

Projet n°3

La Calculatri'ECE



Projet	ING2_ELECPRJ3
Client	ECE Paris.Lyon – Département d'électronique
Chef de projet	Antonio BALTAZAR
Auteurs	Rodolphe TELLIER, Karim-Joseph JABR, Antonio BALTAZAR
Version	[3].[2]
Date de dépôt	03/10, 23h55
Diffusion	restreinte

Table des matières

➤ Avant-propos / présentation

I- Architecture du système	5
- Diagramme structurel du projet	
- Diagramme de branchement	
II- Télécommande et récepteur IR	10
- VS1838B	
- télécommande	
III- Opérateur	12
- FPGA	
- Additionneur 74LS283	
IV- Opérations	14
- Addition	
○ Signé / non-signé	
- Multiplication	
○ Signé / non-signé	
V- Buzzer	18
VI- Affichage	19
- 7-segments	
VII- Gestion du projet	21
- Répartition des tâches / Gestion du temps	
- Outils	
VIII- Résultats	22
- Bilan collectif	

Table des figures :

I- Architecture du système

- **Figure 1.1** : *diagramme structurel*
- **Figure 1.2** : *Diagramme de branchement*
- **Figure 1.3** : *Interface du Pin Planner*
- **Figure 1.4** : *Input du pin planner*
- **Figure 1.5** : *Output du pin planner*
- **Figure 1.6** : *Photo d'ensemble*
- **Figure 1.7** : *Photo du circuit*

II- Télécommande et récepteur

- **Figure 2.1** : *Organigramme réception-conversion opérandes*

III- Opérateur

- **Figure 3.1** : *Structure interne d'un FPGA*
- **Figure 3.2** : *Disposition de la DE10-Lite*
- **Figure 3.3** : *Tableau assignations des opérandes*
- **Figure 3.4** : *Structure du 74LS283*
- **Figure 3.5** : *Tableau d'assignation du résultat*

IV- Opérations

- **Figure 4.1** : *Algorigramme des différents switchs*

V- Buzzer

- **Figure 5.1** : *Organigramme cycle du buzzer*

VI- Affichage

- **Figure 6.1** : *L'afficheur 7-segment*

VII- Gestion du projet

- **Figure 7.1** : *Trombinoscope*
- **Figure 7.2** : *Gestion du temps*

Avant-propos / Présentation :

Dans le cadre de notre projet, nous sommes chargés de créer une calculatrice guidée par une télécommande, tout en utilisant une FPGA ainsi qu'un additionneur. Le but de ce rapport est d'expliquer de manière détaillée et approfondie le fonctionnement de notre Calculatrice.

Grâce au cahier de charges de notre énoncé, et en s'orientant vers la méthodologie du cycle en V, nous nous sommes faits guidés aux conclusions attendues. C'est ainsi que dans cette étude, nous aborderons les différentes étapes permettant l'élaboration d'une calculatrice.

Ce projet est désormais le troisième de notre enseignement d'électronique à l'ECE Paris, nous en sommes ravis !

Quelques définitions :

Quartus :

Logiciel de conception de la logique programmable.

Switch :

Organe de commande qui permet d'ouvrir et de fermer un circuit.

Pin planner :

Fonctionnalité de Quartus permettant d'assigner les variables du code à des composants du circuit en deux catégories : entrée ou sortie

FPGA :

La FPGA (Field Programmable Gate Arrays) donné pour le projet désigne un circuit numérique intégré configurable à la volée.

Additionneur:

Circuit logique permettant de réaliser une addition.

74LS283 :

Additionneur binaire 4 bits.

Buzzer :

Un buzzer est un élément électromécanique produisant du son sous tension.

I- Architecture du système

- Diagramme structurel du projet

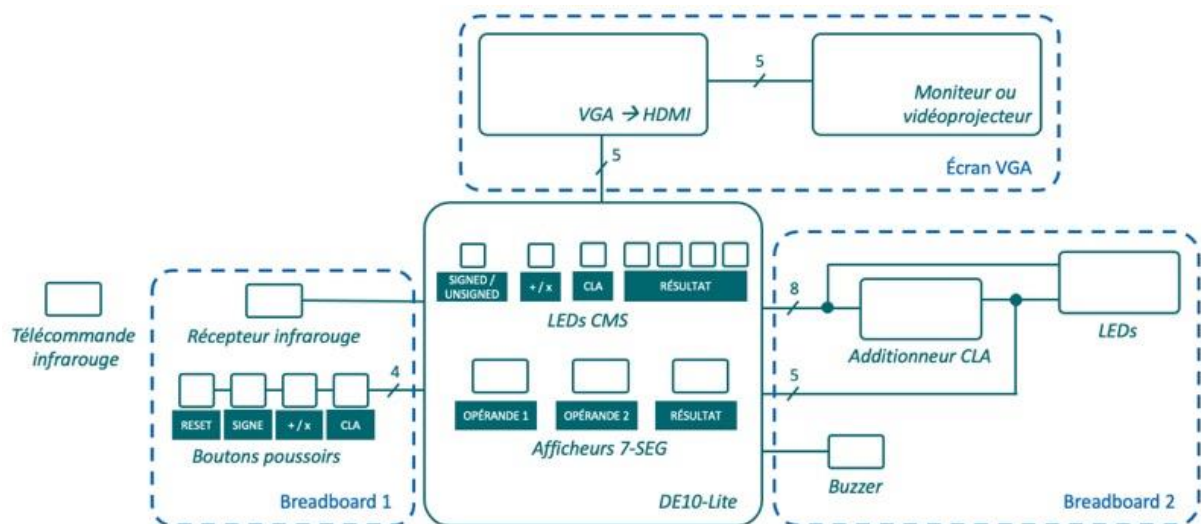


Figure 1.1 : diagramme structurel

Dans ce premier diagramme, nous pouvons identifier 4 principaux blocs permettant de structurer le projet. Un premier bloc interactif avec l'utilisateur, que cela soit par la télécommande ou les boutons poussoirs, un deuxième orienté calcul et affichage pour l'interface FPGA, un troisième identique au deuxième pour l'additionneur, et enfin un dernier pour l'affichage VGA.

Le premier bloc (Breadboard 1), correspond aux différents boutons mis en place afin que l'utilisateur choisisse l'option qu'il souhaite avant résultat, à savoir un reset, le signe des nombres reçus, le type d'opération souhaité ainsi que le type d'environnement choisi pour le calcul (FPGA ou Additionneur). Nous remarquerons également la présence d'un récepteur infrarouge : ce dernier détectera les fréquences émises par la télécommande, nous aurons le temps d'y revenir plus tard.

Pour ce premier bloc, nous avons décidé, après réflexion, de privilégier les switches (présents sur la carte DE10-Lite) à la place des boutons poussoirs sur la breadboard. Pourquoi ce changement ? Tout d'abord, pour une utilisation similaire, un interrupteur sera plus intuitif car plus visuel (distinction entre ouvert et fermé), à contrario du bouton poussoir. De plus, au lancement du projet, nous sommes partis sur l'utilisation des boutons, puisque le diagramme donné pour le projet énonce l'emploi de ces derniers. Cependant, nous avons mis en place une LED pour chacun d'eux afin qu'ils soient plus « parlants ». Très vite, nous nous sommes rendu compte que le circuit devenait trop chargé, et avons donc décidé d'utiliser des switches.

Passons maintenant au deuxième bloc : la carte DE10-Lite. Nous avons gardé la même description ; l’affichage des deux opérandes ainsi que du résultat sur les 6 afficheurs 7-segments (2 pour chaque terme). Quant aux LEDs, elles figureront aux dessus de chaque switch associé. Pour les LEDs affichant le résultat en bits, nous avons décidé de ne pas les utiliser, les afficheurs 7-ségments nous donnant directement le résultat numérique. De plus, les interrupteurs nous donnent directement un aperçu de l’état des switches.

Pour le troisième bloc calcul et affichage sur additionneur, nous avons branché 4 LEDs aux 4 sorties de l’additionneur afin d’afficher le résultat en bits, avant qu’il ne soit retourné à la DE10-Lite.

Pour ce qui est du capteur infra-rouge et du buzzer, nous les avons simplement branchés sur une breadboard, puis reliés à la DE10-Lite afin de recevoir (capteur) ou d’envoyer (buzzer) les informations nécessaires. Tout ceci par l’intermédiaire du pin planner.

- Diagramme de branchement du circuit

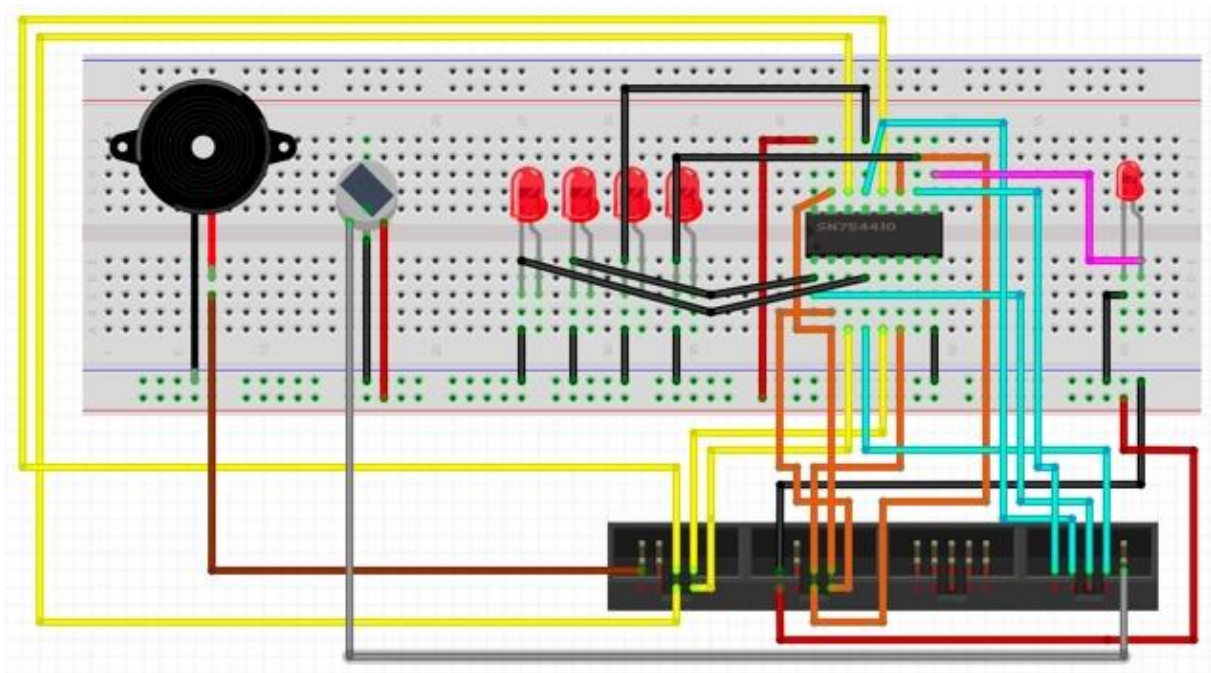


Figure 1.2 : Diagramme de branchement

Voici une représentation effectuée sur **FRITZING**, décrivons les différents composants :

Partons de la gauche :

- Le buzzer (fil marron), relié à la pin **W10** sur notre carte
- Le capteur quant à lui (fil gris) à la pin **AA2**

- Les 4 LEDs reliés aux 4 sortie du résultat de l'additionneurs : $\Sigma 0$, $\Sigma 1$, $\Sigma 2$ et $\Sigma 3$ (noir)
- Une LED pour le signe du résultat sur additionneur (rose)
- L'additionneur : 4 bits du premier nombre (jaune), 4 bits du deuxième nombre (orange) et les 4 bits de sortie du résultat (bleu-ciel)

Il est à noter que ces assignations se font par l'intermédiaire du pin planner, ce dernier permettant d'assigner un composant du code à une pin

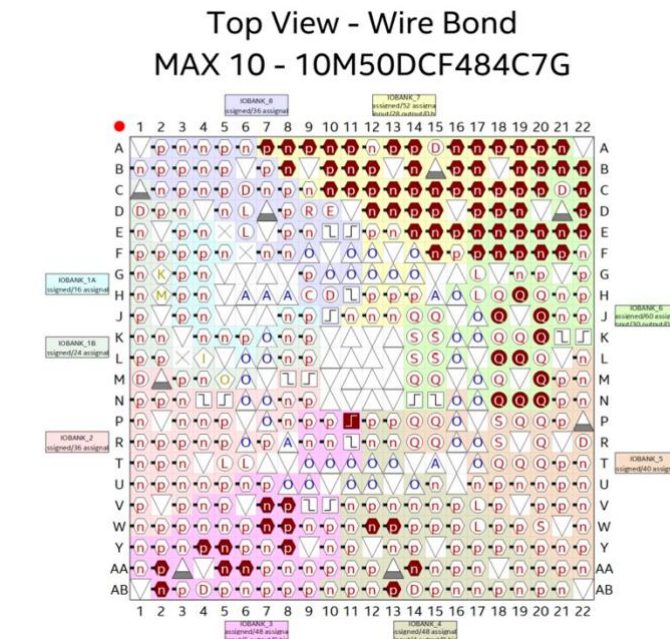


Figure 1.3 : Interface du Pin Planner

Node Name	Direction	Location
b_resetA	Input	PIN_A7
b_resetB	Input	PIN_B8
clock	Input	PIN_P11
data_input	Input	PIN_AA2
int_CLA	Input	PIN_A13
int_operation	Input	PIN_B12
int_saisit	Input	PIN_F15
int_sa...emp[3]	Input	PIN_C12
int_sa...emp[2]	Input	PIN_D12
int_sa...emp[1]	Input	PIN_C11
int_sa...emp[0]	Input	PIN_C10
int_signeA	Input	PIN_B14
int_signeB	Input	PIN_A14
Res_ext[4]	Input	
Res_ext[3]	Input	PIN_AA6
Res_ext[2]	Input	PIN_Y5
Res_ext[1]	Input	PIN_AA5
Res_ext[0]	Input	PIN_Y4

Figure 1.4 : Input du pin planner

OUT A_ext[3]	Output	PIN_W8	3
OUT A_ext[2]	Output	PIN_V8	3
OUT A_ext[1]	Output	PIN_V7	3
OUT A_ext[0]	Output	PIN_W7	3
OUT affich_A[15]	Output	PIN_J20	6
OUT affich_A[14]	Output	PIN_K20	6
OUT affich_A[13]	Output	PIN_L18	6
OUT affich_A[12]	Output	PIN_N18	6
OUT affich_A[11]	Output	PIN_M20	6
OUT affich_A[10]	Output	PIN_N19	6
OUT affich_A[9]	Output	PIN_N20	6
OUT affich_A[8]	Output	PIN_L19	6
OUT affich_A[7]	Output	PIN_F18	6
OUT affich_A[6]	Output	PIN_E20	6
OUT affich_A[5]	Output	PIN_E19	6
OUT affich_A[4]	Output	PIN_J18	6
OUT affich_A[3]	Output	PIN_H19	6
OUT affich_A[2]	Output	PIN_F19	6
OUT affich_A[1]	Output	PIN_F20	6
OUT affich_A[0]	Output	PIN_F17	6
OUT affich_B[15]	Output	PIN_F21	6
OUT affich_B[14]	Output	PIN_E22	6
OUT affich_B[13]	Output	PIN_E21	6
OUT affich_B[12]	Output	PIN_C19	7
OUT affich_B[11]	Output	PIN_C20	6
OUT affich_B[10]	Output	PIN_D19	6
OUT affich_B[9]	Output	PIN_E17	6
OUT affich_B[8]	Output	PIN_D22	6
OUT affich_B[7]	Output	PIN_B20	6
OUT affich_B[6]	Output	PIN_A20	7
OUT affich_B[5]	Output	PIN_B19	7
OUT affich_B[4]	Output	PIN_A21	6
OUT affich_B[3]	Output	PIN_B21	6
OUT affich_B[2]	Output	PIN_C22	6
OUT affich_B[1]	Output	PIN_B22	6
OUT affich_B[0]	Output	PIN_A19	7
OUT affich_Res[15]	Output	PIN_C18	7
OUT affich_Res[14]	Output	PIN_D18	6
OUT affich_Res[13]	Output	PIN_E18	6
OUT affich_Res[12]	Output	PIN_B16	7
OUT affich_Res[11]	Output	PIN_A17	7
OUT affich_Res[10]	Output	PIN_A18	7
OUT affich_Res[9]	Output	PIN_B17	7
OUT affich_Res[8]	Output	PIN_A16	7
OUT affich_Res[7]	Output	PIN_C14	7
OUT affich_Res[6]	Output	PIN_E15	7
OUT affich_Res[5]	Output	PIN_C15	7
OUT affich_Res[4]	Output	PIN_C16	7
OUT affich_Res[3]	Output	PIN_E16	7
OUT affich_Res[2]	Output	PIN_D17	7
OUT affich_Res[1]	Output	PIN_C17	7
OUT affich_Res[0]	Output	PIN_D15	7

OUT B_ext[3]	Output	PIN_AA14
OUT B_ext[2]	Output	PIN_AB13
OUT B_ext[1]	Output	PIN_W12
OUT B_ext[0]	Output	PIN_W13
OUT buzzer	Output	PIN_AB2
OUT led[10]	Output	PIN_Y8
OUT led[9]	Output	PIN_B11
OUT led[8]	Output	PIN_A11
OUT led[7]	Output	PIN_D14
OUT led[6]	Output	PIN_E14
OUT led[5]	Output	PIN_C13
OUT led[4]	Output	PIN_D13
OUT led[3]	Output	PIN_B10
OUT led[2]	Output	PIN_A10
OUT led[1]	Output	PIN_A9
OUT led[0]	Output	PIN_A8

Figure 1.5 : *Output du pin planner*

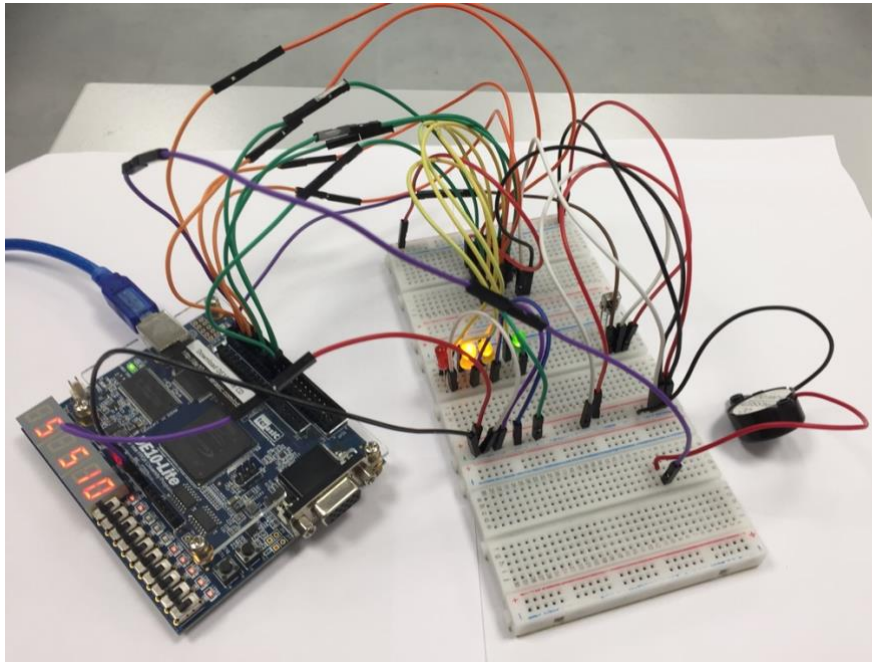


Figure 1.6 : Photo d'ensemble

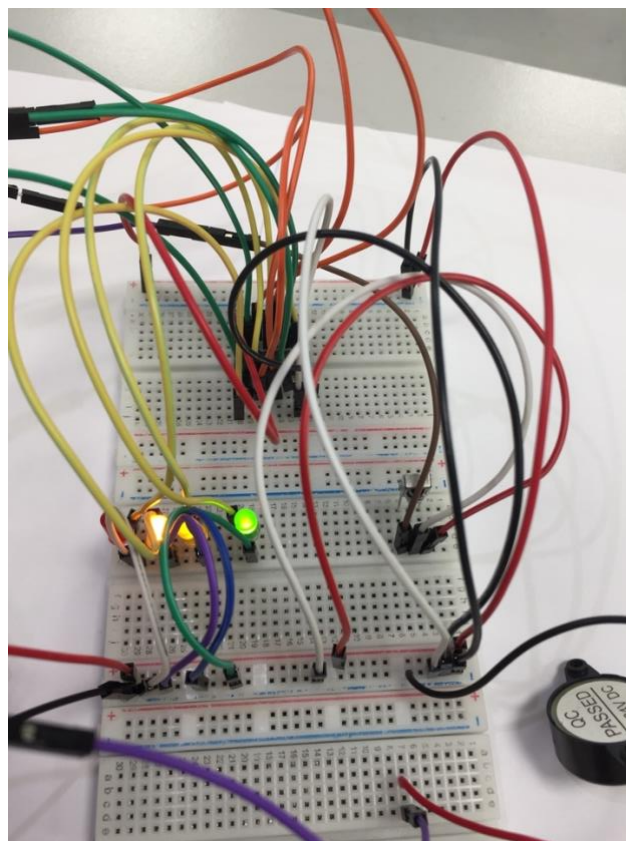


Figure 1.7 : Photo du circuit

II- Télécommande et récepteur infrarouge

- La télécommande



La télécommande de notre kit que nous devons utiliser émet des ondes infra-rouge, et transmet donc un signal de rayonnements infra-rouge. Ces derniers sont hors du champ de la lumière visible.

Lorsque l'utilisateur appuiera sur une touche, un signal sous forme de bits est envoyé en infrarouge. En effet chaque touche émet un signal différent sur 32 bits, mais uniquement les bits 23 à 16 nous intéressent : ils correspondent à la valeur du numéro (ex : '00110001' pour le 2). De plus, nous avons rajouter les numéros 10, 11, 12, 13, 14 et 15 respectivement sur les touches CH-, CH, CH+, PREV, NEXT et play/pause.

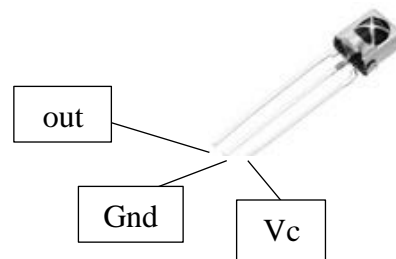
Cette partie du code nous a été envoyée sur notre campus numérique.

Dans notre code, nous réceptionnons donc ces signaux grâce au capteur, que nous assignons donc au nombre correspondant sous forme de bits.

Ex pour la touche 0 :

```
case comm is
| when "00101101" => NA <= X"0";
```

- Le capteur VS1838B



Comme vu précédemment, le récepteur infra-rouge va réceptionner les données de la télécommande. Pour que le signal passe, la distance entre les deux ne doit pas dépasser 5 mètres.

Le capteur infrarouge est un détecteur mesurant le rayonnement infrarouge de la zone qu'il couvre. Si l'utilisateur vient à changer de touche, le capteur infrarouge réagit instantanément et détectera la nouvelle variation du rayonnement.

Une fois le signal reçu, la valeur est directement envoyée à la Carte via le pin planier.

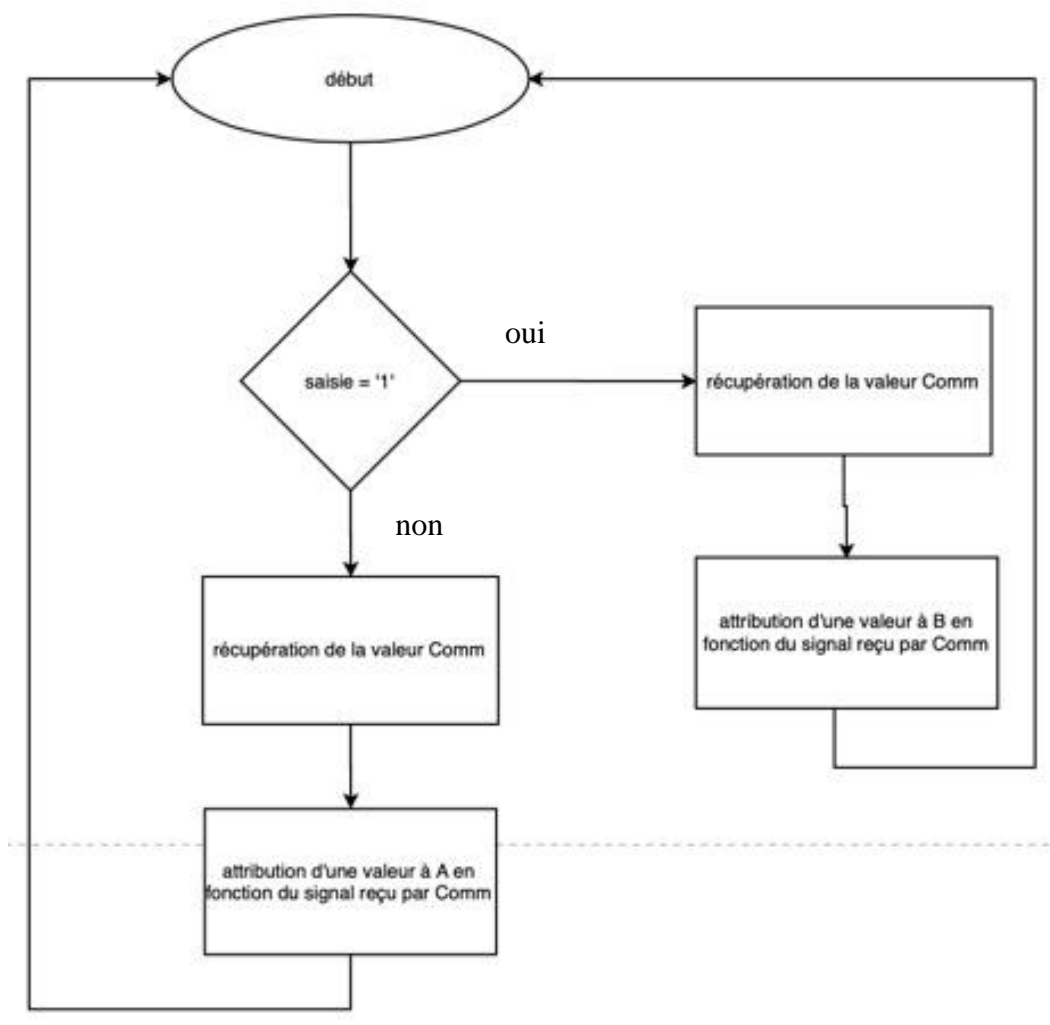


Figure 2.1 : *Organigramme réception-conversion opérandes*

Dans ce diagramme simplifiant ce que nous venons de voir, nous sommes dans le cas où l'utilisateur a déjà appuyé sur une touche de la télécommande, disons la touche 6.

Nous allons d'abord regarder si le switch, nommé « saisi », nous permet de rentrer la valeur du premier nombre (A) ou du deuxième (B). Nous avons décidé d'initialiser l'interrupteur « saisi » à A lorsqu'il est fermé.

Ensuite, le capteur va donc recevoir ce signal de la télécommande, et l'envoyer à la carte ; il s'agit de l'entrée « Comm ». Il ne nous reste plus qu'à lire les 8 bits du signal qui nous intéressent afin de les assigner à un nombre sur 8 bits.

III- Opérateurs

- La FPGA



Qu'est-ce qu'une FPGA ?

La FPGA (Field Programmable Gate Arrays) donné pour le projet désigne un circuit numérique intégré configurable à la volée. Composée d'un réseau de cellules logiques programmables (PLD), chaque cellule est capable de réaliser une fonction, choisie parmi plusieurs possibles. Les interconnexions sont également programmables.

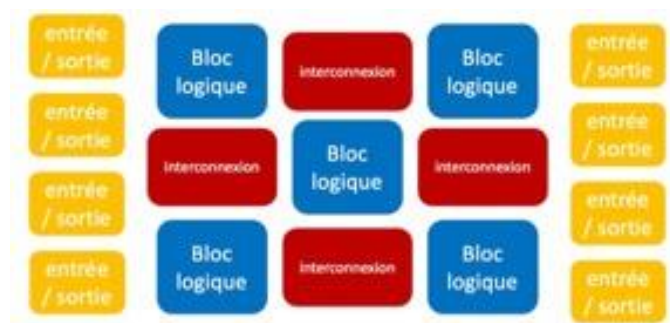


Figure 3.1 : *Structure interne d'une FPGA*

Au final, une FPGA est le juste milieu entre flexibilité et efficacité, le tout avec une consommation raisonnable.

Où se trouve la FPGA sur notre carte DE10-Lite ?

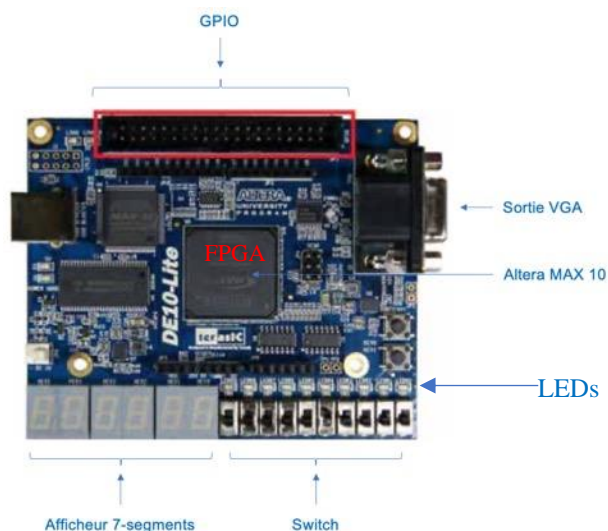


Figure 3.2 : *Disposition de la DE10-Lite*

Pour le calcul sur FPGA, une fois que le signal « Comm » reçu par le capteur est converti en bits pour A et B (A et B étant définie comme des signaux, ex : SIGNAL A,B : STD_LOGIC_VECTOR (7 downto 0)), il ne reste plus qu'à réaliser l'opération souhaitée, décidée par l'utilisateur grâce aux switchs. Nous verrons comment faire cela au prochain point.

- L' additionneur 74LS283



Qu'est-ce qu'un additionneur ?

Un additionneur est un circuit logique permettant de réaliser une addition.

Pour le calcul sur additionneur, il faudra lui envoyer A et B sous forme de bits (A et B seront donc ici des sorties, ex : A,B : out STD_LOGIC_VECTOR (3 downto 0)) et donc assigner 8 pins de sortie. Voici un tableau les représentant :

A1	◆ A[0]	Unknown	PIN_W7
A2	◆ A[1]	Unknown	PIN_V7
A3	◆ A[2]	Unknown	PIN_V8
A4	◆ A[3]	Unknown	PIN_W8
B1	◆ B[0]	Unknown	PIN_W13
B2	◆ B[1]	Unknown	PIN_W12
B3	◆ B[2]	Unknown	PIN_AB13
B4	◆ B[3]	Unknown	PIN_AA14

Figure 3.3 : *Tableau assignation des opérandes*

- A[0] correspond au premier bit de A, relié à la PIN W7
A[1] correspond au deuxième bit de A, relié à la PIN V7
Etc..
- B[0] correspond au premier bit de B, relié à la PIN W13
Etc..
- A quoi donc correspond l'encadré gris ? Il s'agit des PINs d'entrée de l'additionneur (cf : **Figure 3.4**), il nous faudra donc relier le premier bit de A à l'entrée A1 de l'additionneur, correspondant au premier bit de l'opérande 1. Même chose pour B.

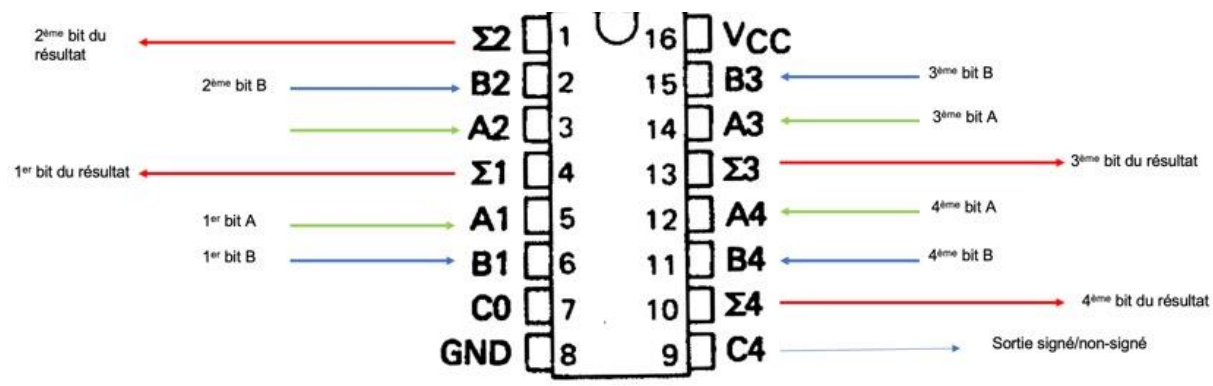


Figure 3.4 : Entrées/sorties du 74LS283

Les quatre bits du résultat sortant de l'additionneur après opération seront directement retournés à la carte par quatre PINs que voici :

Res_ext[3]	Input	PIN_AA6
Res_ext[2]	Input	PIN_Y5
Res_ext[1]	Input	PIN_AA5
Res_ext[0]	Input	PIN_Y4

Figure 3.5 : Tableau d'assignation du résultat

Nous verrons plus tard comment récupérer ces informations.

IV- Opérations

Avant de pouvoir calculer le résultat, il nous faut savoir si nous sommes sur FPGA ou additionneur, si l'opération est + ou *, si les nombres A et B sont signés ou non.

Pour ce faire, et comme vu précédemment, nous avons recouru à l'utilisation de switches.

Pour mieux comprendre, voici un schéma des différents interrupteurs permettant d'arriver à de nombreuses issues possibles :

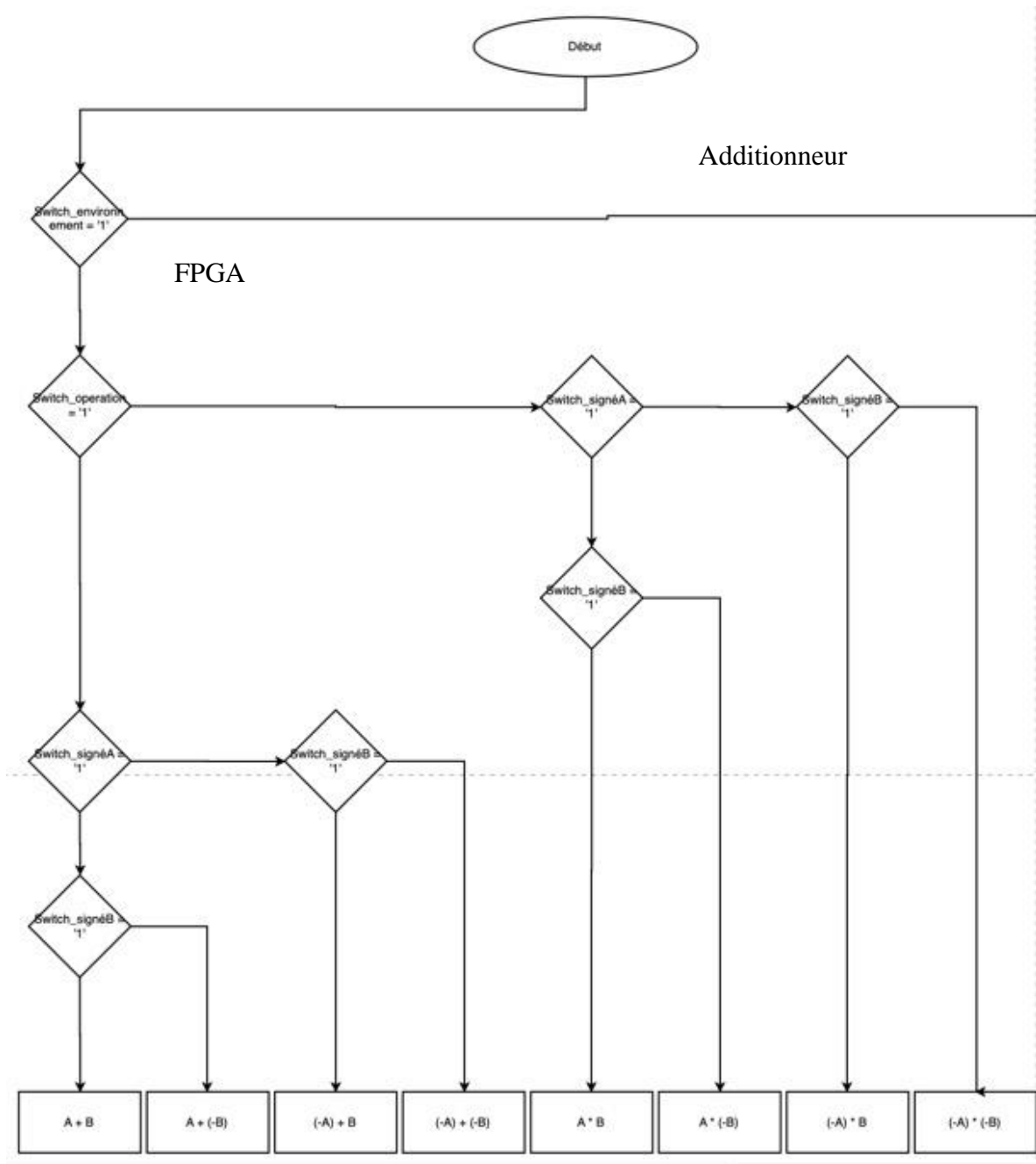


Figure 4.1 : Algorithme des différents switches

Ici, nous pouvons voir les différents « if » des différents chemins possibles. Notons que nous voyons ici uniquement les actions sur FPGA. Sur additionneur, le chemin est identique. Nous aurions donc au total 16 chemins possibles pour l'utilisateur. Cependant, tous ces chemins sont trop nombreux, et nous avons essayé de réduire les cas possibles au sein du code. Par exemple, la multiplication simple et doublement signé (même résultat) ou encore l'addition simple et doublement signé (même valeur absolue).

Après ce cheminement, nous connaissons :

- l'environnement dans lequel l'utilisateur souhaite calculer (FPGA ou additionneur)
- le type d'opération souhaité (+ ou *)
- si A et B sont signés ou non

Nous allons alors pouvoir commencer le cheminement du calcul dans notre code.

- Addition

○ Non-signé

Sur FPGA :

En premier lieu, afin d'utiliser les outils mathématiques nécessaires, nous avons intégré à notre code la bibliothèque : 'IEEE.numeric_std.all'

Lorsque l'on saisit l'opérateur +, ce dernier correspond à une fonction dans la bibliothèque qui permet d'additionner deux std_logic_vector et d'écrire le résultat dans un autre std_logic_vector.

Exemple : Resultat = A + B;

Sur additionneur :

L'addition se fait par l'entrée de deux chiffres binaires codés sur quatre bits chacun et par la sortie du résultat codé sur 4 bits. L'addition s'effectue grâce aux porte logiques à l'intérieur de l'additionneur. Nous recevons uniquement les 4 bits du résultat en « in » dans la carte.

○ Signé

Sur FPGA :

Nous avons intégré à notre code la bibliothèque : 'IEEE.STD_LOGIC_SIGNED'. Cette dernière nous permet de réaliser des opérations de nombres négatifs, en utilisant la syntaxe suivante :

resultat <= signed(A) + signed(B)

- Multiplication

○ Non-signé

Sur FPGA :

Lorsque l'on saisit l'opérateur *, ce dernier correspond à une fonction dans la bibliothèque qui permet de multiplier deux `std_logic_vector` et d'écrire le résultat dans un autre `std_logic_vector`.

Exemple : $\text{Resultat} = A * B$

- Signé

Sur FPGA :

- Si A et B sont signés, alors le résultat équivaut à $A*B$, donc même cas qu'A et B non-signés.
- Si A signé et B non-signé, alors résultat négatif, sinon faire $A*B$
- Vice versa pour A non-signé et B signé

V- Buzzer

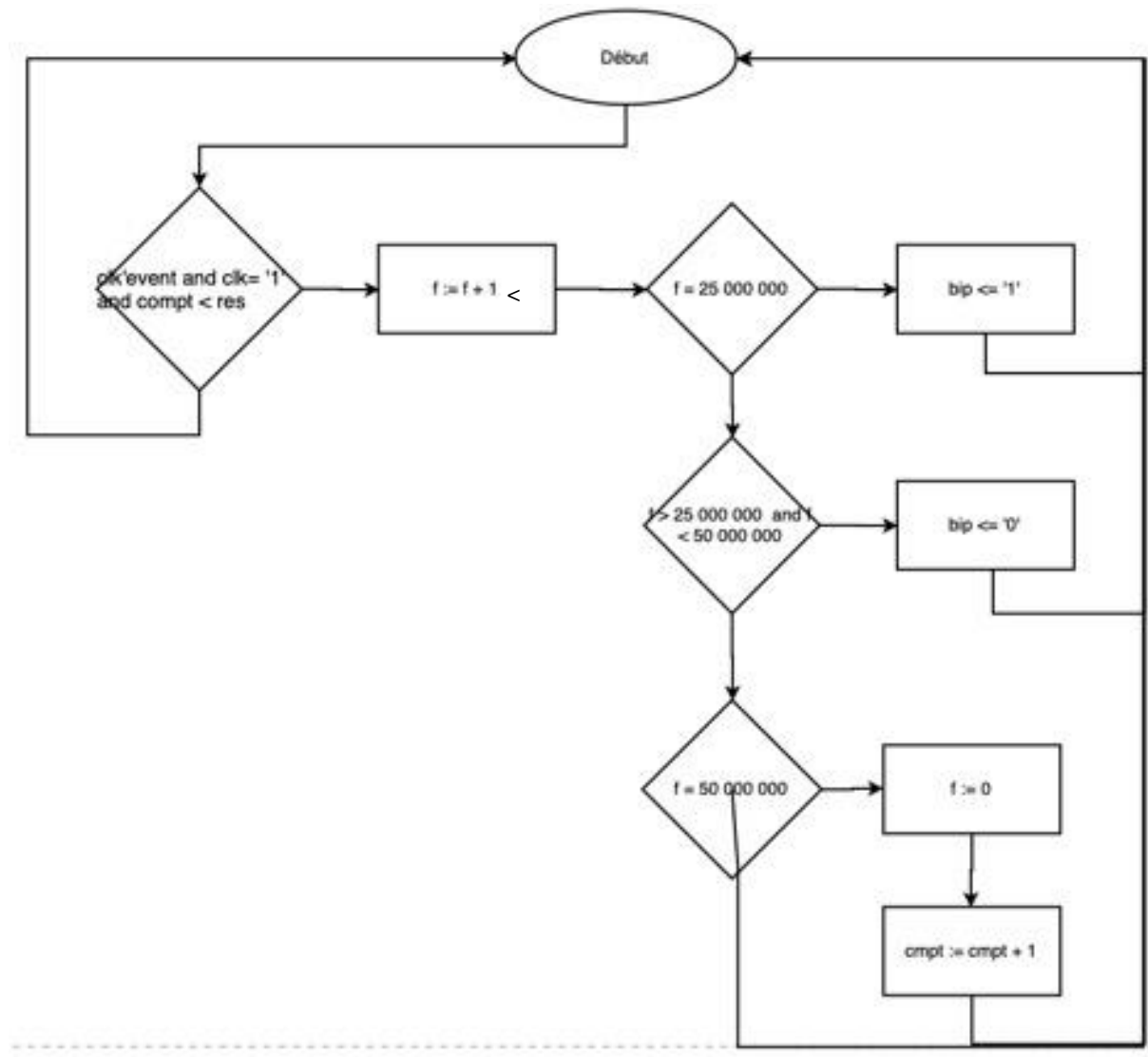


Figure 5.1 : *Organigramme du cycle du buzzer*

Dans cet organigramme, nous apercevons 4 if, que représentent-ils ?

Il faut tout d'abord savoir qu'une variable compteur a été initialisé à 0, nous comprendrons ceci plus tard.

Donc ce premier if correspond au coup d'horloge. En effet, nous allons faire biper notre buzzer à chaque front montant de la clock. Ce premier if s'activera donc à chaque front montant, mais également lorsque notre compteur sera strictement inférieur à au résultat de l'opération (res). Pour l'instant, lors du premier front montant, le compteur vaut 0, donc nous remplissons les critères.

Passons au deuxième if, nous voyons un petit 'f' apparaître : il s'agit d'une variable caractérisant les fronts montants de la clock. Il est à noter que 25 millions de fronts montants représentent une demi-seconde. Donc après une demi-seconde, il nous sera possible de rentrer dans ce second if, dans lequel nous incrémenterons le petit f et ferons bipier le buzzer.

Le buzzer va donc bipier tant que f sera < 25 millions.

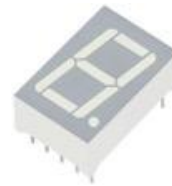
Lorsqu'il passera entre 25 millions et 50 millions, le buzzer ne bipie plus, et exactement 50 millions, incrémentation du compteur et réinitialisation de f.

Donc le compteur va s'incrémenter d'un en un, avec, à chaque fois, une période d'une demi-seconde où le buzzer bipie et une période d'une demi-seconde où le buzzer ne bipie pas. Un cycle dure une seconde. Cela tant que le compteur est inférieur ou égale à res (résultat).

Il ne restera plus qu'à réinitialiser le compteur à 0 à la fin du grand if.

VI- Affichage

- 7-segments



Un afficheur 7-segments, comme son nom l'indique, est formé de 7 LEDs en forme de segment, et d'une LED en forme de point. Comme un opérande ainsi que le résultat peuvent dépasser 9, il faudra donc deux afficheurs 7-segments pour chaque terme. Nous avons donc décidé de n'afficher que les deux opérandes ainsi que le résultat.

Pour afficher un certain chiffre, on doit le faire correspondre avec les différentes configurations d'allumage, comme montre l'exemple ci-dessous :

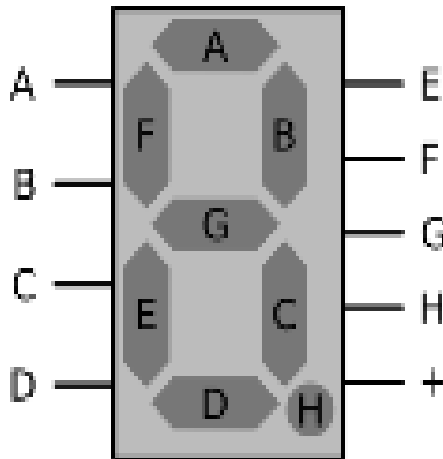


Figure 6.1 : *L'afficheur 7-segments*

Pour afficher le chiffre 3, on aura besoin d'allumer les LEDs A, B, C, D et G.




Dans notre code VHDL, nous avons pour sortie un STD_LOGIC_VECTOR sur seize bits (d'A à G deux fois ainsi que les deux points) qui indique quelle LED devrait être allumée pour afficher le résultat. Le premier bit de gauche correspond à la LED A, le deuxième correspond à la LED B, le huitième bit correspond à la LED du premier point et ainsi de suite... Pour éteindre une certaine LED on utilise le bit 1, et pour l'allumer, le bit 0.

Exemple :

- Pour afficher le nombre 15 on aura besoin de la sortie suivante : '1001111101001001'.
- Et, pour un résultat supérieur à 15 l'écran affichera Er qui signifie ERREUR et qui a pour sortie : '0110000111010101'.

VII- Gestion du projet

- Répartition des tâches / Gestion du temps

	Antonio	Rodolphe	Karim
Affichage 7-segments	X	X	
Addition (signé / non signé) interne	X		
Addition non signé externe	X		
Buzzer	X	X	
IR	X		
Multiplication (signé non signé) interne	X		
Montage du circuit	X	X	
Rapport		X	X

Figure 7.1 : Trombinoscope

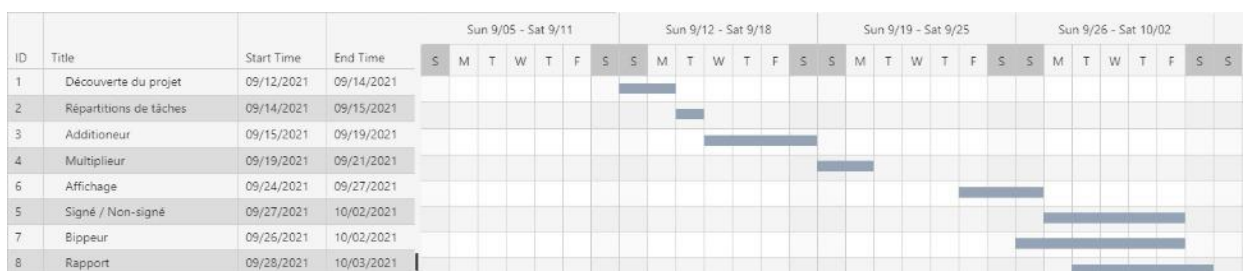


Figure 7.2 : Répartition des tâches

- Outils



VIII- Résultats / Bilan

- Bilan collectif

Ce projet a été relativement bien apprécié par toute l'équipe, en effet il nous a mis à rude épreuve et nous a demandé beaucoup de temps et d'investissement. Nous sommes d'accord pour dire que la plus grande difficulté éprouvée est la méthode de résolution de problèmes appliquée, la programmation VHDL est bien différente de la programmation informatique comme on emploierait dans du C ou du C++.

Il nous a donc été bénéfique car il nous a apporté une nouvelle vision et de nouvelles manières de réfléchir à résoudre un problème.

