Architecture des systèmes numériques et informatiques

Cours 4 - Circuits logiques combinatoires

Halim Djerroud

LISV
Laboratoire d'ingénierie des systèmes de Versailles
UVSQ
université PARIS-SACLAY

révision : 2.0

Plan du cours

Circuits logiques combinatoires :

- L'additionneur binaire
- Le comparateur
- Le décodeur
- Le codeur
- Le multiplexeur et démultiplexeur

Qu'est-ce qu'un circuit combinatoire ?

Définition

Un circuit logique combinatoire est un circuit dont les sorties dépendent uniquement des valeurs actuelles des entrées.

Caractéristiques

- Pas de mémoire (pas d'état interne)
- Réponse instantanée aux changements d'entrée
- Fonction booléenne : $S = f(E_1, E_2, ..., E_n)$

Exemples

Additionneur, comparateur, multiplexeur, décodeur, encodeur

Rappel : Addition binaire

Cas 1 : 0 + 0

$$\begin{array}{rrr} & & 0 \\ + & & 0 \\ \hline & 0 & 0 \end{array}$$

Cas 2 : 0 + 1

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

Cas 3 : 1 + 0

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

Cas 4 : 1 + 1

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

Observation importante

Quand $1 + 1 = 10_2$, on obtient une retenue (carry)

Demi-additionneur (Half Adder)

Objectif

Additionner deux bits A et B **sans retenue entrante**

Table de vérité

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

- S : Somme (Sum)
- R : Retenue (Carry)

Équations logiques :

$$S = A'B + AB' = A \oplus B$$

$$R = AB$$

Limitation

Le demi-additionneur ne peut pas prendre en compte une retenue entrante !

Demi-additionneur : Analyse de la somme

Extraction de l'équation de S à partir de la table de vérité :

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

Méthode des minterms :

- Ligne 2 : $A = 0, B = 1 \rightarrow S = 1$: $A'B$
- Ligne 3 : $A = 1, B = 0 \rightarrow S = 1$: AB'
- Somme des minterms : $S = A'B + AB'$

Simplification

$S = A'B + AB' = A \oplus B$ (porte XOR)

Demi-additionneur : Analyse de la retenue

Extraction de l'équation de R à partir de la table de vérité :

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

Méthode des minterms :

- Ligne 4 : $A = 1, B = 1 \rightarrow R = 1$: AB

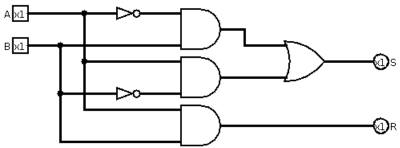
Équation

$R = AB$ (porte AND)

Demi-additionneur : Implémentation

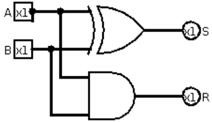
Avec portes de base

$$S = A'B + AB'$$
$$R = AB$$



Avec porte XOR $S = A \oplus B$

$$R = AB$$

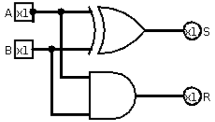


Avantage

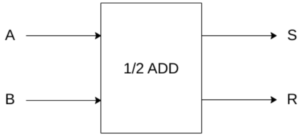
La porte XOR simplifie le circuit (moins de composants)

Demi-additionneur : Symbole

Circuit détaillé



Symbole bloc



Remarque

Le symbole bloc est utilisé pour simplifier les schémas de circuits plus complexes

Nécessité de l'additionneur complet

Problématique

Pour additionner des nombres de plusieurs bits, il faut prendre en compte la retenue de l'addition précédente

Exemple : Addition de 2 nombres de 4 bits

$$
\begin{array}{ccccccccc}
 & & R_2 & \leftarrow & R_1 & \leftarrow & R_0 & \leftarrow & \\
 & & A_3 & \uparrow & A_2 & \uparrow & A_1 & \uparrow & A_0 \\
+ & & B_3 & \uparrow & B_2 & \uparrow & B_1 & \uparrow & B_0 \\
\hline
= & R_3 & S_3 & R_2 & S_2 & R_1 & S_1 & R_0 & S_0
\end{array}
$$

Solution

L'additionneur complet possède 3 entrées : A_i, B_i et R_{i-1} (retenue entrante)

Addition multi-bits : Propagation des retenues

Schéma général de l'addition

$$
\begin{array}{cccccccccccc}
 & & \mathsf{R}_{n-2} & \leftarrow & \mathsf{R}_{n-1} & \leftarrow & \dots & \leftarrow & \mathsf{R}_0 & \leftarrow & (0) \\
 & & \mathsf{A}_{n-1} & \uparrow & \mathsf{A}_{n-2} & \uparrow & \dots & \uparrow & \mathsf{A}_1 & \uparrow & \mathsf{A}_0 \\
+ & & \mathsf{B}_{n-1} & \uparrow & \mathsf{B}_{n-2} & \uparrow & \dots & \uparrow & \mathsf{B}_1 & \uparrow & \mathsf{B}_0 \\
\hline
= & \mathsf{R}_{n-1} & \mathsf{S}_{n-1} & \mathsf{R}_{n-2} & \mathsf{S}_{n-2} & & \dots & \mathsf{R}_1 & \mathsf{S}_1 & \mathsf{R}_0 & \mathsf{S}_0
\end{array}
$$

Principe

- Position 0 : Addition de $A_0 + B_0$ (pas de retenue entrante)
- Position i : Addition de $A_i + B_i + R_{i-1}$ (avec retenue de la position précédente)
- La retenue R_i se propage vers la position suivante

Additionneur complet : Table de vérité

Table de vérité complète

A_i	B_i	R_{i-1}	S_i	R_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Observations :

- 3 entrées : A_i, B_i, R_{i-1}
- 2 sorties : S_i, R_i
- $S_i = 1$ quand nombre impair de 1
- $R_i = 1$ quand au moins deux entrées valent 1

8 combinaisons

$2^3 = 8$ lignes dans la table de vérité

Additionneur complet : Extraction des équations (Somme)

Équation de la somme S_i : **Lignes où $S_i = 1$:**

A_i	B_i	R_{i-1}	S_i
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	1

Somme des minterms :

$$\begin{aligned} S_i &= A_i'B_i'R_{i-1} + A_i'B_iR_{i-1}' + A_iB_i'R_{i-1}' + A_iB_iR_{i-1} \\ &= A_i \oplus B_i \oplus R_{i-1} \end{aligned}$$

Simplification

La somme est un XOR triple !

Additionneur complet : Extraction des équations (Retenue)

Équation de la retenue R_i : **Lignes où $R_i = 1$:**

A_i	B_i	R_{i-1}	R_i
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	1

Somme des minterms :

$$R_i = A_i' B_i R_{i-1} + A_i B_i' R_{i-1} + A_i B_i R_{i-1}' + A_i B_i R_{i-1}$$

Simplification (par tableau de Karnaugh)

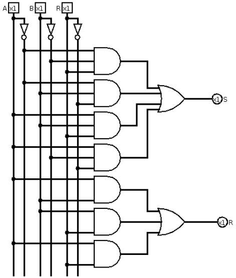
$R_i = A_i B_i + B_i R_{i-1} + A_i R_{i-1}$

Additionneur complet : Implémentation

Équations

$$S_i = A'_i B'_i R_{i-1} + A'_i B_i R'_{i-1}$$
$$+ A_i B_i R_{i-1} + A_i B'_i R'_{i-1}$$
$$R_i = A_i B_i + B_i R_{i-1} + A_i R_{i-1}$$

Circuit

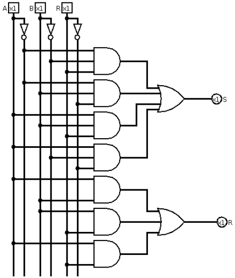


Complexité

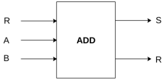
L'implémentation directe nécessite beaucoup de portes logiques

Additionneur complet : Symbole

Circuit complet



Symbole bloc

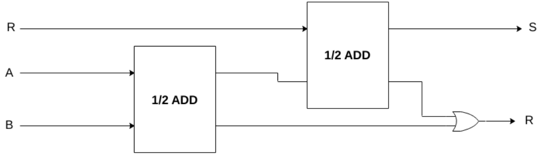


Notation

- FA : Full Adder (Additionneur Complet)
- 3 entrées : A_i, B_i, R_{in} (ou C_{in})
- 2 sorties : S_i, R_{out} (ou C_{out})

Additionneur complet avec demi-additionneurs

Construction avec 2 demi-additionneurs



Principe

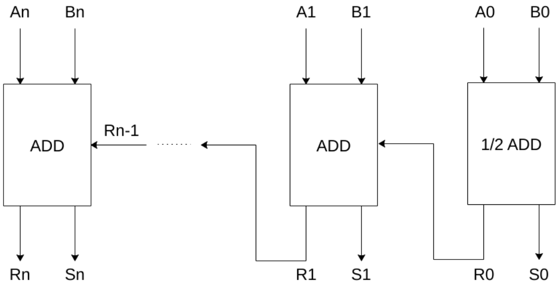
- 1er demi-additionneur : $A_i + B_i \rightarrow S_1, R_1$
- 2ème demi-additionneur : $S_1 + R_{i-1} \rightarrow S_i, R_2$
- Retenue sortante : $R_i = R_1 + R_2$ (porte OR)

Avantage

Réutilisation de blocs existants (modularité)

Additionneur n bits

Additionneur 4 bits en cascade



Architecture

- Bit 0 : Additionneur complet avec $R_{in} = 0$
- Bits 1 à n-1 : Additionneurs complets en cascade
- Les retenues se propagent de droite à gauche

Exemple numérique : Addition 4 bits

Calcul de 1011 + 0110

Position	3	2	1	0
A	1	0	1	1
B	0	1	1	0
R_{in}	0	1	1	0
S	0	0	0	1
R_{out}	1	1	1	0

Vérification :

- $1011_2 = 11_{10}$
- $0110_2 = 6_{10}$
- $10001_2 = 17_{10}$
- $11 + 6 = 17$

Débordement

Le résultat nécessite 5 bits ! La retenue finale $R_3 = 1$ indique un overflow

Comparateur : Principe

Définition

Un comparateur est un circuit qui compare deux nombres binaires et indique leur relation

Sorties d'un comparateur

- $A > B$: A est strictement supérieur à B
- $A = B$: A est égal à B
- $A < B$: A est strictement inférieur à B

Application

Utilisé dans les unités arithmétiques et logiques (ALU), les systèmes de contrôle, etc.

Comparateur 1 bit : Table de vérité

Table de vérité

A	B	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Équations logiques :

$$A > B = AB'$$

$$A = B = A'B' + AB = \overline{A \oplus B}$$

$$A < B = A'B$$

Propriété

Une seule sortie peut être active à la fois

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Comparateur 2 bits : Principe

Comparaison de $A = A_1A_0$ et $B = B_1B_0$

Méthode hiérarchique :

1. Comparer les bits de poids fort A_1 et B_1
2. Si $A_1 = B_1$, comparer les bits de poids faible A_0 et B_0
3. Sinon, le résultat est déterminé par A_1 et B_1

Équations :

$$A > B = A_1B_1' + (A_1 \odot B_1)(A_0B_0')$$
$$A = B = (A_1 \odot B_1)(A_0 \odot B_0)$$
$$A < B = A_1'B_1 + (A_1 \odot B_1)(A_0'B_0)$$

où $A \odot B = \overline{A \oplus B}$ (égalité bit à bit)

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Comparateur à 2 bits : Implémentation

Extension à n bits

Le principe se généralise à n bits en comparant bit par bit du poids fort au poids faible

Comparateur 2 bits : Exemple

Comparaison de 10 et 01

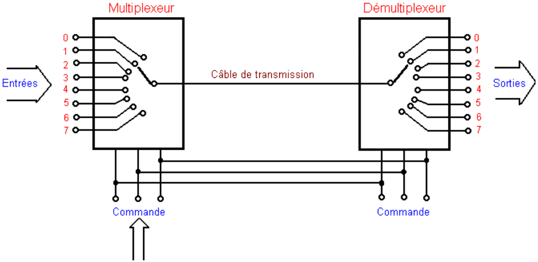
- $A = 10_2$: $A_1 = 1$, $A_0 = 0$
- $B = 01_2$: $B_1 = 0$, $B_0 = 1$

Analyse :

1. Comparaison des bits de poids fort : $A_1 = 1 > B_1 = 0$
2. Résultat immédiat : $A > B = 1$
3. Pas besoin de comparer A_0 et B_0

Vérification : $10_2 = 2_{10}$ et $01_2 = 1_{10}$ donc $2 > 1$

Multiplexeur et Démultiplexeur



Multiplexeur

Sélectionne une entrée parmi n et la dirige vers la sortie

Démultiplexeur

Dirige une entrée vers une sortie parmi n

Analogie

Le MUX est comme un aiguillage qui sélectionne une voie parmi plusieurs

Multiplexeur : Principe

Définition

Un multiplexeur à n entrées de données possède :

- 2^k entrées de données $(I_0, I_1, ..., I_{2^k-1})$
- k entrées de sélection $(S_0, S_1, ..., S_{k-1})$
- 1 sortie Y

Fonctionnement

Les entrées de sélection forment un code binaire qui désigne quelle entrée de données est acheminée vers la sortie

Notation : MUX 2^k : 1 (ex : MUX 4 :1, MUX 8 :1)

Multiplexeur 4 :1 : Table de vérité

Table de sélection

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

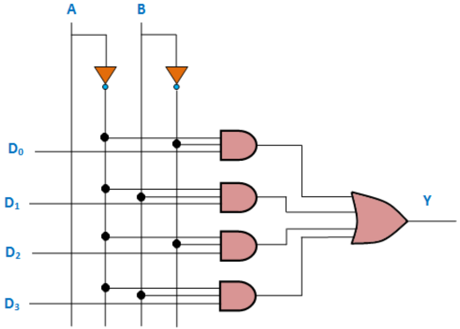
Équation :

$$Y = S_1' S_0' I_0 + S_1' S_0 I_1 + S_1 S_0' I_2 + S_1 S_0 I_3$$

Principe

Pour chaque combinaison de $S_1 S_0$, une seule porte AND est activée

Multiplexeur : Implémentation



Composants

- Portes AND pour chaque entrée (avec décodeur intégré)
- Porte OR pour combiner les sorties
- Les entrées de sélection activent une seule porte AND

Multiplexeur : Exemple d'utilisation

Sélection de données

Exemple

Un MUX 4 :1 avec :

- $I_0 = 1, I_1 = 0, I_2 = 1, I_3 = 0$
- $S_1 S_0 = 10$ (binaire $= 2$ en décimal)

Résultat : $Y = I_2 = 1$

Applications

- Routage de données
- Sélection de sources
- Implémentation de fonctions logiques
- Bus de données partagés

Démultiplexeur : Principe

Définition

Un démultiplexeur possède :

- 1 entrée de données I
- k entrées de sélection $(S_0, S_1, ..., S_{k-1})$
- 2^k sorties $(Y_0, Y_1, ..., Y_{2^k-1})$

Fonctionnement

L'entrée I est dirigée vers une seule sortie Y_j, sélectionnée par le code binaire des entrées de sélection. Toutes les autres sorties sont à 0

Notation : DEMUX 1 : 2^k (ex : DEMUX 1 :4, DEMUX 1 :8)

Démultiplexeur 1 :4 : Table de vérité

Table de sélection

S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

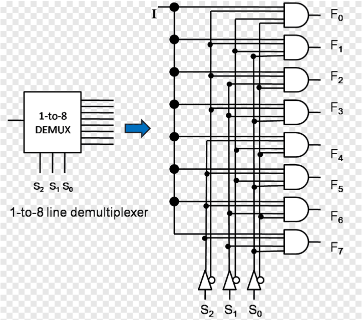
Équations :

$$Y_0 = S_1' S_0' \cdot I$$
$$Y_1 = S_1' S_0 \cdot I$$
$$Y_2 = S_1 S_0' \cdot I$$
$$Y_3 = S_1 S_0 \cdot I$$

Propriété

Une seule sortie est active (égale à I), les autres sont à 0

Démultiplexeur : Implémentation



Structure

- Une porte AND par sortie
- Les entrées de sélection (éventuellement inversées) activent une porte
- L'entrée *I* est multipliée avec le signal de sélection

Démultiplexeur : Applications

Utilisations courantes

- **Distribution de données** : Acheminer un signal vers différentes destinations
- **Adressage mémoire** : Sélectionner une ligne mémoire parmi plusieurs
- **Communication** : Routage de messages vers différents destinataires
- **Affichage** : Contrôle d'afficheurs multiplexés

Relation MUX/DEMUX

Le démultiplexeur est l'opération inverse du multiplexeur :

$$\text{MUX : plusieurs} \rightarrow \text{un}$$
$$\text{DEMUX : un} \rightarrow \text{plusieurs}$$

Décodeur : Principe

Définition

Un décodeur à n entrées et 2^n sorties active une seule sortie correspondant au code binaire en entrée

Caractéristiques

- n entrées binaires $(E_0, E_1, ..., E_{n-1})$
- 2^n sorties $(S_0, S_1, ..., S_{2^n-1})$
- Une seule sortie à 1, les autres à 0

Notation : Décodeur $n : 2^n$ (ex : 2 :4, 3 :8, 4 :16)

Application

Conversion binaire vers unaire (one-hot encoding)

Décodeur 2 :4 : Table de vérité

Table de vérité

E_1	E_0	S_0	S_1	S_2	S_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Équations :

$$S_0 = E_1' E_0'$$
$$S_1 = E_1' E_0$$
$$S_2 = E_1 E_0'$$
$$S_3 = E_1 E_0$$

Observation

Chaque sortie correspond à un minterm de la fonction

Décodeur 3 :8 : Structure

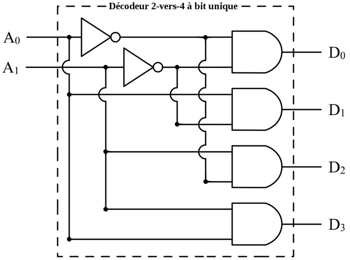


Table de vérité

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Opérations booléennes

$$D_0 = \overline{A_1} \cdot \overline{A_0}$$
$$D_1 = \overline{A_1} \cdot A_0$$
$$D_2 = A_1 \cdot \overline{A_0}$$
$$D_3 = A_1 \cdot A_0$$

Principe du décodeur 3 :8

- 3 entrées : E_2, E_1, E_0
- 8 sorties : S_0 à S_7
- $S_i = 1$ si et seulement si $E_2 E_1 E_0 = i$ (en binaire)

Décodeur : Applications

Utilisations principales

- **Adressage mémoire** : Sélection d'une case mémoire
- **Démultiplexage** : Routing de signaux
- **Afficheurs 7 segments** : Conversion BCD vers segments
- **Implémentation de fonctions** : Génération de minterms

Exemple pratique

Un décodeur 4 :16 peut adresser 16 emplacements mémoire différents avec seulement 4 lignes d'adresse

Avec validation

Certains décodeurs ont une entrée ENABLE qui active/désactive toutes les sorties

Décodeur : Exemple d'utilisation

Décodeur d'adresse mémoire

Scénario

Un système avec 4 modules mémoire :

- Décodeur 2 :4
- Entrées : $A_1 A_0$ (2 bits d'adresse)
- Sorties : $\overline{CS_0}, \overline{CS_1}, \overline{CS_2}, \overline{CS_3}$ (Chip Select, actif bas)

Fonctionnement :

- $A_1 A_0 = 00 \rightarrow \overline{CS_0} = 0$ (module 0 sélectionné)
- $A_1 A_0 = 01 \rightarrow \overline{CS_1} = 0$ (module 1 sélectionné)
- etc.

Un seul module est actif à la fois !

Codeur : Principe

Définition

Un codeur (encodeur) réalise l'opération inverse du décodeur : il convertit 2^n entrées en un code binaire de n bits

Caractéristiques

- 2^n entrées $(E_0, E_1, ..., E_{2^n-1})$
- n sorties binaires $(S_0, S_1, ..., S_{n-1})$
- Une seule entrée doit être active à la fois

Notation : Codeur 2^n : n (ex : 4 :2, 8 :3, 16 :4)

Contrainte

Le codeur standard suppose qu'une seule entrée est active. Si plusieurs entrées sont actives, le résultat est imprévisible

Codeur 4 :2 : Table de vérité

Table de vérité

E_3	E_2	E_1	E_0	S_1	S_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Équations :

$$S_0 = E_1 + E_3$$
$$S_1 = E_2 + E_3$$

Observation

$S_i = 1$ si une entrée de poids $\geq 2^i$ est active

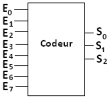
Attention

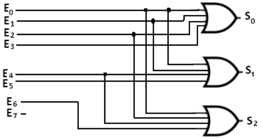
Les combinaisons avec plusieurs entrées à 1 ne sont pas définies

Codeur : Implémentation

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	S_0	S_1	S_2
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$S_0 = E_0 + E_1 + E_2 + E_3$
$S_1 = E_0 + E_1 + E_4 + E_5$
$S_2 = E_0 + E_2 + E_4 + E_6$





Structure simple

- Portes OR pour chaque bit de sortie
- Chaque sortie combine les entrées correspondantes

Codeur prioritaire

Problème du codeur simple

Que se passe-t-il si plusieurs entrées sont actives simultanément ?

Solution : Codeur prioritaire

- Attribue une priorité à chaque entrée
- Encode l'entrée active de plus haute priorité
- Généralement : E_i a priorité sur E_j si $i > j$

Exemple

Si $E_3 = 1$ et $E_1 = 1$: le codeur prioritaire encode E_3 (sortie $= 11$)

Sortie supplémentaire

Un codeur prioritaire a souvent une sortie VALID indiquant si au moins une entrée est active

Codeur prioritaire 4 :2 : Table de vérité

Table avec priorités

E_3	E_2	E_1	E_0	S_1	S_0	VALID
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Notation

- X = Don't Care (la valeur n'importe pas)
- VALID = 1 si au moins une entrée est active

Règle

E_3 a la priorité la plus haute, E_0 la plus basse

Codeur : Applications

Utilisations principales

- **Claviers** : Conversion touche pressée $\rightarrow$ code
- **Compression de données** : Réduction du nombre de lignes
- **Interruptions** : Gestion des priorités d'interruption
- **Détection de position** : Trouver le bit à 1 le plus significatif

Exemple : Clavier matriciel

Un clavier 16 touches peut être encodé en 4 bits (4 :2 pour lignes + 4 :2 pour colonnes)

En pratique

Les codeurs prioritaires sont beaucoup plus utilisés que les codeurs simples

Récapitulatif

Circuits étudiés

- **Additionneur** : Demi-additionneur, additionneur complet, additionneur n bits
- **Comparateur** : Compare deux nombres binaires
- **Multiplexeur** : Sélectionne une entrée parmi plusieurs ($2^n : 1$)
- **Démultiplexeur** : Route une entrée vers plusieurs sorties ($1 : 2^n$)
- **Décodeur** : Convertit code binaire en signal unaire ($n : 2^n$)
- **Codeur** : Convertit signal unaire en code binaire ($2^n : n$)

Points clés

- Tables de vérité → Équations logiques → Circuits
- Modularité : réutilisation de blocs (ex : FA avec HA)
- Extensibilité : passages de n à m bits