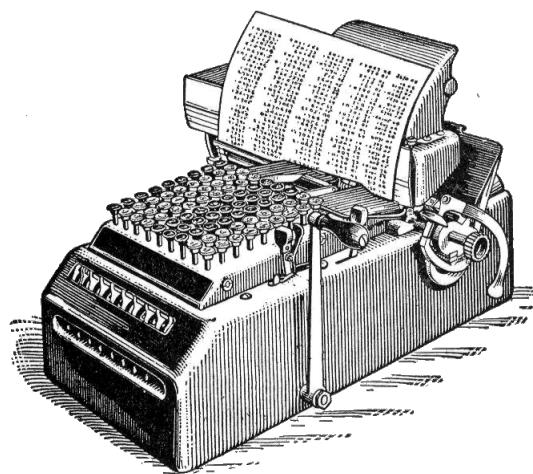


RAPPORT DE PROJET : CALCULATRI'ECE

Semestre 3

Projet en équipe d'électronique	
Projet	ING2_ELECPRJ2 : Calculatri'ece
Client	ECE Paris - Lyon - Département électronique
Document	Rapport de projet
Auteurs	Léopold ROMBAUT, Tristan QUERTON Victor RATTI, Matia CILLY
Type	Programmation sur FPGA en VHDL
Publication	03/10/2021
Version	1.0
Révision	0
Diffusion	Restreinte



Historique des révisions

Ci-après, le tableau des révisions, historique de toutes les modifications apportées au document.

Nom	Date	Modification	Version	Révision
Tous	09/09	Répartition des tâches	0	0
Rombaut	09/09	Création du rapport & mise en page du rapport	0	0
Cilly	15/09	Description des sigles et termes	0	0
Ratti	15/09	Création de la page de garde	0	0
Querton	17/09	Rédaction de l'introduction	0	1
Rombaut	18/09	Rédaction du protocole NEC	0	1
Ratti	18/09	Rédaction du module d'affichage	0	1
Rombaut	21/09	Rédaction du module de sonorisation	0	1
Querton	22/09	Rédaction du module de calcul	0	1
Ratti	22/09	Rédaction du module d'affichage	0	1
Rombaut	25/09	Rédaction du module d'infrarouge	0	1
Rombaut	25/09	Création de schémas pour le module infrarouge	0	1
Cilly	25/09	Modification de sources	0	1
Querton	25/09	Rédaction du module de calcul	0	1
Ratti	25/09	Relecture des modules	0	1
Querton	27/09	Création des diagrammes pour le module de calcul	0	1
Tous	28/09	Relecture et mise à jour des modules	0	2
Tous	28/09	Ajout de schémas descriptifs	0	2
Querton	29/09	Rédaction des références	0	2
Ratti	30/09	Ajout des annexes	0	2
Ratti	02/10	Rédaction de la conclusion	0	2
Cilly	1/10	Relecture des modules	0	2
Rombaut	1/10	Complément des annexes	0	2
Tous	3/10	Rendu du projet	1	0

Table 1 – Tableau de l'historique de révisions

Nous attestons que ce travail est original, qu'il est le fruit d'un travail commun au groupe et qu'il a été rédigé de manière autonome.

Paris, le 13/1/2023

ROMBAUT Leopold

QUERTON Tristan

CILLY Matia

RATTI Victor

Table des matières

I	Introduction	7
A	Avant-propos	7
B	Table des références	7
C	Présentation d'un FPGA	8
D	Sigles et termes	9
II	Module de réception	10
A	Fonctionnement de la télécommande	10
1	Le protocole NEC	10
B	Fonctionnement du code	11
C	Réception des opérandes signées	12
D	Montage	13
III	Module de calcul	14
A	Mode signé et non signé	14
1	Les nombres binaires non signés	14
2	Les nombres binaires signés	14
B	Méthodologie de calcul	15
C	Quelques mots sur l'additionneur externe CLA	16
D	Additions	17
1	Addition non signée interne	18
2	Addition signée interne	18
3	Addition non signée externe	21
4	Addition signée externe	22
E	Multiplications	24
1	Multiplication non signée interne	25
2	Multiplication signée interne	26
IV	Module de sonorisation	29
A	Fonctionnement du code	29
B	Montage du module	30
C	Résultat signé	31
V	Module d'affichage	32
A	Fonctionnement d'un afficheur 7-segment	32
B	Exemple d'affichage d'un chiffre	33
C	Affichage du résultat sur des LEDS externes	33
D	Affichage des modes sur des LEDS internes au FPGA	34
VI	Architecture du projet	35
A	Architecture logicielle	35
B	Architecture électronique	36

VII Partie bonus : une calculatrice 8 bits	37
A Un résultat sur 8 bits	37
B Division euclidienne	37
C Soustraction	37
VIII Organisation du travail de groupe	38
IX Conclusion	38
X Sources	38
XI Annexes	39
A Vues RTL	39
B Chronogrammes	41
C Images	43

Table des figures

1	FPGA DE10-Lite	8
2	Signal reçu d'une télécommande utilisant le protocole NEC	10
3	Photo de la télécommande	11
4	Diagramme de lecture de la commande infrarouge	12
5	Branchemet du capteur infrarouge	13
6	Photo et architecture logique du 74LS283	16
7	Addition de deux nombres binaires	17
8	Diagramme de l'addition de deux nombres binaire non signés	18
9	Diagramme de l'addition de deux nombres binaire signés	19
10	Cablage de l'additionneur 4 bits	21
11	Diagramme du fonctionnement de l'addition externe non signée	22
12	Diagramme du fonctionnement de l'addition externe signée	23
13	Exemple d'une multiplication binaire	24
14	Diagramme de la multiplication interne non signée	25
15	Diagramme de la multiplication interne signée	27
16	Diagramme du fonctionnement du buzzer	30
17	Branchemet du buzzer	30
18	Schéma de la sélection de la fréquence	31
19	Diagramme d'affichage	32
20	Schéma afficheur 7-segments	33
21	Résultat de l'affichage	33
22	Architecture logicielle du projet	35
23	schéma de branchement du projet	36
24	Vue RTL du module de réception	39
25	Vue RTL des opérations signées interne	39
26	Vue RTL des opérations non signées interne	39
27	Vue RTL des opérations signées externe	40
28	Vue RTL des opérations non signées externe	40
29	Vue RTL du module de sonorisation	40
30	Vue RTL du module d'affichage	41
31	Chronogramme de test de l'addition interne non signée	41
32	Chronogramme de test de l'addition interne non signée avec overflow	41
33	Chronogramme de test de l'addition interne non signée	42
34	Chronogramme de test de l'addition interne non signée avec overflow	42
35	Chronogramme de test de l'addition interne signée	42
36	Chronogramme de test de la multiplication interne signée	42
37	Addition interne signée	43
38	Multiplication interne signée	43
39	Addition externe signée	44
40	Overflow après une addition externe non signée	44

Liste des tableaux

1	Tableau de l'historique de révisions	2
2	Valeur hexadécimale de chaque touche de la télécommande	11
3	Table de vérité de la porte XOR	26
4	Fréquence de 50Mhz	29
5	Fréquence de 10Mhz	29

I Introduction

A Avant-propos

L'objectif de ce projet est de réaliser une calculatrice à partir d'une carte programmable de type FPGA. La calculatrice à réaliser est une version simplifiée des calculatrices disponibles sur le marché. Cette dernière doit pouvoir additionner ou multiplier les opérandes reçues. Ces opérandes sont saisies à l'aide d'une télécommande infrarouge. Enfin, le résultat de l'opération doit être présenté sur les afficheurs de la carte, bippé par une buzzer externe et affiché en binaires sur des LEDS externes.

La calculatrice doit aussi pouvoir effectuer les opérations à l'aide d'un composant externe, l'additionneur à retenue anticipé 74LS283L. De plus, la calculatrice propose un mode signé, qui permet à l'utilisateur de réaliser des opérations avec des nombres positifs et négatifs. Ces modes devront être représentés à l'aide des LEDS embarquées dans la carte.

Ce projet sera réalisé entièrement en langage VHDL. Ce langage de description, utilisé avec une carte FPGA, offre de nombreuses possibilités et des ressources presque illimité pour la réalisation du projet. La carte utilisée pour ce projet est la DE10-Lite proposée par Intel FPGA.

B Table des références

Ci-après, le tableau des références, contenant une liste des documents qui ont aidé à comprendre et réaliser le projet.

titre	nom	date
PRJ_VHDL.pdf	Mr Schneider - ECE Paris	Septembre 2021
cours VHDL	Mr Schneider, Mr Lopez - ECE Paris	néant
DE10_Lite user manual.pdf	Intel Terasic	Novembre 2016
IEEE Standard VHDL Language Reference Manual	IEEE SA	2011

C Présentation d'un FPGA

Un FPGA (Field Programmable Gate Arrays) est un composant entièrement reconfigurable et qui sert à implémenter un système numérique, même si un simple micro-contrôleur peut souvent faire l'affaire car ils sont peu coûteux et facile à monter sur une carte à circuit imprimé. Alors vous vous demandez : pourquoi ne pas utiliser un micro-contrôleur ? Les 2 raisons pour lesquelles il faut utiliser un FPGA :

- La 1ère raison est la flexibilité. Grâce aux blocs logiques configurables, il n'y aura jamais d'impasse avec votre matériel.

- La 2ème est la vitesse. Un micro-contrôleur exécute les instructions l'une après l'autre, de manière séquentielle. De par sa nature matérielle, une structure FPGA est fondamentalement parallèle. Plusieurs actions peuvent ainsi être faites simultanément.

Selon ces 2 raisons et sans oublier le but de ce projet qui est d'acquérir des compétences techniques en concevant, fabriquant et testant un projet en langage de description hardware VHDL, la Calculatri'ECE sera un bon moyen de comprendre les applications d'un FPGA qui est utilisé dans beaucoup de domaines dont télécommunications, aéronautique, transports...



Figure 1 – FPGA DE10-Lite

D Sigles et termes

Voici la définition de quelques termes essentiels à la compréhension de ce document :

- Pin : Petite pointe en métal conducteur qui permet de faire entrer ou sortir des données des micro-contrôleurs ou des FPGA.

- Bit : Le terme bit est une contraction des mots "binary digit" (que l'on peut traduire par chiffre binaire en français). Cette unité, directement associée au système binaire, ne peut prendre que deux valeurs : 0 et 1. En informatique, le bit définit une quantité minimale d'information pouvant être transmise par un message.

- Octet : Unité de mesure de la quantité de données informatiques. Il se compose toujours de huit bits (définit précédemment) et permet de coder une information.

- FPGA : ("Field Programmable Gate Arrays" qui signifie "Portail programmable par l'utilisateur") est un circuit intégré composé d'un réseau de portes logiques configurables reliées par des interconnexions également configurables.

- VHDL : Langage de description matériel destiné à représenter le comportement ainsi que l'architecture d'un système électronique numérique.

- MSB : ("Most Significant Bit" ou le bit de poids fort) correspond au bit le plus à gauche. Par exemple : '0111' il s'agit ici du '0'.

- LSB : Le bit de poids faible (en anglais "Least Significant Bit") correspond au bit le plus à droite. Par exemple : '1110', il s'agit ici du '0'.

- GPIO : Les *General Purpose Input/Output* sont des ports d'entrées-sorties qui permettent de relier des composants électroniques comme des capteurs à des micro contrôleur et autres circuit intégrés.

- Test bench : Mise en situation permettant d'effectuer des tests sur un comportement électrique codé en VHDL

II Module de réception

La communication des opérandes vers la FPGA se fait via infrarouge. Le signal est envoyé à l'aide d'une télécommande, décodé et transmis au reste de la calculatrice. Pour ce faire l'utilisation de la télécommande Carmp3 est nécessaire.

A Fonctionnement de la télécommande

La télécommande transmet des données grâce à la lumière infrarouge, comprise sur le spectre électromagnétique entre 700nm et 1mm, elle n'est donc pas visible à l'oeil nu. La télécommande est composée d'une diode émettrice de cette lumière, qui envoie des informations à l'aide du protocole NEC.

1 Le protocole NEC

Ce protocole permet de transmettre des informations en suivant une structure de signal. Ce signal représente des bits, les '0' ayant une longueur de 1.125ms et les '1' une longueur de 2.25ms. Ceci est du au fait qu'un bit contient un temps haut de $562\mu s$ et un temps bas de $562\mu s$ pour les '0' et 1.675ms pour les '1'.

Le protocole suit une structure pour envoyer ces données. Deux informations envoyées par la télécommande sont l'adresse de la télécommande et l'instruction envoyée.

L'adresse est codée sur 8 bits. Elle permet de différencier les télécommandes des différents fabricants afin d'éviter de faux déclenchements. Suivi de cette adresse est l'inverse logique de l'adresse. L'inverse logique permet au protocole d'envoyer un signal qui fera toujours 27ms pour l'adresse, peu importe la valeur de cette dernière.

L'instruction est aussi codée sur 8 bits et envoyée suivie son inverse logique. C'est cette information qui va permettre de déterminer quel bouton est pressé sur la télécommande.

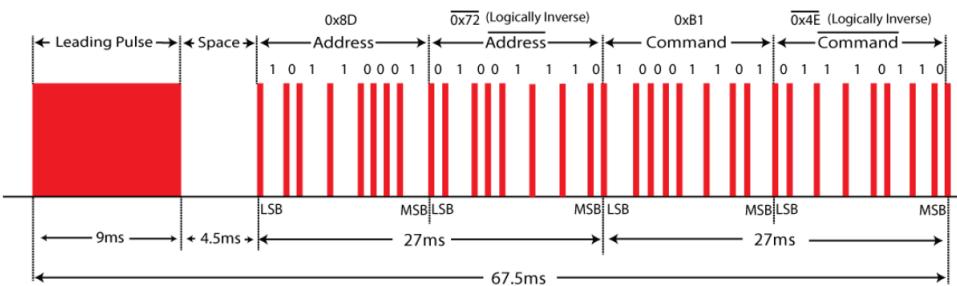


Figure 2 – Signal reçu d'une télécommande utilisant le protocole NEC

Pour le développement du projet, il faut réceptionner deux opérandes et 4 paramètres. Pour ce faire la télécommande [Carmp3](#) utilise le protocole NEC. Et de ce fait, contrôler les 4 paramètres d'opération avec des boutons en bascule.



Figure 3 – Photo de la télécommande

La télécommande est constituée de 21 boutons. 16 de ses boutons sont assigné à un chiffre, 1 au reset et 3 autres aux opérateurs. Les options additionnelles sont assignées avec des boutons.

B Fonctionnement du code

Le code fourni par M. Derraz sera utilisé pour décoder le signal infrarouge. Ce code respecte les instructions du protocole NEC et permet de différencier les 0 des 1. Il va stocker l'adresse et la commande (avec leurs inverses logiques) dans un signal "data_reg" sur 32 bits. La commande (située sur les bits 16 à 23) est ensuite stockée dans un signal nommé "command" qui prend la valeur hexadécimale de la touche pressée, décrite si dessous.

touche	0	1	2	3	4	5	6	7	8	9	+	-	EQ	100+	200+	CH-	CH+	CH	<	>	>
valeur ₍₁₆₎	2D	19	31	BD	11	39	B5	85	A5	95	2B	0F	13	33	1B	8B	8D	8F	80	81	87

Table 2 – Valeur hexadécimale de chaque touche de la télécommande

En créant une machine à état il est possible d'exécuter une opération. 3 états sont définis, le premier pour stocker l'opérande A, le deuxième pour l'opérateur et le dernier pour stocker l'opérande B. La machine accepte seulement certaines valeurs dans chaque états différents ce qui permet de blinder l'opération désirée. Les modes d'opérations sont contrôlés avec des boutons poussoirs (additionneur externe, le mode signé) pour accommoder toutes les fonctionnalités. La figure 4 illustre se fonctionnement.

Le module de réception utilise donc une machine à état, plus précisément une machine de Mealy. La sortie dépend du bouton pressé mais aussi de l'état de la machine.

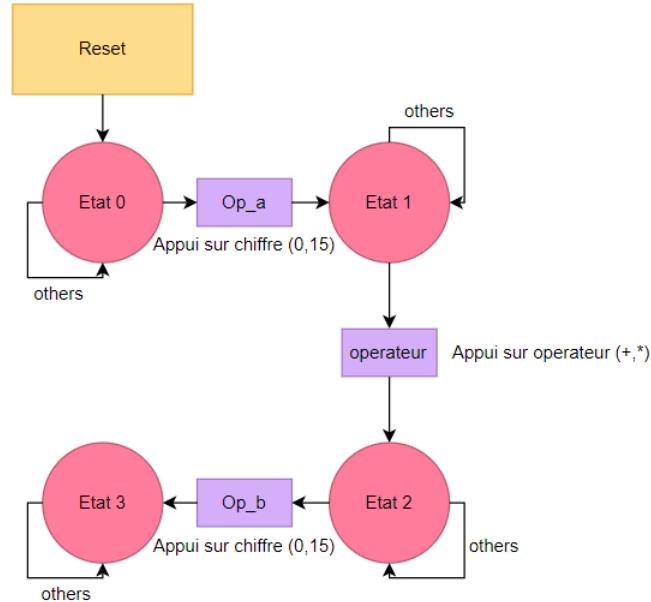


Figure 4 – Diagramme de lecture de la commande infrarouge

Par convention, la remise à 0 remet le mode non signé interne par défaut. Il faut donc sélectionner à nouveau le mode signé et/ou additionneur externe après un reset.

C Réception des opérandes signées

La télécommande permet aussi de réceptionner des valeurs négatives. En appuyant sur le bouton 'signed' au début d'une opération, cette dernière passe en mode signé. Les valeurs au dessus de 7 sont blindées à 7. Quand l'utilisateur appuie sur le bouton '-' de la télécommande avant d'entrer une opérande, un signal indique que le MSB prend 1 pour que le chiffre soit négatif, ou 0 si le bouton n'est pas pressé. A la fin de l'entrée de l'opération, le MSB prendra la valeur indiquée par le signal correspondant.

D Montage

Le capteur infrarouge est composé de 3 branches. Celle de gauche, le **récepteur**, se branche sur un pin GPIO et est lire comme un bit par la carte. La branche centrale est reliée à la **masse** et celle de droite à **5V**. La figure 5 présente son branchement.

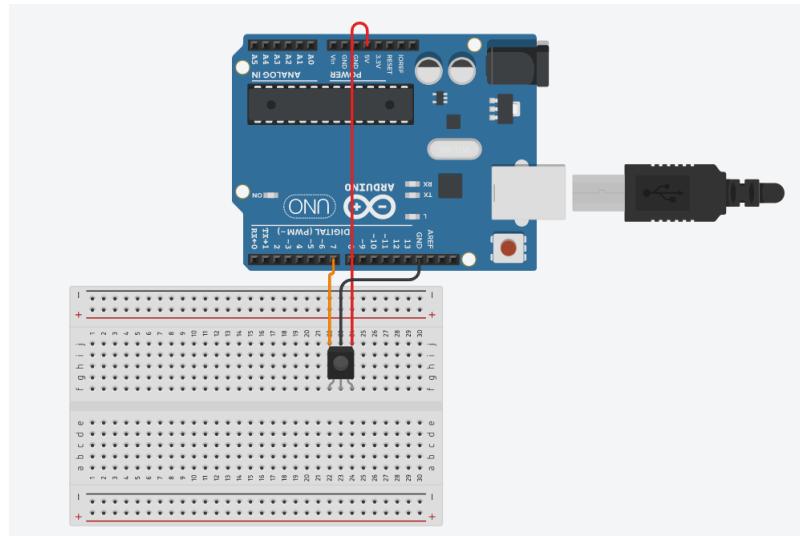


Figure 5 – Branchement du capteur infrarouge

RTL du module de réception

III Module de calcul

Dans cette partie du projet le but est de réaliser différentes opérations sur les deux opérandes reçus. Il y a deux types d'opérations à réaliser : la multiplication et la somme. Ces opérations doivent pouvoir être réalisées de façon signé et non signé. Une autre fonctionnalité est attendue, celle de pouvoir faire les opérations de façon strictement interne au FPGA (seulement en VHDL) et de façon interne et externe (à l'aide de composants numériques externes et de code VHDL). Il y a donc 8 opérations différentes à créer.

Les vues RTL et chronogrammes des opérations détaillées ci-après sont disponibles en [annexe](#).

A Mode signé et non signé

Pour pouvoir mieux comprendre le fonctionnement du mode signé ou non signé de notre calculatrice, voici le détail du fonctionnement des nombres binaires signés et non signés.

1 Les nombres binaires non signés

Les nombres binaires non signés sont les plus courants. Ils représentent, dans la base décimale, tout les entiers naturels, donc tout les entiers positifs. Le type non signé est utilisable en VHDL *unsigned*. Ce type permet d'utiliser n'importe quel vecteur binaire (tableau de bits) comme un seul et même nombre binaire , ce nombre binaire peut ensuite être utilisé dans des opérations arithmétiques.

2 Les nombres binaires signés

Ces nombres binaires représentent en base décimale l'ensemble des entiers relatifs. Cela permet donc de couvrir tout les entiers, positifs comme négatifs. Ce mode nécessite donc l'utilisation d'un bit spécifique qui porte le signe du nombre binaire. Ce bit de signe est porté par le bit de plus grande valeur (Most Significant Bit => MSB). Cela a une conséquence majeure sur le nombre binaire. Le MSB étant utilisé pour porter le signe du nombre, le nombre de bits utilisable pour contenir la valeur du nombre binaire est donc $n-1$ bits pour un nombre binaire de taille n bits.

Quand le MSB est un '0' le nombre binaire est positif et quand le MSB est un '1' le nombre est négatif. Pour un nombre binaire signé positif, la valeur du nombre est la valeur donnée par tout les autres bits. Pour retrouver la valeur d'un nombre binaire signé négatif il est obligatoire de faire le complément à 2 de ce nombre. Cette technique consiste à prendre l'inverse de tout les bits (les '0' deviennent des '1' et inversement) puis d'ajouter '1' au nombre binaire obtenu après l'inversion.

Par exemple :

0100 1011 a pour MSB '0' donc c'est un chiffre positif de valeur 75
1111 a pour MSB '1' donc c'est un chiffre négatif de valeur -1 après calcul du complément à 2

B Méthodologie de calcul

L'utilisation des signaux intermédiaires en 8 bits permet de réaliser les opérations pour ainsi ne jamais perdre d'information sur le résultat de l'opération. En effet, la plus grande valeur possible à obtenir est le résultat de la multiplication non signée suivante :

$$\begin{array}{r}
 1111 \\
 \times \quad 1111 \\
 \hline
 1110001
 \end{array}$$

Donc il est obligatoire d'utiliser des signaux intermédiaire de 8 bits pour stocker le résultat des opérations. Par ailleurs, le cahier des charges attend un résultat sur seulement 4 bits. Alors pour donner le résultat final, ainsi il faut réduire notre résultat intermédiaire de 8 bits vers 4 bits. Cette opération n'est pas un problème si le résultat intermédiaire ne nécessite que 4 bits pour contenir l'information. Par ailleurs, si le résultat intermédiaire excède 4 bits, la réalisation d'une troncature et l'utilisation d'un bit "d'overflow" est nécessaire. Ce bit indique que le résultat de l'opération dépasse 4 bits.

Par exemple, ci-après des résultats intermédiaire d'opérations :

0000 1011 ce résultat peut être stocké dans seulement 4 bits sans perdre d'information donc les valeurs seront : résultat final = 1011 et overflow = 0 ;

0110 0010 ce résultat ne peut pas être stocké dans seulement 4 bits sans perdre d'information donc les valeurs seront : résultat final = 0010 et overflow = 1 ;

L'objectif pour les opérations est donc de rendre 2 informations, un résultat sur 4 bits et une information d'overflow sur 1 bit.

Il faut réaliser toutes les opérations du même type dans la même entité VHDL. Par exemple, les opérations signés feront parti de la même entité et de même pour les opérations non signées. Ainsi tout les types calculs seront effectués pour chaque opérandes et on ne garde que le résultat souhaité à l'aide des différents paramètres réglables par l'utilisateur. A savoir : type signé ou non signé, le type d'opération et le fait de réaliser l'opération de façon interne au FPGA ou externe au FPGA à l'aide d'un additionneur.

C Quelques mots sur l'additionneur externe CLA

L'additionneur 74LS283 qui est fourni pour ce projet est un additionneur 4 bits de type CLA. CLA signifie "Carry Lookahead adder". Ce type de composant est dit "à retenue anticipée" ou "à retenue rapide". Pour comprendre le fonctionnement d'un CLA il faut premièrement comprendre le fonctionnement d'un additionneur classique.

Un additionneur classique additionne bit à bit les deux opérandes. Pour les retenues, l'additionneur effectue le calcul de la retenue de rang n en même temps qu'il calcule la somme des bits de rang n des deux opérandes. La somme des deux bits est rapide mais le calcul de la retenue est lui plus long. Il est impossible pour l'additionneur de passer au calcul du rang suivant tant que la retenue précédente n'est pas calculée. Cela est source d'une certaine lenteur dans le calcul de l'addition.

Les CLA sont conçus de telle façon que le calcul de la retenue est anticipé (lookahead en anglais). Cela permet de réduire considérablement le temps de traitement de l'addition. Le principe pour ces additionneur est de calculer la/les retenue(s) avant de calculer la somme des opérandes pour anticiper les calculs et ainsi réduire le temps de calcul.

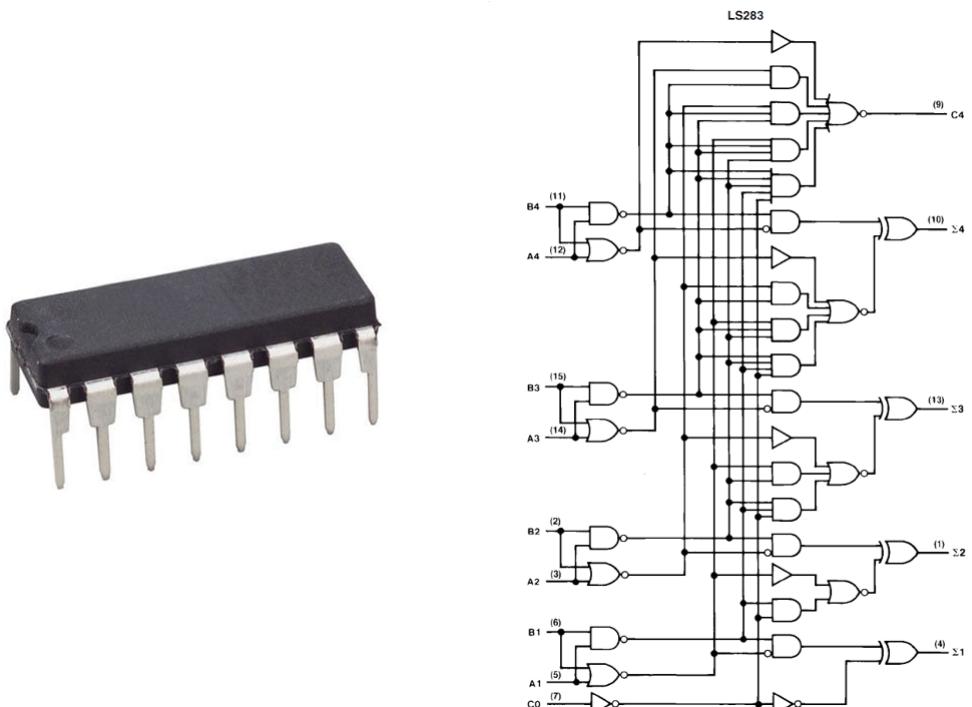


Figure 6 – Photo et architecture logique du 74LS283

D Additions

Dans cette partie, il y aura le détail du fonctionnement des différentes additions. Il est essentiel de rappeler que l'addition binaire a exactement le même fonctionnement que l'addition décimale, elle reprend la même logique de retenue. Voici un exemple d'addition binaire :

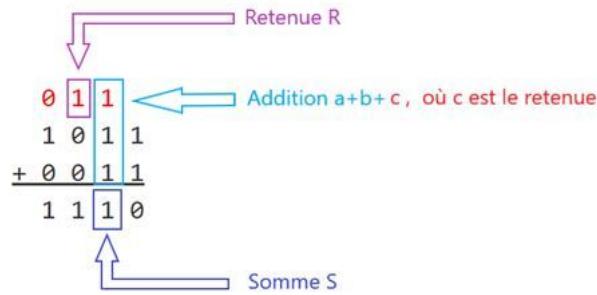


Figure 7 – Addition de deux nombres binaires

Il serait possible pour réaliser une addition de recréer en VHDL un additionneur 4 bits. Cela n'est pas souhaitable sachant que le langage VHDL permet, en incluant la bonne les bon paquetages, de réaliser des opérations arithmétiques.

Le cahier des charges stipule que les additions doivent aussi pouvoir être réalisées de façon interne comme externe au FPGA, c'est-à-dire, avec l'additionneur 74LS283. La calculatrice doit proposer un mode signé. Donc la somme de nombres signés est un soustraction ou une addition de nombres du même signe.

1 Addition non signée interne

Cette opération est la plus simple qui existe dans notre calculatrice. Cette première addition n'utilisera que des opérandes binaires positives et le calcul se fera de façon interne au FPGA.

Pour réaliser cette addition à partir de deux opérandes binaires non signées codées sur 4 bits, il faut concaténer un vecteur binaire de valeur "0ooo" au début des deux opérandes puis convertir les vecteurs binaires en type *unsigned*. Et ainsi obtenir deux signaux sur 8 bits qui servent d'opérandes. Puis la somme des deux opérandes est réalisé et stocke le résultat dans un signal intermédiaire de 8 bits.

Sum est un signal de type *unsigned*. Il permet de stocker temporairement le résultat de l'opération en attendant de choisir le résultat à rendre.

Voici le diagramme récapitulatif du fonctionnement d'une somme non signée interne au FPGA :

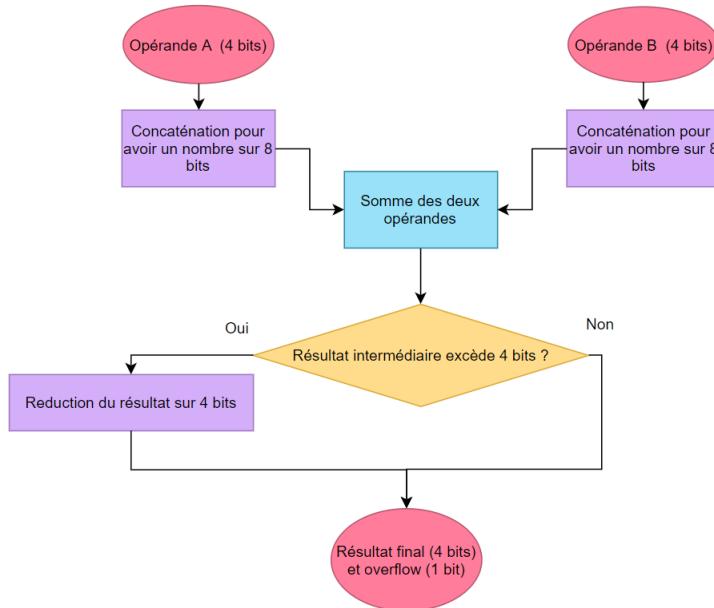


Figure 8 – Diagramme de l'addition de deux nombres binaires non signés

2 Addition signée interne

Pour cette opération, il faut utiliser maintenant des nombres signés, donc positifs et négatifs. Pour avoir des opérandes négatives, l'utilisation d'une méthode légèrement dérivée de la méthode d'écriture rigoureuse des nombres binaires négatifs est essentiel. Il faut aussi garder le principe fondamental des nombres signés qui repose sur le fait que le signe du nombre est porté par le MSB. Pour la valeur numérique du nombre, le complément à 2 n'est pas utilisé.

Par exemple, les nombres binaires sont supposés signés selon notre méthode :

1011 a pour MSB '1' donc c'est un chiffre négatif de valeur -3 (11 (binaire) = 3(décimale))

1111 a pour MSB '1' donc c'est un chiffre négatif de valeur -7

0110 a pour MSB '0' donc c'est un chiffre positif de valeur 6

La règle à appliquer pour la réalisation d'une somme binaire est la suivante : si le nombre est négatif, son complément à 2 est utilisé dans la somme, si il est positif la valeur est gardée pour réaliser la somme.

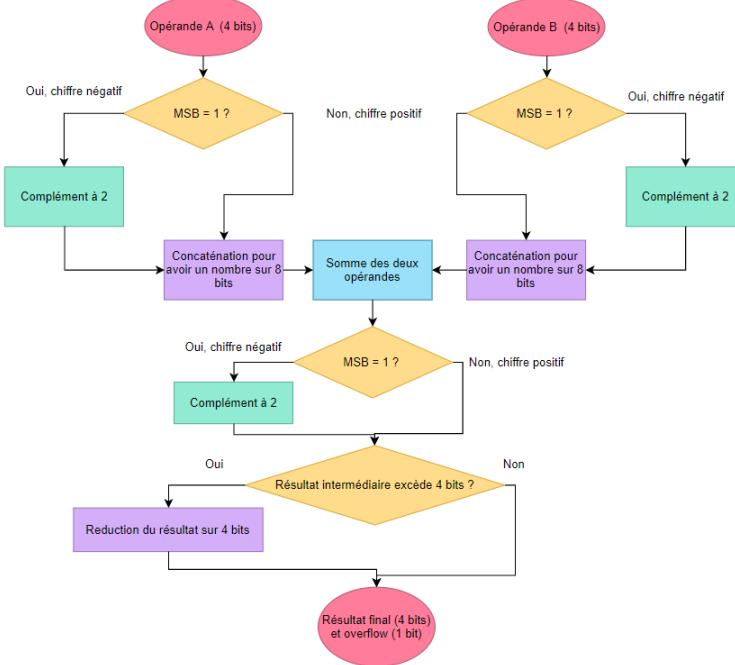


Figure 9 – Diagramme de l'addition de deux nombres binaire signés

L'addition des deux nombres suivant montre le fonctionnement de la méthode de calcul.

$$\begin{aligned} A &= 1001_{(2)} = -1_{(10)} \\ B &= 0010_{(2)} = 2_{(10)} \end{aligned}$$

A est un nombre binaire signé négatif. Pour calculer le complément à 2 d'un nombre binaire, le MSB est passé à 0, les bits sont inversés (le MSB sera donc conservé si il est égal à 1) puis on ajoute 1 :

Nombre binaire	1001
Complément à 1	0110
ajout de 1	+ 1
Complément à 2	1111

La somme du complément à 2 de A et B est à présent réalisée :

$$A_{complement2} + B = 1111 + 0010 = 0001$$

Ici le résultat de la somme à pour MSB '0', donc le nombre est positif. Il est inutile de réaliser le complément à 2 du résultat.

Vérification en base décimale :

$$A + B = -1 + 2 = 1 = 0001_{(2)}$$

Au autre exemple avec les nombres binaires signés suivants :

$$A = 1101_{(2)} = -5_{(10)}$$

$$B = 0010_{(2)} = 2_{(10)}$$

$$A = 1101$$

$$A_{complement1} = 1010$$

$$A_{complement2} = 1010 + 1 = 1011$$

La somme du complément à 2 de A et B est réalisée ci-dessous :

$$A_{complement2} + B = 1011 + 0010 = 1101$$

Ici le résultat de la somme à pour MSB '1', donc il est nécessaire de réaliser le complément à 2 du résultat.

$$resultat_{complement1} = 1010$$

$$resultat_{complement2} = 1010 + 1 = 1011$$

Vérification en base décimale :

$$A + B = -5 + 2 = -3 = 1011_{(2)}$$

La méthode de calcul pour les sommes de nombres binaires signés fonctionne correctement, aussi bien pour les résultats positifs que négatifs.

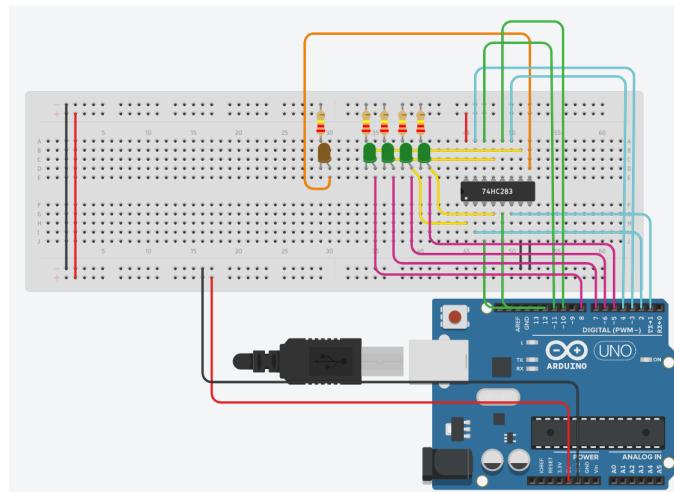
Quant au code VHDL de cette fonctionnalité, les opérandes sont converties en type *unsigned* (type non signe) pour pouvoir le considérer comme un nombre arithmétique qui sera sommable ou multipliable. Par ailleurs, pour le résultat est attendu en tant que *std_logic_vector (3 DOWNTO 0)*, alors il est impératif de reconvertis le résultat de la somme en vecteur binaire pour ensuite le représenter à l'aide des fonctionnalités d'affichage.

[Photo de l'addition signée interne en annexe.](#)

3 Addition non signée externe

Pour réaliser l'additionneur avec des composants externe, il est nécessaire de passer par l'additionneur 4 bits SN74LS283N. Ce composant numérique permet de réaliser l'addition de deux nombres binaires sur 4 bits et de rendre un résultat sur 4 bits et une retenue de sortie sur 1 bit. Il est même possible d'utiliser une retenue d'entrée avec cette additionneur mais cela ne sera pas traité.

L'enjeu pour cette opération est de pouvoir sortir les opérandes de la FPGA pour les utiliser avec l'additionneur externe. Pour cela, les pins GPIO (General Purpose Input/Output) sont utilisés.



Voici un diagramme récapitulatif du fonctionnement de l'addition non signée externe :

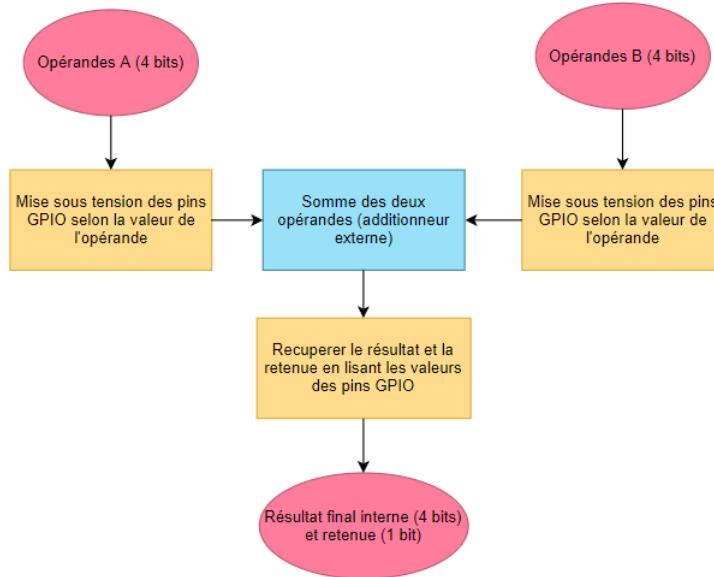


Figure 11 – Diagramme du fonctionnement de l'addition externe non signée

Dans cette fonctionnalité, l'entièreté des calculs sont réalisés dans l'additionneur, donc le code VHDL est très réduit et se résume à envoyer les deux opérandes à l'additionneur via les pins GPIO et réceptionner le résultat et la retenue du calcul avec d'autres pins GPIO. L'affichage du résultat est fait par une autre partie du programme.

[Photo de l'addition non signée externe en annexe.](#)

4 Addition signée externe

Après avoir développé les additions externes non signées, il est à présent temps de développer l'addition externe signée en utilisant le CLA. La difficulté de cette opération réside dans le fait que l'additionneur ne puisse additionner que 2 opérandes binaires sur 4 bits. Donc plusieurs traitements sont à réaliser avant d'utiliser le CLA pour faire la somme finale. Il est essentiel de rappeler que l'addition d'un complément à 2 d'un nombre binaire avec un autre nombre binaire, revient à faire une différence.

Voici dans un premier temps le diagramme fonctionnel de l'addition externe signée :

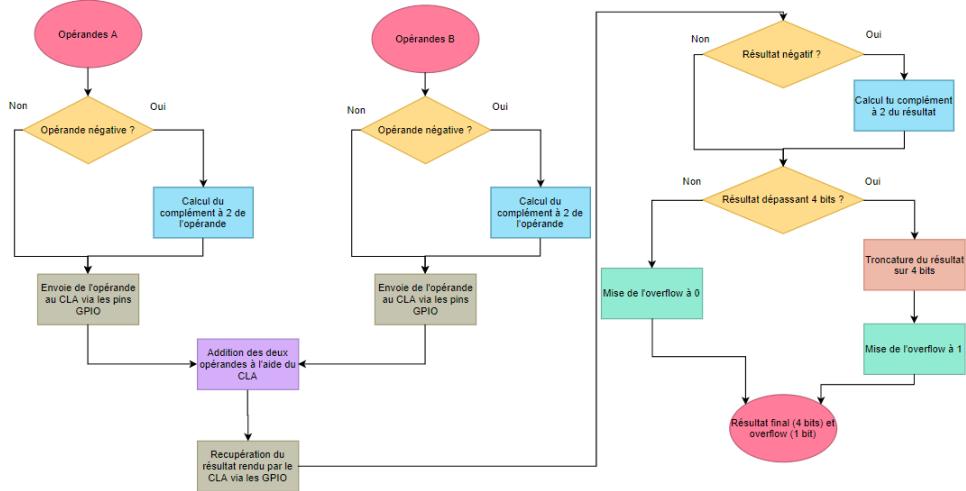


Figure 12 – Diagramme du fonctionnement de l'addition externe signée

La première étape est de déterminer le signe des opérandes reçus. Si l'opérande est négative on réalise le complément à 2 de cette dernière. Puis, il faut ensuite envoyer les deux opérande binaire vers le CLA à travers les pins GPIO.

Le CLA réalise l'addition des deux opérandes et renvoie le résultat et la retenue de sortie à la carte DE10-Lite à l'aide des pins GPIO. Plusieurs tests sont effectués sur le résultat obtenu. Le premier test concerne le signe du résultat. Pour répondre à ce test, le programme étudie le signe des opérandes. Si les deux opérandes ont le même signe, alors le programme en déduit que le signe du résultat sera le même que celui des opérandes. Si les deux opérandes n'ont pas le même signe, le résultat sera obligatoirement compris entre -7 et 7. Alors le bit de retenue donnera le signe du résultat.

Ensuite, il faut déterminer si le résultat de la somme excède 3 bits (car le dernier bit est réservé pour le signe). Dans le cas où les opérandes ont le même signe, le programme vérifie que le 4ème bit du résultat est à 0, si c'est bien le cas, l'opération n'excède pas 4 bit et donc le résultat est de la bonne dimension. Dans le cas où les opérandes n'ont pas le même signe, le résultat est strictement compris entre -7 et 7 (4 bits) donc il suffit de lire le résultat en sortie du CLA.

[Photo de l'addition signée externe en annexe.](#)

E Multiplications

Le principe de multiplication en base binaire reprend le même fonctionnement que les multiplications en base dix. La multiplication binaire doit respecter les règles de calcul suivants :

$$1 \times 0 = 0 \quad (1)$$

$$1 \times 1 = 1 \quad (2)$$

$$0 \times 0 = 0 \quad (3)$$

Ci-après, un exemple de multiplication binaire non signée. Cette multiplication respecte les règles de calculs énoncées plus tôt et reprend la méthode suivante :

- multiplier l'opérande du "dessus" par chaque bit de l'opérande du "dessous"
- ajouter n 'o' avant d'écrire le résultat de la multiplication du n-ième bit (la numérotation du rang des bits commencent à 0)
- sommer les résultats partiels selon les règles de l'addition binaire citées précédemment

1 0 1 0	→	Multiplicand
× 1 0 1 1	→	Multiplier
<hr/>		
1 0 1 0	→	Partial product 1
1 0 1 0	→	Partial product 2
0 0 0 0	→	Partial product 3
1 0 1 0	→	Partial product 4
<hr/>		
1 1 0 1 1 1 0		
<hr/>		

Figure 13 – Exemple d'une multiplication binaire

Le résultat maximum qu'il est possible d'obtenir grâce à la multiplication de deux opérandes sur 4 bits est la suivant :

$$1111 \times 1111 = 11100001 \quad (4)$$

Ce résultat exige d'être codé sur 8 bits pour ne pas perdre d'information. Or, dans un premier temps, le résultats doit être sur 4 bits. Il est donc nécessaire d'utiliser, comme pour l'addition, un résultat sur 4 bits et un autre information sur 1 bit d'overflow (le résultat dépasse 4 bits et donc il est nécessaire d'en faire une troncature).

1 Multiplication non signée interne

L'objectif est de réaliser un multiplication non singée, interne au FPGA. Comme montré précédemment le résultat maximum que la multiplication de deux nombres binaire non signée est $1110\ 0001$. Donc il faut à nouveau utiliser un vecteur binaire de 8 bits pour stocker temporairement le résultat de la multiplication.

Or en VHDL, il est impératif que les opérandes et les résultats aient la même dimension, dans ce cas un vecteur binaire de 8 bits. Pour cela, nous concaténons quatre zéros avec les opérandes. Puis il faut convertir la valeur obtenue après concaténation en type *unsigned* pour l'utiliser comme unité arithmétique. Ainsi, après la conversion, le résultat est stocké dans un signal temporaire de type *unsigned*. Il ne reste plus qu'à convertir encore le résultat temporaire en vecteur binaire (`std_logic_vector(3 DOWNTO 0)`).

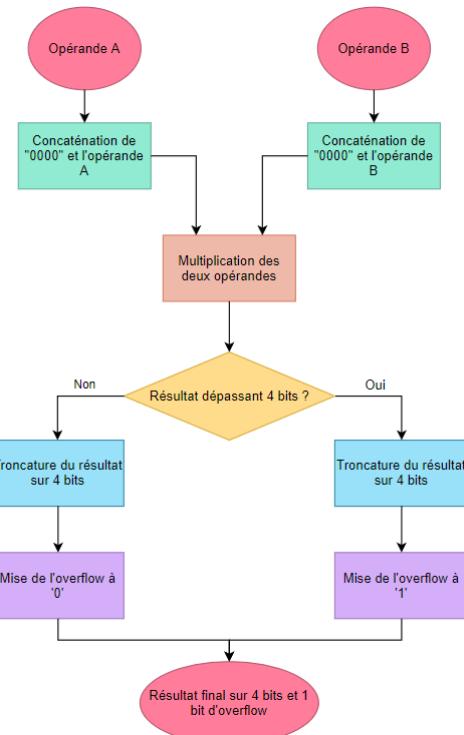


Figure 14 – Diagramme de la multiplication interne non signée

L'utilisation du type *unsigned* permet de considérer le vecteur binaire comme une unité arithmétique ainsi on peut utiliser l'opérateur "*" pour réaliser simplement la multiplication des opérandes.

2 Multiplication signée interne

Passons à présent à la multiplication interne signée. Il est intéressant de remarquer que les règles arithmétiques de multiplication des signes sont les mêmes que la table de vérité de la porte ou exclusive (XOR). Voici la table de vérité de cette porte

Table 3 – Table de vérité de la porte XOR

XOR	A=0	A=1
B=0	0	1
B=1	1	0

Voici les règles de calculs sur les signes dans une multiplication. Associons le 0 de la table de vérité à +1 des règles suivantes et le 1 de la table de vérité au -1 des règles arithmétiques.

$$1 \times 1 = 1 \quad (5)$$

$$-1 \times 1 = -1 \quad (6)$$

$$1 \times -1 = -1 \quad (7)$$

$$-1 \times -1 = 1 \quad (8)$$

Donc pour déterminer le signe d'une multiplication interne signée, il faut réaliser le XOR des bits de signe des deux opérandes. Le signe de la multiplication est stocké dans un bit temporaire et il sera concaténé au résultat de la multiplication des 3 derniers bits de chaque opérandes.

Tout comme pour les additions internes, il est nécessaire d'utiliser le type *unsigned* pour réaliser des opérations arithmétiques simples. Ainsi, le programme réalise la multiplication des 3 derniers bits de chaque opérande à l'aide l'opérateur "*".

Pour obtenir le résultat final de l'opération, il faut réaliser la concaténation du bit de signe obtenue avec la porte XOR et le résultat de la multiplication du reste des opérandes tronqué sur 3 bits. Ainsi, la concaténation des deux éléments donne un vecteur binaire de 4 bits, ce vecteur est notre résultat final.

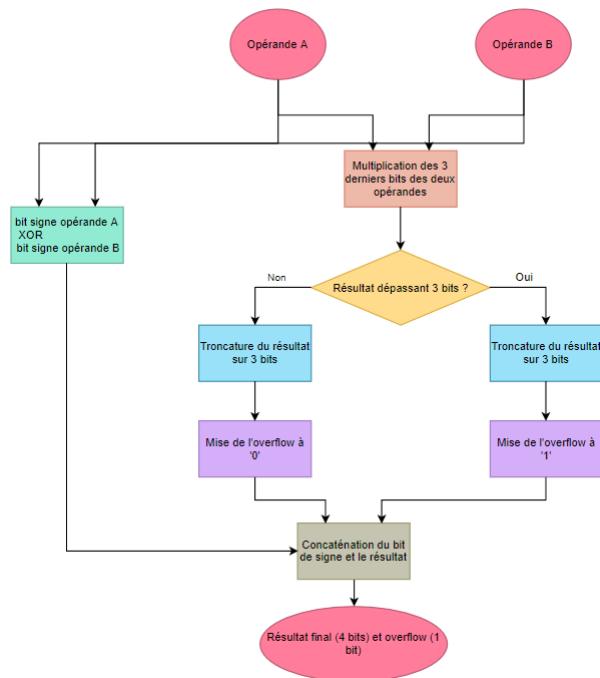


Figure 15 – Diagramme de la multiplication interne signée

Ce diagramme représente le fonctionnement logique et les opérations à mener pour réaliser une multiplication signée interne.

Vérifions à présent le bon fonctionnement de cette méthode de calcul, soient deux nombres binaire signés selon la convention utilisée au cours du projet : $A = 1010_{(2)}$ et $B = 0011_{(2)}$. Soit la multiplication de A par B :

Calcul du signe du résultat de la multiplication :

$$\text{signe_res} = A(3) \text{ XOR } B(3) = 1 \text{ XOR } 0 = 1$$

Voici le résultat de la multiplication du reste des deux opérandes (3 derniers bits) :

$$\text{res_multi} = A(2- > 0) \times B(2- > 0) = 010 \times 011 = 110$$

Pour obtenir le résultat final de cette multiplication, il reste à concaténer le bit de signe avec le résultat de la multiplication :

$$(\text{signe_res}) \text{ concat } (\text{res_multi}) = 1 \text{ concat } 110 = 1110$$

Vérification en base 10 :

$$A = 1010_{(2)} = -2_{(10)}$$

$$B = 0011_{(2)} = 3_{(10)}$$

Or :

$$A \times B = -2 \times 3 = -6 = 1110_{(2)}$$

Donc cette méthodologie de calcul pour la multiplication signé interne au FPGA fonctionne correctement et nous permet de calculer des résultats positifs comme négatifs, avec ou sans retenues de sortie.

[Photo de la multiplication signée interne en annexe.](#)

IV Module de sonorisation

Pour ce module, le résultat de l'opération doit être bipé. A cette fin, un buzzer est utilisé, branché sur les pins GPIO de la carte. Le buzzer a besoin de recevoir un signal périodique et peut donc changer de fréquence en fonction du nombre de périodes en une seconde.

A Fonctionnement du code

Pour ce faire, l'horloge interne de la carte de 10MHz est utilisée (L'horloge de 50MHz est utilisée par le module de réception). L'objectif est de biper le résultat à une fréquence audible (entre 20Hz et 20KHz). En mode signé, il faut aussi différencier un résultat négatif d'un résultat positif. Deux fréquences différentes seront générées pour les chiffres négatifs et positifs.

Pour générer une fréquence, un compteur va incrémenter à chaque front montant de l'horloge interne. Quand ce compteur va atteindre une certaine valeur, le compteur va se réinitialiser, et basculer la sortie.

Voici ci-dessous, un tableau décrivant les valeurs nécessaires pour obtenir différentes fréquences :

Table 4 – Fréquence de 50Mhz

Fréquence (Hz)	valeur du compteur	temps (ms)
1	25,000,000	1000
10	2,500,000	100
100	250,000	10
200	125,000	5
400	62,500	2.5

Table 5 – Fréquence de 10Mhz

Fréquence (Hz)	valeur du compteur	temps (ms)
1	5,000,000	1000
10	500,000	100
100	50,000	10
200	25,000	5
400	12,500	2.5

A ce stade, le buzzer sonne à une certaine fréquence en continu, mais il faut pouvoir le faire bipper et l'arrêter. Pour cela un deuxième compteur 'temps' dépendant de la clock doit être créé. Si le buzzer veut bipper et que ce signal est inférieur à 0,2 secondes, le buzzer sonne. Si ce signal dépasse 0,2 secondes, il s'arrête de sonner. Quand le signal atteint 0,8 secondes il se remet à 0 pour pouvoir sonner à nouveau.

La dernière chose à faire est de sonner le bon nombre de fois le buzzer, en fonction du résultat. Un dernier signal (décompteur) est créé qui va prendre une valeur correspondant à la sortie en binaire. Ce décompteur est activé à l'aide d'un bouton. Quand le bouton est pressé, le décompteur prend la valeur absolue du résultat. Il va diminuer à chaque remise à 0 du compteur temps, jusqu'à atteindre 0. Enfin, si le décompteur est inférieur à 0, le buzzer s'arrête. En fonction du signe, le buz-

zé sonne avec une fréquence différente en variant la valeur que le premier compteur doit atteindre.

Le schéma si dessous représente la fonctionnalité.

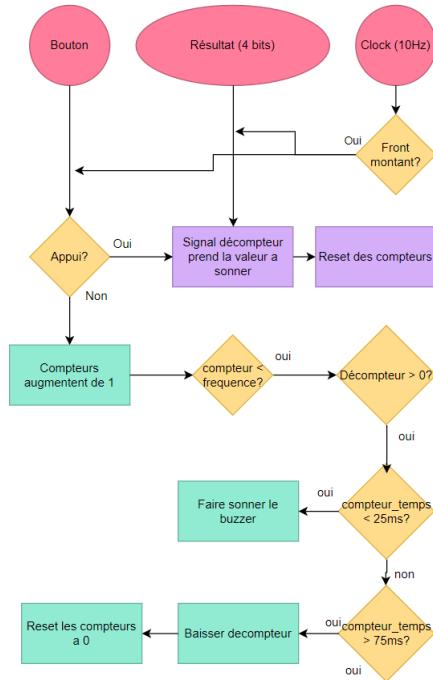


Figure 16 – Diagramme du fonctionnement du buzzer

B Montage du module

Le buzzer n'est pas intégré dans la DE10-Lite, il est donc externe. Ce buzzer se branche en **entrée** GPIO et en **sortie** à la masse.

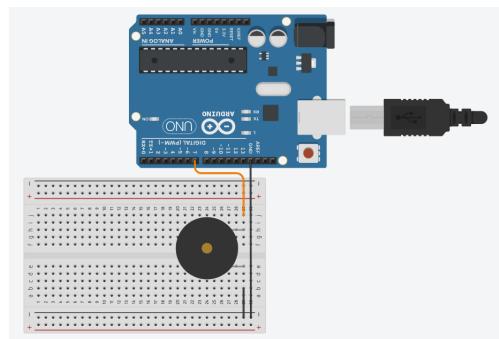


Figure 17 – Branchement du buzzer

C Résultat signé

La fréquence d'un résultat positif est de 800Hz. Quand le mode signé est actif, un résultat négatif doit sonner avec une fréquence différente. En changeant la valeur que doit atteindre le premier compteur, la fréquence varie. Pour une valeur positive le compteur (10MHz) doit atteindre 6,250 (800Hz) et pour une valeur positive 25,000 (200Hz).

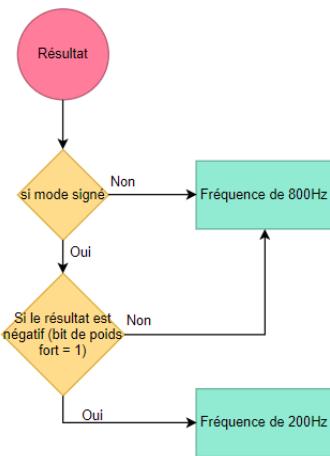


Figure 18 – Schéma de la sélection de la fréquence

RTL du module de sonorisation

V Module d'affichage

L'objectif de cette partie est d'afficher, sur les afficheurs 7-segments les opérandes et le résultat de notre opération.

A Fonctionnement d'un afficheur 7-segment

Un afficheur 7-segments comporte huit pins d'entrée, nécessaire à l'utilisation de celui-ci. Sept d'entre eux sont dédiés à l'affichage du chiffre en question. Le huitième, est lui utilisé pour afficher un point (pour les nombres décimaux).

Pour ce projet, six afficheurs 7-segments sont disponibles. Deux seront pour le premier opérande, deux autres pour le second opérande et deux derniers pour le résultat. Cependant il faut prendre en compte le signe du résultat pour savoir comment afficher ces nombres. Dans le cas où le résultat est positif l'affichage se fait comme dit précédemment. En revanche, dans le cas où le nombre rendu serait négatif, le signe doit être afficher. De ce fait, les opérandes ne seront pas affichées afin de pouvoir afficher le résultat (qui nécessite deux afficheurs) et son signe (qui nécessite un afficheur à lui seul). Seul trois afficheurs seront donc utilisés en cas de résultat négatif.

Voici un diagramme fonctionnel permettant de visualiser le bon fonctionnement de ce code.

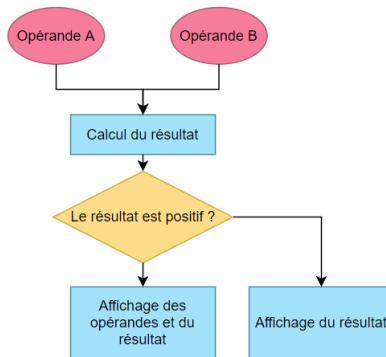


Figure 19 – Diagramme d'affichage

B Exemple d'affichage d'un chiffre

Dans cet exemple, il sera montré comment afficher le chiffre 2.

Comme expliqué précédemment, l'affichage d'un chiffre nécessite une information sur huit bits. A l'aide du schéma ci-dessous, il est possible de déterminer le nombre binaire à envoyer pour obtenir le bon chiffre.

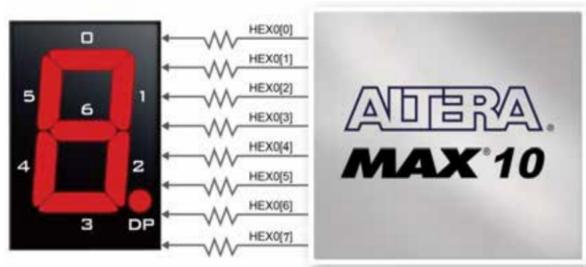


Figure 20 – Schéma afficheur 7-segments

Les afficheurs à 7 segments de la carte sont des afficheurs "common anode", ils sont donc allumés en envoyant un état bas de la carte (et éteint à l'état haut). Les pins à alimenter pour obtenir un '2' sont les pins 0, 1, 3, 4 et 6, le code binaire correspondant est donc "10100100".



Figure 21 – Résultat de l'affichage

C Affichage du résultat sur des LEDS externes

Le calcul du résultat s'affiche sur les afficheurs 7 segments, comme présenté précédemment. Sur les afficheurs 7 segments, le résultat est rendu en base 10. Le résultat est également donné sur cinq LEDS. Quatre LEDS jaunes sont utilisées pour le nombre en lui même et une LED rouge pour la retenue de sortie.

En effet, le résultat est transmis sur quatre bits. Il peut donc être affiché (en base 2). La cinquième LED est réservé pour le cas où le nombre ne pourrait pas être transmis sur quatre bits (lorsqu'il est supérieur à 15 en mode non signé ou supérieur à 7 / inférieur à -7 en mode signé). Dans ce cas, la LED de retenue est allumée. En ce qui concerne les autres LEDS seuls les quatre derniers bits du résultat sont retranscrits par les quatre LEDS.

D Affichage des modes sur des LEDS internes au FPGA

Cette calculatrice propose plusieurs modes de calculs listé ci-après :

- choix du type d'opération
- mode signé et non signé
- utilisation d'un additonneur externe de type CLA pour réaliser les opérations

Il est préférable pour faciliter l'utilisation de la calculette de pouvoir représenter ces différents modes. La carte DE-10 Lite embarque 10 LEDS rouges programmables. Les différents modes sont en réalité des états logique dans le fonctionnement interne de la calculatrice. Il suffit donc de déclare 3 sorties reliées à 3 LEDS pour ainsi afficher les modes de calculs.

Voici les règles d'affichage, le 1 correspond à une LED allumée et le 0 à une LED éteinte :

- LED représentant le type d'opération : 0 => addition, 1 => multiplication
- LED représentant le mode signé : 0 => non signé, 1 => signé
- LED représentant le mode externe (CLA) : 0 => calcul interne, 1 => calcul externe avec le CLA

[RTL du module d'affichage](#)

VI Architecture du projet

Les différents modules ont été détaillés dans les parties supérieures. Ces modules sont interdépendants et il est donc nécessaire de créer une entité supérieure qui regroupe les modules et établie les connexions entre ces derniers. C'est notamment dans cette entité supérieure que les entrées et les sorties seront assignés aux pins de la carte DE10-Lite. Cette architecture décrit le fonctionnement global du code VHDL et l'utilisation des variables du projet.

Ces modules comportent aussi des composants externes à la carte FPGA, il est donc nécessaire de définir une architecture électronique. Cette architecture représente toutes les connexions physique entre les composants. Cette partie ne concerne que les éléments matériels.

A Architecture logicielle

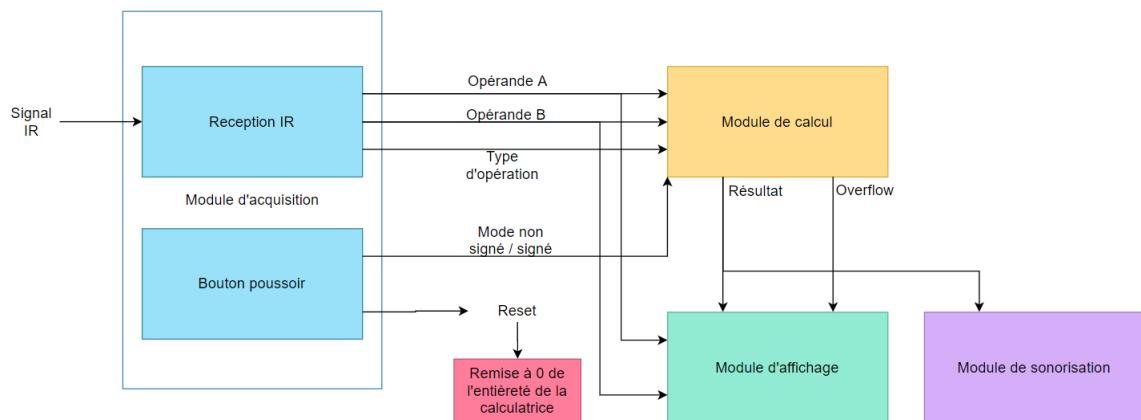


Figure 22 – Architecture logicielle du projet

Dans le schéma ci-dessus est représenté les inter-connections logicielles entre les différents modules. Les entrées, sorties, variables et signaux essentiels ont été représentés sur le schéma, voici le type et la dimension de ces données :

- signal IR : valeurs hexadécimale envoyées par la télécommande
- opérande A, opérande B, résultat : vecteur binaire de 4 bits
- type d'opération, reset, overflow, mode signé / non signé : état logique standard (1 bit)

Ces différentes données sont envoyées et récupérées par les nombreuses entités du projet pour ainsi permettre de réaliser un calcul complet (saisie, calcul, affichage).

L'entité supérieur est une machine d'état du type machine de Moore. Elle se comporte comme un aiguillage. Le programme calcule et stocke les résultat de toutes les opérations de chaque modes dans des signaux temporaires. En fonction des différents modes choisis par l'utilisateur, la machine de Moore adopte l'état correct et aiguille le bon résultat vers les modules d'affichage et de sonorisation.

B Architecture électronique

ci-après le schéma de branchement du projet. Les différents modules y sont aussi représentés. La carte DE10-Lite n'étant pas disponible dans le logiciel de schématisation utilisé, on considérera la carte Arduino Uno utilisée comme étant la carte DE10-Lite.

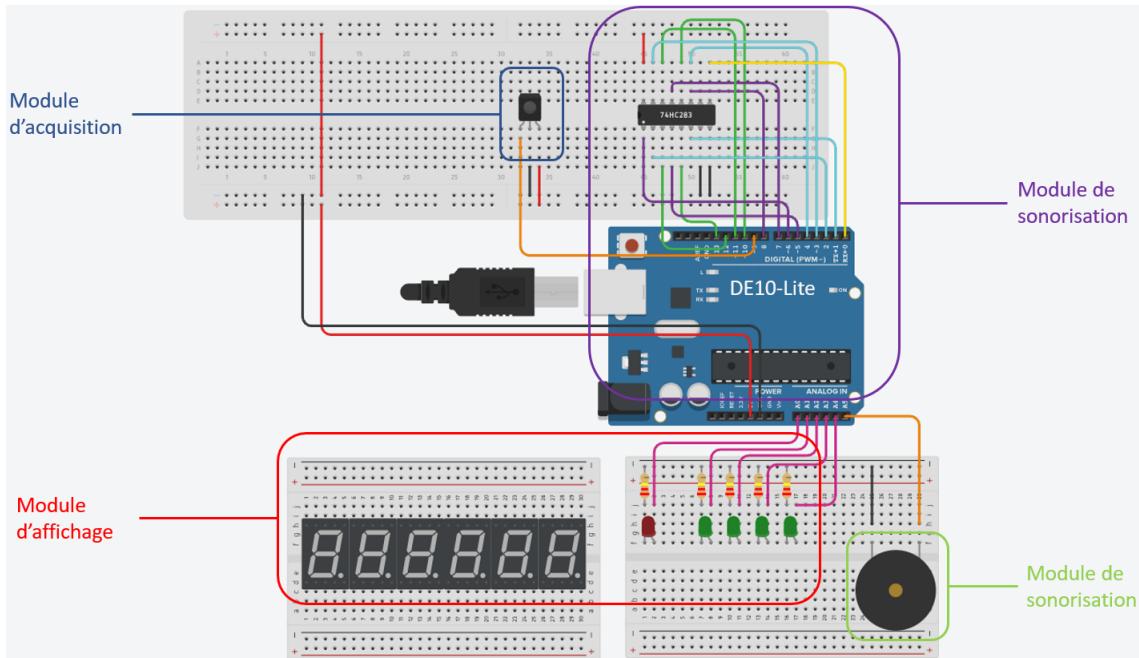


Figure 23 – schéma de branchement du projet

La carte DE10-Lite est bien plus complète qu'une Arduino Uno, la carte FPGA embarque les 6 afficheurs 7-segments, 2 boutons poussoir, 10 LEDS et 10 bouton switch. Nous utilisons les afficheurs pour présenter les opérandes utilisées et le résultat. Un bouton switch permet de choisir de réaliser une opération en mode CLA. Les modes utilisés pour le calcul sont affichés sur 3 des LEDS embarqués. Enfin, un bouton poussoir permet de passer le calcul en mode signé et l'autre permet le reset de la calculatrice.

VII Partie bonus : une calculatrice 8 bits

Une fois le projet réalisé, il est possible d'imaginer de poursuivre avec certaines fonctionnalités supplémentaires.

A Un résultat sur 8 bits

Une amélioration possible du rendu du résultat est de sortir celui-ci sur 8bits. En effet, en cas de multiplication, le résultat peut monter jusqu'à 225 (15×15). Dans ce cas, trois afficheurs sont nécessaire. Il n'est donc pas possible d'afficher les opérandes. Cependant, cela n'est valable que pour les opérations non-signées. Si l'opération est signée le MSB (Most Significant Bit) sera dédié au signe. Il n'est donc possible de monter que jusqu'à 49 (7×7), le dernier afficheur est donc dédié au signe.

B Division euclidienne

La calculatrice permet à ce stade de faire les opérations suivantes : addition signé et non signé (interne et externe) et multiplication signé et non signé (interne). Or les calculatrices classiques permettent de faire la division. Il serait donc intéressant d'implémenter cette nouvelle fonctionnalité.

L'amélioration proposée ici est une division euclidienne dans laquelle seul le résultat serait renvoyé. Le reste, lui ne serait pas communiqué. La division est calculé selon les mêmes principes arithmétiques que la multiplication signée et non signée interne.

C Soustraction

La calculatrice, réalisée ici, permet d'obtenir une soustraction. Mais pour cela, il faut passer par l'addition signé. Cela limite donc la calculatrice car dans ce cas, le MSB, sera dédié au signe de l'opérande. La réalisation d'une calculatrice permettant de soustraire une opérande B sur quatre bits à une opérande B également sur quatre est détaillé ci-dessous.

Le fonctionnement de cette soustraction nécessite que l'opérande A soit positive et supérieur à l'opérande B (positive également). Le résultat renvoyé est donc sur quatre bite au lieu de trois avec l'original. Ce dernier est calculé de la même manière que l'addition non signé, avec le type *unsigned*. A la différence que dans notre cas l'opérateur est le moins.

VIII Organisation du travail de groupe

Le projet de la calculatrice a nécessité une réelle organisation au sein du groupe. Voici un diagramme de Gantt permettant d'avoir un aperçu de la répartition des tâches dans le groupe.

	Semaine 1	Semaine 2	Semaine 3	Semaine 4
Module de réception			Léopold	
Module de calcul (opération interne)	Tristan	Tristan		
Module de calcul (opération externe)			Tristan	
Module de sonorisation	Matia	Léopold		
Module d'affichage	Victor	Victor		
Rapport			Tristan, Victor, Léopold, Matia	Tristan, Léopold, Victor, Matia

IX Conclusion

Ce projet très concret a permis au groupe d'appliquer et intégrer les différents concepts du langage VHDL et de la programmation sur FPGA. Le projet et les cours ont débuté simultanément ce qui a permis au groupe de s'exercer sur les notions apprises.

Un des étudiants du groupe étant nouveau dans l'école et n'ayant jamais fait d'électronique par avant, ce projet a été l'occasion de revoir des connaissances basiques en électronique et être attentif aux difficultés rencontrées par ce dernier. Par ailleurs, cette différence de niveau a pu se faire ressentir au cours du développement des parties complexes du projet.

Le groupe avait commencé le développement de la fonctionnalité bonus qui est la présentation à l'utilisateur des résultats sur un moniteur via le port VGA de la carte. Cette fonctionnalité étant longue est complexe à réaliser. C'est pourquoi le groupe a décidé d'abandonner le développement de cette dernière au profit du reste du projet. Avec le recul, il aurait fallut se renseigner plus tôt sur la complexité de ce bonus et en débuter le développement dès le début du projet.

X Sources

- Chaine youtube de Ben Eater : [Youtube](#)
- Datasheet du 74LS283 : [mil.ufl.edu](#)
- Référence VHDL : [VDLANDE.com](#)
- Module de réception IR : [www.circuitvalley.com](#)
- Cours de VHDL S3 - ECE Paris : [Moodle](#)

XI Annexes

A Vues RTL

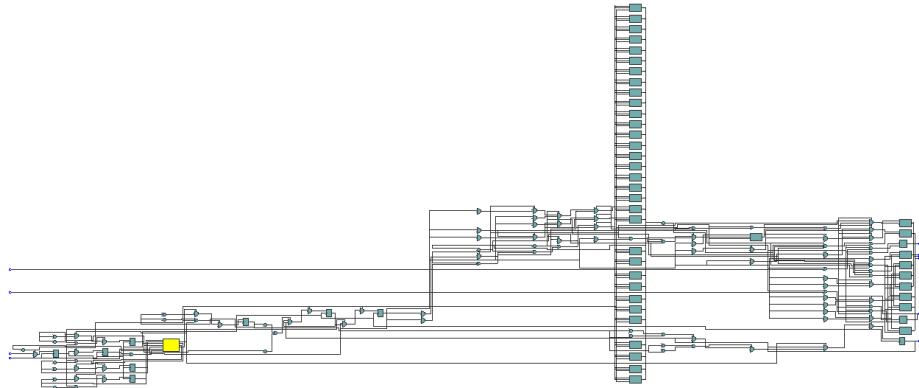


Figure 24 – Vue RTL du module de réception

[Retour a module de réception.](#)

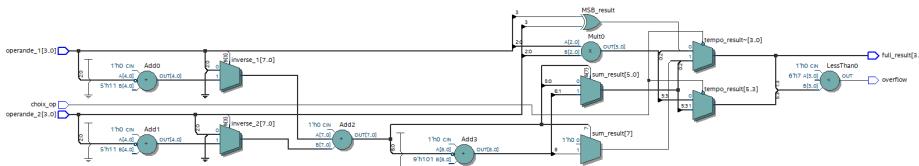


Figure 25 – Vue RTL des opérations signées interne

[Retour a la section addition signée interne.](#)

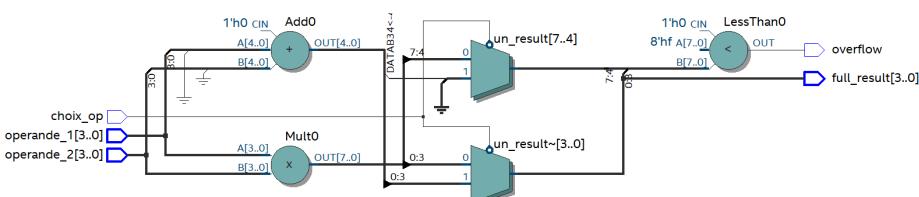


Figure 26 – Vue RTL des opérations non signées interne

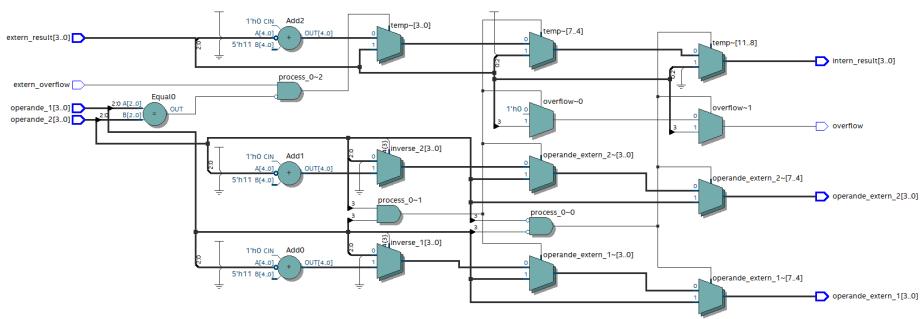


Figure 27 – Vue RTL des opérations signées externe

[Retour a la section addition signée externe.](#)

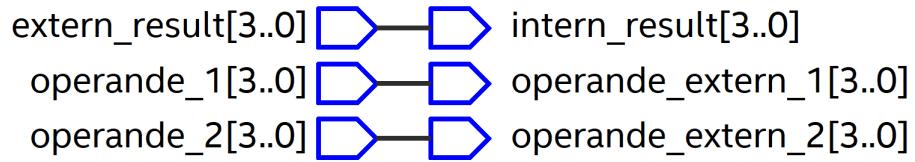


Figure 28 – Vue RTL des opérations non signées externe

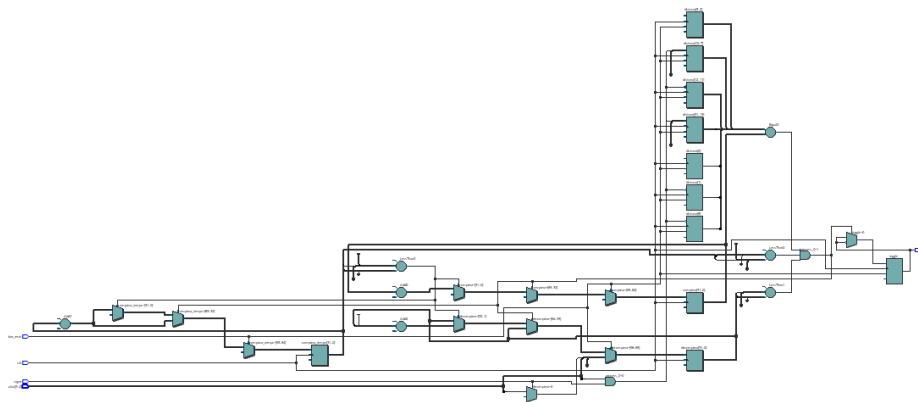


Figure 29 – Vue RTL du module de sonorisation

[Retour a module de sonorisation.](#)

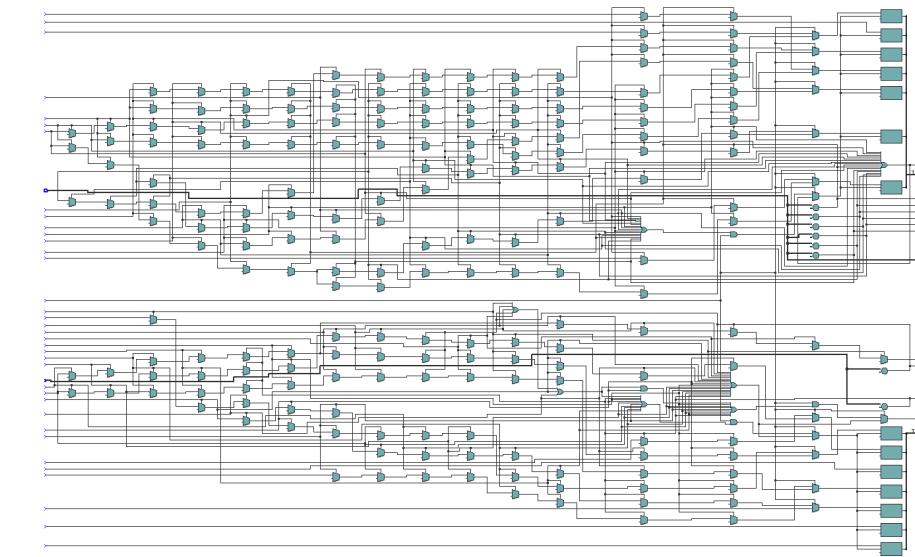


Figure 30 – Vue RTL du module d'affichage

[Retour a module d'affichage.](#)

B Chronogrammes

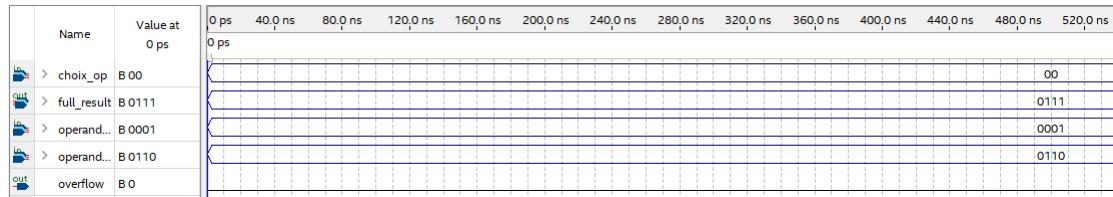


Figure 31 – Chronogramme de test de l'addition interne non signée

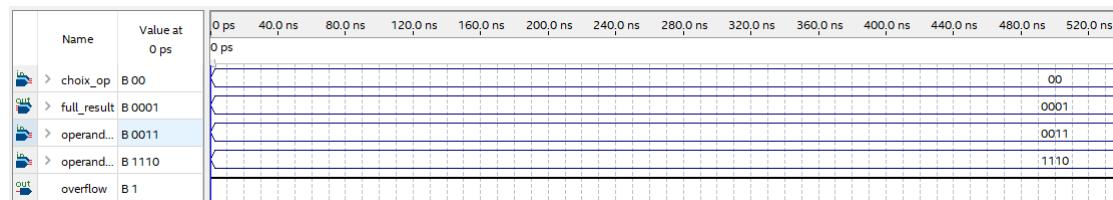


Figure 32 – Chronogramme de test de l'addition interne non signée avec overflow

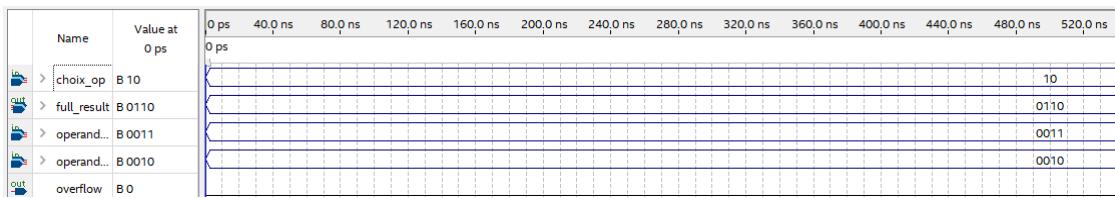


Figure 33 – Chronogramme de test de l'addition interne non signée

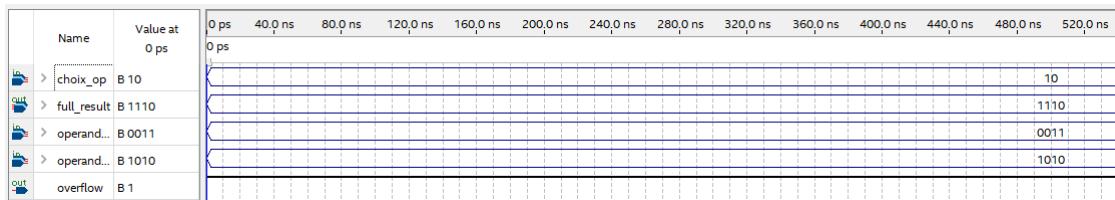


Figure 34 – Chronogramme de test de l'addition interne non signée avec overflow

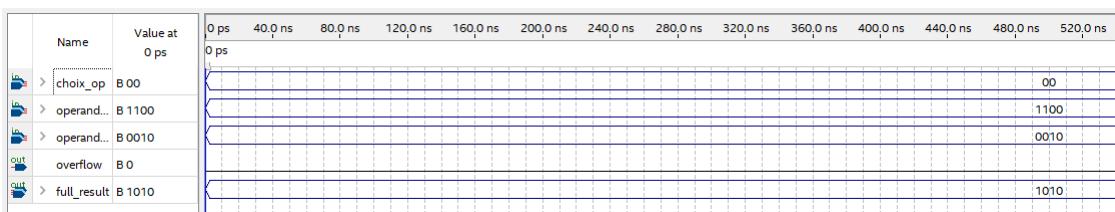


Figure 35 – Chronogramme de test de l'addition interne signée

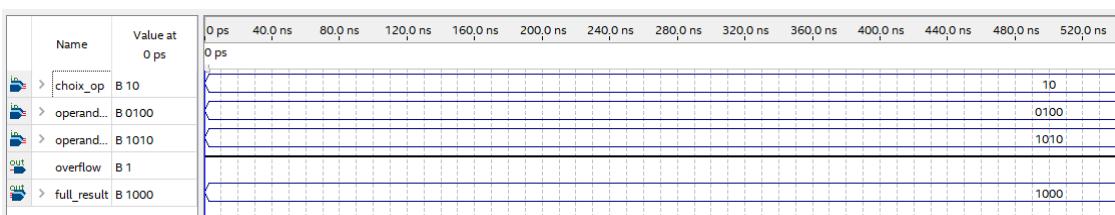


Figure 36 – Chronogramme de test de la multiplication interne signée

C Images

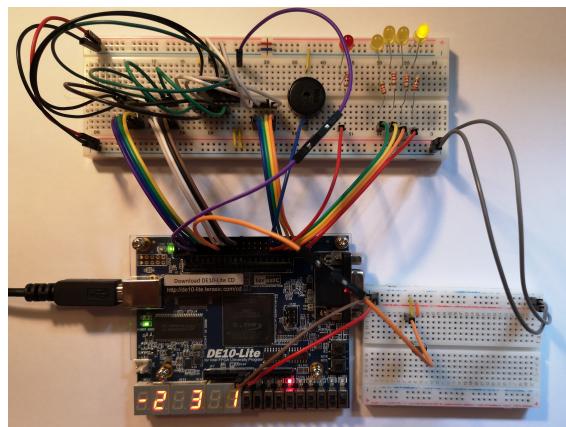


Figure 37 – Addition interne signée

[Retour à la section addition signée interne.](#)

Voici un exemple permettant d'illustrer le bon fonctionnement de l'addition interne signée. En effet, la LED 5 de la carte est allumé donc le mode utilisé est le mode signé, la LED 3 est éteinte donc l'opération réalisée est une somme. La LED 1 de la DE10-Lite est également éteinte donc cette opération n'est pas réalisé par le CLA, elle est donc interne. Le résultat est bien présenté sur les afficheurs et sur les LEDs externes.

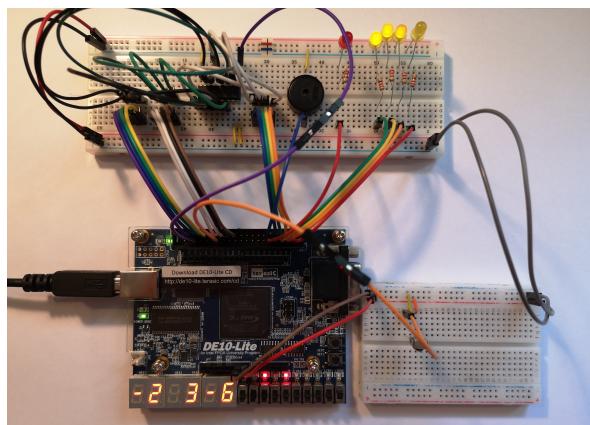


Figure 38 – Multiplication interne signée

[Retour à la section multiplication signée interne.](#)

Ci-dessus un test réalisé sur la multiplication interne signée. La LED 3 de la DE10-Lite est allumée pour montrer à l'utilisateur qu'il réalise une multiplication.

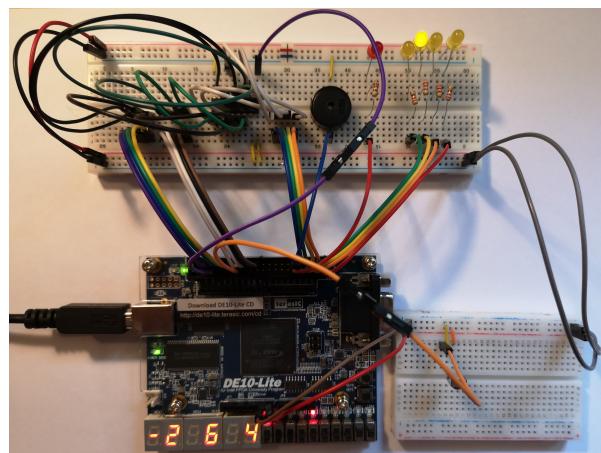


Figure 39 – Addition externe signée

[Retour a la section addition signée externe.](#)

A présent, un exemple l'addition externe signée, donc pour indiquer ces information à l'utilisateur la LED 1 de la carte représentant le mode CLA est allumée, la LED 3 est éteinte car c'est une somme et la LED 5 est allumée car le mode est signé.

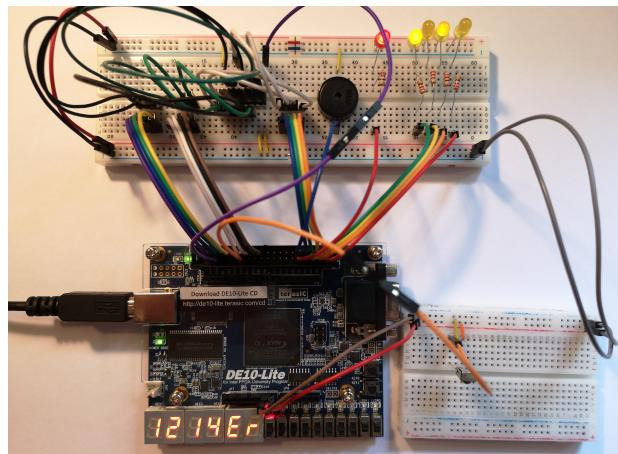


Figure 40 – Overflow après une addition externe non signée

[Retour a la section addition non-signée externe.](#)

Ici un exemple d'une somme externe non signée qui rend un résultat dépassant 4 bits, alors les afficheurs représente ce problème et la LED externe rouge d'overflow s'allume.