IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

ELP111

Fonctions électroniques logiques et analogiques

Fiche séance

PC7 - ADDITIONNEUR BINAIRE

Sommaire

IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PC7 – Additionneur binaire

1. LES OBJECTIFS D'APPRENTISSAGE

OA7 Expliquer la différence entre un algorithme et une architecture matérielle

OA8 Produire l'architecture à retenue propagée de l'additionneur binaire

Enjeu : Notion de complexité dans un processeur matériel de traitement de l'information.

2. LES CONCEPTS

La section 2.1 traite des opérations arithmétiques effectuées sur les nombres binaires signés ou non signés : addition et soustraction, multiplication et division.

La section 2.2 traite des opérateurs arithmétiques, c'est-à-dire des circuits combinatoires permettant la mise en œuvre matérielle de ces opérations.

Ces opérateurs constituent les circuits élémentaires des unités arithmétiques et logiques qui constituent le cœur des micro-processeurs actuels.

Dans cette séance, nous partons de la fonctionnalité que l'on souhaite obtenir, l'addition, puis à partir de l'algorithme de calcul dit « scolaire » (addition posée en colonne), nous en déduisons une architecture matérielle. Cette architecture matérielle pourra être ensuite mise en œuvre dans un circuit programmable type FPGA (*Field Programmable Gate Array* ; voir module SIT212) ou bien dans un circuit intégré, de type ASIC (*Application Specific Integrated Circuit*) par exemple.

2.1 OPERATIONS ARITHMETIQUES

2.1.1 Addition et soustraction

Addition

Le principe de l'addition est, dans toutes les bases, similaire à celui de l'addition décimale : on additionne symbole par symbole en partant des poids faibles, et en propageant éventuellement une retenue.

Si le format des nombres est fixe (en binaire naturel ou en complément à 2), le résultat de l'addition peut donner lieu à un dépassement de capacité (*overflow*, en anglais). Cependant, selon le codage des nombres binaires, ce « problème » est géré différemment.

Dans le cas des nombres codés en binaire naturel, le résultat de l'addition de 2 nombres binaires codés sur *n* bits peut dépasser la plus grande valeur codable sur *n* bits (égale à $2^n - 1$ en binaire naturel). Dans ce cas, il suffit de prendre en compte la retenue sortante de l'addition, qui devient le bit de poids fort du résultat. Cela revient à prendre un bit supplémentaire sur le résultat (soit *n*+1 bits), pour obtenir le résultat correct.

Dans le cas où les nombres sont codés en complément à 2, ce n'est pas aussi simple.

Le résultat de l'addition de 2 nombres exprimés sur *n* bits sur le même format fournit toujours un résultat correct, sauf dans le cas où le résultat n'est pas représentable sur *n* bits. **Il y a alors dépassement de capacité. On observe que, dans ce cas, les deux opérandes sont de même signe et le résultat est de signe opposé (ce qui est incorrect)**.

Dans le registre d'état d'un ordinateur, deux indicateurs (ou *flags*, en anglais) viennent renseigner le programmeur (ou le système d'exploitation) sur la validité des résultats obtenus : la retenue sortante (*carry C*) et le débordement (overflow *OVF*).

L'indicateur *C* signale la présence d'une retenue au niveau des poids forts; l'indicateur *OVF* indique que le résultat de l'opération n'est pas représentable **dans le système du complément à 2 sur le format défini au départ** (c'est-à-dire le nombre de bits fixé des opérandes). Ces 2 indicateurs sont directement générés par le circuit arithmétique présent dans le micro-processeur.

⚠ le circuit arithmétique ne connaît pas le format des nombres qui lui sont donnés. Il calcule toujours : le résultat sur *n* bits, et les 2 indicateurs *C* et *OVF*.

Nous allons illustrer le positionnement de la retenue et du débordement par quatre exemples pour des nombres binaires signés (dans le format CA2) :

0000 0110 (+6)	0111 1111 (+127)	0000 0100 (+4)	0000 0100 (+4)
+ 0000 0100 (+4)	+ 0000 0001 (+1)	+ 1111 1110 (-2)	+ 1111 1100 (-4)
0000 1010 (+10)	1000 0000 (-128)	1 0000 0010 (+2)	1 0000 0000 (0)
OVF=0	OVF=1	OVF=0	OVF=0
C=0	C=0	C=1 ignoré	C=1 ignoré
résultat correct	résultat incorrect	résultat correct	résultat correct

Dans le 1er exemple, +10 est représentable sur 8 bits, le résultat est donc correct. Dans le 2ème exemple, +128 n'est pas représentable sur 8 bits, on obtient ainsi -128 en représentation CA2. Le résultat de l'addition de 2 nombres positifs est un nombre négatif et l'indicateur OVR est positionné à 1.

Pour que le résultat d'une opération sur *n* bits soit correct dans la représentation du complément à 2, il faut que les retenues de rang *n* et de rang *n*+1 soient identiques.

Reprenons maintenant les mêmes exemples mais en binaire naturel :

Dans les 1er et 2e exemples, le résultat en binaire naturel sur 8 bits est correct, la retenue est égale à 0. Dans le 3e exemple, on ajoute 4 et 126 en binaire naturel, on obtient 2 sur 8 bits (0000 0010). Le résultat est incorrect sur 8 bits. Cela est dû au dépassement de capacité puisque 130 n'est pas représentable sur 8 bits. Cependant, si maintenant on prend en compte la retenue, on obtient bien 130 (1 0000 0010), mais cette fois-ci sur 9 bits.

Soustraction

La soustraction, en arithmétique binaire, est appliquée sur des nombres signés. Dans ce cas, cette opération se ramène dans tous les cas à une addition. Ainsi, si l'on souhaite réaliser l'opération $A - B$, on effectue tout d'abord le complément à 2 de B, $\mathrm{CA2}(B) = -B$ puis on réalise l'addition $A + (-B)$.

Pour plus d'explications sur le complément à 2, consulter la PC3.

Illustration

La figure 2.1 illustre le principe du débordement (OVR) lors d'opérations arithmétiques en CA2.

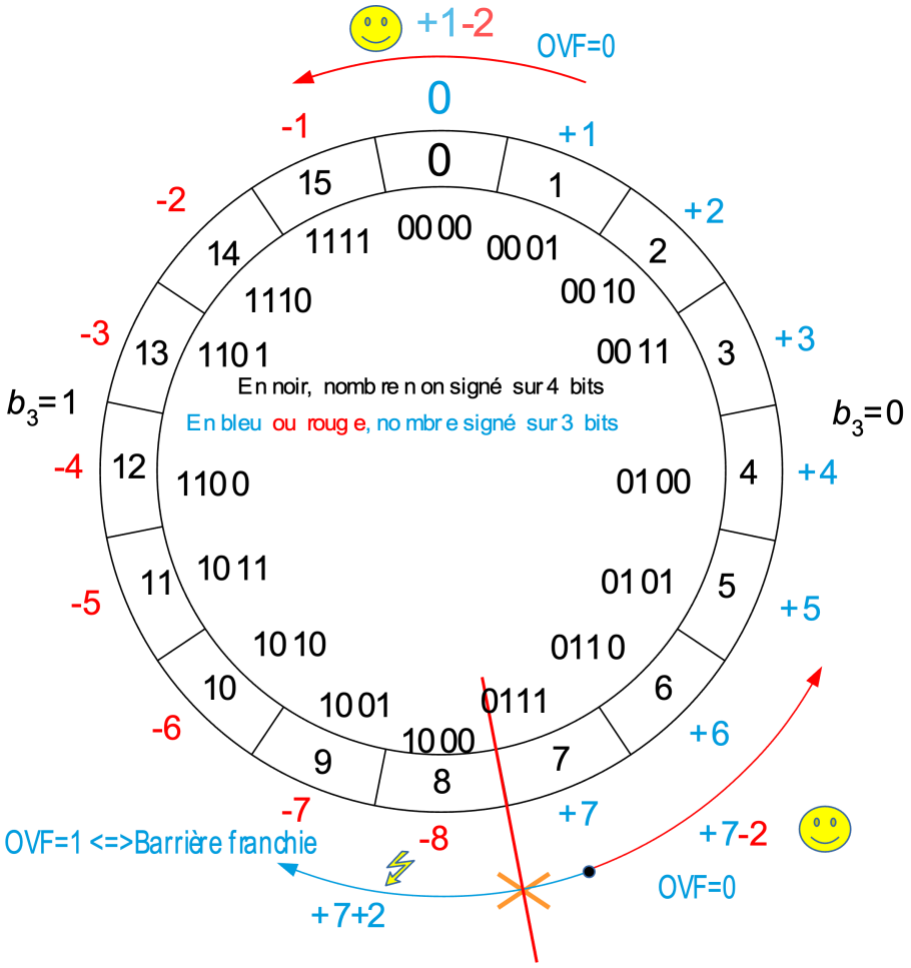


figure 2.1 : illustration de l'addition et soustraction en CA2 sur 4 bits. Principe de débordement (OVR).

Extension du bit de signe en CA2

Si on augmente le format d'un nombre positif en rajoutant un ou plusieurs "0" à gauche du nombre, on ne change pas la valeur du nombre :

0111 = 00111 = 000000111 = +7

Même chose pour les nombres négatifs où on ne change pas la valeur du nombre en ajoutant un ou plusieurs 1.

Donc, si on veut additionner deux nombres codés sur 4 bits (compris entre –8 et +7), on peut coder ces nombres sur 5 bits par une extension du bit de signe et on est sûr que le résultat sera correct sur 5 bits.

2.1.2 Multiplication et division

Les opérations de multiplication et de division sont plus complexes que l'addition, et ne sont traitées que partiellement dans ce chapitre.

Multiplication et division par 2^k

Le nombre 2 occupant une place privilégiée en numération binaire (tout comme le nombre 10 en numération décimale), les cas de la multiplication et de la division par 2^k peuvent être traités à part.

- Multiplication par 2^k

Multiplier un nombre binaire par 2^k consiste à décaler la virgule de *k* positions vers la droite, ou à ajouter *k* "0" au niveau des bits de poids faible dans le cas des entiers. Par exemple, soit $N = 18_{(10)} = 10010_{(2)}$. La multiplication de *N* par 2 donne $N.2_{(10)} = 36_{(10)} = 100100_{(2)}$, et sa multiplication par 4, $N.4_{(10)} = 72_{(10)} = 1001000_{(2)}$. Le mécanisme est le même que celui appliqué lors de la multiplication d'un nombre décimal par 10^k.

- Division par 2^k

Le mécanisme est inverse de celui de la multiplication. Diviser un nombre binaire par 2^k consiste à décaler la virgule de *k* positions vers la gauche, ou à enlever les *k* bits de poids faible dans le cas d'une division entière. Par exemple, soit $N = 37_{(10)} = 100101_{(2)}$. La division entière de *N* par 4 donne $N / 4_{(10)} = 9_{(10)} = 1001_{(2)}$, tandis que la même division considérée sur des réels donne $N / 4_{(10)} = 9,25_{(10)} = 1001,01_{(2)}$.

2.2 LES OPERATEURS ARITHMETIQUES

2.2.1 Les additionneurs

Dans la section 2.1, nous avons détaillé l'algorithme, très simple, qui permet d'additionner deux nombres binaires. Dans cette section, nous présentons les circuits combinatoires, appelés additionneurs, qui mettent en œuvre matériellement la fonction d'addition. L'étude détaillée de tous les algorithmes de calcul arithmétique sort du cadre de ce cours.

La structure des additionneurs a été intensément étudiée depuis les premières recherches sur l'architecture des calculateurs. En effet, la conception d'additionneurs rapides est importante pour effectuer des additions, et donc des multiplications et des divisions, de la manière la plus efficace possible.

Ainsi, après avoir posé le principe de base de l'addition de 2 chiffres binaires dans la section 2.1, deux architectures d'additionneurs de nombres binaires sont présentées : l'additionneur à **retenue propagée** et l'additionneur à **retenue anticipée**. Une architecture correspond à une instanciation matérielle d'un algorithme, c'est-à-dire à un agencement spécifique de ressources matérielles, typiquement présenté sous la forme d'un logigramme dans le cas d'un circuit combinatoire. Les deux architectures présentées dans cette section correspondent à deux compromis vitesse/encombrement différents.

Addition de deux bits

Le **demi-additionneur binaire** prend en entrée 2 éléments binaires A_k et B_k, et délivre en sortie leur somme S_k et une retenue C_k suivant la table de vérité du tableau 2.1.

A_k	B_k	S_k	C_k

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

tableau 2.1 : table de vérité du demi-additionneur binaire

Le comportement du circuit est régi par les équations $S_k = A_k \oplus B_k$ et $C_k = A_k . B_k$. L'opérateur d'addition arithmétique binaire est donc le OU exclusif.

Le schéma de la figure 2.2 représente la structure du demi-additionneur binaire.

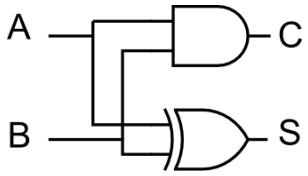


figure 2.2 : structure du demi-additionneur binaire

Pour additionner deux nombres binaires à plusieurs bits, une technique consiste à additionner successivement les bits de même poids avec la retenue de l'addition précédente. L'opérateur de base, appelé **additionneur complet**, doit alors prendre en compte une retenue entrante C_{k-1}.

A_k	B_k	retenue entrante C_{k-1}	somme S_k	retenue sortante C_k
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

tableau 2.2 : table de vérité de l'additionneur complet

Les équations logiques correspondantes peuvent s'écrire :

$$S_k = A_k \oplus B_k \oplus C_{k-1}$$

$$C_k = A_k\, B_k + (A_k + B_k)\, C_{k-1} = A_k\, B_k + (A_k \oplus B_k)\, C_{k-1}$$

L'additionneur complet peut, par exemple, être réalisé à partir de deux demi-additionneurs et d'un opérateur OU (figure 2.3).

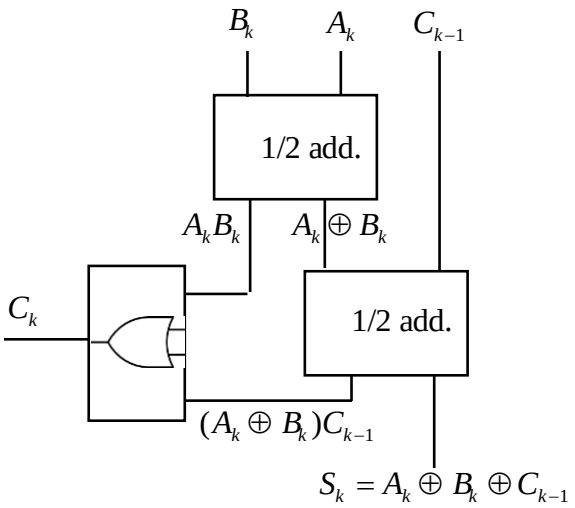


figure 2.3 : une réalisation de l'additionneur complet

Addition de deux nombres binaires

Additionneur à retenue propagée (ripple carry adder)

La solution la plus naturelle pour additionner deux nombres binaires *A* et *B* de *n* bits s'écrivant en numération de position $A_{n-1}A_{n-2}\cdots A_k\cdots A_1A_0$ et $B_{n-1}B_{n-2}\cdots B_k\cdots B_1B_0$ consiste à associer en cascade *n* étages d'additionneurs complets. Le résultat de l'addition est constitué de $n+1$ bits, soit *n* bits de somme $S_{n-1}S_{n-2}\cdots S_k\cdots S_1S_0$ et la retenue finale C_{n-1}. Chaque additionneur complet de rang *k* calcule la somme S_k et la retenue C_k nécessaire à l'étage suivant. La figure 2.4 présente la structure d'un additionneur de deux mots de *n* bits.

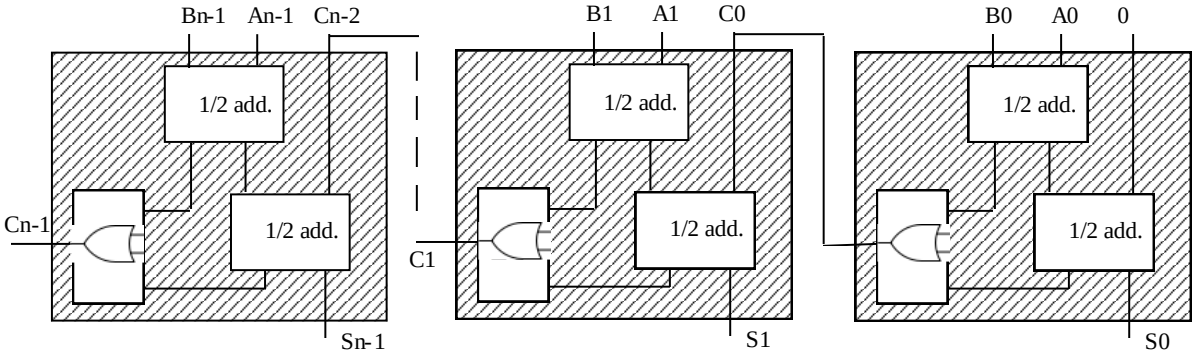


figure 2.4 : additionneur à retenue propagée ou série

Cette solution est intéressante d'un point de vue matériel car répétitive. Cependant, le résultat de l'opération n'est disponible en sortie de l'additionneur que lorsque toutes les retenues, du poids le plus faible au poids le plus fort, ont été calculées (d'où le nom d'additionneur à **retenue propagée**). Le temps de calcul du résultat est donc directement proportionnel à la taille des nombres manipulés et donc au nombre d'étages du dispositif, soit *n*. La technique de calcul anticipé de la retenue permet de remédier à ce défaut.

Additionneur à retenue anticipée (carry look-ahead adder)

Voir la section 4.

2.2.2 Les soustracteurs

Pour la soustraction $A - B$, on peut adopter la même approche que pour l'addition. On commence par définir l'opérateur binaire de base et on l'utilise pour réaliser des soustractions de nombres binaires. En pratique, se pose le problème de la représentation des nombres signés dans le cas où $B > A$. Pour résoudre ce problème, on convient d'une représentation des nombres négatifs adaptée et la soustraction est alors ramenée à une addition. La représentation généralement utilisée est celle du complément vrai ou complément à 2.

2.2.3 Les multiplieurs et diviseurs

Voir la section 4.

2.2.4 Les unités arithmétiques et logiques

Les fabricants de circuits intégrés proposent de nombreux composants (sous forme de circuits standard ou bien de macrofonctions) capables d'effectuer un ensemble d'opérations arithmétiques (addition, soustraction, ...) et logiques (ET, OU, OU exclusif, …). L'opération à effectuer est déterminée par des entrées de commande ou de sélection. Ces opérateurs, appelés UAL (Unité Arithmétique et Logique) ou ALU (Arithmetic and Logic Unit), sont présents dans de nombreux dispositifs de calcul comme les microprocesseurs. Un exemple du marché est donné à la section 4.1.

2.3 BIBLIOGRAPHIE

[BH90] J.-M. Bernard et J. Hugon, *Pratique des circuits logiques,* Collection technique et scientifique des télécommunications, Eyrolles, 1990.

[Mot] http://www.motorola.com/

[Mul89] J.-M. Muller, *Arithmétique des ordinateurs, opérateurs et fonctions élémentaires*, Etudes et recherches en informatique, Masson, 1989.

[NS] http://www.national.com/

[TI] http://www.ti.com/

[Aum96] M. Aumiaux, *Logique arithmétique et techniques synchrones*, 1ère partie : *Arithmétique binaire et décimale*, Enseignement de l'électronique, Masson, 1996.

3. LES EXERCICES D'APPLICATION

Dans cette séance, nous partons de la fonctionnalité que l'on souhaite obtenir, l'addition, puis à partir de l'algorithme de calcul dit « scolaire » (addition posée en colonne), nous en déduisons une architecture matérielle. Cette architecture matérielle pourra être ensuite mise en œuvre dans un circuit programmable type FPGA (*Field Programmable Gate Array* ; voir module SIT212) ou bien dans un circuit intégré, de type ASIC (*Application Specific Integrated Circuit*) par exemple.

A une fonction donnée peut correspondre plusieurs algorithmes. Et à chaque algorithme peut correspondre plusieurs architectures, et donc plusieurs circuits.

3.1 OPERATION ARITHMETIQUE : L'ADDITION, ALGORITHME ET CIRCUIT

3.1.1 Algorithme « scolaire »

Le bit de retenue *Carry* est généralement utilisé lorsque les opérandes sont **en binaire naturel.** Ce bit supplémentaire permet ainsi d'obtenir un résultat toujours correct sur *n+1*, si *n* est la taille des mots binaires en entrée.

Le bit de débordement (OVF, *overflow*) est utilisé lorsque les opérandes sont **en CA2**, c'est à dire nombres signés. Il indique que le résultat de l'addition est incorrect (OVF=1).

L'Unité Arithmétique et Logique (UAL) n'a pas connaissance du codage des opérandes. Elle calcule donc les 3 sorties : résultat, carry et OVR indépendamment du codage.

Compléter le tableau suivant. Vous calculez OVF en supposant que A et B sont en CA2.

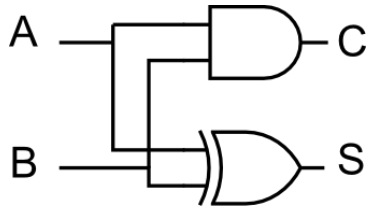
A A3 A2 A1 A0	B B3 B2 B1 B0	A+B S3 S2 S1 S0	Carry	OVF
1100	0010			
1010	1001			
0010	0111			
0011	1111			

3.1.2 Circuit Additionneur binaire complet

Le demi-additionneur binaire est un circuit qui prend en entrée 2 nombres de 1 bit (A_k et B_k) et génère leur somme (S_k) et une retenue (C_k).

A_k	B_k	S_k	C_k
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Le schéma ci-après représente la structure d'un "demi-additionneur binaire" :



A partir des informations ci-dessus, on souhaite réaliser un additionneur complet qui prenne en compte une retenue (C_{k-1}) générée à partir d'un étage précédent.

Table de vérité

- Etablir la table de vérité donnant la somme S_k et la retenue sortante C_k en fonction de A_k, B_k et C_{k-1}.
- Donner la 1ère forme canonique pour chacune de ces fonctions.

On souhaite réaliser un additionneur complet **à partir de deux demi-additionneurs et d'un opérateur combinatoire élémentaire *g***. L'objectif est donc de faire apparaître les fonctions somme (AND) et retenue sortante (XOR) des ½ additionneurs dans les fonctions de l'additionneur complet.

Obtention des fonctions logiques à partir de AND et XOR

- Pour la somme S_k, tracer le tableau de Karnaugh correspondant et simplifier la fonction.
- Pour la retenue sortante C_k, simplifier algébriquement à partir de la 1ère forme canonique et à l'aide des propriétés logiques (cf. PC3), pour faire apparaître les opérateurs AND et XOR.

Schéma de l'additionneur complet

- Identifier l'opérateur combinatoire élémentaire manquant *g*, puis
- Donner le *schéma d'architecture* de l'additionneur complet à partir de deux demi-additionneurs et de l'opérateur combinatoire élémentaire *g*.

Schéma de l'additionneur de mots de n bits (bonus)

En utilisant le module de l'additionneur complet, donnez le schéma d'architecture, d'un additionneur de mots de n bits capable d'effectuer l'opération $S = A + B$, (+ représente ici l'opérateur d'addition arithmétique), avec $A=(A_{n-1},\ldots A_k,\ldots,A_1,A_0)$, $B=(B_{n-1},\ldots B_k,\ldots,B_1,B_0)$, et $S=(S_{n-1},...S_k,...,S_1,S_0)$. La retenue finale est désignée par C_{n-1}.

3.2 OPERATION ARITHMETIQUE : LA SOUSTRACTION, ALGORITHME ET CIRCUIT (A FAIRE A LA MAISON)

3.2.1 Algorithme « scolaire », quelques exemples

Compléter le tableau suivant (A et B sont signés, représentés en CA2) :

+A A3 A2 A1 A0	+B B3 B2 B1 B0	-B	+A+(-B) S3 S2 S1 S0	OVF
1100	0010			

1010	1001			
0010	0111			
0011	1111			

1.1.1. Circuit Additionneur/soustracteur

On souhaite transformer le montage précédent en un additionneur / soustracteur. On rappelle que dans la représentation en complément à 2, $A - B = A + (-B) = A + \overline{B} + 1$.

Proposer le schéma d'un additionneur / soustracteur capable de manipuler des nombres de 4 bits codés en complément à 2.

Cet additionneur / soustracteur possèdera une entrée de commande *SOUSTRAC* qui sera utilisée comme suit :

- SOUSTRAC = 0, fonctionnement en additionneur
- SOUSTRAC = 1, fonctionnement en soustracteur

Pour cela, vous ré-utiliserez le schéma proposé à la question 3.2.4, auquel vous ajouterez des portes logiques et vous connecterez astucieusement les entrées.

Astuces :

- Commencez par trouver la fonction qui permet d'obtenir $\bar{B}$ ou B en fonction de la valeur de SOUSTRAC.
- Puis connectez les entrées astucieusement pour réaliser +1 ou +0 en fonction de la valeur de SOUSTRAC.

3.3 DEPASSEMENT DE CAPACITE : OVERFLOW (A FAIRE A LA MAISON)

Le montage précédent ne traite pas les problèmes de débordement de capacité. Le bit de sortie *OVF* (*OVF* = overflow) sera positionné à "1" lorsqu'il y aura débordement de capacité en arithmétique signée (CA2).

- A partir de la table de vérité de l'additionneur complet donnée dans l'énoncé, donner la table de vérité de C_{n-1}, S_{n-1} et *OVF* à partir de A_{n-1}, B_{n-1} et C_{n-2}.

Cn-2	*An*-1	*Bn*-1⊕*SOUSTRAC*	*Sn*-1	*Cn*-1	*OVF*

- En remarquant que *OVF* peut s'écrire uniquement en fonction des retenues C_{n-1} et C_{n-2}, donner l'équation logique du bit de sortie *OVF* en fonction des retenues C_{n-1} et C_{n-2}. Ainsi, dans la table de vérité de *OVF*, vous ne considèrerez que 2 entrées C_{n-1} et C_{n-2}.

4. POUR ALLER PLUS LOIN

Dans cette section, nous détaillons le cas de la multiplication et de son opérateur arithmétique, le multiplieur. Nous distinguons donc la fonction, de l'algorithme et de sa réalisation matérielle (architecture). Puis nous présentons un exemple d'unité arithmétique et logique sous forme de circuit discret.

4.1 CAS GENERAL DE LA MULTIPLICATION

La multiplication de deux nombres binaires de *n* bits fournit un résultat sur 2*n* bits. Lorsque les nombres ne sont pas signés, le principe est le même qu'en décimal et fait intervenir des produits partiels de bits, des additions et des décalages. A titre d'exemple, la multiplication de 2 nombres de 4 bits non signés, *A* et *B*, se décompose comme suit :

				A_3	A_2	A_1	A_0	Multiplicande *A*
			$\times$	B_3	B_2	B_1	B_0	Multiplieur *B*
				A_3B_0	A_2B_0	A_1B_0	A_0B_0	Produit partiel $A \times B_0$
			A_3B_1	A_2B_1	A_1B_1	A_0B_1		$A \times B_1$ décalé de 1 rang
		A_3B_2	A_2B_2	A_1B_2	A_0B_2			$A \times B_2$ décalé de 2 rangs
	A_3B_3	A_2B_3	A_1B_3	A_0B_3				$A \times B_3$ décalé de 3 rangs
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	Produit $P = A \times B$

La multiplication de *A* par *B* se déroule en trois étapes :

- Multiplication de A par chacun des symboles de B : en base 2, la multiplication de A par le symbole B_i revient à faire un ET logique entre chaque symbole de A et B_i,
- Décalage de $A \times B_i$ de i rangs vers la gauche,
- Addition des résultats de l'étape précédente.

Lorsque les nombres à multiplier sont signés, le principe de l'opération devient plus complexe et fait appel à des algorithmes non traités dans ce cours (cf. par exemple [Aum96]). Néanmoins, dans le cas d'une représentation en complément à 2, l'algorithme de multiplication précédent est applicable, moyennant une légère modification, si le multiplieur est positif :

- Si le multiplicande est positif, pas de changement,
- Si le multiplicande est négatif, tous les produits partiels sont négatifs ou nuls. Il faut étendre les produits partiels non nuls à 2*n* bits en rajoutant des 1 sur les bits de poids forts.

Exemple :

1 0 1 1 (-5)

			×	0	0	1	1	(+3)
1	1	1	1	1	0	1	1	
1	1	1	1	0	1	1		
0	0	0	0	0	0			
0	0	0	0	0				
1	1	1	1	0	0	0	1	(-15)

Dans les systèmes numériques, la division binaire n'est pas réalisée suivant la méthode utilisée en décimal [Aum96].

4.2 ADDITIONNEUR A RETENUE ANTICIPEE

Additionneur à retenue anticipée (carry look-ahead adder)

Le principe de l'additionneur à retenue anticipée consiste à calculer toutes les retenues en parallèle.

L'équation logique de la retenue sortante de l'additionneur complet établie en section 2.2.1 peut s'écrire :

$$C_k = A_k B_k + (A_k \oplus B_k) C_{k-1} = G_k + P_k C_{k-1}$$

Le tableau 4.3 permet d'analyser l'obtention de la retenue sortante à partir des termes P_k et G_k.

n° de ligne	A_k	B_k	C_{k-1}	C_k	P_k	G_k	
1	0	0	0	0	0	0	Retenue sortante à 0
2	0	0	1	0	0	0	
3	0	1	0	0	1	0	Retenue entrante
4	0	1	1	1	1	0	propagée vers
5	1	0	0	0	1	0	la sortie
6	1	0	1	1	1	0	
7	1	1	0	1	0	1	Retenue sortante à 1
8	1	1	1	1	0	1	

tableau 4.3 : table de vérité de la retenue pour un additionneur complet

On observe que :

- Lignes 1 et 2 : la retenue sortante C_k est à 0,
- Lignes 3 à 6 : la retenue entrante C_{k-1} est propagée vers la sortie, $C_k = C_{k-1}$. Ces 4 lignes de la table de vérité sont caractérisées par la relation $P_k = A_k \oplus B_k = 1$. P_k est ici appelé terme de propagation de retenue.
- Lignes 7 à 8 : une retenue sortante à 1 est délivrée en sortie. Ces 2 lignes sont caractérisées par la relation $G_k = A_k . B_k = 1$. G_k est appelé terme de génération de retenue. C'est encore l'expression de la retenue R_k en sortie d'un demi-additionneur (figure 2.2).

Dans le cas de l'addition de 2 nombres *A* et *B* de *n* bits, le bit de retenue de rang *k* est donc donné par la relation $C_k = G_k + C_{k-1}P_k$.

Le principe des additionneurs à retenue anticipée consiste à calculer :

- les couples (P_k, G_k) à l'aide de demi-additionneurs, pour $0 \le k \le n-1$,
- les retenues C_k, pour $0 \le k \le n-1$, avec $C_k = G_k + C_{k-1}P_k$, directement à partir des différents termes de propagation et de génération, et de la retenue entrante du premier étage C_{-1} : $C_0 = G_0 + C_{-1}P_0$, $C_1 = G_1 + C_0P_1 = G_1 + G_0P_1 + C_{-1}P_0P_1$, etc.
- les bits de somme $S_k = P_k \oplus C_{k-1}$, pour $0 \le k \le n-1$.

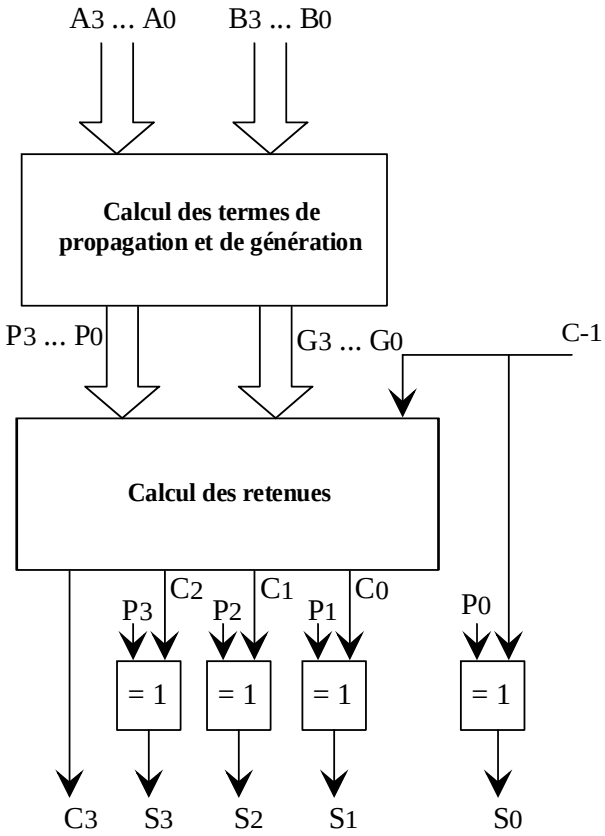


figure 4.5 : structure d'un additionneur de mots de 4 bits à retenue anticipée

Cette solution architecturale mène à des circuits plus rapides que les additionneurs à retenue propagée, car le nombre de blocs logiques traversés pour le calcul des sorties de l'additionneur est indépendant de la taille des mots en entrée. Néanmoins, ce type d'additionneur est plus encombrant que l'additionneur à retenue propagée, la logique de génération des retenues étant plus complexe. Les constructeurs proposent des circuits connus sous le nom de CLU *(Carry Look-ahead Unit)* qui assurent le calcul des termes précédents (ex : référence *xx* 182).

Les solutions étudiées ne permettent pas de traiter le cas de l'addition de deux nombres de signes différents. Ce problème est lié à celui de la soustraction qui est traitée dans le paragraphe suivant.

4.3 LES MULTIPLIEURS ET DIVISEURS (OPERATEURS ARITHMETIQUES)

Dans les années 70-80, les opérations de multiplication et de division étaient effectuées de manière algorithmique dans les processeurs. Ces opérateurs sous maintenant intégrés sous forme matérielle pour des raisons de rapidité.

Le principe de la multiplication en base 2 a été présenté à la section précédente. Il existe deux types de solutions pour la réalisation des opérateurs correspondants.

- Solution combinatoire : réseau de portes ET et d'additionneurs binaires permettant de réaliser simultanément les opérations élémentaires de la multiplication. La figure 4.6 donne une réalisation possible d'un multiplieur combinatoire de deux nombres A et B de 4 bits. Cette structure transcrit directement sous forme de circuit le réalisation « à la main » du calcul.

- Solution séquentielle : cette solution sort du cadre de ce chapitre car non combinatoire. La multiplication est réalisée en plusieurs étapes faisant intervenir des opérations successives d'addition et de décalage (cf. chapitre 5) réutilisant les mêmes opérateurs. L'implantation de cette solution est beaucoup moins encombrante que la précédente, mais présente des performances de vitesse dégradées, car elle nécessite plusieurs étapes de calcul.

La réalisation de diviseurs est en fait plus délicate que celle des multiplieurs mais les divisions sont en pratique beaucoup moins courantes dans les systèmes numériques que les additions/soustractions et les multiplications. L'approche la plus simple, qui consiste à implanter la division sous forme d'additions/soustractions et de décalages, conduit à des performances médiocres en termes de vitesse de calcul et n'est guère utilisée.

Nous ne détaillerons pas davantage les familles de circuits arithmétiques. De nombreuses architectures existent : pipe-line, parallèle, série-parallèle, cellulaires, etc. Leur étude, qui sort du cadre de ce cours, est détaillée dans [Mul89].

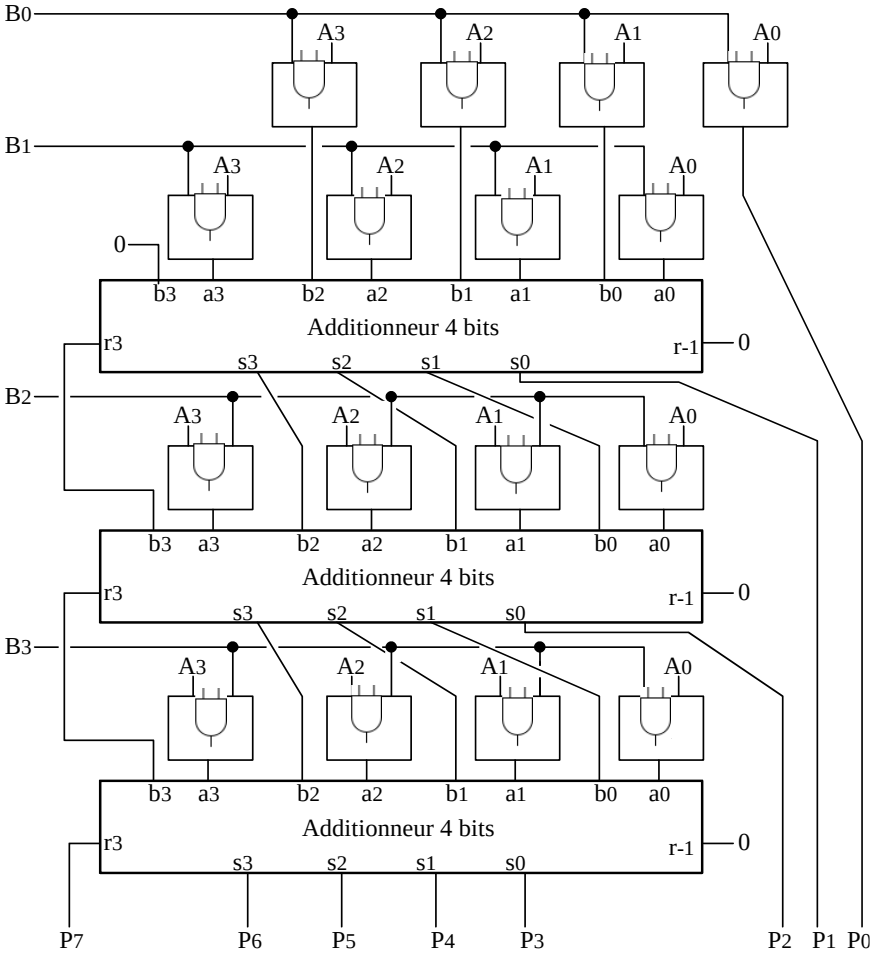
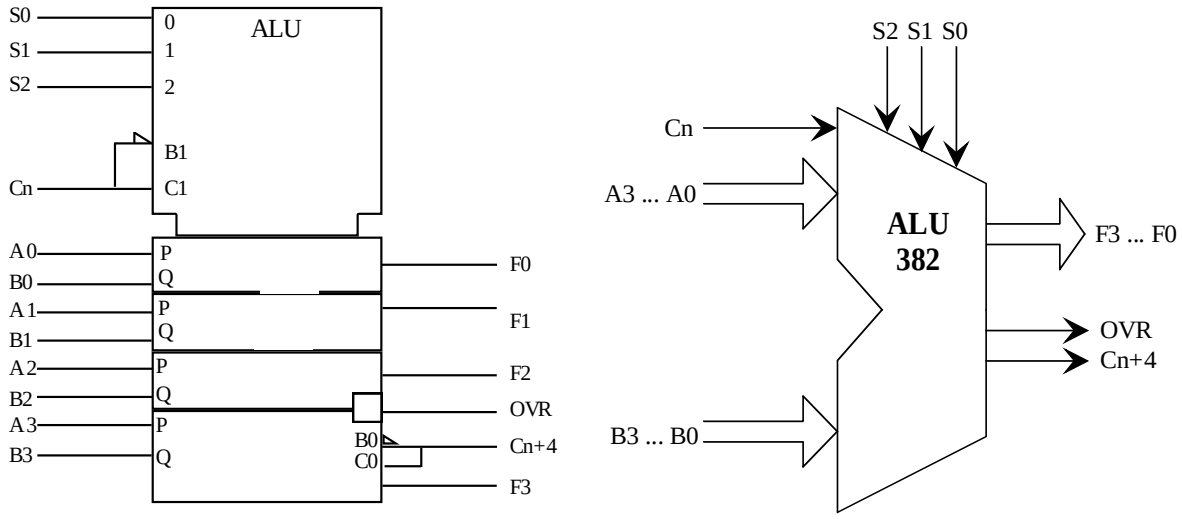


figure 4.6 : réalisation d'un multiplieur combinatoire de 2 mots de 4 bits

4.4 EXEMPLE D'UNE UNITE ARITHMETIQUE ET LOGIQUE DU MARCHE : LE CIRCUIT INTEGRE DE REFERENCE XX 382

L'UAL référencée *XX* 382 est un circuit qui est constitué d'environ 80 portes logiques élémentaires. A l'aide des entrées de sélection $S0$ à $S2$, l'utilisateur peut choisir une opération parmi les huit disponibles. Cette unité accepte en entrée des mots de 4 bits.



représentation normalisée IEEE

représentation usuelle

figure 4.7 : unité arithmétique et logique *XX* 382

Le tableau 4.4 présente une table de vérité partielle de l'UAL de référence *XX* 382. Seuls les cas où *A* et *B* sont égaux à 0000 ou 1111 sont traités à titre d'exemple.

Pour les modes arithmétiques, « *A* plus *B* », « *A* moins *B* », et « *B* moins *A* », les nombres *A* et *B* sont codés en complément à 2. Dans les trois cas, l'opérateur utilisé est un additionneur dont la retenue entrante est C_n.

- $S_2S_1S_0 = 011$, « *A* plus *B* » :

  Les nombres *A* et *B* sont directement appliqués en entrée de l'additionneur. Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est en fait $F = A + B + 1$, sinon $F = A + B$ (+ désigne ici l'addition).
- $S_2S_1S_0 = 001$, « *B* moins *A* » :

  Le complément à 1 de *A*, $\overline{A}$, est calculé et additionné à *B*. Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est $F = B + \overline{A} + 1 = B - A$, sinon $F = B + \overline{A} = B - A - 1$ (– désigne ici la soustraction).
- $S_2S_1S_0 = 010$, « *A* moins *B* » :

  Le complément à 1 de B, $\overline{B}$, est calculé et additionné à A. Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est $F = A + \overline{B} + 1 = A - B$, sinon $F = A + \overline{B} = A - B - 1$.

Dans ces trois modes arithmétiques, la sortie C_{n+4} est activée ($C_{n+4} = 1$) lorsque l'addition génère une retenue. La sortie *OVF* indique un dépassement de capacité de l'additionneur. Cette situation n'est pas considérée dans le tableau 4.4. A titre d'exemple, dans le mode « *A* plus *B* », la sortie *OVF* est activée si $A = B = 0111$.

Entrées de sélection $S2\,S1\,S0$	Opération arithmétique/logique	C_n [1]	$A =$ $A3\,A2\,A1\,A0$	$B =$ $B3\,B2\,B1\,B0$	$F =$ $F3\,F2\,F1\,F0$	OVF	C_{n+4}
000	Mise à 0 (*Clear*)	X	XXXX	XXXX	0000	—	—
001	*B* moins *A*	0	0000 0000 1111 1111	0000 1111 0000 1111	1111 1110 0000 1111	0 0 0 0	0 1 0 0
		1	0000 0000 1111 1111	0000 1111 0000 1111	0000 1111 0001 0000	0 0 0 0	1 1 0 1
010	*A* moins *B*	0	0000 0000 1111 1111	0000 1111 0000 1111	1111 0000 1110 1111	0 0 0 0	0 0 1 0
		1	0000 0000 1111 1111	0000 1111 0000 1111	0000 0001 1111 0000	0 0 0 0	1 0 1 1
011	*A* plus *B*	0	0000 0000 1111 1111	0000 1111 0000 1111	0000 1111 1111 1110	0 0 0 0	0 0 0 1
		1	0000 0000 1111 1111	0000 1111 0000 1111	0001 0000 0000 1111	0 0 0 0	0 1 1 1
100	$A \oplus B$	X	0000 0000 1111 1111	0000 1111 0000 1111	0000 1111 1111 0000	—	—
101	*A* OU *B*	X	0000 0000 1111 1111	0000 1111 0000 1111	0000 1111 1111 1111	—	—
110	*A* ET *B*	X	0000 0000	0000 1111	0000 0000	—	—

[1] C_n : entrée de retenue pour l'addition et entrée de retenue inversée pour la soustraction

			1111 1111	0000 1111	0000 1111		
111	Mise à 1 (*Preset*)	X	XXXX	XXXX	1111	—	—

tableau 4.4 : table de fonctionnement partielle de l'UAL XX 382

Cette UAL permet également d'effectuer 3 opérations logiques sur les mots *A* et *B* : OU exclusif, OU et ET. Ces trois opérations s'effectuent bit à bit. En outre, les sorties $F3F2F1F0$ peuvent être forcées à 0 ou à 1. Les sorties C_{n+4} et *OVF* ne sont, *a priori*, utilisées que dans les modes arithmétiques, leurs valeurs ne sont donc pas données dans les modes logiques car sans intérêt.