POLYTECH NANTES

Spécialité Génie Electrique

Polycopié d'Informatique Industrielle

Partie I : Logique

D. DELFIEU

- Troisième année -

Table des matières

Introduction

Ce cours de logique à pour objectif de vous permettre de comprendre les circuits combinatoires comme les Codeurs, les Décodeurs, les Multiplexeurs, les Démultiplexeurs, les Additionneurs, comment on réalise une soustraction, le Multiplicateur le comparateur. Tous ces circuits participent à la conception d'une Unité Arithmétique et Logique d'un microcontrôleur ou d'un microprocesseur. En logique Séquentielle, on verra la notion de *bit* ainsi qu'une méthode d'analyse des circuits séquentiels ainsi qu'une méthode de synthèse d'automate.

Après une partie sur les systèmes basés, il y aura deux grands chapitres, la logique combinatoire et la logique séquentielle. Comme l'indique la figure 1, en logique combinatoire on établit des relations logiques entre des entrées et des sorties.

FIGURE 1 – Logique combinatoire

Ainsi, l'exemple de la figure 2.1 illustre une serrure électronique qui s'ouvre lorsque les boutons (b_3, b_2, b_1, b_0) sont positionnés sur la combinaison $(1, 1, 1, 0)$.

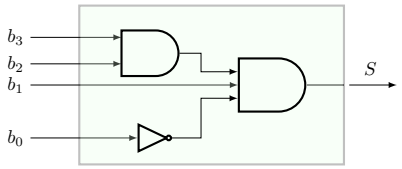


FIGURE 2 – Exemple de logique combinatoire

L'équation de la sortie est alors $S = b_3.b_2.b_1.\overline{b_0}$.

En logique séquentielle, la ou les sorties peuvent re-boucler sur les entrées ! Comme l'indique la figure 3, ici la sortie Q ne peut être définie en effet on aurait la définition autoréférente : $Q = \overline{rst + \overline{set + Q}}$. En effet, définir Q en fonction de Q n'a aucun sens en logique combinatoire !

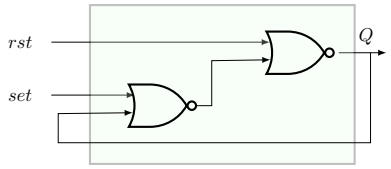


FIGURE 3 – Rebouclage en logique séquentielle

On verra dans le chapitre logique séquentielle une méthode qui permettra d'établir tout de même, des équations qui décrivent ces systèmes et de produire la machine à états qui décrit le comportement d'un système séquentiel.

Chapitre 1

Système de numération

Le coeur d'un processeur, appelé Unité Arithmétique et Logique (UAL), n'effectue en fait que des opérations élémentaires, qui peuvent être soit des opérations arithmétiques soit logiques. Toutes ces opérations sont réalisés en base 2. Nous étudierons dans cette section comment sont représentés les nombres négatifs dans une représentation binaire et comment l'UAL réalise les additions et les soustractions.

1.1 Les systèmes basés

Un nombre N exprimé en base B se représente comme une juxtaposition d'éléments a_i exprimés en base B. Concernant la Partie Entière (PE) d'un nombre :

- $N_B = a_i a_{i-1} ... a_1 a_0$
- a_i exprimé en base B signifie que $a_i \in [0...B-1]$

Pour convertir ce nombre en base 10 on utilise la formule de Horner :

$$N_{10} = \sum_{i=0}^{n} a_i B^i$$

Concernant la Partie Fractionnaire (PF) :

- $N_B = 0, a_{-1} a_{-2} ... a_{-m}$
- $N_{10} = \sum_{i=-m}^{-1} a_i B^i$

1.2 Conversion de bases

On va étudier les méthodes de conversion d'une base à une autre. Il existe la méthode par multiplication et la méthode par division. Donnons l'algorithme de la méthode par soustraction :

1.2.1 Conversion d'un entier : $10 \Rightarrow B$

Algorithm 1: Par Soustraction

Données: N : Entier à convertir en base B
Répéter
  On cherche un couple (a_j, B^i) tel que a_j est le plus grand entier dans $[0, B-1]$ et B^i est le plus grand entier tel que $a_j * B^i < N$;
  On pose a_j au rang i ;
  On réalise $N = N - a_i * B^i$;
Jusqu'à $N < B$;
On depose le dernier reste au rang zéro ;

Exercice : 77_{10} en base 3 ?

Maintenant donnons l'algorithme par division :

Algorithm 2: Par division

Données: N : Entier à convertir en base B
Répéter
  Division Euclidienne de N par B : $N = Q * B + R$;
  On pose R au rang dans l'ordre inverse où il a été obtenu (le premier reste etant le poids faible du résultat) ;
Jusqu'à $(Q == 0)$;
Le dernier quotient est posé au rang le plus fort ;

Exercice : 77_{10} en base 3 ?

1.2.2 Conversion d'une partie fractionnaire : $10 \Rightarrow B$

Algorithm 3: Par Multiplication

Méthode par multiplication
Données: PF : Partie Fractionnaire à convertir en base B
Répéter
  R= $PF * B$;
  PE = PartieEntiere(R) ;
  $PF = R - PE$;
  On pose PE à droite du précédent résultat ;
Jusqu'à $(PF == 0)$ *ou (précision est obtenue) ou (détection séquence infinie)*;

Exemple : $0,45_{10}$ en base 2 ?

1.2.3 Conversion $2^i \Rightarrow 2^j$

Pour convertir de la base 2 vers une base 2^N. On regroupe par paquets de n bits et on convertit à l'intérieur de chaque paquets de n bits
Exemple : $0,01110011_2 \Longrightarrow 0,011100110 \Longrightarrow 0,346_8$
Pour convertir d'une base 2^N vers la base 2, on converti chaque terme a_i en base 2 sur n bits.

On considère par la suite, que la base 2 est le mode de représentation des nombres.

1.3 L'arithmétique des systèmes basés

Une arithmétique en base 2, se pose la question de la représentation des nombres négatifs. Pour représenter les nombres nous avons :

— Signe et valeur absolue
— Complément à deux : CA_2
— Complément à un : CA_1

1.3.1 Signe + VA

7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0+	0000
0−	1000
−1	1001
−2	1010
−3	1011
−4	1100
−5	1101
−6	1110
−7	1111

— Un bit de signe :
  — 1 pour " -"
  — 0 pour " +"
— Deux représentations pour le zéro :
  — 0000 pour 0^+
  — 1000 pour $0-$
— Si l'on fait l'addition d'un nombre positif et d'un nombre négatif le résultat est incorrect.

1.3.2 Complément Restreint (base B) - Complément à 1 (Base 2)

7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0+	0000
0−	1111
−1	1110
−2	1101
−3	1100
−4	1011
−5	1010
−6	1001
−7	1000

Soit n le nombre de bits de représentation des nombres, pour une base B on a :

$$CA_{B-1}(N) = B^n - N - 1$$

Base 2 :

$$CA_1(N) = 2^n - N - 1$$

S'obtient facilement par inversion binaire du positif.

Ce mode de représentation produit des erreurs selon les cas :

Démonstration. L'usage de nombres négatifs contient deux cas de figure :
Premier cas : R=A-B
$R = A - B = A + CA_1(B) = A + 2^n - B - 1 = A - B + 2^n - 1$
Posons $K = A - B$
Si $K > 0$:

$R = K + 2^n - 1 = K - 1$: **Erreur de +1**
Si $K < 0$:
$R = 2^n - |K| - 1 = CA_1(|K|)$: **Pas d'erreur**

Second cas : R=-A-B
$R = -A - B = CA_1(A) + CA_1(B) = 2^n + (2^n - A - B - 1) - 1$
$= 2^n + CA_1(A + B) - 1$
$= CA_1(A + B) - 1$: **Erreur de +1**

□

En utilisant la formule CA_1 précédente :
— Sur 4 bits, trouver la représentation en base 2 de -5 en complément restreint ?
— Sur 4 bits, trouver la représentation en base 2 de -1 en complément restreint ?

1.3.3 Complément Vrai (base B) - Complément à 2 (Base 2)

7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
−1	1111
−2	1110
−3	1101
−4	1100
−5	1011
−6	1010
−7	1001
−8	1000

Soit n le nombre de bits de représentation des nombres, en base B on a :

$$CA_B(N) = B^n - N$$

Base 2 :

$$CA_2(N) = 2^n - N$$

— S'obtient par inversion binaire du positif puis en rajoutant 1.
— "De la droite vers la gauche, reproduire les zéros, au premier "un" rencontré, le reproduire puis inverser tous les bits à sa gauche"

En CA_2 il n'y a jamais d'erreur :

Démonstration. L'usage de nombres négatifs contient deux cas de figure :
Premier cas : R=A-B
$R = A - B = A + CA_2(B) = A + 2^n - B = A - B + 2^n - 1$
Posons $K = A - B$
Si $K > 0$:
$R = K + 2^n = K$: **Pas d'erreur**
Si $K < 0$:
$R = 2^n - |K| = CA_2(|K|)$: **Pas d'erreur**

Second cas : R=-A-B
$R = -A - B = CA_2(A) + CA_2(B) = 2^n - A + 2^n - B = 2^n - A - B + 2^n$
Posons $K = A + B$
$R = 2^n - K + 2^n = CA_2(K) + 2^n$
Or on sait que $X + 2^n = X$ **sur n bits**
D'ou $R = CA_2(K)$ **Pas d'erreur**

□

En utilisant la formule précédente :
— Sur 4 bits, trouver la représentation en base 2 de -3 en complément vrai ?
— Sur 4 bits, trouver la représentation en base 2 de -7 en complément vrai ?

1.3.4 Bilan : CA_1 vs CA_2

Si l'on fait un bilan des deux façons de coder les nombres négatifs
— CA_1
  — plage des positifs : $2^{n-1} - 1$ ... 0^+
  — plage des négatifs : 0^- ... $-(2^{n-1} - 1)$
— CA_2 :
  — plage des positifs : $2^{n-1} - 1$ ... 0
  — plage des négatifs : -1 ... $-(2^{n-1})$
— Posons n=8 :

— Plage positif : 0 ... +127
— Plage négatifs : -1 ... -128
— Représentation négative de 100 ? $100 = 01100100_2$
  — $CA_1(100) = 2^8 - 100_{10} - 1 = 256_{10} - 100_{10} - 1 = 155_{10} = 10011011_2$
  10011011 s'obtient facilement par complément binaire de 01100100
  — $CA_2(100) = 2^8 - 100_{10} = 256_{10} - 100_{10} = 156_{10} = 10011011_2$
  10011011 s'obtient moins facilement par complément binaire et addition de +1 :
  $01100100 \rightarrow^{Comp.Bin.} 10011011 \rightarrow^{+1} 10011100$

Complément à un :
— deux zéros,
— correction dans certains cas,
— facile à implémenter,
— autant de positif que de négatifs.

Complément à deux :
— un seul zéro,
— pas de correction,
— plus difficile à implémenter,
— un nombre négatif en plus.

Si Au final c'est le CA_2 qui est considéré comme le meilleur compromis. Pratiquement quasiment 100% des processeurs utilisent le CA_2

1.3.5 Débordement de capacité

Un débordement de capacité se produit lors de l'addition de deux nombres positifs ou lors de l'addition de deux nombres négatifs. Il se produit alors un changement de signe du résultat. Ce phénomène est simplement détecté quand la retenue entrante dans le bit de signe est différente de la retenue sortante. Cela est testé par un ou exclusif.
Exemple sur 8 bits :

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

Il y a débordement ! Les retenues entrantes et sortantes dans l'addition du bits de signe sont différentes : $r_7 \oplus r_6 = 0 \oplus 1 = 1$

1.4 Codes non pondérés

Les codages basés sont des codes dit pondérés. Un code est dit pondéré si la position de chaque symbole dans chaque mot correspond à un poids fixé. La formule de Horner, indique qu'en multipliant chaque symbole par son poids et en additionnant le tout, on obtient la conversion en décimal.

Des codes non pondérés sont des codes pour lesquels on ne peut repérer de poids. Ils sont définis par des tables de correspondances ou par le biais de symétrie et/ou de propriétés logiques ou arithmétiques.

- Code Pondéré : binaire, octal, hexadécimal, Décimal Codé Binaire
  Le "Décimal Codé Binaire" (BCD) est un code dans lequel chaque chiffre de la représentation décimale est codé sur un groupe de 4 bits.
  - Exemple : 1789 se code 0001 0111 1000 1001
  - Avantage : Affichage décimal grandement facilité
  - Inconvénient : Code redondant 6 combinaisons sur 16 ne sont pas utilisées

  Remarque : 1789 prend 13 bits en BCD, seulement 11 en binaire naturel.
- Code Non Pondéré : *Code excess* 3
  L'Excess 3, permet une transcription rapide en décimal. Il permet de plus une précision infinie en arithmétique (limité par les temps de calcul) au dépend d'une représentation qui gaspille des combinaisons.
  Il apporte par rapport au BCD une rapidité par rapport à la soustraction.
- Code Non Pondéré : Code de Gray

1.4.1 Code Excess 3

Le code excess 3 ou "code plus 3" (XS_3) appelé aussi "code STIBIZ" du nom de son inventeur est un code non pondéré issu du $DCBN$ auquel on ajoute systématiquement 3 à chaque chiffre. Donc un chiffre est représenté sur 4 bits auquel on rajoute 3. Les retenues ont donc un poids de 16.

Définition 1.4.1. *Additions de Nombres positifs Soit deux nombres positifs A et B exprimés en XS_3, et $R = A + B$.*

- *Si $R < 16_{10}$ alors il faut retrancher 3 à R.*
- *Si $R \geq 16_{10}$ alors il faut ajouter 3.*

Définition 1.4.2. *Nombres négatifs La particularité essentielle est que la représentation négative d'un nombre N se réalise par le Complément à 9 du nombre N. Etant donné deux nombres positifs A et B, pour $A - B$, il y aura des corrections à faire dans le cas ou $A > B$. Et une correction aura lieu aussi pour $-A - B$.*

Exemple :

Décimal	BCD	Excess-3
5	0101	1000
7	0111	1010

Addition binaire :

$$\begin{array}{cccccc} & 1 & 0 & 0 & 0 \\ + & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 \end{array}$$

$$\begin{array}{ccccc} 1 & 0 & 0 & 1 & 0 \\ + & 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 1 \end{array}$$

On a la retenue, puis le groupe de 4 bits qui fait 5 : On a donc 15 en XS_3 qui représente bien la valeur 12.

Exemples d'additions Notons N_{XS_3} un nombre N exprimé en "DCBN XS_3"

— $2_{10} + 5_{10} = 5_{XS_3} + 8_{XS_3} = 13_{XS_3} - 3 = 10_{XS_3}$(correction de -3)

— $8_{10} + 5_{10} = 11_{XS_3} + 8_{XS_3} = 19_{XS_3} - 3 = 13_{XS_3}$ (correction de +3)

— Addition sur plusieurs digits : 21+37

Donc l'addition donne :

Nombres	XS_3
2	5
1	4
3	6
7	A

$$\begin{array}{cccc} & 5 & 4 \\ + & 6 & A \\ \hline = & B & E \\ + & -3 & -3 \\ \hline = & 8 & B \end{array}$$

Les soustractions en Codage Excess 3

Les nombres négatifs sont représentés en complément à 9.

$$CA_9(x) = 9 - x$$

Pour réaliser $A_{XS3} - B_{XS3}$:

— Calcul de $CA_9(B)$
On additionne : $A_{XS3} + CA_9(B)$
Si le résultat sur 4 bits est >9 on ajoute 3.

— Rappel :

Exemples de soustraction 27-13

— $5A - CA(13)$
Détaillons : $CA_9(13) = CA_9(1)CA_9(3) = 86 =_{XS3} B9$

$$5A + B9 = 11 + CA_1(8) = 8 + CA_1(1000)$$

1011 (8)
+0111 (-5)
1 $\overline{0010}$ (+2)
+0001
+0011
$DCBN$

Exercices

Correction exercice $42-12$ Second algorithme : Le soustracteur 12 est exprimé en XS_3 et devient 45. Ld résultat est en $DCBN$. La théorie nous dit qu'il y aura une erreur de $+1$ car le résultat attendu $+30$ de la soustraction est positif.

$$42-12 = 75 + CA_1(45) = 01110101 + CA_1(01000101)$$

Détaillons :
01110101 (75)
$+\underline{10111010}$ $(CA_1(45))$
$1\ \overline{00101111}$ (1 2 F)
$+\underline{00000001}$
$1\ \overline{00110000}(DCBN)$

Correction exercice $21-37$ Premier algorithme : Le soustracteur 37 n'est pas exprimé en XS_3. Le résultat ser en XS_3. La théorie nous dit qu'il n'y aura pas d'erreur car le résultat attendu -16 est négatif.

$$21-37 = 54 + CA_9(37) = 75 + CA_1(00110111)$$

Détaillons :
01010100 (54)
$+\underline{11001000}$ $(CA_1(37))$
$1\ \overline{01100010}$ (1 6 2)
$+\underline{00000001}$
$1\ \overline{01100011}$
1 6 3 (DCBN-XS_3)

1.4.2 Codage de Gray

Ce code est construit de facon récurrente à l'aide de symétries. Il a comme propriétées d'être réfléchi, circulaire, et la distance de hamming entre deux codes est de 1.

La distance de Hamming entre deux nombre se définit comme le nombre de bite qui diffèrent.

Construisons un code de gray à trois variables :

On réalise d'abord une symétrie :

0
1

1
0

On remplit la colonne 2
par moitié de zéro et de un

0 0
0 1
1 1
1 0

On réalise encore une symétrie :

0 0
0 1
1 1
1 0
---
1 0
1 1
0 1
0 0

On remplit la colonne 3
par moitié de zéro et de un

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Chapitre 2

Logique Combinatoire

La logique permet de définir des fonctions logiques qui s'expriment sur des variables logiques. Une variable logique est une variable qui prend ses valeurs dans $[0, 1]$. Ce domaine peut s'interpréter comme $[Faux, Vrai]$ et également comme $[0v, 5v]$. L'algèbre de Boole (Georges Boole 1815 - 1864) est une algèbre binaire n'acceptant que deux valeurs numériques : 0 et 1. Cette algèbre est définie par la donnée d'un ensemble non vide muni de 3 lois de composition interne : ET, OU, NON satisfaisant à certain nombre de propriétés (commutativité, distributivité...).

Un système combinatoire se définit par les fonctions logiques qui correspondent aux sorties du systèmes :

FIGURE 2.1 – Exemple de logique combinatoire

Définition 2.0.1. *Fonction logique Une fonction logique F_i est une fonction de n variables qui est définie par sa valeur dans $[0, 1]$ pour les 2^n combinaisons de ces variables d'entrées. Ces 2^n définitions constituent ce que l'on appelle la table de vérité d'une fonction.*

Problématique du cours

La problématique de cette section est de traduire un problème en équations logiques, puis de simplifier ces équations et enfin de les implémenter ces équations à l'aide de portes et/ou de composants logiques.

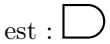
Un problème de type combinatoire - c.a.d . qui ne requiert pas le concept de mémoire - peut généralement s'exprimer dans le cas d'une algèbre :

$$\mathcal{A} = (\ .\ ,\ +\ ,\ ^-, (0,\ 1))$$

2.1 Opérateurs logique élémentaires

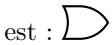
2.1.1 ET

Noté "." on trouve aussi $\wedge$, & :
— il a un élément neutre noté 1 : $1.x = x$
— un élément absorbant noté 0 : $0.x = 0$
— il est commutatif $x.y = y.x$
— il est associatif $x.(y.z) = (x.y).z$
— il est distributif sur le OU : $x.(y + z) = xy + xz$
— il est idempotent : $x = x.x.....x$

Son symbole électronique est : 

2.1.2 OU

Noté "+" on trouve aussi $\vee$:
— il a un élément neutre noté 0 : $0 + x = x$
— un élément absorbant noté 1 : $1 + x = 1$
— il est commutatif $x + y = y + x$
— il est associatif $x + (y + z) = (x + y) + z$
— il est distributif sur le ET : $x + (y.z) = (x + y).(x + z)$
— il est idempotent : $x = x + x.... + x$

Son symbole électronique est : 

2.1.3 NON

Noté $^-$ on trouve aussi $\neg$, c'est une fonction unaire.
— il est involutif $\bar{\bar{x}} = x$
— $x + \overline{x} = 1$
— $x.\overline{x} = 0$
— théorème de Morgan (Augustus De Morgan 1806-1871)
$\overline{x \vee y \vee z \vee ...} = \overline{x} \wedge \overline{y} \wedge \overline{z} \wedge ...$
$\overline{x \wedge y \wedge z \wedge ...} = \overline{x} \vee \overline{y} \vee \overline{z} \vee ...$

Son symbole électronique est : 

2.1.4 NAND

Noté $\overline{a.b}$
— il n'est pas associatif : $\overline{\overline{a.b}.c} \neq \overline{a.\overline{b.c}}$
— $NAND$ est un opérateur logique COMPLET :
— On peux réaliser le NOT avec un NAND : $\overline{a} = \overline{a.1}$
— On peux réaliser le OU avec un NAND : $a + b = \overline{\overline{a.1}.\overline{b,1}}$
— On peux réaliser le ET avec un NAND : $a.b = \overline{\overline{a.b}.1}$

Son symbole électronique est : 

2.1.5 NOR

Noté $\overline{a + b}$
— il n'est pas associatif : $\overline{\overline{a + b} + c} \neq \overline{a + \overline{b + c}}$

— NOR est un opérateur logique COMPLET :
  — On peux réaliser le NOT avec un NOR : $\overline{a} = \overline{a+0}$
  — On peux réaliser le ET avec un NOR : $a.b = \overline{\overline{a+0}+\overline{b+0}}$
  — On peux réaliser le OU avec un NOR : $a+b = \overline{\overline{a+b}+0}$

Son symbole électronique est : 

2.1.6 XOR : $\oplus$

Noté $a \oplus b = a.\overline{b} + \overline{a}.b$
— il est associatif : $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
— XOR n'est pas un opérateur logique COMPLET :
  On ne peux réaliser que le NOT avec un XOR : $\overline{a} = a \oplus 1$

Son symbole électronique est : 

2.1.7 IMPLIQUE : $\Rightarrow$

Noté $a \Rightarrow b = \overline{a} + b$

a	b	$a \Rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

2.1.8 EQUIVALENCE : $\odot$

Noté $a \Leftrightarrow b = a.b + \overline{a}.\overline{b} = \overline{a \oplus b} = a \odot b$

a	b	$a \Leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

Son symbole est : 

2.1.9 Exemples de simplification algébrique d'expressions logiques

Simplifier une fonction logique permettra d'optimiser sa consommation électrique.
— $F(x,y) = x + \bar{x}.\bar{y} = x + \bar{y}$
— $F(x,y) = x + \bar{x}.y = x + y$
— $F(x,y) = x + x.y = x$
— $f(a,b,c) = \bar{a}\bar{b}.c + \bar{a}.b.c + a(\bar{b}\bar{c} + b\bar{c} + \bar{b}c + bc) = a + c$

2.2 Simplification de fonction logique

On va dans cette section présenter comment réduire la complexité d'un fonction logique. Réduire cette complexité, c'est permettre lors de son implémentation électronique, de diminuer le coût et la consommation électrique, mais aussi augmenter la vitesse de traitement du circuit électronique qui implémente la fonction.

Une fonction logique peut être représentée par une expression algébrique, une table de vérité ou une table de Karnaugh (1924-2022).

Un problème en logique est qu'il existe une infinité de formes équivalente à f, par contre on pourra toujours ramener une fonction f en une forme canonique disjonctive (les opérateurs de plus haut niveau sont des OU) ou une forme canonique conjonctive (les opérateurs de plus haut niveau sont des ET) . Une forme canonique est une une forme unique.

Théorème de décomposition de Shannon (1916-2001)

Ce théorème donne une méthode pour représenter une fonction sous une forme unique conjonctive ou disjonctive.

Définition 2.2.1. *Shannon : Toute fonction logique peut se décomposer par rapport à l'une de ces variables sous la forme d'une somme de deux produits logiques tel que :*

$$F(x, y, ...z) = x.F(1, y, ...z) + \bar{x}.F(0, y, ...z)$$

Toute fonction logique peut se décomposer par rapport à l'une de ces variables sous la forme d'un produit de deux sommes logiques tel que :

$$F(x, y, ...z) = [\bar{x} + F(1, y, ...z)].[x + F(0, y, ...z)]$$

Démonstration Pour la première assertion posons x=1, on a :

$$\begin{aligned} F(1, y, ...z) &= 1.F(1, y, ...z) + \bar{1}.F(0, y, ...z) \\ &= 1.F(1, y, ...z) + 0.F(0, y, ...z) \\ &= F(1, y, ...z) \text{ ce qui est VRAI} \end{aligned}$$

Posons x=0, on a :

$$\begin{aligned} F(0, y, ...z) &= 0.F(1, y, ...z) + \bar{0}.F(0, y, ...z) \\ &= 1.F(0, y, ...z) \\ &= F(0, y, ...z) \text{ ce qui est VRAI} \end{aligned}$$

Pour toutes les valeurs de x, la première assertion est valide. On fera un raisonnement similaire pour la seconde assertion.

Forme disjonctive de Shannon Soit la fonction F : définie par $F(x, y) = x + \bar{x}.\bar{y}$

x	y	$f(x, y)$
0	0	1
0	1	0
1	0	1
1	1	0

$$\begin{aligned} F(x, y) &= x.F(1, y) + \bar{x}.F(0, y) \\ &= x.[y.F(1, 1) + \bar{y}.F(1, 0)] + \bar{x}.[y.F(0, 1) + \bar{y}.F(0, 0)] \\ &= x.y.1 + x.\bar{y}.1 + \bar{x}.y.0 + \bar{x}.\bar{y}.1 \\ &= x.y + x.\bar{y} + \bar{x}.\bar{y} \end{aligned}$$

Lorsque $F(.,.) = 0$, il y a une absorption du terme.

Définition 2.2.2. *Remarques Appliquer le théorème de Shanon pour la forme disjonctive revient, in fine, à la lecture de la table de vérité sur les "uns".*

Forme conjonctive de Shannon Soit la fonction F : définie par $F(x, y) = x + \bar{x}.\bar{y}$

x	y	$f(x, y)$
0	0	1
0	1	0
1	0	1
1	1	1

$$\begin{aligned} F(x,y) &= [\bar{x} + F(1,y)].[x + F(0,y)] \\ &= [\bar{x} + (\bar{y} + F(1,1))(y + F(1,0)][x + (\bar{y} + F(0,1))(y + F(0,0))] \\ &= [\bar{x} + (\bar{y} + 1)(y + 1)][x + (\bar{y} + 0)(y + 1)] \\ &= [\bar{x} + 1] \ \ [x + \bar{y}] \\ &= 1 \ . \ [x + \bar{y}] \end{aligned}$$

Lorsque $F(.,.) = 1$, il y a une absorption du terme.

Définition 2.2.3. *Remarques Appliquer le théorème de Shanon pour la forme conjonctive revient, in fine, à la lecture de la table de vérité sur les "zeros", en inversant termes et opérateurs.*

2.2.1 Lecture d'une table de vérité

les deux formes du théorème de Shannon, nous donnent deux méthodes de lecture des tables de vérité.

Table de vérité et Shannon en forme disjonctive

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

- Lecture sur les "uns".
- "zéro" est absorbant et élimine les autres lignes.
- Les termes sont associés en conjonction, les lignes sont en disjonction,
- $f(a, b, c) = \bar{a}\bar{b}.c + \bar{a}.b.c + a.\bar{b}\bar{c} + a\bar{b}c + a.b\bar{c} + a.bc$

Table de vérité et Shannon en forme Conjonctive

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

— Lecture sur les "zéros".
— "un" est absorbant et élimine les autres lignes.
— Les termes inversés sont associés en disjonction, les lignes sont en conjonction,
— $f(a,b,c) = (a+b+c).(a+\bar{b}+c)$

Table de vérité

Une table de vérité sert à exprimer exhaustivement un cahier des charges. Elle est composée de deux parties : les combinaisons et la valeur de vérité de la fonction. La Partie combinaison on exhibe l'ensemble 2^n combinaisons correspondant aux n variables.

Si une combinaison est interdite ou si la valeur de la fonction est indifférente pour cette combinaison :
Alors la valeur de vérité de la fonction reçoit $*$. La valeur de vérité de la fonction est exprimée sur $0, 1, *$. $*$ signifie que la valeur vaux un ou zéro.

Il est à noter qu'il est périlleux d'effectuer des simplifications dans une table de vérité.

Karnaugh

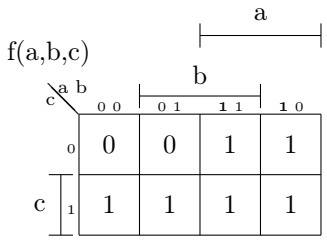
Construction d'une table de Karnaugh pour une fonction à n variables :
— Partager l'ensemble des variables en 2 sous-ensembles s_1 de dimension n_1 et s_2 de dimension n_2 avec $n_1 + n_2 = n$.
— Ecrire un tableau avec 2^{n_1} lignes et 2^{n_2} colonnes.
— Construire un codage de Gray à n_1 et à n_2 variables, que l'on placera dans la première colonne et dans la première ligne du tableau.
— A partir de la table de vérité, reporter la (les) valeur(s) de vérité de(s) la fonction(s) à partir des combinaisons des variables d'entrée du tableau.

Table de vérité et table de Karnaugh.

$$f(a,b,c) = \bar{a}\bar{b}.c + \bar{a}.b.c + a(\bar{b}\bar{c} + b\bar{c} + \bar{b}c + bc)$$

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

Sous-ensembles : $s_1 = c, s_2 = (a,b)$



Règles de simplification

— Dans une table de Karnaugh on regroupe deux paquets de 2^n "uns" pour former un paquet de 2^{n+1} variables.

— Le principe de regroupement de 2 paquets X est basé sur le théorème suivant :

$$x_1....x_i....x_n + x_1....\bar{x_i}...x_n = x_1...(x_i + \bar{x_i})...x_n = x_1...x_{i-1}.x_{i+1}...x_n$$

$$X.\bar{x} + X.x = X$$

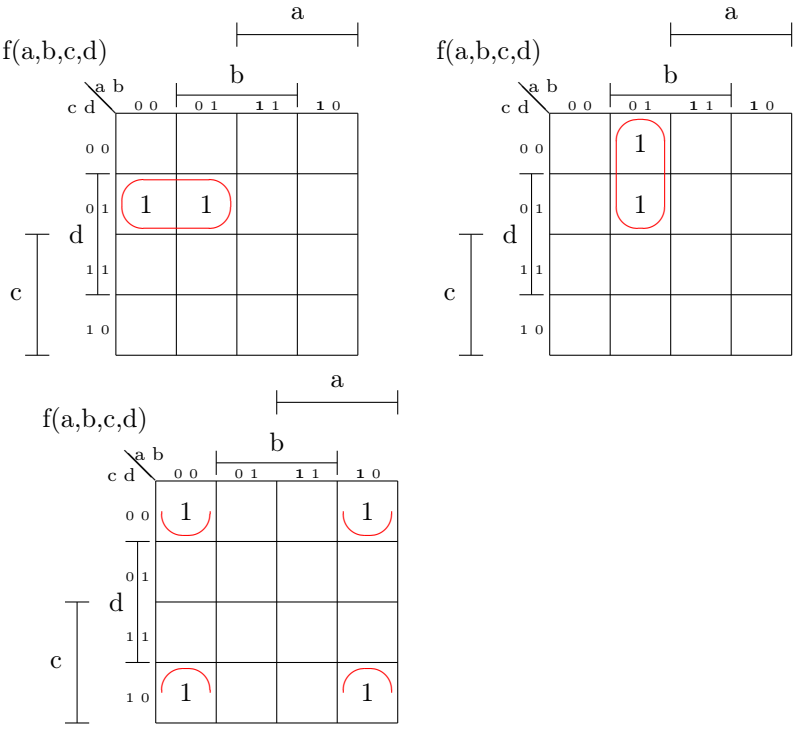
— Ainsi un regroupement de 2^n variables représente n applications SUCCESSIVES de ce théorème.

D'un point de vue graphique les regroupements peuvent se faire suivant les critères suivants :

— 2 cases adjacentes : le Code de Gray assure que 2 codes adjacents ont une distance de Hamming de 1 ;
— un axe de symétrie partage la table de Karnaugh en deux parties égales ;
— d'autres axes de symétrie peuvent partager en 2 une sous-table de Karnaugh issue elle-même d'un axe de symétrie ;
— 2 blocs de 2^n cases peuvent être regroupés si ils présentent entre eux d'un axe de symétrie ;
— le Code de Gray étant circulaire, les colonnes et les lignes extrêmes sont adjacentes.

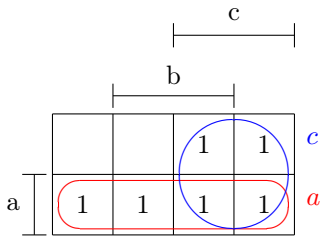
Définition 2.2.4. *Règles de simplification Pour simplifier une table de Karnaugh , on commencera par les "uns" qui n'ont qu'une seule façon de se regrouper puis on essaiera de faire les plus grands regroupements possibles*

Exemples de regroupement de cases adjacentes :

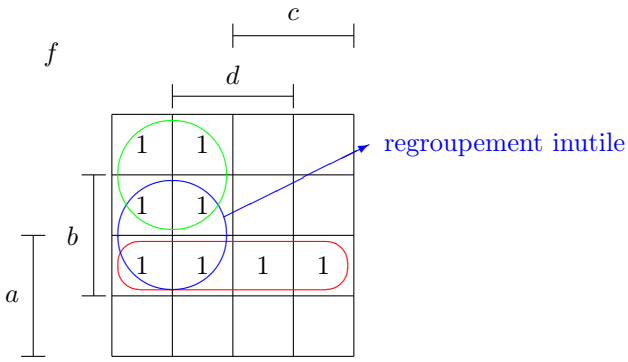


Regroupements de blocs de 2^n cases

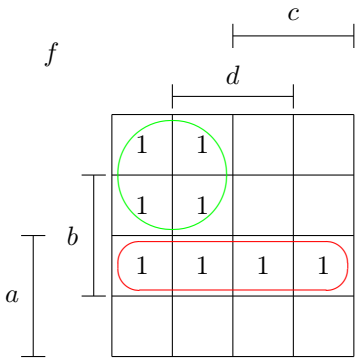
$f(a,b,c) = a + c$



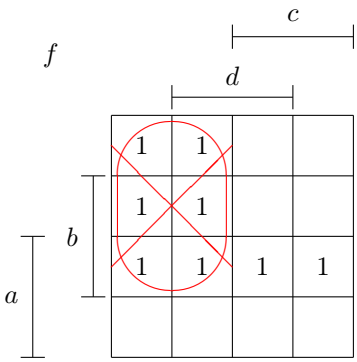
$f(a,b,c,d) = ab + \bar{c}b + \bar{c}\bar{a}$



$f(a,b,c,d) = ab + \bar{c}\bar{a}$

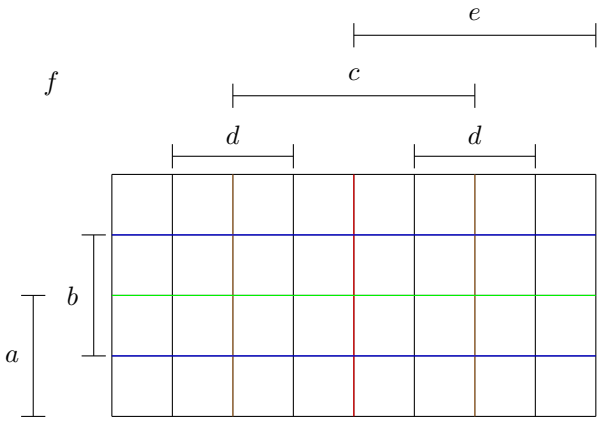


Erreurs classiques !

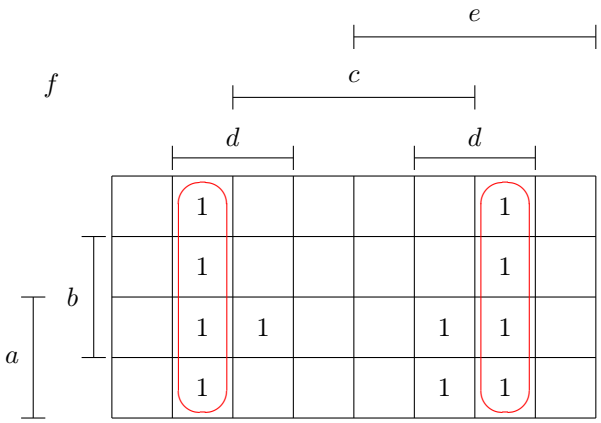


Remarque :
Ce regroupement de six est interdit. Six n'est pas une puissance de 2.

Axes de symétrie Voici quelques axes de symétries possibles : $f(a,b,c,d,e) = \bar{c}d$

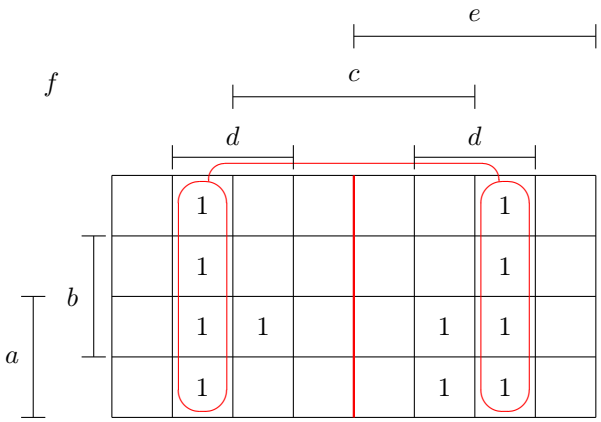


$f(a,b,c,d,e) = \bar{c}d\bar{e} + \bar{c}de$

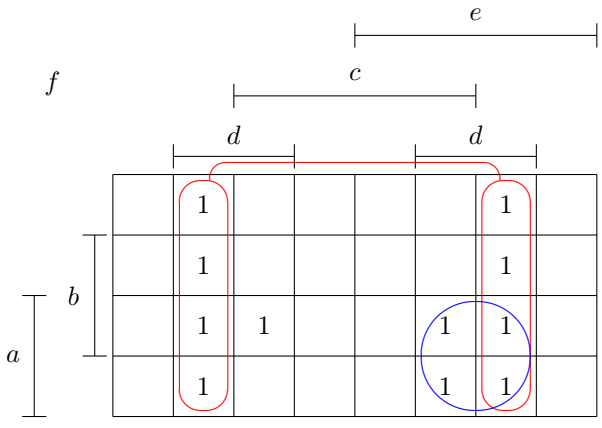


Existe-t-il un axe de symétrie ?

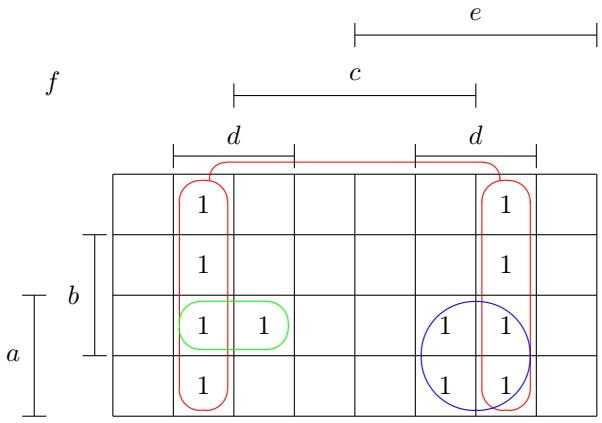
$f(a,b,c,d,e) = \bar{c}d$



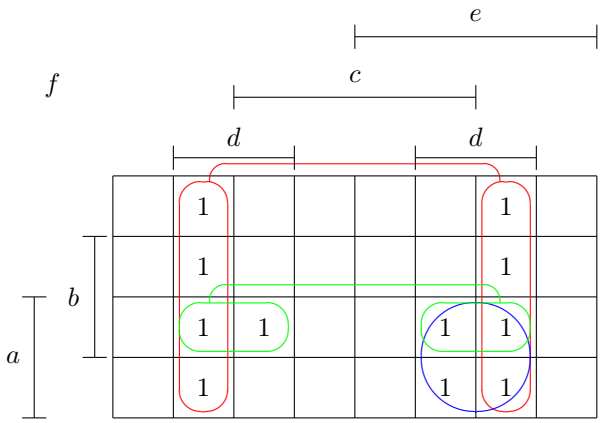
$f(a,b,c,d,e) = \bar{c}d + ade$



$f(a,b,c,d,e) = \bar{c}d + ade + abd\bar{e}$



$f(a,b,c,d,e) = \bar{c}d + ade + abd$



Cas des fonctions incomplètement définies

Lorsque l'on a des combinaisons interdites, impossibles ou indifférentes alors on considère que ces combinaisons sont disponibles à la simplification et sont notées $*$.

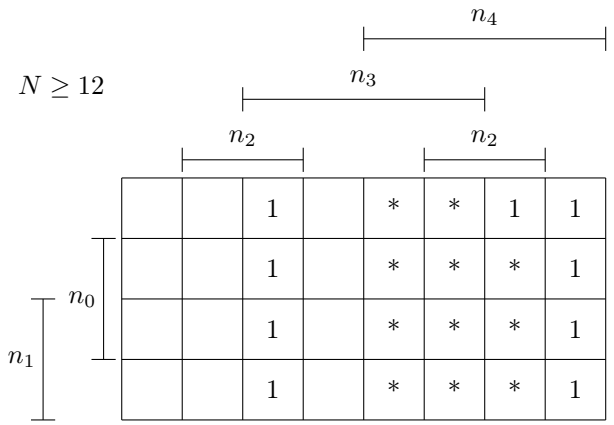
Remarque :
Dans le cas ou la combinaison qualifiée d'impossible dans le cahier des charges se produit

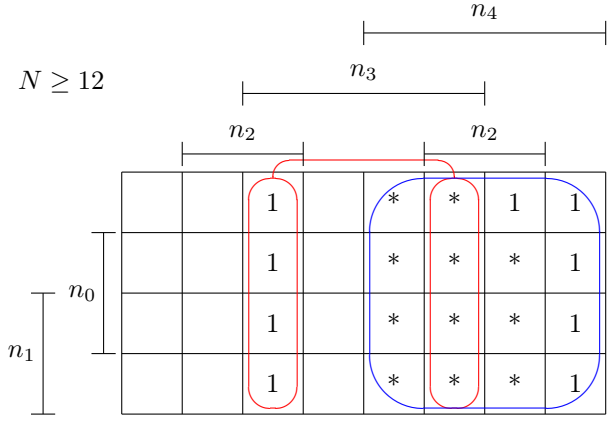
quand même (perturbation électromagnétique), cela provoquera un aléa de fonctionnement. La fréquence et la dangerosité de cet évènement, peut nous amener alors à revoir le cahier des charges et la réalisation du circuit électronique.

Soit N un chiffre décimal représentant une note sur 20 traduit en binaire. Donner la fonction logique qui permet de tester si $N \geq 12$.

La table de vérité de $N \geq 12$:

N	n_4	n_3	n_2	n_1	n_0	f
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	1	0	0
3	0	0	0	1	1	0
4	0	0	1	0	0	0
5	0	0	1	0	1	0
6	0	0	1	1	0	0
7	0	0	1	1	1	0
8	0	1	0	0	0	0
9	0	1	0	0	1	0
10	0	1	0	1	0	0
11	0	1	0	1	1	0
12	0	1	1	0	0	1
13	0	1	1	0	1	1
14	0	1	1	1	0	1
15	0	1	1	1	1	1
16	1	0	0	0	0	1
17	1	0	0	0	1	1
18	1	0	0	1	0	1
19	1	0	0	1	1	1
20	1	0	1	0	0	1
21	1	0	1	0	1	*
...	...	...	...	...	...	*
31	1	1	1	1	1	*





$$N \geq 12 = n_4 + n_2.n_3$$

2.2.2 Simplification d'expressions algébriques

Il y a une infinité de théorèmes ou de règles. Je propose ici un sous-ensemble de 3 théorèmes complètement arbitraire. La règle de remplacement est la plus étrange et j'en déconseille l'usage car elle peut être couteuse en temps. Elle utile pour démontrer la règle appelée "Simplification". Par contre le sous-ensemble, Simplification, Absorption et Consensus est suffisant pour simplifier toute fonction logique.

Remplacement

$$E + F = E + G \iff F \leq G \leq E + F$$

Exemple :

$$E(x_1, x_2) = x_1$$
$$F(x_1, x_2) = \bar{x}_1.x_2 \; G(x_1, x_2) = x_2$$

x_1	x_2	E	F	G	E+F
0	0	0	0	0	0
0	1	0	1	1	1
1	0	1	0	0	1
1	1	1	0	1	1

D'où le nouveau théorème :

$$x_1 + \bar{x}_1.x_2 = x_1 + x_2$$

Simplification, Absorption et Consensus

Simplification

$$E + \bar{E}.F = E + F$$

Régle duale :

$$E.(\bar{E} + F) = E.F$$

Absorption

$$E + EF = E$$

Régle duale :

$$E.(E + F) = E$$

Consensus

$$E.F + \bar{E}.G + FG = E.F + \bar{E}.G$$

Régle duale :

$$(E + F)(\bar{E} + G).(F + G) = (E + F)(\bar{E} + G)$$

Propriétées

$$a.b = x.y \Rightarrow a.b.u = x.y.u$$
$$a + b = x + y \Rightarrow a + b + u = x + y + u$$

PAR CONTRE :

$$x + a = x + b \not\Rightarrow a = b$$
$$a.x = b.x \not\Rightarrow a = b$$

2.3 Les circuits Combinatoires

Les circuits que nous allons aborder sont présents dans les microcontrôleur ou dans les carte mère. Par exemple, les codeurs ou décodeurs permettent d'accéder sélectivement d'accéder à tel ou tel registre, les multiplexeurs permettent de sérialiser ou de de-sérialiser l'information sortant ou entrant dans un bus de données.

— Les transcodeurs

1. Les codeurs
2. Les décodeurs

— Les circuits d'aiguillages

1. Les multiplexeurs
2. Les démultiplexeurs

2.3.1 Circuits de transcodage

Codeur

Ce sont les circuits qui transforment une information de 2^n bits vers n bits.

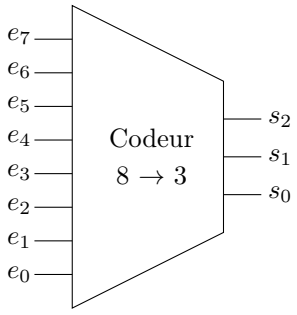


FIGURE 2.2 – Codeur 8 entrées vers 3 sorties

Si une entrée e_i est active alors le nombre i sera représenté en binaire sur les sorties s_2, s_1, s_0

Equation des sorties d'un codeur non prioritaire 8 vers 3 :

$$\begin{aligned} s_0 &= e_1 + e_3 + e_5 + e_7(impair) \\ s_1 &= e_2 + e_3 + e_6 + e_7 \\ s_2 &= e_4 + e_5 + e_6 + e_7 \end{aligned}$$

Exemple de codeur prioritaire : 74148 ce schéma est tiré du "datasheet" du composant :

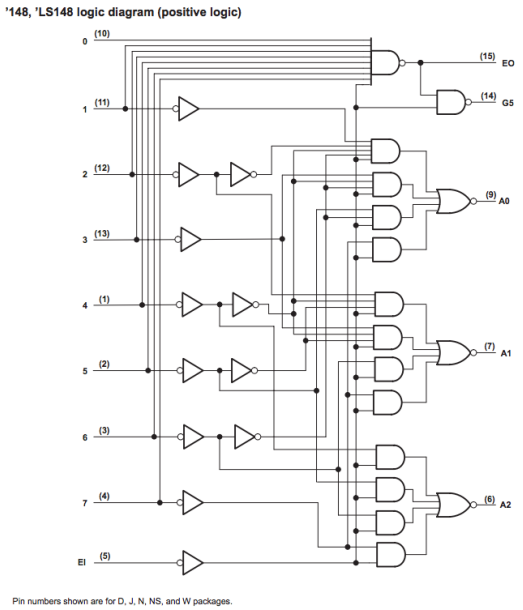


FIGURE 2.3 – Schéma du 74148

Dans la figure 2.3.1 on peu voir noter les entrées 1 à 7 les trois sorties A_2, A_1, A_0. On note une entrée supplémentaire Ei qui si elle est à un désactive toute les sorties.

Table de vérité du 74148 La table de vérité donne le niveau d'activation des entrées et des sorties et les pattes annexes.

	Inputs								Outputs				
$\overline{E_I}$	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{A_2}$	$\overline{A_1}$	$\overline{A_0}$	$\overline{GS}$	E_0
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

E_I (Enable Input) : Active le fonctionnement du codeur.

Rôle de E_O et GS On peut aussi examiner dans cette table le rôles des pattes E_O et GS. E_O (Enable Output) est activé si au moins une entrée a été sélectionnée. Tandis que GS (Gate Select) est activé lorsque soit aucune entrée n'a été sélectionnée soit E_I n'a pas été activé.

Codeur 74148 : Sélection d'entrée Considérons dans la représentation simplifié d'un codeur que l'entrée 6 est active :

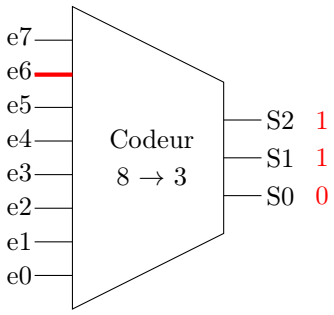


FIGURE 2.4 – Codeur 8 entrées vers 3 sorties

Au niveau de la Table de vérité du 74148 on aura :

	Inputs								Outputs				
$\overline{E_I}$	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{A_2}$	$\overline{A_1}$	$\overline{A_0}$	GS	E_0
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

Les entrees sont ici actives à l'état bas tandis que les sorties sont aussi actives à l'état bas. 6 est une entrée active. On a ici un codeur prioritaire, c.a.d. que si une entrée de numéro inférieure à 6, elle ne sera pas considérée (d'ou les X dans la ligne) alors on a sur les sorties $A_2, A_1, A_0 \rightarrow L\ L\ H$: 6 en logique négative

Décodeur

Ce sont les circuits qui transforment une information de n bits vers 2^n bits.

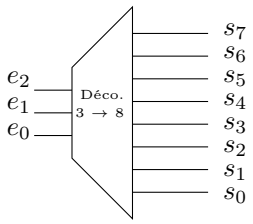


FIGURE 2.5 – Décodeur 3 entrées vers 8 sorties

Pour chaque combinaison de variables d'entrées on a une seule sortie active. Les combinaisons d'entrées sont appelées adresses car elles expriment en numérotation binaire le numéro décimal de la sortie activée.

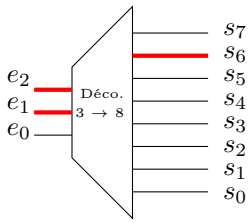


FIGURE 2.6 – Les entrées e_2, e_1 donne le code (110) qui active la sortie 6

Décodeur 74138 : Table de vérité du datasheet

Inputs						Outputs							
Enable			Select										
G_{2A}	$\overline{G_{2B}}$	G_1	A	B	C	$\overline{Y_7}$	$\overline{Y_6}$	$\overline{Y_5}$	$\overline{Y_4}$	$\overline{Y_3}$	$\overline{Y_2}$	$\overline{Y_1}$	$\overline{Y_0}$
L	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H	L
H	L	L	L	L	H	H	H	H	H	H	H	L	H
H	L	L	L	H	L	H	H	H	H	H	L	H	H
H	L	L	L	H	H	H	H	H	H	L	H	H	H
H	L	L	H	L	L	H	H	H	L	H	H	H	H
H	L	L	H	L	H	H	H	L	H	H	H	H	H
H	L	L	H	H	L	H	L	H	H	H	H	H	H
H	L	L	H	H	H	L	H	H	H	H	H	H	H

Si on a la combinaison $HHL = 6$ sur les entrées A, B, C alors la sortie $\overline{Y_6}$ sera active (L) en logique négative.

Le distributeur de boisson *Un appareil comporte 3 cuves contenant de l'eau, du concentré de cassis, du concentré de menthe. Ce distributeur permet d'obtenir de l'eau, de la menthe à l'eau et du cassis à l'eau par le moyen des boutons e, m, c qui commandent les electrovannes E, M, C. Le mélange cassis-menthe étant interdit. Une pièce p doit être introduite sauf pour l'eau pure qui est gratuite. L'appui sur un bouton e, m, c pour l'introduction déclenche une temporisation. Le déclenchement et l'échéance de la temporisation ne seront pas traités. Si la temporisation arrive à échénace avant qu'un choix cohérent soit fait, la pièce éventuellement présente est rendue par la fonction de restitution P. Cette fonction est aussi activée si le choix est incohérent (mc).*

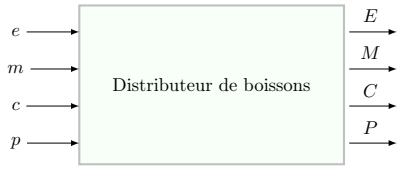
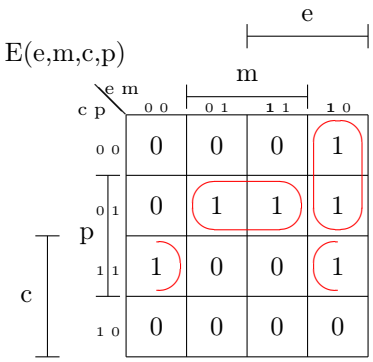


FIGURE 2.7 – Analyse des entrées et des sorties

La table de vérité L'analyse du cahier des charges combinaison par combinaison donne la table suivante :

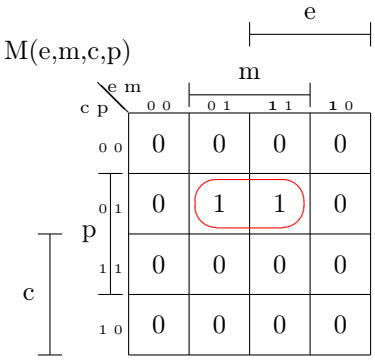
e	m	c	p	E	M	C	P
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	0
0	0	1	1	1	0	1	0
0	1	0	0	0	0	0	0
0	1	0	1	1	1	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	1
1	0	1	0	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	0	0	0	0	0
1	1	0	1	1	1	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

Tables de Karnaugh de la variable E



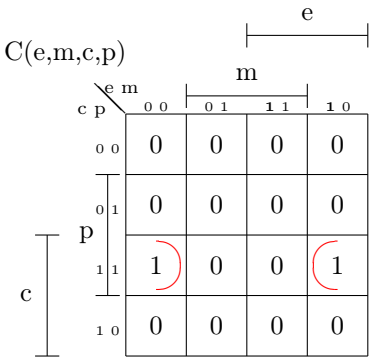
$$E(e,m,c,p) = \bar{m}.c.p + m.\bar{c}.p + e.\bar{m}.\bar{c}$$

Tables de Karnaugh de la variable M



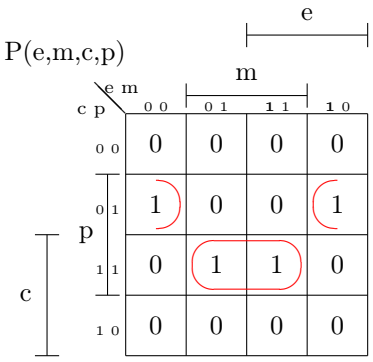
$$M(e,m,c,p) = m.\bar{c}.p$$

Tables de Karnaugh de la variable C



$$C(e, m, c, p) = \bar{m}.c.p$$

Tables de Karnaugh de la variable P



$$P(e, m, c, p) = \bar{m}.\bar{c}.p + m.c.p$$

Mise en équations

$$E(e, m, c, p) = \bar{m}.c.p + m.\bar{c}.p + e.\bar{m}.\bar{c}$$

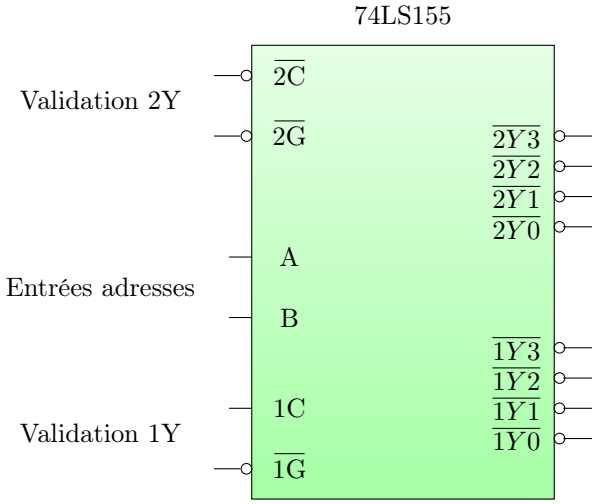
$$M(e, m, c, p) = m.\bar{c}.p$$

$$C(e, m, c, p) = \bar{m}.c.p$$

$$P(e, m, c, p) = \bar{m}.\bar{c}.p + m.c.p$$

Implémentons maintenant ce distributeur à l'aide d'un double décodeur le $74LS155$

Réalisation avec un 74LS155 Ce composant comporte deux ensembles de sorties $1Y$, $2Y$ pilotable par les entrées A et B



A partir d'une même combinaison sur A et B on peut activer 2 lignes à la fois : une dans $1Y_i$ et une dans $2Y_i$:

— $AB == 00$ les lignes $\overline{1Y_0}$ et $\overline{2Y_0}$ sont actives
— $AB == 10$ les lignes $\overline{1Y_2}$ et $\overline{2Y_2}$ sont actives

D'autre part, les entrées de validation de $\overline{1G}$ $1C$ et $\overline{2G}$ $\overline{2C}$ permettent de sélectionner le(s) groupes de sortie actif(s). Les équations du multiplexeur 74155 sont :

$$\overline{2Y_3} = A.B.\overline{2C}.\overline{2G} \qquad \overline{1Y_3} = A.B.1C.\overline{1G}$$
$$\overline{2Y_2} = A.\overline{B}.\overline{2C}.\overline{2G} \qquad \overline{1Y_2} = A.\overline{B}.1C.\overline{1G}$$
$$\overline{2Y_1} = \overline{A}.B.\overline{2C}.\overline{2G} \qquad \overline{1Y_1} = \overline{A}.B.1C.\overline{1G}$$
$$\overline{2Y_0} = \overline{A}.\overline{B}.\overline{2C}.\overline{2G} \qquad \overline{1Y_0} = \overline{A}.\overline{B}.1C.\overline{1G}$$

Dans toutes les expressions de E, M, C, P on trouve toujours m et c avec e ou p en facteur. Donc plaçons :

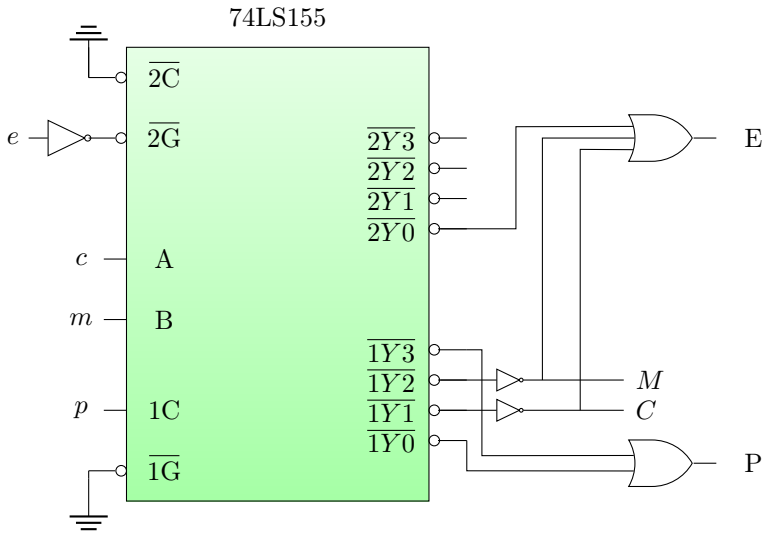
— m et c sur les entrées adresses A et B ;
— p sur l'entrée de validation $1C$;
— e sur l'entrée de validation $2G$;
— les validations $1G$ et $2C$ à la masse ;

Les équations du multiplexeur deviennent :

$$\overline{2Y_3} = m.c.\overline{e} \qquad \overline{1Y_3} = m.c.p$$
$$\overline{2Y_2} = m.\overline{c}.\overline{e} \qquad \overline{1Y_2} = m.\overline{c}.p$$
$$\overline{2Y_1} = \overline{m}.c.\overline{e} \qquad \overline{1Y_1} = \overline{m}.c.p$$
$$\overline{2Y_0} = \overline{m}.\overline{c}.\overline{e} \qquad \overline{1Y_0} = \overline{m}.\overline{c}.p$$

D'ou les expressions de E, M, C, P en fonction des $\overline{iY_j}$:

— $M = m.\overline{c}.p = \overline{1Y_2}$;
— $C = \overline{m}.c.p = \overline{1Y_1}$;
— $P = \overline{m}.\overline{c}.p + mcp = \overline{1Y_0} + \overline{1Y_3}$;
— $E = \overline{m}.c.p + m.\overline{c}.p + \overline{m}.\overline{c}e = \overline{1Y_1} + \overline{1Y_2} + \overline{2Y_0}$

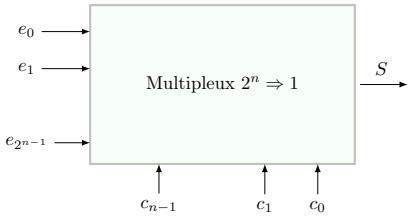


Multiplexeur

Définition 2.3.1. *Multiplexeur Un multiplexeur réalise l'aiguillage de 2^n entrées vers une sortie avec n bits de commande.*

Equation de la sortie :

$$\begin{aligned} s &= e_0.\overline{c_{n-1}}.\overline{c_{n-2}}.....\overline{c_1}.\overline{c_0} \\ &+ e_1.\overline{c_{n-1}}.\overline{c_{n-2}}.....\overline{c_1}.c_0 \\ &+ e_2.\overline{c_{n-1}}.\overline{c_{n-2}}.....c_1.\overline{c_0} \\ &+ ... \\ &+ e_n.c_{n-1}.c_{n-2}....c_1.c_0 \end{aligned}$$

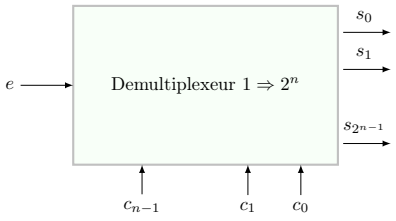


Le Démultiplexeur

Définition 2.3.2. *Démultiplexeur Un démultiplexeur réalise l'aiguillage d'une entrée vers 2^n sorties avec n bits de commande.*

Equation des sorties :

$$\begin{aligned} s_0 &= e.\overline{c_{n-1}}.\overline{c_{n-2}}.....\overline{c_1}.\overline{c_0} \\ s_1 &= e.\overline{c_{n-1}}.\overline{c_{n-2}}.....\overline{c_1}.c_0 \\ s_n &= e.c_{n-2}....c_1.c_0 \end{aligned}$$



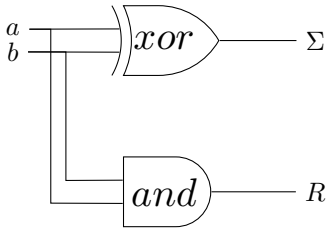
2.3.2 L'additionneur

C'est un circuit qui réalise l'addition de deux nombres binaires et produit la somme et la retenue :

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

$$\Sigma = \bar{a}.b + a\bar{b} = a \oplus b$$

$$Retenue = a.b$$



L'additionneur complet

Pour réaliser une addition sur n bits, il faut un additionneur capable de réaliser l'addition sur 3 éléments : a_i, b_i et la retenue produite par le rang précédent r_i.

a_i	b_i	r_i	r_{i+1}	Σ_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$\Sigma_i(a_i,b_i,r_i)$

r_i \ $a_i b_i$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$\Sigma_i = a_i \oplus b_i \oplus r_i$$

$r_{i+1}(a_i,b_i,r_i)$

r_i \ $a_i b_i$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$\begin{aligned} r_{i+1} &= a_i.b_i + r_i.b_i + r_i.a_i \\ &= a_i.b_i + r_i.(a_i + b_i) \end{aligned}$$

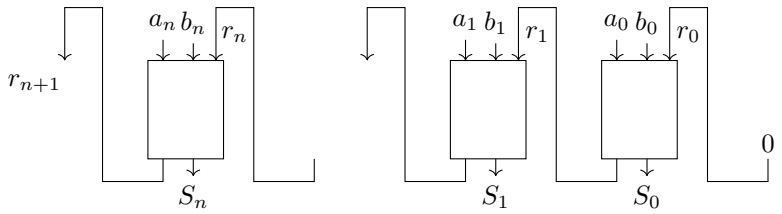
$r_{i+1}(a_i,b_i,r_i)$

r_i \ $a_i b_i$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$\begin{aligned} r_{i+1} &= a_i.b_i + r_i.(\bar{a}_i.b_i + a_i.\bar{b}_i) \\ r_{i+1} &= a_i.b_i + r_i.(a_i \oplus b_i) \end{aligned}$$

Définition 2.3.3. *Equations de l'additionneur complet*

$$\begin{aligned} \Sigma_i &= (a_i \oplus b_i) \oplus r_i \\ r_{i+1} &= a_i.b_i + r_i.(a_i \oplus b_i) \end{aligned}$$



L'additionneur à retenue anticipée

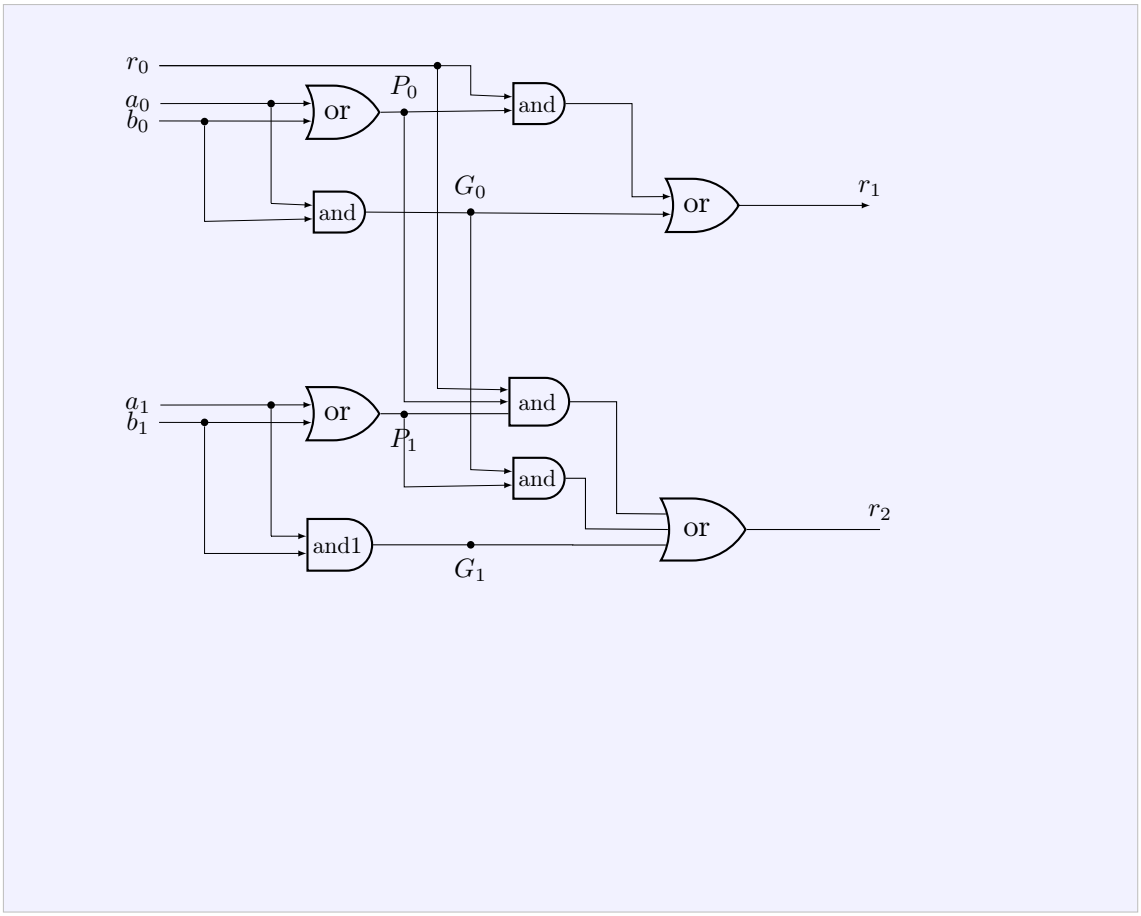
Prenons la seconde forme de l'additionneur :

$$r_{i+1} = a_i.b_i + r_i.(a_i \oplus b_i) = a_i.b_i + r_i.(a_i + b_i)$$

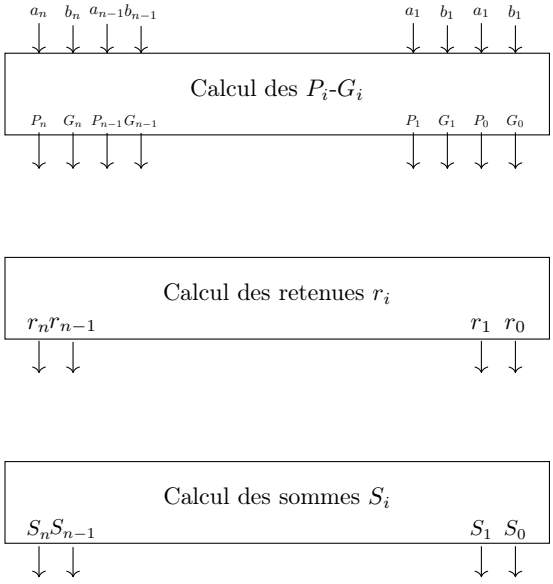
Posons : $P_i = a_i + b_i$ et $G_i = a_i.b_i$

On a :

$$\begin{aligned} r_1 &= a_0.b_0 + (a_0 + b_0).r_0 = G_0 + P_0.r_0 \\ r_2 &= G_1 + P_1.r_1 \\ &= G_1 + P_1.(G_0 + P_0.r_0) = G_1 + P_1.G_0 + P_1.P_0.r_0 \\ r_3 &= G_2 + P_2.r_2 \\ &= G_2 + P_2.(G_1 + P_1.G_0 + P_1.P_0.r_0) \\ &= G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.r_0 \\ &... \end{aligned}$$



Les temps de calcul des retenues sont tous égaux à condition de réaliser des portes à plus de deux entrées. Ils correspondent aux temps de calcul des P_i, G_i, et donc à l'étage des ET et à l'étage des OU.



Le temps de calcul est donc indépendant du nombre de bits au dépend de la complexité des portes. En effet certaines portes auront $3, 4, \ldots$ jusqu'à n entrées.

2.3.3 Le comparateur

Un comparateur compare 2 nombres A et B sur n bits :

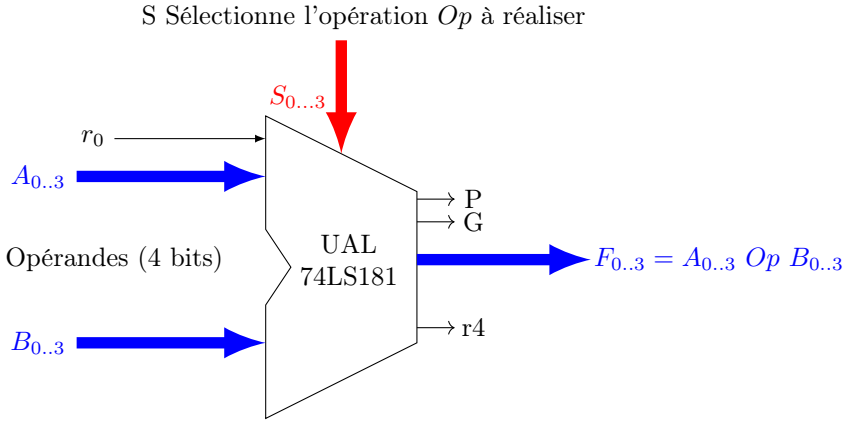
— GT : sortie active si $A > B$
— EQ : sortie active si $A = B$
— LT : sortie active si $A < B$
— GE : sortie active si $A \geq B$
— LE : sortie active si $A \leq B$
— NE : sortie active si $A \neq B$

Le comparateur délivre six sorties :

$$
\begin{aligned}
GT &= A_n.\bar{B_n} + (A_n \odot B_n).(A_{n-1}.\overline{B_{n-1}}) + \ldots + (A_n \odot B_n)\ldots(A_1 \odot B_1).(A_0.\overline{B_0}) \\
EQ &= (A_n. \odot B_n).(A_{n-1}. \odot B_{n-1})\ldots(A_0 \odot B_0) \\
LT &= \overline{GT} \\
GE &= GT + EQ \\
LE &= LT + EQ \\
EQ &= \overline{EQ}
\end{aligned}
$$

2.3.4 L'UAL : 74181

L'UAL est composé de 2 entrées de 4 bits A et B, d'un bus de commande S bits qui permet d'effectuer 16 opérations différentes. Le résultat de l'opération produit un résultat sur le bus F. Il y a une retenue entrante r_0 et sortante r_4.



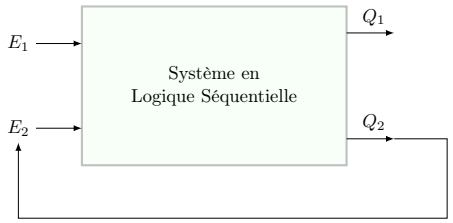
— Arithmétiques :
  * Additions : $A + B$, $A + B + 1$ (lorsque $r_0 = 1$)
  * $-A = CA_2(A)$
  * Soustractions : $A - B$, $A - B - 1$
— Logiques :
  * ET : $(A_3.B_3).(A_2.B_2).(A_1.B_1).(A_0.B_0)$
  * OU : $(A_3 + B_3).(A_2 + B_2).(A_1 + B_1).(A_0 + B_0)$ ($+ =$ ou)
  * $NOT(A) = CA_1(A)$
  * Nor, Nand ...
  * Décalages $A \leftarrow LSL(A)$ ...

Chapitre 3

Logique Séquentielle

3.1 Introduction

Dans un système en logique séquentielle au moins une sortie reboucle sur une entrée et il devient, dès lors, impossible de décrire le système par une fonction logique.



Une sortie reboucle sur une entrée

Dans ce chapitre nous allons définir d'abord comment la mémoire advient avec la notion de re-bouclage, puis nous introduirons les circuits séquentiels de base : les bascules et le séquenceurs. Puis nous présenterons une méthode d'analyse des circuits séquentiel.

3.1.1 La notion de mémoire

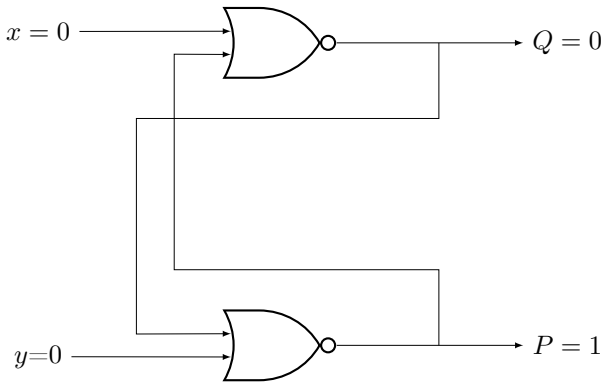
Le concept d'élément mémoire élémentaire - Binary digIT : BIT - émerge au travers des concepts suivants : Une lecture non destructrice de l'état du bit, la mise à un un ou à zéro, et le maintien en l'état du bit en absence de toute modification. Cela peut se traduire par la table de vérité suivante :

R	S	Q^t	Q^{t+1}
0	0	Q^t	Q^t
0	1	Q^t	1
1	0	Q^t	0
1	1	Q^t	*

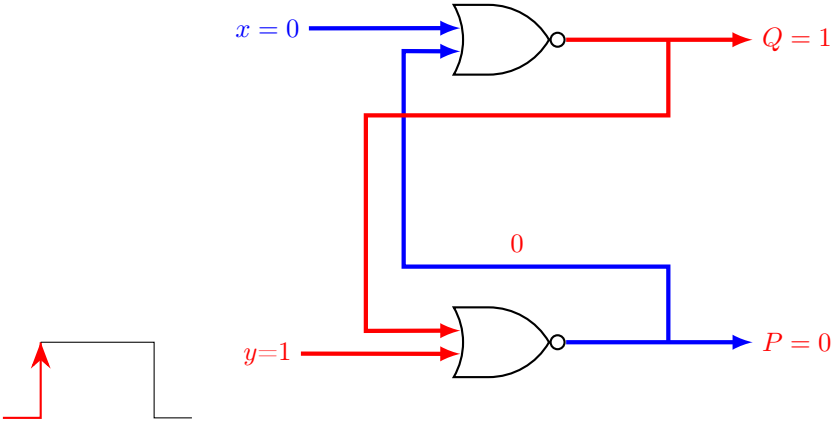
Dans cette table la première ligne exprime que en l'absence d'action le bit garde sa valeur. La second exprime le set, la troisième le reset. La quatrième est que l'on va s'interdire les action simultanées du set et du reset.

Examinons maintenant ce qui se passe si on re-bouble deux portes *nor* :

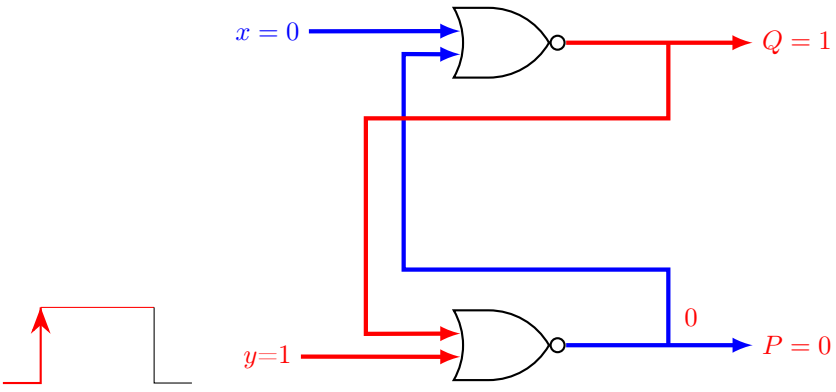
Ces test nous ont permis de comprendre qu'en re-bouclant des portes on peut mettre en œuvre le principe de mémoire élémentaire qui correspond aussi à la notion de bascule.



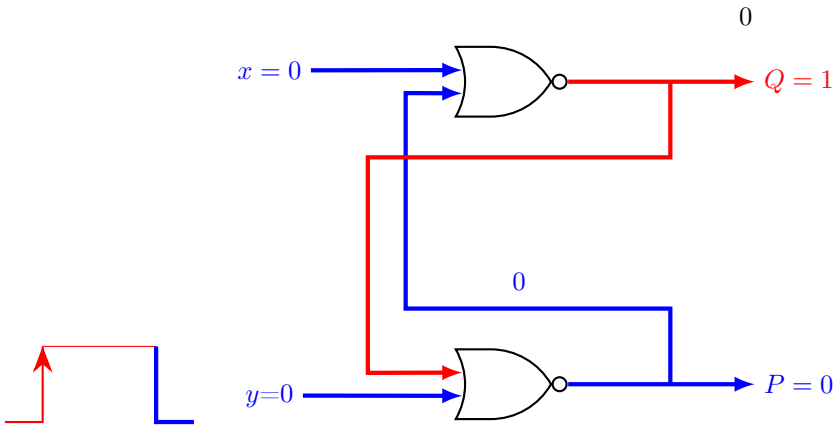
L'état est stable



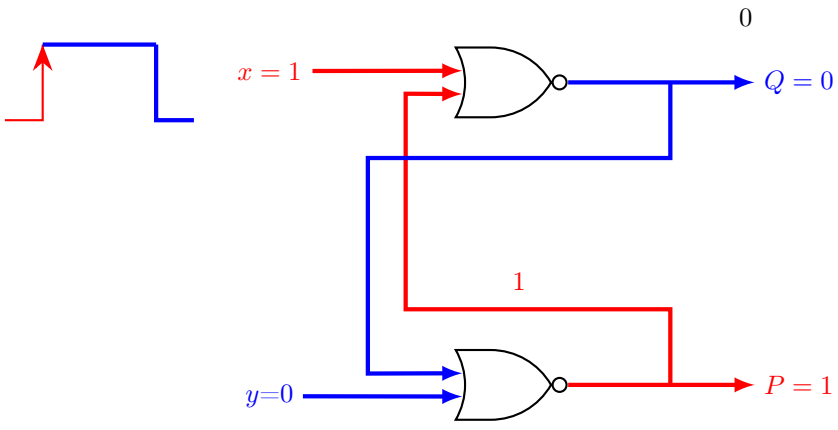
Sur un front montant de y, Q passe à un, P passe à zéro



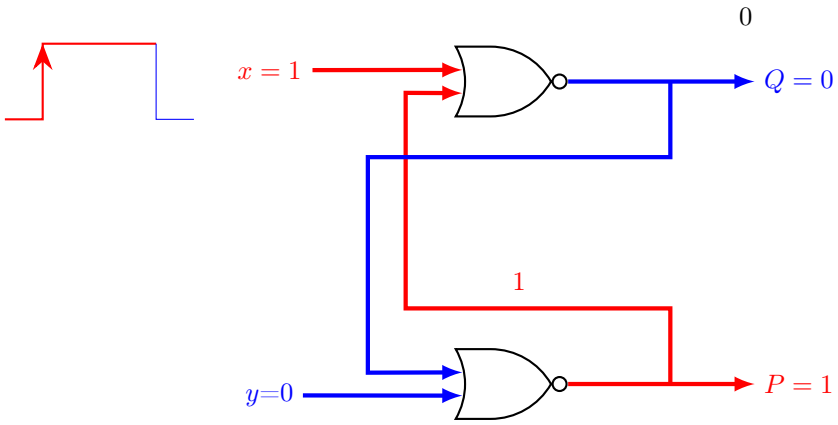
L'état est stable pendant le niveau haut de y



Après la retombée de y l'état Q reste à un



Après le front montant de x, Q passe à zéro, P à un



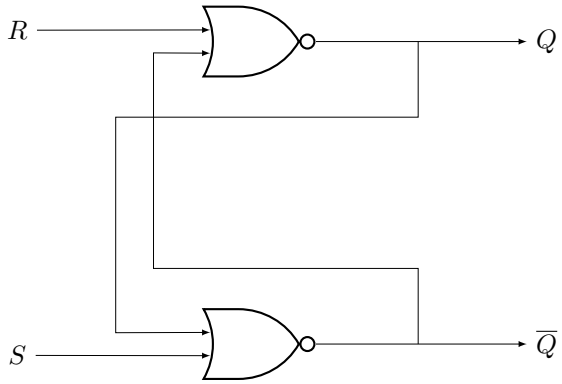
Pendant le niveau haut de x, Q reste à zéro, P à un

3.1.2 La bascule initiale : la RS asynchrone

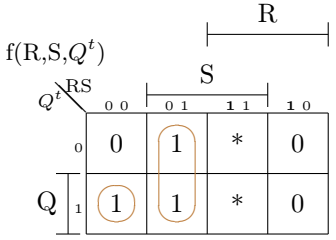
Dans une bascule RS, R sert à mettre la bascule à zéro et S à un. De plus, on interdit de faire un R et un S en même temps. Une bascule RS peut aussi se définir par sa table de vérité :

R	S	Q^t	Q^{t+1}
0	0	Q^t	Q^t
0	1	Q^t	1
1	0	Q^t	0
1	1	Q^t	*

Comme nous l'avons vu, une RS peut être réalisée à l'aide deux portes nor re-bouclées :

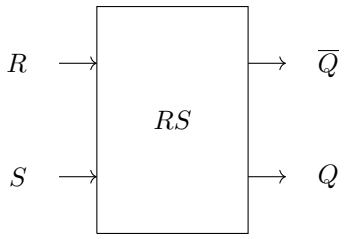


On la représentera maintenant par :



$$f(R,S,Q^t) = Q^{t+1} = \overline{R}.\overline{S}.Q^t + \overline{R}.S + RS$$

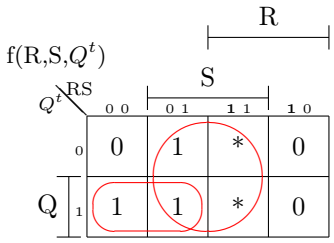
$$= \overline{R}.\overline{S}.Q^t + \overline{R}.S$$



Maintenant considérons une autre simplification dans la table de Karnaugh :

$$Q^{t+1} = \overline{R}.\overline{S}.Q^t + \overline{R}.S + RS$$

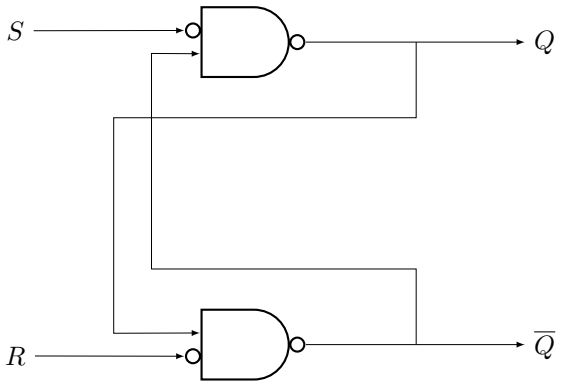
$$= \overline{R}.Q^t + S$$



L'équation caractéristique est donc : $Q^{t+1} = \overline{R}.Q^t + S$

Réalisation de la RS asynchrone en NAND

En NAND, à partir d'un forme en somme : $\overline{\overline{Q^{t+1}}} = \overline{\overline{\overline{R}.Q^t + .S}} = \overline{\overline{\overline{R}.Q^t}.\overline{S}}$

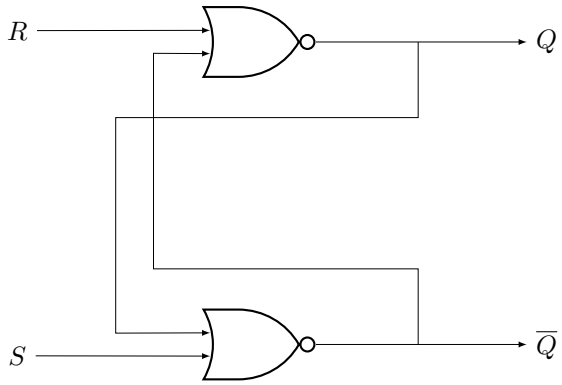


Réalisation de la RS asynchrone en NOR

En NOR, à partir d'un forme en produit :

$$Q^{t+1} = \overline{R}.\overline{S}.Q^t + \overline{R}.S = \overline{R}.(\overline{S}.Q^t + S) = \overline{R}.(Q^t + S)$$
$$Q^{t+1} = \overline{\overline{Q^{t+1}}} = \overline{\overline{\overline{R}.(Q^t + S)}} = \overline{R + \overline{Q^t + S}}$$

Et on retrouve la forme précédemment étudiée :



Race conditions

Maintenant on peut se poser la question de qui se passe si on applique R et S en même temps ? Si $S = R = 1$ alors $Q = \overline{Q} = 0$ (en NOR). Cet état est alors stable. Considérons maintenant l'application simultanée de $S = R = 0$, les deux portes n'étant jamais réellement identiques, l'une est plus rapide et donc, l'une commute sa sortie à un en premier verrouillant la seconde porte dont la sortie restera bloquée à zéro désormais. On a ici une situation de course : c'est la porte la plus rapide qui définit l'état de la bascule ! ! ! Comment éviter cette situation ?

Une première idée est de synchroniser les signaux d'entrée avec un signal d'autorisation par une porte ET.

— Si ce signal est à zéro, les entrées sont à zéros,

— Si ce signal est à UN, les entrées sont susceptibles de faire évoluer les sorties.

Ce concept permet de se prémunir contre des variations indésirables, qui seraient susceptibles de provoquer l'apparition simultanée de ces deux signaux.

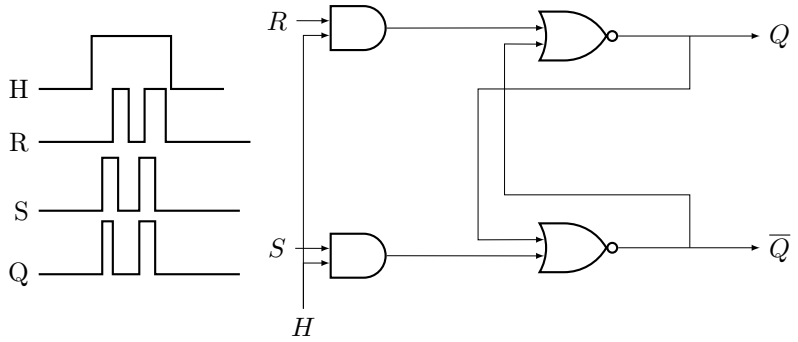
3.2 Typologie des bascules

Il y a trois types de bascules : (i) les "Latch" : à Déclenchement sur niveau (haut ou bas), (ii) Les "Edge triggered" : à déclenchement sur front, (iii) et les Les maîtres-esclaves.

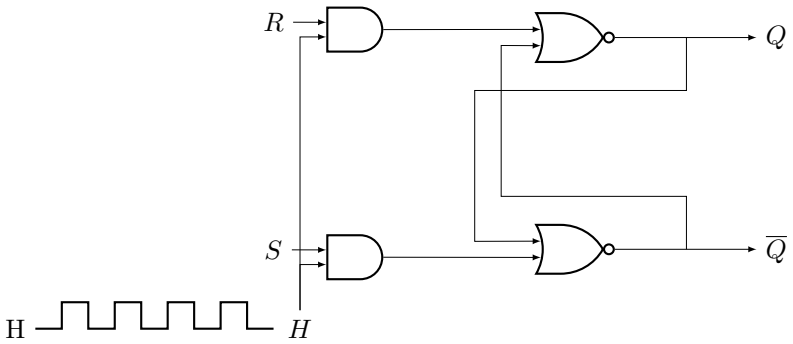
3.2.1 Les bascules à verrou : les "Latch"

Un bascule latch est une bascule synchrone dont toute variation sur les entrées pendant le niveau d'activation du signal d'horloge est pris en compte sur ses sorties.

Dans la figure ci-dessous, on peut remarquer les deux portes *et* qui verrouillent les entrées sur un niveau bas de l'horloge H :



RST Latch

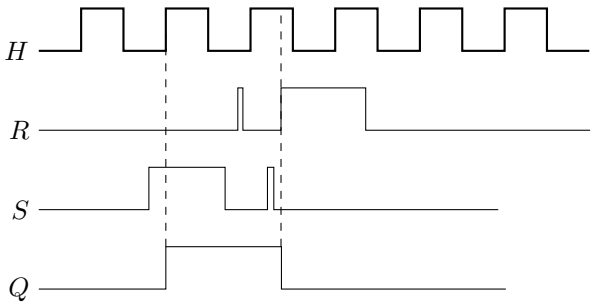


La table de vérité de la *RST* Latch

Cette table fait apparaître la nouvelle colonne H qui correspond à l'horloge :

H	R	S	Q^{t+1}
0	$*$	$*$	Q^t
1	0	0	Q^t
1	0	1	1
1	1	0	0
1	1	1	$*$

Traitons maintenant un exemple. Pour un signal d'horloge donné H, donnons-nous aussi deux signaux R et S comprennant aussi des petits pics parasites :



On va prendre en compte sur la sortie Q les niveaux hauts de l'horloge pour le set S et le reset R.

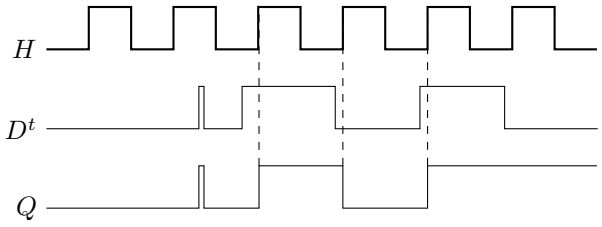
DT Latch

Cette bascule possède une entrée D et une entrée horloge T. Cette bascule fonctionne en mode recopie sur une niveau haut de T et en mode mémoire sur un niveau bas de T.

Table de vérité de la *DT* Latch

T	D	Q^{t+1}
0	$*$	Q^t
1	0	0
1	1	1

Traitons à nouveau un exemple ou pour un signal d'horloge donné, le signal D^t comprend un pic parasite. Ce pic apparaît sur la sortie Q car il s'est produit pendant le niveau haut.

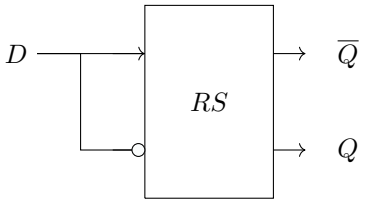


Réalisation d'une DT Latch

A partir d'une RS :

$$Q^{t+1} = S + \overline{R}.Q \tag{3.1}$$
$$Q^{t+1} = D \tag{3.2}$$
$$\text{or } D = D + DQ \tag{3.3}$$
$$\Rightarrow S = D \text{ ET } D = \overline{R} \tag{3.4}$$

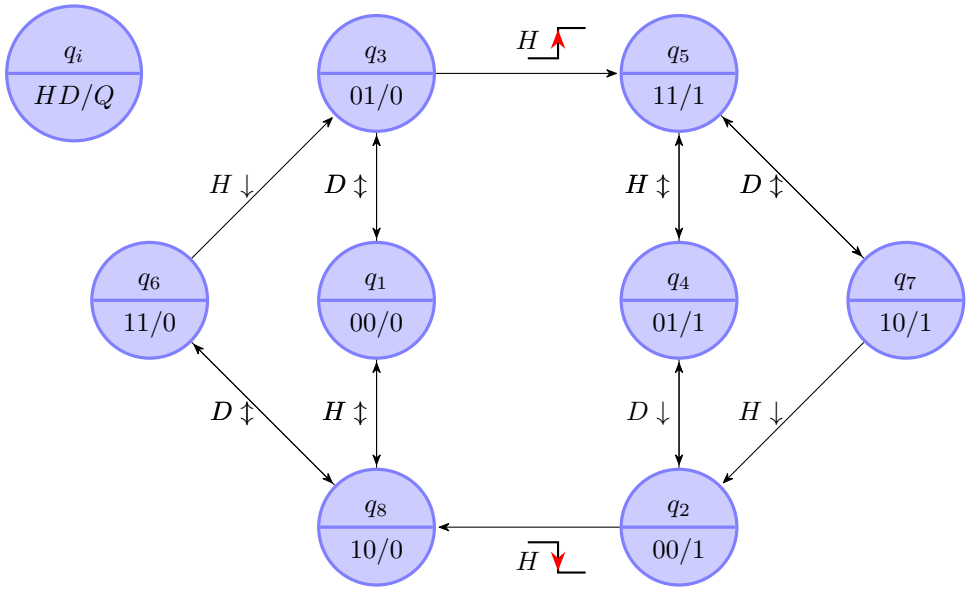
(1) est l'équation caractéristique de la RS
(1) est l'équation caractéristique de la D
(3) correspond à un des théorèmes de la logique : $D + DQ = D(1 + Q) = D$
(4) correspond à la façon de réaliser la bascule D



3.2.2 Les Bascules à déclenchement sur front : "Edge Triggered"

Une bascule à déclenchement sur front est une solution plus performante pour se prémunir des parasites par rapport aux bascules latch. Elles ne seront sensibles que sur les fronts soit des intervalles de temps très courts définis par la technologie utilisée dabs les transistors qui constituent les portes. Examinons dans la prochaîne section, une D Positive Edge Triggered.

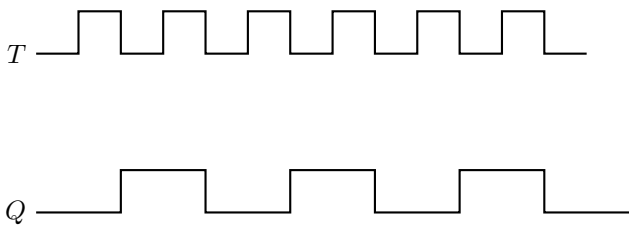
D Positive Edge Triggered



1. Quand l'horloge est stable
   — $6 \leftrightarrow 8, 3 \leftrightarrow 1$ pour $Q = 0$
   — $5 \leftrightarrow 7, 4 \leftrightarrow 2$ pour $Q = 1$
   LA SORTIE NE VARIE PAS.
2. Quand l'horloge passe de 1 vers 0 : $5 \rightarrow 4, 8 \rightarrow 1$: LA SORTIE NE VARIE PAS
3. Quand l'horloge passe de 0 vers 1 avec D = 0 et Q = 0 : $1 \rightarrow 8$ L'entrée D est appliquée sur la sortie mais comme Q était déjà à zéro il n'y a PAS DE CHANGEMENT D'ÉTAT.
4. Quand l'horloge passe de 0 vers 1 avec D = 1 et Q = 1 : $4 \rightarrow 5$ L'entrée D est appliquée sur la sortie mais comme Q était déjà à un il n'y a PAS DE CHANGEMENT D'ÉTAT.
5. La sortie CHANGE D'ÉTAT lors d'un front montant de H avec $D \neq Q$ c.a.d. lors des transitions : $3 \rightarrow 5$ et $2 \rightarrow 8$

T Negative Edge Triggered

Une bascule fonctionnant suivant le type T dispose d'une entrée unique T. La sortie Q change d'état à chaque front descendant de T. Elle divise la fréquence d'entrée de T par 2.



Réalisation de la T Negative Edge Triggered

Réalisation :

$$D = Q_{t+1} = \overline{Q_t}$$

On réalise cette bascule en re-bouclant la sortie Q sur l'entrée D d'une bascule Délai. Seule reste alors l'entrée horloge appelée T. Attention, cette bascule ne peut être réalisée avec une bascule de type "latch", mais avec une "Edge" : à déclenchement sur front.

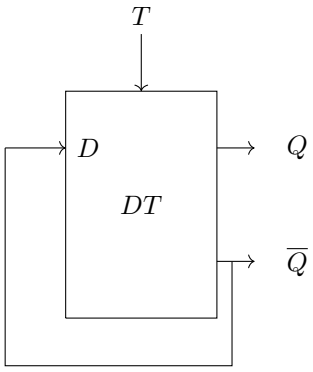


FIGURE 3.1 – Trigger à partir d'une DT

3.2.3 La JKT

— J et K similaire à S et R
— $J = K = 1$ est autorisé : fonctionne comme une bascule T

Réalisation d'une JKT

J	K	Q^{t+1}
0	0	Q^t
0	1	0
1	0	1
1	1	$\overline{Q^t}$

$$\begin{aligned} Q^{t+1} &= J.K.\overline{Q^t} + J.\overline{K} + \overline{J}.\overline{K}.Q^t \\ &= J.K.\overline{Q^t} + J.\overline{K}(\overline{Q^t} + Q^t) + \overline{J}.\overline{K}.Q^t \\ &= J.K.\overline{Q^t} + J.\overline{K}.\overline{Q^t} + J.\overline{K}.Q^t + \overline{J}.\overline{K}.Q^t \\ &= J.\overline{Q^t}(K + \overline{K}) + \overline{K}.Q^t(J + \overline{J}) \\ &= J.\overline{Q^t} + \overline{K}.Q^t \end{aligned}$$

Réalisation de la Bascule JKT

Pour la réaliser, on va identifier les termes 2 à 2 dans l'équation suivante :

$$J.\overline{Q^t} + \overline{K}.Q^t = S + \overline{R}Q^t$$

En posant : $S = J.\overline{Q^t}$ et $R = K.Q^t$

En effet, avec ce choix pour R, on obtient alors :
$\overline{KQ}.Q = (\overline{K} + \overline{Q}).Q = \overline{K}.Q$

La bascule JKTavec une RST

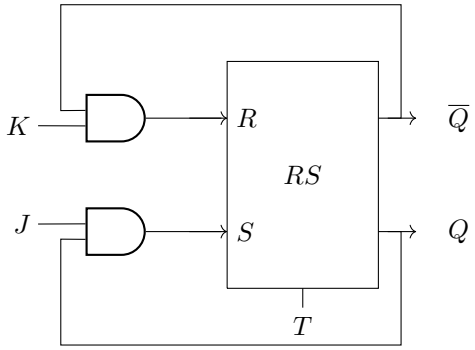
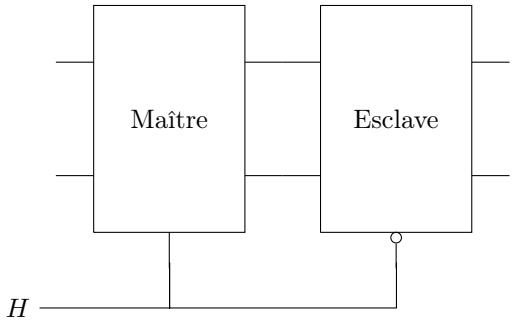


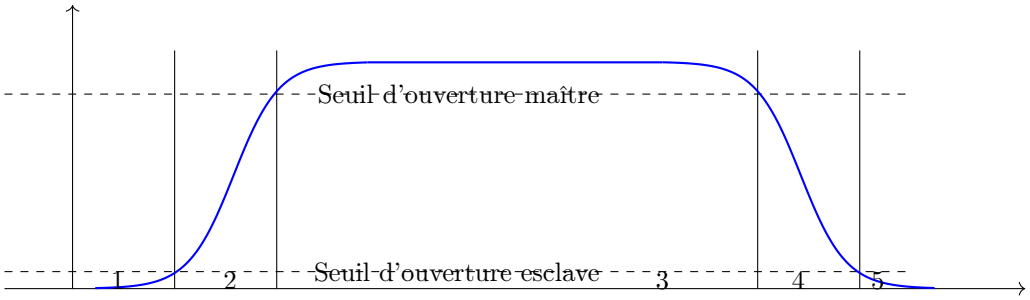
FIGURE 3.2 – JK à partir d'une RST

Avec une JKT on peux faire :
— Une DT, avec $J = \overline{K} = D$
— Une T, avec $J = K = 1$
— Une RST en s'interdisant $J = K = 1$

3.2.4 Bascules Maîtres Esclaves



— Composées de 2 bascules synchrones des D Positive Edge Triggered par exemple.
— Maître et l'esclave
— Lecture et écriture de l'état de la bascules sur le même pulse.

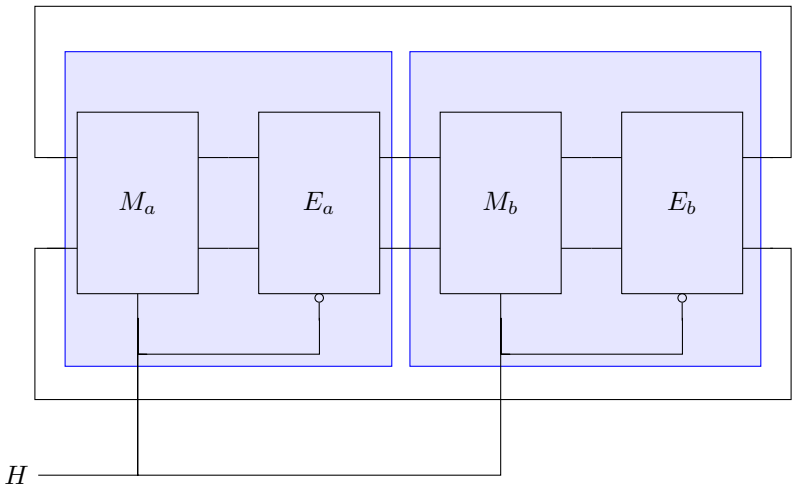


	Maître	Esclave
Zone 1	—/—	———
Zone 2	—/—	—/—
Zone 3	———	—/—
Zone 4	—/—	—/—
Zone 5	—/—	———

— Il n'existe pas de configuration ou le maître et l'esclave sont passant en même temps et donc :
— On peut re-boucler l'entrée et la sortie de la ME et donc :
— On peut implémenter : Echanger i,j ou i=i+1

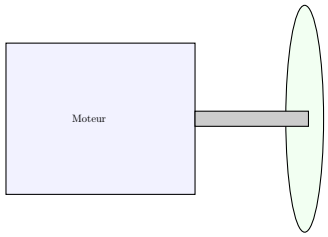
Exemple 1

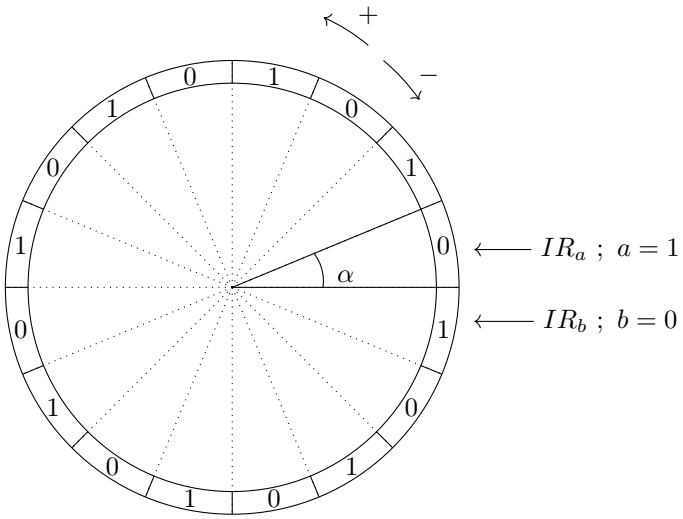
Echange des valeurs des bascules A et B



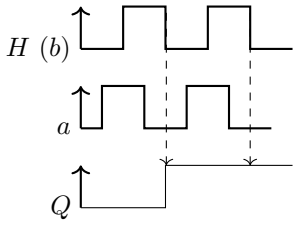
	Maître	Esclave	M_A	E_A	M_B	E_B
Zone 1	—/—	———	0	0	1	1
Zone 2	—/—	—/—	0	0	1	1
Zone 3	———	—/—	1	0	0	1
Zone 4	—/—	—/—	1	0	0	1
Zone 5	—/—	———	1	1	0	0

Exemple 2



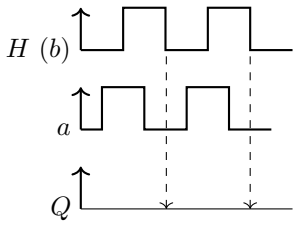


Sens Positif



Exemple 2 : sens Positif

Sens Négatif



Exemple 2 : sens Négatif

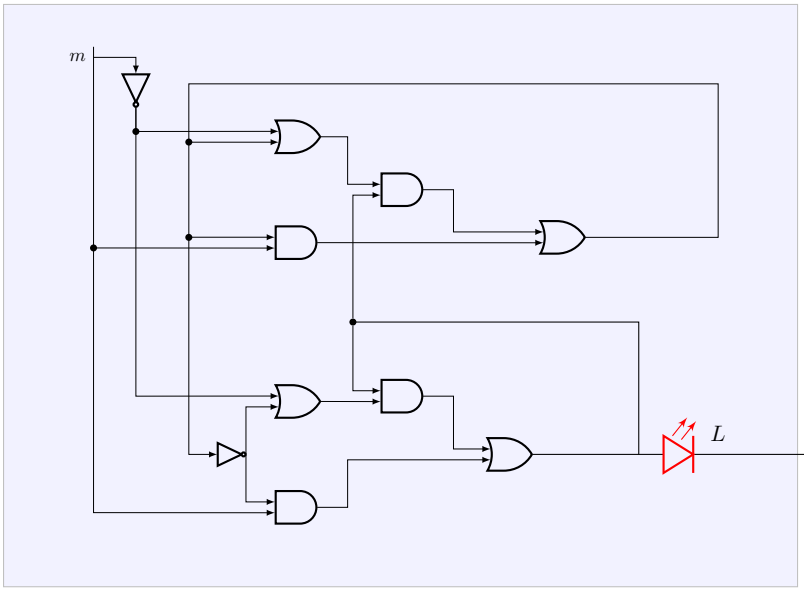
3.3 Analyse de circuits séquentiel

L'analyse à pour objet de proposer une méthode pour analyser un circuit composé de portes élémentaires mais qui comporte des re-bouclages. Ce schéma comporte des entrées et des sorties. L'analyse produit au final, une machine à états. C'est à dire un graphe qui comporte des états stables du système reliés par des évènements qui sont des fronts montants ou descendants sur les entrées du système.

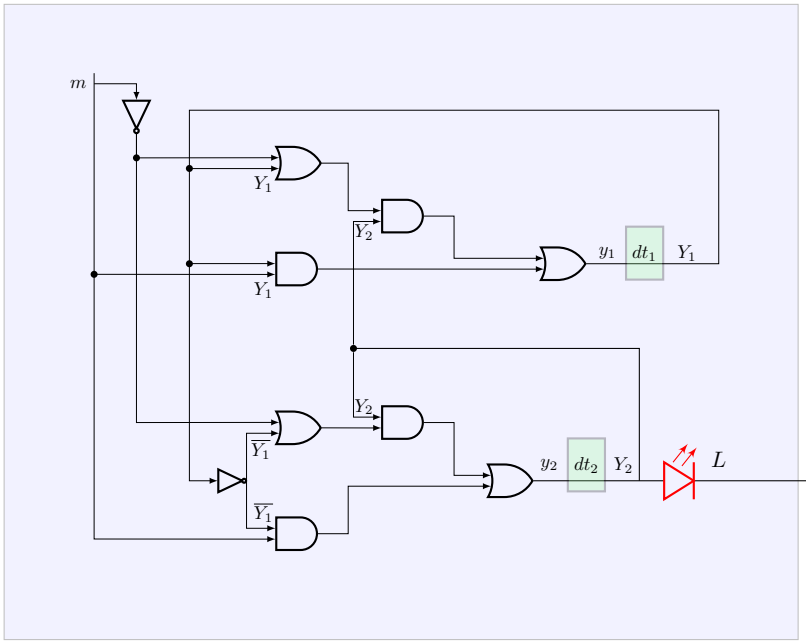
3.3.1 Phases de l'analyse

— Identification des entrées, des sorties principales et des re-bouclages qui correspondent à l'introduction de variables variables internes Y_i et y_i.
— Equations logiques des y_i et s_i
— Table d'excitation et de sortie des y_i et s_i
— identification des états stables
— Table des états nommés
— Construction du graphe

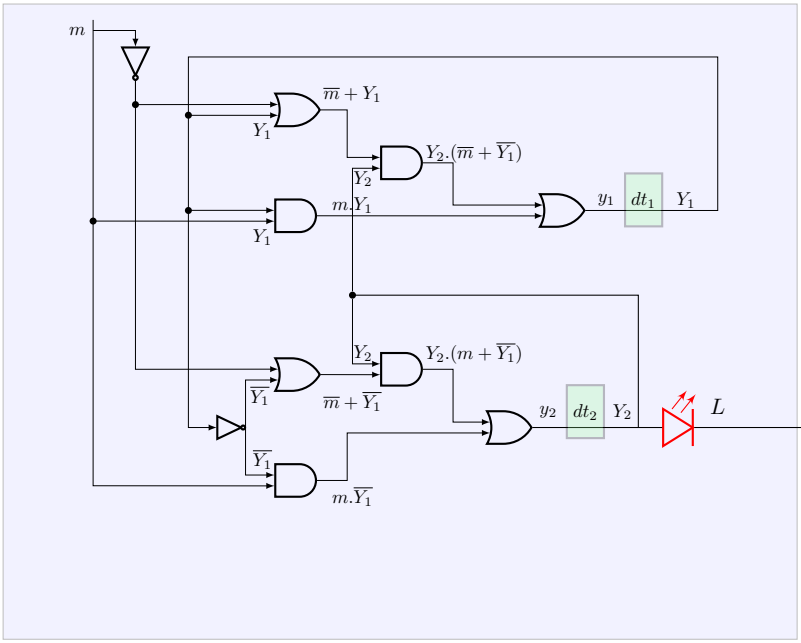
3.3.2 Exemple d'analyse de circuits séquentiel



Identification des re-bouclages



Equations intermédiaires



Equations logiques :

$$\begin{cases} y_1{=}Y_2.(\bar{m}+Y_1)+m.Y_1 \\ y_2{=}Y_2.(\bar{m}+\bar{Y_1})+m.\bar{Y_1} \\ L{=}Y_2 \end{cases}$$

Table d'excitation

Exitation

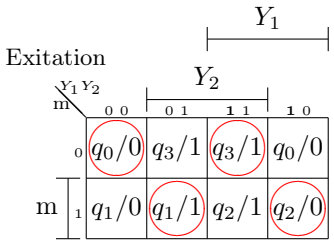
m \ Y_1Y_2	00	01	11	10
0	00_0	11_1	11_1	00_0
1	01_0	01_1	10_1	10_0

Table d'excitation : état stables

Exitation

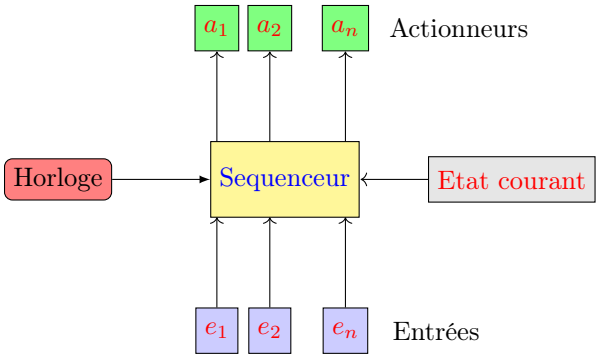
m \ Y_1Y_2	00	01	11	10
0	$\mathbf{00_0}$	11_1	$\mathbf{11_1}$	00_0
1	01_0	$\mathbf{01_1}$	10_1	$\mathbf{10_0}$

Table des états nommés



3.4 Les séquenceurs

Un séquenceur est un système logique séquentiel construit à partir de bascules JKT qui implémente un automatisme. Cet automatisme peut délivrer des signaux sur des lignes à partir d'une horloge en fonction de son état courant. Cette méthode était utilisée, à l'origine, avant l'avènements des automates. Cette méthode, très économique, peut encore être utilisé pour réaliser de petits automatismes dans des environnements peu contraints. Le schéma suivant illustre un séquenceur qui agit sur des actionneurs a_i en fonction de son état courant et dentrées e_i.



3.4.1 Table d'excitation de la bascule JKT

La table d'excitation de la JKT permet de déterminer le prochain état de la bascule JK en fonction de son état courant et de son excitation (état de ses entrées). Par exemple, avec l'excitation $J = 1$ et $K = *$, c.a.d $J = 1$ et $K = 0$ ou bien $J = 1$ et $K = 1$ et dans l'état ou la bascule vaut 0 ($Q = 0$) alors la sortie Q passera à un.

J	K	Q^t	Q^{t+1}
0	*	0	0
1	*	0	1
*	1	1	0
*	0	1	1

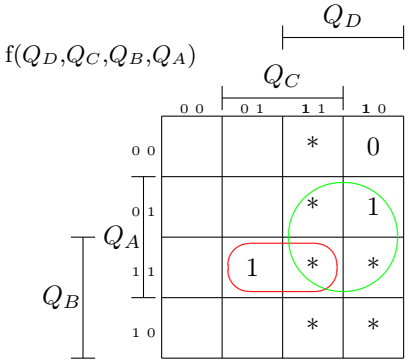
3.4.2 Exemple

On veut réaliser un séquenceur qui affiche sur 4 leds la valeur de comptage en binaire. Ce compteur est cyclique : 0, 1, 2, 3 .... 9. Arrivé à 9 son prochain comptage reboucle à zéro et continue sa séquence infiniment à chaque toip d'horloge. On va réaliser ce séquenceur à l'aide de 4 bascules.

$etat$	Q_D	Q_C	Q_B	Q_A	J_D	K_D	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	0	*	0	*	0	*	1	*
1	0	0	0	1	0	*	0	*	1	*	*	1
2	0	0	1	0	0	*	0	*	*	0	1	*
3	0	0	1	1	0	*	1	*	*	1	*	1
4	0	1	0	0	0	*	*	0	0	*	1	*
5	0	1	0	1	0	*	*	0	1	*	*	1
6	0	1	1	0	0	*	*	0	0	*	1	*
7	0	1	1	1	1	*	*	1	*	1	*	1
8	1	0	0	0	*	0	0	*	0	*	1	*
0	1	0	0	1	*	1	0	*	0	*	*	1

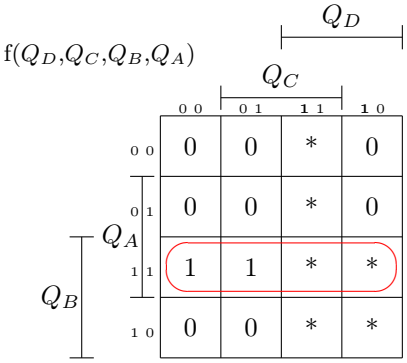
J_DK_D On pose $J_D = K_D$

$$\begin{aligned} J_D &= K_D \\ &= Q_C.Q_B.Q_A + Q_A.Q_D \end{aligned}$$



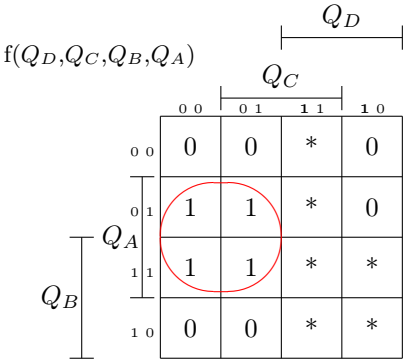
J_CK_C On pose $J_C = K_C$

$$\begin{aligned} J_C &= K_C \\ &= Q_B.Q_A \end{aligned}$$



J_BK_B On pose $J_B = K_B$

$$\begin{aligned} J_B &= K_B \\ &= Q_A.\overline{Q_D} \end{aligned}$$



J_BK_B On pose $J_A = K_A$

$$\begin{aligned} J_A &= K_A \\ &= 1 \end{aligned}$$

Voici maintenant le câblage final :