Algèbre de Boole

Série Complète d'Apprentissage

De la Théorie aux Applications Pratiques

10 Fiches Pédagogiques

Pour Radioamateurs et Passionnés d'Électronique

73 de F4HXN

https://f4hxn.fr

Table des Matières

FICHE 8 : Machines à États Finis

FSM de Moore et Mealy, diagrammes d'états

FICHE 9 : Applications en Radio Amateur

CTCSS, DCS, relais, DTMF, modes numériques

FICHE 10 : Programmation et Logique

Opérateurs bit à bit, FPGA, VHDL, simulation

FICHE 1

Le Système Binaire

Comprendre le langage fondamental des ordinateurs et de l'électronique numérique

Qu'est-ce que le Système Binaire ?

Le système binaire est un **système de numération en base 2**, utilisant uniquement deux chiffres : **0 et 1**. C'est le système naturel de l'électronique numérique, où un signal électrique peut être soit présent (1) soit absent (0).

💡 Pourquoi le binaire ?

L'électronique numérique utilise le binaire car il est facile de distinguer deux états électriques (tension haute/basse) avec une grande fiabilité, même en présence de bruit. Contrairement au système décimal (base 10) que nous utilisons quotidiennement, le binaire est parfaitement adapté au fonctionnement physique des circuits électroniques.

Comparaison des systèmes

- **Système décimal (base 10) :** utilise 10 chiffres (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- **Système binaire (base 2) :** utilise 2 chiffres (0, 1)

- **Système octal (base 8) :** utilise 8 chiffres (0 à 7)
- **Système hexadécimal (base 16) :** utilise 16 symboles (0 à 9, A à F)

Le Bit : Unité Fondamentale

Le **bit** (contraction de "binary digit") est la plus petite unité d'information en informatique. Un bit peut prendre la valeur **0** ou **1**.

📏 **Hiérarchie des unités :**

- **1 bit** = une valeur binaire (0 ou 1)
- **1 nibble** = 4 bits
- **1 octet (byte)** = 8 bits
- **1 mot (word)** = 16, 32 ou 64 bits selon l'architecture

Capacité de représentation

Avec **n bits**, on peut représenter **2^n valeurs différentes** :

Nombre de bits	Valeurs possibles	Plage (0 à ...)
1 bit	$2^1 = 2$	0 à 1
2 bits	$2^2 = 4$	0 à 3
3 bits	$2^3 = 8$	0 à 7
4 bits	$2^4 = 16$	0 à 15
8 bits	$2^8 = 256$	0 à 255
16 bits	$2^{16} = 65\,536$	0 à 65 535
32 bits	$2^{32} = 4\,294\,967\,296$	0 à 4 294 967 295

Conversion Binaire ↔ Décimal

Table de correspondance (0 à 15)

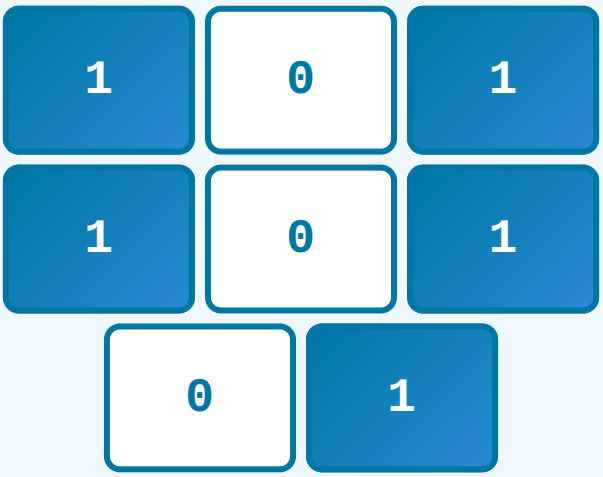
Décimal	Binaire (4 bits)	Décimal	Binaire (4 bits)
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Du Binaire vers le Décimal

Chaque position dans un nombre binaire représente une **puissance de 2** :

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Exemple : Convertir 10110101_2 en décimal



$$1\times128 + 0\times64 + 1\times32 + 1\times16 + 0\times8 + 1\times4 + 0\times2 + 1\times1$$

$$128 + 32 + 16 + 4 + 1 = 181_{10}$$

$$10110101_2 = 181_{10}$$

Du Décimal vers le Binaire

Pour convertir un nombre décimal en binaire, on utilise la **méthode des divisions successives par 2** :

Exemple : Convertir 45_{10} en binaire

1. 45 ÷ 2 = 22 reste **1** ← bit de poids faible (LSB)
2. 22 ÷ 2 = 11 reste **0**
3. 11 ÷ 2 = 5 reste **1**
4. 5 ÷ 2 = 2 reste **1**
5. 2 ÷ 2 = 1 reste **0**
6. 1 ÷ 2 = 0 reste **1** ← bit de poids fort (MSB)

On lit les restes de bas en haut : 101101_2

$45_{10} = 101101_2$

Opérations Arithmétiques en Binaire

Addition Binaire

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10 \text{ (0 avec retenue de 1)}$$

Exemple :

```
  1011  (11)
+  110  (6)
------
 10001  (17)
```

Soustraction Binaire

```
0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
10 - 1 = 1 (avec emprunt)
```

Exemple :

```
  1101  (13)
-  101  (5)
------
  1000  (8)
```

Multiplication Binaire

```
0 × 0 = 0
0 × 1 = 0
1 × 0 = 0
1 × 1 = 1
```

Exemple :

```
   101  (5)
 ×  11  (3)
 -----
   101
+  101
 -----
  1111  (15)
```

Division Binaire

Similaire à la division décimale, mais avec les règles binaires.

Applications Pratiques

En Radio Amateur

Codes de sous-tonalités CTCSS

Les fréquences CTCSS sont souvent programmées en binaire dans les transceiveurs numériques. Par exemple, la fréquence 88.5 Hz pourrait être codée sur 8 bits.

Modes numériques

Les modes comme **FT8, WSPR, PSK31** transmettent des données binaires. Chaque caractère est converti en binaire avant la transmission.

Adressage mémoire

Les canaux mémoire d'un transceiver sont numérotés en binaire en interne. Un appareil avec 100 mémoires utilise 7 bits ($2^7 = 128$ positions).

En Informatique

Représentation des données :

- **Texte :** Chaque caractère est codé en binaire (ASCII, Unicode)
- **Images :** Chaque pixel est représenté par des valeurs binaires (RGB)
- **Son :** L'amplitude du signal est échantillonnée et convertie en binaire
- **Instructions processeur :** Chaque instruction machine est un code binaire

Masques et Opérations Bit à Bit

Les opérations bit à bit permettent de manipuler des bits individuels :

Décalages (Shifts)

Décalage à gauche (Left Shift) : nombre × 2

```
00101101 << 1 = 01011010
```

Équivalent à multiplier par 2

Décalage à droite (Right Shift) : nombre ÷ 2

```
00101100 >> 1 = 00010110
```

Équivalent à diviser par 2

Masquage de bits

Extraire un bit spécifique

Pour lire le bit 3 d'un octet :

```
Valeur : 10110101
Masque : 00001000 (2³)
Résultat AND : 00000000 → bit = 0
```

Mettre un bit à 1

Pour activer le bit 5 :

```
Valeur : 10010101
Masque : 00100000 (2⁵)
Résultat OR : 10110101
```

Exercices Pratiques

Exercice 1 : Conversion

Convertir en décimal :

- 11010_2 = ?
- 101011_2 = ?
- 11111111_2 = ?

Voir les réponses

- $11010_2 = 26_{10}$
- $101011_2 = 43_{10}$
- $11111111_2 = 255_{10}$

Exercice 2 : Addition

Calculer :

- $1010_2 + 0111_2$ = ?
- $1101_2 + 1011_2$ = ?

Voir les réponses

- $1010_2 + 0111_2 = 10001_2$ (10 + 7 = 17)
- $1101_2 + 1011_2 = 11000_2$ (13 + 11 = 24)

Exercice 3 : Puissances de 2

Combien de valeurs peut-on représenter avec :

- 5 bits ?
- 12 bits ?
- 20 bits ?

Voir les réponses

- 5 bits : $2^5 = 32$ valeurs (0 à 31)
- 12 bits : $2^{12} = 4\,096$ valeurs
- 20 bits : $2^{20} = 1\,048\,576$ valeurs

Points Clés à Retenir

- Le binaire utilise uniquement **0 et 1**
- Chaque position représente une **puissance de 2**
- Avec **n bits**, on peut représenter **2^n valeurs**
- L'addition binaire suit des règles simples avec retenues
- Les décalages multiplient ou divisent par 2
- Le binaire est le langage natif de tous les circuits numériques

73 de F4HXN - Prochaine fiche : L'algèbre de Boole ! 📡

FICHE 2

Les Bases de l'Algèbre de Boole

Découvrez les fondements mathématiques de l'électronique numérique et de l'informatique moderne

Introduction à l'Algèbre de Boole

L'algèbre de Boole, développée par le mathématicien britannique **George Boole** en 1854, est un système algébrique qui manipule des valeurs binaires. Cette algèbre est aujourd'hui fondamentale pour l'électronique numérique, l'informatique et les télécommunications.

💡 **Principe fondamental :** En algèbre de Boole, les variables ne peuvent prendre que deux valeurs possibles : **0 (FAUX)** ou **1 (VRAI)**.

Ces deux états peuvent représenter :

- **En logique :** Vrai / Faux
- **En électronique :** Tension haute / Tension basse
- **En informatique :** Bit à 1 / Bit à 0
- **En radio amateur :** Signal présent / Signal absent

Les Opérations de Base

1. L'opération ET (AND) - Conjonction

L'opération ET retourne **VRAI (1)** uniquement si **toutes** les entrées sont vraies.

A · B ou A ∧ B ou A AND B

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	**1**

Exemple radio amateur

Un relais ne s'active que si le signal CTCSS est correct **ET** que la porteuse est présente.

2. L'opération OU (OR) - Disjonction

L'opération OU retourne **VRAI (1)** si **au moins une** des entrées est vraie.

A + B ou A ∨ B ou A OR B

A	B	A + B
0	0	0
0	1	**1**
1	0	**1**
1	1	**1**

Exemple radio amateur

Une alarme se déclenche si la température est trop élevée **OU** si la tension d'alimentation est trop faible.

3. L'opération NON (NOT) - Négation

L'opération NON inverse la valeur de l'entrée.

Ā ou ¬A ou NOT A

A	Ā
0	**1**
1	0

Exemple radio amateur

Un système de coupure coupe l'émetteur (sortie à 0) quand l'interrupteur PTT est **NON** activé.

Les Opérations Composées

4. L'opération NON-ET (NAND)

L'opération NAND est l'inverse du ET. C'est une opération **universelle** : on peut créer toutes les autres portes logiques uniquement avec des NAND.

$$A \text{ NAND } B = A \cdot B$$

A	B	A NAND B
0	0	**1**
0	1	**1**
1	0	**1**
1	1	0

5. L'opération NON-OU (NOR)

L'opération NOR est l'inverse du OU. C'est également une opération universelle.

```
A NOR B = A + B
```

A	B	A NOR B
0	0	**1**
0	1	0
1	0	0
1	1	0

6. L'opération OU Exclusif (XOR)

L'opération XOR retourne VRAI si les entrées sont **différentes**.

```
A ⊕ B ou A XOR B
```

A	B	A ⊕ B
0	0	0
0	1	**1**
1	0	**1**
1	1	0

Utilisation en cryptographie

Le XOR est fondamental en cryptographie pour chiffrer/déchiffrer des données, car A ⊕ B ⊕ B = A

Les Lois de l'Algèbre de Boole

Lois de commutativité

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

Lois d'associativité

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

$$A + (B + C) = (A + B) + C$$

Lois de distributivité

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Lois d'identité

$$A \cdot 1 = A$$

$$A + 0 = A$$

Lois de complémentation

$$A \cdot \bar{A} = 0$$

$$A + \bar{A} = 1$$

Lois de De Morgan

🎓 **Très importantes !** Ces lois permettent de transformer les expressions logiques.

$$A \cdot B = \bar{A} + \bar{B}$$

$$A + B = \bar{A} \cdot \bar{B}$$

Applications Pratiques

Processeurs

Tous les calculs dans un processeur sont effectués par des milliards de portes logiques.

Circuits intégrés

Les circuits CMOS utilisent l'algèbre de Boole pour créer des fonctions complexes.

Télécommunications

Encodage, décodage, et correction d'erreurs dans les transmissions numériques.

Intelligence artificielle

Les réseaux de neurones artificiels utilisent des opérations logiques.

Radio numérique

Modes numériques (FT8, WSPR, etc.) basés sur le traitement logique des signaux.

Sécurité

Algorithmes de chiffrement et de hachage utilisent intensivement le XOR.

Exemple Pratique : Circuit de Sécurité

Imaginons un système de sécurité pour un émetteur radio qui doit respecter les conditions suivantes :

Cahier des charges :

- L'émetteur peut fonctionner SI l'alimentation est OK **ET** la température est normale
- Une alarme sonne SI la température est élevée **OU** si le ROS est mauvais
- Le ventilateur s'active SI l'émetteur fonctionne **ET** que la puissance > 50W

Variables

- **A** = Alimentation OK (1) ou KO (0)
- **T** = Température normale (1) ou élevée (0)
- **R** = ROS bon (1) ou mauvais (0)
- **P** = Puissance > 50W (1) ou ≤ 50W (0)

Équations logiques

$$\text{Émission} = A \cdot T$$

$$\text{Alarme} = \bar{T} + \bar{R}$$

$$\text{Ventilateur} = \text{Émission} \cdot P = A \cdot T \cdot P$$

Simplification

Grâce aux lois de De Morgan, on peut simplifier :

$$\text{Alarme} = \bar{T} + \bar{R} = T \cdot R$$

Ce qui signifie : "L'alarme sonne si ce n'est PAS le cas que (température ET ROS sont bons)"

Conclusion

L'algèbre de Boole est un outil mathématique puissant et élégant qui constitue la base de toute l'électronique numérique moderne. Sa simplicité apparente cache une grande richesse et permet de concevoir des systèmes extrêmement complexes.

📚 **Pour aller plus loin :** L'algèbre de Boole s'étend à des concepts plus avancés comme les tableaux de Karnaugh pour la simplification de circuits, les bascules et les mémoires, et les machines à états finis utilisées dans les protocoles de communication numérique.

73 de F4HXN - Prochaine fiche : Les portes logiques ! 📡

FICHE 3

Les Portes Logiques et Circuits

Découvrez les composants électroniques qui réalisent les opérations booléennes

Qu'est-ce qu'une Porte Logique ?

Une **porte logique** est un circuit électronique qui réalise une opération booléenne. Elle possède une ou plusieurs entrées et une sortie dont l'état dépend des états des entrées selon une fonction logique définie.

💡 Principe de fonctionnement :

Les portes logiques sont constituées de transistors qui agissent comme des interrupteurs électroniques. Un transistor peut être dans deux états : passant (1) ou bloqué (0). En combinant plusieurs transistors, on crée des fonctions logiques complexes.

Représentation des portes logiques

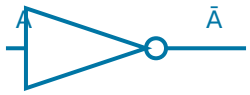
Il existe deux normes principales pour représenter les portes logiques :

- **Norme américaine (ANSI/MIL) :** utilise des formes géométriques distinctives
- **Norme européenne (IEC) :** utilise des rectangles avec des symboles

Dans cette fiche, nous utiliserons principalement la norme américaine, plus répandue en électronique amateur.

Les Portes Logiques Fondamentales

1. Porte NON (NOT / Inverseur)



Symbole : Porte NON

Fonction : Inverse l'état de l'entrée

Équation : Sortie = Ā

2. Porte ET (AND)

Symbole : Porte ET

Fonction : Sortie à 1 si TOUTES les entrées sont à 1

Équation : Sortie = A · B

3. Porte OU (OR)

Symbole : Porte OU

Fonction : Sortie à 1 si AU MOINS UNE entrée est à 1

Équation : Sortie = A + B

4. Porte NON-ET (NAND)

Fonction : ET suivi d'une inversion (porte universelle)

Équation : Sortie = A · B

⭐ **Porte universelle :** La NAND est dite "universelle" car on peut créer toutes les autres portes logiques uniquement avec des NAND. C'est la raison pour laquelle le circuit intégré 7400 (quad NAND) est l'un des plus utilisés.

5. Porte NON-OU (NOR)

Fonction : OU suivi d'une inversion (porte universelle)

Équation : Sortie = A + B

6. Porte OU Exclusif (XOR)

Fonction : Sortie à 1 si les entrées sont DIFFÉRENTES

Équation : Sortie = A ⊕ B

Application pratique : Détecteur de changement

Une porte XOR peut détecter si deux signaux sont différents. En radio amateur, on peut l'utiliser pour comparer un signal reçu avec un signal de référence et détecter les variations.

Tableau Récapitulatif des Portes

Porte	Symbole	Équation	Sortie = 1 si...
NOT	Triangle + cercle	$\bar{A}$	A = 0
AND	Forme D	$A \cdot B$	A = 1 ET B = 1
OR	Forme bouclier	$A + B$	A = 1 OU B = 1
NAND	AND + cercle	$A \cdot B$	PAS (A = 1 ET B = 1)
NOR	OR + cercle	$A + B$	A = 0 ET B = 0
XOR	OR avec double entrée	$A \oplus B$	$A \neq B$

Familles Logiques

Les portes logiques sont fabriquées selon différentes technologies, appelées **familles logiques**. Chaque famille a ses propres caractéristiques de vitesse, consommation et niveaux de tension.

Principales familles

TTL

Transistor-Transistor Logic

Tension :	5V
Vitesse :	Moyenne
Conso :	Élevée

CMOS

Complementary MOS

Tension :	3-15V
Vitesse :	Variable
Conso :	Très faible

74HC

High-Speed CMOS

Tension :	2-6V
Vitesse :	Rapide
Conso :	Faible

74LS

Low-Power Schottky

Tension :	5V
Vitesse :	Rapide
Conso :	Moyenne

🎯 Choix de la famille :

- **74HC** : Excellent choix pour la plupart des projets amateurs (faible consommation, bonne vitesse)
- **74LS** : Pour compatibilité avec circuits anciens
- **CMOS 4000** : Pour très faible consommation (batteries)
- **74AC/ACT** : Pour applications haute fréquence

Circuits Intégrés Classiques

Les portes logiques sont regroupées dans des **circuits intégrés** (CI ou IC en anglais). Voici les plus courants en format DIP (Dual In-line Package) à 14 ou 16 broches :

7400 Quad NAND

4 portes NAND à 2 entrées

La porte universelle par excellence. Indispensable pour tout projet de logique combinatoire.

7402 Quad NOR

4 portes NOR à 2 entrées

Autre porte universelle, très utile pour certains montages.

7404 Hex NOT

6 inverseurs

Pour inverser des signaux logiques.

7408 Quad AND

4 portes AND à 2 entrées

Pour fonctions ET simples.

7432 Quad OR

4 portes OR à 2 entrées

Pour fonctions OU simples.

7486 Quad XOR

4 portes XOR à 2 entrées

Comparateurs, détecteurs de parité.

7410 Triple NAND-3

3 portes NAND à 3 entrées

Quand plus de 2 entrées sont nécessaires.

7411 Triple AND-3

3 portes AND à 3 entrées

ET à 3 entrées.

⚠️ **Attention :** Les circuits intégrés logiques nécessitent :

- Condensateur de découplage (100nF) entre Vcc et GND, proche du CI
- Respect strict de la polarité d'alimentation
- Ne jamais laisser d'entrée CMOS "en l'air" (flottante)
- Protection contre les décharges électrostatiques (ESD) pour les CMOS

◌ Caractéristiques Électriques Importantes

1. Fan-out (Facteur de charge)

Le **fan-out** est le nombre maximal d'entrées qu'une sortie peut piloter sans dégrader le signal.

Exemple

Un 7400 TTL a un fan-out de 10 : une sortie peut piloter jusqu'à 10 entrées TTL standard.

Un 74HC00 CMOS peut théoriquement piloter des centaines d'entrées (très haute impédance d'entrée).

2. Temps de Propagation

Le **temps de propagation** (t_{pd}) est le délai entre un changement d'entrée et la réponse de la sortie.

Famille	Temps de propagation	Fréquence max
74LS	~10 ns	~25 MHz
74HC	~8 ns	~30 MHz
74AC	~4 ns	~100 MHz
CMOS 4000	~60 ns	~5 MHz

3. Niveaux Logiques

Les niveaux de tension définissant les états logiques varient selon la famille :

Pour TTL 5V :

- Niveau BAS (0) : 0V à 0.8V
- Niveau HAUT (1) : 2V à 5V
- Zone interdite : 0.8V à 2V

Pour CMOS 5V :

- Niveau BAS (0) : 0V à 1.5V
- Niveau HAUT (1) : 3.5V à 5V
- Zone interdite : 1.5V à 3.5V

Lire une Datasheet (Fiche Technique)

La **datasheet** est le document indispensable pour utiliser correctement un circuit intégré. Voici les sections importantes :

Sections essentielles d'une datasheet

1. Description générale

- Type de circuit (quad NAND, etc.)
- Famille logique (TTL, CMOS, etc.)
- Boîtier disponible (DIP, SOIC, etc.)

2. Brochage (Pin Configuration)

- Numéro et fonction de chaque broche
- Repère de la broche 1 (encoche ou point)
- Broches d'alimentation (Vcc et GND)

3. Caractéristiques électriques maximales

- Tension d'alimentation (min/typ/max)
- Courant de sortie maximal
- Puissance dissipée maximale
- Température de fonctionnement

4. Caractéristiques de fonctionnement

- Niveaux logiques (VIL, VIH, VOL, VOH)
- Temps de propagation
- Consommation électrique

5. Table de vérité

- Comportement logique du circuit

🔍 Où trouver les datasheets ?

- Site du fabricant (Texas Instruments, NXP, etc.)
- Distributeurs (Mouser, Digi-Key, Farnell)
- AllDataSheet.com
- DatasheetCatalog.com

◌ Projet Pratique : Circuit Simple

Réaliser un OU Exclusif avec des NAND

Objectif : construire une fonction XOR uniquement avec des portes NAND (démonstration de la porte universelle).

Matériel nécessaire

- 1× Circuit intégré 7400 (quad NAND) ou 74HC00
- 1× Plaque d'essai (breadboard)
- 2× Interrupteurs ou boutons poussoirs
- 1× LED + résistance 330Ω
- Alimentation 5V
- Fils de connexion
- Condensateur 100nF (découplage)

Équation XOR avec NAND

$A \oplus B = (A \cdot (A \cdot B)) \cdot (B \cdot (A \cdot B))$

Il faut **4 portes NAND** pour réaliser un XOR.

Étapes :

1. NAND1 : calculer $A \cdot B$
2. NAND2 : calculer $A \cdot$ (sortie NAND1)
3. NAND3 : calculer $B \cdot$ (sortie NAND1)
4. NAND4 : calculer (sortie NAND2) $\cdot$ (sortie NAND3)

🎓 Vérification :

Testez toutes les combinaisons d'entrées :

- A=0, B=0 → LED éteinte (0)
- A=0, B=1 → LED allumée (1)
- A=1, B=0 → LED allumée (1)
- A=1, B=1 → LED éteinte (0)

C'est bien le comportement d'un XOR ! ✓

Applications en Radio Amateur

1. Séquenceur d'émission

Problème

Lors du passage en émission, il faut activer les composants dans le bon ordre : d'abord l'accordeur d'antenne, puis l'amplificateur, enfin l'émetteur.

Solution avec portes logiques

Utilisation de portes AND pour créer des délais logiques et assurer la séquence correcte. Chaque étage ne s'active que si l'étage précédent est prêt (signal OK) ET que le PTT est activé.

2. Détecteur de tonalité CTCSS

Application

Un décodeur CTCSS utilise des portes XOR pour comparer la fréquence reçue avec une référence locale. Quand les signaux sont en phase, le XOR produit un niveau continu permettant l'ouverture du squelch.

3. Commutateur d'antenne logique

Fonctionnement

Des portes logiques contrôlent des relais pour commuter entre plusieurs antennes selon la bande sélectionnée. Un encodeur binaire (3 bits) permet de sélectionner jusqu'à 8 antennes différentes.

4. Interface CAT (Computer Aided Transceiver)

Utilité

Les portes logiques sont utilisées dans les interfaces RS-232 pour adapter les niveaux de tension entre l'ordinateur (±12V) et le transceiver (0-5V). Le circuit MAX232 contient des inverseurs et des amplificateurs de niveau.

Points Clés à Retenir

- Les **portes logiques** réalisent physiquement les opérations booléennes
- Il existe **deux normes de symboles** (américaine et européenne)
- **NAND et NOR** sont des portes universelles
- Les **familles logiques** (TTL, CMOS, HC, LS) ont des caractéristiques différentes
- Le **74HC** est un excellent choix pour les projets amateurs
- Toujours consulter la **datasheet** avant utilisation
- Ne jamais oublier les **condensateurs de découplage**
- Respecter le **fan-out** et les niveaux logiques

73 de F4HXN - Prochaine fiche : Les systèmes de numération ! 📡

FICHE 4

Les Systèmes de Numération

Octal, hexadécimal, BCD et autres systèmes pour représenter les nombres

Introduction aux Bases de Numération

Un **système de numération** (ou base) est une méthode pour représenter des nombres. La base détermine combien de chiffres différents sont utilisés.

💡 Principe général :

Dans un système en base **b**, on utilise **b chiffres** différents (de 0 à b-1). Chaque position représente une puissance de la base.

Exemple : En base 10, on utilise 10 chiffres (0-9) et chaque position représente une puissance de 10 (unités, dizaines, centaines, etc.)

Récapitulatif des bases courantes

Base	Nom	Chiffres utilisés	Suffixe	Usage principal
2	Binaire	0, 1	$_2$ ou 0b	Électronique, informatique
8	Octal	0, 1, 2, 3, 4, 5, 6, 7	$_8$ ou 0o	Permissions Unix, anciens systèmes
10	Décimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	$_{10}$ (souvent omis)	Usage courant
16	Hexadécimal	0-9, A-F	$_{16}$ ou 0x	Adresses mémoire, couleurs

8️⃣ Le Système Octal (Base 8)

Le système octal utilise **8 chiffres** : 0, 1, 2, 3, 4, 5, 6, 7

🎯 Avantage de l'octal :

Chaque chiffre octal correspond exactement à **3 bits** en binaire, ce qui simplifie grandement les conversions.

Correspondance Octal ↔ Binaire

Octal	Binaire (3 bits)	Décimal
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

Conversion Binaire → Octal

Méthode rapide

Grouper les bits par 3 en partant de la droite, puis convertir chaque groupe :

$$\text{Binaire : } 110101011_2$$

Grouper par 3 : **110 101 011**

$$6\ 5\ 3 = 653_8$$

$$110101011_2 = 653_8$$

Applications de l'octal

Permissions Unix/Linux

Les droits d'accès aux fichiers sont souvent exprimés en octal :

```
chmod 755 fichier.txt
```

- **7** (111) = lecture + écriture + exécution pour le propriétaire
- **5** (101) = lecture + exécution pour le groupe
- **5** (101) = lecture + exécution pour les autres

Le Système Hexadécimal (Base 16)

Le système hexadécimal utilise **16 symboles** :

0 1 2 3 4 5 6 7 8 9 A B C D E F

A = 10

B = 11

C = 12

D = 13

E = 14

F = 15

🎯 Avantage de l'hexadécimal :

Chaque chiffre hexadécimal correspond exactement à **4 bits** (un nibble), ce qui permet de représenter un octet (8 bits) avec seulement 2 chiffres hexa.

Table de Correspondance Complète (0-255)

Décimal	Hexadécimal	Binaire (8 bits)
0	00	00000000
15	0F	00001111
16	10	00010000
31	1F	00011111
32	20	00100000
127	7F	01111111
128	80	10000000
255	FF	11111111

Conversion Binaire ↔ Hexadécimal

Binaire → Hexa : Grouper par 4

$$\text{Binaire : } 11010110101111_2$$

Grouper par 4 : **0011 0101 1010 1111**

$$3\ 5\ A\ F = 35AF_{16}$$

$$11010110101111_2 = 35AF_{16}$$

Hexa → Binaire : Convertir chaque chiffre

$$\text{Hexadécimal : } 2F8A_{16}$$

2 = 0010 | F = 1111 | 8 = 1000 | A = 1010

$$0010111110001010_2$$

Applications de l'hexadécimal

Adresses mémoire

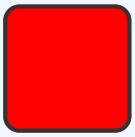
Les adresses en informatique sont exprimées en hexadécimal :

`0x7FFF2AC4` `0xDEADBEEF`

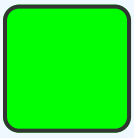
Plus compact que le binaire et plus lisible !

Couleurs RGB

Les couleurs web sont codées en hexadécimal (RR GG BB) :



#FF0000 = Rouge pur



#00FF00 = Vert pur

#0076A5 = Bleu F4HXN !

Adresses MAC en réseau

Les adresses MAC sont exprimées en hexa :

`00:1A:2B:3C:4D:5E`

Chaque paire représente un octet (8 bits)

Conversions Entre Bases

Méthode Générale : Base quelconque → Décimal

Formule

Multiplier chaque chiffre par la puissance correspondante de la base :

$$\text{Nombre} = d_{(n)} \times \text{base}^{n} + d_{(n-1)} \times \text{base}^{n-1} + \ldots + d_{1} \times \text{base}^{1} + d_{0} \times \text{base}^{0}$$

Exemple 1 : $2A3_{16}$ → Décimal

1. Identifier les chiffres : 2, A (=10), 3
2. Appliquer les puissances de 16 :

$$2\times16^{2} + 10\times16^{1} + 3\times16^{0}$$

3. Calculer :

$$2\times256 + 10\times16 + 3\times1 = 512 + 160 + 3 = 675_{10}$$

Exemple 2 : 157_8 → Décimal

$$1\times8^2 + 5\times8^1 + 7\times8^0 = 64 + 40 + 7 = 111_{10}$$

Décimal → Base quelconque

Méthode des divisions successives

Convertir 234_{10} en hexadécimal :

1. 234 ÷ 16 = 14 reste **10 (A)** ← chiffre de poids faible
2. 14 ÷ 16 = 0 reste **14 (E)** ← chiffre de poids fort

Lire de bas en haut : **EA_{16}**

Vérification : 14×16 + 10 = 224 + 10 = 234 ✓

Tableau de Conversion Rapide

Décimal	Binaire	Octal	Hexadécimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
8	1000	10	8
10	1010	12	A
15	1111	17	F
16	10000	20	10
32	100000	40	20
64	1000000	100	40
255	11111111	377	FF

Le Code BCD (Binary Coded Decimal)

Le **BCD** code chaque chiffre décimal sur 4 bits. C'est un compromis entre binaire pur et décimal.

💡 **Principe :**

Chaque chiffre décimal (0-9) est représenté par son équivalent binaire sur 4 bits.

Attention : Les valeurs 1010 à 1111 (10-15) ne sont jamais utilisées en BCD !

Table BCD

Décimal	BCD (4 bits)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Exemple : Coder 147 en BCD

Chaque chiffre séparément :

```
1 → 0001
4 → 0100
7 → 0111
```

147_{10} = **0001 0100 0111** (BCD)

⚠️ **Différence avec binaire pur :**

147_{10} en binaire = **10010011** (8 bits)

147_{10} en BCD = **0001 0100 0111** (12 bits)

Le BCD utilise plus de bits mais simplifie l'affichage décimal !

Applications du BCD

Afficheurs 7 segments

Les décodeurs BCD vers 7 segments (comme le 7447) prennent directement une entrée BCD pour commander un afficheur.

Calculs financiers

Le BCD évite les erreurs d'arrondi en virgule flottante. Utilisé dans les systèmes bancaires et les calculatrices.

Afficheurs de fréquence

Les fréquencemètres et transceiveurs avec affichage numérique utilisent souvent du BCD pour commander les afficheurs LED ou LCD.

Le Code Gray

Le **code Gray** (ou code binaire réfléchi) est un système où deux valeurs consécutives ne diffèrent que d'un seul bit.

🎯 Avantage majeur :

Élimine les erreurs de transition ! Quand un compteur passe de 3 à 4 en binaire, tous les bits changent (011→100). En Gray, un seul bit change.

Table de Conversion (4 bits)

Décimal	Binaire	Code Gray	Bits changés
0	0000	0000	-
1	0001	0001	1
2	0010	0011	1
3	0011	0010	1
4	0100	0110	1
5	0101	0111	1
6	0110	0101	1
7	0111	0100	1
8	1000	1100	1

Applications du code Gray

◌ Encodeurs rotatifs

Les encodeurs de position utilisent le code Gray pour éviter les lectures erronées lors des transitions.

Tableaux de Karnaugh

Les lignes et colonnes des tableaux de Karnaugh (simplification logique) utilisent l'ordre Gray pour faciliter le regroupement des cases adjacentes.

Conversion A/N

Certains convertisseurs analogique-numérique utilisent le Gray pour minimiser les erreurs pendant la conversion.

Représentation des Nombres Signés

Pour représenter des nombres négatifs en binaire, plusieurs méthodes existent :

1. Signe et Valeur Absolue

Le bit de poids fort indique le signe (0 = positif, 1 = négatif).

Sur 8 bits :

- **0**0101010 = +42
- **1**0101010 = -42

Problème : Deux représentations du zéro (+0 et -0) !

2. Complément à 1

Inverser tous les bits pour obtenir le négatif.

Exemple :

- +42 = 00101010
- -42 = 11010101 (tous les bits inversés)

Problème : Encore deux représentations du zéro !

3. Complément à 2 (méthode standard)

La méthode universellement utilisée en informatique !

💡 Procédure :

1. Prendre le binaire du nombre positif
2. Inverser tous les bits (complément à 1)
3. Ajouter 1

Exemple : -42 en complément à 2 sur 8 bits

1. +42 en binaire : 00101010
2. Inverser tous les bits : 11010101
3. Ajouter 1 : 11010101 + 1 = **11010110**

-42 = **`11010110`** (complément à 2)

Plages de valeurs

Nombre de bits	Plage signée (complément à 2)
8 bits	-128 à +127
16 bits	-32 768 à +32 767
32 bits	-2 147 483 648 à +2 147 483 647
64 bits	-9×10^{18} à $+9 \times 10^{18}$

✨ Avantages du complément à 2 :

- Une seule représentation du zéro
- L'addition fonctionne de la même manière pour les positifs et négatifs
- Pas besoin de circuit séparé pour la soustraction
- Détection de dépassement simple

Nombres à Virgule

Virgule Fixe

Un nombre de bits est réservé pour la partie entière, le reste pour la partie décimale.

Format 8.8 (8 bits entiers, 8 bits décimaux)

Le nombre **42.25** serait représenté :

```
00101010 . 01000000
(42) (0.25)
```

Virgule Flottante (IEEE 754)

Format utilisé par tous les processeurs modernes.

📐 Format 32 bits (simple précision) :

- **1 bit** : signe (0 = positif, 1 = négatif)
- **8 bits** : exposant (polarisé de 127)
- **23 bits** : mantisse (partie fractionnaire)

$$\text{Valeur} = (-1)^{\text{signe}} \times 2^{(\text{exposant}-127)} \times (1 + \text{mantisse})$$

Valeurs spéciales

- **Exposant = 0, Mantisse = 0** : Zéro (+0 ou -0)
- **Exposant = 255, Mantisse = 0** : Infini (+∞ ou -∞)
- **Exposant = 255, Mantisse ≠ 0** : NaN (Not a Number)

⚠️ Erreurs d'arrondi :

Les nombres comme 0.1 ou 0.3 ne peuvent pas être représentés exactement en virgule flottante binaire ! C'est pourquoi 0.1 + 0.2 ≠ 0.3 en informatique.

Utiliser le BCD pour les calculs financiers nécessitant une précision décimale exacte.

◌ Applications Pratiques

Programmation

En C/C++ :

```
int decimal = 255;
int hexa = 0xFF;        // Préfixe 0x
int octal = 0377;       // Préfixe 0
int binaire = 0b11111111; // Préfixe 0b (C++14+)
```

Fréquences radio

Affichage LCD d'un transceiver :

La fréquence **14.250.000 Hz** peut être stockée :

- En binaire : pour les calculs internes
- En BCD : pour l'affichage direct
- En hexa : pour les commandes CAT

Graphisme

Couleur 24 bits (True Color) :

#FF6347 = RGB(255, 99, 71) = Tomate

FF = Rouge (255), 63 = Vert (99), 47 = Bleu (71)

Points Clés à Retenir

- **Octal** : 8 chiffres, grouper par 3 bits, permissions Unix
- **Hexadécimal** : 16 symboles (0-F), grouper par 4 bits, adresses mémoire
- **BCD** : Chaque chiffre décimal sur 4 bits, afficheurs
- **Code Gray** : Un seul bit change entre valeurs consécutives
- **Complément à 2** : Méthode standard pour les nombres signés
- L'**hexadécimal** est le plus utilisé en électronique et informatique
- Bien distinguer **binaire pur** et **BCD**
- Le code Gray évite les erreurs de **transition**

73 de F4HXN - Prochaine fiche : Les tableaux de Karnaugh ! 📡

FICHE 5

Les Tableaux de Karnaugh

Simplifiez vos circuits logiques avec une méthode graphique puissante

Pourquoi Simplifier les Circuits ?

En électronique numérique, une même fonction logique peut être réalisée de plusieurs façons différentes. La **simplification** permet de :

- Réduire le **nombre de portes logiques** nécessaires
- Diminuer le **coût** du circuit
- Réduire la **consommation électrique**
- Augmenter la **vitesse** (moins de portes = moins de délais)
- Améliorer la **fiabilité** (moins de composants = moins de pannes)

Exemple concret

Expression non simplifiée :

$$F = A \cdot B \cdot C + A \cdot B \cdot \bar{C} + A \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot C$$

Nécessite : 4 portes AND à 3 entrées + 1 porte OR à 4 entrées + inverseurs = **~12 portes**

Expression simplifiée :

$$F = A \cdot B + B \cdot C + A \cdot C$$

Nécessite : 3 portes AND à 2 entrées + 1 porte OR à 3 entrées = **~4 portes**

Économie de 66% de composants !

Méthodes de Simplification

1. Méthode Algébrique

Utilise les lois de l'algèbre de Boole pour simplifier les expressions.

Exemple :

$$F = A \cdot B + A \cdot \bar{B}$$
$$F = A \cdot (B + \bar{B}) \text{ [factorisation]}$$
$$F = A \cdot 1 \text{ [B + } \bar{B} = 1]$$
$$F = A$$

⚠️ **Problème :** Fastidieuse pour les expressions complexes, nécessite de l'expérience et de l'intuition. On peut facilement passer à côté de la meilleure simplification.

2. Tableau de Karnaugh (Méthode Graphique)

Méthode **systématique et visuelle** qui garantit une simplification optimale !

✨ Avantages :

- Méthode **visuelle** et intuitive
- Garantit une **simplification optimale**
- Fonctionne bien jusqu'à **4-5 variables**
- Plus rapide que l'algèbre pour les circuits moyens
- Facile à vérifier

◌ Principe du Tableau de Karnaugh

Le tableau de Karnaugh est un tableau où :

- Chaque **case** représente une combinaison des variables d'entrée
- Les cases sont ordonnées selon le **code Gray** (un seul bit change entre cases adjacentes)
- On place dans chaque case la **valeur de sortie** (0 ou 1)
- On regroupe les **cases adjacentes contenant des 1** par groupes de 2, 4, 8, etc.
- Chaque groupe donne un **terme simplifié**

🔑 Principe clé :

Les cases adjacentes ne diffèrent que d'une seule variable. En les regroupant, on peut **éliminer** cette variable de l'expression !

Exemple : $A \cdot B \cdot C + A \cdot B \cdot \bar{C} = A \cdot B \cdot (C + \bar{C}) = A \cdot B$

2️⃣ Tableau de Karnaugh à 2 Variables

Pour 2 variables (A et B), le tableau a **4 cases** ($2^2 = 4$ combinaisons).

Structure du tableau

	B	
A	0	1
0	$\bar{A} \cdot \bar{B}$	$\bar{A} \cdot B$
1	$A \cdot \bar{B}$	$A \cdot B$

Exemple : F = A·B + Ā·B

Étape 1 : Placer les 1 dans le tableau

	B	
A	0	1
0	0	1
1	0	1

Étape 2 : Regrouper les cases adjacentes contenant des 1

Les deux 1 sont verticalement adjacents → on peut les regrouper

Étape 3 : Lire le groupe

Le groupe couvre A=0 et A=1, donc **A varie** (on l'élimine)

Le groupe est entièrement dans la colonne B=1, donc **B reste**

$$F = B$$

Expression simplifiée : F = B

3️⃣ Tableau de Karnaugh à 3 Variables

Pour 3 variables (A, B, C), le tableau a **8 cases** ($2^3 = 8$ combinaisons).

Structure du tableau (disposition Gray)

	BC			
A	**00**	**01**	**11**	**10**
0	**Ā·B̄·C̄**	**Ā·B̄·C**	**Ā·B·C**	**Ā·B·C̄**
1	**A·B̄·C̄**	**A·B̄·C**	**A·B·C**	**A·B·C̄**

⚠️ **Important :** L'ordre des colonnes est **00, 01, 11, 10** (code Gray), pas l'ordre numérique ! Un seul bit change entre cases adjacentes.

Exemple complet : Simplifier F(A,B,C) = Σ(1,3,5,7)

Les mintermines 1, 3, 5, 7 correspondent aux combinaisons où F=1 :

- 1 = 001 = $\bar{A} \cdot \bar{B} \cdot C$
- 3 = 011 = $\bar{A} \cdot B \cdot C$
- 5 = 101 = $A \cdot \bar{B} \cdot C$
- 7 = 111 = $A \cdot B \cdot C$

Étape 1 : Placer les 1 dans le tableau

	BC			
A	00	01	11	10
0	0	1	1	0
1	0	1	1	0

Étape 2 : Identifier les regroupements possibles

On observe que tous les 1 forment un grand groupe de 4 cases (2×2)

Étape 3 : Analyser le regroupement

- Le groupe couvre A=0 et A=1 → **A varie, on l'élimine**
- Le groupe couvre uniquement B=0,C=1 (01) et B=1,C=1 (11) → **C=1 reste**
- B varie dans le groupe → **B s'élimine**

$$F = C$$

Résultat : F = C (au lieu de 4 termes !)

4️⃣ Tableau de Karnaugh à 4 Variables

Pour 4 variables (A, B, C, D), le tableau a **16 cases** ($2^4 = 16$ combinaisons).

Structure du tableau 4×4

	CD			
AB	**00**	**01**	**11**	**10**
00	0	1	5	4
01	2	3	7	6
11	10	11	15	14
10	8	9	13	12

Les numéros dans les cases correspondent aux valeurs décimales des combinaisons

Exemple : F(A,B,C,D) = Σ(0,1,2,5,8,9,10)

	CD			
AB	00	01	11	10
00	1	1	1	0
01	1	0	0	0
11	0	0	0	0
10	1	1	0	1

Regroupements identifiés :

1. **Groupe de 4 :** cases 0, 1, 8, 9 (colonne CD=00 et CD=01)

$$\bar{A}\cdot\bar{B}\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot D + A\cdot\bar{B}\cdot\bar{D} + A\cdot\bar{B}\cdot D = \bar{B}\cdot\bar{D}$$ (C varie, A et C s'éliminent, mais attendez...)

En fait : CD=00 ou 01 → $\bar{D}$ et $\bar{B}$ → **$\bar{B}\cdot\bar{D}$**

2 **Groupe de 2 :** cases 0, 2 (colonne AB=00 et 01, CD=00)

$$\bar{A} \cdot \bar{C} \cdot \bar{D}$$

3 **Groupe de 2 :** cases 1, 5 (ligne AB=00, colonnes CD=01 et 11)

$$\bar{A} \cdot \bar{B} \cdot D$$

Expression finale simplifiée :

$$F = \bar{B} \cdot \bar{D} + \bar{A} \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot \bar{B} \cdot D$$

Note : Cette simplification peut varier selon les regroupements choisis. L'objectif est de minimiser le nombre de termes.

Règles de Regroupement

✅ Règles à respecter :

1. Les groupes doivent contenir un **nombre de cases qui est une puissance de 2** : 1, 2, 4, 8, 16
2. Les groupes doivent être **rectangulaires** (pas de forme en L)
3. Les groupes doivent être **aussi grands que possible**
4. Chaque 1 doit être dans **au moins un groupe**
5. Les groupes peuvent **se chevaucher**
6. Les **bords opposés** du tableau sont adjacents (effet torique)

Cas des bords adjacents

Le tableau est "torique"

	CD			
AB	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

Les 4 coins forment un groupe valide ! (gauche et droite sont adjacents)

$$F = \bar{B} \cdot \bar{D}$$

Les Conditions Indifférentes (Don't Care)

Parfois, certaines combinaisons d'entrées **ne se produisent jamais** ou leur sortie **n'a pas d'importance**. On les note **X** ou **d** (don't care).

💡 Utilisation stratégique :

On peut considérer les X comme des **0 ou des 1** selon ce qui simplifie le mieux le circuit !

Exemple : Afficheur BCD

Un décodeur BCD n'utilise que les combinaisons 0-9 (1010 à 1111 sont invalides en BCD).

	CD			
AB	**00**	**01**	**11**	**10**
00	0	1	1	0
01	0	1	1	0
11	X	X	X	X
10	X	X	X	X

Stratégie : On peut inclure certains X dans nos groupes pour les agrandir !

En considérant les X comme des 1, on peut former un **grand groupe de 8 cases** (colonnes 01 et 11) :

$$F = D$$

Simplification maximale grâce aux don't care !

Méthode Pas à Pas

1 **Remplir le tableau**

Placer les 1, 0 et X (don't care) dans chaque case selon la table de vérité

2 **Identifier les groupes de 1**

Chercher les plus grands groupes possibles (8, 4, 2, 1) en incluant éventuellement des X

3 **Vérifier la couverture**

S'assurer que tous les 1 sont dans au moins un groupe

4 **Écrire les termes**

Pour chaque groupe, écrire le produit des variables qui ne changent pas dans le groupe

5 **Combiner avec OR**

L'expression finale est la somme (OR) de tous les termes trouvés

Application : Contrôleur de Relais Radio

Cahier des charges

Un relais amateur s'active SI :

- **P** : PTT activé (1=appuyé, 0=relâché)
- **T** : Tonalité CTCSS correcte (1=OK, 0=KO)
- **S** : Signal suffisant (1=bon, 0=faible)
- **B** : Batterie OK (1=OK, 0=faible)

Règles :

- Le relais s'active si PTT ET CTCSS ET Signal sont OK
- Si batterie faible, le relais ne s'active que si signal très fort (à définir)
- Certaines combinaisons impossibles (ex: PTT sans signal)

Table de vérité simplifiée

P	T	S	B	Relais (F)
0	0	0	0	0
1	1	1	1	1
1	1	1	0	0
1	0	1	1	0
...	...	...	...	...

Après simplification par Karnaugh :

$$F = P \cdot T \cdot S \cdot B + \text{autres termes...}$$

Le circuit final nécessite moins de portes grâce à la simplification !

◌ Comparaison des Méthodes

Critère	Algèbre de Boole	Karnaugh
Nombre de variables	Illimité	Pratique jusqu'à 4-5
Garantie d'optimum	Non	Oui
Facilité d'utilisation	Nécessite expérience	Méthodique et visuel
Rapidité	Variable	Rapide pour 2-4 variables
Don't care	Difficile à gérer	Facile à intégrer
Automatisation	Difficile	Facile à programmer

🎯 **Recommandation :**

- **2 à 4 variables :** Utiliser Karnaugh (rapide et optimal)
- **5-6 variables :** Karnaugh possible mais fastidieux
- **7+ variables :** Utiliser des logiciels (Quine-McCluskey, outils CAO)

Points Clés à Retenir

- Les tableaux de Karnaugh permettent une **simplification visuelle et systématique**
- L'ordre des cases suit le **code Gray** (un seul bit change)
- Les groupes doivent être de taille **2^n** (1, 2, 4, 8, 16...)
- Toujours former les **groupes les plus grands** possibles
- Les **bords opposés** du tableau sont adjacents
- Les **don't care (X)** peuvent être 0 ou 1 selon ce qui simplifie le mieux
- Chaque groupe élimine les **variables qui varient** à l'intérieur
- Méthode **optimale** pour 2 à 4 variables

73 de F4HXN - Prochaine fiche : Les circuits combinatoires ! 📡

FICHE 6

Les Circuits Combinatoires

Des additionneurs aux multiplexeurs : les briques de base de l'informatique

Qu'est-ce qu'un Circuit Combinatoire ?

Un **circuit combinatoire** est un circuit logique dont les sorties dépendent **uniquement** des valeurs actuelles des entrées, sans mémoire du passé.

📐 Caractéristiques :

- **Pas de mémoire :** aucun état interne conservé
- **Pas de boucle de rétroaction**
- **Réponse instantanée** (à la vitesse des portes)
- Peut être décrit par une **table de vérité** ou une **fonction logique**

Circuit Combinatoire

Sortie = f(Entrées actuelles)

Exemples :

- Additionneur
- Multiplexeur
- Décodeur
- Comparateur

⏰ Circuit Séquentiel

Sortie = f(Entrées, État précédent)

Exemples :

- Bascules
- Compteurs
- Registres
- Mémoires

Les Additionneurs

Demi-Additionneur (Half Adder)

Le **demi-additionneur** additionne deux bits sans retenue d'entrée.

A	B	Somme (S)	Retenue (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \oplus B \text{ (XOR)}$$
$$C = A \cdot B \text{ (AND)}$$

Réalisation

Nécessite : **1 porte XOR** + **1 porte AND**

Additionneur Complet (Full Adder)

L'**additionneur complet** additionne deux bits PLUS une retenue d'entrée (Cin).

A	B	Cin	Somme (S)	Retenue (Cout)
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

$$S = A \oplus B \oplus Cin$$

$$Cout = (A \cdot B) + (Cin \cdot (A \oplus B))$$

Circuits intégrés

7483 Additionneur 4 bits **74283** Additionneur 4 bits rapide

Additionneur N bits

Addition de nombres multi-bits

Pour additionner des nombres de N bits, on cascade N additionneurs complets :

1. Le bit de poids faible utilise un additionneur complet avec Cin = 0
2. Chaque Cout devient le Cin de l'étage suivant
3. Le Cout final indique un dépassement de capacité

Exemple : 5 + 3 sur 4 bits

```
  0101  (5)
+ 0011  (3)
------
  1000  (8)
```

Le Soustracteur

La soustraction peut être réalisée en utilisant le **complément à 2** :

💡 **Astuce :**

A - B = A + (-B) = A + (complément à 2 de B)

On peut donc réutiliser un additionneur !

Réalisation d'un soustracteur

1. Inverser tous les bits de B (avec des portes XOR)
2. Ajouter 1 en mettant Cin = 1
3. Utiliser un additionneur pour A + $\bar{B}$ + 1

Avantage : Le même circuit peut faire addition ET soustraction avec un signal de contrôle !

◌ Le Comparateur de Magnitude

Le **comparateur** compare deux nombres binaires et indique si A > B, A = B, ou A < B.

Comparateur 1 bit

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

$$A > B = A \cdot \bar{B}$$

$$A = B = A \odot B \text{ (XNOR)}$$

$$A < B = \bar{A} \cdot B$$

Circuits intégrés

7485 Comparateur 4 bits

Application radio

Comparer la valeur du ROS avec un seuil de sécurité. Si ROS > seuil, couper l'émetteur.

Le Multiplexeur (MUX)

Le **multiplexeur** sélectionne une entrée parmi N et la route vers la sortie, selon un code de sélection.

📐 **Principe :**

Un MUX à N entrées nécessite **$\log_2(N)$ lignes de sélection**.

- MUX 2:1 → 1 ligne de sélection
- MUX 4:1 → 2 lignes de sélection
- MUX 8:1 → 3 lignes de sélection

Multiplexeur 2:1

S (Sélection)	Sortie Y
0	I_0 (entrée 0)
1	I_1 (entrée 1)

$$Y = \bar{S} \cdot I_0 + S \cdot I_1$$

Multiplexeur 4:1

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

Circuits intégrés

74157 Quad MUX 2:1 **74153** Dual MUX 4:1 **74151** MUX 8:1

Applications

1. Sélection de source audio

Choisir entre microphone local, liaison téléphonique, ou réception radio

2. Commutation d'antennes

Sélectionner une antenne parmi plusieurs selon la bande

3. Transmission série

Convertir des données parallèles en série (un bit à la fois)

4. Implémentation de fonctions logiques

Un MUX peut réaliser n'importe quelle fonction logique !

Le Démultiplexeur (DEMUX)

Le **démultiplexeur** fait l'opération inverse : il route une entrée vers l'une des N sorties.

📐 Principe :

Un DEMUX 1:N utilise $\log_2(N)$ lignes de sélection pour choisir la sortie active.

Démultiplexeur 1:4

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

Circuits intégrés

74139 Dual DEMUX 1:4 **74138** DEMUX 1:8

Application : Adressage mémoire

Un DEMUX permet de sélectionner une case mémoire parmi plusieurs selon son adresse binaire.

Encodeur et Décodeur

Encodeur (Encoder)

Un **encodeur** convertit 2^N entrées en un code binaire de N bits.

Encodeur 8:3 (Octal vers Binaire)

Entrée active	A_2	A_1	A_0	Code
I_0	0	0	0	0
I_1	0	0	1	1
I_2	0	1	0	2
I_3	0	1	1	3
I_4	1	0	0	4
I_5	1	0	1	5
I_6	1	1	0	6
I_7	1	1	1	7

Circuits intégrés

74148 Encodeur prioritaire 8:3 74147 Encodeur décimal vers BCD

Décodeur (Decoder)

Un **décodeur** fait l'opération inverse : convertit N bits en 2^N sorties (une seule active).

Décodeur 3:8 (Binaire vers Octal)

Si l'entrée est **101** (5), seule la sortie Y_5 est active (1), toutes les autres sont à 0.

Circuits intégrés

74138 Décodeur 3:8 7442 Décodeur BCD vers décimal

Applications

- **Sélection de périphériques :** Activer un composant selon son adresse
- **Affichage :** Piloter des LEDs ou segments
- **Distribution de signaux :** Router un signal vers une destination

Décodeur 7 Segments

Un **décodeur 7 segments** convertit un code BCD (0-9) en signaux pour piloter un afficheur 7 segments.

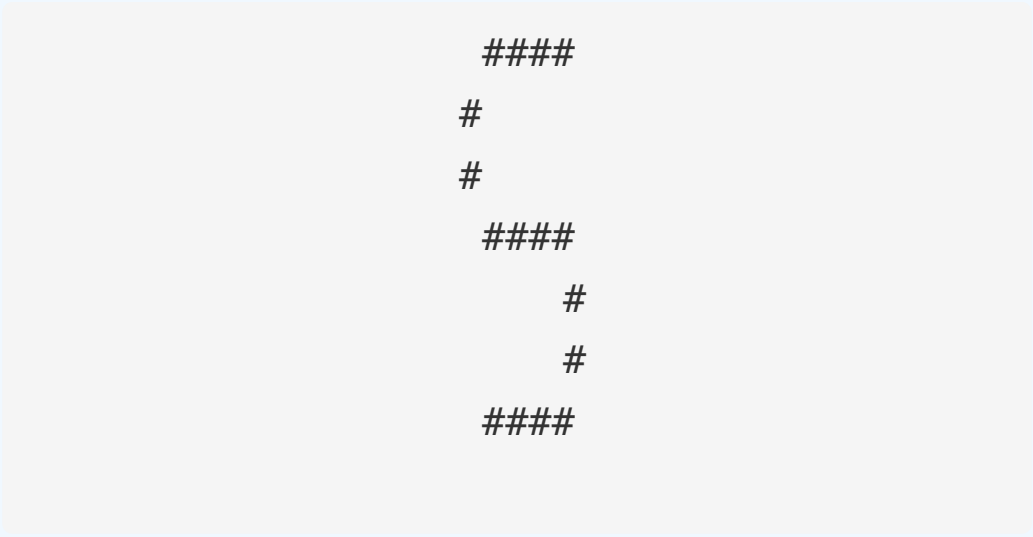
📟 Structure d'un afficheur 7 segments :

7 segments (a, b, c, d, e, f, g) disposés en forme de "8" :

```
  aaaa
f    b
f    b
  gggg
e    c
e    c
  dddd
```

Exemple : Afficher le chiffre "5"

Segments actifs : a, f, g, c, d



Circuits intégrés

7447 Décodeur BCD vers 7 seg. (anode commune) **7448** Décodeur BCD vers 7 seg. (cathode commune)

Application radio

Afficher la fréquence, le canal, le niveau S-meter, ou le numéro de mémoire sur un transceiver.

L'ALU (Unité Arithmétique et Logique)

L'**ALU** est le cœur de tout processeur. Elle réalise des opérations arithmétiques ET logiques selon un code de commande.

🎯 Fonctions typiques d'une ALU :

- **Arithmétiques :** Addition, Soustraction, Incrémentation, Décrémentation
- **Logiques :** AND, OR, XOR, NOT
- **Comparaison :** Égalité, Supérieur, Inférieur
- **Décalages :** Shift left, Shift right

Structure d'une ALU simple

Composants principaux

1. **Additionneur/Soustracteur** pour les opérations arithmétiques
2. **Portes logiques** (AND, OR, XOR) pour les opérations logiques
3. **Multiplexeur** pour sélectionner le résultat selon l'opération
4. **Flags** : Carry (retenue), Zero (résultat nul), Negative (signe), Overflow

Table des opérations (exemple 4 bits)

Code Op	Opération	Résultat
000	A AND B	A · B
001	A OR B	A + B
010	A XOR B	A ⊕ B
011	NOT A	Ā
100	ADD	A + B
101	SUB	A - B
110	INC	A + 1
111	DEC	A - 1

Circuits intégrés

74181 ALU 4 bits (classique !) **74381** ALU 4 bits rapide

⚠️ **Note historique :**

Le 74181 fut l'une des premières ALU intégrées (1970). Elle peut réaliser 16 opérations logiques et 16 opérations arithmétiques ! C'est un composant légendaire de l'histoire de l'informatique.

Projet Pratique : Additionneur 4 bits

Objectif

Construire un additionneur 4 bits avec affichage 7 segments du résultat.

Matériel nécessaire

- 4× Circuits intégrés 7483 (additionneur 4 bits) ou équivalent
- 1× Circuit 7447 (décodeur BCD vers 7 segments)
- 1× Afficheur 7 segments
- 8× Interrupteurs DIP (pour A et B)
- 5× LEDs (pour afficher le résultat en binaire)
- Résistances 330Ω
- Alimentation 5V
- Plaque d'essai

1 Entrées

Connecter 4 interrupteurs pour A ($A_3A_2A_1A_0$) et 4 pour B ($B_3B_2B_1B_0$)

2 Addition

Utiliser le 7483 pour additionner A + B. Connecter Cin à 0.

3 Affichage binaire

Connecter 5 LEDs aux sorties ($S_3S_2S_1S_0$ + Cout) pour voir le résultat en binaire

4 Affichage décimal

Connecter les 4 bits de somme au 7447, puis au 7-segments pour affichage 0-9

5 Test

Tester plusieurs additions : 5+3=8, 7+6=13 (affichera 3 en décimal + Cout=1)

🎓 Extension possible :

Ajouter un inverseur et un MUX pour créer un additionneur/soustracteur sélectionnable !

Applications en Radio Amateur

◌ Synthétiseur de fréquence

Les additionneurs calculent les fréquences intermédiaires dans les PLL.

S-Meter numérique

Comparateurs pour convertir le niveau du signal en indication visuelle par paliers.

Sélecteur de mémoire

Encodeur pour convertir les touches en code binaire de canal.

Affichage de fréquence

Décodeurs BCD vers 7-segments pour afficher les MHz, kHz.

Commutateur d'antenne

Décodeur 3:8 pour sélectionner 1 parmi 8 antennes selon la bande.

Contrôle de gain

MUX pour sélectionner différents niveaux d'atténuation RF.

Résumé des Circuits Combinatoires

Circuit	Fonction	CI typique
Demi-additionneur	A + B (2 bits)	-
Additionneur complet	A + B + Cin	7483
Soustracteur	A - B	-
Comparateur	A ? B	7485
Multiplexeur	Sélection N:1	74151
Démultiplexeur	Distribution 1:N	74138
Encodeur	2^N → N bits	74148
Décodeur	N bits → 2^N	74138
Décodeur 7-seg	BCD → 7 segments	7447
ALU	Arith. + Logique	74181

Points Clés à Retenir

- Les circuits combinatoires n'ont **pas de mémoire**
- L'**additionneur complet** est la brique de base de l'arithmétique
- La soustraction utilise le **complément à 2**
- Le **multiplexeur** sélectionne une entrée parmi N
- Le **décodeur** active une sortie parmi N
- Les **encodeurs** convertissent $2^N \rightarrow N$ bits
- L'**ALU** combine arithmétique et logique
- Ces circuits sont les **briques de base** de tout processeur

73 de F4HXN - Prochaine fiche : Les circuits séquentiels ! 📡

FICHE 7

⏰ Les Circuits Séquentiels et Mémoires

Bascules, registres, compteurs et mémoires : les circuits qui se souviennent

Circuit Séquentiel vs Combinatoire

Un **circuit séquentiel** possède une **mémoire** : sa sortie dépend non seulement des entrées actuelles, mais aussi de l'**état précédent**.

Circuit Combinatoire

```
Sortie = f(Entrées)
```

- Pas de mémoire
- Réponse immédiate
- Pas d'horloge
- Table de vérité fixe

Circuit Séquentiel

```
Sortie = f(Entrées, État)
```

- Possède une mémoire
- Évolution dans le temps
- Souvent synchronisé par horloge
- Diagramme d'états

🔑 Élément clé :

La **boucle de rétroaction** permet au circuit de mémoriser son état. La sortie est réinjectée à l'entrée, créant ainsi une mémoire.

Les Bascules (Flip-Flops)

Une **bascule** est l'élément de mémoire de base. Elle peut stocker **1 bit** d'information.

Bascule RS (Set-Reset)

La bascule RS est la plus simple. Elle a deux entrées : **S (Set)** et **R (Reset)**.

S	R	Q (sortie)	$\bar{Q}$	Action
0	0	Q	$\bar{Q}$	Maintien
1	0	1	0	Set (mise à 1)
0	1	0	1	Reset (mise à 0)
1	1	X	X	Interdit !

⚠️ **État interdit :**

S=1 et R=1 simultanément créent un état instable. Cette combinaison doit être évitée !

Réalisation avec portes NOR

Deux portes NOR avec rétroaction croisée forment une bascule RS.

$$Q = \bar{S} + R \cdot Q \text{ (simplifiée)}$$
$$\bar{Q} = \bar{R} + S \cdot \bar{Q}$$

Bascule D (Data)

La bascule D capture la valeur de l'entrée D au moment du **front d'horloge**.

CLK	D	Q (après CLK)	Action
↑	0	0	Copie D → Q
↑	1	1	Copie D → Q
0 ou 1	X	Q	Maintien

💡 **Avantage :**

Pas d'état interdit ! La bascule D élimine le problème de la RS. Elle est la plus utilisée dans les circuits modernes.

7474 Dual D Flip-Flop

Application radio

Synchroniser des signaux de commande avec une horloge système pour éviter les états transitoires indésirables.

Bascule JK

La bascule JK est une amélioration de la RS qui élimine l'état interdit.

J	K	Q (après CLK)	Action
0	0	Q	Maintien
0	1	0	Reset
1	0	1	Set
1	1	$\bar{Q}$	Toggle (bascule)

✨ Caractéristique unique :

Quand J=K=1, la bascule **inverse** son état (toggle). C'est très utile pour créer des compteurs !

7476 Dual JK Flip-Flop

Bascule T (Toggle)

La bascule T est une simplification de la JK. Elle a une seule entrée T.

T	Q (après CLK)	Action
0	Q	Maintien
1	$\bar{Q}$	Toggle (inverse)

Application typique

Division de fréquence par 2. À chaque front d'horloge avec T=1, la sortie change d'état.

Exemple : Horloge 1 MHz → Sortie 500 kHz

⏱️ Synchrone vs Asynchrone

Bascule Asynchrone

Les entrées S et R agissent **immédiatement**, sans attendre l'horloge.

Usage : Initialisation, remise à zéro rapide

Bascule Synchrone

Les changements se font uniquement sur les **fronts d'horloge**.

Usage : Circuits numériques synchronisés

🎯 Front d'horloge :

- **Front montant (↑) :** Transition de 0 vers 1
- **Front descendant (↓) :** Transition de 1 vers 0

La plupart des circuits modernes utilisent le front montant.

Les Registres

Un **registre** est un ensemble de bascules (généralement D) permettant de stocker plusieurs bits.

Registre Parallèle

Toutes les bascules sont chargées **simultanément** sur le même front d'horloge.

Registre 8 bits

8 bascules D en parallèle stockent un octet complet.

Application : Mémoriser temporairement une donnée pendant un traitement.

74173 Registre 4 bits **74374** Registre 8 bits avec sortie 3 états

Registre à Décalage (Shift Register)

Les bits se déplacent d'une position à chaque front d'horloge.

Exemple : Décalage vers la droite

```
État initial : [1][0][1][1]
Après CLK 1  : [0][1][0][1]  (nouvelle donnée = 0)
Après CLK 2  : [1][0][1][0]  (nouvelle donnée = 1)
Après CLK 3  : [0][1][0][1]  (nouvelle donnée = 0)
```

🎯 Applications des registres à décalage :

- **Conversion série ↔ parallèle**
- **Délai numérique** (retard de N bits)
- **Génération de séquences**
- **Multiplication/division par 2** (décalage = × ou ÷ 2)

74164 Registre à décalage 8 bits (série → parallèle) **74165** Registre à décalage 8 bits (parallèle → série) **74595** Registre à décalage 8 bits avec sortie tampon

Application : Transmission série

Les données de commande CAT vers le transceiver sont envoyées en série. Un registre à décalage convertit les 8 bits parallèles du microcontrôleur en flux série.

Les Compteurs

Un **compteur** est un circuit séquentiel qui parcourt une séquence d'états à chaque impulsion d'horloge.

Compteur Asynchrone (Ripple Counter)

Chaque bascule est déclenchée par la sortie de la bascule précédente. L'horloge se "propage" en cascade.

Compteur 4 bits asynchrone (modulo 16)

```
Compte : 0000 → 0001 → 0010 → 0011 → ... → 1111 → 0000
         (0)    (1)    (2)    (3)          (15)   (0)
```

⚠️ **Problème :**

Délai de propagation cumulatif. À haute fréquence, des états transitoires indésirables peuvent apparaître.

Compteur Synchrone

Toutes les bascules reçoivent la **même horloge**. Pas de délai cumulatif !

✨ **Avantages :**

- Pas d'états transitoires (glitches)
- Fonctionne à haute fréquence
- Plus prévisible

7490 Compteur décade (0-9) **7493** Compteur binaire 4 bits

74161 Compteur synchrone 4 bits **74193** Compteur/décompteur BCD synchrone

Types de Compteurs

Compteur UP

Compte de manière croissante

`0 → 1 → 2 → 3 → 4...`

Compteur DOWN

Compte de manière décroissante

`...4 → 3 → 2 → 1 → 0`

Compteur UP/DOWN

Bidirectionnel selon un signal de contrôle

Signal DIR = montant ou descendant

Compteur Modulo N

Compte de 0 à N-1 puis revient à 0

Exemple : modulo 10 = 0-9

Application : Diviseur de fréquence

Un compteur modulo N divise la fréquence d'entrée par N.

Exemple : Compteur modulo 100 avec horloge 10 MHz → sortie 100 kHz

Utilisé dans les synthétiseurs de fréquence (PLL) des transceivers.

Projet : Compteur décimal avec affichage

1. Utiliser un 7490 (compteur décade 0-9)
2. Connecter un 7447 (décodeur BCD vers 7 segments)
3. Afficher sur un afficheur 7 segments
4. Ajouter un bouton poussoir pour incrémenter

Résultat : Compteur manuel de 0 à 9 !

Les Mémoires

Une **mémoire** est un circuit capable de stocker une grande quantité d'informations binaires.

Organisation d'une mémoire

📐 Paramètres clés :

- **Capacité :** Nombre de bits stockés (ex: 1 Ko = 8192 bits)
- **Organisation :** Nombre de mots × largeur (ex: 1024×8)
- **Bus d'adresse :** N bits → 2^N adresses
- **Bus de données :** Largeur d'un mot (8, 16, 32 bits...)

Exemple : Mémoire 1K×8

- Capacité : 1024 mots de 8 bits = 8192 bits = 1 Ko
- Bus d'adresse : 10 bits (2^{10} = 1024 adresses)
- Bus de données : 8 bits

RAM (Random Access Memory)

Mémoire **volatile** : perd son contenu à la coupure de l'alimentation.

SRAM (Static RAM)

Utilise des bascules

- ✅ Très rapide
- ✅ Pas de rafraîchissement
- ❌ Plus coûteuse
- ❌ Moins dense

Usage : Mémoire cache

DRAM (Dynamic RAM)

Utilise des condensateurs

- ✅ Haute densité
- ✅ Moins coûteuse
- ❌ Nécessite rafraîchissement
- ❌ Plus lente

Usage : Mémoire principale

6264 SRAM 8K×8 **62256** SRAM 32K×8

ROM (Read-Only Memory)

Mémoire **non volatile** : conserve les données sans alimentation.

ROM (Mask ROM)

Programmée en usine

Non modifiable. Coût élevé pour faibles volumes.

PROM

Programmable une fois

Fusibles "grillés". Ancienne technologie.

EPROM

Effaçable aux UV

Fenêtre en quartz. Effacement 15-20 min.

EEPROM

Effaçable électriquement

Effacement octet par octet. Lente en écriture.

Flash

EEPROM par blocs

Rapide, haute densité. Technologie moderne.

2764 EPROM 8K×8 **27C256** EPROM 32K×8 **28C256** EEPROM 32K×8

Mémoires dans un transceiver

- **ROM/Flash :** Firmware du transceiver
- **EEPROM :** Canaux mémoire, configuration
- **RAM :** Buffer audio, traitement DSP

Signaux de contrôle

🎛️ Signaux typiques d'une mémoire :

- **A0-An :** Bus d'adresse
- **D0-Dm :** Bus de données (bidirectionnel)
- **CS (Chip Select) :** Sélection du circuit
- **OE (Output Enable) :** Activation des sorties
- **WE (Write Enable) :** Autorisation d'écriture

Lecture d'une mémoire

1. Placer l'adresse sur le bus A0-An
2. Activer CS (Chip Select) = 0
3. Activer OE (Output Enable) = 0
4. Attendre le temps d'accès
5. Lire les données sur le bus D0-Dm

Temps Caractéristiques

⏱️ Paramètres temporels importants :

Pour les bascules :

- **Setup time (tsu) :** Temps minimum avant le front d'horloge où l'entrée doit être stable
- **Hold time (th) :** Temps minimum après le front où l'entrée doit rester stable
- **Propagation delay (tpd) :** Délai entre front d'horloge et changement de sortie

Pour les mémoires :

- **Access time (tacc) :** Temps entre adresse valide et données valides
- **Cycle time (tcyc) :** Temps minimum entre deux accès
- **Write time (twr) :** Durée d'impulsion d'écriture nécessaire

⚠️ Important :

Le non-respect des temps setup et hold peut causer des **métastabilités** : état indéterminé de la bascule entre 0 et 1. Cela peut bloquer tout le système !

◌ Projet Pratique : Chenillard LED

Objectif

Créer un effet de chenillard avec 8 LEDs qui s'allument séquentiellement.

Matériel

- 1× Circuit 74164 (registre à décalage 8 bits)
- 1× Circuit 555 (générateur d'horloge) ou Arduino
- 8× LEDs + résistances 330Ω
- 1× Résistance 10kΩ, 1× condensateur 10µF (pour 555)
- Alimentation 5V

1 Générateur d'horloge

Utiliser un 555 en mode astable pour générer une horloge ~1 Hz

2 Registre à décalage

Connecter le 74164. L'entrée série A et B = 1

3 LEDs

Connecter 8 LEDs (+ résistances) aux sorties Q0-Q7

4 **Boucle**

Pour faire boucler : connecter Q7 à l'entrée série (inversée avec un NOT)

5 **Test**

Les LEDs s'allument l'une après l'autre en continu !

🎨 **Variantes :**

- Changer la vitesse (ajuster R et C du 555)
- Faire un aller-retour (compteur UP/DOWN)
- Plusieurs patterns avec un multiplexeur

Applications Radio Amateur

◌ Mémoires de fréquence

EEPROM pour sauvegarder canaux, CTCSS, modes

Buffer DSP

RAM rapide pour traitement audio numérique

Affichage de fréquence

Registres pour maintenir l'affichage stable

⏱️ Timer/Watchdog

Compteur pour temporisation (TOT, squelch tail)

◌ Contrôle de gain

Registre à décalage pour CAG numérique

Synthétiseur PLL

Compteurs programmables pour division de fréquence

Points Clés à Retenir

- Les circuits séquentiels possèdent une **mémoire**
- Les **bascules** sont les éléments de mémoire de base (1 bit)
- La bascule **D** est la plus utilisée (pas d'état interdit)
- Les **registres** stockent plusieurs bits en parallèle
- Les **registres à décalage** convertissent série ↔ parallèle
- Les **compteurs** parcourent une séquence d'états
- La **RAM** est volatile, la **ROM** non volatile
- La **Flash** est la technologie moderne pour stocker firmware
- Respecter les temps **setup** et **hold** est crucial

73 de F4HXN - Prochaine fiche : Les machines à états ! 📡

FICHE 8

Les Machines à États Finis (FSM)

Modéliser et concevoir des systèmes séquentiels complexes

Qu'est-ce qu'une Machine à États Finis ?

Une **Machine à États Finis** (FSM - Finite State Machine) est un modèle mathématique de calcul qui se trouve toujours dans l'un d'un **nombre fini d'états**. Elle passe d'un état à un autre selon les entrées et des **règles de transition** définies.

📐 Définition formelle :

Une FSM est définie par :

- **S** : Ensemble fini d'états
- **I** : Ensemble fini d'entrées
- **O** : Ensemble fini de sorties
- **δ** : Fonction de transition (état suivant = δ(état actuel, entrée))
- **λ** : Fonction de sortie
- **s_0** : État initial

Exemple simple : Feu tricolore

- **États :** {VERT, ORANGE, ROUGE}
- **Entrée :** Signal d'horloge (temporisation)
- **Transitions :** VERT → ORANGE → ROUGE → VERT → ...

Les Deux Types de FSM

Machine de Moore

Dans une **machine de Moore**, les sorties dépendent **uniquement de l'état actuel**.

```
Sortie = λ(État actuel)
```

Avantages

- Sorties synchronisées avec l'horloge
- Pas de glitches de sortie
- Plus simple à concevoir
- Plus prévisible

Inconvénients

- Peut nécessiter plus d'états
- Réponse retardée d'un cycle

Machine de Mealy

Dans une **machine de Mealy**, les sorties dépendent de **l'état actuel ET des entrées**.

```
Sortie = λ(État actuel, Entrées)
```

Avantages

- Réponse plus rapide
- Moins d'états nécessaires
- Plus compact

Inconvénients

- Risque de glitches sur les sorties
- Plus complexe à concevoir
- Sorties asynchrones

🎯 Choisir entre Moore et Mealy :

- **Moore :** Quand la stabilité des sorties est cruciale
- **Mealy :** Quand la vitesse de réaction est prioritaire

Diagramme d'États

Le **diagramme d'états** est une représentation graphique de la FSM. C'est l'outil principal pour concevoir et documenter une machine à états.

🎨 Convention de notation :

- **Cercle** : Représente un état
- **Flèche** : Transition entre états
- **Double cercle** : État initial (optionnel)
- **Étiquette de flèche :** Condition / Sortie

Exemple : Détecteur de séquence "101"

Objectif

Détecter la séquence "101" dans un flux binaire série. La sortie passe à 1 quand la séquence est détectée.

Diagramme d'états (Moore)

S0 (Initial) : Rien détecté

→ Si entrée=1 : aller en S1

→ Si entrée=0 : rester en S0

S1 : "1" détecté

→ Si entrée=0 : aller en S2

→ Si entrée=1 : rester en S1

S2 : "10" détecté

→ Si entrée=1 : aller en S3 (séquence complète !)

→ Si entrée=0 : retour en S0

S3 : "101" détecté → **Sortie = 1**

→ Si entrée=0 : aller en S2 (pour détecter "1010...")

→ Si entrée=1 : rester en S1

Table de transition

État actuel	Entrée = 0	Entrée = 1	Sortie
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	S1	**1**

Implémentation Matérielle

Pour implémenter une FSM en matériel, on utilise :

1 **Registre d'état**

Bascules D pour mémoriser l'état actuel (N bascules pour 2^N états)

2 **Logique combinatoire de transition**

Calcule l'état suivant = f(état actuel, entrées)

3 **Logique de sortie**

Moore : f(état) | Mealy : f(état, entrées)

Encodage des états

Chaque état doit être représenté par un code binaire unique.

Encodage binaire

États numérotés séquentiellement

```
S0=00, S1=01
S2=10, S3=11
```

Simple mais pas optimal

Encodage Gray

Un seul bit change entre états adjacents

```
S0=00, S1=01
S2=11, S3=10
```

Réduit les glitches

One-Hot

Un bit par état (1 seul à 1)

```
S0=0001, S1=0010
S2=0100, S3=1000
```

Plus rapide, plus de bascules

🎯 Choix de l'encodage :

- **Binaire :** Minimise le nombre de bascules
- **Gray :** Réduit les transitions multiples
- **One-Hot :** Simplifie la logique de décodage, plus rapide

Conception avec Karnaugh

Méthode

1. Choisir un encodage pour les états
2. Créer la table de vérité complète (état actuel + entrées → état suivant + sorties)
3. Simplifier avec Karnaugh pour chaque bit de l'état suivant
4. Simplifier la logique de sortie
5. Implémenter avec bascules D et portes logiques

Exemple Complet : Contrôleur de Feu Tricolore

Cahier des charges

Feu tricolore simple avec temporisations :

- **VERT :** 30 secondes
- **ORANGE :** 5 secondes
- **ROUGE :** 30 secondes

Entrée : Signal d'horloge (1 Hz) + compteur interne

États et transitions

État	Durée	Sorties (R,O,V)	État suivant
S_VERT	30s	0,0,1	S_ORANGE
S_ORANGE	5s	0,1,0	S_ROUGE
S_ROUGE	30s	1,0,0	S_VERT

Implémentation

Utiliser :

- 2 bascules D pour encoder 3 états (encodage binaire : 00, 01, 10)
- Compteur 6 bits pour compter jusqu'à 30
- Comparateurs pour détecter les fins de temporisation
- Logique combinatoire pour les transitions

Application : Décodeur de Protocole Série

Problème

Décoder un protocole série simple :

- **START bit :** 0
- **8 bits de données**
- **STOP bit :** 1

États nécessaires

État	Description	Action
IDLE	Attente de START	Surveiller ligne série
START	START détecté	Initialiser compteur
DATA	Réception données	Décaler les bits (×8)
STOP	Vérification STOP	Valider trame
ERROR	Erreur détectée	Signal d'erreur

Chronogramme d'une trame

```
Ligne : ___\___/‾‾\___/‾‾\___/‾‾...___/‾‾\___
États : IDLE START D0  D1  D2  D3...D7  STOP IDLE
```

Application concrète

Ce type de FSM est utilisé dans :

- Interface CAT des transceivers
- Réception RTTY (radioteletype)
- Décodage APRS
- Communication avec GPS

Implémentation en VHDL/Verilog

Les FSM sont souvent décrites en langages HDL (Hardware Description Language) pour implémentation sur FPGA.

Exemple en pseudo-code VHDL

```
-- Machine à états pour détecteur de séquence
PROCESS(clk, reset)
BEGIN
    IF reset = '1' THEN
        state <= S0;
    ELSIF rising_edge(clk) THEN
        CASE state IS
            WHEN S0 =>
                IF input = '1' THEN
                    state <= S1;
                ELSE
                    state <= S0;
                END IF;

            WHEN S1 =>
                IF input = '0' THEN
                    state <= S2;
                ELSE
                    state <= S1;
                END IF;

            WHEN S2 =>
                IF input = '1' THEN
                    state <= S3;
                ELSE
                    state <= S0;
                END IF;

            WHEN S3 =>
                IF input = '0' THEN
                    state <= S2;
                ELSE
                    state <= S1;
                END IF;
        END CASE;
    END IF;
END PROCESS;

-- Logique de sortie (Moore)
output <= '1' WHEN state = S3 ELSE '0';
```

🎯 Avantages de HDL :

- Description de haut niveau
- Simulation possible avant fabrication
- Synthèse automatique en portes logiques
- Facile à modifier et maintenir

◌ Méthodologie de Conception

1 **Spécification**

Définir clairement le comportement souhaité et les entrées/sorties

2 **Diagramme d'états**

Dessiner le diagramme avec tous les états et transitions

3 **Choix du type**

Décider entre Moore et Mealy selon les contraintes

4 **Encodage des états**

Choisir binaire, Gray ou One-Hot

5 **Table de transition**

Créer la table complète état actuel → état suivant

6 **Simplification**

Utiliser Karnaugh pour minimiser la logique

7 **Implémentation**

Réaliser avec bascules + portes OU HDL + FPGA

8 **Vérification**

Tester tous les cas, y compris les cas limites

◌ Pièges à Éviter

1. États non atteignables

Vérifier que tous les états peuvent être atteints depuis l'état initial.

2. Blocage (deadlock)

S'assurer qu'il n'existe pas d'état sans sortie possible.

3. États non spécifiés

Avec encodage binaire, certains codes peuvent ne correspondre à aucun état défini. Prévoir un état de récupération.

4. Conditions de transition ambiguës

Deux transitions depuis un même état ne doivent jamais être vraies simultanément.

5. Métastabilité

Synchroniser les signaux asynchrones avec des bascules en cascade (double-flop).

Autres Applications Radio

Encodeur CTCSS

FSM générant la séquence de tonalité sous-audio

⏰ Temporisateur TOT

Time-Out Timer : FSM surveillant la durée d'émission

◌ Séquenceur TX/RX

Gestion de la commutation émission/réception

Décodeur DTMF

FSM détectant les tonalités duales

Contrôle de squelch

FSM avec hystérésis pour éviter le "battement"

Analyseur de protocole

Décodage de trames AX.25, APRS, etc.

Projet : Décodeur DTMF Simple

Objectif

Créer une FSM qui détecte une séquence de 4 chiffres DTMF pour déverrouiller un système.

Code secret : 1234

États de la FSM

État	Description	Sortie
IDLE	Attente touche 1	Verrouillé
DIGIT1	1 reçu, attente 2	Verrouillé
DIGIT2	12 reçu, attente 3	Verrouillé
DIGIT3	123 reçu, attente 4	Verrouillé
UNLOCK	1234 complet	**Déverrouillé**
ERROR	Mauvaise touche	Verrouillé

Extensions possibles

- Ajouter un timeout (retour IDLE après 5 secondes d'inactivité)
- Compteur de tentatives (blocage après 3 erreurs)
- Code programmable en mémoire
- Indicateur LED de progression

Points Clés à Retenir

- Les **FSM** modélisent des systèmes séquentiels complexes
- **Moore** : sorties = f(état) - plus stable
- **Mealy** : sorties = f(état, entrées) - plus rapide
- Le **diagramme d'états** est l'outil de conception principal
- Encodage des états : **binaire, Gray ou One-Hot**
- Implémentation : **bascules + logique combinatoire**
- Les FSM sont essentielles dans les **protocoles de communication**
- HDL (VHDL/Verilog) facilite la conception sur **FPGA**
- Toujours prévoir la gestion des **états d'erreur**

73 de F4HXN - Prochaine fiche : Applications radio amateur ! 📡

FICHE 9

Applications en Radio Amateur

L'algèbre de Boole au service de la radio : circuits pratiques et projets DIY

La Logique dans nos Transceivers

Les **circuits logiques** sont omniprésents dans l'équipement radioamateur moderne. De la simple LED d'indication au microprocesseur qui gère tout le transceiver, l'algèbre de Boole est au cœur de nos stations !

🔧 Où trouve-t-on de la logique dans une station radio ?

- **Contrôle PTT :** Logique de commutation TX/RX
- **Tonalités CTCSS/DCS :** Encodage et décodage
- **Afficheurs :** Décodeurs 7 segments, LCD
- **Interfaces CAT :** Communication série
- **Modes numériques :** Traitement DSP, codage/décodage
- **Relais :** Logique de contrôle, identification
- **Antennes :** Commutation automatique

CTCSS (Continuous Tone-Coded Squelch System)

Le **CTCSS** utilise une tonalité sous-audio (67-254 Hz) pour filtrer les appels. Seuls les correspondants utilisant le même code peuvent ouvrir votre squelch.

Principe de fonctionnement

Génération de la tonalité

Un **oscillateur numérique** génère la fréquence sub-audio :

1. **Compteur diviseur**

   Divise une horloge de référence (ex: 1 MHz) pour obtenir la fréquence CTCSS

2. **Table de forme d'onde**

   ROM ou RAM contenant les échantillons d'une sinusoïde

3. **CNA (Convertisseur Numérique-Analogique)**

   Convertit les échantillons numériques en signal analogique

4. **Filtre passe-bas**

   Lisse le signal pour obtenir une sinusoïde propre

Table des fréquences CTCSS standard

Code	Fréquence (Hz)	Code	Fréquence (Hz)	Code	Fréquence (Hz)
01	67.0	11	97.4	21	136.5
02	71.9	12	100.0	22	141.3
03	74.4	13	103.5	23	146.2
04	77.0	14	107.2	24	151.4
05	79.7	15	110.9	25	156.7

... et 38 autres codes jusqu'à 254.1 Hz

Décodeur CTCSS

Méthode par filtre actif

Utilise des filtres passe-bande très sélectifs :

- Chaque fréquence CTCSS a son propre filtre
- Un comparateur détecte la présence du signal filtré
- Une **porte AND** combine la détection + signal RF pour ouvrir le squelch

```
Squelch_Ouvert = Signal_RF_OK AND CTCSS_Détecté
```

Projet : Encodeur CTCSS Simple

Matériel nécessaire

- 1× NE555 ou ICM7555 (CMOS, moins de bruit)
- 1× Potentiomètre 10kΩ pour réglage fréquence
- 1× Condensateur 100nF
- 1× Condensateur 10µF
- Résistances diverses
- 1× Ampli op TL071 (buffer de sortie)

1. Câbler le 555 en mode astable
2. Ajuster R et C pour obtenir la fréquence souhaitée (ex: 88.5 Hz)
3. Filtrer la sortie carrée avec un filtre RC passe-bas
4. Buffer avec ampli op pour adapter l'impédance
5. Injecter dans l'entrée micro du transceiver (niveau ~10-50 mV)

💡 Amélioration :

Remplacer le 555 par un microcontrôleur (Arduino, PIC) pour générer plusieurs fréquences CTCSS programmables avec une meilleure précision !

DCS (Digital-Coded Squelch)

Le **DCS** est l'équivalent numérique du CTCSS. Au lieu d'une tonalité, un code numérique de 23 bits est transmis en continu à 134.4 bits/s.

🎯 Avantages du DCS :

- Plus de codes disponibles (104 codes vs 50 CTCSS)
- Meilleure immunité aux interférences
- Pas d'impact sur la qualité audio
- Codage numérique plus robuste

Structure du code DCS

Trame DCS

Le code est transmis en boucle :

```
Bit 0-11  : Code DCS (12 bits - ex: 023 = 000010011011)
Bit 12-22 : Code inversé (pour vérification)
Bit 23    : Bit de synchronisation
```

Débit : 134.4 bps → période d'un bit = 7.44 ms

Durée d'une trame complète : 23 bits × 7.44 ms ≈ 171 ms

Codes DCS standard

Code	Valeur octale	Code	Valeur octale	Code	Valeur octale
D023	023	D047	047	D114	114
D025	025	D051	051	D115	115
D026	026	D054	054	D116	116
D031	031	D065	065	D125	125
D032	032	D071	071	D131	131

... et 99 autres codes

Implémentation

Circuit DCS

Encodeur :

- Registre à décalage 23 bits chargeant le code en boucle
- Horloge 134.4 Hz (compteur + oscillateur)
- Modulateur pour générer les bits 0/1

Décodeur :

- Détecteur de bits (comparateur + échantillonneur)
- Registre à décalage 23 bits pour recevoir la trame
- Comparateur numérique vérifiant le code attendu
- FSM pour synchronisation et validation

⚠️ **Note :**

L'implémentation DCS nécessite un microcontrôleur ou un circuit dédié. Les CI spécialisés comme le TP3120 ou équivalents simplifient grandement la réalisation.

Logique de Contrôle de Relais

Un **relais radioamateur** est un système automatisé complexe nécessitant une logique de contrôle sophistiquée.

Fonctions logiques d'un relais

Gestion PTT

Détection de porteuse RF + CTCSS/DCS valide → activation émetteur

⏱️ Time-Out Timer (TOT)

Compteur limitant la durée d'émission (ex: 3 minutes)

Tail (queue)

Maintien TX quelques secondes après relâchement PTT

Identification

Envoi automatique de l'indicatif en morse toutes les 10 minutes

Anti-kerchunk

Filtre les émissions trop courtes (<0.5s)

Courtesy Tone

Bip de fin de transmission signalant que le relais est libre

Exemple : Logique d'activation du relais

Signal RF	CTCSS OK	TOT OK	Anti-kerchunk	TX activé
0	X	X	X	0
1	0	X	X	0
1	1	0	X	0
1	1	1	0	0
1	1	1	1	**1**

```
TX_Enable = RF_Present AND CTCSS_Valid AND
NOT(TOT_Exceeded) AND Kerchunk_OK
```

Projet : Contrôleur de Relais Simple

Matériel nécessaire

- 1× Arduino Nano ou équivalent
- 1× Module décodeur CTCSS (ex: SA828 intégré)
- 1× Relais 12V pour commutation TX/RX
- 1× Transistor 2N2222 (driver relais)
- 1× Diode 1N4148 (protection)
- Résistances et condensateurs

Pseudo-code Arduino

```
// Entrées
#define PIN_COR     2  // Carrier Operated Relay (RF détecté)
#define PIN_CTCSS   3  // CTCSS décodé
#define PIN_PTT_OUT 4  // Commande PTT émetteur

// Temporisations
#define TOT_LIMIT   180000  // 3 minutes en ms
#define TAIL_TIME   3000    // 3 secondes
#define MIN_TIME    500     // Anti-kerchunk 0.5s

void loop() {
    bool rf_present = digitalRead(PIN_COR);
    bool ctcss_ok = digitalRead(PIN_CTCSS);

    // Logique de contrôle
    if (rf_present && ctcss_ok && !tot_exceeded && kerchunk_ok) {
        digitalWrite(PIN_PTT_OUT, HIGH);  // Active TX
        // Gestion timers...
    } else {
        // Gestion tail...
        digitalWrite(PIN_PTT_OUT, LOW);   // Désactive TX
    }
}
```

🎯 Extensions possibles :

- Identification automatique en CW
- Courtesy tones multiples
- Télécommande DTMF
- Liaison Internet (EchoLink, IRLP)
- Enregistrement audio des QSO

Décodeur DTMF

Le **DTMF** (Dual-Tone Multi-Frequency) est utilisé pour la télécommande de relais, de stations, ou pour la numérotation.

Principe DTMF

🎵 Codage :

Chaque touche génère **deux fréquences simultanées** :

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Exemple : Touche "5" = 770 Hz + 1336 Hz

Décodage DTMF

Méthodes de décodage

1. Circuit intégré dédié

`MT8870` `CM8870`

Décodeur DTMF complet avec sortie 4 bits parallèle

2. Analyse par FFT (logiciel)

Transformée de Fourier pour détecter les fréquences présentes

3. Filtres Goertzel (DSP)

Algorithme optimisé pour détecter des fréquences spécifiques

Projet : Télécommande DTMF

Matériel nécessaire

- 1× MT8870 (décodeur DTMF)
- 1× Arduino Nano
- 1× Cristal 3.579545 MHz
- 1× Module relais 4 canaux
- Condensateurs et résistances

Fonctionnalités

- **Code d'accès :** 1234 pour déverrouiller
- **Commandes :**
  - *1 : Activer sortie 1
  - *2 : Activer sortie 2
  - #1 : Désactiver sortie 1
  - #0 : Tout désactiver
- **Sécurité :** Verrouillage après 3 échecs

1. Connecter MT8870 au récepteur audio
2. Lire les 4 bits de sortie (D0-D3) avec Arduino
3. Implémenter FSM pour vérification code et commandes

4. Contrôler les relais selon les commandes validées

Commutateur d'Antennes Automatique

Un commutateur d'antennes intelligent sélectionne automatiquement la meilleure antenne selon la bande.

Principe

- Détecter la bande utilisée (VHF, UHF, HF...)
- Utiliser un décodeur pour activer le relais d'antenne correspondant
- Assurer une seule antenne active à la fois

Logique de sélection

Bande détectée	Code	Antenne activée
144 MHz (2m)	00	ANT1 (VHF)
432 MHz (70cm)	01	ANT2 (UHF)
28 MHz (10m)	10	ANT3 (HF haute)
7 MHz (40m)	11	ANT4 (HF basse)

Implémentation

Utiliser un **décodeur 2:4** (74139) :

- 2 bits d'entrée → sélection parmi 4 antennes
- Chaque sortie commande un relais RF
- Logique de verrouillage pour éviter deux antennes actives

Analyseur de Spectre Numérique (FFT)

L'analyse spectrale utilise la **FFT** (Fast Fourier Transform) pour afficher le spectre RF. C'est un exemple parfait d'application des mathématiques et de la logique numérique.

🧮 Principe de la FFT :

- Échantillonnage du signal RF (ADC)
- Stockage dans une mémoire tampon (registres)
- Calcul FFT par un processeur ou FPGA
- Affichage du spectre sur écran

Applications pratiques

- **Panadapter :** Visualiser l'activité autour de la fréquence
- **Waterfall :** Suivi temporel du spectre
- **Détection signaux faibles :** Modes numériques (FT8, WSPR)
- **Analyse d'interférences**

◌ Contrôle Automatique de Gain (CAG/AGC)

Le **CAG numérique** utilise des circuits logiques pour maintenir un niveau de sortie constant malgré les variations du signal d'entrée.

Boucle de régulation

1 **Détection du niveau**

ADC mesure l'amplitude du signal RF

2 **Comparaison à la consigne**

Comparateur numérique (soustracteur)

3 **Calcul de correction**

ALU ou microprocesseur

4 **Application du gain**

DAC + amplificateur à gain variable

💡 Avantages du CAG numérique :

- Réponse plus rapide et précise
- Paramètres programmables (temps d'attaque, de relâchement)
- Plusieurs modes (lent, rapide, adaptatif)
- Pas de dérive due aux composants analogiques

Modes Numériques : FT8, WSPR, PSK31

Les **modes numériques** modernes sont entièrement basés sur le traitement numérique du signal (DSP).

FT8

Traitement :

- Échantillonnage audio 12 kHz
- FFT pour détection des 8 tonalités
- Décodage FEC (Forward Error Correction)
- Synchronisation temporelle précise

WSPR

Traitement :

- 4 tonalités FSK espacées de 1.46 Hz
- Durée : 110.6 secondes
- Corrélation pour extraction sous le bruit
- Décodage jusqu'à -28 dB S/N

⌨️ PSK31

Traitement :

- Modulation de phase (BPSK)
- Débit : 31.25 bauds
- Démodulateur cohérent
- Code Varicode pour compression

🧮 Circuits logiques impliqués :

- **ADC/DAC :** Conversion audio ↔ numérique
- **FFT :** Analyse fréquentielle
- **FIR/IIR :** Filtres numériques
- **Décodeurs FEC :** Correction d'erreurs
- **Corrélateurs :** Synchronisation
- **Mémoire :** Buffer audio, tables de décision

◌ Projet Final : Station de Relais Complète

Objectif

Réaliser un contrôleur de relais VHF complet avec toutes les fonctions modernes.

Liste du matériel

- 1× Arduino Mega 2560 (ou Teensy 4.0)
- 1× Module radio SA828 VHF (TX/RX intégré)
- 1× Décodeur DTMF MT8870
- 1× Ampli audio LM386
- 1× DAC MCP4725 (génération tonalités)
- 1× Écran LCD 20×4 I2C
- 1× Module RTC DS3231
- 1× Carte SD pour logging
- Relais, transistors, résistances, etc.

Fonctionnalités implémentées

🔐 **Sécurité d'accès :**

- CTCSS 88.5 Hz obligatoire
- Ou code DCS D023
- Ou code DTMF 1234#

⏱️ **Temporisations :**

- TOT : 3 minutes
- Tail : 5 secondes
- Anti-kerchunk : 0.5 seconde

🆔 **Identification :**

- Automatique toutes les 10 minutes
- En CW (morse) : F4HXN/R
- Synthèse vocale optionnelle

🎵 **Tonalités :**

- Courtesy tone configurable
- Tonalité d'avertissement TOT
- Bips de statut

📞 **Télécommande DTMF :**

- *00# : Statut vocal
- *01# : Température
- *02# : Tension batterie
- *99# : Redémarrage

📊 **Monitoring :**

- Température interne
- Tension d'alimentation
- ROS de l'antenne

- Puissance TX
- Nombre de QSO

💾 **Enregistrement :**

- Log des activations (SD card)
- Statistiques horaires/journalières
- Alertes système

Ressources et Références

Documentation

- ARRL Handbook
- The Art of Electronics
- AN444 (Motorola CTCSS)
- Datasheets des CI

Sites web

- REF (réseau des émetteurs)
- ARRL.org
- QRZ.com
- GitHub (projets open source)

Logiciels

- WSJT-X (FT8, WSPR)
- Fldigi (modes numériques)
- Arduino IDE
- KiCad (conception PCB)

Formations

- Examen radioamateur
- Cours en ligne (Coursera)
- YouTube (tutoriels)
- Clubs locaux

Points Clés à Retenir

- Le **CTCSS** utilise des tonalités sub-audio pour le squelch sélectif
- Le **DCS** est la version numérique avec 104 codes disponibles
- Les **relais** nécessitent une logique complexe (TOT, tail, identification)
- Le **DTMF** permet la télécommande par tonalités duales
- Le **CAG numérique** maintient un niveau audio constant
- Les **modes numériques** (FT8, WSPR) utilisent massivement le DSP
- Un **microcontrôleur** simplifie grandement les réalisations
- L'algèbre de Boole est **omniprésente** dans nos équipements radio

73 de F4HXN - Dernière fiche à venir ! 📡

FICHE 10 - FINALE

Programmation et Logique

De l'algèbre de Boole au code : opérateurs bit à bit, FPGA et au-delà

L'Algèbre de Boole en Programmation

Tout ce que nous avons appris sur l'algèbre de Boole et les circuits logiques trouve une **application directe en programmation**. Les langages modernes offrent des opérateurs bit à bit qui permettent de manipuler directement les bits.

🔑 Pourquoi les opérations bit à bit ?

- **Performance :** Opérations extrêmement rapides (1 cycle CPU)
- **Compacité :** Stocker plusieurs informations dans un seul entier
- **Bas niveau :** Contrôle précis des ports I/O, registres
- **Algorithmes :** Certains algorithmes s'appuient sur les bits
- **Protocoles :** Manipulation de trames, masques réseau

Opérateurs Bit à Bit

Les langages C, C++, Java, Python, JavaScript, etc. offrent tous des opérateurs bit à bit.

Table des opérateurs

Opération	C/C++/Java	Python	Description
AND	&	&	ET bit à bit
OR	|	|	OU bit à bit
XOR	^	^	OU exclusif
NOT	~	~	Complément
Shift Left	<<	<<	Décalage gauche
Shift Right	>>	>>	Décalage droite

Exemples en C

```
// Opérations de base
uint8_t a = 0b10110010;  // 178
uint8_t b = 0b11001100;  // 204

uint8_t and_result = a & b;  // 0b10000000 = 128
uint8_t or_result  = a | b;  // 0b11111110 = 254
uint8_t xor_result = a ^ b;  // 0b01111110 = 126
uint8_t not_result = ~a;     // 0b01001101 = 77 (sur 8 bits)

// Décalages
uint8_t left  = a << 2;  // 0b11001000 = 200 (× 4)
uint8_t right = a >> 2;  // 0b00101100 = 44  (÷ 4)
```

Applications pratiques

1️⃣ Tester un bit (masquage)

```
// Tester si le bit 3 est à 1
uint8_t value = 0b00101010;
if (value & (1 << 3)) {
    printf("Bit 3 est à 1\n");
}
```

2️⃣ Mettre un bit à 1 (set)

```
// Mettre le bit 5 à 1
value |= (1 << 5);  // value = 0b00101010 | 0b00100000 = 0b00101010
```

3️⃣ Mettre un bit à 0 (clear)

```
// Mettre le bit 3 à 0
value &= ~(1 << 3);  // value = 0b00101010 & 0b11110111 = 0b00100010
```

4️⃣ Inverser un bit (toggle)

```
// Inverser le bit 1
value ^= (1 << 1);  // XOR avec 1 inverse le bit
```

5️⃣ Extraire des champs de bits

```
// Registre de contrôle 8 bits : [MODE:2][SPEED:3][ENABLE:1][RESERVED:2]
uint8_t reg = 0b10101101;

// Extraire SPEED (bits 3-5)
uint8_t speed = (reg >> 3) & 0b111;  // Décale puis masque 3 bits
printf("Speed: %d\n", speed);  // Speed: 5

// Extraire MODE (bits 6-7)
uint8_t mode = (reg >> 6) & 0b11;
printf("Mode: %d\n", mode);  // Mode: 2
```

Exemple : Contrôle des GPIO sur Arduino

Les microcontrôleurs comme l'Arduino utilisent massivement les opérations bit à bit pour contrôler les ports I/O.

Contrôle direct des registres

```
// Port D = pins 0-7 sur Arduino
// Au lieu de digitalWrite() qui est lent...

// Méthode lente (1 pin à la fois)
digitalWrite(8, HIGH);   // ~5 µs

// Méthode rapide (manipulation directe des registres)
PORTB |= (1 << 0);       // ~125 ns (40× plus rapide !)

// Configurer plusieurs pins en une seule opération
DDRB = 0b00111111;  // Pins 8-13 en sortie
PORTB = 0b00101010; // Écrire pattern binaire sur les 6 pins

// Lire l'état d'un port complet
uint8_t portState = PINB;
if (portState & (1 << 2)) {
    // Pin 10 est HIGH
}
```

🚀 Performance :

Les opérations bit à bit directes sur les registres sont jusqu'à **50× plus rapides** que les fonctions Arduino standard ! Crucial pour les applications temps réel.

Application Radio : Décodage de Trame

Décoder une trame APRS

Les trames APRS contiennent des informations codées en bits. Exemple simplifié :

```
// Trame APRS simplifiée (en réalité beaucoup plus complexe)
typedef struct {
    uint8_t flags;      // [TYPE:3][PRIORITY:2][RESERVED:3]
    uint16_t position;  // Latitude/Longitude codée
    uint8_t data;       // Altitude, vitesse, etc.
} APRSFrame;

void decodeAPRS(APRSFrame *frame) {
    // Extraire le type de message
    uint8_t type = (frame->flags >> 5) & 0b111;

    // Extraire la priorité
    uint8_t priority = (frame->flags >> 3) & 0b11;

    // Décoder la position (exemple simplifié)
    float lat = (frame->position >> 8) * 0.01;  // 8 bits MSB
    float lon = (frame->position & 0xFF) * 0.01; // 8 bits LSB

    printf("Type: %d, Priority: %d\n", type, priority);
    printf("Position: %.2f, %.2f\n", lat, lon);
}
```

FPGA et Logique Programmable

Les **FPGA** (Field-Programmable Gate Array) sont des circuits qui permettent d'implémenter n'importe quelle logique numérique de façon matérielle.

🔧 Qu'est-ce qu'un FPGA ?

Un FPGA contient des milliers (voire millions) de cellules logiques configurables interconnectées. On peut "programmer" ces connexions pour créer n'importe quel circuit numérique.

Architecture d'un FPGA

LUT (Look-Up Table)

Tables de vérité programmables

Implémentent les fonctions logiques

Flip-Flops

Bascules D pour séquentiel

Mémoire et registres

Interconnexions

Réseau de routage programmable

Connexions entre cellules

DSP Blocks

Multiplicateurs dédiés

Traitement du signal

Block RAM

Mémoire embarquée

Buffers, FIFOs, caches

I/O Blocks

Interfaces d'entrée/sortie

Communication externe

Avantages des FPGA

✨ Pourquoi utiliser un FPGA ?

- **Parallélisme massif :** Opérations simultanées
- **Performance :** Logique dédiée, pas de software overhead
- **Flexibilité :** Reconfigurable à volonté
- **Faible latence :** Traitement en temps réel
- **Prototypage :** Test avant fabrication d'ASIC

Applications en radio amateur

- **SDR (Software Defined Radio) :** Traitement numérique du signal RF
- **Filtres numériques :** FIR/IIR haute performance
- **Décodeurs rapides :** FT8, WSPR en temps réel
- **FFT temps réel :** Spectrogramme, waterfall
- **Modulateurs/démodulateurs**

VHDL et Verilog

Les **langages HDL** (Hardware Description Language) permettent de décrire du matériel de façon textuelle. Le code est ensuite "synthétisé" en portes logiques.

Exemple en VHDL : Additionneur 4 bits

```
-- Additionneur 4 bits en VHDL
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity adder_4bit is
    Port (
        A : in  STD_LOGIC_VECTOR (3 downto 0);
        B : in  STD_LOGIC_VECTOR (3 downto 0);
        Cin : in  STD_LOGIC;
        Sum : out  STD_LOGIC_VECTOR (3 downto 0);
        Cout : out  STD_LOGIC
    );
end adder_4bit;

architecture Behavioral of adder_4bit is
    signal temp : unsigned(4 downto 0);
begin
    -- Addition avec retenue
    temp <= ('0' & unsigned(A)) + ('0' & unsigned(B)) + ("0000" & Cin);

    -- Sortie
    Sum <= std_logic_vector(temp(3 downto 0));
    Cout <= temp(4);
end Behavioral;
```

Exemple en Verilog : Détecteur de séquence

```
// Détecteur de séquence "101" en Verilog
module sequence_detector (
    input clk,
    input reset,
    input data_in,
    output reg detected
);

    // États de la FSM
    parameter S0 = 2'b00;  // État initial
    parameter S1 = 2'b01;  // "1" détecté
    parameter S2 = 2'b10;  // "10" détecté
    parameter S3 = 2'b11;  // "101" détecté

    reg [1:0] state, next_state;

    // Logique séquentielle (registre d'état)
    always @(posedge clk or posedge reset) begin
        if (reset)
            state <= S0;
        else
            state <= next_state;
    end

    // Logique combinatoire (transitions)
    always @(*) begin
        case (state)
            S0: next_state = data_in ? S1 : S0;
            S1: next_state = data_in ? S1 : S2;
            S2: next_state = data_in ? S3 : S0;
            S3: next_state = data_in ? S1 : S2;
            default: next_state = S0;
        endcase
    end

    // Sortie (Moore : dépend uniquement de l'état)
    always @(*) begin
        detected = (state == S3);
    end

endmodule
```

💡 **Différences VHDL vs Verilog :**

- **VHDL :** Plus verbeux, typé fortement, style Ada/Pascal
- **Verilog :** Plus concis, style C, plus permissif
- **Choix :** Les deux sont équivalents en puissance

Simulation de Circuits

Avant de programmer un FPGA ou de fabriquer un circuit, il est essentiel de **simuler** son fonctionnement.

Outils de simulation

Logisim Evolution

Simulateur graphique éducatif

Gratuit, parfait pour apprendre

ModelSim

Simulateur HDL professionnel

Industrie, très complet

GTKWave

Visualiseur de formes d'onde

Open source, léger

Vivado Simulator

Xilinx FPGA

Intégré à Vivado IDE

Testbench en VHDL

Tester l'additionneur 4 bits

```
-- Testbench pour l'additionneur
entity adder_4bit_tb is
end adder_4bit_tb;

architecture Behavioral of adder_4bit_tb is
    -- Déclaration du composant à tester
    component adder_4bit
        Port ( A, B : in STD_LOGIC_VECTOR(3 downto 0);
               Cin : in STD_LOGIC;
               Sum : out STD_LOGIC_VECTOR(3 downto 0);
               Cout : out STD_LOGIC );
    end component;

    -- Signaux de test
    signal A, B, Sum : STD_LOGIC_VECTOR(3 downto 0);
    signal Cin, Cout : STD_LOGIC;

begin
    -- Instanciation du composant
    UUT: adder_4bit port map (A, B, Cin, Sum, Cout);

    -- Processus de test
    stim_proc: process
    begin
        -- Test 1 : 5 + 3 = 8
        A <= "0101"; B <= "0011"; Cin <= '0';
        wait for 10 ns;
        assert (Sum = "1000" and Cout = '0')
            report "Test 1 failed!" severity error;

        -- Test 2 : 15 + 1 = 16 (overflow)
        A <= "1111"; B <= "0001"; Cin <= '0';
        wait for 10 ns;
        assert (Sum = "0000" and Cout = '1')
            report "Test 2 failed!" severity error;

        -- Fin de simulation
        wait;
    end process;
end Behavioral;
```

Projet Final : Récepteur SDR sur FPGA

Objectif ambitieux

Créer un récepteur SDR VHF simple sur FPGA pour décoder le trafic radio amateur.

Architecture du système

1 **Frontend RF**

ADC rapide (50-100 MSPS) pour numériser le signal RF directement

2 **Mélangeur numérique (DDC)**

Digital Down Converter : transposer la fréquence RF en bande de base

3 **Filtres FIR/IIR**

Filtrage passe-bande pour sélectionner le canal désiré

4 **Démodulateur FM**

Extraction du signal audio à partir de la FM

5 Décodeur CTCSS/DCS

Filtrage et détection des codes d'accès

6 Interface audio

DAC ou I2S vers codec audio

7 Contrôle et affichage

Microcontrôleur ou soft-core pour l'interface utilisateur

🛠️ Matériel suggéré :

- **FPGA :** Xilinx Artix-7 ou Intel Cyclone V
- **Board :** Digilent Arty A7 ou similaire (~100€)
- **ADC :** Module AD9361 ou LTC2208 (IF sampling)
- **Frontend :** RTL-SDR modifié ou circuit discret

⚠️ Projet avancé :

Ce projet nécessite des connaissances solides en DSP, FPGA et RF. C'est un excellent objectif à long terme après avoir maîtrisé les bases !

Python et Logique

Python peut aussi être utilisé pour travailler avec la logique, notamment pour le prototypage rapide et la simulation.

Simuler un circuit logique

```
# Simulation d'un additionneur complet en Python
def full_adder(a, b, cin):
    """Additionneur complet : retourne (sum, cout)"""
    sum_bit = a ^ b ^ cin          # XOR à 3 entrées
    cout = (a & b) | (cin & (a ^ b))  # Logique de retenue
    return sum_bit, cout

def ripple_carry_adder(a_bits, b_bits, cin=0):
    """Additionneur 4 bits par propagation de retenue"""
    result = []
    carry = cin

    for a, b in zip(a_bits, b_bits):
        sum_bit, carry = full_adder(a, b, carry)
        result.append(sum_bit)

    return result, carry

# Test
a = [1, 0, 1, 0]  # 5 en binaire (LSB first)
b = [1, 1, 0, 0]  # 3 en binaire
sum_bits, cout = ripple_carry_adder(a, b)

print(f"A = {a} (5)")
print(f"B = {b} (3)")
print(f"Sum = {sum_bits} (8)")
print(f"Carry out = {cout}")
```

Bibliothèques Python pour l'électronique

- **MyHDL :** Écrire du HDL en Python
- **Amaranth (ex-nMigen) :** Framework Python pour FPGA
- **PySpice :** Simulation de circuits analogiques
- **SciPy/NumPy :** Traitement du signal (FFT, filtres)

Pour Aller Plus Loin

Livres recommandés

- Digital Design (Morris Mano)
- Computer Organization (Patterson & Hennessy)
- RTL Hardware Design (Pong Chu)
- The Art of Electronics (Horowitz & Hill)

Cours en ligne

- Nand2Tetris (Build a computer)
- MIT 6.004 (Computation Structures)
- Coursera : Digital Systems
- YouTube : Ben Eater, Computerphile

◌ Kits et plateformes

- Arduino (microcontrôleurs)
- Raspberry Pi Pico (RP2040)
- FPGA boards (Arty, DE0-Nano)
- Kits d'électronique numérique

Logiciels gratuits

- Logisim Evolution
- KiCad (conception PCB)
- Arduino IDE
- Icarus Verilog (simulateur)

Félicitations !

Vous avez terminé la série complète sur l'Algèbre de Boole !

De la simple conversion binaire aux FPGA, en passant par les circuits combinatoires, séquentiels, et les machines à états, vous avez maintenant une solide compréhension des fondements de l'électronique numérique.

Cette connaissance est la base de TOUT ce qui est numérique : ordinateurs, smartphones, transceivers, satellites...

Récapitulatif de la Série

FICHE 1 : Binaire

Conversions, arithmétique binaire

FICHE 2 : Algèbre de Boole

Opérations logiques, lois fondamentales

FICHE 3 : Portes Logiques

Symboles, familles TTL/CMOS

FICHE 4 : Numération

Octal, hexa, BCD, Gray, signés

FICHE 5 : Karnaugh

Simplification graphique des circuits

FICHE 6 : Combinatoires

Additionneurs, MUX, décodeurs, ALU

FICHE 7 : Séquentiels

Bascules, registres, compteurs, mémoires

FICHE 8 : FSM

Machines à états Moore et Mealy

FICHE 9 : Applications Radio

CTCSS, DCS, relais, DTMF, modes numériques

FICHE 10 : Programmation

Opérateurs bit à bit, FPGA, VHDL

Vos Prochaines Étapes

1 **Pratiquez !**

Réalisez les projets proposés dans les fiches. Rien ne vaut l'expérience pratique.

2 **Explorez les FPGA**

Investissez dans une carte FPGA d'entrée de gamme et suivez des tutoriels VHDL/Verilog.

3 **Approfondissez le SDR**

Expérimentez avec GNU Radio, GQRX, et les modes numériques (FT8, WSPR).

4 **Rejoignez la communauté**

Participez aux activités de votre club radioamateur local, partagez vos projets.

5 **Continuez d'apprendre**

L'électronique et l'informatique évoluent constamment. Restez curieux !

Points Clés de la FICHE 10

- Les **opérateurs bit à bit** permettent de manipuler directement les bits en programmation
- Les opérations bit à bit sont **extrêmement rapides** (1 cycle CPU)
- Essentielles pour le **contrôle de GPIO**, protocoles, et optimisations
- Les **FPGA** implémentent de la logique matérielle programmable
- **VHDL et Verilog** sont les langages pour décrire le matériel
- La **simulation** est cruciale avant l'implémentation
- Les **SDR sur FPGA** offrent une flexibilité exceptionnelle
- L'algèbre de Boole est la **fondation** de toute l'informatique moderne

Série Complétée !

10 fiches • 10 heures de contenu • 100% terminé

Vous maîtrisez maintenant les fondements de l'électronique numérique,
de la théorie à la pratique, du circuit aux FPGA.

73 de F4HXN 📡

Bon trafic et bonnes expérimentations !