POLYTECHNIQUE
MONTRÉAL

INF1500
Logique des systèmes numériques

Laboratoire 2

Soumis par:
Mohamad Awad - 2034584
Lourhmati Oussama - 2081643
Groupe 05

Le 22 octobre 2020

1- Description du système

Vue de haut du système:

Notre système est une mini-UAL (Unité arithmétique et logique), un élément de base des microprocesseurs qui fait des opérations arithmétiques et logiques. Elle est composée de trois modules essentiels : un diviseur par 4, un comparateur et un multiplexeur, dont la conception sera détaillée dans les pages suivantes, la figure 1.0 introduit le schéma du système reliant les trois modules entre eux.

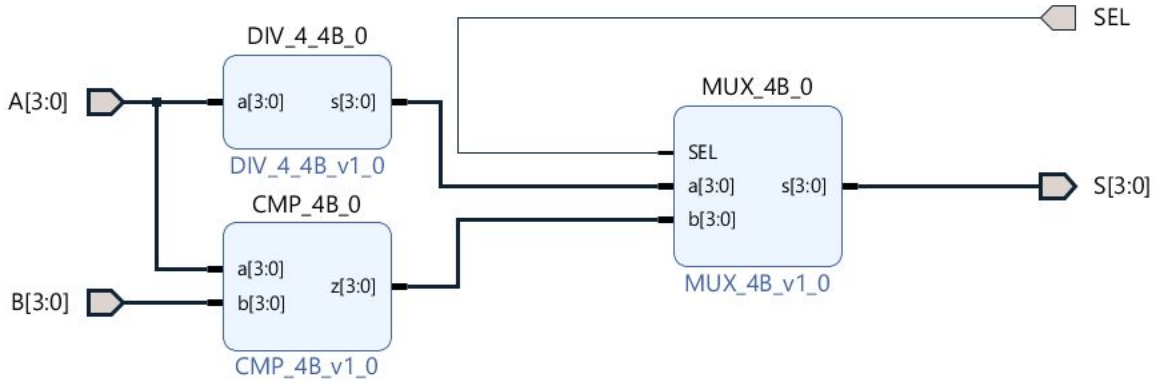

Figure 1.0 : les 3 modules essentiels formant UAL

Les deux entrées principales aux systèmes sont deux bus de 4 bits chacun : **A** et **B**. Premièrement, **A** entre dans le diviseur par 4 DIV_4_4B. Deuxièmement, **A** et **B** entrent dans le comparateur CMP_4B. Finalement, les sorties du diviseur et du comparateur qui sont aussi des bus de 4 bits chacun, entrent dans un multiplexeur MUX_4B avec une variable de sélection **SEL**. Dépendamment de la valeur de SEL, la sortie **S** prendra la valeur de sortie **s** du diviseur (SEL=0) ou de sortie **z** du comparateur (SEL=1). Enfin la sortie du système **S** est un bus de 4 bits aussi.

Le comparateur CMP_4B:

Le comparateur fait une comparaison bit par bit de **A** et de **B**, les entrées du système. Il reçoit alors, dans ses deux entrées, ces deux bus de 4 bits. Si le bit ***n*** de A est égale au bit ***n*** de B (***n*** ∈ {0,1,2,3}), alors le bit ***n*** de la sortie sera 1, sinon, il sera 0. Pour cela, le comparateur emploie 4 portes *XNOR (NON OU EXCLUSIF)* à 2 entrées chacune.La porte *XNOR* a une sortie 1 si ses deux entrées ont la même valeur, et 0 si elles ont différentes valeurs, parce que , en fait l'équation de la porte XNOR est **f = a•b + a'•b' . *Si a=b=1 , f= 1•1 +0•0 =1 , si a=b=0 , f= 0•0 +1•1=1 , si a=1 et b=0 , f = 1•0 + 0•1 = 0 + 0 =0.***

Alors le bit ***n*** de A et le bit ***n*** de B entre ensemble dans la porte *XNOR_**n*** qui sortira le bit ***n*** de la sortie **z** du comparateur. Les sorties des portes *XNOR* sont réunies dans un bus de sortie **z** de 4 bits. La figure 1.1 illustre le schéma du comparateur.

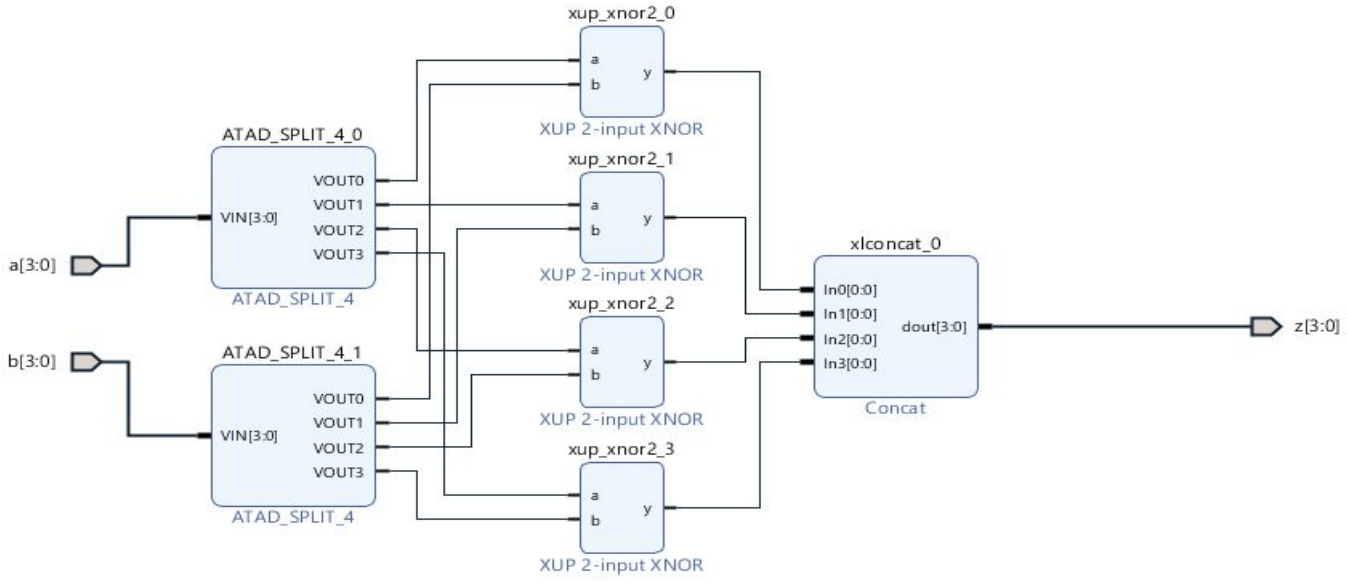

Figure 1.1: Comparateur CMP_4B

Le diviseur par 4 DIV_4_4B: Effectuer une division entière par 4

La table de vérité (figure 1.2) du *DIV_4_4B*, qui prend les deux bus d'entrées du système comme ses propres deux entrées et dont la sortie un bus de 4 bits, nous guide vers sa conception.

Entrées				Sorties			
a[3]	a[2]	a[1]	a[0]	s[3]	s[2]	s[1]	s[0]
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	0	1
0	1	1	0	0	0	0	1
0	1	1	1	0	0	0	1
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	0	0	1	0
1	0	1	1	0	0	1	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	1
1	1	1	1	0	0	1	1

Figure 1.2 : Table de vérité de DIV_4_4B

NB: la division entière à un résultat entier sans virgule, nous avons rempli cette table en faisant la division entière par 4 des 16 nombres à 4 bits et en copiant les résultats en binaire.

D'après la table de vérité du *DIV_4_4B* on constate les points suivants:

- Les bits de sorties **s[2]** et **s[3]** ont toujours la valeur 0. En effet une division **entière** par 4 d'un nombre de 4 bits donne au maximum 3 (0011 en binaire) (15/4=3). Donc les bits 2 et 3 de la sortie sont toujours 0.
- Le bit de sortie **s[1]** est égale au bit d'entrée **a[3]** dans tous les cas. En effet le bit **a[3]** représente $\mathbf{2^3}$, en divisant par 4, le bit qui représente $\mathbf{2^1}$ prendra sa valeur.
- Le bit de sortie **s[0]** est égale au bit d'entrée **a[2]** dans tous les cas. En effet le bit **a[2]** représente $\mathbf{2^2}$, en divisant par 4, le bit qui représente $\mathbf{2^0}$ prendra sa valeur.
- Ayant 4 sorties, on aura quatres équations des sorties en fonction des entrées:

$$\rightarrow \mathbf{s[0] = a[2]; s[1] = a[3]; s[2] = 0; s[3] = 0}$$

Enfin on pourrait concevoir notre *DIV_4_4B*; il y a plusieurs façons de le faire. Nous avons décidé de le faire de la manière suivante; on relie directement les bits 2 et 3 du bus d'entrée **a** aux bits 0 et 1 du bus de sortie **s respectivement**. Pour donner à s[2] et s[3] la valeur 0, on emploie une porte XOR à deux entrées: le bit a[0] dans chacune des entrées, la sortie va être normalement 0 car les deux entrées de la porte *XOR* ont la même valeur. On relie cette sortie aux deux bits s[2] et s[3]. On laisse le bit a[1] sans liens. La figure 1.3 introduit le schéma du diviseur.

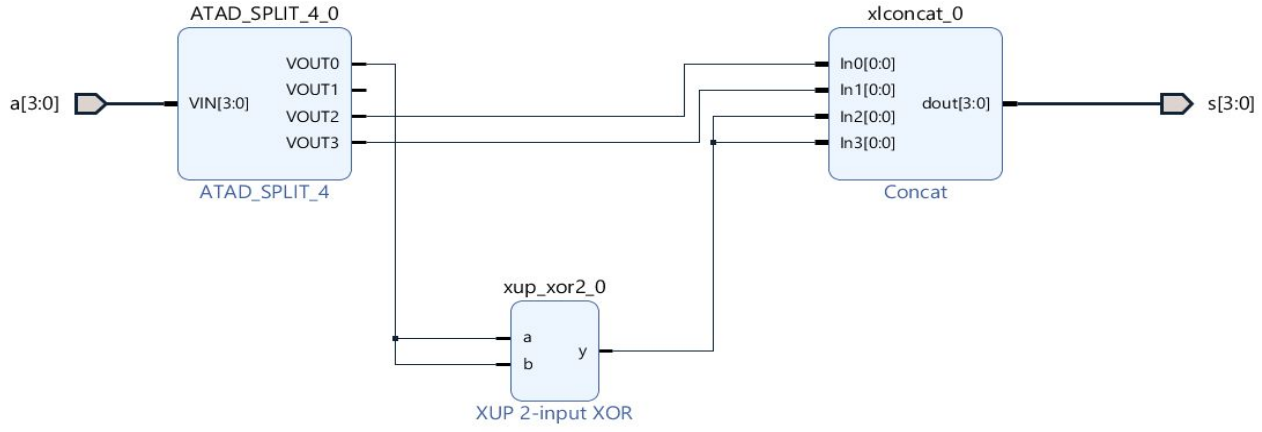

Figure 1.3: Diviseur DIV_4_4B

Le multiplexeur MUX_4B.

En fonction du signal de sélection SEL le MUX_4B choisira entre ses deux entrées: les bus des sorties du comparateur ou du diviseur. Le MUX_4B sortira un bus de 4 bits **S** qui sera la sortie de tout le système UAL. La table de vérité du multiplexeur est dans la figure 1.4 , si SEL est à 0, le bit ***i*** de la sortie sera le bit ***i*** de **a** et si SEL est à 1, le bit ***i*** de la sortie sera le bit ***i*** de **b**.

a[i]	b[i]	SEL	sortie: s[i]
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Figure 1.4 : table de vérité indice du MUX_4B

Puisque SEL est un signal de sélection qui choisit entre les deux entrées, on aura alors l'équation suivante de la sortie s[i] : **s[i] = a[i]• SEL'➕ b[i]•SEL.**

Si SEL est à 0 **s[i] = a[i]•1 ➕ b[i]•0 = a[i].** Si SEL est à 1 **s[i] = a[i]•0 ➕ b[i]•1 = b[i].**

On traduit alors l'équation du *MUX_4B* en circuit. La multiplication est traduite en porte **AND** dont les deux entrées sont **a[i]** et **SEL inversé** / **b[i]** et **SEL**, alors un total de huit portes **AND.** Ensuite, la sortie de la porte **AND** correspondant à **a[i]** et la sortie de la porte AND correspondant à **b[i]** , entrent ensemble dans une porte **OR_i** à 2 entrées (traduit la somme)

pour sortir le bit **s[i]** du bus de la sortie **S** du **MUX_4B** et du système entier; alors 4 portes **OR** en total. La figure 1.5 introduit le schéma du **MUX_4B.**

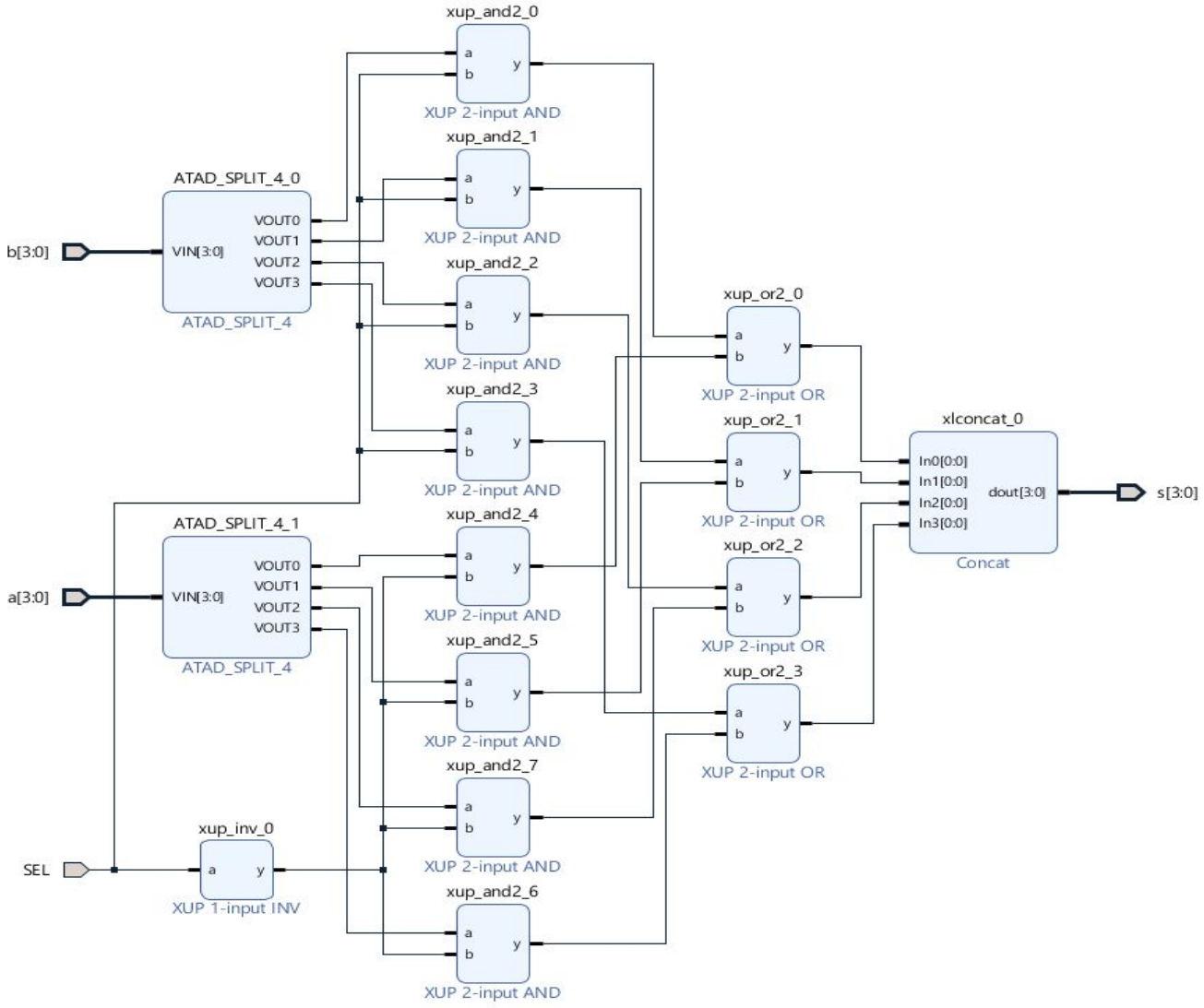

Figure 1.5: Multiplexeur MUX_4B

2- Vérification du système

Des simulations des différents modules ainsi que du système entier sont nécessaires pour vérifier le bon fonctionnement du système. Un test exhaustif serait utile pour couvrir toutes les possibilités mais pas vraiment nécessaire pour s'assurer qu'un système fonctionne

proprement. D'ailleurs, on utilise le test exhaustif (force clock) dans la plupart de nos simulations. Les figures 2.0, 2.1, 2.2 et 2.3 introduisent respectivement les simulations du CMP_4B , DIV_4_4B , MUX_4B et UAL.

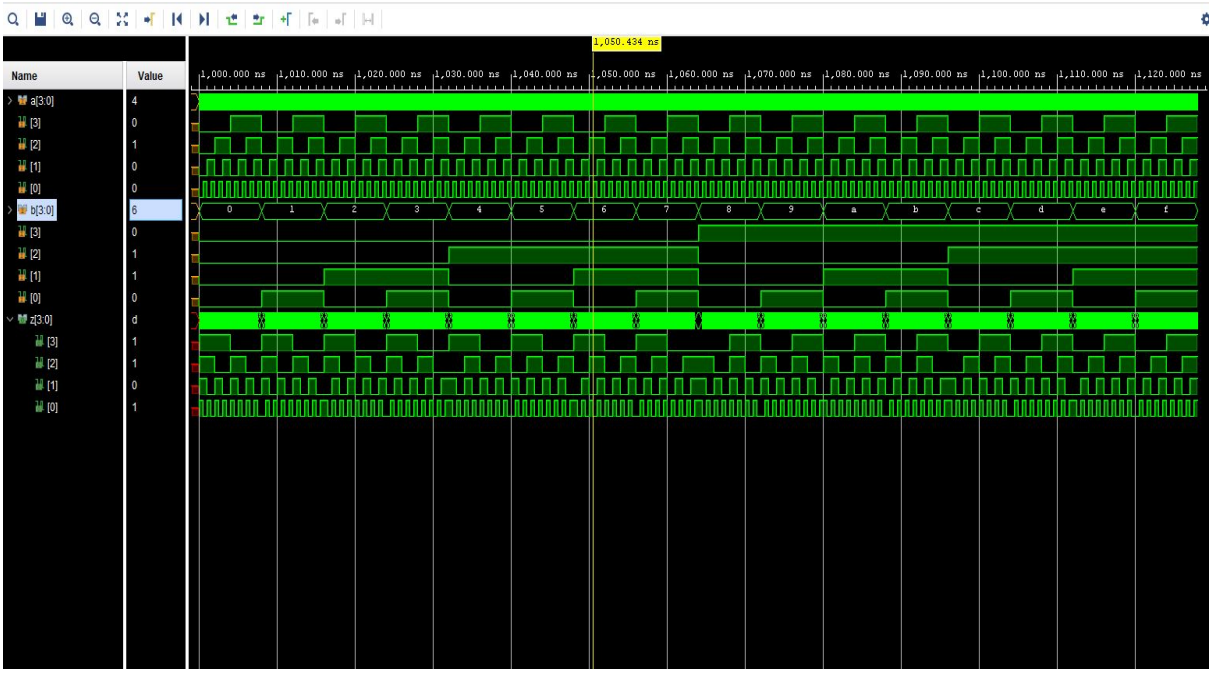

Figure 2.0 : Test exhaustif du CMP_4B.

Pour la vérification du **CMP_4B**, un force clock couvrira 256 possibilités de comparaison (16 possibilités de a * 16 possibilités de b)**.** Pour le bus ***"a"***, nous avons ajusté les périodes d'alternances des bits: ***0*** à 1 ns, ***1*** à 2 ns, ***2*** à 4 ns, ***3*** à 8 ns. Et pour le bus ***"b"***, nous avons continué à doubler les périodes , donc le ***0*** est à 16ns, ***1*** à 32ns, ***2*** à 64 ns, ***3*** à 128 ns. Le doublement des périodes est important pour que chaque possibilité soit représentée pour la même période de temps également avec les autres qui partagent la même probabilité (si on choisit aléatoirement une). La figure 2.0 valide que le sous-système fonctionne correctement.

Par exemple, dans cette simulation, on choisit aléatoirement (ligne jaune), une comparaison entre **0100** (4 en hexadécimal) et **0110** (6 en hexadécimal). Tel que prévu théoriquement, le sous-système **CMP_4B** compare bit par bit, lorsque le bit n de a et le bit n de b sont égaux, le bit n de sortie est 1, sinon il sera 0. **a[3] =b[3]** ;**a[2]=b[2]** ; **a[1]!= b[1]** et **a[0]=b[0]**. Pour cela la sortie z est **1101** (d en hexadécimal) .

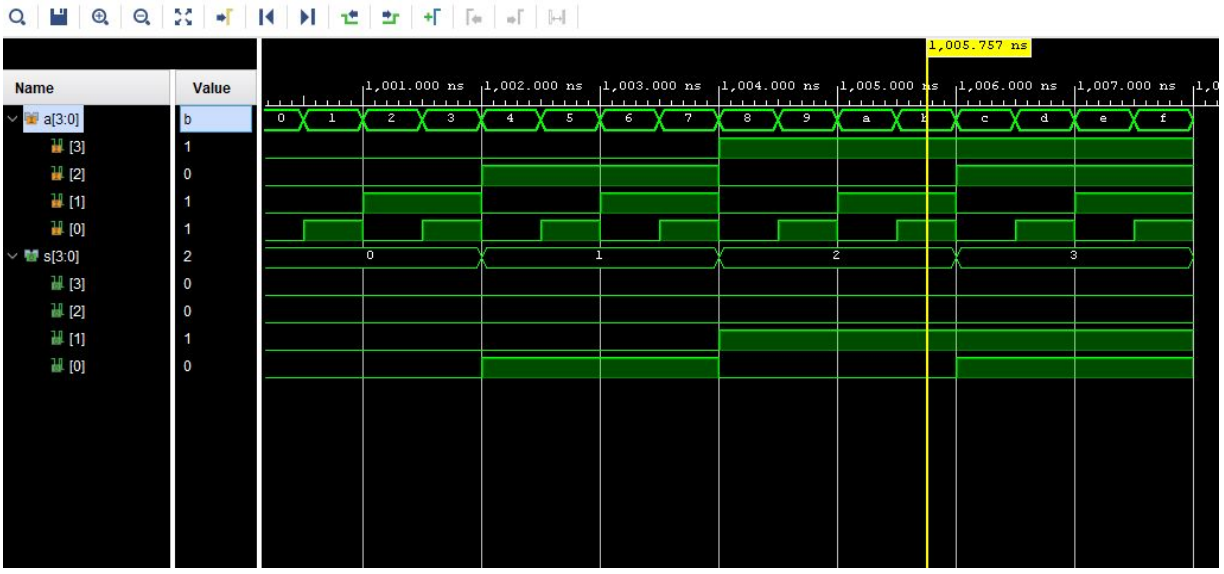

Figure 2.1 : Test exhaustif du DIV_4_4B.

Pour la vérification du diviseur par 4 **DIV_4_4B**, un force clock couvrira 16 possibilité de divisions: les 16 nombres de 4 bits divisés par 4 chacun**.** Pour le bus entrée ***"a"***, nous avons forcé les périodes suivantes: ***0*** à 1 ns, ***1*** à 2 ns, ***2*** à 4 ns, ***3*** à 8 ns. La figure 2.1 valide que ce sous-système fonctionne. Par exemple, dans la simulation, une division de **1011** (b en hexadécimal) par 4, donnera **0010** qui correspond à 2 en hexadécimal. Rappelons que cette division est **entière,** donc une virgule n'est pas comptée. Dans notre cas, 11/4=2.75, et puisque nous gardons seulement l'entier, le résultat est 2.

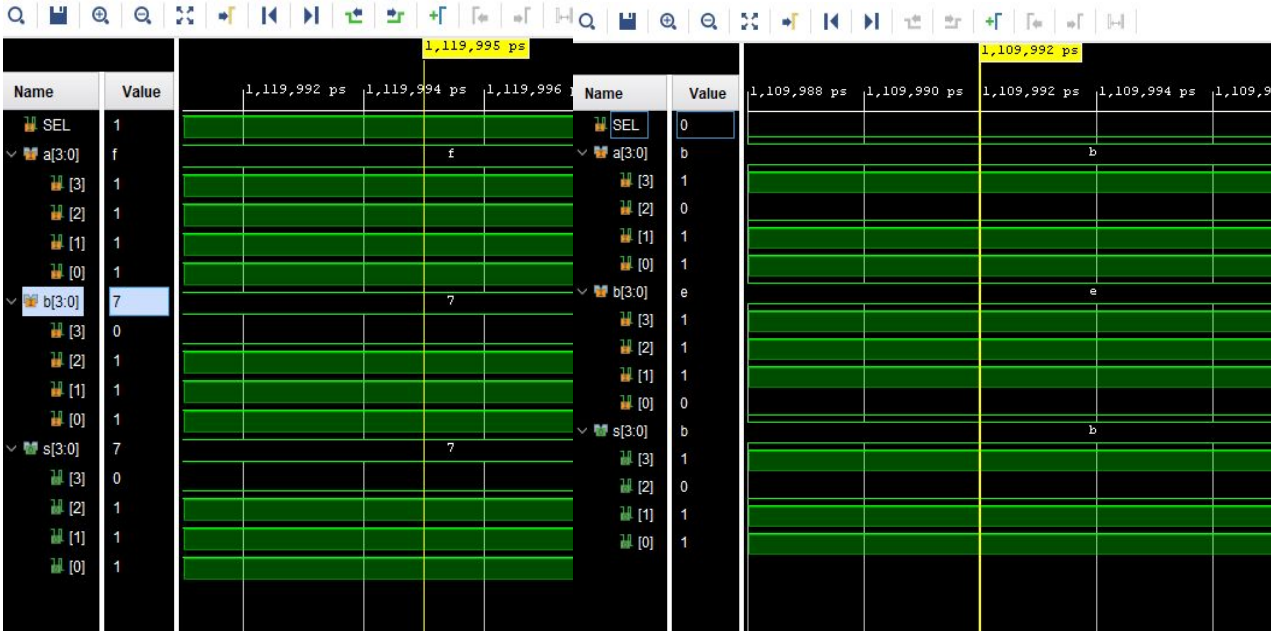

Figure 2.2: Simulation MUX_4B: Deux tests non-exhaustifs.

Pour la vérification du multiplexeur **MUX_4B**, deux simples tests non exhaustifs sont suffisants: Un où SEL est à 0 et l'autre où SEL est à 1. La figure 2.2 valide le fonctionnement du multiplexeur décrit dans la théorie : Lorsque SEL est à 0 la sortie a les mêmes bits que le bus a et lorsque SEL est à 1 la sortie a les mêmes bits que le bus b.

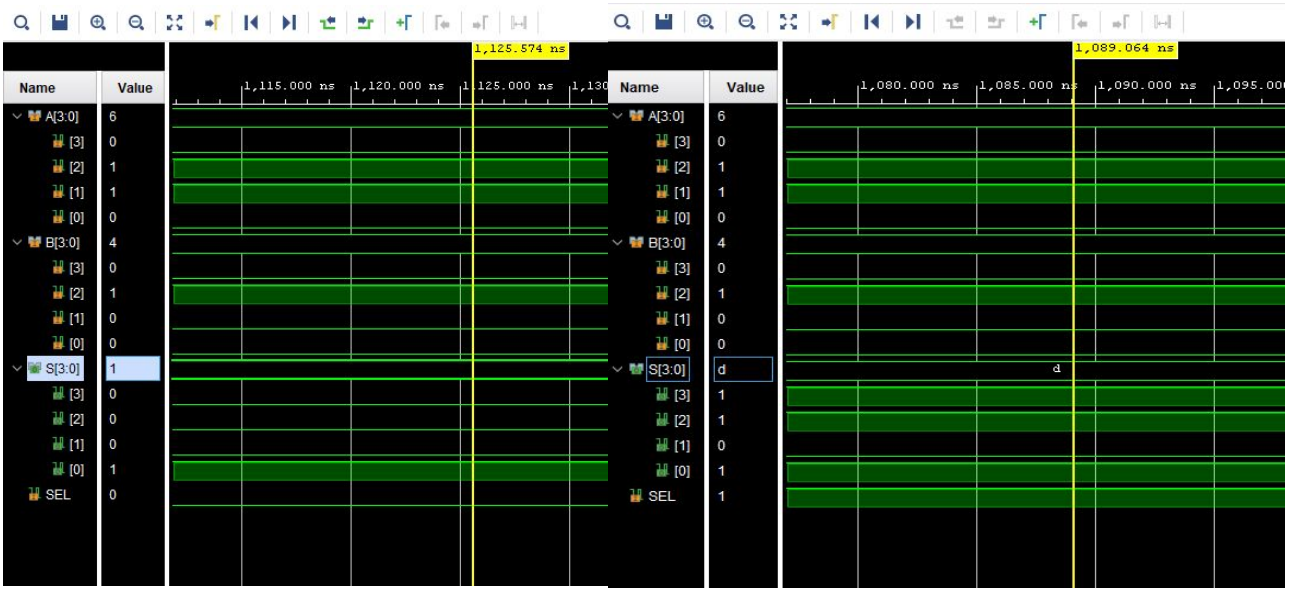

Figure 2.3: Simulation UAL: Deux tests non exhaustifs.

Enfin, pour vérifier que le système entier UAL fonctionne, nous avons décidé de faire deux tests non exhaustifs où SEL est une fois 0 et l'autre fois 1. On choisit comme entrées les entrées de la possibilité choisie aléatoirement dans le test exhaustif du comparateur : 0110 (6) et 0100 (4). Mais on met 6 dans le bus "a" au lieu de 4 pour deux raisons:

1. Vérifier que l'opération de la comparaison faite par CMP_4B est commutative. Donc pour SEL=1, le multiplexeur sélectionne la comparaison et le système sortira : 1101 (vu dans le test du CMP_4B page 8).
2. Pour que 6 soit l'entrée du DIV_4B et pour bien s'assurer que la division entière fonctionne (6/4=1.5) . Pour SEL = 0 , le multiplexeur sélectionne alors la division et la sortie de l'UAL est 1 (0001).

La figure 2.3 valide les résultats et confirme que le système fonctionne.

3- Réponses aux questions de l'énoncé

Question 1: Après avoir fait la table de vérité du diviseur, que remarquez-vous ? À quelle opération vous fait penser la division par 4 ?

Dans la table de vérité du diviseur, en comparant le nombre entré au nombre sorti dans toutes les cases, on remarque que les deux bits à gauche (2 et 3) du nombre entré deviennent les deux bits à droite (0 et 1) du nombre sortie, alors que les deux bits à gauche de ce dernier deviennent 0. Cela nous fait rappeler d'une opération **shift à droite** de deux bits.

Fichier de contrainte: Finalement, concernant le fichier de contrainte .xdc, nous avons suivi la même syntaxe que le code donné dans le guide (pratique avec le lab 1). Nous avons liés les sorties aux LED, et les entrées aux commutateurs (switch). Puisque dans notre laboratoire 2, nos entrées et sorties étaient sur 4 bits (sauf le SEL), nous avons reliés chaque bit des bus avec l'entrée ou la sortie correspondante, en commençant par 0 jusqu'au bit 3. Concernant le choix des LED et commutateurs, nous avons tout simplement respecté l'ordre du fichier *Nexys-4-DDR-Master.xdc* (de Moodle).