Chap 1 Nombres et logique combinatoire

1.1 Numération de position

On a appris à utiliser le décimal sans le comprendre. Essayons de comprendre le binaire en admirant au passage l'invention géniale du zéro et de la numération de position.

Exemples

En décimal	En binaire	En base 16 (hexa)
Rang 2 1 0 Poids 10^2 10^1 10^0 100 10 1 En hexa 0x64 0xA 0x1	Rang 3 2 1 0 Poids 2^3 2^2 2^1 2^0 1000 100 10 1 En déc. 8 4 2 1	Rang 2 1 0 Poids 0x10^2 0x10^1 0x10^0 0x100 0x10 0x01 En déc. 256 16 1

Opérations

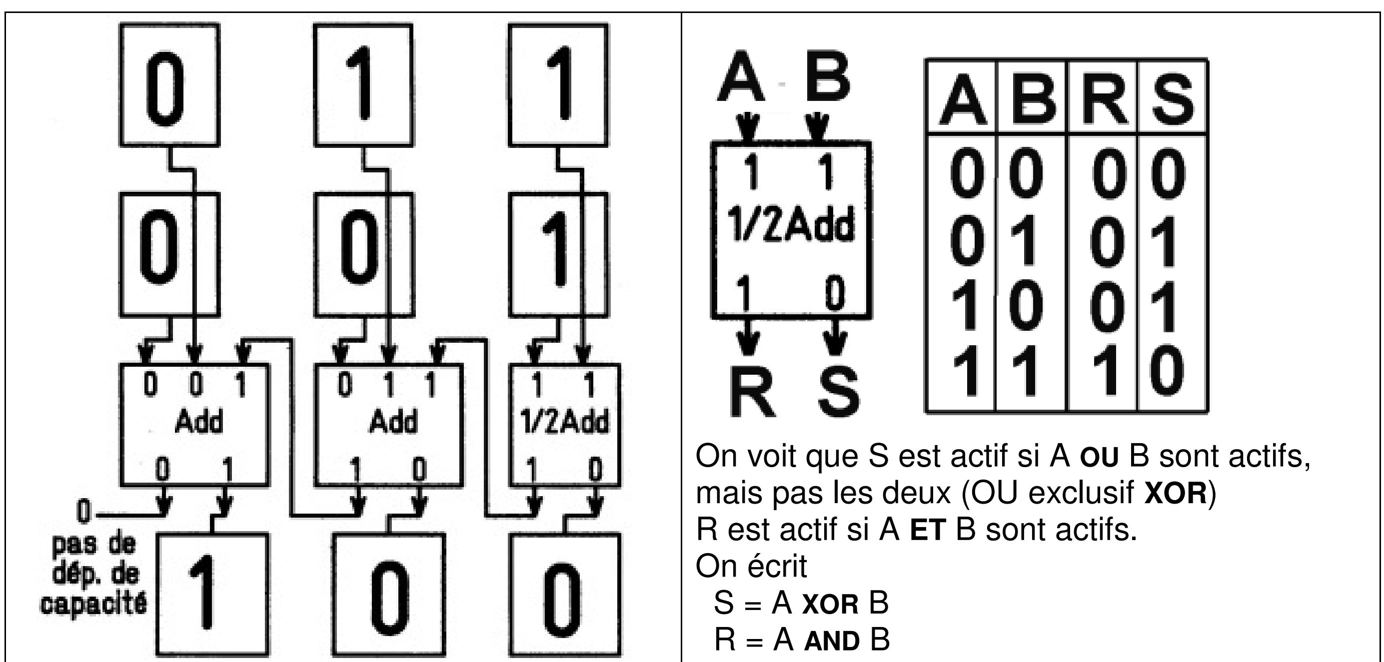
0 0 1 report **2 1 7** **+ 3 4 5** **= 5 6 2**	0 0 0 0 1 1 report **0 1 0 0 1** **+ 0 0 1 0 1** **= 0 1 1 1 0**	0 0 1 1 report **2 1 7** **+ 8 4 D** **= A 6 4**

Fig 1

Dans toutes les bases, les chiffres sont les entiers inférieurs à la base.
On parle de rang des chiffres, de poids pour chaque rang (base puissance rang).
En exprimant les poids dans une base destination, on peut convertir (et reconstituer les règles de conversion que l'on trouve sur internet).

1.2 Structure de l'additionneur

On remarque que la loi d'addition du premier rang est plus simple, puisqu'il n'y a pas de report. Le module qui fait cette première addition est un demi-additionneur. Il a 2 entrées A et B et 2 sorties S et R; comment le fabriquer?
Analysons d'abord son comportement. Il y a 4 cas possibles en entrée, tabulons les sorties. C'est une table de vérité. Elle ne montre que la sortie



A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

On voit que S est actif si A **OU** B sont actifs, mais pas les deux (OU exclusif **XOR**)
R est actif si A **ET** B sont actifs.
On écrit
S = A **XOR** B
R = A **AND** B

Fig 2

On peut matérialiser les entrées/sorties de la "boite" demi-additionneur avec des leviers, des tuyaux, et naturellement avec des signaux électriques.
Mais commençons avec de la programmation, puisque vous connaissez bien.

1.3 Programme en C

En C, on a deux variables booléennes en entrée et deux sorties booléennes également. La table de vérité montre que la somme vaut 1 dans deux cas et le report dans un seul cas.
Les variables a,b, s,r sont déclarées globales bool a; etc

```
void  DemiAdd () {
  ( if (a && !b) )  ||  ( if (!a && b) ) { s =t rue;}  else { s = false;}
  if (a & b) ) { r=true;}  else {r= false;}
}
Autre Formulation
              if ((a=true) && (b=false)) {s=true;}
              if ((a=false) && (b=true)) {s=true;}
              if ((a=true) &&(b=true)) {c=true;}
```

Le C/Arduino ne documente pas les variables booléennes, qui n'ont en fait pas d'existence sous forme de bits isolés. C'est le bit de poids faible d'un mot de 8 bits (octet, byte) et le C propose des instructions pour manipuler un byte qui vaut 0 (false) ou 1 (true). Mais il faut masquer les 7 bits non significatifs quand on fait des opérations 8 bits. La fonction s'écrit alors
Variables globales à déclarer: byte a; etc

```
void  DemiAdd () {
  ( if (a & (~b&0x01) )  |   ( if ((~a&0x01) & b) ) { s =1;}  else { s = 0;}
  if (a & b) ) { r = true;}  else { r =  false;}
}
```

Très bon exercice pour réviser les opérateurs & && | | s'ils on été vus; mais je ne pense pas que cela soit du niveau gymnasial.
Note: c'est un peu plus simple à écrire (pas de masquage) si false = 0xFF.

1.4 Programme en Python ou autre.

1.5 Demi-additioneur avec des interrupteurs

On a vu dans la figure 2 que les conditions **XOR** et **ET** conduisent à S et R.

Avec deux commutateurs doubles (type 2p2t) et deux Leds, le câblage électrique d'un demi-additionneur est simple. Deux commutateurs en série réalisent la fonction **ET** (en parallèle on crée la fonction **OU)**.
Pour le **XOR**, c'est le "schéma 2" des installations électriques, que l'on retrouve dans les pièces qui ont 2 interrupteurs pour allumer la même lampe.

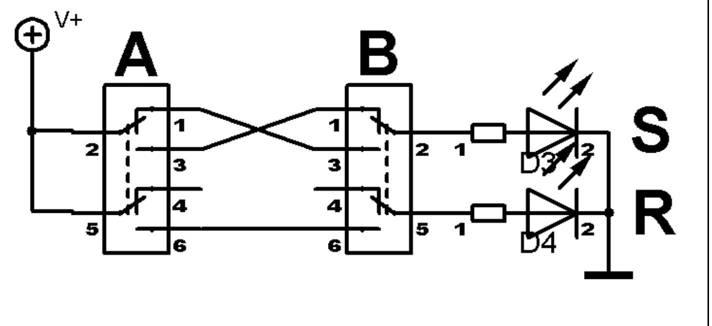


Fig 3

A noter que dans les années 30-40, on a utilisé des relais pour construire des ordinateurs. La différence était que les interrupteurs étaient activés par des bobines (relais) et que les Leds étaient remplacées par les bobines d'autres relais continuant le traitement logique.

1.6 Circuits électroniques

Avec les transistors, puis les circuits intégrés, les circuits logiques sont devenus très fiables et très rapides, en ne consommant que peu d'énergie. Fiabilité veut dire durée de vie et insensibilité aux perturbations, aux fluctuations de la tension d'alimentation, aux différences d'un composant à l'autre, au vieillissement, aux perturbations électromagnétiques.

La tension de 5V s'est généralisée, avec des tensions plus basses à l'intérieur des composants. Ce qui compte, ce sont les marges entre la tension de sortie d'un module et la tension d'entrée sur le récepteur. Le passage d'un état haut à un état bas doit toujours être très rapide (risque d'oscillations si la transition est trop lente).

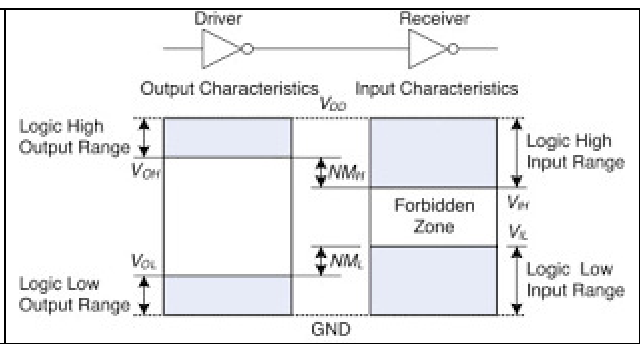


Fig 4

Les modules logiques sont alors vus comme des modèles abstraits que l'on interconnecte en ne se préoccupant que de la fonction mathématique réalisée. Evidemment, il faut tenir compte des temps de propagation, pour lesquels il y a aussi des temps min et max documentés.
La théorie des fonctions logiques et des outils informatiques permet de réaliser les systèmes informatiques actuels. Tout commence par le demi-additionneur et un catalogue de modules, le Lego avec lequel tout se construit. Pour expérimenter, on va utiliser les Logidules et leur simulateur.

1.7 Portes logiques

De nombreux documents décrivent les fonctions logiques de base, avec leurs symboles et leur table de vérité. Inutile de répéter. Les symboles US sont pratiquement les seuls utilisés.

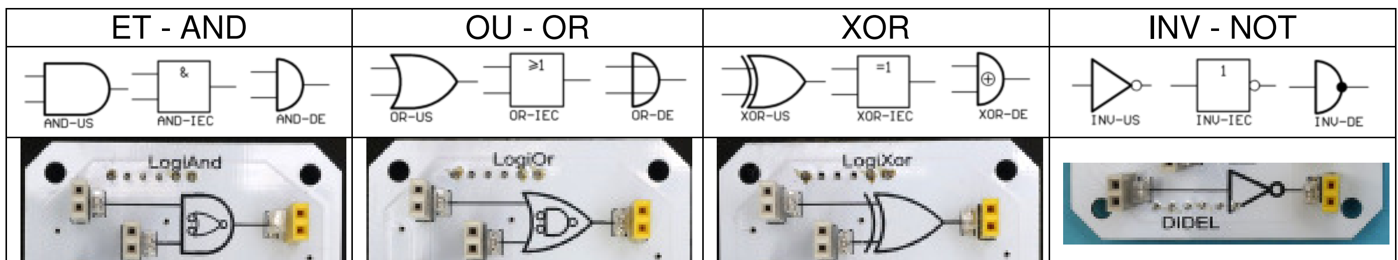


Fig 5

Sautons les étapes de conception d'un système logique, passionnantes à étudier comme une langue ancienne. Pour concevoir un automate, une machine logique, on décrit actuellement les comportements logiques en VHDL et on programme une FPGA ou on réalise un circuit intégré spécial. Les Logidules EPFL utilisaient des circuit s TTL et HC, avec leurs contraintes. Les nouveaux logidules ont des microprocesseurs, leurs petits programmes "temps réel" permettent de montrer des faces trop peu connues de l'informatique (voir compléments).

1.8 Demi-additionneur en logique câblée

Revenons à la figure 2 section 1.2. Le schéma logique du demi-additionneur en découle immédiatement.

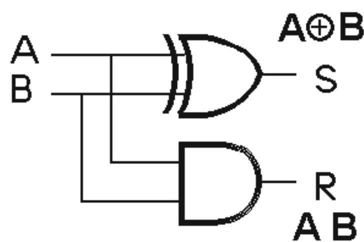


Fig 6

Pour l'additionneur complet, if faut cascader deux demi-additionneur, en comprenant que le report final peut provenir du report du premier additionneur **OU** du deuxième.

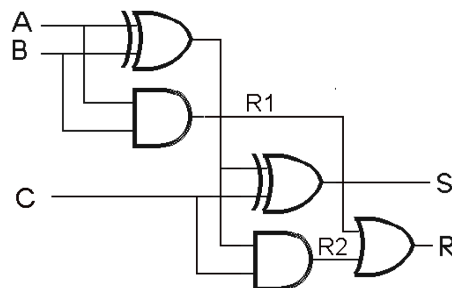


Fig 7

Il y a d'autres schémas d'additionneurs; la théorie des systèmes logiques montre comment transformer les équations logiques. Les équations, donc le schéma, sont établis et simplifiés à partir de tables de vérité.

1.9 Expérimentons

Vérifions le câblage sur logidules du demi-additionneur et de l'additionneur.

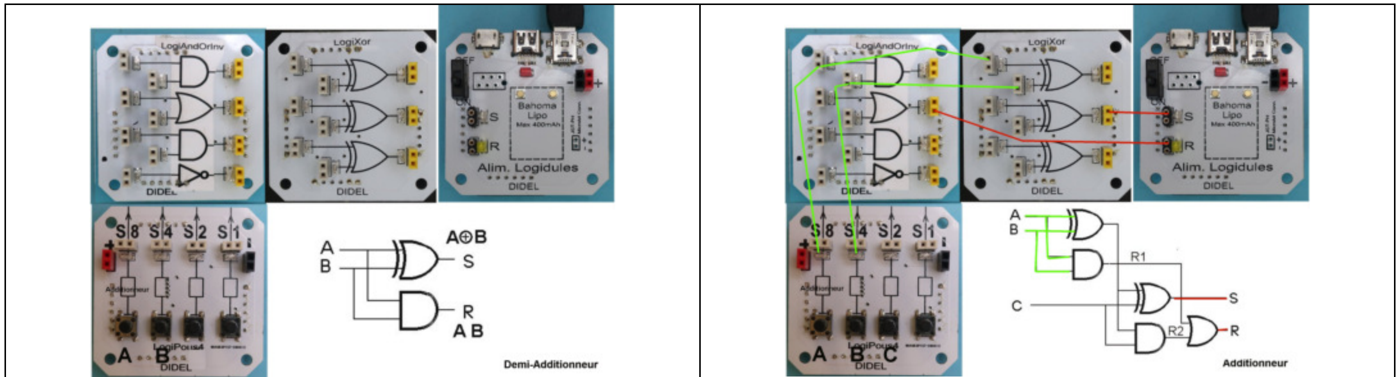


Fig 8

1.10 Aiguillages

Comme en chemin de fer, il faut pouvoir aiguiller l'information. On parle de multiplexeurs, de démultiplexeurs, de décodeurs ou sélecteurs.
La figure ci-dessous donne le schéma d'un multiplexeur et d'un démultiplexeur à 2 positions.
Comme exercice, on peut écrire les équations et dresser les tablesde vérité.

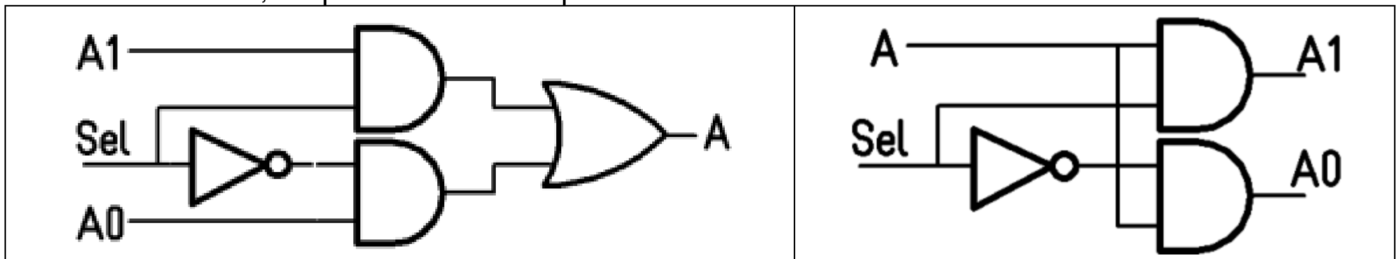


Fig 9 Fig 10

Avec n lignes de sélection, on sélectionne 2^n positions. Dessinez un décodeur à 4 sorties.
Une mémoire a besoin d'un décodeur géant, réalisé évidemment avec des trucs technologiques astucieux

1.11 Décodeur 7 segments

Un affichage 7-segments a besoin d'un circuit logique assez complexe. Il a 4 entrées D8 D4 D2 D1 et pour chaque segment, la fonction à réaliser est issue d'une table de vérité de 16 lignes. On peut voir des schémas loiques sur internet. Une solution est d'utiliser un décodeur (démultiplexeur) à 16 sorties et d'utiliser des portes OU pour rassembler les bits qui doivent activer chaque segment.

a f g b e d c

g f e d c b a

D8 D4 D2 D1

```
-gfedcba
00111111
00000110
01011011
01001111

01100110
01101101
01111101
00000111

01111111
01101111
01110111
01111100

00111001
01011110
01111001
01110001
```

Fig 11

En 1970, avant les microprocesseurs et les mémoires mortes, un décodeur 7-segments se faisait avec des diodes à souder à la bonne intersection entre la sortie d'un décodeur et une ligne jouant la fonction d'un "OU" câblé.

D3 D2 D1 D0 est aussi correct (rang des bits), mais pas D4 D3 D2 D1.

1.12 Signaux en sortie

Une sortie est un signal avec une valeur 0 ou 1 que l'on convertit avec un schéma électronique en effet visible (Led, affichage 7 segments), en mouvement (relais, moteur), etc, en passant par un amplificateur adéquat. Une sortie a toujours deux états, mais en pulsant cette sortie et en utilisant des propriétés physiques ou physiologiques qui intègrent les impulsions binaires, on génère une information qui apparaît analogique.

1.13 Signaux en entrée

Une circuiterie électronique astucieuse permet de convertir par une suite d'opérations un signal analogique en un mot binaire de précision donnée.
Les interrupteurs et claviers génèrent des signaux apparemment logiques, en tout-ou rien. En pratique, ces signaux ont un temps de montée ou des rebonds de contact pendant plusieurs millisecondes, et peuvent être vus comme une multitude d'actions.
Les techniques de filtrage électronique, logique et par programmation sont décrites dans le complément "Rebonds de contact".

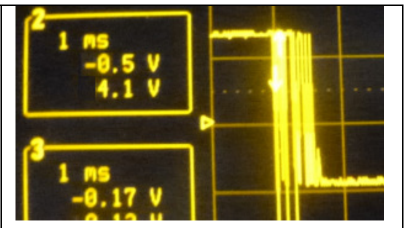
Bornons-nous à préciser ce qui se passe. La résistance de contact est facile à observer à l'oscilloscope, le contact étant connecté à une résistance assurant la tension "contact ouvert". Mécaniquement le début du déplacement agit sur la force de frottement, donc la résistance électrique. Il peut y avoir ensuite de vrais rebonds, à l'arrivée surtout comme c'est facile à imaginer.	

Fig 12

Par logiciel, on échantillonne le signal toutes les 20ms et on peut augmenter la fonctionnalité d'un poussoir, comme expliqué dans le complément.

1.14 Comparateur

Comparer deux nombres binaires c'est vérifier que les bits de même rang sont identiques. Avec un OU exclusif, la sortie vaut 1 si les deux entrées sont différentes. Un inverseur donne l'égalité et une porte ET est nécessaire pour s'assurer que les deux paires de bits sont égaux.
Il existe des variantes de ce schéma. Vérifiez!

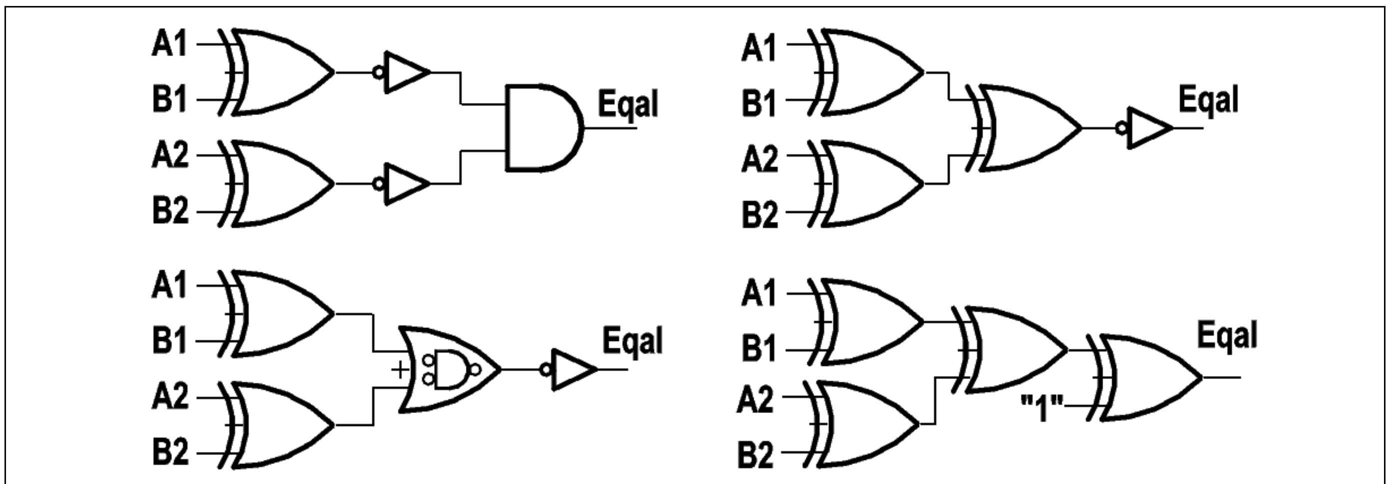


Fig 13

On devine que pour comparer 8 bits, il faut multiplier les portes XOR.

Comparer pour l'inégalité est plus complexe, et dans les processeurs, on utilise le circuit additionneur/soustracteur. En soustrayant les deux nombres à comparer, on sait si l'un est plus grand ou égal à l'autre. Le soustracteur est complété par un comparateur d'égalité pour fournir au compilateur les instructions de base pour compiler les instructions

```
  if (a>b)   if (a=b)   et   if (a<b)
```

1.15 Logidule comparateur

Un circuit qui compare 4 bits doit être "cascadable". La comparaison se fait à partir des poids faibles et le premier module recoit une information "égal". Chaque module transmet plus loin un signal EQ (égal) et un signal GT (greater than). Avec les deux sorties du dernier étage et des inverseurs et une porte logique on peut générer les informations manquantes: GE (greater or equal) LT (lower than) et LE (lower or Equal).

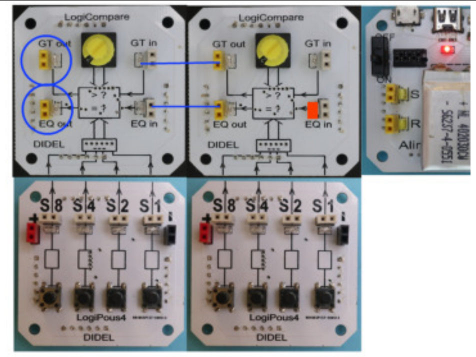


Fig14

1.16 **Programmer le logidule comparateur**

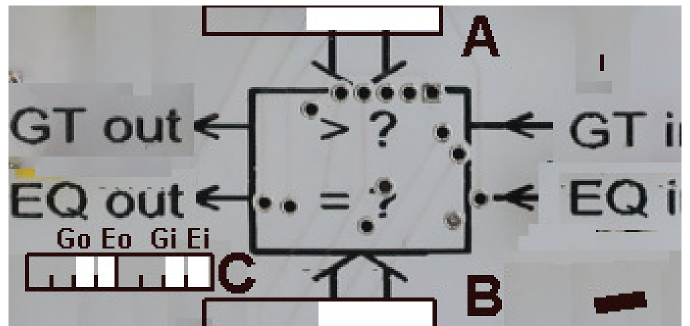
C'est ce qu'il a fallu faire pour que ce logidule fonctionne!

On a 10 entrées et 2 sorties, réparties sur 3 "ports" (registre d'entée-sortie), équivalent à des variables.

En C, on peut agir sur les bits d'une variable avec les instructions

```
set (var,noBit); clear (var,noBit);
```

Il faut programmer 3 variables. Les noms des variables et bits sont donnés dans la figure ci-contre.



(figure à refaire)

Fig 15

1.17 Circuits de parité

Dans les transmissions d'information, des erreurs peuvent se produire et des logiques plus ou moins complexes sont utilisées.

La solution la plus simple est d'ajouter un bit de parité.

Faisons l'exemple avec 4 bits. Des OU exclusifs donnent un signal qui est à "1" si le nombre de bits est pair (ou impair, est-ce que cela change?).

En réception, on câble une logique qui vérifie cette parité. Le signal doit toujours être à zéro.

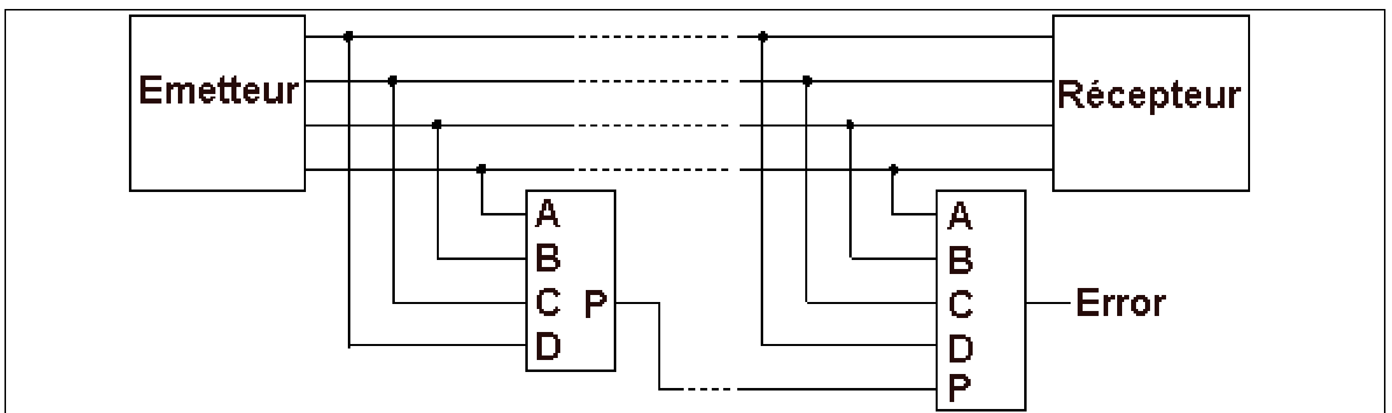


Fig 16

En ajoutant plusieurs bits, on peut utiliser des codes détecteurs et correcteurs d'erreur, nécessairement câblés avec de la logique pour ne pas ralentir les transmissions.

jdn 20200812