que la funció valgui 1.

Figura 23

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Si en un codificador no hi dibuixem l'entrada de validació, assumirem que aquesta està a 1.

Quan dissenyem un circuit usant blocs podem deixar una o més sortides dels blocs sense connectar enlloc. Però, no podem deixar **cap** entrada sense connectar, perquè altrament el comportament del circuit seria indeterminat.

3.3. Decaladors lògics i aritmètics

Un **decalador** és un bloc combinacional que té la funció de desplaçar bits cap a l'esquerra o cap a la dreta.

Els decaladors tenen un senyal d'entrada i un senyal de sortida, tots dos d'n bits. El senyal de sortida s'obté desplaçant els bits d'entrada m vegades cap a la dreta o cap a l'esquerra.

- Si el desplaçament és cap a l'esquerra, els m bits de menys pes de la sortida es posen a 0.
- Si el desplaçament és cap a la dreta, hi ha dues possibilitats per als m bits de més pes de la sortida:

  - en els **decaladors lògics** es posen a 0
  - en els **decaladors aritmètics** prenen el valor del bit de més pes de l'entrada. Fixem-nos que, si interpretem les entrades i sortides com a números codificats en complement a 2 amb n bits, el que fan els decaladors aritmètics és mantenir a la sortida el signe de l'entrada.

Les figures 24a i 24b mostren la representació gràfica i la implementació interna d'un decalador d'1 bit a l'esquerra i d'un decalador aritmètic de 2 bits a la dreta, respectivament, de senyals de 4 bits.

Si interpretem les entrades i sortides com a codificacions de números, podem dir que els decaladors fan les funcions de multiplicar i dividir per potències de 2 (divisió entera). Un decalador d'm bits a l'esquerra multiplica per 2^m, i un decalador d'm bits a la dreta fa la divisió entera per 2^m. Si interpretem els números en binari natural hem d'usar decaladors lògics, si els interpretem en complement a 2 cal usar decaladors aritmètics.

Figura 24

Figura 24a: Decalador (lògic o aritmètic) d'1 bit a l'esquerra

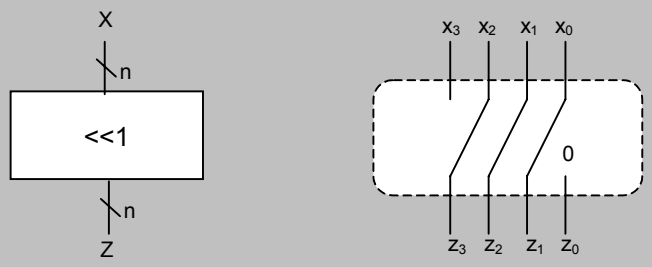
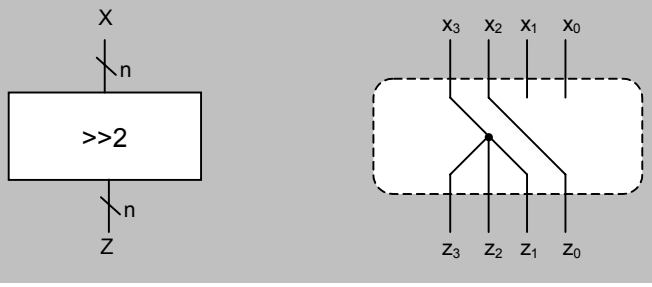
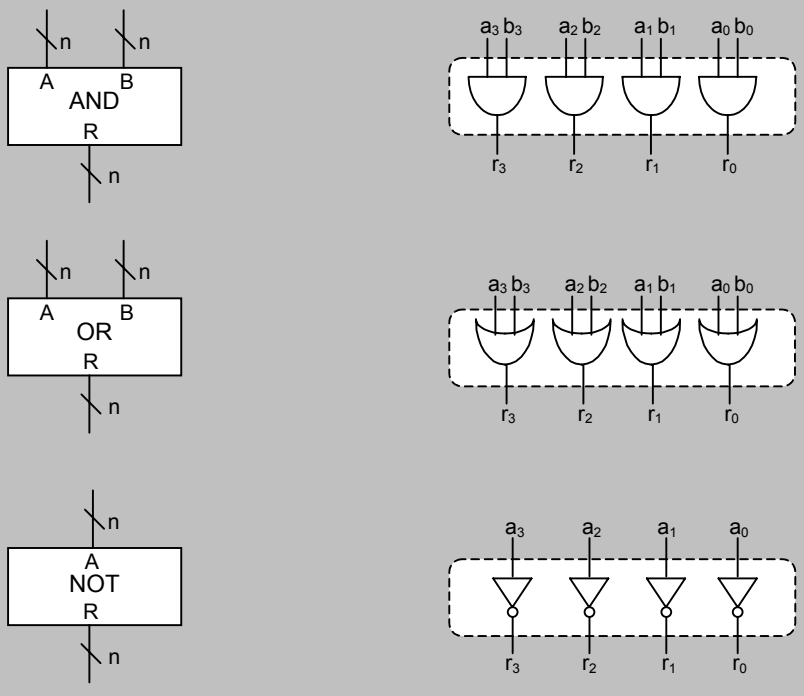


Figura 24b: Decalador aritmètic de 2 bits a la dreta



3.4. Blocs AND, OR i NOT

Figura 25



Aquests blocs fan les operacions AND, OR i NOT bit a bit sobre entrades d'n bits.

Tenen dues entrades de dades (només una en el cas del bloc NOT) i una sortida, totes elles d'n bits. La figura 25 mostra la seva representació gràfica i la implementació interna per al cas n = 4.

3.5. Memòria ROM

La denominació "ROM" deriva de l'anglès *Read Only Memory*, perquè es refereix a memòries en les quals no es poden realitzar escriptures sinó només lectures.

La **memòria ROM** és un bloc combinacional que permet guardar el valor de 2^m mots d'n bits.

Podem veure una memòria ROM com un arxivador amb calaixos que guarden bits. Cada calaix té una certa capacitat (tots tenen la mateixa), i l'arxivador té un nombre determinat de calaixos, que és sempre una potència de 2.

La memòria ROM té els elements següents:

- 2^m mots o *dades* d'n bits, cadascuna en una **posició** (calaix) diferent de la memòria ROM. Les posicions que contenen les dades estan numerades des del 0 fins al 2^m-1, i a aquests números se'ls diu **adreces**.
- una *entrada d'adreces* d'm bits, que s'identifica pel símbol @. Els m bits de l'entrada d'adreces s'interpreten com a números codificats en binari (i per tant cal determinar quin és el pes de cada bit).
- una sortida de dades d'n bits.

El funcionament de la ROM és el següent:

Quan els m bits de l´entrada d´adreces (interpretats en binari) codifiquen el número *i*, llavors la sortida pren el valor de la dada que hi ha emmagatzemada a l´adreça *i*. Per a referir-nos a aquesta dada usarem la notació **M[i]**, i direm que estem **llegint** la dada de l'adreça *i*.

Així doncs, només es pot accedir al valor d'un mot (llegir-lo) en cada instant (és com si en cada moment només es pugués obrir un calaix).

La figura 26a mostra com representarem la memòria ROM. La figura 26b mostra un possible contingut d´una memòria ROM de 4 mots de 3 bits. En aquest exemple,

M[0] = 001
M[1] = 100
M[2] = 111
M[3] = 001

Figura 26

Figura 26a

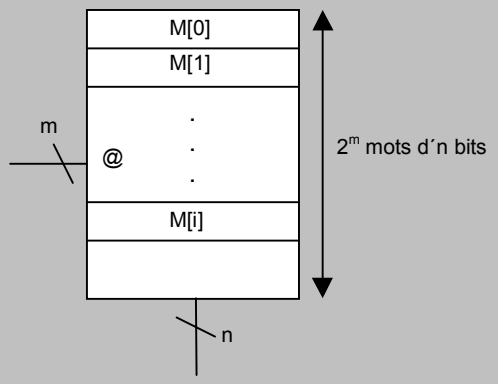


Figura 26b

@	M[@]
0	0 0 1
1	1 0 0
2	1 1 1
3	0 0 1

La memòria ROM també es pot fer servir per a sintetitzar funcions lògiques. Per exemple, si a les entrades d'adreces de la ROM de la figura 26b hi connectem les variables x_1 i x_0 (en ordre de pes), llavors podem interpretar els 3 bits de sortida com la implementació de les 3 funcions següents:

x_1	x_0	f_2	f_1	f_0
0	0	0	0	1
0	1	1	0	0
1	0	1	1	1
1	1	0	0	1

3.6. Comparador

El **comparador** és un bloc combinacional que compara 2 números codificats en binari i indica quina relació existeix entre ells.

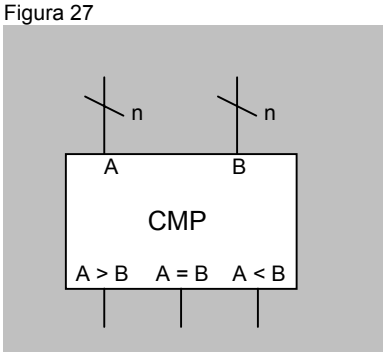
Disposa dels senyals següents:

- dues entrades de dades d'n bits, que reben els noms *A* i *B*. S'interpreten com a números codificats en binari.
- 3 sortides d'un bit, de les quals només una val 1 en cada moment:

  - la sortida $A > B$ val 1 si el número que arriba per l'entrada A és més gran que el que arriba per l'entrada B
  - la sortida $A = B$ val 1 si els dos números d'entrada són iguals
  - la sortida $A < B$ val 1 si el número que arriba per l'entrada A és més petit que el que arriba per l'entrada B

La figura 27 mostra la representació gràfica d'un comparador.

Figura 27



3.7. Sumador

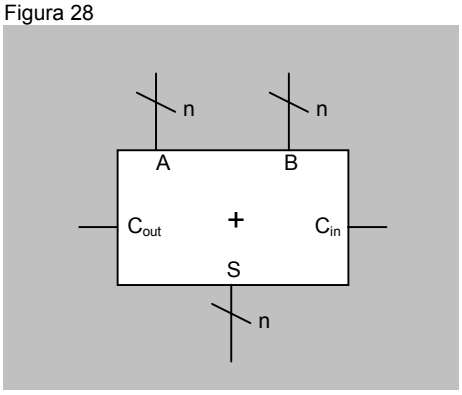
El **sumador** és un bloc combinacional que realitza la suma de dos números codificats en binari o bé en complement a 2.

Tal com s'ha estudiat en el mòdul "Representació de la informació", la representació dels enters en complement a 2 permet realitzar les sumes fent servir la mateixa mecànica que en les sumes de números codificats en binari. Per tant, si un bloc realitza la suma de números codificats en binari, la seva sortida també serà la suma correcta si interpretem els números d'entrada com a enters codificats en complement a 2. Per això diem que el sumador suma números codificats en binari *o bé* en complement a 2.

La figura 28 mostra la representació gràfica d'un sumador d'n bits. Els senyals de què disposa són els següents:

- dues entrades de dades d'n bits, que anomenarem *A* i *B*, per on arribaran els números a sumar
- una sortida d'n bits, *S*, que prendrà el valor de la suma dels números A i B
- una sortida d'un bit, C_{out}, que valdrà 1 si en fer la suma es produeix transport en el bit de més pes
- una entrada d'un bit, C_{in}, per on arriba un transport d'entrada.

Figura 28

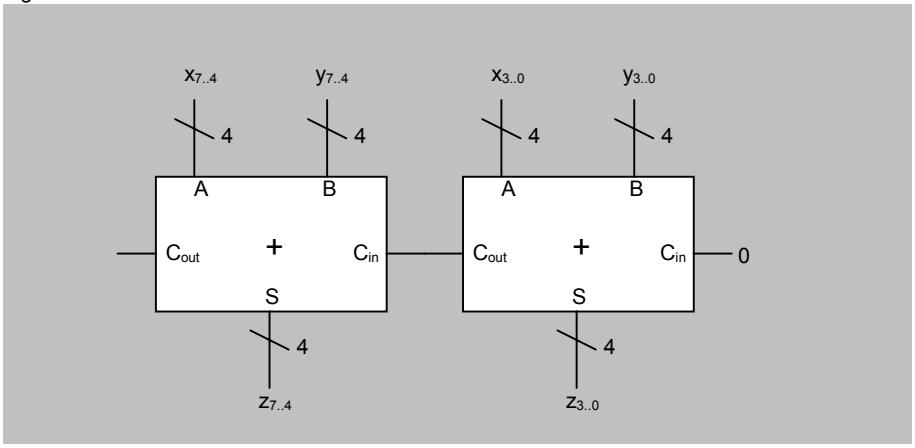


La nomenclatura dels senyals d'entrada i sortida de transport (C) deriva de la paraula "***Carry***", que és el mot anglès per transport.

L'entrada C_{in} és útil quan es volen sumar números de 2·n bits i només es disposa de sumadors d'n bits. En aquest cas s'encadenen dos sumadors: el primer suma els n bits més baixos dels números i el segon els n bits més alts. La sortida C_{out} del primer sumador es connecta amb l'entrada C_{in} del segon, per tal que el resultat sigui correcte. La figura 29 mostra un exemple per al cas n = 4, en què fem la suma Z = X+Y essent X, Y i Z números de 8 bits. Fixem-nos que el sumador que suma els bits més baixos el dibuixem a la dreta, perquè així els bits queden ordenats de la manera com estem acostumats a veure'ls.

Si no necessitem tenir en compte un transport d'entrada, connectarem un 0 a l'entrada C_{in}.

Figura 29



Com ja sabem, el fet de limitar la longitud dels números a un determinat nombre de bits (n) té com a conseqüència que el resultat d'una suma no sigui sempre correcte (és incorrecte quan es produeix sobreeiximent, és a dir, quan el resultat requereix més d'n bits per ser codificat). En les sumes binàries, sabem que si es produeix transport en el bit de més pes llavors el resultat és incorrecte. En canvi, en les sumes en complement a 2 no hi ha cap relació entre el transport i el sobreeiximent.

Recordeu...

...el concepte de sobreeiximent que s'ha estudiat al mòdul "Representació de la informació".

Per tant, la sortida C_{out} d'un sumador ens indica que s'ha produït sobreeiximent si interpretem les entrades en binari, però no ens diu res sobre la correctesa del resultat si les interpretem en complement a 2.

Donat que X - Y = X+(-Y), els sumadors es poden utilitzar també per a realitzar la resta de dos números, canviant el signe del segon operand abans de connectar-lo al sumador.

Recordeu...

...l'operació de canvi de signe que s'ha estudiat al mòdul "Representació de la informació".

3.8. Unitat aritmètica i lògica (UAL)

Una **unitat aritmètica i lògica** és un aparell capaç de realitzar un determinat conjunt d'operacions aritmètiques i lògiques sobre dos números d'entrada codificats en binari o en complement a 2.

A les unitats aritmètiques i lògiques se'ls anomena UAL o, també, ALU (de l'anglès *Arithmetic and Logic Unit*).

Per a dissenyar una UAL cal especificar el conjunt d'operacions que volem que realitzi. Per exemple, una UAL pot fer la suma, la resta, l'AND i l'OR dels operands d'entrada. En cada moment s'especificarà a la UAL quina és l'operació que ha de fer.

En aquest curs només estudiarem UALs que operin amb números naturals i enters. Els processadors tenen, a més, UALs per a operar amb números reals codificats en coma flotant.

Els senyals de què disposa una UAL són els següents:

- dues entrades de dades d'n bits, *A* i *B*, per on arribaran els números sobre els quals s'han de fer les operacions.
- una sortida de dades d'n bits, *R*, on s'obtindrà el resultat de l'operació

- un cert nombre d'entrades de control, c_i, d'1 bit cadascuna. Si la UAL és capaç de realitzar 2^m operacions diferents, ha de tenir m entrades de control. Cada combinació de les entrades de control indicarà a la UAL que faci una operació concreta. Per exemple, la UAL de la figura 30 pot fer 4 operacions.
- un cert nombre de sortides d'1 bit, que s'anomenen **bits d'estat**, i tenen la funció d'indicar algunes circumstàncies que es poden haver produït durant el càlcul.

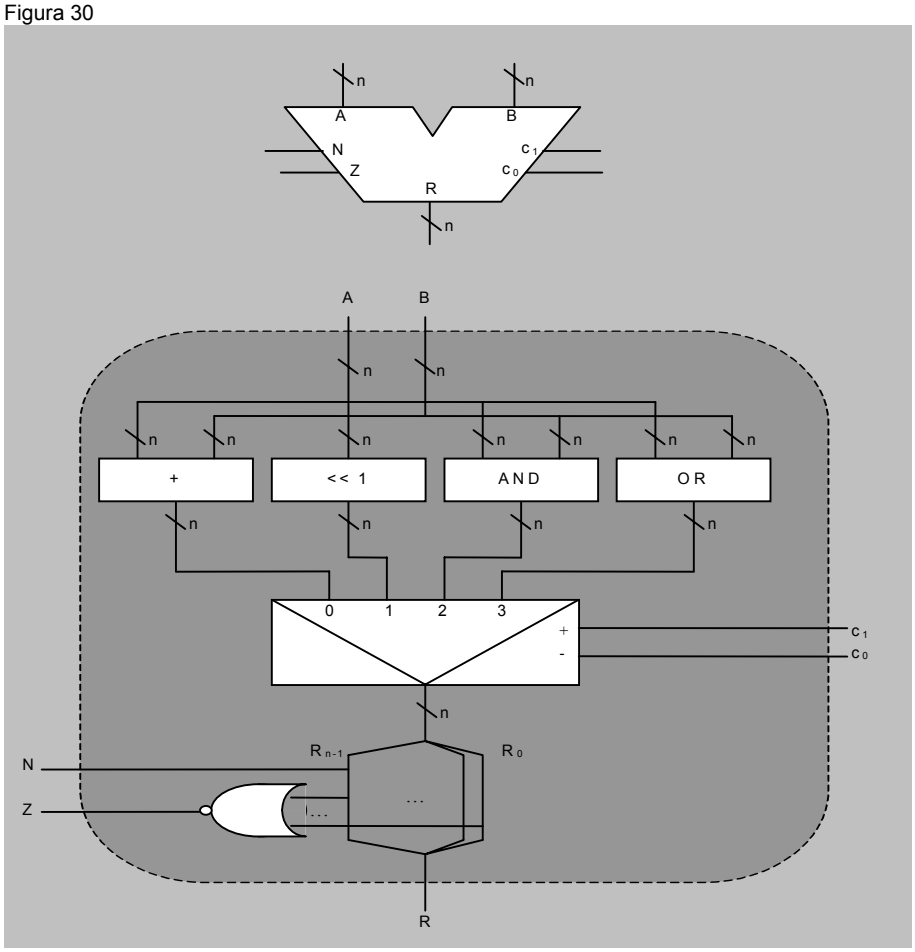
Els bits d'estat més habituals són els que indiquen si s'ha produït transport en l'últim bit (s'anomena *C*), si s'ha produït sobreeiximent (*V*), si el resultat de l'operació ha estat negatiu (*N*) o si el resultat de l'operació ha estat zero (*Z*).

El bit de sobreeiximent se sol identificar amb les lletres "O" o "V", que deriven de la paraula anglesa per sobreeiximent, "***Overflow***".

La figura 30 mostra la representació gràfica i la implementació interna d'una UAL que genera els bits d'estat N i Z i que realitza les operacions següents:

c_1	c_0	R
0	0	A + B
0	1	2*A
1	0	A AND B
1	1	A OR B

Figura 30



En aquesta figura...

...hem disgregat els bits del bus corresponent al senyal de sortida R per a poder dibuixar la implementació dels bits N i Z. S'ha d'interpretar que tots els bits d'R es connecten a la porta NOR (d'n entrades) que computa Z.

Aquesta és una forma senzilla però ineficient per dissenyar ALUs. Al capítol 5 del llibre “Principios de diseño digital”, de Daniel D. Gajski, trobareu una forma més eficient, que és la que s’explica a classe.

Resum

En aquest mòdul s'han estudiat els fonaments dels circuits electrònics digitals. S'ha vist que es construeixen a partir dels mateixos elements que la lògica natural, formalitzada matemàticament per l'àlgebra de Boole: els elements bàsics són els valors 0 i 1 i les operacions suma lògica, producte lògic i negació. A partir d'aquí s'han introduït les variables i funcions lògiques.

S'han conegut dues maneres de representar funcions lògiques: les expressions algebraiques i les taules de veritat. S'ha vist que són especialment còmodes les expressions en suma de productes.

Després s'ha vist com les operacions lògiques bàsiques s'implementen físicament per construir circuits, mitjançant les portes lògiques. S'ha après la manera de sintetitzar circuits el més ràpids, petits i barats possible, mitjançant el mètode de minimització de Karnaugh.

Finalment, s'han conegut diversos blocs combinacionals, i s'ha après a utilitzar la funcionalitat de cadascun per a dissenyar i entendre circuits complexos.

Bibliografia

Hermida R., Del Corral A., Pastor E. y Sánchez F. (1998)
Fundamentos de Computadores Madrid: Editorial Síntesis

Gajsky, D.D. (1997) *Principios de diseño Digital* Prentice Hall.