ÉTUDE D'UN ADDITIONNEUR COMPLET À PROPAGATION DE RETENUES

("Full ripple carry adder")

1. Spécification

a/ Spécification de l'interface

Un additionneur complet à N bits effectue la somme de deux opérandes A et B (mots binaires à N bits) et d'une entrée de retenue I (un seul bit) et fournit la somme tronquée à gauche S (mot binaire de N bits) et le bit de retenue C sortante.

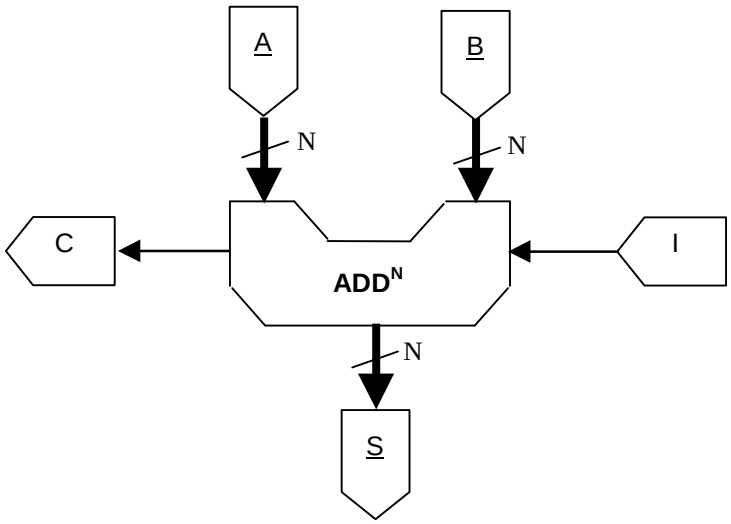


Figure 1: Interface de l'additionneur

b/ Spécification du comportement

Nous allons exprimer ici les sorties au moyen d'une expression algébrique des entrées.
Le mot obtenu en concaténant C et S est bien la somme de A, B et I.
En appelant ici # l'opérateur d'addition (pour éviter de la confondre avec l'opérateur OU +) et : l'opérateur de concaténation :

$$C : S = A \# B \# I$$

2. Analyse

a/ analyse préliminaire du traitement à effectuer

Étudions ce que valent chacun des bits.

Pour ce faire, sur un exemple, effectuons l'addition à la main comme à l'école primaire, mais en binaire,
des opérandes de 4 bits 1100 et 1110 :

	1	*1*			
		1	1	0	0
#		1	1	1	0
	1	1	0	1	0
	C		S		

On découpe en tranches, et on fait le calcul tranche par tranche, de droite à gauche ;
Dans chaque tranche, pour calculer le bit résultat n°n S_n , on ajoute:

- le bit de retenue C_{n-1} venant de la tranche précédente n°n-1 (C comme "Carry", report, retenue);
- le bit n°n A_n du mot A ;
- le bit n°n B_n du mot B .

	4	*3*	*2*	*1*	*0*	←*rang du bit*
	1	*1*	*0*	*0*	*0*	←*retenues venant de la tranche précédente à ajouter*
		1	1	0	0	←mot *opérande* <u>*A*</u>
#		1	1	1	0	←mot *opérande* <u>*B*</u>
		1	0	1	0	←mot *résultat* <u>*S*</u>

Nous voyons déjà qu'il nous faut garder le bit de retenue C_{N-1} sortante à gauche que nous appellerons C;
le vrai résultat complet non tronqué de l'addition est : C **:** <u>S</u> = <u>A</u> # <u>B</u> # I où **:** est ici le symbole de concaténation.

Bien sûr, on a effectué ici <u>A</u> # <u>B</u> = <u>A</u> # <u>B</u> # 0.

Supposons que nous voulions effectuer <u>A</u> # <u>B</u> # 1
Il suffirait d'ajouter 1

	4	*3*	*2*	*1*	*0*	←*rang du bit*
	1	*1*	*0*	*0*	*0*	←*retenues venant de la tranche précédente à ajouter*
		1	1	0	0	←mot *opérande* <u>*A*</u>
#		1	1	1	0	←mot *opérande* <u>*B*</u>
					1	←*1 en plus*
		1	0	1	1	←mot *résultat* <u>*S*</u>

Mais c'est par pure convention que nous avions placé 0 pour la retenue de droite (il n'y avait pas de retenue venant de droite).
Nous pouvons tout simplement mettre ce 1 dans cette case et l'appeler "retenue d'entrée I":

	4	*3*	*2*	*1*	*0*	←*rang du bit*
	1	*1*	*0*	*0*	*1*	←*retenues venant de la tranche précédente à ajouter*
		1	1	0	0	←mot *opérande* <u>*A*</u>
#		1	1	1	0	←mot *opérande* <u>*B*</u>
		1	0	1	1	←mot *résultat* <u>*S*</u>

De manière générale, et en appelant la retenue d'entrée I :

	4	*3*	*2*	*1*	*0*	←*rang du bit*
	C_3	C_2	C_1	C_0	I	←*retenues venant de la tranche précédente à ajouter*
		A_3	A_2	A_1	A_0	←mot *opérande* <u>*A*</u>
#		B_3	B_2	B_1	B_0	←mot *opérande* <u>*B*</u>
		S_3	S_2	S_1	S_0	←*résultat* <u>*S*</u>

b/ Décomposition en tranches de un bit

Nous allons donc affecter un opérateur (une tranche de un bit) pour chaque bit de sortie S_n.
Nous voyons que nous effectuons le même calcul sur chaque colonne. En extrayant la tranche n°n:

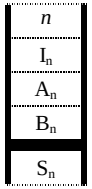
		n	←*rang du bit*
retenue vers tranche suivante →	C_n	C_{n-1}	←*retenue venant de la tranche précédente à ajouter*
		A_n	←*bit opérande An*
		B_n	←*bit opérande Bn*
		S_n	←*bit résultat* S_n

Appelons I_n la retenue à ajouter à la tranche n°n, et C_n la retenue sortant de la tranche n°n.

On a la propagation des retenues :

$I_n = C_{n-1}$ pour $n \in [1 .. N-1]$
$I_0 = I$
$C = C_{N-1}$

retenue vers tranche suivante →



←*rang du bit*
←*retenue I_n venant de la tranche précédente à ajouter*
←*bit opérande A_n*
←*bit opérande B_n*
←*bit résultat S_n*

Puisque le traitement est identique dans toutes les tranches, on peut donc remplacer chaque tranche n°n par une copie (instance) ADD^1_n d'une tranche modèle ALU^1 avec des entrées de un seul bit $A=A_n$, $B=B_n$; la sortie somme S_n est la sortie S de la tranche n°n ADD^1_n .
L'entrée de retenue I de la tranche n°n est $I_n = C_{n-1}$ = sortie de retenue C de la tranche n°n-1;
la retenue sortante C de l'additionneur complet ADD^N est la sortie de retenue C de la dernière tranche à gauche ADD^1_{N-1} :

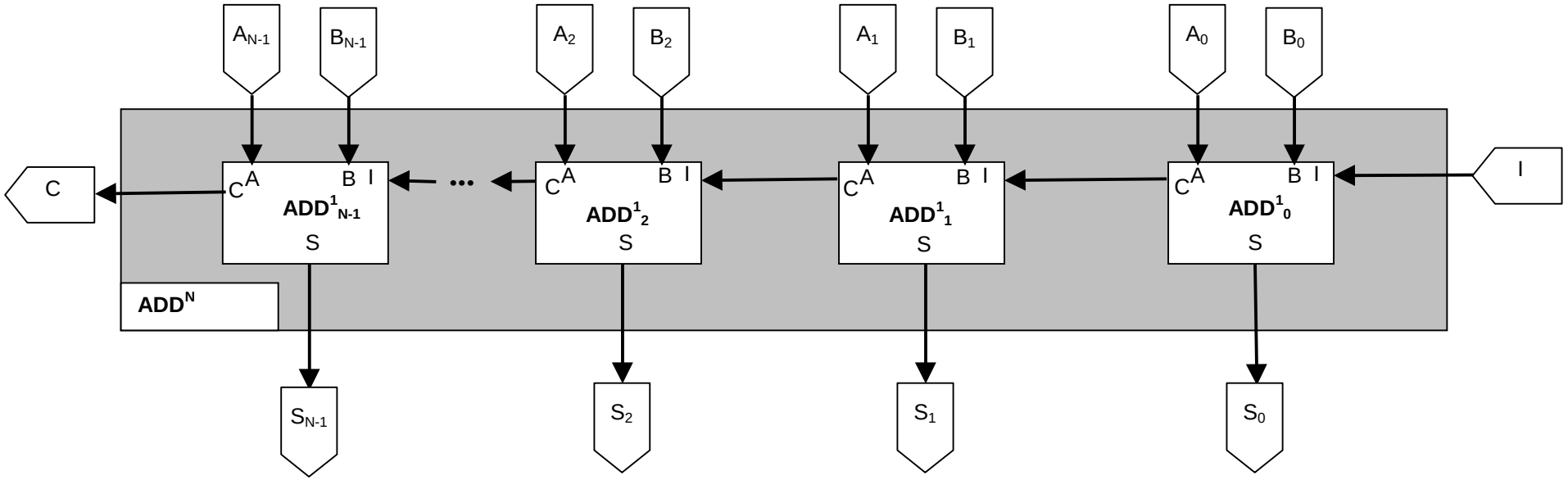
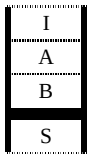


Figure 2: Décomposition de l'additionneur en tranches de 1 bit

retenue de sortie →



←*retenue d'entrée I*
←*bit opérande A*
←*bit opérande B*
←*bit résultat S*

Nous allons maintenant spécifier la tranche modèle.
Son interface est :

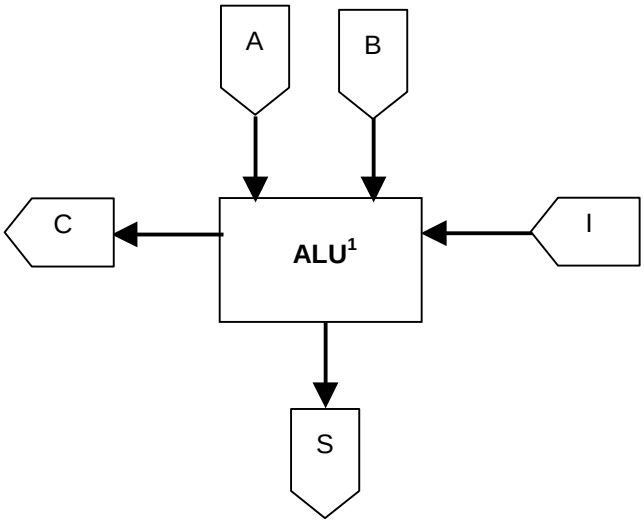


Figure 3: Interface d'une tranche modèle: additionneur à 1 bit

Étudions maintenant son comportement. Cette tranche est un additionneur à un seul bit qui effectue la somme non tronquée:

C **:** S = A # B # I

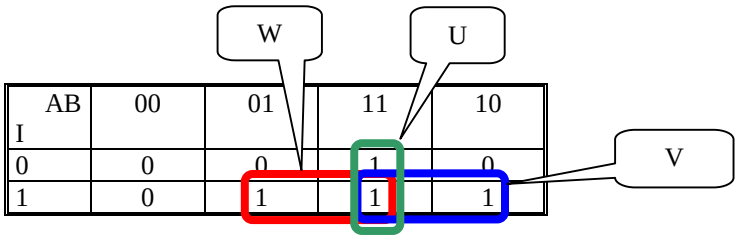
Puisqu'il y a peu de bits d'entrée et de sortie, on peut aussi représenter le comportement par une table de vérité pour C et S (ici juxtaposées avec une seule colonne par sortie) est donc la table d'addition :

ABI	C	S	
000	0	0	
001	0	1	
010	0	1	
011	1	0	$\leftarrow CS = 0\ \#\ 1\#\ 1 = (2)_{10} = 10 \Rightarrow C = 1$ *et* $S = 0$
100	0	1	
101	1	0	
110	1	0	
111	1	1	$\leftarrow CS = 1\ \#\ 1\#\ 1 = (3)_{10} = 11 \Rightarrow C = 1$ *et* $S = 1$

c/ Polynôme booléen de chaque sortie

Retenue de sortie

On va donc l'exprimer directement sous forme d'un polynôme booléen réduit, en dressant sa table de Karnaugh:



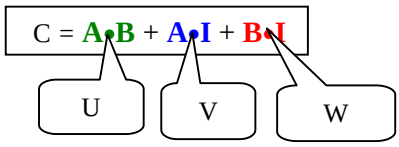
AB / I	00	01	11	10
0	0	0	1	0
1	0	1	1	1

On recouvre tous les 1 et aucun 0 avec des rectangles de largeur et hauteur puissance de 2,
aussi grands et peu nombreux que possible.

On considère que la table est en fait :
- enroulée autour d'un axe perpendiculaire à un côté d'au moins 4 cases (par exemple 00 est voisin de 10)
  (c'est le cas ici, elle est donc enroulée verticalement, le bord gauche coïncide avec le bord droit);
- repliée autour d'une médiane perpendiculaire à un côté d'au moins 8 cases
  (sur une table à 8 colonnes ou lignes, 011 est voisin de 111 et 001 de 101 car ils ne diffèrent que d'un seul bit).

On obtient 3 rectangles représentant chacun un monôme (un produit).

Sur le rectangle U, seules les coordonnées de A et B sont constantes. Le produit U ne contient donc que A et B. De plus, AB=11 sur ce rectangle.
Donc A et B sont respectivement représentés par A et B dans le produit qui vaut donc : U = A • B.
(Si l'on avait AB=01 sur le rectangle par exemple, on aurait : U = /A • B). C est le OU de tous les produits :

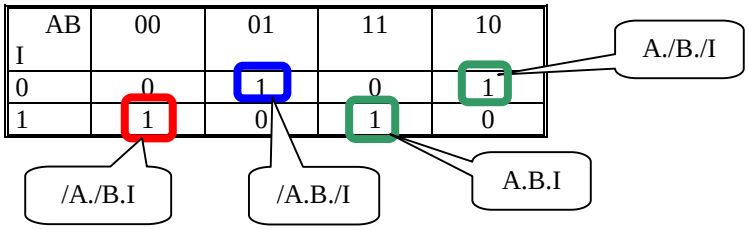


Note : Fonction phi
L'opérateur donnant C à partir de A, B et I n'est pas associatif mais est commutatif; il donne 1 quand au moins 2 bits sont à 1; c'est la fonction booléenne standard φ(A, B, I) *(cf. maths discrètes)*

C = φ(A, B, I)

Bit de somme

La table de Karnaugh correspondant à la table de vérité de S (A, B, I) est :



Le polynôme booléen réduit du bit somme S est donc :

S = A • B • I + /A • B • I + /A • /B • I + /A • B • /I + A • / B • /I

Note: À quelle opération o connue sur A, B et I correspond S ?

Méthode 1: D'après la table de vérité de S :

S = 0 quand on ajoute un nombre pair (0 ou 2) de bits à 1
S = 1 quand on ajoute un nombre impair (1 ou 3) de bits à 1

Ceci est la propriété essentielle du XOR, évidente pour 2 bits, et facile à étendre par récurrence à un nombre quelconque N de bits (cf. annexe). On en déduit:

S = A ⊕ B ⊕ I *ceci n'est pas un polynôme booléen !*

Méthode 2: Cette opération o est associative, puisque si l'on effectue d'abord (A o B) puis on ajoute I on a le même résultat que si on effectue d'abord B o I puis on l'ajoute à A :

(A o B) o I = A o (B o I)

Elle est aussi commutative: A o B = B o A

Considérons donc l'opération sur deux bits seulement A et B ; la table de vérité de A o B est :

A B	0	1
0	0	1
1	1	0

On reconnaît la table de vérité du OU exclusif ("eXclusive OR" ou XOR) qui est effectivement associatif et commutatif. Par conséquent l'opérateur o recherché est le XOR noté ⊕. Donc l'expression algébrique de S est A ⊕ B ⊕ I .

Méthode 3: Pour 3 bits, on peut aussi directement vérifier la formule ci-dessus a posteriori ; en effet, développons S en utilisant l'expression du XOR A ⊕ B = /A • B + A • / B

$A \oplus B \oplus I = (A \oplus B) \oplus I = (/A \bullet B + A \bullet /B) \oplus I = /(/A \bullet B + A \bullet /B) \bullet I + (/A \bullet B + A \bullet /B) \bullet /I$

On a, utilisant les lois de Morgan puis en développant :

$/(/A \bullet B + A \bullet /B) = /(/A \bullet B) \bullet /(A \bullet /B) = (A + /B) \bullet (/A + B) = A \bullet /A + A \bullet B + /B \bullet /A + /B \bullet B$

Or il est facile de vérifier que (il y a deux cas: A=1 et A=0) : $/A \bullet A = 0$. De même : $/B \bullet B = 0$

Il s'ensuit que : $/(/A \bullet B + A \bullet /B) = A \bullet B + /A \bullet /B$

D'où, en reportant puis en développant :

$A \oplus B \oplus I = (A \bullet B + /A \bullet /B) \bullet I + (/A \bullet B + A \bullet /B) \bullet /I = A \bullet B \bullet I + /A \bullet B \bullet I + /A \bullet /B \bullet I + /A \bullet B \bullet /I + A \bullet /B \bullet /I$

Ceci correspond bien au polynôme booléen réduit déterminé ci-dessus.

Note: Table de vérité étendue

On remarque que le comportement d'une tranche d'additionneur peut s'exprimer au moyen de la table de vérité étendue:

C	S
$\varphi(A, B, I)$	$A \oplus B \oplus I$

Annexe: Démonstration de la propriété du XOR

Soit N bits $\underline{x}_N = (x_1, x_2, ..., x_N) \in B^N$ Appelons $Q_N(\underline{x})$ = nombre de ces bits à 1

Nous allons montrer par récurrence que: $\forall (x_1, x_2, ..., x_N) \in B^N$, $S_N = x_1 \oplus x_2 \oplus ... \oplus x_N = 1 \Leftrightarrow Q_N(\underline{x}_N)$ impair

La propriété est vraie pour N = 2: on peut le vérifier sur chaque cas dans le tableau ci-dessous :

x_0 x_1	0		1	
0	Q_2 pair	0	Q_2 impair	1
1	Q_2 impair	1	Q_2 pair	0

Supposons que la propriété soit vraie pour N bits.

Pour N+1 bits, $\underline{x}_{N+1} = (x_1, x_2, ..., x_N , x_{N+1})$ et $S_{N+1} = x_1 \oplus x_2 \oplus ... \oplus x_N \oplus x_{N+1} = (x_1 \oplus x_2 \oplus ... \oplus x_N) \oplus x_{N+1} = S_N \oplus x_{N+1}$

La propriété étant vraie pour N bits, $S_N = 1 \Leftrightarrow Q_N(\underline{x}_N)$ impair
Donc pour Q_N pair, $S_N=0$ et pour Q_N impair, $S_N=1$.
Par ailleurs, si Q_N est pair et $x_{N+1} = 0$, alors Q_{N+1} ne change pas et est donc toujours pair.
En même temps $S_{N+1} = S_N \oplus x_{N+1} = 0 \oplus 0 = 0$.

On peut résumer les 4 cas en donnant la parité de Q_{N+1} et la valeur de S_{N+1} pour Q_N pair ou impair et $x_{N+1} = 0$ ou 1 dans le tableau suivant:

Q_N	S_N x_{N+1}	0		1	
pair	0	Q_{N+1} pair	0	Q_{N+1} impair	1
impair	1	Q_{N+1} impair	1	Q_{N+1} pair	0

On note que $S_{N+1} = 1 \Leftrightarrow Q_{N+1}$ impair ; la propriété est donc vraie pour N+1 bits

Elle est donc vraie pour tout $N \geq 2$.