Architecture des systèmes numériques et informatiques
TD4 : Circuits Logiques Combinatoires

Halim Djerroud

5 novembre 2025

Exercice 1 : Demi-additionneur

On souhaite concevoir un demi-additionneur qui additionne deux bits A et B sans retenue entrante.

1. Établir la table de vérité complète avec les sorties S (somme) et R (retenue).
2. Extraire les expressions booléennes de S et R à partir de la table de vérité.
3. Simplifier l'expression de S en utilisant la porte XOR.
4. Dessiner le logigramme avec :
   — Des portes de base (AND, OR, NOT)
   — Une porte XOR pour la somme
5. Vérifier le fonctionnement pour tous les cas : $(A, B) = (0, 0), (0, 1), (1, 0), (1, 1)$

Solution : Table de vérité :

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

Expressions booléennes :
— $S = A'B + AB' = A \oplus B$ (XOR)
— $R = AB$ (AND)
Vérifications :
— $(0, 0)$: $S = 0,\ R = 0 \to 0 + 0 = 00_2$
— $(0, 1)$: $S = 1,\ R = 0 \to 0 + 1 = 01_2$
— $(1, 0)$: $S = 1,\ R = 0 \to 1 + 0 = 01_2$
— $(1, 1)$: $S = 0,\ R = 1 \to 1 + 1 = 10_2$
Le circuit nécessite : 1 porte XOR et 1 porte AND.

Exercice 2 : Additionneur complet

Un additionneur complet possède trois entrées : A, B et R_{in} (retenue entrante).

1. Établir la table de vérité complète (8 lignes).
2. Extraire les expressions de S (somme) et R_{out} (retenue sortante) en somme de mintermes.
3. Utiliser des tableaux de Karnaugh pour simplifier :
   — S (vous devriez trouver une fonction XOR triple)
   — R_{out} (fonction majorité)
4. Vérifier que $R_{out} = AB + AR_{in} + BR_{in}$
5. Montrer comment construire un additionneur complet avec deux demi-additionneurs.

Solution : Table de vérité :

A	B	R_{in}	S	R_{out}
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

K-map pour R_{out} (A en ligne, B, R_{in} en colonnes) :

	00	01	11	10
$A = 0$	0	0	1	0
$A = 1$	0	1	1	1

Groupements :
— Groupe de 2 (ligne $A = 1$, colonnes 01-11) : $A \cdot R_{in}$
— Groupe de 2 (ligne $A = 1$, colonnes 10-11) : $A \cdot B$
— Groupe de 2 (colonne 11, lignes 0-1) : $B \cdot R_{in}$
Expression simplifiée : $R_{out} = AB + AR_{in} + BR_{in}$
Pour S : $S = A \oplus B \oplus R_{in}$ (XOR triple)
Construction avec 2 demi-additionneurs :
— HA1 : $A + B \to S_1, R_1$
— HA2 : $S_1 + R_{in} \to S, R_2$
— $R_{out} = R_1 + R_2$ (OR)

Exercice 3 : Additionneur 4 bits

On souhaite additionner deux nombres de 4 bits : $A = A_3A_2A_1A_0$ et $B = B_3B_2B_1B_0$.

1. Dessiner le schéma bloc d'un additionneur 4 bits en cascade (ripple carry).
2. Calculer : $A = 1011_2$ (11) et $B = 0110_2$ (6)
   — Donner l'état de chaque additionneur complet
   — Identifier les retenues intermédiaires
   — Vérifier le résultat final
3. Quel est le problème de l'additionneur en cascade ?
4. Calculer le délai de propagation total si chaque additionneur complet a un délai de 10 ns.
5. Comment pourrait-on accélérer ce circuit ? (Mentionner le concept de retenue anticipée)

Solution : Calcul de $1011_2 + 0110_2$:

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

Résultat : $10001_2 = 17_{10}$
Vérification : $11 + 6 = 17$
Délai de propagation :
— Chaque additionneur : 10 ns
— 4 additionneurs en cascade : $4 \times 10 = 40$ ns
— La retenue doit se propager à travers tous les étages
Solution : Additionneur à retenue anticipée (carry look-ahead) qui calcule toutes les retenues en parallèle.

Exercice 4 : Comparateur 2 bits

Concevoir un comparateur qui compare deux nombres de 2 bits : $A = A_1A_0$ et $B = B_1B_0$.
Les sorties sont :
- E : égal ($A = B$)
- G : plus grand ($A > B$)
- L : plus petit ($A < B$)

1. Établir la table de vérité complète (16 lignes).
2. Construire les tableaux de Karnaugh pour E, G et L.
3. Simplifier les trois expressions.
4. Vérifier que $E = (A_1 \odot B_1) \cdot (A_0 \odot B_0)$ où $\odot$ est XNOR.
5. Tester votre circuit pour :
   - $A = 11_2$, $B = 10_2$
   - $A = 01_2$, $B = 10_2$
   - $A = 10_2$, $B = 10_2$

Solution partielle : Pour E (égalité) :
- $E = 1$ ssi $A_1 = B_1$ ET $A_0 = B_0$
- $E = (A_1 \odot B_1) \cdot (A_0 \odot B_0)$
- $E = \overline{A_1 \oplus B_1} \cdot \overline{A_0 \oplus B_0}$

Pour G (A $>$ B) :
- Si $A_1 > B_1$: automatiquement $A > B$
- Si $A_1 = B_1$: comparer A_0 et B_0
- $G = A_1\overline{B_1} + (A_1 \odot B_1) \cdot A_0\overline{B_0}$

Pour L (A $<$ B) :
- $L = \overline{A_1}B_1 + (A_1 \odot B_1) \cdot \overline{A_0}B_0$

Tests :
- $11_2 > 10_2$: $G = 1$, $E = 0$, $L = 0$
- $01_2 < 10_2$: $G = 0$, $E = 0$, $L = 1$
- $10_2 = 10_2$: $G = 0$, $E = 1$, $L = 0$

Propriété : $E + G + L = 1$ (une seule sortie active)

Exercice 5 : Multiplexeur 4 :1

Un multiplexeur 4 :1 possède 4 entrées de données (I_0, I_1, I_2, I_3), 2 entrées de sélection (S_1, S_0) et 1 sortie (Y).

1. Établir la table de vérité du multiplexeur.
2. Écrire l'expression booléenne de Y en fonction de S_1, S_0 et des entrées.
3. Dessiner le logigramme complet du multiplexeur.
4. Application : Utiliser un MUX 4 :1 pour implémenter la fonction $F(A, B) = A'B + AB'$ (XOR).
   - Choisir A et B comme entrées de sélection
   - Déterminer les valeurs à connecter sur I_0, I_1, I_2, I_3
5. Généralisation : Peut-on implémenter n'importe quelle fonction de 2 variables avec un MUX 4 :1 ?

Solution : Table de vérité :

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

Expression booléenne :

$$Y = \overline{S_1}\,\overline{S_0}\,I_0 + \overline{S_1}S_0I_1 + S_1\overline{S_0}I_2 + S_1S_0I_3$$

Pour implémenter $F(A, B) = A \oplus B$:

— Utiliser $S_1 = A$ et $S_0 = B$
— Table de vérité de XOR :

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

— Connexions : $I_0 = 0$, $I_1 = 1$, $I_2 = 1$, $I_3 = 0$

Généralisation : Oui ! Un MUX 2^n : 1 peut implémenter n'importe quelle fonction de n variables en connectant les valeurs de la table de vérité aux entrées de données.

Exercice 6 : Démultiplexeur 1 :4

Un démultiplexeur 1 :4 possède 1 entrée de données (I), 2 entrées de sélection (S_1, S_0) et 4 sorties (Y_0, Y_1, Y_2, Y_3).

1. Établir la table de vérité complète.
2. Écrire les expressions booléennes pour chaque sortie.
3. Dessiner le logigramme.
4. Quelle est la relation entre MUX et DEMUX ?
5. Application : Utiliser un DEMUX 1 :4 pour contrôler 4 LEDs avec un seul signal.
   — Comment sélectionner quelle LED allumer ?
   — Peut-on allumer plusieurs LEDs simultanément ?

Solution : Table de vérité :

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

Expressions :

$$Y_0 = \overline{S_1}\,\overline{S_0} \cdot I$$
$$Y_1 = \overline{S_1} S_0 \cdot I$$
$$Y_2 = S_1 \overline{S_0} \cdot I$$
$$Y_3 = S_1 S_0 \cdot I$$

Relation MUX/DEMUX : Ce sont des opérations inverses
— MUX : plusieurs entrées → une sortie
— DEMUX : une entrée → plusieurs sorties

Pour les LEDs :
— $S_1 S_0$ sélectionne quelle LED
— $I = 1$ pour allumer, $I = 0$ pour éteindre
— Une seule LED à la fois avec un DEMUX standard
— Pour plusieurs LEDs : utiliser plusieurs DEMUX ou un décodeur

Exercice 7 : Décodeur 3 :8

Un décodeur 3 :8 convertit un code binaire de 3 bits en 8 sorties (une seule active).

1. Établir la table de vérité pour les entrées E_2, E_1, E_0 et sorties S_0 à S_7.
2. Écrire l'expression de chaque sortie S_i.

3. Comment utiliser un décodeur 3 :8 pour :
   — Adresser 8 emplacements mémoire ?
   — Implémenter n'importe quelle fonction de 3 variables ?
4. Dessiner comment cascader deux décodeurs 3 :8 avec une entrée ENABLE pour créer un décodeur 4 :16.

Solution : Table de vérité :

E_2	E_1	E_0	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Expressions :

$$S_0 = \overline{E_2}\,\overline{E_1}\,\overline{E_0}$$
$$S_1 = \overline{E_2}\,\overline{E_1}E_0$$
$$\vdots$$
$$S_7 = E_2E_1E_0$$

Chaque sortie S_i correspond au minterm i.
Adressage mémoire :
— Les 3 bits d'adresse sont connectés à $E_2E_1E_0$
— Chaque sortie S_i active le module mémoire i
— Un seul module actif à la fois
Implémentation de fonctions :
— Chaque sortie représente un minterm
— Pour implémenter F : faire un OR des sorties correspondant aux mintermes de F
— Exemple : $F(A, B, C) = \sum m(0, 2, 5, 7) \rightarrow \text{OR}(S_0,\ S_2,\ S_5,\ S_7)$
Décodeur 4 :16 :
— Utiliser le 4ème bit comme ENABLE
— Premier décodeur actif si bit 3 = 0
— Deuxième décodeur actif si bit 3 = 1

Exercice 8 : Encodeur prioritaire 4 :2

Un encodeur prioritaire 4 :2 encode 4 entrées en un code binaire de 2 bits, avec gestion de priorités.
Priorités : $E_3 > E_2 > E_1 > E_0$

1. Établir la table de vérité avec :
   — Entrées : E_3, E_2, E_1, E_0
   — Sorties : S_1, S_0 (code binaire) et V (valid)
2. Utiliser des "don't care" (X) pour les entrées de priorité inférieure.
3. Simplifier les expressions avec des K-maps.
4. Que représente la sortie V ?
5. Tester pour :
   — $E_3E_2E_1E_0 = 0000 \rightarrow ?$
   — $E_3E_2E_1E_0 = 0101 \rightarrow ?$ (entrées 2 et 0 actives)
   — $E_3E_2E_1E_0 = 1111 \rightarrow ?$ (toutes actives)

Solution : Table de vérité avec priorités :

E_3	E_2	E_1	E_0	S_1	S_0	V	Commentaire
0	0	0	0	X	X	0	Aucune entrée
0	0	0	1	0	0	1	E_0 active
0	0	1	X	0	1	1	E_1 prioritaire
0	1	X	X	1	0	1	E_2 prioritaire
1	X	X	X	1	1	1	E_3 prioritaire

Expressions simplifiées :

$$S_1 = E_3 + E_2$$
$$S_0 = E_3 + \overline{E_2} \cdot E_1$$
$$V = E_3 + E_2 + E_1 + E_0$$

V (Valid) indique si au moins une entrée est active.
Tests :
— 0000 : $S_1S_0 = XX$, $V = 0$ (pas d'entrée valide)
— 0101 : E_2 prioritaire $\rightarrow S_1S_0 = 10$, $V = 1$
— 1111 : E_3 prioritaire $\rightarrow S_1S_0 = 11$, $V = 1$

Exercice 9 : Unité Arithmétique et Logique (ALU) simple

Concevoir une ALU 1 bit qui effectue 4 opérations sélectionnées par S_1S_0 :
— 00 : AND ($Y = A \cdot B$)
— 01 : OR ($Y = A + B$)
— 10 : XOR ($Y = A \oplus B$)
— 11 : ADD ($Y = A + B$ avec retenue)

1. Établir la table de vérité complète (4 entrées : A, B, S_1, S_0).
2. Concevoir le circuit en utilisant :
   — Un bloc de portes logiques (AND, OR, XOR)
   — Un additionneur complet
   — Un multiplexeur 4 :1 pour sélectionner la sortie
3. Dessiner le schéma bloc de l'ALU.
4. Comment généraliser cette ALU à n bits ?
5. Quelles autres opérations pourrait-on ajouter ? (NOT, NAND, décalage...)

Solution partielle : Pour $A = 1$, $B = 0$:

S_1S_0	Opération	Y
00	AND	0
01	OR	1
10	XOR	1
11	ADD	1

Architecture :

```
Entrées A, B
    ↓

 Portes
 - AND gate     → I
 - OR gate      → I
 - XOR gate     → I   MUX 4:1
 - Additionneur→ I      ↓
           Y
                         ↑
                    SS (sélection)
```

Pour n bits :
— Dupliquer le circuit n fois
— Chaîner les retenues pour l'addition
— Les opérations logiques sont indépendantes bit à bit
Opérations supplémentaires :
— NOT : $\overline{A}$
— NAND : $\overline{A \cdot B}$
— Décalage gauche/droite
— Soustraction : $A - B$ (utiliser complément à 2)
— Incrémentation : $A + 1$

Exercice 10 : Convertisseur BCD vers 7 segments

Un afficheur 7 segments affiche les chiffres 0-9 en BCD (Binary Coded Decimal).
Les segments sont notés a, b, c, d, e, f, g :

```
   a
  ---
f|   |b
 |_g_|
e|   |c
 |___|
   d
```

1. Pour les chiffres 0 à 9, établir la table de vérité indiquant quels segments sont allumés.
2. Les codes BCD 10-15 sont invalides. Que devrait afficher le circuit ?
3. Simplifier l'expression du segment a avec un K-map :
   — $a = 1$ pour : 0, 2, 3, 5, 6, 7, 8, 9
   — Utiliser "don't care" (X) pour les codes 10-15
4. Faire de même pour le segment g (barre horizontale centrale).
5. Estimer le nombre total de portes logiques nécessaires pour les 7 segments.

Solution partielle : Table segments pour 0-9 :

Chiffre	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10-15	-	-	-	-	X	X	X	X	X	X	X

K-map pour segment a (codes invalides = X) :

$AB \backslash CD$	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

Expression simplifiée (en utilisant les X) :

$$a = A + C + B\overline{D} + \overline{B}D$$

Ou : $a = A + C + B \oplus D$
Pour le segment g :

— $g = 0$ pour : 0, 1, 7
— $g = 1$ pour : 2, 3, 4, 5, 6, 8, 9
Nombre de portes : Chaque segment nécessite 3-5 portes → total 30 portes pour les 7 segments.
Codes invalides : Peuvent afficher n'importe quoi (utiliser X dans K-map pour simplifier).

Exercice 11 : Problème de synthèse

On souhaite concevoir un système de contrôle d'ascenseur simple avec 3 étages.
Entrées :
— E_1E_0 : Étage actuel (00, 01, 10 pour étages 0, 1, 2)
— D_1D_0 : Étage de destination
Sorties :
— UP : Monter (1 si destination > position actuelle)
— $DOWN$: Descendre (1 si destination < position actuelle)
— $STOP$: Arrêt (1 si arrivé à destination)

1. Établir la table de vérité complète.
2. Simplifier UP, $DOWN$ et $STOP$ avec des K-maps.
3. Dessiner le circuit logique complet.
4. Vérifier le fonctionnement pour quelques cas :
   — Actuel = 0, Destination = 2
   — Actuel = 2, Destination = 0
   — Actuel = 1, Destination = 1
5. Que se passe-t-il si on code l'étage 2 en binaire avec 11 ? Faut-il gérer ce cas ?

Solution partielle : La sortie $STOP = 1$ ssi $E_1E_0 = D_1D_0$ (comparateur d'égalité).
$STOP = (E_1 \odot D_1) \cdot (E_0 \odot D_0)$
Pour UP : il faut comparer les étages comme des nombres. Si $E_1E_0 < D_1D_0$ alors $UP = 1$
Pour $DOWN$: si $E_1E_0 > D_1D_0$ alors $DOWN = 1$
C'est essentiellement un comparateur 2 bits !
Cas spécial 11 (étage 3 inexistant) :
— Utiliser "don't care" dans le K-map
— Ou interdire explicitement ces combinaisons
Vérifications :
— $(0, 2)$: $UP = 1$, $DOWN = 0$, $STOP = 0$
— $(2, 0)$: $UP = 0$, $DOWN = 1$, $STOP = 0$
— $(1, 1)$: $UP = 0$, $DOWN = 0$, $STOP = 1$

Exercice 12 : Projet - Calculatrice 4 bits

Projet de synthèse : Concevoir une calculatrice simple 4 bits.
Spécifications :
— Deux nombres de 4 bits : $A = A_3A_2A_1A_0$ et $B = B_3B_2B_1B_0$
— Opérations (codées sur 2 bits OP_1OP_0) :
   — 00 : Addition
   — 01 : Soustraction (utiliser complément à 2)
   — 10 : AND bit à bit
   — 11 : OR bit à bit
— Sorties : Résultat 4 bits + Retenue/Borrow + Zero flag
Questions :

1. Dessiner l'architecture globale (schéma bloc).
2. Concevoir l'additionneur/soustracteur 4 bits.
3. Comment détecter un résultat nul (Zero flag) ?
4. Comment implémenter la soustraction avec un additionneur ?
5. Tester avec :

— $A = 1010$, $B = 0011$, $OP = 00$ (addition)
— $A = 1100$, $B = 0101$, $OP = 10$ (AND)

6. Estimer le nombre de portes logiques nécessaires.

Solution partielle : Architecture :

```
A(4)    B(4)
 ↓       ↓

Additionneur→ R_add
/Soustracteur

      ↓

  AND/OR    → R_logic
  bit à bit

      ↓

 MUX 2:1    → Résultat(4) + Carry + Zero
 (sél: OP)
```

Soustraction $A - B$:
— Calculer $A + \overline{B} + 1$ (complément à 2)
— Inverser B avec des portes XOR contrôlées par OP_0
— Mettre $R_{in} = OP_0$ pour le $+1$
Zero flag :

$$Zero = \overline{R_3 + R_2 + R_1 + R_0}$$

Une grande porte NOR sur les 4 bits de résultat.
Tests :
— $1010 + 0011 = 1101$ $(10 + 3 = 13)$
— $1100 \wedge 0101 = 0100$
Estimation : 50-70 portes logiques (4 additionneurs complets + logique de contrôle + MUX).