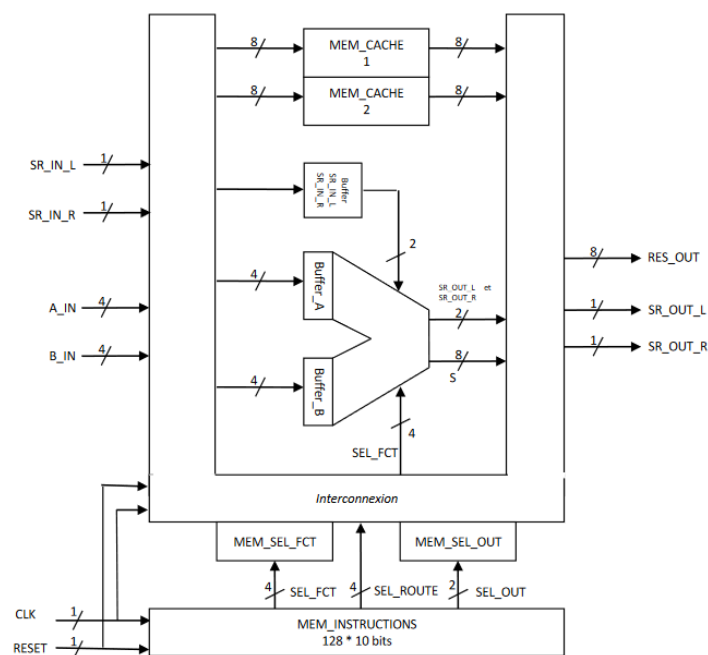


## Rapport de projet – Conception de circuit numérique

### Réalisation d'un cœur de microcontrôleur



**Auteurs :** Guillaume Dumas  
Jérémy Grelaud



## Sommaire

---

1	Introduction .....	3
2	Conception du cœur de l’UAL .....	5
3	Conceptions des mémoires buffers .....	7
3.1	Buffers du cœur de processeur .....	7
3.2	Buffers de commandes (SEL_FCT et SEL_OUT) .....	8
3.3	Mémoire d’instruction .....	9
4	Interconnexion et routage des données .....	10
4.1	Sel_out .....	10
4.2	Sel_route .....	10
5	Intégration du cœur de microcontrôleur sur la carte ARTY .....	11
6	Conception des trois fonctions spécifiques .....	12
6.1	Res_out_1 : une multiplication, c’est tout ? .....	13
6.2	Res_out_2 : l’addition, le xor et le not .....	14
6.3	Res_out_3 : la multiplication et l’addition des décalages ! .....	14
7	Conclusion.....	15

# 1 Introduction

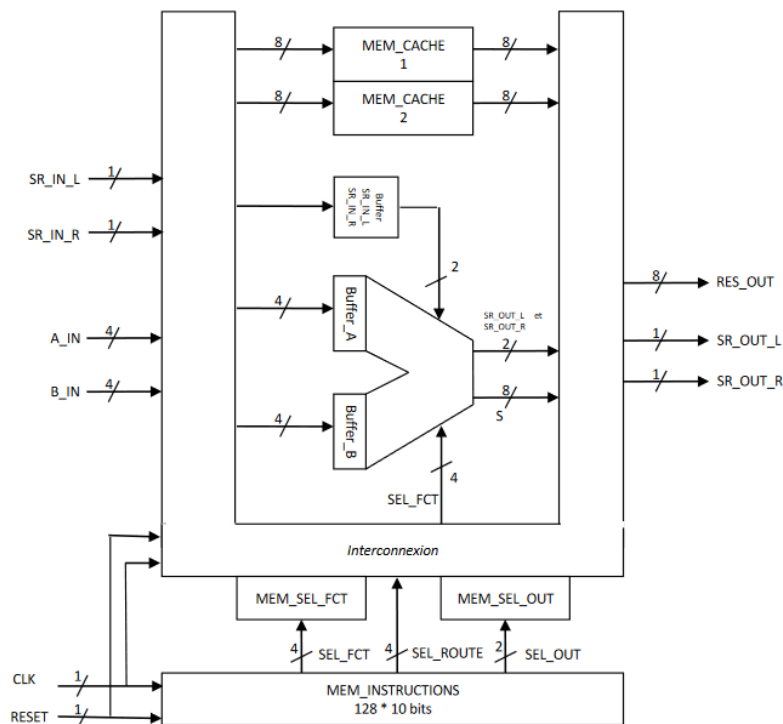
« Les ordinateurs, plus on s'en sert moins, et moins ça a de chance de mal marcher. » Mais comment un ordinateur fonctionne-t-il ? Comment sont orientées les données et comment sont effectuées les opérations complexes, qui demande plusieurs étapes ?


Nous avons pu aborder ces questions grâce au VHDL, un langage de description matériel permettant de représenter le comportement et l'architecture d'un système électronique numérique.

Lors du module d'Information numérique en L1, nous avons déjà pu mieux appréhender les fonctionnements de l'addition et de la multiplication. Nous avons réalisé ces deux opérations grâce à des portes logiques. Ce projet nous a particulièrement marqué, car c'était là l'occasion d'enfin comprendre comment un ordinateur fait pour effectuer des calculs. Si  $9+2$  nous est évident, et si  $1001 + 0010$  nous est facile, comment un ordinateur, qui ne fonctionne qu'avec des signaux électriques, s'en sort-il ? Ces questions avaient trouvé leurs réponses en L1.

Faire des opérations, c'est bien, mais ce n'est pas suffisant. Il faut que les informations puissent transiter dans le microcontrôleur, qu'elles puissent y entrer et en sortir. Comment piloter ces informations ? Comment choisir les opérations que l'UAL doit effectuer ? Et, en sachant que la plupart des opérations complexes ne sont que la combinaison d'opérations simples, comment organiser une suite d'instruction pour que la sortie soit celle qu'on attend ?

Pour comprendre cela, nous allons réaliser un cœur de processeur, dont voici l'architecture.





Nous verrons comment organiser de A à Z ce système électronique numérique de tel sorte à pouvoir réaliser n'importe quelle fonction (assez élémentaire tout de même). Nous élaborerons grâce au VHDL ce cœur de microcontrôleur. L'étape finale est d'intégrer le code sur une carte de développement ARTY. Nous pourrons ainsi voir et interagir physiquement avec notre cœur de microcontrôleur.

## 2 Conception du cœur de l'UAL

Ce cœur devait réaliser 16 fonctions différentes pilotées par la valeur de SEL\_FCT sur 4 bits.

Ces fonctions sont diverses comme la première, ne réaliser aucune opération, ou alors mettre en sortie la valeur de l'entrée A ou B ou le conjugué de A ou B.

Les 16 comportements de notre cœur ont été définis dans le design, le processus contient un « case » associant les instructions nécessaires à chacun des 16 cas. Nous avons ensuite réalisé un test\_bench pour vérifier la cohérence de nos résultats. Afin de tester un certain nombre de valeurs pour chacune des fonctions, nous avons utilisé, comme vous nous l'aviez montré lors de la première séance, un compteur « my\_cpt\_sim » sur 15 bits que l'on initialise à une certaine valeur arbitraire puis que l'on incrémente. Les 8 bits de poids faible de ce compteur correspondent aux entrées A et B. Puis les 2 retenues d'entrées et enfin les 4 bits de notre SEL\_FCT. Il n'est pas possible sur EDA de simuler toutes les valeurs jusqu'à la 16<sup>e</sup> fonction car au bout de 5000 lignes la simulation s'arrête. Il faut donc initialiser et relancer plusieurs fois la simulation pour observer le comportement des 16 fonctions. On a ensuite vérifié que toutes les fonctions étaient justes :

SEL\_FCT = 5 en base 10 : '0101' en binaire donc c'est la fonction  $S = A \text{ and } B$  :

```
testbench.vhd:70:8:@1107ns:(report note): SEL_FCT=5 ||| S=0 | A=2 | B=5 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1108ns:(report note): SEL_FCT=5 ||| S=1 | A=3 | B=5 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1109ns:(report note): SEL_FCT=5 ||| S=4 | A=4 | B=5 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1110ns:(report note): SEL_FCT=5 ||| S=5 | A=5 | B=5 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1111ns:(report note): SEL_FCT=5 ||| S=4 | A=6 | B=5 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1112ns:(report note): SEL_FCT=5 ||| S=5 | A=7 | B=5 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
```

Cela fonctionne. Exemple pour A=4 '0100' en binaire et B=5 '0101' en binaire, on obtient bien '0100'=4 dans S.

SEL\_FCT = 7 en base 10 : '0111' en binaire donc c'est la fonction  $S = A \text{ xor } B$  :

```
testbench.vhd:70:8:@3360ns:(report note): SEL_FCT=7 ||| S=14 | A=-1 | B=1 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@3361ns:(report note): SEL_FCT=7 ||| S=2 | A=0 | B=2 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@3362ns:(report note): SEL_FCT=7 ||| S=3 | A=1 | B=2 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@3363ns:(report note): SEL_FCT=7 ||| S=0 | A=2 | B=2 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@3364ns:(report note): SEL_FCT=7 ||| S=1 | A=3 | B=2 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@3365ns:(report note): SEL_FCT=7 ||| S=6 | A=4 | B=2 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
```

Cela fonctionne. Exemple pour A = 4, '0100' en binaire et B = 2 '0010' en binaire. Le XOR teste la parité.

### Table de vérité du XOR

<b>00</b>	0
<b>01</b>	1
<b>10</b>	1
<b>11</b>	0

On obtient donc  $A \text{ xor } B = '0110' = 6$ .

$SEL\_FCT = 10$  en base 10 : '1010' en binaire donc c'est la fonction décalage à droite de B :

```
testbench.vhd:70:8:@1360ns:(report note): SEL_FCT=10 ||| S=2 | A=-1 | B=4 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1361ns:(report note): SEL_FCT=10 ||| S=2 | A=0 | B=5 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=1 | SR_OUT_L=0
```

Cela fonctionne, le bit de poids faible de B est désormais dans la retenue de sortie  $SR\_OUT\_R$  et le bit de poids fort est désormais la valeur de  $SR\_IN\_L$ .

$SEL\_FCT = 13$  en base 10 : '1101' en binaire donc c'est la fonction d'addition binaire sans retenue d'entrée :

```
testbench.vhd:70:8:@4126ns:(report note): SEL_FCT=13 ||| S=-2 | A=-3 | B=1 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@4127ns:(report note): SEL_FCT=13 ||| S=-1 | A=-2 | B=1 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@4128ns:(report note): SEL_FCT=13 ||| S=0 | A=-1 | B=1 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@4129ns:(report note): SEL_FCT=13 ||| S=2 | A=0 | B=2 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@4130ns:(report note): SEL_FCT=13 ||| S=3 | A=1 | B=2 | SR_IN_R=0 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
```

On voit bien que cela fonctionne.

$SEL\_FCT = 14$  en base 10 : '1110' en binaire donc c'est la fonction de soustraction binaire ( $A-B$ ) :

```
testbench.vhd:70:8:@1401ns:(report note): SEL_FCT=14 ||| S=-15 | A=-8 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1402ns:(report note): SEL_FCT=14 ||| S=-14 | A=-7 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1403ns:(report note): SEL_FCT=14 ||| S=-13 | A=-6 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1404ns:(report note): SEL_FCT=14 ||| S=-12 | A=-5 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1405ns:(report note): SEL_FCT=14 ||| S=-11 | A=-4 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1406ns:(report note): SEL_FCT=14 ||| S=-10 | A=-3 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1407ns:(report note): SEL_FCT=14 ||| S=-9 | A=-2 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1408ns:(report note): SEL_FCT=14 ||| S=-8 | A=-1 | B=7 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1409ns:(report note): SEL_FCT=14 ||| S=8 | A=0 | B=-8 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1410ns:(report note): SEL_FCT=14 ||| S=9 | A=1 | B=-8 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1411ns:(report note): SEL_FCT=14 ||| S=10 | A=2 | B=-8 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1412ns:(report note): SEL_FCT=14 ||| S=11 | A=3 | B=-8 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1413ns:(report note): SEL_FCT=14 ||| S=12 | A=4 | B=-8 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@1414ns:(report note): SEL_FCT=14 ||| S=13 | A=5 | B=-8 | SR_IN_R=1 | SR_IN_L=0 | SR_OUT_R=0 | SR_OUT_L=0
```



SEL\_FCT = 15 en base 10 : '1111' en binaire donc c'est la fonction de multiplication :

```
testbench.vhd:70:8:@706ns:(report note): SEL_FCT=15 ||| S=-4 | A=1 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@707ns:(report note): SEL_FCT=15 ||| S=-8 | A=2 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@708ns:(report note): SEL_FCT=15 ||| S=-12 | A=3 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@709ns:(report note): SEL_FCT=15 ||| S=-16 | A=4 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@710ns:(report note): SEL_FCT=15 ||| S=-20 | A=5 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@711ns:(report note): SEL_FCT=15 ||| S=-24 | A=6 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@712ns:(report note): SEL_FCT=15 ||| S=-28 | A=7 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@713ns:(report note): SEL_FCT=15 ||| S=32 | A=-8 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
testbench.vhd:70:8:@714ns:(report note): SEL_FCT=15 ||| S=28 | A=-7 | B=-4 | SR_IN_R=0 | SR_IN_L=1 | SR_OUT_R=0 | SR_OUT_L=0
```

On voit bien au vu des résultats que les fonctions fonctionnent correctement.

## 3 Conceptions des mémoires buffers

### 3.1 Buffers du cœur de processeur

La deuxième étape, tout aussi importante, du projet consistait à réaliser les mémoires et les buffers. Les mémoires permettent de stocker les valeurs utiles à l'exécution de nos fonctions dans le cœur de processeur. Il y a :

- Les buffers A et B permettant de stocker temporairement les valeurs de A et B avant qu'elles soient traitées par le cœur de processeur.
- Les buffer SR\_IN\_L et SR\_IN\_R permettent, selon le même principe, de stocker les valeurs des retenues d'entrées.
- Les mémoires mem1 et mem2 permettent de garder en mémoire des valeurs intermédiaires. Ici, elles ont l'avantage d'être sur 8 bits, ce qui permet de stocker le résultat complet d'une multiplication.

Ces six sous-composants fonctionnent selon le même principe. Si le reset vaut 1, alors toutes les sorties valent 0. Sinon, à chaque front montant de l'horloge, et quand la valeur du Cheap Enable est à 1, alors la sortie du buffer (ou de la mémoire) prend la valeur de l'entrée de ce dernier.

```
testbench.vhd:98:9:@100us:(report note): reset = '1' ||| Buff_A_in = 3 | Buff_A_out = 0
testbench.vhd:105:9:@200us:(report note): reset = '0' ||| Buff_A_in = 3 | Buff_A_out = 3
testbench.vhd:112:9:@300us:(report note): reset = '0' ||| Buff_SR_IN_L_in = '1' | Buff_SR_IN_L_out = '1'
testbench.vhd:119:9:@400us:(report note): reset = '0' ||| Buff_SR_IN_L_in = '0' | Buff_SR_IN_L_out = '1'
testbench.vhd:127:9:@500us:(report note): reset = '0' ||| mem_1_in = 243 | mem_1_out = 243
testbench.vhd:129:9:@500us:(report note): Test ok (no assert ...)
```

On voit que quand reset vaut 1, la sortie du buffer 1 vaut 0, bien que son entrée vaille 3 (ligne 1). Quand reset vaut 0, la sortie prend bien la valeur de l'entrée (ligne 2, 3 et 5). Ligne 4, le Cheap Enable est désactivé ( $CE = 0$ ), c'est pour cela que bien que la valeur de l'entrée vaille 0, la sortie vaut 1.

### 3.2 Buffers de commandes (SEL\_FCT et SEL\_OUT)

Ces buffers servent, sur le même principe que les buffers précédents, à stocker temporairement les valeurs de SEL\_FCT et de SEL\_OUT :

- SEL\_FCT est un vecteur de 4 bits, permettant de sélectionner l'une des 16 fonctions du cœur de processeur, en suivant ce tableau de correspondance.

SEL_FCT[3]	SEL_FCT[2]	SEL_FCT[1]	SEL_FCT[0]	Significations
0	0	0	0	nop (no operation) $S = 0$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
0	0	0	1	$S = A$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
0	0	1	0	$S = B$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
0	0	1	1	$S = \text{not } A$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
0	1	0	0	$S = \text{not } B$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
0	1	0	1	$S = A \text{ and } B$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
0	1	1	0	$S = A \text{ or } B$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
0	1	1	1	$S = A \text{ xor } B$   $SR\_OUT\_L = 0$ et $SR\_OUT\_R = 0$
1	0	0	0	$S = \text{Déc. droite } A \text{ sur 4 bits (avec } SR\_IN\_L)   SR\_IN\_L \text{ pour le bit entrant et } SR\_OUT\_R \text{ pour le bit sortant}$
1	0	0	1	$S = \text{Déc. gauche } A \text{ sur 4 bits (avec } SR\_IN\_R)   SR\_IN\_R \text{ pour le bit entrant et } SR\_OUT\_L \text{ pour le bit sortant}$
1	0	1	0	$S = \text{Déc. droite } B \text{ sur 4 bits (avec } SR\_IN\_L)   SR\_IN\_L \text{ pour le bit entrant et } SR\_OUT\_R \text{ pour le bit sortant}$
1	0	1	1	$S = \text{Déc. gauche } B \text{ sur 4 bits (avec } SR\_IN\_R)   SR\_IN\_R \text{ pour le bit entrant et } SR\_OUT\_L \text{ pour le bit sortant}$
1	1	0	0	$S = A + B$ addition binaire avec $SR\_IN\_R$ comme retenue d'entrée
1	1	0	1	$S = A + B$ addition binaire sans retenue d'entrée
1	1	1	0	$S = A - B$ soustraction binaire
1	1	1	1	$S = A * B$ multiplication binaire

- SEL\_OUT est un vecteur de 2 bits permettant de décider vers où orienter la sortie du cœur de processeur, en suivant ce tableau de correspondance.

SEL_OUT[1]	SEL_OUT[0]	Significations
0	0	Aucune sortie : $RES\_OUT = 0$
0	1	$RES\_OUT = MEM\_CACHE\_1$
1	0	$RES\_OUT = MEM\_CACHE\_2$
1	1	$RES\_OUT = S$

```
testbench.vhd:70:9:@100us:(report note): reset = '1' ||| SEL_FCT_in = 3 | SEL_FCT_out = 0 ||| SEL_OUT_in = 1 ||| SEL_OUT_out = 0
testbench.vhd:77:9:@200us:(report note): reset = '0' ||| SEL_FCT_in = 3 | SEL_FCT_out = 3 ||| SEL_OUT_in = 1 ||| SEL_OUT_out = 1
testbench.vhd:80:9:@200us:(report note): Test ok (no assert ...)
```

Quand reset vaut 1, toutes les sorties sont à 0, mais quand reset vaut 0, les sorties valent les entrées.



### 3.3 Mémoire d'instruction

On test sur les 4 premiers reports, les 4 instructions de la fonction 1 puis on test à la dernière ligne que la sortie ne change pas si le CE (cheap enabler) n'est pas activé.

Résultats attendus (d'après le design) :

```
-- première fonction A mult. B RES_OUT_1 sur 8 bits
--SEL_FCT      SEL_ROUTE      SEL_OUT
Instr_memory(0) <= "0000000000"; --no op      A -> Buf A      | 0
Instr_memory(1) <= "1111011100"; --A * B      B -> Buf B      | 0
Instr_memory(2) <= "0000111000"; --no op      S -> Mem 1      | 0
Instr_memory(3) <= "0000000001"; --no op      A -> Buf A      | Mem 1 --ici peut importe les 2 premières
```

La sortie est un chiffre 10 bits contenant SEL\_FCT, SEL\_ROUTE et SEL\_OUT permettant de piloter le reste des composants.

```
testbench.vhd:71:9:@200us:(report note): reset = '0' | CE_Instr = '1' | Instr_addr = 0 | SORTIE = 0
testbench.vhd:76:9:@400us:(report note): reset = '0' | CE_Instr = '1' | Instr_addr = 1 | SORTIE = 988
testbench.vhd:81:9:@600us:(report note): reset = '0' | CE_Instr = '1' | Instr_addr = 2 | SORTIE = 56
testbench.vhd:87:9:@800us:(report note): reset = '0' | CE_Instr = '1' | Instr_addr = 3 | SORTIE = 1
testbench.vhd:95:9:@1ms:(report note): reset = '0' | CE_Instr = '0' | Instr_addr = 2 | SORTIE = 1
```

Les sorties correspondent bien aux résultats attendus.

On effectue le même procédé pour les deux autres fonctions.

```
-- deuxième fonction (A add. B) xnor A RES_OUT_2 sur 4 bits et x_nor sur les 4 bits de poids faible
--SEL_FCT      SEL_ROUTE      SEL_OUT
Instr_memory(32) <= "0000000000"; --no op      A -> Buf A      | 0
Instr_memory(33) <= "1101011100"; --A add B      B -> Buf B      | 0
Instr_memory(34) <= "0111110000"; --A xor S      S -> Buf B      | 0
Instr_memory(35) <= "0100110000"; --not B (xnor) S -> Buf B      | 0
Instr_memory(36) <= "0000111000"; --no op      S -> Mem 1      | 0
Instr_memory(37) <= "0000000001"; --no op      A -> Buf A      | Mem 1
```

On fait l'addition avec le cœur, on fait ensuite le xor de ce résultat avec le A grâce au cœur encore une fois. Puis on stocke le résultat de cette opération dans un des buffers reliés au cœur, le buffer B en l'occurrence. On fait ensuite l'opération not sur ce résultat ce qui nous permet d'obtenir le résultat du xnor. On effectue ensuite le même procédé que pour la fonction 1 pour affecter le résultat à RES\_OUT.

```
-- troisième fonction (A0.B1 + A1.B0) RES_OUT_3 sur le bit de poids faible
--SEL_FCT      SEL_ROUTE      SEL_OUT
Instr_memory(64) <= "0000000000"; --no op      A -> Buf A      | 0
Instr_memory(65) <= "1000011100"; --déc droite A B -> Buf B      | 0
Instr_memory(66) <= "0101010100"; --A1 and B0    S -> Buf A      | 0
Instr_memory(67) <= "1010111000"; --déc droite B S -> Mem 1      | 0
Instr_memory(68) <= "0000110000"; --no op      S -> Buf B      | 0
Instr_memory(69) <= "0101000000"; --A0 and B1    A -> Buf A      | 0
Instr_memory(70) <= "0000110000"; --no op      S -> Buf B      | 0
Instr_memory(71) <= "0110000100"; --Buf A or Buf B Mem1 -> Buf A | 0
Instr_memory(72) <= "0000111000"; --no op      S -> Mem 1      | 0
Instr_memory(73) <= "0000000001"; --no op      A -> Buf A      | Mem1 --le dernier sel_route ne sert à rien
```

Dans les deux autres fonctions, le dernier SEL\_ROUTE stockant A dans le Buffer A n'est pas nécessaire mais comme il n'existe pas de cas no\_op (pas d'opérations) pour SEL\_ROUTE on met la valeur '0000' qui correspond à cette instruction.

## 4 Interconnexion et routage des données

### 4.1 Sel\_out

SEL_OUT[1]	SEL_OUT[0]	Significations
0	0	Aucune sortie : RES_OUT = 0
0	1	RES_OUT = MEM_CACHE_1
1	0	RES_OUT = MEM_CACHE_2
1	1	RES_OUT = S

Résultats du testbench conformes aux résultats attendus :

```
testbench.vhd:54:9:@100us:(report note): SEL_OUT = 0 | S = 1 | mem_1 = 2 ||| mem_2 = 3 ||| res_out = 0
testbench.vhd:63:9:@200us:(report note): SEL_OUT = 1 | S = 1 | mem_1 = 2 ||| mem_2 = 3 ||| res_out = 2
testbench.vhd:73:9:@300us:(report note): SEL_OUT = 2 | S = 1 | mem_1 = 2 ||| mem_2 = 3 ||| res_out = 3
testbench.vhd:84:9:@400us:(report note): SEL_OUT = 3 | S = 1 | mem_1 = 2 ||| mem_2 = 3 ||| res_out = 1
```

### 4.2 Sel\_route

SEL_ROUTE[3]	SEL_ROUTE[2]	SEL_ROUTE[1]	SEL_ROUTE[0]	Significations
0	0	0	0	Stockage de l'entrée A_IN dans Buffer_A
0	0	0	1	Stockage de MEM_CACHE_1 dans Buffer_A (4 bits de poids faibles)
0	0	1	0	Stockage de MEM_CACHE_1 dans Buffer_A (4 bits de poids forts)
0	0	1	1	Stockage de MEM_CACHE_2 dans Buffer_A (4 bits de poids faibles)
0	1	0	0	Stockage de MEM_CACHE_2 dans Buffer_A (4 bits de poids forts)
0	1	0	1	Stockage de S dans Buffer_A (4 bits de poids faibles)
0	1	1	0	Stockage de S dans Buffer_A (4 bits de poids forts)
0	1	1	1	Stockage de l'entrée B_IN dans Buffer_B
1	0	0	0	Stockage de MEM_CACHE_1 dans Buffer_B (4 bits de poids faibles)
1	0	0	1	Stockage de MEM_CACHE_1 dans Buffer_B (4 bits de poids forts)
1	0	1	0	Stockage de MEM_CACHE_2 dans Buffer_B (4 bits de poids faibles)
1	0	1	1	Stockage de MEM_CACHE_2 dans Buffer_B (4 bits de poids forts)
1	1	0	0	Stockage de S dans Buffer_B (4 bits de poids faibles)
1	1	0	1	Stockage de S dans Buffer_B (4 bits de poids forts)
1	1	1	0	Stockage de S dans MEM_CACHE_1
1	1	1	1	Stockage de S dans MEM_CACHE_2

Les 15 cas de stockage dans les mémoires et buffers fonctionnent correctement :

```
testbench.vhd:84:9:@100us:(report note): SEL_ROUTE = 0 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 2 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:90:9:@200us:(report note): SEL_ROUTE = 1 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 6 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:96:9:@300us:(report note): SEL_ROUTE = 2 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:102:9:@400us:(report note): SEL_ROUTE = 3 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 7 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:107:9:@500us:(report note): SEL_ROUTE = 4 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:113:9:@600us:(report note): SEL_ROUTE = 5 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 1 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:119:9:@700us:(report note): SEL_ROUTE = 6 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:126:9:@800us:(report note): SEL_ROUTE = 7 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 3 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:131:9:@900us:(report note): SEL_ROUTE = 8 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 6 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:137:9:@1000us:(report note): SEL_ROUTE = 9 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:143:9:@1100us:(report note): SEL_ROUTE = 10 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 7 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:149:9:@1200us:(report note): SEL_ROUTE = 11 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:154:9:@1300us:(report note): SEL_ROUTE = 12 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 1 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:160:9:@1400us:(report note): SEL_ROUTE = 13 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 0
testbench.vhd:166:9:@1500us:(report note): SEL_ROUTE = 14 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 1 | mem_2_in = 0
testbench.vhd:171:9:@1600us:(report note): SEL_ROUTE = 15 | S = 1 | A = 2 | B = 3 | buff_A_out = 4 | buff_B_out = 5 | mem_1_out = 6 | mem_2_out = 7 ||| Les sorties ||| buff_A_in = 0 | buff_B_in = 0 | mem_1_in = 0 | mem_2_in = 1
```

## 5 Intégration du cœur de microcontrôleur sur la carte ARTY

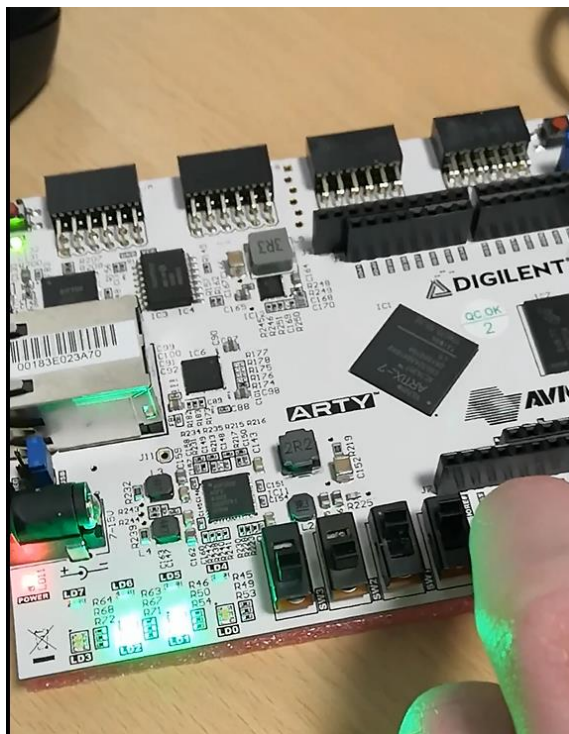
Lors de la dernière séance d'avancement de notre projet, le but était d'intégrer notre UAL sur la carte Xilinx Artix-35T FPGA. Malheureusement, nous n'étions pas assez avancés à ce moment-là et nous avons donc commencé par intégrer notre cœur d'UAL sur la carte et ainsi vérifier son bon fonctionnement. Pour ce faire, on utilise Vivado. Il y a besoin d'avoir un fichier de contrainte, fourni sur Moodle, ce fichier permet de connecter les propriétés/broches du FPGA (de la carte) à un « package\_pin\_lettre1 ». Cela permet de faire les liens entre les broches sur la carte et nos composants que nous avons déclaré dans notre fichier VHDL.

### Exemple:

```
set_property -dict {PACKAGE_PIN E3          IOSTANDARD LVCMOS33} [get_ports {CLK100MHZ}];
```

Le package\_pin\_e3 est maintenant connecté à notre clock100mhz de notre Top\_level\_entity.

Les 4 switch correspondent aux 4 bits de notre A et B, ainsi A sera égal à B. On relie les 4 boutons au SEL\_FCT permettant de sélectionner l'opération à effectuer. On relie ensuite le signal de sorties aux leds voulues ainsi que les retenues.



Exemple de l'addition de '0011' et '0011' -> '0110'

## 6 Conception des trois fonctions spécifiques

L'étape finale de ce projet que nous n'avons malheureusement pas pu tester sur la carte Arty consistait en l'élaboration des 3 algorithmes afin d'associer chacune des 3 fonctions à l'appui d'un des 4 boutons.

**RES\_OUT\_1 = (A mult. B) (*RES\_OUT\_1 sur 8 bits*)**

**RES\_OUT\_2 = (A add. B) xnor A (*xnor sur les 4 bits de poids faibles – RES\_OUT\_2 sur 4 bits*)**

**RES\_OUT\_3 = (A<sub>0</sub> and B<sub>1</sub>) or (A<sub>1</sub> and B<sub>0</sub>) (*RES\_OUT\_3 sur le bit de poids faible*)**

On utilise les 3 boutons de poids fort pour exécuter les 3 fonctions et le bouton de poids faible pour remettre à 0, réinitialiser les mémoires.

Comme indiqué sur le sujet, lorsque l'on appuie sur un des boutons associés à une des fonctions, on charge les instructions indiquant qu'il va falloir exécuter la fonction en question au prochain front (montant) d'horloge.

On lit ensuite sur le front descendant le résultat de la fonction exécutée puis on peut recommencer à charger des instructions en appuyant à nouveau sur un bouton.

Comme pour l'intégration du cœur, on choisit les valeurs de A et B avec les 4 switch de la carte modélisant les 4 bits de A et B. On affiche le résultat sur 8 bits avec les 8 leds rouges, on affiche les retenues éventuelles en sortie avec les 2 leds de poids fort bleu et on allume la led de poids faible verte pour indiquer quand le résultat est disponible (affiché sur la carte à l'aide des leds).

### **Description de l'algorithme :**

On doit gérer les 2 cas principaux, le premier cas où  $btn(0) = '1'$ , dans ce cas-là on reset les mémoires. Et l'autre cas quand le bouton de reset n'est pas pressé et que nous sommes sur front montant.

Dans ce cas-là on regarde l'état courant (*Current* dans le code) et on fait un « case » sur les 5 cas possible :

- S'il est au repos c'est qu'aucune instruction de fonction n'a été chargée pour le moment, il faut donc charger les instructions pour le prochain front montant. En fonction du bouton sur lequel on appuie, on change l'état courant et on initialise le compteur à l'indice de la mémoire d'instruction correspondant à la fonction souhaitée.
- S'il est à l'état F1, cela veut dire qu'on a chargé l'instruction au dernier front montant et qu'on doit alors exécuter la fonction 1 (A mult B). Pour ce faire, on incrémente le compteur jusqu'à la dernière case de la mémoire d'instruction correspondant aux étapes de notre fonction et une fois la dernière case atteinte, on s'arrête et on allume la led0 verte pour dire que le résultat est disponible.
- Même principe pour l'état F2
- Même principe pour l'état F3





## 6.2 Res\_out\_2 : l'addition, le xor et le not

Ce sera exactement le même procédé que pour la fonction 1 sauf que cette fois-ci il faudra appuyer sur le btn(2) donc le troisième bouton. Cela initialisera le compteur à l'indice 32 de la mémoire d'instruction s'incrémentera 5 fois (car on a 6 cases mémoires dans la mémoire d'instruction) pour effectuer toutes les opérations de calcul et de stockage déjà décrit précédemment dans la rubrique de la mémoire d'instruction. La led verte s'allume et le résultat s'affiche. Si on arrête d'appuyer sur le btn(2) la led verte s'éteint et on réinitialise l'état de notre automate et le compteur pour éventuellement réaliser une nouvelle fonction en appuyant sur un autre bouton.

```
-- deuxième fonction (A add. B) xnor A RES_OUT_2 sur 4 bits et x_nor sur les 4 bits de poids faible
```

	--SEL_FCT	SEL_ROUTE	SEL_OUT
Instr_memory(32) <= "0000000000";	--no op	A -> Buf A	0
Instr_memory(33) <= "1101011100";	--A add B	B -> Buf B	0
Instr_memory(34) <= "0111110000";	--A xor S	S -> Buf B	0
Instr_memory(35) <= "0100110000";	--not B (xnor)	S -> Buf B	0
Instr_memory(36) <= "0000111000";	--no op	S -> Mem 1	0
Instr_memory(37) <= "0000000001";	--no op	A -> Buf A	Mem 1

(Screen de la mémoire d'instruction pour la fonction 2)

## 6.3 Res\_out\_3 : la multiplication et l'addition des décalages !

Cette dernière fonction est la plus complexe des 3 car elle nécessite un plus grand nombre d'opérations et met donc plus de temps (fronts d'horloge) à se réaliser. Sinon le principe reste le même que pour les deux fonctions précédentes sauf que cette fois-ci on initialise le compteur à l'indice 64 de la mémoire d'instructions et on l'incrmente 9 fois car il y a 10 cases mémoire dans la mémoire d'instruction associées à la fonction n°3.

```
-- troisième fonction (A0.B1 + A1.B0) RES_OUT_3 sur le bit de poids faible
```

	--SEL_FCT	SEL_ROUTE	SEL_OUT
Instr_memory(64) <= "0000000000";	--no op	A -> Buf A	0
Instr_memory(65) <= "1000011100";	--déc droite A	B -> Buf B	0
Instr_memory(66) <= "0101010100";	--A1 and B0	S -> Buf A	0
Instr_memory(67) <= "1010111000";	--déc droite B	S -> Mem 1	0
Instr_memory(68) <= "0000110000";	--no op	S -> Buf B	0
Instr_memory(69) <= "0101000000";	--A0 and B1	A -> Buf A	0
Instr_memory(70) <= "0000110000";	--no op	S -> Buf B	0
Instr_memory(71) <= "0110000100";	--Buf A or Buf B	Mem1 -> Buf A	0
Instr_memory(72) <= "0000111000";	--no op	S -> Mem 1	0
Instr_memory(73) <= "0000000001";	--no op	A -> Buf A	Mem1 --le dernier

sel\_route ne sert à rien

(Screen de la mémoire d'instruction pour la fonction 2)





## 7 Conclusion

---

Ce projet fut véritablement intéressant car il nous aura permis d'encore mieux comprendre le fonctionnement d'un cœur de microcontrôleur.

Nous avons pour objectif de réaliser une UAL, une unité arithmétique et logique, capable de réaliser plusieurs traitements arithmétiques. Il nous fallait également réaliser les composants servant de mémoire temporaire, permettant à l'UAL d'avoir toutes les valeurs nécessaires à dispositions, ainsi que l'interconnexion, permettant de relier ces dits composants.

Cela nous aura également permis de nous conforter dans notre maîtrise du VHDL et dans notre compréhension des circuits numériques.