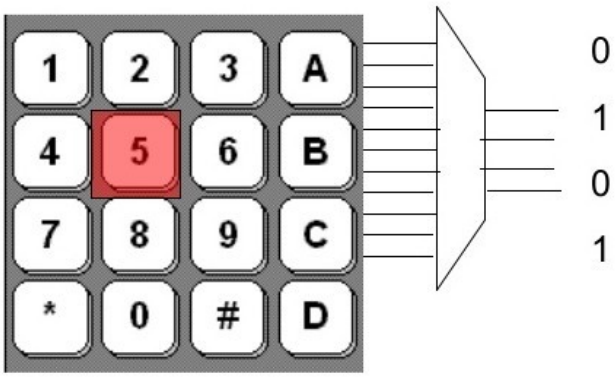
Structure Machine 2

A la découverte des circuits logiques

Avril à Juin 2020

Lhadi BOUZIDI
Lhadi.bouzidi@gmail.com



Les circuits logiques combinatoires (CLC)

Table des matières

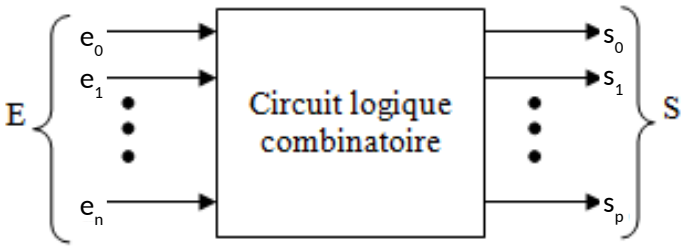
Chapitre 1 : Les circuits logiques combinatoires (CLC)

1 Introduction

Je vous rappel que l'on distingue deux types de circuits logiques : les circuits combinatoires et les circuits logiques séquentiels. Commençons d'abord par les circuits combinatoires. Mathématiquement parlant, on peut les définir par une équation reliant ses sorties à ses entrées. En considérant $E = (e_0, e_1, \ldots, e_n)$ comme les entrées de notre circuit et $S = (s_0, s_1, \ldots, s_p)$ comme ses sorties alors, on peut écrire :

$S = F(E)$ où $s_i = f_i(e_0, e_1, \ldots, e_n)$ avec i allant de 0 à p.

D'un point vue schématique on aura :

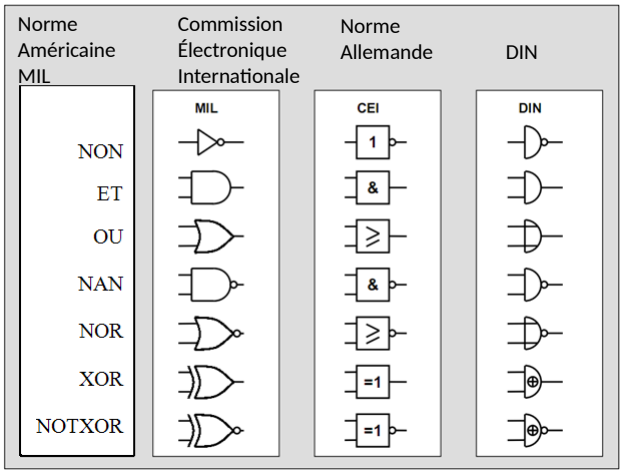


Dans ce qui suit, nous allons définir ce que c'est que la synthèse et l'analyse de circuits logiques combinatoires et nous allons présenter la synthèse que quelques circuits courants.

Remarque :

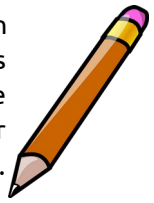
- Dans le chapitre précédent, nous parlions d'opérateurs logiques de base. Dans ce chapitre nous parleront plutôt de porte logiques (ET, OU, NON, NAND, NOR et XOR). Pour traiter ce chapitre, vous devez impérativement connaître l'algèbre de Boole.

- Nous allons nous servir de symboles pour représenter les portes logiques. Vous allez vous rendre compte que dans la littérature, on utilise des symboles différents pour représenter les mêmes portes logiques. En fait, il existe plusieurs normes de représentations :



Dans ce qui suit, nous utiliserons la norme MIL.

2 Analyse et synthèse de circuit logiques combinatoires



La synthèse d'un circuit logique combinatoire a pour but la réalisation d'une fonction logique qui remplit un cahier des charges et qui satisfait également à d'autres critères tels que le coût et l'encombrement minimum par exemple. Le nombre de circuits à produire, le matériel à disposition, le délai de réalisation, etc. sont d'autres paramètres dont il faut tenir compte lors de la synthèse. De façon générale, la simplification d'un circuit est toujours utile.

Concrètement, il s'agit de déterminer le logigramme associé à une fonction logique connaissant la définition de cette dernière.

Voici les étapes à suivre pour réaliser la synthèse d'un circuit logique combinatoire :

1. Établir la table de vérité de chacune des fonctions impliquées dans le problème à traiter
2. Établir les équations logiques ou la table de Karnaugh
3. Simplifier les équations de chacune des fonctions logiques
4. Établir le logigramme du circuit logique
5. Réaliser le circuit logique

L'analyse d'un circuit logique consiste à trouver à partir d'un logigramme ses fonctions logiques. Le principe de l'analyse d'un circuit logique consiste à :

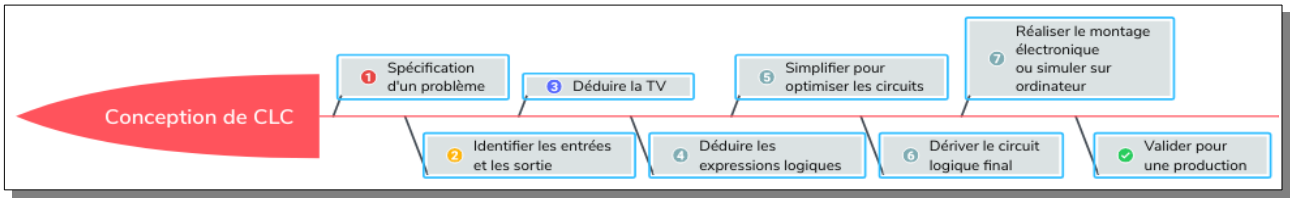
- Donnez l'expression de chaque porte en fonction des valeurs de ses entrées.
- En déduire au final la ou les fonction(s) logique(s) du circuit analysé.

On peut ensuite établir la table de vérité du circuit analysé et opéré à une simplification à 'laide des propriétés de l'algèbre de Boole de la table de Karnaugh.

3 Conception d'un CLC

La procédure de conception des circuits logiques combinatoires commence par la spécification du problème et comprend les étapes suivantes :

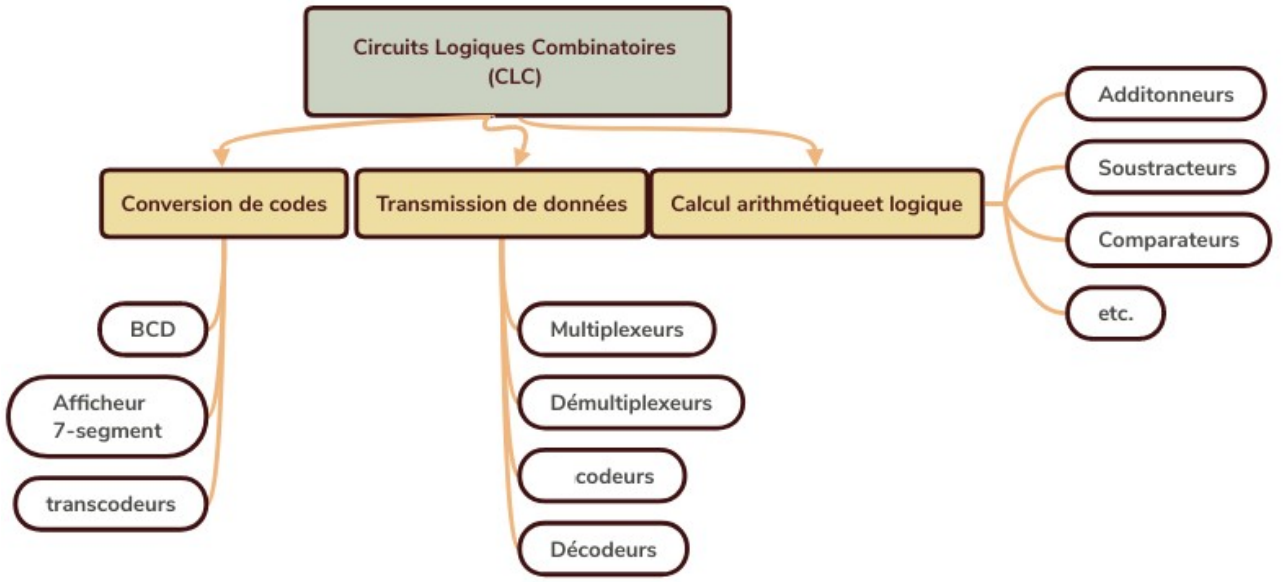
1. Déterminer le nombre requis d'entrées et de sorties à partir des spécifications du problème.
2. Déterminer la table de vérité pour chacune des sorties en fonction de leurs liens avec les entrées.
3. Simplifier l'expression booléenne pour chaque sortie. Utilisez les tableaux de Karnaugh ou l'algèbre booléenne.
4. Dessinez un diagramme logique (logigramme) qui représente l'expression booléenne simplifiée. Vérifier la conception en analysant ou en simulant le circuit.



4 Classification

On distingue au moins 3 classes de circuits logiques combinatoires :

- Les circuits de calcul arithmétiques et logiques,
- les circuits d'aiguillage et de transmission de données,
- les convertisseurs de codes.



Les circuits de calcul arithmétiques et logiques

Ce sont généralement des circuits logiques combinatoires permettant d'effectuer des calculs arithmétiques (addition, soustraction, multiplication) sur des entiers ou des nombre en virgule flottantes et des opérations logiques comme des négations, des ET, des OU ou des OU Exclusifs. On les trouve le plus souvent dans les unité de calculs des ordinateur communément appelées **UAL** ou unité arithmétique et logique (*ALU* en anglais!).

Les circuits de transmission de données

C'est un groupe de circuits permettant d'aiguiller les informations (données) binaires à travers des lignes électriques (souvent appelé BUS) d'une source (une petite mémoire appelée registre ou des capteurs, interrupteurs ou boutons poussoirs) vers une destination (registre ou un afficheur par exemple). Le décodeur, le multiplexeur en sont des exemples.

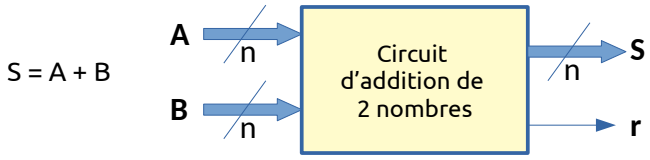
Les convertisseurs de code

Les nombres sont habituellement codés sous une forme ou une autre afin de les représenter ou de les utiliser au besoin. Par exemple, un nombre 'sept' est codé en décimal à l'aide du symbole $(7)_{10}$. Ce nombre est affiché sur votre calculatrice en se servant du codage 7 segments, mais au sein de l'unité de calcul de votre calculatrice, ce même nombre est codé en général en complément à 2. Bien que les ordinateurs numériques traitent tous des nombres binaires, il y a des situations où la représentation binaire naturelle des nombres n'est pas pratiques ou est inefficace ce qui nécessite des codes plus appropriés. Cette situation fait cohabiter, dans une même machine, diverses codes pour représenter une même information. Des circuits de conversion d'un code vers un autres sont donc utiliser.

5 Étude de quelques CLC

Additionneur

Il s'agit de faire la synthèse d'un circuit d'addition de 2 nombres ***A*** et ***B*** sur ***n bits*** chacun. Ce circuit doit avoir en entrée 2n variables (ce qui correspondant aux nombres A et B qui sont sur n bits chacun) et en sortie n bits qui correspondront à la sortie S et un seul bit qui correspond à la retenue finale du calcul.

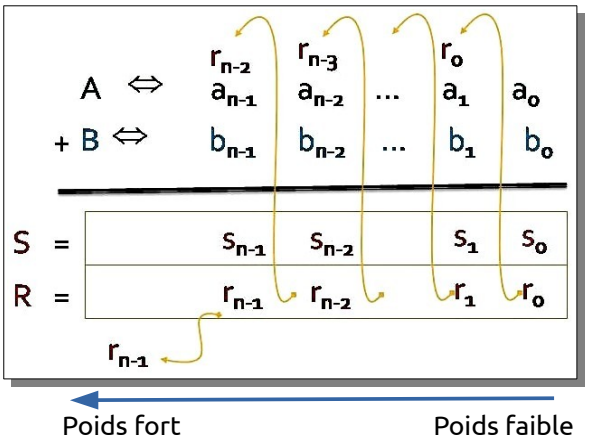


L'analyse rapide de ce problème nous apprend que nous avons ***2n*** entrées et ***n+1*** sorties au moins. Maintenant essayons de comprendre le lien entre les sorties et les entrées. Ce lien (ou cette relation) est déduite, bien évidemment des règles d'addition que nous avons déjà vue dans le chapitre sur les systèmes de numération (rappelez-vous l'arithmétique binaire!).

Si je me rappel bien, j'avais présenté 4 règles que voici :

0+0 = 0	retenue 0
0+1 = 1 + 0 = 1	retenue 0
1 + 1 = 0	retenue 1
1+1+1 = 1	retenue 1

Ces règles sont applicables pour chaque niveau des bits des deux nombres A et B. Ainsi le calcul se fait en allant du bit de poids faible vers le bits de poids fort.



Comme je l'ai dit ci-dessus, nous avons ***n*** sorties pour constituer la somme ***S*** et une sortie supplémentaire pour représenter la dernière retenue. Ainsi, nous pourrions être tentés de penser que ces ***n+1*** sorites dépendent des ***2n*** variables en entrées (rappelez-vous, n entrées pour constituer A et n entrées pour constituer B!). Mais l'analyse plus fine des règles de l'addition nous apprend que ce n'est pas vraie. En effet, une sortie s_i ne dépend pas de toutes les entrées a_j (j allant de 0 à n-1) du nombre ***A*** et b_k (k allant de 0 à n-1) du nombre **B**. En fait, le calcul de l'addition se fait étage par étage en allant de la gauche vers la droite (du poids faible vers le poids forts de nos deux nombres A et B).

Pour le premier étage nous avons 2 sorites : s_0 et r_0. Ces sorties sont des fonctions f_0 et g_0 de 2 entrées uniquement a_0 et b_0.

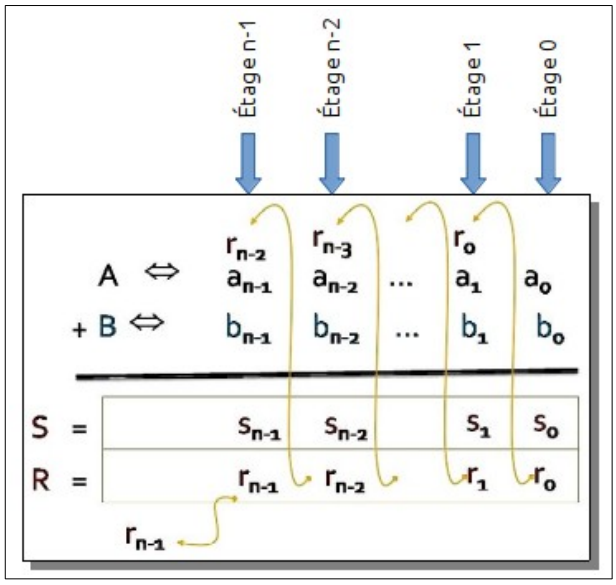
On écrira donc :

- ✔ $s_0 = f_0(a_0, b_0)$
- ✔ $r_0 = g_0(a_0, b_0)$.

Pour les autres étages, on voit aussi que les sorties s_i et r_i ne dépendent que des entrées a_i, b_i et r_{i-1}. Pour chacun de ces étage, on utilisera les mêmes règles de calcul (l'addition binaire).

On écrira donc :

- ✔ $s_i = f_1(a_i, b_0, r_{i-1})$
- ✔ $ri = g_1(a_0, b_0, r_{i-1})$.



En appliquant les règles de l'addition binaire, trouvons maintenant les tables de vérité des fonctions f_0 , g_0 , f_1 et g_1 en d'autres termes s_0 , r_0 , s_i et r_i

	a_0	b_0	s_0	r_0
m_0	0	0	0	0
m_1	0	1	1	0
m_2	1	0	1	0
m_3	1	1	0	1

	a_i	b_i	r_{i-1}	s_0	r_0
m_0	0	0	0	0	0
m_1	0	0	1	1	0
m_2	0	1	0	1	0
m_3	0	1	1	0	1
m_4	1	0	0	1	0
m_5	1	0	1	0	1
m_6	1	1	0	0	1
m_7	1	1	1	1	1

La détermination des équations de s_0 et r_0 est très simple :

- $s_0 = m_1 + m_2 = \bar{a}_0 . b_0 + a_0 . \bar{b}_0 = a_0 \oplus b_0$
- $r_0 = a_0 . b_0$

La détermination des équations de s_i et r_i n'est pas trop compliqué ! :

- $s_i = m_1 + m_2 + m_4 + m_7$
- $s_i = m_3 + m_5 + m_6 + m_7$

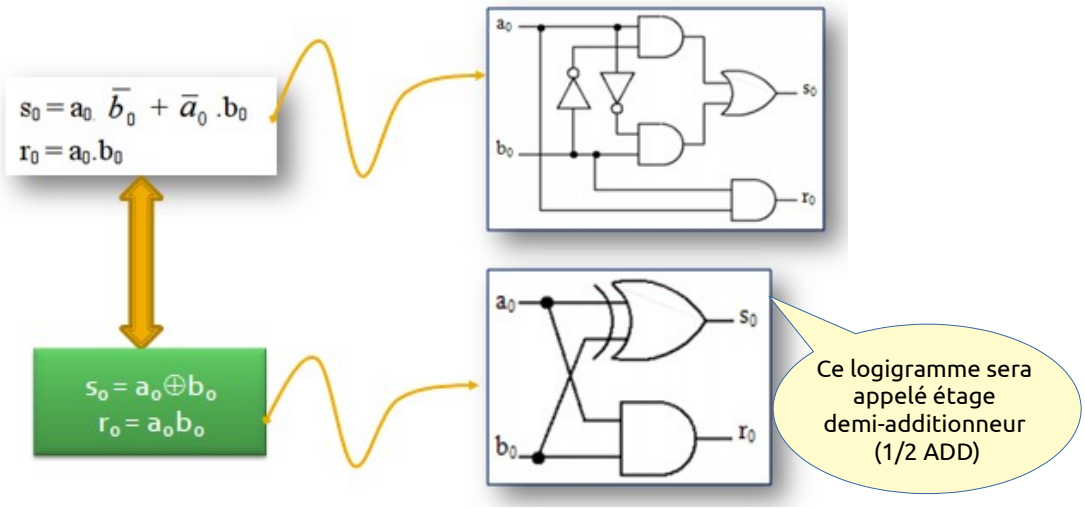
Essayons de simplifier la somme s_i:

$$
\begin{aligned}
s_i &= m_1 + m_2 + m_4 + m_7 \\
&= \bar{a_i}\,\bar{b_i}\,r_{i-1} + \bar{a_i}\,b_i\,\bar{r}_{i-1} + \\
&\quad a_i\,\bar{b_i}\,\bar{r}_{i-1} + a_i\,b_i\,r_{i-1} + \\
&= \bar{a_i}\,(\bar{b_i}\,r_{i-1} + b_i\,\bar{r}_{i-1}) + \\
&\quad a_i\,(\bar{b_i}\,\bar{r}_{i-1} + b_i\,r_{i-1}) \\
&= \bar{a_i}\,(b_i \oplus r_{i-1}) + a_i\,\overline{(b_i \oplus r_{i-1})} \\
&= a_i \oplus (b_i \oplus r_{i-1})
\end{aligned}
$$

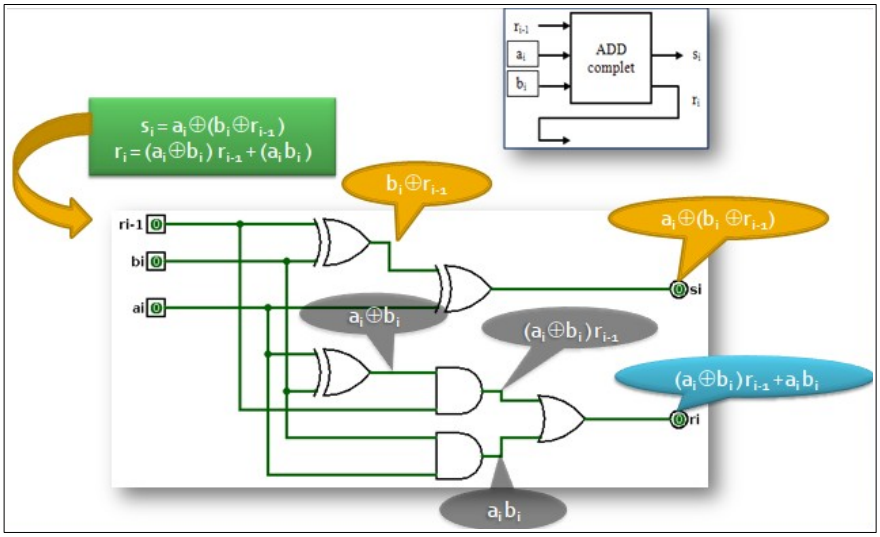
Simplifions maintenant la retenue r_i :

$$
\begin{aligned}
r_i &= m_3 + m_5 + m_6 + m_7 \\
&= \bar{a}_i b_i r_{i-1} + a_i \bar{b}_i r_{i-1} + \\
&\quad a_i b_i \bar{r}_{i-1} + a_i b_i r_{i-1} \\
&= \bar{a}_i b_i r_{i-1} + a_i \bar{b}_i r_{i-1} + a_i b_i (\bar{r}_{i-1} + r_{i-1}) \\
&= \bar{a}_i b_i r_{i-1} + a_i \bar{b}_i r_{i-1} + a_i b_i \\
&= (\bar{a}_i b_i + a_i \bar{b}_i) r_{i-1} + a_i b_i \\
&= (a_i \oplus b_i) r_{i-1} + a_i b_i
\end{aligned}
$$

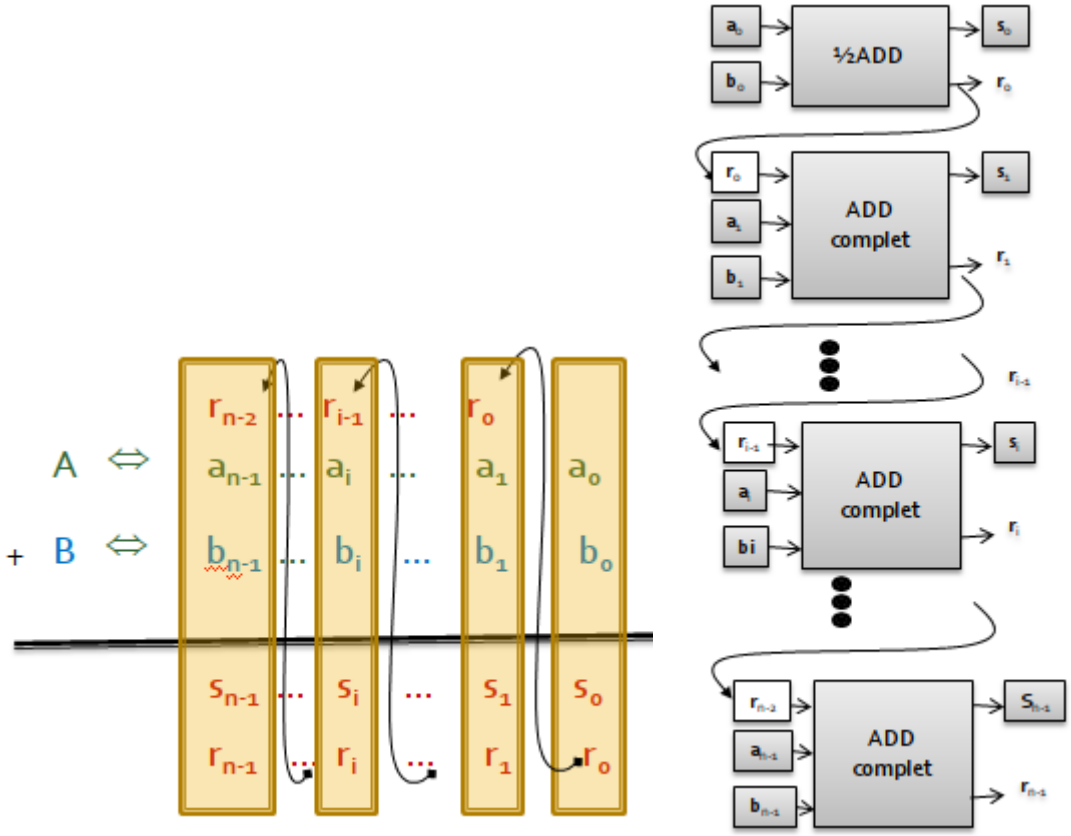
Au finale nous obtenons les logigrammes pour le premier étage de notre circuit d'addition:



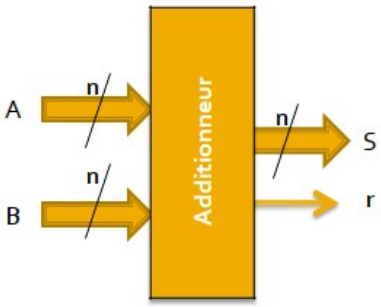
Nous obtenons le circuit logique suivant pour les sommes s_i et retenues r_i (i allant de 1 à n-1) :

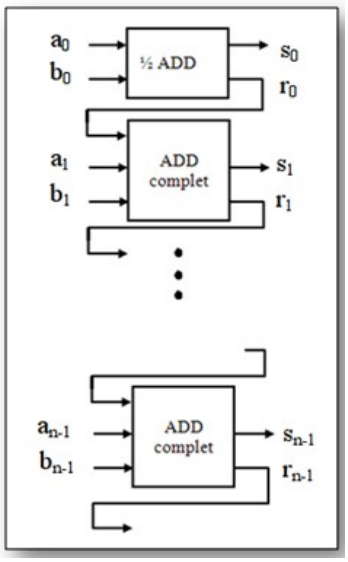


En utilisant le 1/2ADD pour le premier étage de l'additionneur et l'ADD complet pour les étages suivants on obtient le circuit suivant avec un montage en cascade :



Ce qui donne en définitif le circuit suivant :

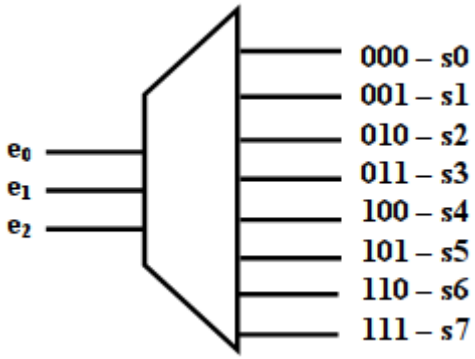




Il faut noter que le circuit additionneur fait partie d'une unité dite Unité Arithmétique et Logique (**UAL**) des ordinateurs. Il peut aussi être vendu sous forme d'un circuit intégré. Nous le verront dans le chapitre 3.

Décodeur

Un décodeur dispose de ***n*** entrées et de 2^n sorties. Il permet d'activer une ligne de sortie (sélection) correspondante à la configuration présentée en entrée. Le schéma suivant illustre un exemple de décodeur à trois entrées:



Décodeur 3:8

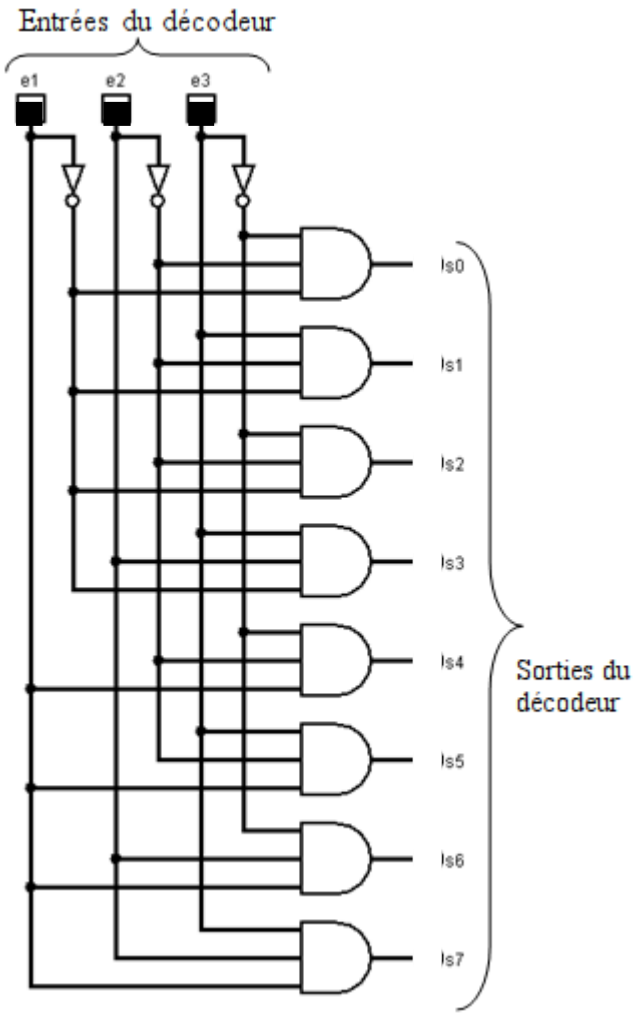
Afin de trouver les équations des sorties s_i du décodeur, il suffit de constater que $s_i = 1$ si le code représenté par les variables est ***i*** , c'est à dire que $m_i = 1$. On déduit donc que $s_i = m_i$ pour ***i*** allant de ***0*** à 2^n. Voici par exemple le décodeur 3→8 avec indication de l'équation de chacune de ses sorties.

Il faut noter, par ailleurs, que le plus souvent, une entrée de validation ***E*** (pour dire *Enable*) est utiliser pour valider l'utilisation du décodeur. Ainsi, lorsque cette entrée est à « ***0*** », toutes les sorties si sont à zéro (c'est à dire qu'aucune n'est sélectionnée). Par contre lorsque cette entrée est à « 1 », notre décodeur va fonctionner normalement, c'est à dire la sortie s_i (***i*** allant de ***0*** à 2^n , ***n*** étant le nombre d'entrées) sera à « 1 » (donc sélectionnée) si le code indiquée par les entrées est ***i***.

On aura donc les équations suivantes :

$s_i = E.m_i$, avec ***i*** allant de ***0*** à 2^n.

Voici le logigramme de notre décodeur 3→8 :

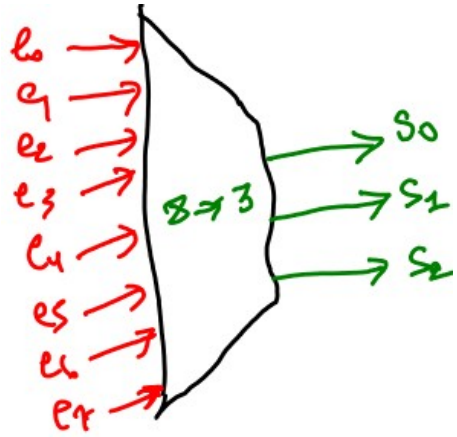


Codeur

Le codeur (ou encodeur) est un circuit qui possède (en général) ***n*** sorties et $N=2^n$ entrées. Son principe de fonctionnement repose sur le fait que la configuration indiquée par les ***n*** sorites correspond au numéro de la ligne d'entrée à « 1 ». Cela suppose donc qu'à un moment donnée, une seule entrée à « 1 » !

Lorsque plusieurs entrées peuvent être à « 1 » au même temps, on défit un ordre de priorité entre ces entrées de sorte à indiquer en sorite l'entrée prioritaire.

Dans ce qui suit, nous présenterons, à titre d'exemple, un codeur 8→3 (8 entrées et 3 sorties) :



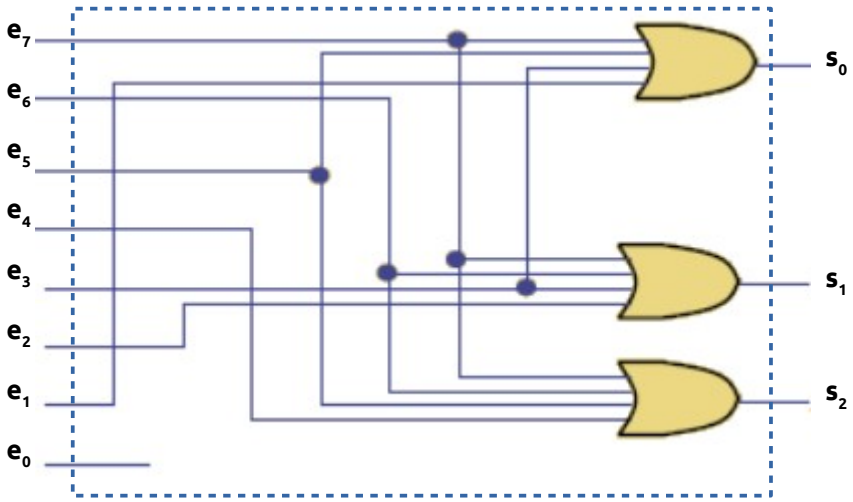
Voici la table de vérité que nous pouvons déduire du principe de fonction de l'encodeur 8→3 :

Entrées	**s_2**	**s_1**	**s_0**	**L'entrée indiqué par les 3 sorites**
e_0	0	0	0	l'entrée 0 (e_0)
e_1	0	0	1	l'entrée 1 (e_1)
e_2	0	1	0	l'entrée 2 (e_2)
e_3	0	1	1	l'entrée 3 (e_3)
e_4	1	0	0	l'entrée 4 (e_4)
e_5	1	0	1	l'entrée 5 (e_5)
e_6	1	1	0	l'entrée 6 (e_6)
e_7	1	1	1	l'entrée 7 (e_7)

Nous constatons ceci :

- s_2 est à « **1** » lorsque e_4 **OU** e_5 **OU** e_6 **OU** e_7 sont à « **1** », ce qui donne $s_2 = e_4+e_5+e_6+e_7$
- s_1 est à « **1** » lorsque e_2 **OU** e_3 **OU** e_6 **OU** e_7 sont à « **1** », ce qui donne $s_1 = e_2+e_3+e_6+e_7$
- s_0 est à « **1** » lorsque e_1 **OU** e_3 **OU** e_5 **OU** e_7 sont à « **1** », ce qui donne $s_0 = e_1+e_3+e_5+e_7$

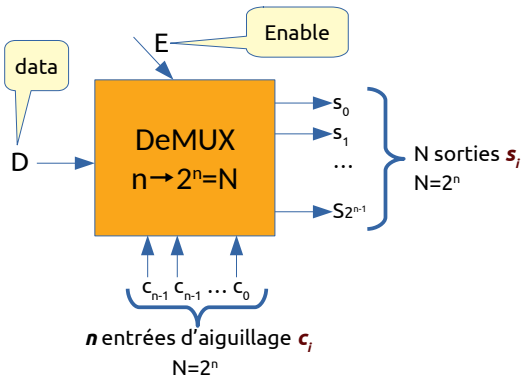
Ce qui nous donne le logigramme suivant :



Démultiplexeur

Un démultiplexeur (**DeMUX**) est un circuit comptant :

- une entrée (de données ou d'information)
- ***N*** sorties (destinations possibles)
- ***n*** entrée de commande (ou de sélection)
- une entrée ***E*** de validation (*Enable*)



Le principe de fonctionnement de ce circuit repose sur l'idée de permettre d'aiguiller (de faire passer) l'entrée ***D*** vers une seule sortie S_i selon le code (ou adresse) indiquée par les entrées de commandes c_i. Ceci fait que les ***n*** entrées de commande (sélection ou aiguillage) doivent être suffisantes pour coder ***N*** valeurs chacune associées aux ***N*** sorties du DeMUX. Ainsi, nous avons cette règle qui doit être respectée : $N=2^n$.

Voici la table de vérité qui découle de ce principe (sans prise en comptes de l'entrée de validation E) :

Entrée de commande (c_{n-1}, …, c_1, c_0)		s_0	s_1	…	s_i	…	S_N N=2^n
m_0	0… 00	D	0		0	…	0
m_1	0…01	0	D		0	…	0
. . .	. . .	0 . . . 0	0 . . . 0		0 . . . 0	…	0 . . . 0
m_i		0	0		D	…	0
. . .	. . .	0 . . . 0	0 . . . 0		0 . . . 0	…	0 . . . 0
m_N	1…11	0	0		0	…	D

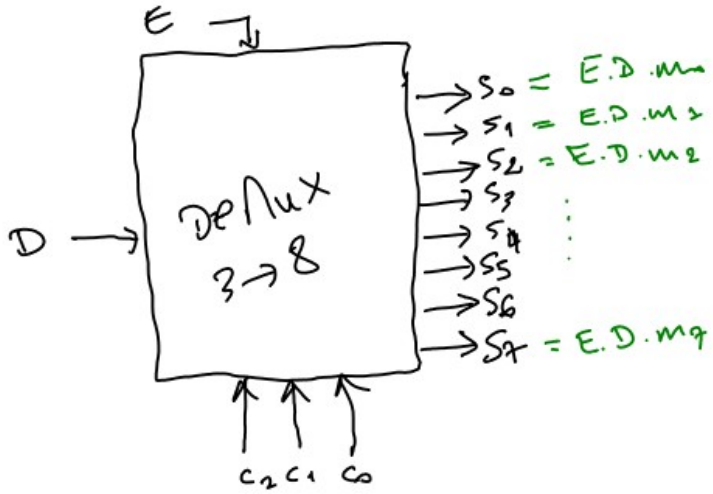
En se référant à la table de vérité ci-dessus, on déduit les équation des sorties :

$s_i = D \cdot m_i$ pour ***i*** allant de ***0*** à $N=2^{n-1}$.

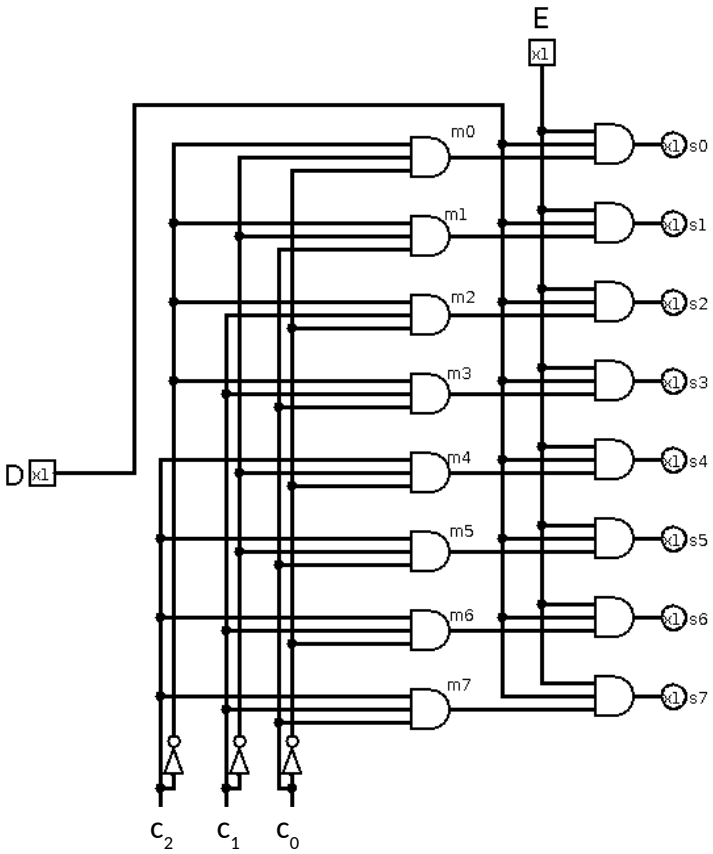
En prenant en compte l'entrée de validation ***E*** qui fait que si elle est à « 0 » aucune sortie ne sera sélectionnée et si elle est à « 1 » la sortie ***i*** sera sélectionnée si le minterme m_i est réalisé alors on aura cette équation :

$s_i = E. D \cdot m_i$

A titre d'exemple, prenons un DeMUX 3→8 :



Voici le logigramme qui en découle :

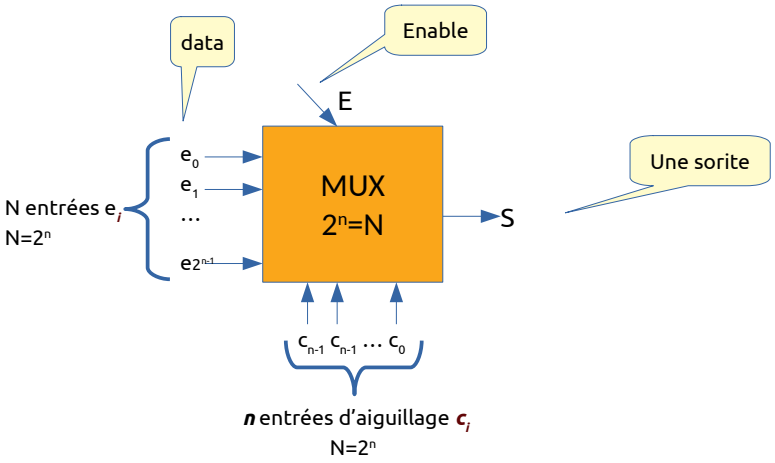


Multiplexeur

Un multiplexeur, réalise l'opération inverse du démultiplexeur. Il dispose de :

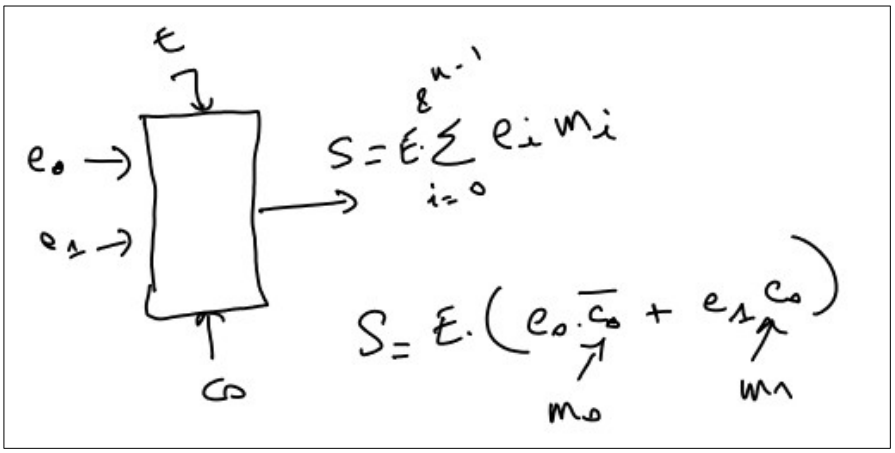
- ***n*** entrées de commande (sélection)
- ***N=2^n^*** entrées de données e_i
- une seule sortie ***S***
- une entrées de validation ***E***

Son principe de fonctionnement repose sur l'idée de sélectionner une entrée e_i parmi les 2^n en se servant des entrées de commandes ($c_{n-1}..c_1c_0$).

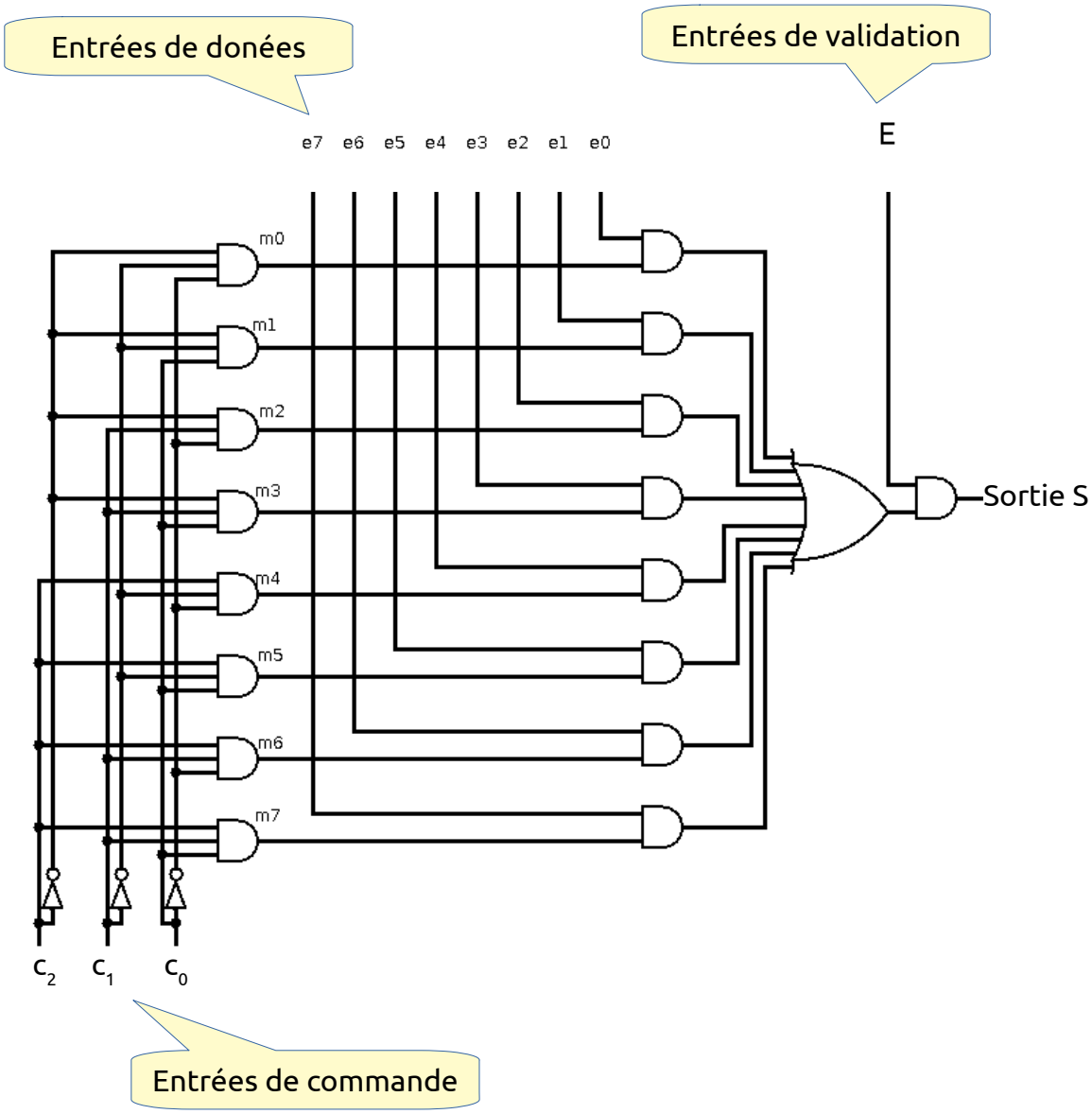


L'équation de la sortie S est données par : $S = E . \sum_{i=0}^{2^{n-1}} e_i . m_i$

Voici un exemple de MUX 2-1 (2 entrées de données et 1 entrées de commande)

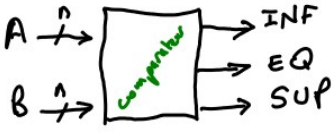


Voici un exemple de MUX 4-2 (4 entrées de données et 2 entrées de commande)



Comparateur

Un comparateur est un circuit logique combinatoire (en général) permettant de comparer entre deux valeurs *A* et *B* (codées, en général, en binaire pure) et possédant généralement 3 sorties *EQ*, *SUP* et *INF* indiquant respectivement si A=B ou A>B ou A<B.



Avant de procéder à la synthèse de ce circuit, il est utile d'étudier les règles (fonctions) de comparaison entre 2 bits ***a*** et ***b***. Nous avons 3 fonctions ***inf***, ***sup*** et ***eq*** qui correspondent respectivement à ***a<b***, ***a>b*** et ***a=b***. Voici leur table de vérité :

a b	Eq = (a=b)	Inf = (a<b)	Sup = (a>b)
0 0	1	0	0
0 1	0	1	0
1 0	0	0	1
1 1	1	0	0
Ce qui donne	$Eq=\overline{a \oplus b}$	$inf=\overline{a}.b$	$sup= a.\overline{b}$

Passant maintenant à la synthèse de notre circuit. Nous pouvons écrire mathématiquement nos nombres comme suit :

- ***A = (a_{n-1} a_{n-2} … a_1 a_0)***
- et ***B = (b_{n-1} b_{n-2} … b_1 b_0)***
- avec a_i et b_i les chiffres des nombres ***A*** et ***B*** pour ***i*** allant de ***0*** à ***n-1***.

A première vue, notre système dispose de **2n** entrées et de **3** sorites. Bien évidement, il est nécessaire de trouver des astuces pour simplifier notre problème et notamment identifier une relation de récurrence.

Pour cela nous constatons 3 choses :

- **Constatation 1** : Pour vérifier si le nombre ***A*** est égale au nombre ***B*** *(fonction **EQ**)* , il suffit de vérifier l'égalité au rang ***n-1***, c'est à dire que (a_{n-1}=b_{n-1}) et l'égalité au rang inférieur (n-2).

  On écrira donc :

  - ***EQ = eq_{n-1} = (a_{n-1}=b_{n-1}) ET eq_{n-2}***
    ce qui donne plus formellement : $EQ=eq_{n-1}=(\overline{a_{n-1} \oplus b_{n-1}}).eq_{n-2}$
  - et au rang i : $eq_i=(\overline{a_i \oplus b_i}).eq_{i-1}$
  - et au rang 0 : $eq_0=(\overline{a_0 \oplus b_0})$ *(je vous rappelle que le non ou exclusif exprime l'égalité !)*
- **Constatation 2** : Pour vérifier si le nombre ***A*** est supérieur au nombre ***B***, il suffit de vérifier si le bit de poids fort de ***A*** (a_{n-1}) est supérieur au bit lui correspondant dans ***B*** (b_{n-1}). 2 possibilités se présentent :
  - Possibilité 1 : Si a_{n-1} est différent de b_{n-1} alors on peut dire que ***A*** est supérieur ou non à ***B***. En effet si $a_{n-1}>b_{n-1}$ alors on peut déduire que ***A>B,*** donc la fonction ***SUP=Sup_{n-1} = (a_{n-1}>b_{n-1}) =*** $(a_{n-1}.\overline{b_{n-1}})$
  - Possibilité 2 : Si $a_{n-1} = b_{n-1}$ alors on ne peut pas dire si A est supérieur (ou inférieur) ou non à B. Il faut vérifier (récursivement) exactement de la même façon les 2 possibilités (1 et 2) pour les bits qui vient juste avant (n-2, n-3 et ainsi de suite). On peut écrire ceci
    - ***SUP=Sup_{n-1} =(a_{n-1}=b_{n-1}) ET sup_{n-2}*** ce qui donne :

      $(\overline{a_{n-1} \oplus b_{n-1}})$. sup_{n-2}
    - *au rang i nous aurons :* $sup_i =$ $(\overline{a_i \oplus b_i})$. sup_{i-1}
    - *au rang 0 :* $sup_0 =$ $(a_0\overline{b_0})$

En résumé : on aura en prenant en compte les 2 possibilités :

pour i allant de n-1 à 1 , on a : ***sup_i =*** $a_i . \overline{b_i} + (\overline{a_i \oplus b_i})$ ***. sup_{i-1}***

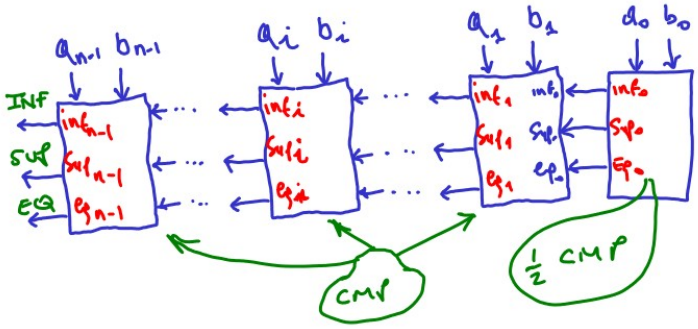
pour i = 0, on a : ***sup_0 =*** $a_0 . \overline{b_0}$

- **Constatation 3** : La fonction INF (A<B) peut être générée comme on a fait pour la fonction SUP. Nous trouverons les équations suivantes :

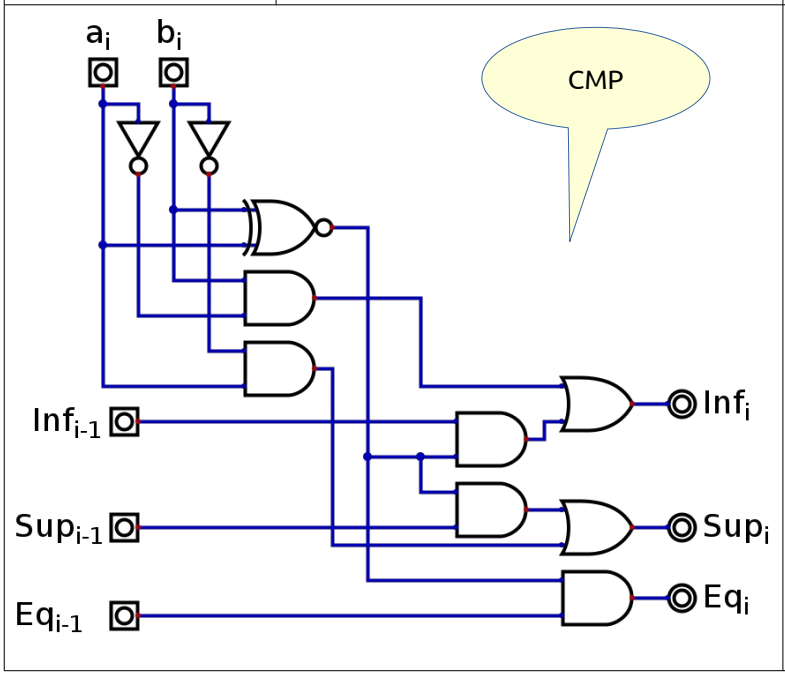
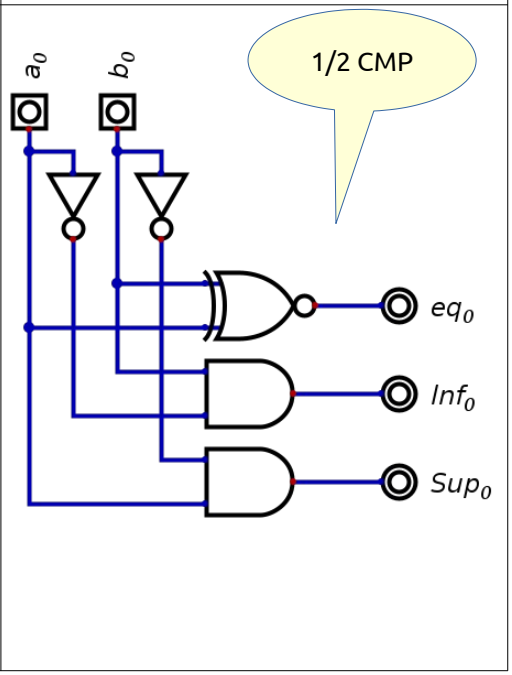
pour i allant de n-1 à 1, on a : ***inf_i =*** $\overline{a_i} . b_i + (\overline{a_i \oplus b_i})$ ***. inf_{i-1}***

pour i = 0, on a : ***inf_0 =*** $\overline{a_0} . b_0$

Au final, nous allons concevoir un demi comparateur pour le première étage (rang 0) et un comparateur complet pour les étages (rangs) i allant de 1 à n-1. Ceci nous donne un montage en cascade comme suit :



Je rappelle les équations :

Fonctions	Étage i (Comparateur complet CMP)	Étage 0 (demi comparateur 1/2 CMP)
A<B : *INF = inf_{n-1}*	*inf_i =* $\overline{a_i} . b_i + (\overline{a_i \oplus b_i})$ *. inf_{i-1}*	*inf_0 =* $\overline{a_0} . b_0$
A>B : *SUP = Sup_{n-1}*	*sup_i =* $a_i . \overline{b_i} + (\overline{a_i \oplus b_i})$ *. sup_{i-1}*	*sup_0 =* $a_0 . \overline{b_0}$
A=B : *EQ= Eq_{n-1}*	$eq_i = (\overline{a_i \oplus b_i}) . eq_{i-1}$	$eq_0 = (\overline{a_0 \oplus b_0})$
		

Génération de fonctions

Il est tout à fait possible de générer des fonction de façon extrêmement simple en se servant de circuits comme le décodeur (DEC) ou le démultiplexeur (DeMUX)

Générer une fonction en se servant d'un décodeur :

Nous savons que toute fonction booléenne peut être écrite sous la forme :

$$f(e_{n-1}, e_{n-2}, \dots, e_0) = \sum_{i=0}^{2^{n-1}} v_i . m_i$$

Nous savons par ailleurs que les sorties d'un décodeur correspondent aux mintermes composés des variables d'entrée (e_{n-1}, e_{n-2}, …, e_0).

En considérant les variables de commandes comme les variables de notre fonction, il suffit de faire un OU Logique entre les sorties si du décodeur qui correspondent aux mintermes pour lesquels la fonction est à « 1 ».

Exemple : $f(x,y,z) = \Sigma(1,2,5,6)$

Nous avons ici 3 variables pour notre fonction ***f***. Nous allons donc nous servir de 3 variables d'entrée du décodeur comme suit : ce qui nous donne le circuit suivant :