**Lab 3 – Unité Arithmétique et Logique**

**CEG2536 A – Architecture des Ordinateurs I**

**Automne 2017**

**École de Science Informatique et de Génie Électrique**

**Assistant de Professeur: Iymen Abdella**

Session de Lab 1

**Charbel Azzi**, 7663780

**Georges Ghossein,** 7653513

Date d’Expérience: 20/10/17

Date de Soumission: 10/11/17

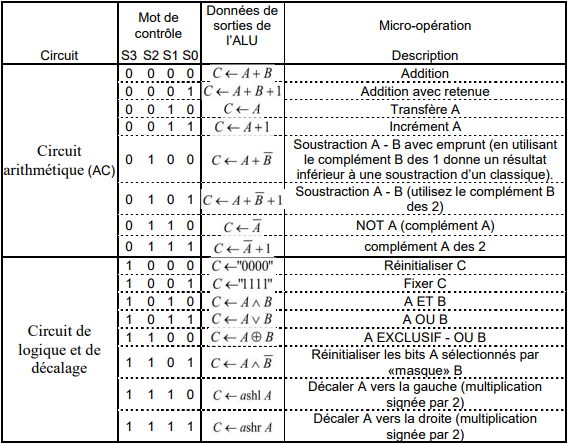
Partie théorique

Introduction du problème ou au laboratoire

L’unité arithmétique et logique (ALU) est la partie des ordinateurs qui est responsable de faire les calculs. Ils existent plusieurs types d’ALU cependant celle qu’on va construire dans ce laboratoire est un type simple qui contient 16 opérations unique divisé en 8 opérations arithmétique et 8 opérations logique.

Discussion du problème

Le problème, comme introduit dans la partie précédant est de construire une unité arithmétique et logique. Puisque ALU qu’on va construire contient 16 opérations différentes, une des exigences importantes a considéré sont les 16 opérations. Dans le tableau ci-dessous, on peut retrouver les opérations arithmétiques et logiques.



Discussion de la solution algorithmique

Il y a plusieurs méthodes à résoudre ce laboratoire. La méthode qu’on a décidé de suivre et d’asseye de séparer la partie arithmétique et logique. La première composante qu’on va regarder à est les colonnes des bits de sélections S3S2S1S0.

En observant cette colonne, on peut noter que la partie arithmétique et logique sont séparée en deux parties suivant le bit de S3 : Lorsque S3 est 0, ALU devrait être en mode de circuit arithmétique (AC) et lorsque S3 est 1, l’ALU devrait être dans le mode du circuit logique et de décalage (LC).

En suivant cette condition, on peut ensuite construire la partie AC et LC séparément et le rejoindre avec un MUX 2 à 1 pour choisir les deux différentes parties avec le bit de sélection S3. Pour construire le reste du circuit, on va suivre les 3 autres bits de sélection S2, S1 et S0.

Partie de conception

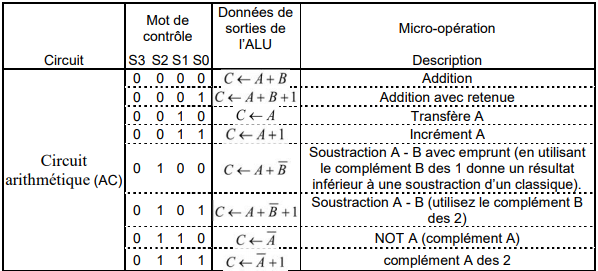
Présentation de la méthodologie de conception appliquée à la résolution des problèmes de laboratoire

Lors de la première partie de l’expérience nous avons indépendamment construit les composantes de l’ALU, soit le registre 4bit, le circuit logique 4bit, le circuit arithmétique 4bit ainsi que l’indicateur d’état.

**Le circuit arithmétique 4bit :**

La table de vérité qu’on a suivie pour construire le circuit arithmétique 4 bit est la suivante en oubliant le bit de sélection S3 :

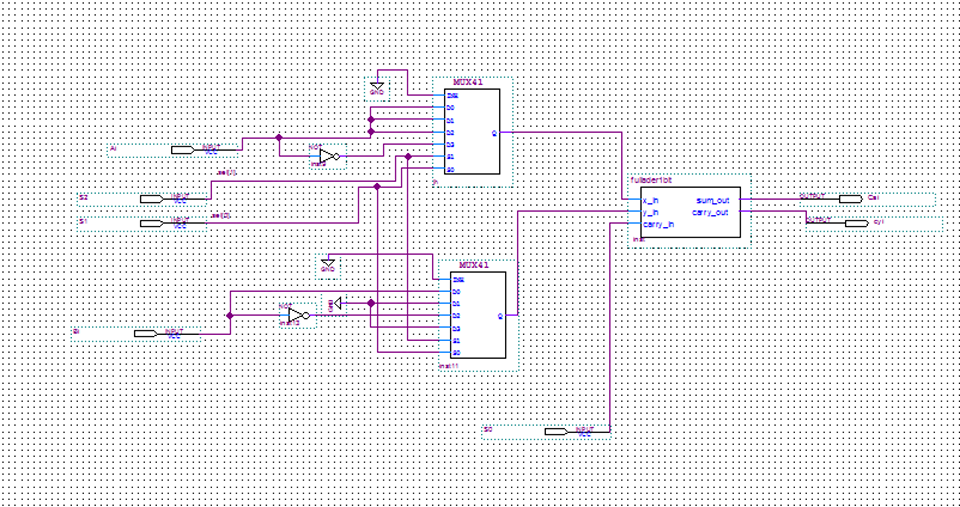
*Table 1: Tableau de micro-opération à être effectuer par le circuit arithmétique*



Le circuit arithmétique doit être de 4 bit. Une façon de construire ceci est de faire tous les opérations AC sur 1 bit, ensuite répliqué le circuit 4 fois pour avoir un circuit qui représente 4 bits. Habituellement la méthode à suivre est d’utilisé les tables de karnaugh, cependant en regardant le circuit, on peut faire plusieurs remarques :

* Chaque opération est une addition
* Le bit S0 est directement lié à l’addition. Ceci veut dire que lorsque le bit S0 est 1, on doit ajouter +1 à l’addition et lorsque le bit S0 est 0 on ajoute +0 au circuit.

En utilisant ces remarques on peut simplement utiliser un additionneur complet avec l’entrée du carry in comme le bit S0 et les entré de X et Y comme les valeurs d’A et B. Si l’entré est un complément, on a simplement à passer l’entrer à travers un porte logique NOT pour faire le complément de l’entrée. Le circuit ci-dessous représente le circuit 1 bit AC :



*Figure 2: Diagramme à bloc du circuit arithmétique (AC) 1-bit*

Pour le premier MUX 4-1 qui va représenter les bits d’A, les entrées sont les suivantes :

1. A
2. A
3. A
4. A’

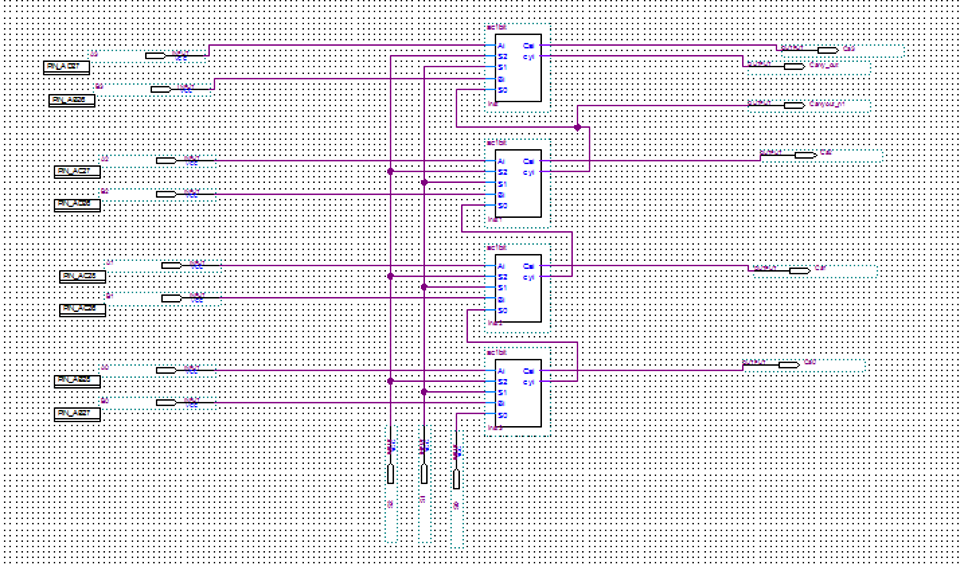
Pour le deuxièmes MUX 4- 1 qui va représenter les bits de B, les entrées sont les suivantes :

1. B
2. 0 (GND)
3. B’
4. 0 (GNG)

On peut trouver ces valeurs en utilisant la table de vérité et en utilisant les bits de sélection S2 et S1. Ensuite, on va lier le bit de sélection S0 à l’entre Carryin de l’additionneur complet, La sortie du première MUX 4-1 à l’entré X in de l’additionneur et la sortie du deuxièmes MUX 4-1 à l’entré Y in de l’additionneur. Le circuit maintenant représente l’équation suivante :

A + B + S0

Ensuite, on créer un symbole qui représente l’AC 1 bit et on le réplique 4 fois pour avoir 4 bit. Le circuit final AC 4 bit est le suivant :



*Figure 3: Diagramme à bloc du circuit arithmétique (AC) 4-bit*

Le résultat du circuit AC 4 bit peut être résumé:

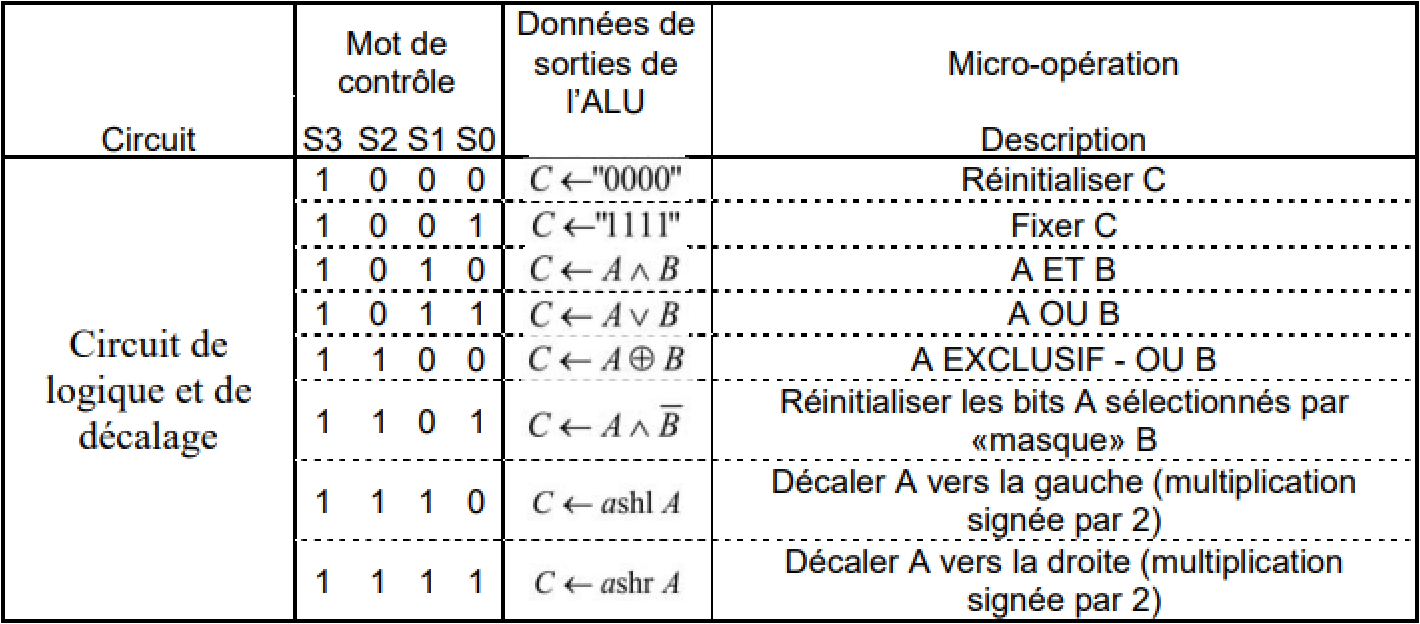
* Connecté les bits Ai et Bi aux symboles
* Connecté la sorite Cy des symbole AC (i-1) à l’entré ‘S0’ de symbole AC(i) pour représenter le bit du carry out qui est produit de l’additionneur complet
* La sortie CA(i) est la valeur produit par l’additionneur complet

Ceci conclu la conception du circuit AC 4 bit.

**Le circuit logique 4bit :**

La table de vérité qu’on a suivie pour construire le circuit logique 4 bit est la suivante en oubliant le bit de sélection S3 :

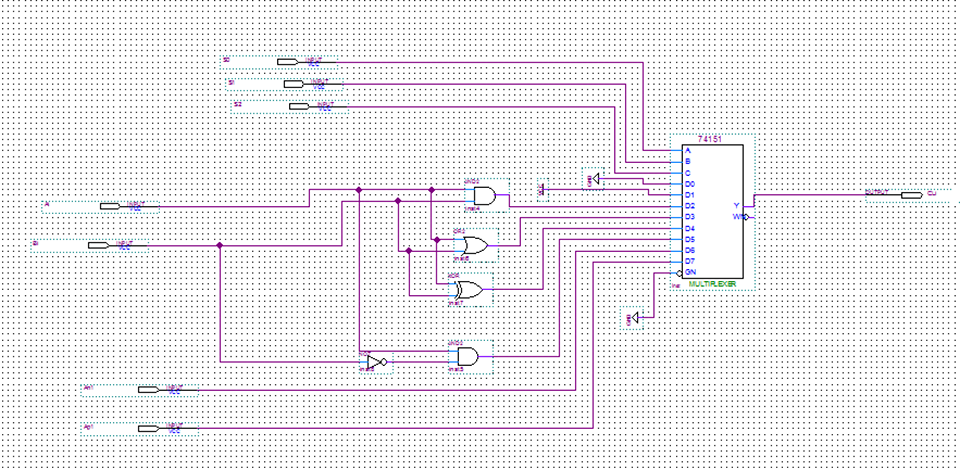
*Table 2: Tableau de micro-opération à être effectuer par le circuit logique*



Le circuit logique doit aussi être de 4 bit. Une façon de construire ceci est de faire tous les opérations LC sur 1 bit, ensuite répliqué le circuit 4 fois pour avoir un circuit qui représente 4 bits. Habituellement la méthode à suivre est d’utilisé les tables de karnaugh, cependant en regardant le circuit, on peut faire plusieurs remarques :

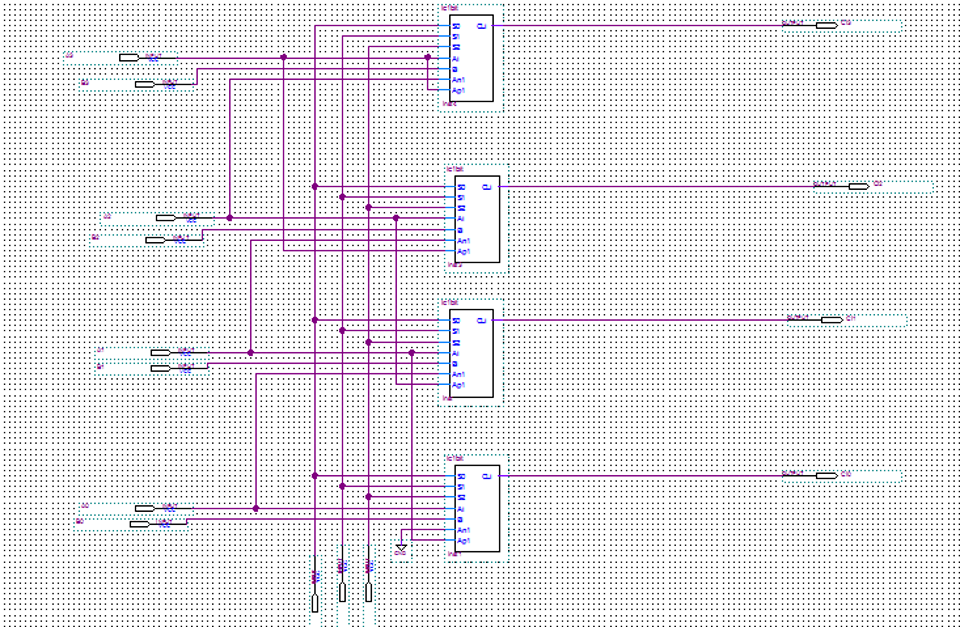
* Chaque opération est différente et unique

En utilisant cette remarque, on peut construire ce circuit sans faire les tables de karnaugh. Il est possible de simplement utilisé un MUX 8 – 1 et d’utilisé des portes logique quand nécessaire pour faires les opérations avec l’entrée du MUX. Ensuite, les bits de sélections S2, S1 et S0 vont choisir qu’elle rangé est utilisé. Voici le circuit 1 bit LC :



*Figure 4: diagramme à bloc du circuit logique(LC) 1-bit*

Ensuite, en créant un symbole de de circuit 1 bit, on peut le répliquer 4 fois pour obtenir le circuit LC 4 bit :



*Figure 5: Diagramme à bloc du circuit logique et de décalage(LSC) 4-bit construit à l’aide de LC 1-bit*

Le résultat du circuit LC 4 bit peut être résumé:

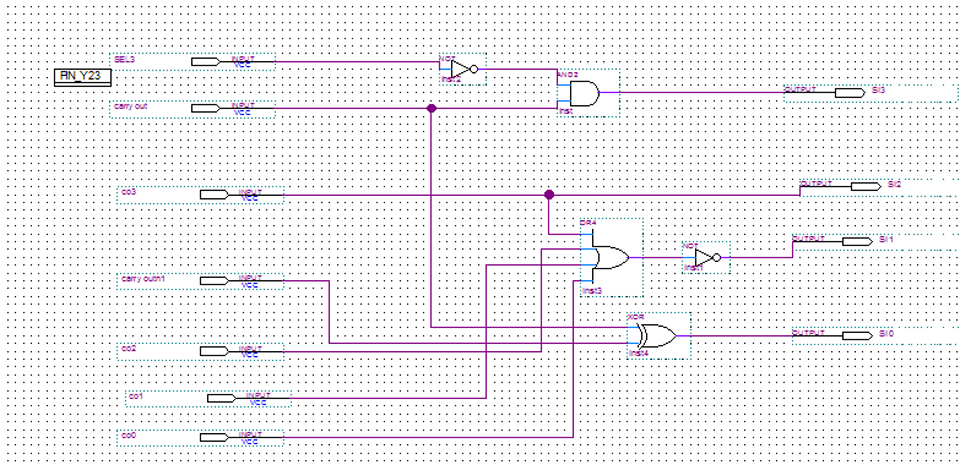
* Connecté les bits Ai et Bi aux symboles
* La sortie CL(i) est la valeur produit par le circuit
* Connecter l’entré An-1 du symbole à Ai-1 pour représenter le décalage à gauche (le premier bit va être connecté a 0 puisqu’il n’a pas un bit précédant à prendre)
* Connecter l’entrée Anp1 du symbole à Ai+1 pour représenter le décalage à droit (le denier bit va simplement être connecter à Ai pour garder le même signe

**L’indicateur d’état :**

L’indicateur d’état est la dernière composante de l’ALU qu’on doit construire. L’important de l’indicateur d’état est qu’il fournies de l’information additionnelle à l’ordinateur qui est lié à la sortie C de l’ALU. Les quatre états que l’indicateur doit avoir sont les suivantes :

1. Cy : Ce bit représente la retenue qui est produit par le circuit arithmétique. Cette valeur devrait être seulement 1 si l’opération est une opération arithmétique que si la sortie Carry out du dernier additionneur complet est aussi 1.
   * 1. Conception : Porte Logique AND comment Entré S3’ et Carry out du dernier bit. Le bit S3 doit être complémenter puisque lorsqu’on a une opération arithmétique la valeur de S3 est 0.
2. S : Ce bit représente le signe de la réponse. Cette valeur devrait être 1 seulement lorsque le bit le plus significatif (C3) à une valeur de 1.
   * 1. Conception : Connecté S à la valeur de C3 pour qu’elle prenne la même valeur
3. Z : Ce bit représente la valeur de zero. Ce bit doit prendre la valeur de 1 lorsque tous les bit ce sortie ( C3, C2, C1 et C0) sont 0.
   * 1. Conception : Utilisé une porte logique NOR à 4 entré et connecté chaque entre à 1 bit de Ci. Si tous les bits sont zero on va obtenir 1. Voici une démonstration – ( C3 + C2 + C1 + C0 )’ = (0 + 0 + 0 + 0)’ = (0)’ = 1. Donc, en utilisant une porte logique NOR on assure que la valeur de Z va seulement être 1 lorsque tous les bits de sorite Ci sont égales a 0.
4. V : Ce bit représente un débordement. Ce bit doit prendre la valeur de 1 lorsqu’un débordement est produit lors de l’exécution des opérations sur des nombres signées dans la représentation de complément des 2.
   * 1. Conception : Utilisé une porte logique XOR à 2 entrées de connecté le bit Carry out de l’additionneur complément du quatrième bit (CA4) et le carry out de l’additionneur complet du troisième bit (CA3).

Le circuit de l’indicateur d’état est montré ci-dessous :



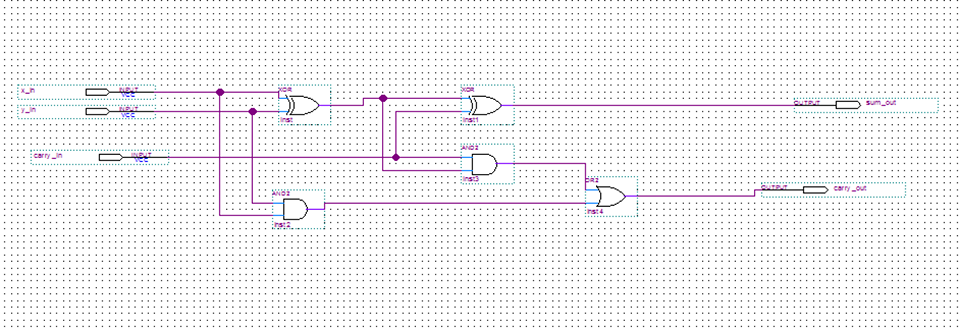
*Figure 6: Diagramme à bloc du circuit d’état*

Discussion sur les composantes utilisée

Deux composantes importantes ont été utilisé pour construire l’ALU : L’additionneur complet et le registre 4 bit.

**L’additionneur complet :**

L’additionneur complet a été utilisé pour faire la conception du circuit arithmétique puisque tous les opérations arithmétiques sont des additions. La façon le plus simple a été de construire un additionneur complet, le créée comme symbole et l’utiliser dans le circuit AC. Le circuit et table de vérité utilisé pour l’additionneur complet sont démontré dans les figures suivantes :



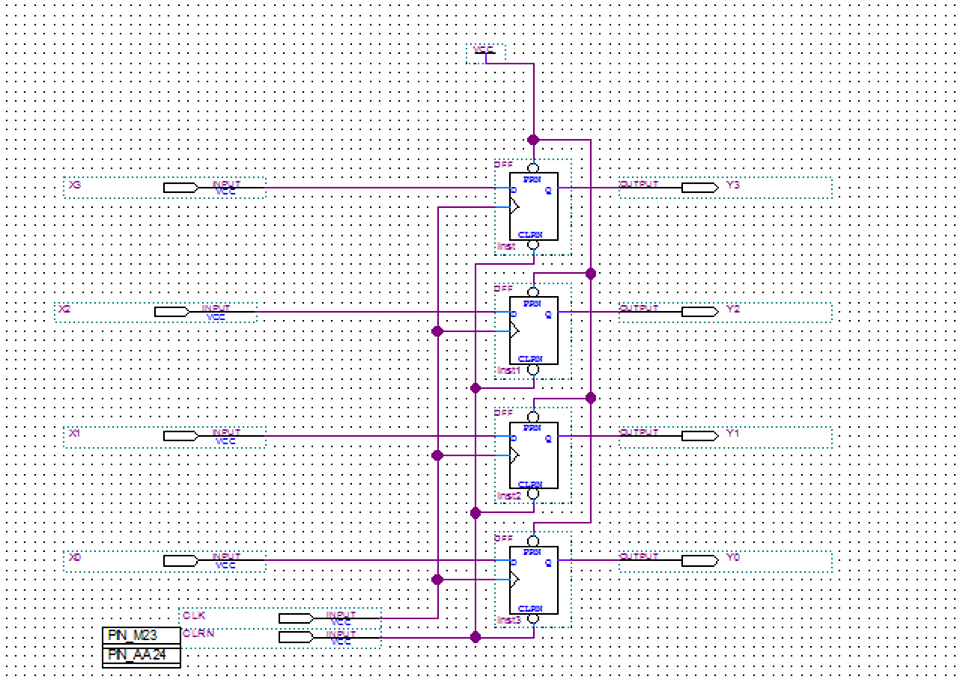
*Figure 7: Diagramme à bloc d’un additionneur complet*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Xin | Yin | Cin | Cout | S |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Registre 4 bit :**

Les registre 4 bit on aussi jouer un rôle important dans la création de l’ALU. L’importance des registre 4 bit est de fournir les 4bits d’entré d’A (A3A2A1A0) & de B (B3B2B1B0) et stocker les valeurs obtenues par l’ALU. Sans les registre 4bit, les données ne seront pas sauvegardées.

Pour concevoir le registre de 4 bit, on a simplement utilisé 4 flipflops D, 1 par bit. Le circuit du schéma du registre 4 bit est démontré ci-dessous :



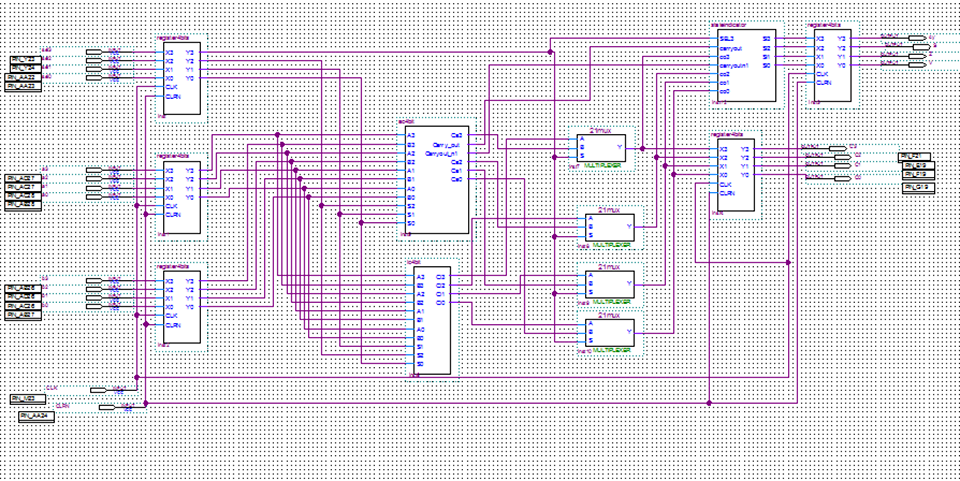
*Figure 8: Diagramme à bloc du registre 4-bit.*

Discussion de la solution réelle

Suite à la conception de tous les sous-parties de l’ALU, il est possible de concevoir le circuit final qui va représenter le circuit 4 bit de l’ALU. Les composantes qu’on va avoir besoin sont les suivantes :

1. 5 registre de 4 bits
   * 1. Registre pour les bits de sélection : S3, S2, S1, S0
     2. Registre pour les bits d’entrée A : A3, A2, A1, A0
     3. Registre pour les bits d’entrée B : B3, B2, B1, B0
     4. Registre pour les bits de sortie d’état : Cy, S, Z, V
     5. Registre pour les bits de sortie C : C3, C2, C1, C0
2. 1 circuit AC de 4 bit qui est représenté par un symbole
3. 1 circuit LC de 4 bit qui est représenté par un symbole
4. 1 circuit pour le circuit d’état qui est représenté par un symbole
5. 4 MUX 2 à 1 pour choisir entre le circuit LC et AC, 1 pour chaque Bit

Ensuite, il faut simplement faire la connexion de tous les entrées et tous les sorties et ce qu’on obtient est le circuit final pour l’ALU qui est démontré ci-dessous :



*Figure 9: l’ALU complet composé de registre 4 bit, LSC, AC, le circuit d’état*

Discussion sur les points difficile (bonus)

En général, le laboratoire c’est bien déroulé cependant on a rencontré quelques difficultés. Les difficultés qu’on a rencontrées a été d’utiliser les symboles que Quartus nous fournit comme le mux 8-1, 4-1 et 2-1. Les bits de sélection n’était pas très claire et la première fois qu’on a fait la connexion de tous les composantes on a pas simuler les bonnes opérations.

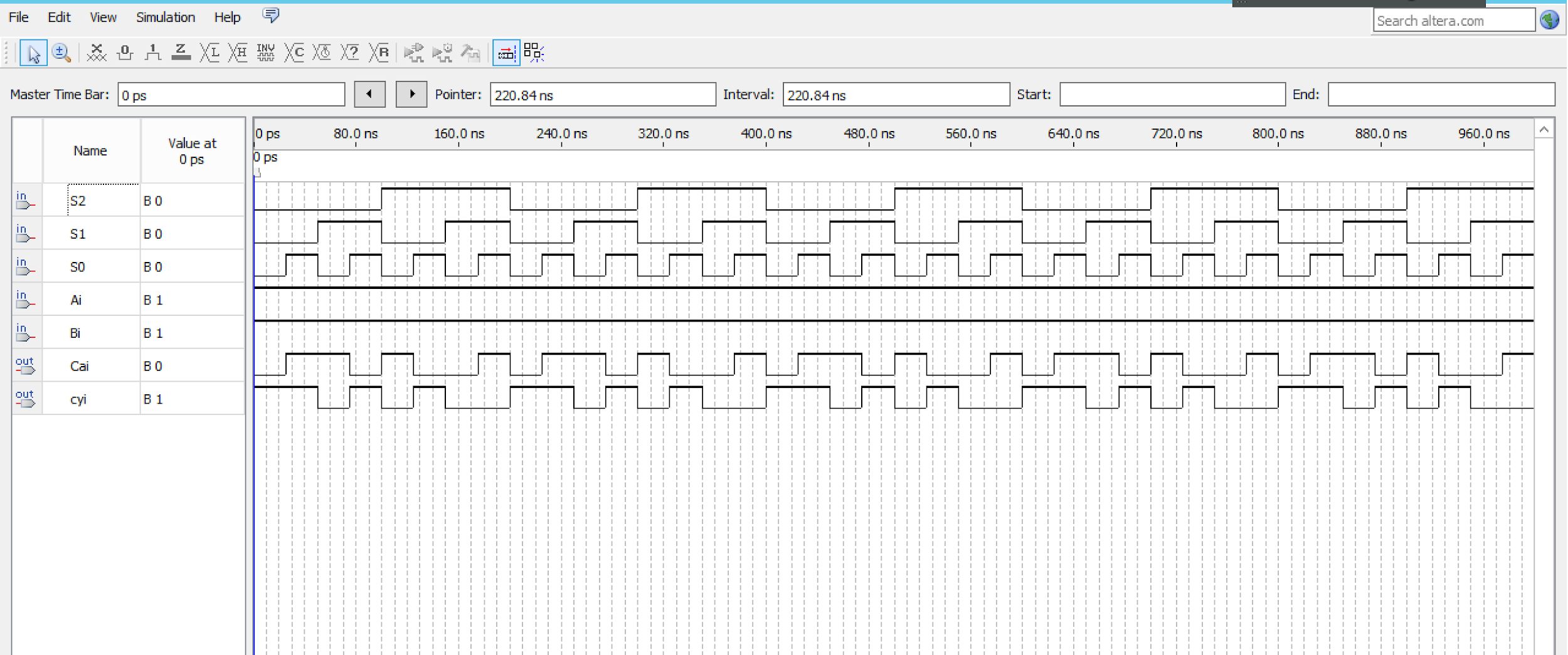
En fessant du debbuging, on a réalisé que la raison que notre ALU ne fonctionner pas est parce que tous notre bit de sélection a été inverser (S2 a été connecté à la sélection pour S0).

Mises-en ouvre réelle

Montrer les résultats de la simulation/la synthèse

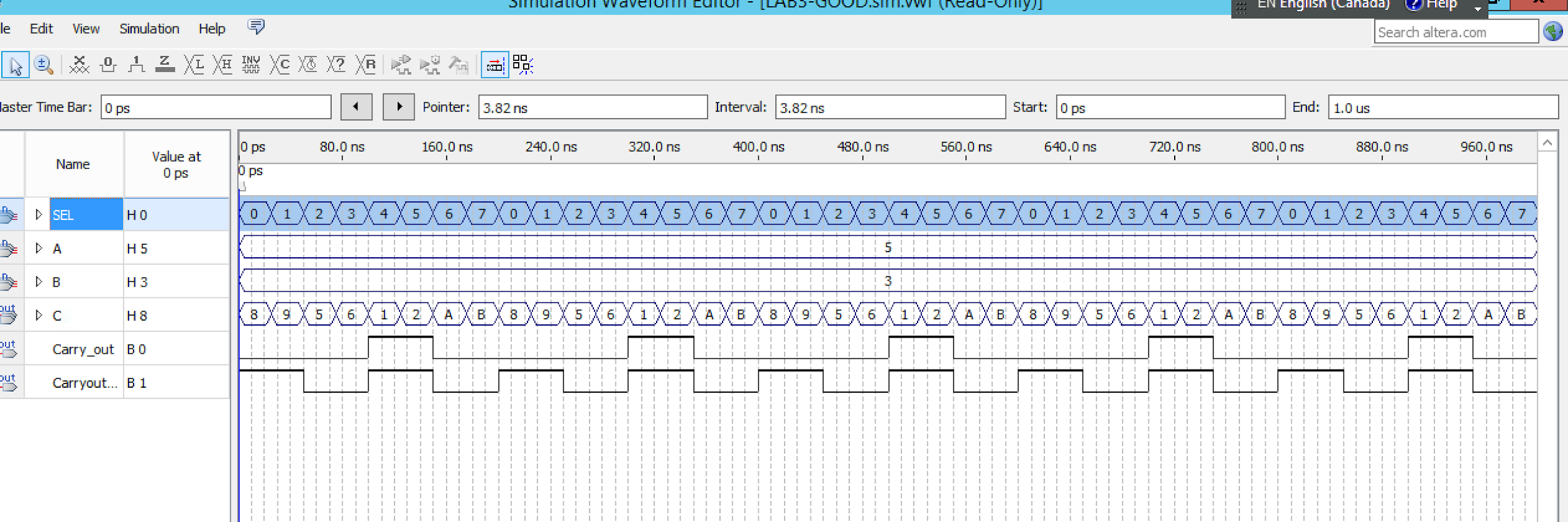
Pour bien assurée que notre ALU fonctionne, on a fait la simulation de chaque composante de l’ALU (AC et LC). Ceci nous a confirmé qu’on a bien fais la conception de chaque sous-partie de l’ALU. En ordre, on va démontrer un exemple de simulation pour chaque circuit

Simulation du circuit AC à 1 bit avec A = 1 et B = 1 :



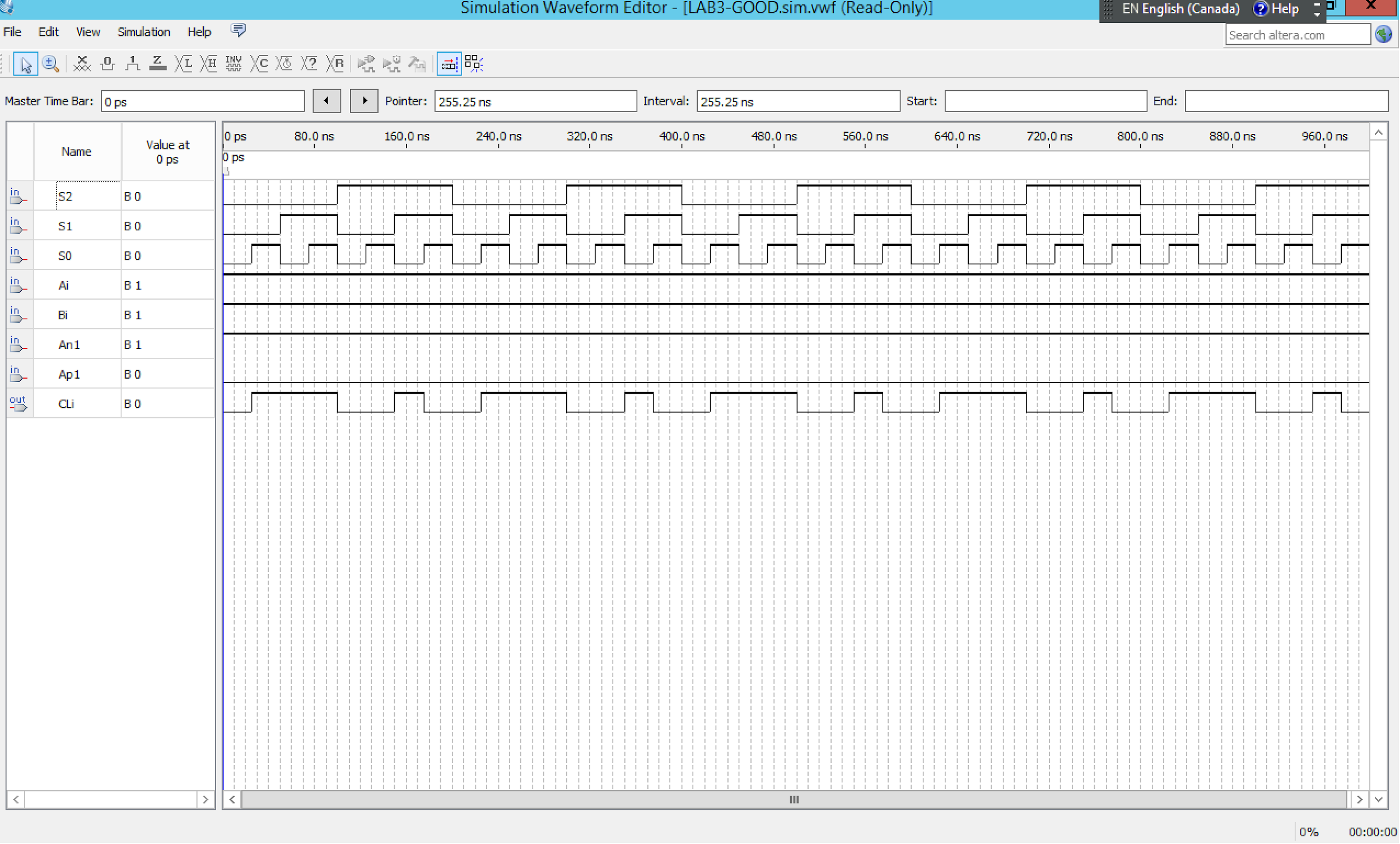
*Figure 10: simulation du circuit arithmétique à 1 bit*

Simulation du circuit AC à 4 bit avec A = 5 et B = 3 :



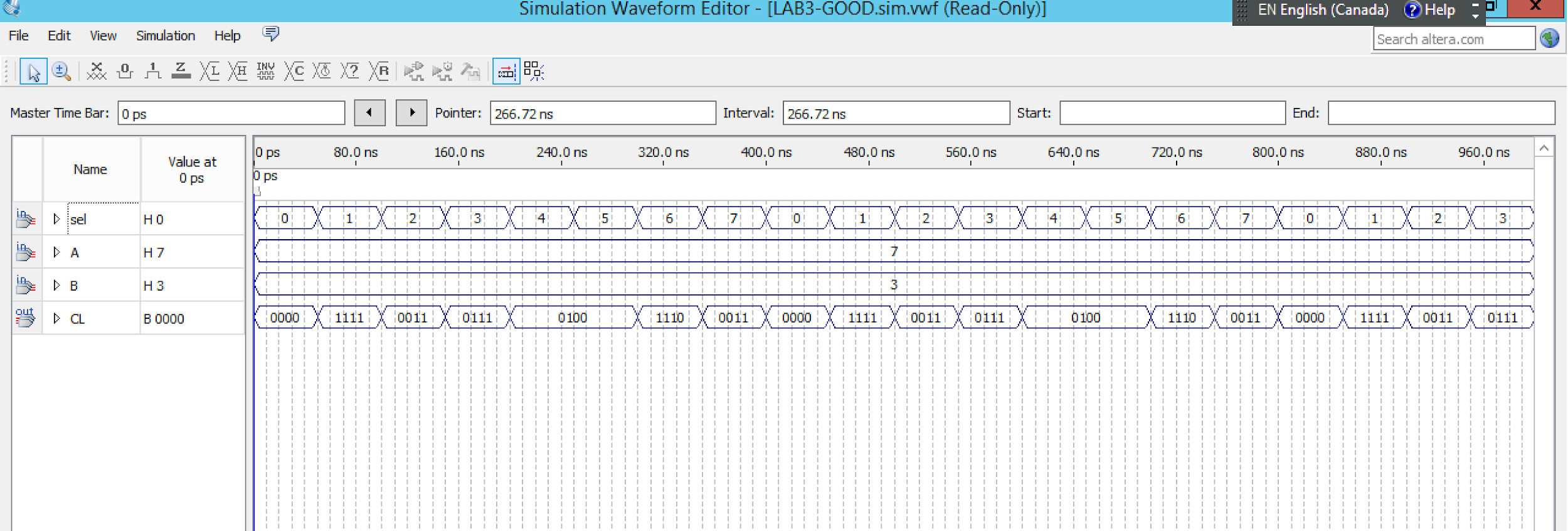
*Figure 11: simulation du circuit arithmétique à 4 bit*

Simulation du circuit LC à 1 bit avec A = 1 et B = 1 :



*Figure 12: Simulation du circuit logique à 1 bit*

Simulation du circuit LC à 4 bit avec A = 7 et B = 3 :



*Figure 13: Simulation du circuit logique à 4 bit*

Ayant comme objectif l’exécution de divers micro-opérations, nous avons simulé le fonctionnement de notre ALU à l’aide de Quartus II afin de visualiser la sortie en fonction des entrées A et B à 4-bit. Nous avons alors fait une simulation où A = 0001 et B=0000, soit les valeurs 1 et 0 en hexadécimal :

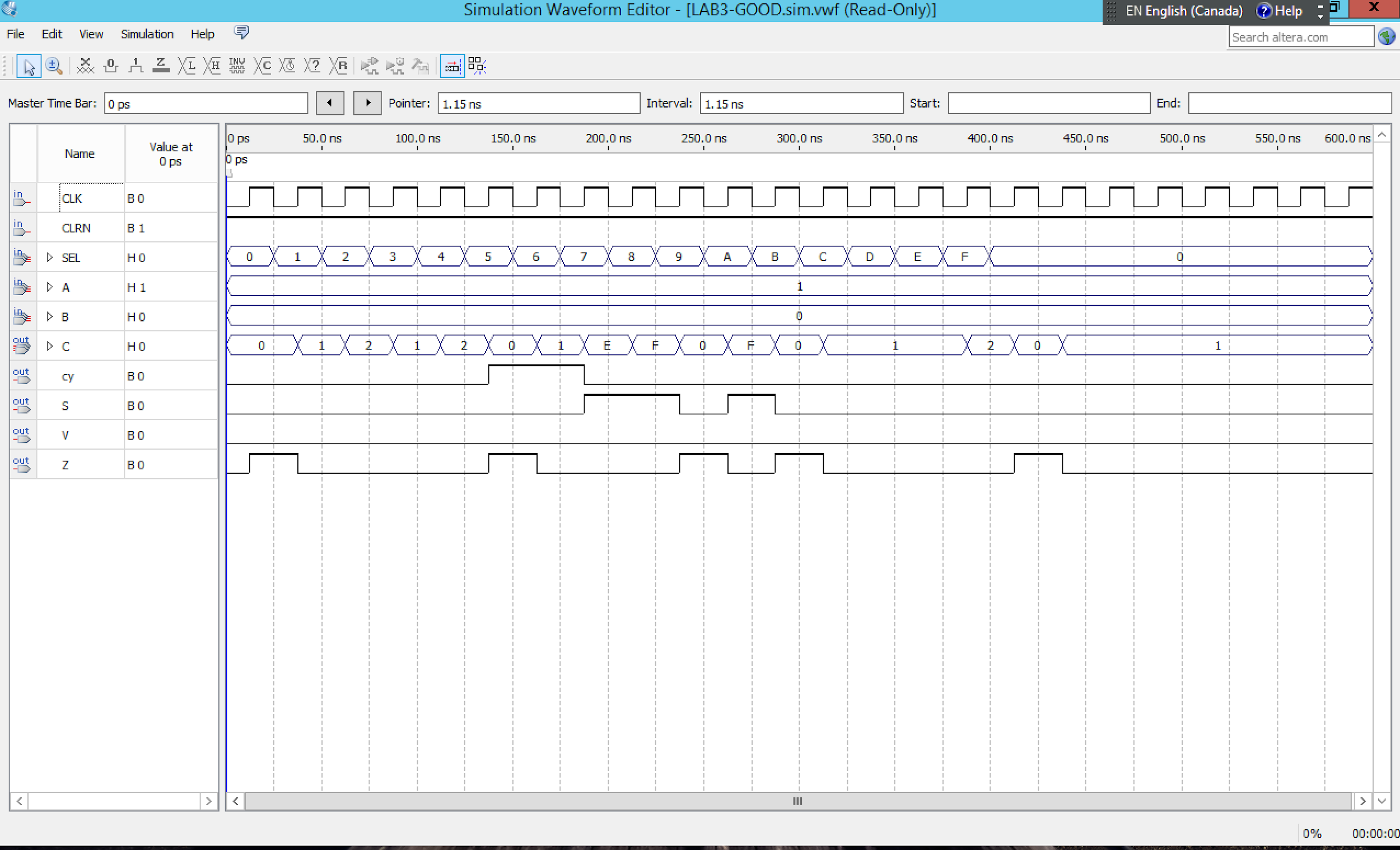


Figure 14: Simulation des micro-opérations effectué par l’ALU ou A = 1, B =0 (Valeurs en hexadécimal)

Vérification (démonstration en direct)

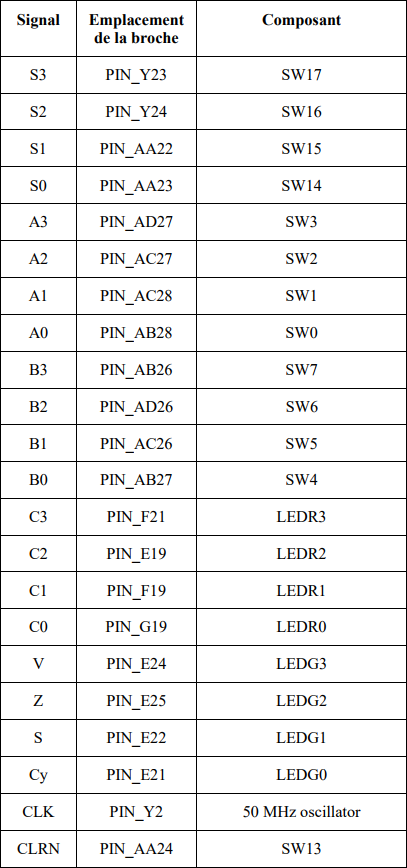
Étant donnée que notre simulation démontre que notre ALU est fonctionel, nous avons pu remplir le tableau de séquence de micro-opérations retrouver dans le fichier d’instruction de laboratoire :

*Table 3: Liste de micro-opérations à être effectué selon le guide de laboratoire*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Cycle d’horloge** | **Micro-opérations RTL** | **S3** | **S2** | **S1** | **S0** | **A←op1** | **B←op2** | **C** | **V,Z,N,Cy** | **S16** | **A16** | **B16** | **C16** | **St16** |
| **1** | C← A ^ B’ | 1 | 0 | 0 | 1 | 1010 | 0011 | 1000 | 0010 | D | A | 3 | 8 | 2 |
| **2** | C← ashl A | 1 | 1 | 1 | 0 | 0110 | xxxx | 1100 | 0010 | E | 6 | x | C | 2 |
| **3** | C← A + B | 0 | 0 | 0 | 0 | 0011 | 0101 | 1000 | 1010 | 0 | 3 | 5 | 8 | A |
| **4** | C← A + 1 | 0 | 0 | 1 | 1 | 1100 | xxxx | 1101 | 0010 | 3 | C | x | D | 2 |
| **5** | C← A ⊕ B | 1 | 1 | 0 | 0 | 0011 | 0101 | 0110 | 0000 | C | 3 | 5 | 6 | 0 |
| **6** | C← A’ | 0 | 1 | 1 | 0 | 1010 | xxxx | 0101 | 0000 | 6 | A | x | 5 | 0 |
| **7** | C← 0000 | 1 | 0 | 0 | 0 | xxxx | xxxx | 0000 | 0100 | 8 | x | x | 0 | 4 |
| **8** | C← A + B’ +1 | 0 | 1 | 0 | 1 | 0101 | 0011 | 0010 | 0001 | 5 | 5 | 3 | 2 | 1 |
| **9** | C← A | 0 | 0 | 1 | 0 | 1110 | xxxx | 1110 | 0010 | 2 | E | x | E | 2 |
| **10** | C← A’ + 1 | 0 | 1 | 1 | 1 | 0110 | xxxx | 1010 | 0010 | 7 | 6 | x | A | 2 |
| **11** | C← A^B | 1 | 0 | 1 | 0 | 0101 | 0011 | 0001 | 0000 | A | 5 | 3 | 1 | 0 |
| **12** | C← A + B + 1 | 0 | 0 | 0 | 1 | 0001 | 0010 | 0100 | 0000 | 1 | 1 | 2 | 4 | 0 |
| **13** | C← ashr A | 1 | 1 | 1 | 1 | 1101 | xxxx | 1110 | 0010 | F | D | x | E | 2 |
| **14** | C← A + B’ | 0 | 1 | 0 | 0 | 0110 | 0101 | 0000 | 1101 | 4 | 6 | 5 | 0 | D |
| **15** | C← 1111 | 1 | 0 | 0 | 1 | xxxx | xxxx | 1111 | 0010 | 9 | x | x | F | 2 |
| **16** | C← A v B | 1 | 0 | 1 | 1 | 1100 | 1010 | 1110 | 0010 | B | C | A | E | 2 |

De plus, on a aussi chargé notre circuit la carte Altera pour assurer que notre circuit fonctionne en simulation et en réalité. Les broches d’entrée de l’ALU (S, A, B) où S signifie la sélection S0-S3, A et B étant les données d’entré suit la configuration de broche dans le tableau ci-dessous :

*Table 4: Liste d’assignement des broches pour la carte Altera*



Discussion

Pour faire la conception de l’ALU, nous avons construit un registre 4-bit en utilisant des bascule D synchronisées avec l’horloge afin de stocké les donné d’entrer. Par la suite, nous avons schématisé le circuit arithmétique et logique qui a pour but d’exécuter les opérations booléennes en prenant les valeurs stockées dans les registres comme entré.

Afin d’exécuter des opérations arithmétiques nous avons schématisé un circuit arithmétique basé sur un additionneur complet qu’on a construit nous-même. Ensuite, à l’aide d’un multiplexeur 8x1 et quelques portes logiques, on a pu concevoir un circuit logique. Ce circuit logique peut également performer un décalage vers la droite et la gauche.

Enfin, nous avons schématisé les diverses composantes nécessaires afin d’obtenir un ALU complet et fonctionnel.

Ce laboratoire nous à démontrée qu’il est possible de construire un circuit de plusieurs bits, quatre dans notre cas, qui a le potentiel de faire plusieurs opérations uniques et qui est aussi capable de fournir de l’information additionnelle telle de l’état du circuit. En construisant ce circuit, nous avons aussi remarqué comment fonctionnes quelques opérations que notre ordinateur utilise à chaque seconde et à chaque minute.