

## TD 1 - CIRCUITS COMBINATOIRES

Simplification de fonctions booléennes, familiarisation avec l’outil de conception de circuits, circuits combinatoires, représentation des nombres, arithmétique en complément à 2, arithmétique flottante.

Page web : <http://www-rocq.inria.fr/~acohen/teach/archi.html>

### Exercice 1.1 - Afficheur numérique

On souhaite réaliser l’afficheur numérique décimal décrit sur la figure 1. Il est constitué de 7 diodes électro-luminescentes (LEDs) nommées *N*, *NW*, *NE*, *C*, *SW*, *SE* et *S*.

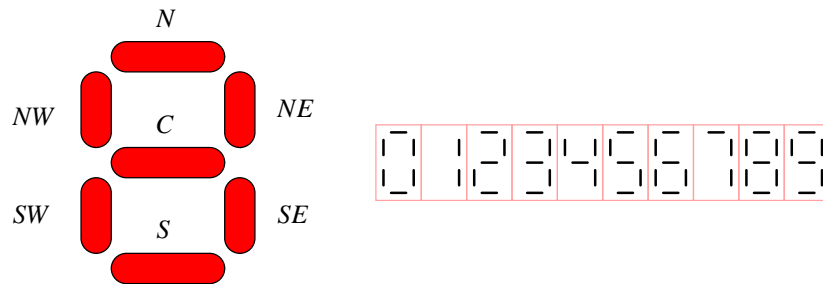


FIG. 1 – Afficheur numérique décimal

#### Question 1.1.1

Écrire une table de vérité pour chaque LED de l’afficheur, puis simplifier les fonctions algébriques correspondantes à l’aide des tableaux de Karnaugh. Pour ne pas perdre de temps, vous avez la possibilité de ne réaliser que deux tableaux, par exemple ceux du sud ouest (*SW*) et du centre (*C*).

#### Réponse

Le signal d’entrée est codé sur 4 bits *dcb*. Les tables suivantes représentent les valeurs de vérité de chaque LED : *ba* horizontalement et *dc* verticalement.

<i>N</i>	00	01	11	10	<i>NW</i>	00	01	11	10	<i>NE</i>	00	01	11	10	<i>C</i>	00	01	11	10
00	1		1	1	00	1				00	1	1	1	1	00			1	1
01		1	1	1	01	1	1		1	01	1		1		01	1	1		1
11	x	x	x	x	11	x	x	x	x	11	x	x	x	x	11	x	x	x	x
10	1	1	x	x	10	1	1	x	x	10	1	1	x	x	10	1	1	x	x

<i>SW</i>	00	01	11	10	<i>SE</i>	00	01	11	10	<i>S</i>	00	01	11	10
00	1			1	00	1	1	1		00	1		1	1
01				1	01	1	1	1	1	01		1		1
11	x	x	x	x	11	x	x	x	x	11	x	x	x	x
10	1		x	x	10	1	1	x	x	10	1	1	x	x

Fonctions simplifiées :

$$\begin{aligned}
 N &= \bar{a}\bar{c} + ac + b + d, \\
 NW &= \bar{a}\bar{b} + \bar{a}c + \bar{b}c + d, \\
 NE &= \bar{a}\bar{b} + ab + \bar{c}, \\
 C &= b\bar{c} + \bar{a}b + \bar{b}c + d, \\
 SW &= \bar{a}\bar{c} + \bar{a}b, \\
 SE &= a + \bar{b} + c, \\
 S &= \bar{a}\bar{b}c + b\bar{c} + \bar{a}\bar{c} + \bar{a}b + d.
 \end{aligned}$$

#### Question 1.1.2

Réaliser l’afficheur en *DigLog*. Pour le circuit, on respectera la représentation des fonctions logiques sous forme de *sommes de produits*.

Support : commencer par récupérer le fichier `numeric_mask.lgf`, puis lancer *DigLog* par la commande `cohen/bin/diglog numeric_mask.lgf`.

#### Réponse

Voir la figure 2 et le fichier `numeric.lgf`.

### Exercice 1.2 - Deux circuits combinatoires importants

On étudie deux circuits élémentaires fréquemment utilisés dans les composants des processeurs.

#### Question 1.2.1

Un *décodeur*  $n \rightarrow 2^n$  est un circuit combinatoire comprenant une entrée *I* sur *n* bits et  $2^n$  sorties  $O_0, \dots, O_{2^n-1}$  sur 1 bit ; seul le signal  $O_m$  doit être activé lorsque l’entrée *I* vaut *m* (en binaire sur *n* bits).

Construire un décodeur  $3 \rightarrow 8$ . On pourra construire une table de Karnaugh ou chercher un schéma général pour un décodeur  $n \rightarrow 2^n$ .

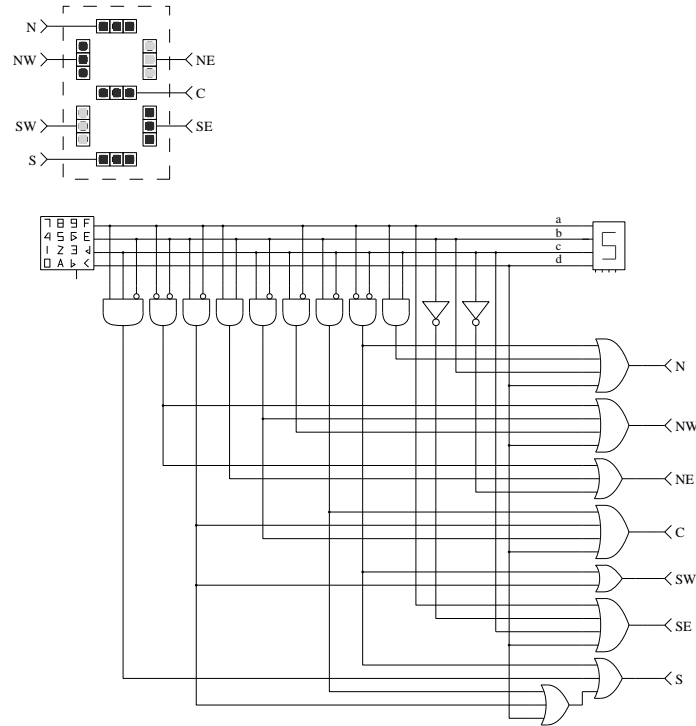


FIG. 2 – Circuit de l’afficheur numérique

**Réponse**

Il suffit de sélectionner une unique combinaison des signaux d’entrée à l’aide de 8 portes ET (à trois entrées). Chaque valeur d’entrée active ainsi un unique signal de sortie du décodeur. Voir la figure 3 et le fichier `mux_dec.lgf`.

**Question 1.2.2**

Un *multiplexeur*  $2^n \times 1$  est un circuit combinatoire comprenant une entrée  $A$  sur  $n$  bits,  $2^n$  entrées  $I_0, \dots, I_{2^n-1}$  sur 1 bit, et une sortie  $O$  sur 1 bit ; le signal  $O$  est égal à l’entrée  $I_m$  lorsque  $A$  vaut  $m$  (en binaire sur  $n$  bits).

Construire un multiplexeur  $8 \times 1$ . On pourra construire une table de Karnaugh, réutiliser le décodeur de la question précédent ou chercher un schéma général.

**Réponse**

En suivant le même schéma que le décodeur, on ajoute une quatrième entrée à chacune des 8

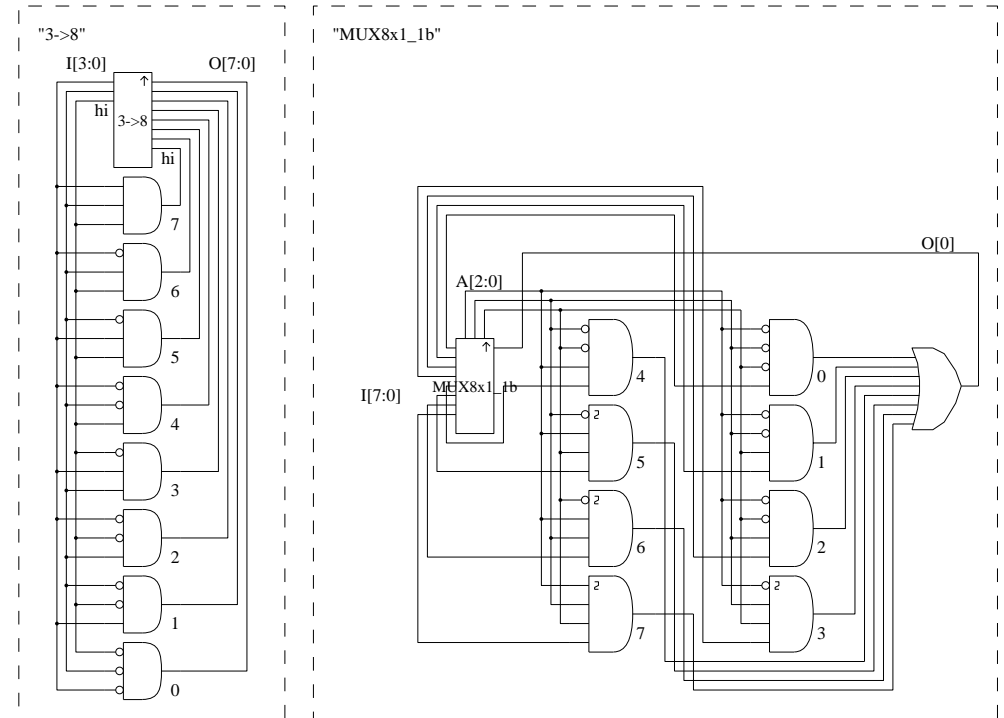


FIG. 3 – Circuits du décodeur et du multiplexeur

portes ET, recevant le signal  $I_k$  correspondant. La porte ET sélectionnée lorsque l’entrée  $A$  vaut  $k$  (sur 3 bits) fournit alors la valeur du signal  $I_k$ , les sorties des autres portes ET sont nulles. Il suffit donc d’une porte OU à 8 entrées pour produire le signal de sortie du multiplexeur. Voir la figure 3 et le fichier `mux_dec.lgf`.

**Exercice 1.3 - Unité arithmétique et logique**

Support : `alu_mask.lgf` propose une Unité Arithmétique et Logique (ALU) 1-bit (comprenant notamment un additionneur 1-bit et un multiplexeur  $4 \times 1$ ) et un masque pour une ALU 16-bits.

L’ALU 1-bit vue en cours propose les opérateurs d’addition, de conjonction et de négation ; voir le fichier `alu_mask.lgf`. On utilise cette « brique de base » pour

construire une unité arithmétique et logique 16-bits. On a transformé l'additionneur en additionneur/soustracteur en lui ajoutant un signal de contrôle  $ALLK_{SUB}$  analogue au signal  $c$  vu en cours, et en insérant des XOR pour inverser la deuxième opération de l'addition.

**Question 1.3.1**

La plupart des unités arithmétiques et logiques génèrent des signaux de condition; il s'agit de signaux de sortie utilisés par les instructions de branchement conditionnel. Réaliser une ALU 16-bits avec les trois signaux de condition suivants :

- $N$  vaut 1 lorsque la sortie (16-bits) est strictement négative ;
- $Z$  vaut 1 lorsque la sortie est nulle ;
- $P$  vaut 1 lorsque la sortie est strictement positive.

**Réponse**

Le signal  $Z$  est obtenu en calculant le NOR des 16-bits de sortie. Le signal  $N$  est égal au 15<sup>e</sup> bit de sortie de l'ALU (bit de poids fort). Le signal  $P$  est facilement obtenu à partir du NOR de  $Z$  et  $N$ , puisqu'il est à 1 lorsque la sortie est ni nulle ni négative. Voir le fichier a1u.1gf.

**Question 1.3.2**

Montrer que l'on peut effectuer une soustraction non signée à l'aide d'un opérateur de soustraction en complétant à 2.

**Réponse**

Pour  $x, y \in \{0, \dots, 2^{16} - 1\}$ , effectuer la soustraction au sens du complément à 2 est équivalent à :

$$(1) \quad x - y = x + \overline{2^{16} - y} = x + \overline{2^{16} - y} + 1 = x + \overline{2^{16} - y} + 1 \pmod{2^{16}}$$

Comme l'addition  $+2^{16}$  n'influe pas sur les 16 bits de poids faible du résultat, la soustraction en complément à 2 et de la soustraction sur les entiers naturels conduisent au même résultat. En revanche, cette addition supplémentaire va modifier la valeur de la retenue :  $2^{16}$  correspond au bit de la retenue, lui ajouter l revient à l'inverser.

**Question 1.3.3**

On suppose que l'ALU dispose d'un signal  $SIGNED$  qui vaut 1 lorsque les calculs ont lieu en complément à 2 (calculs signés) et 0 sinon (calculs non signés). Définir un signal  $V$  de dépassement de capacité (overflow) pour cette ALU.

**Réponse**

On appelle  $V$  le signal de dépassement de capacité. Supposons que  $SIGNED = 0$ . Lorsque l'on effectue une addition, il y a dépassement (i.e., un résultat supérieur ou égal à  $2^{16}$ ) si la retenue de sortie vaut 1. Inversement, lorsque l'on effectue une soustraction, on a vu à la question précédente que le dépassement (i.e., un résultat négatif) correspond à une retenue de sortie à 0. Supposons désormais que  $SIGNED = 1$ . Pour une addition il y a dépassement lorsque les deux entrées ont le même signe et que le résultat a un signe différent; pour une soustraction il y

a dépassement lorsque les deux entrées sont de signe différent et que le résultat n'a pas le signe de  $ALLSRC1$ . Le signal  $V$  de dépassement (overflow) est donc défini de la manière suivante :

$$V = \overline{Cout} \overline{SIGNED} ALK_{SUB} + \overline{Cout} \overline{SIGNED} ALK_{SUB} + \overline{ALLSRC1} \overline{ALUSRC2} \overline{ALURES} \overline{SIGNED} ALK_{SUB} + \overline{ALLSRC1} \overline{ALUSRC2} \overline{ALURES} \overline{SIGNED} ALK_{SUB} + \overline{ALLSRC1} \overline{ALUSRC2} \overline{ALURES} \overline{SIGNED} ALK_{SUB} + \overline{ALLSRC1} \overline{ALUSRC2} \overline{ALURES} \overline{SIGNED} ALK_{SUB}$$

En réalité, le signal de dépassement ne doit être activé que dans le cas d'une addition ou d'une soustraction; on masque donc le résultat précédent par le NAND des signaux de contrôle de l'ALU,  $ALK_0$  et  $ALK_1$ . Voir la figure 4 et le fichier a1u.1gf.

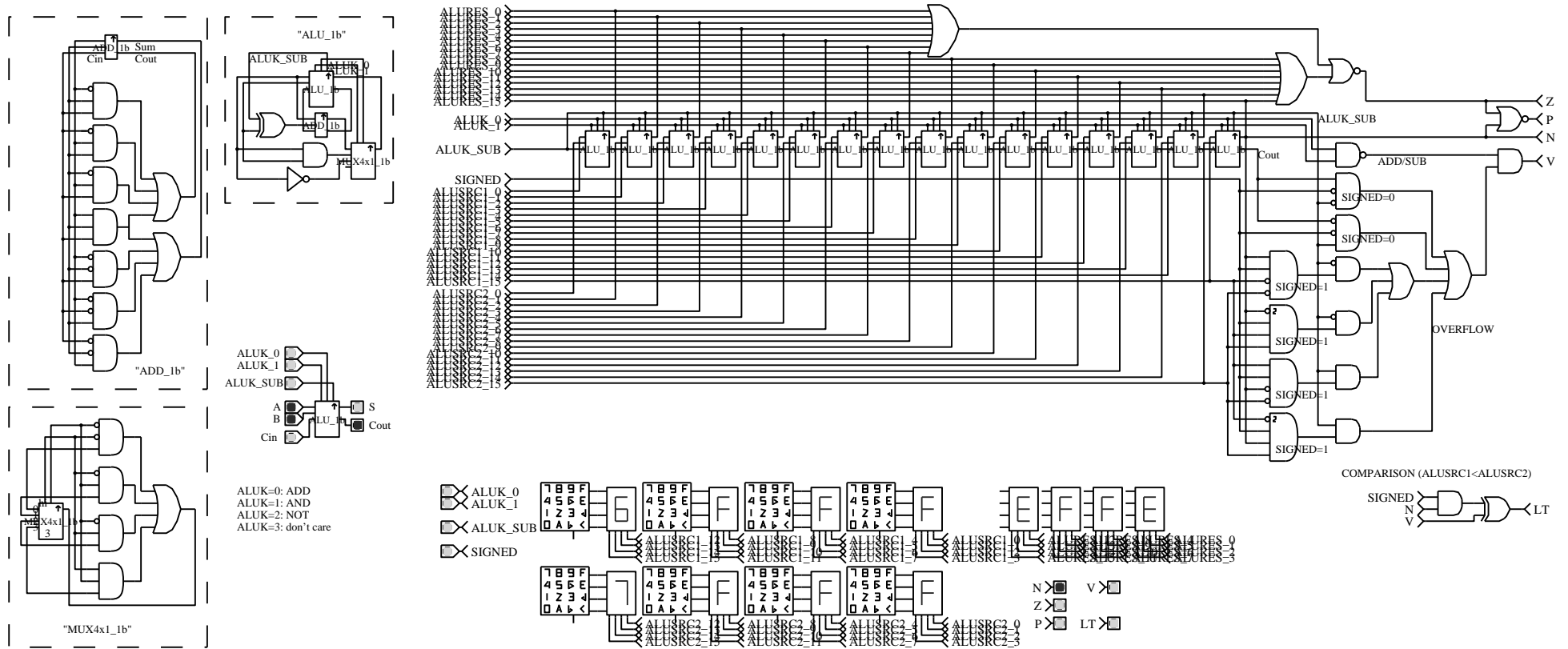


FIG. 4 – Circuit de l'ALU 16-bits

**Question 1.3.4**

En utilisant l'ALU ci-dessus et en minimisant le nombre de modifications, générer un signal de comparaison : soient  $a$  et  $b$  les deux entrées de l'ALU, le résultat de la comparaison est 1 si  $a < b$ , et 0 sinon ; le résultat de la comparaison doit être fourni sur un signal de sortie supplémentaire.

**Réponse**

On effectue la soustraction de  $a$  et  $b$  en choisissant

$$ALUK_1 ALUK_0 = 00 \text{ et } ALUK_{SUB} = 1.$$

Le signal  $LT$  (less than) est mis à 1, soit lorsque la différence est négative, soit lorsqu'elle est positive et qu'il y a dépassement ; voir la figure 4 et le fichier `alu.lgf`.

**Exercice 1.4 (facultatif) - Accélération de l'addition**

Cet exercice compare plusieurs compromis temps-espace pour des circuits d'accélération de l'addition. Il n'est pas nécessaire d'utiliser *DigLog* dans cet exercice.

On souhaite effectuer l'addition de  $n$  nombres  $a_1, \dots, a_n$  sur 4 bits. Pour cela, on peut utiliser une ALU pour calculer  $a_1 + a_2$  puis ajouter le résultat à  $a_3$ , etc.

**Question 1.4.1**

Pour la somme de 3 entiers sur 4 bits, proposer un circuit dont le temps de traversée vaut 6 fois celui de l'additionneur 1-bit.

**Réponse**

On « aligne » un additionneur 4 bits calculant la somme partielle des deux premières opérandes avec un additionneur 5 bits dont les opérandes sont le résultat du premier additionneur et la troisième opérande (le résultat comporte 5 bits). Le circuit résultant est décrit sur la figure 5.

**Question 1.4.2**

Pour la somme de 3 entiers sur 4 bits, proposer un circuit dont le temps de traversée ne vaut que 5 fois celui de l'additionneur 1-bit, contre 6 pour le circuit de la question précédente. Indication : utiliser l'associativité de l'addition pour retarder la propagation des retenues jusqu'à la dernière étape du calcul ; le circuit obtenu est appelé additionneur *carry-save*.

Proposer une généralisation de ce circuit pour la somme de  $n$  opérandes et une estimation de son temps de traversée.

**Réponse**

IL FAUT SUGGÉRER AUX ÉTUDIANTS DE CONCEVOIR UN ADDITIONNEUR POUR 3 NOMBRES À PARTIR D'ADDITIONNEURS 1-BIT, SANS SE PRÉOCCUPER DES PROBLÈMES DE TEMPS, PUIS D'ÉVALUER, SUR CET ADDITIONNEUR, AU BOUT DE COMBIEN DE TEMPS LE RÉSULTAT EST

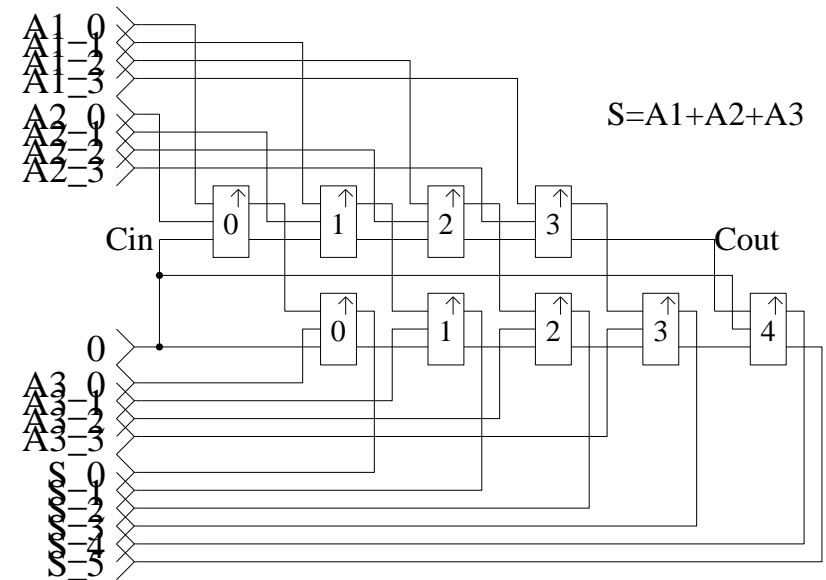


FIG. 5 – Circuit de l'additionneur à trois opérandes

VALIDE ; CELA LES MÈNE NATURELLEMENT À LA SOLUTION (NOTE : UNE UNITÉ DE TEMPS = PASSAGE À TRAVERS UN ADDITIONNEUR DANS CET EXERCICE).

On sépare le calcul des sommes partielles sans retenue et des retenues pour éviter de propager ces dernières. Dans un deuxième temps, les sommes et les retenues sont ajoutées de manière classique pour retrouver la somme totale.

Le cas de l'addition de trois opérandes est décrit sur la figure 6. Par rapport à la solution de la question précédente, on remarque qu'un additionneur 1-bit a pu être supprimé alors même que le temps de traversée global a été diminué.

Le cas général consiste à grouper les opérandes par 3 et à calculer les sommes partielles sans retenue et les retenues en une seule traversée parallèle, puis à recommencer sur les 2/3 d'opérandes résultant (en tenant compte des alignements respectifs des sommes partielles et des retenues), jusqu'à n'obtenir que deux opérandes que l'on somme de manière classique.

Le nombre  $k$  d'étages d'additionneurs carry-save 1-bit vérifie  $n(2/3)^k \leq 2$ , i.e.,  $k \leq (\lg n -$

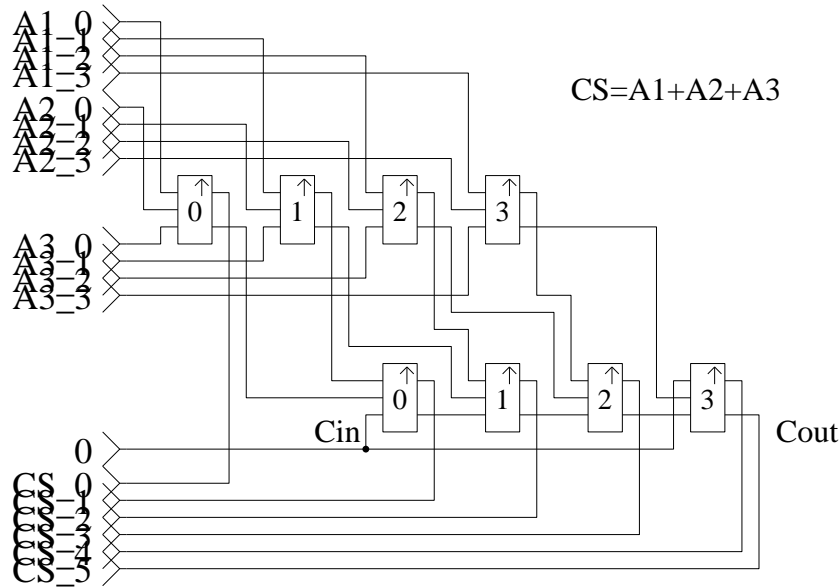


FIG. 6 – Circuit de l'additionneur *carry-save* à trois opérandes

$1)/\lg 3$ . D'autre part, la somme finale séquentielle porte sur  $\lceil n/2 \rceil + 4$  bits. Le temps de traversée global est de l'ordre de  $n/2 + \lg n/\lg 3$ .

Cette structure réursive fondée sur l'additionneur *carry-save* est appelée *arbre de Wallace*. Ces arbres sont utilisés pour accélérer les multiplieurs, pour compter le nombre de bits à 1 dans un registre, etc. Voir Patterson et Hennessy « *Computer Organization and Design* » pp. 331 et 332.

**Exercice 1.5 (facultatif) - Multiplieur flottant**

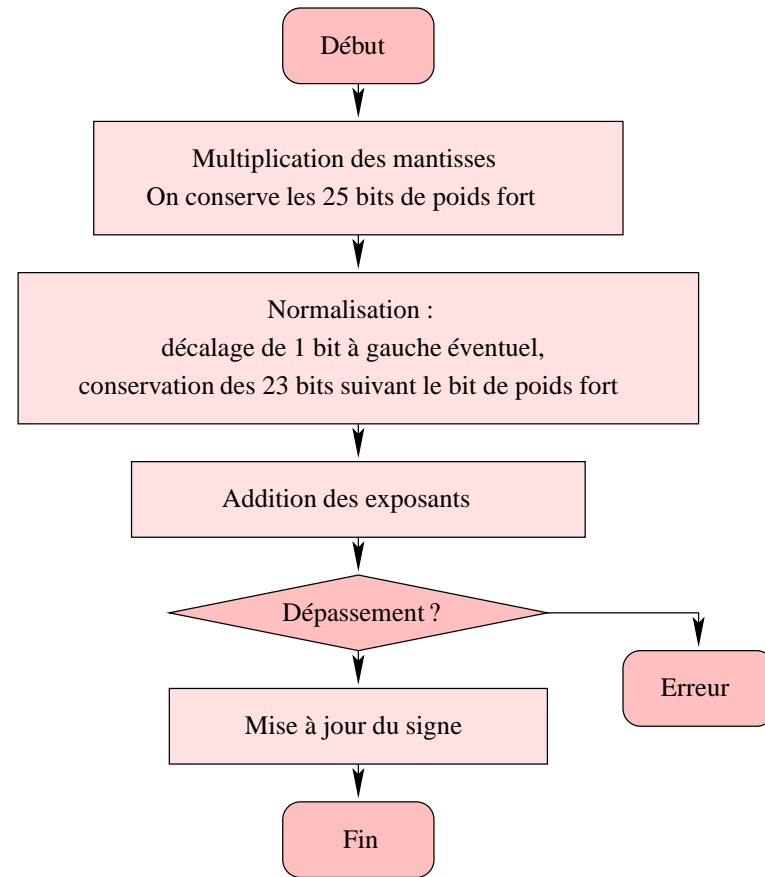
On étudie la structure d'un multiplieur pour des nombres à virgule flottante. On considère une représentation simplifiée, inspirée de la norme IEEE 754 pour les flottants 32 bits :

- on ne considérera que des nombres normalisés ;
- lorsqu'une valeur ne peut être représentée par un nombre normalisé, on déclenche un signal d'erreur *ERROR* ;
- le circuit ne pourra effectuer qu'un arrondi par élimination des bits de poids faible non représentables.

**Question 1.5.1**

Déterminer les différentes étapes nécessaires à l'opération de multiplication flottante

Réponse



**Question 1.5.2**

Identifier tous les composants nécessaires à la réalisation du multiplicateur (hors contrôle). Indiquer les jonctions nécessaires entre les différents composants et leur taille.

Réponse

D'une part un registre 48 bits alimenté par un multiplicateur  $96 \times 48$  pour choisir entre les données initiales — 0 sur les 24 bits de poids fort et l'entrée 1 du multiplicateur sur les 24 bits de poids faible — ou le décalage à gauche. D'autre part, un additonneur 24-bits alimenté par les 24 bits de poids fort du registre et par le produit entre le bit 23 du registre et l'entrée 2 du multiplicateur. La mantisse est obtenue par un décalage à gauche de 1-bit éventuel — lorsque le bit 47 est à 0 — à l'aide d'un multiplicateur  $48 \times 24$ .  
L'exposant est calculé à l'aide de 3 additions 8 bits enchainées — pour soustraire le biais de 127 puis ajouter les exposants et éventuellement de 1 si la mantisse n'a pas été décalée.

**Question 1.5.3**

Indiquer, pour chaque étape, les valeurs que doivent prendre les différents signaux d'entrée des composants.

Le circuit de contrôle du multiplicateur flottant sera réalisé dans le cadre du prochain

TD.

Réponse

Il y a dépassement de capacité —  $ERROR = 1$  — lorsque la somme modulo 2 des retenues des trois additonneurs du circuit de calcul de l'exposant vaut 0. En effet, les sommes des deux premières retenues correspondent à la négation du bit 8 de l'exposant (il manque le bit 8 de l'exposant de l'entrée 1), et la deuxième somme correspond à la négation du bit 8 de l'exposant final. On calcule donc  $ERROR$  en combinant les trois retenues de sorties avec une porte  $XNOR$  et une porte  $XOR$ .  
Pour le reste, voir la correction du multiplicateur entier et le circuit microprogrammé du multiplicateur flottant dans le TD suivant.