POLYTECHNIQUE
MONTRÉAL

INF1500
Logique des systèmes numériques

Laboratoire 1

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

Le 2 octobre 2020

1- Description du système

Le système considéré pour ce laboratoire est un circuit à base de portes logiques conçu pour faire l'addition et soustraction sur 4 bits de deux nombres binaires. Ce système est formé de 4 blocs de *FullAdder1B*, un sous-système qui additionne trois nombres de 1 bit chacun. *Voir la figure de page 3 et 4 en lisant les descriptions suivantes*

Les entrées ***"a"*** de chaque *FullAdder1B* sont les 4 bits du nombre A et les entrées ***"b"*** de chaque *FullAdder1B* sont les 4 bits du nombre B, qui sont reliées d'abord chacune à une porte ***XOR*** avec l'entrée ***"op"***, qui prend la valeur 1 pour une soustraction et qui va inverser toutes les bits de B en passant par les portes ***XOR,*** cette inversion n'est autre que le processus du complément à 1. Si ***"op"*** a une valeur de 0, les portes ***XOR*** reliées aux bits de B n'auront aucun effet.

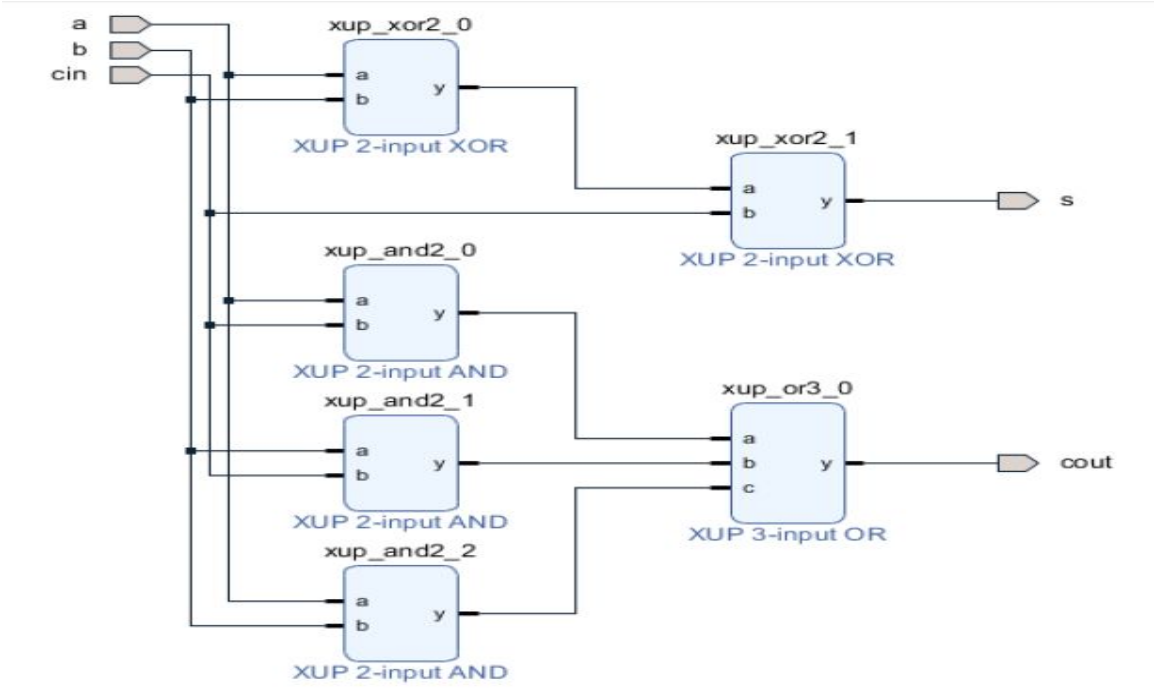
Description des 4 *FullAdder1B :* le *FullAdder1B* additionne 3 bits : ***a, b et cin,*** **a et b** joueront les rôles des bits de **A et B** respectivement et **cin** jouera en général le rôle d'une retenue résultante d'une addition précédente, le *FullAdder1B* effectue l'addition par un sous système équivalent à une porte **XOR** à trois entrées: **a, b et cin** et qui aura une sortie **s** : le résultat de l'addition. un autre sous système du *FullAdder1B* est formé de 3 portes **AND** qui ont comme entrées : **a.b , a.cin , b.cin** . Les sorties des portes **AND** entrent ensemble dans une finale porte **OR** qui fait sortir une **cout**, une façon de l'expliquer : si au moins deux des entrées du *FullAdder1B* ont une valeur de 1,une des portes **AND** sortira 1 qui entre dans la porte **OR** et la **cout** prendra la valeur 1.

l'entrée ***"cin"*** du *FullAdder1B_0* (lié à la première bit de **A**), n'est autre que l'entrée ***"op"*** du système , en cas de soustraction, ***"op"***=1 , va à la fois inverser les bits de **B** à travers les portes **XOR** et additionner 1 à la première bit de **A**: ce processus n'est autre que le complément à 2. La sortie ***"cout"*** de chaque *FullAdder1B* est liée à l'entrée ***"cin"*** du *FullAdder1B* suivant (ceci permet de prendre une retenue s'il y a lieu en additionnant les bits de A aux bits de B ou du complément à 2 de B), le ***"cout"*** du *FullAdder1B_3* indiquera s'il y a une retenue résultante de cette addition.

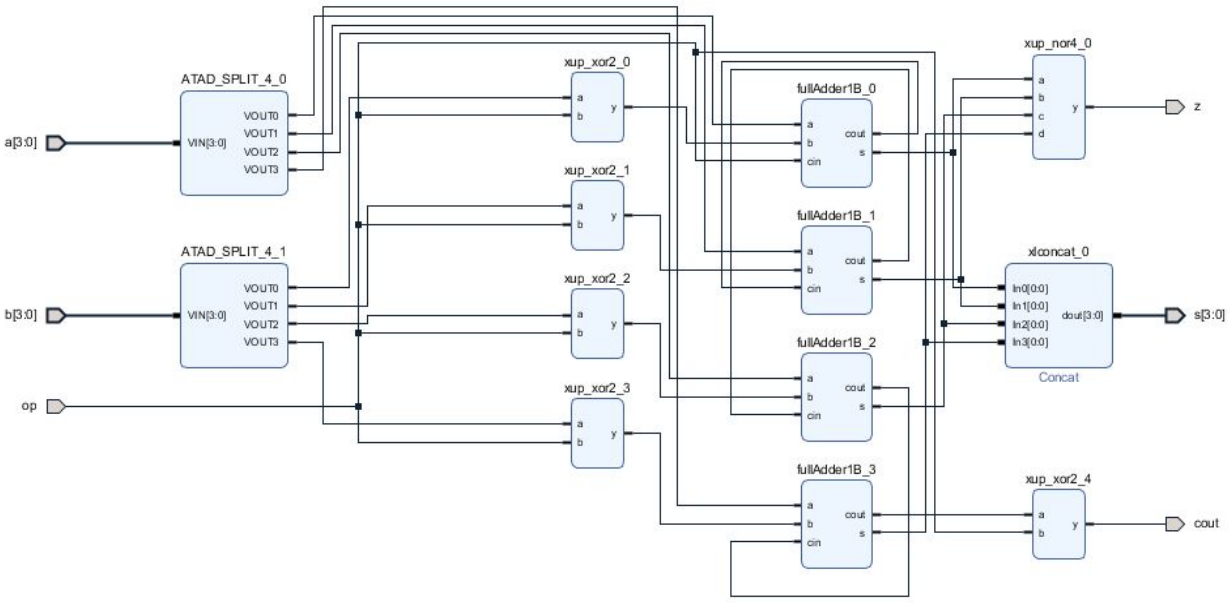
Les sorties ***"s"*** de chaque *FullAdder1B* entrent ensemble dans une porte ***NOR*** à 4 entrées, qui permet de vérifier si le résultat est nul: *si chacune de ces entrées est égale à 0, le résultat est nul et la porte* ***NOR*** *sortira "****z=1"*****.** Elles représentent aussi les bits du résultat dans le bus de sortie ***"S"*** . La sortie **"*COUT du système*",** d'une dernière porte **XOR_4** à deux entrées : ***"cout"*** du *FullAdder1B_3* et ***"op",*** indique s'il y a une retenue en cas de

débordement : au cas où l'addition dépasse les 4 bits, le résultat est différent de celui qu'on attend si on faisait l'addition en décimal, bref, ***COUT=1***). Elle peut aussi montrer que, pour une soustraction (op =1), la soustraction en décimal a donné un résultat négatif qui s'exprime différemment en binaire (par son complément à 2) et par une sortie ***COUT=1*** aussi. Ce fait sera expliqué en détail dans la section 2.

UN *FULLADDER1B*



UN *FULLADDER4B*



2- Vérification du système

Nous pouvons affirmer que notre système fonctionne en se basant sur sa conception, qui est expliquée dans la section 1. Pour approfondir la compréhension de notre système, on explique encore deux faits importants :

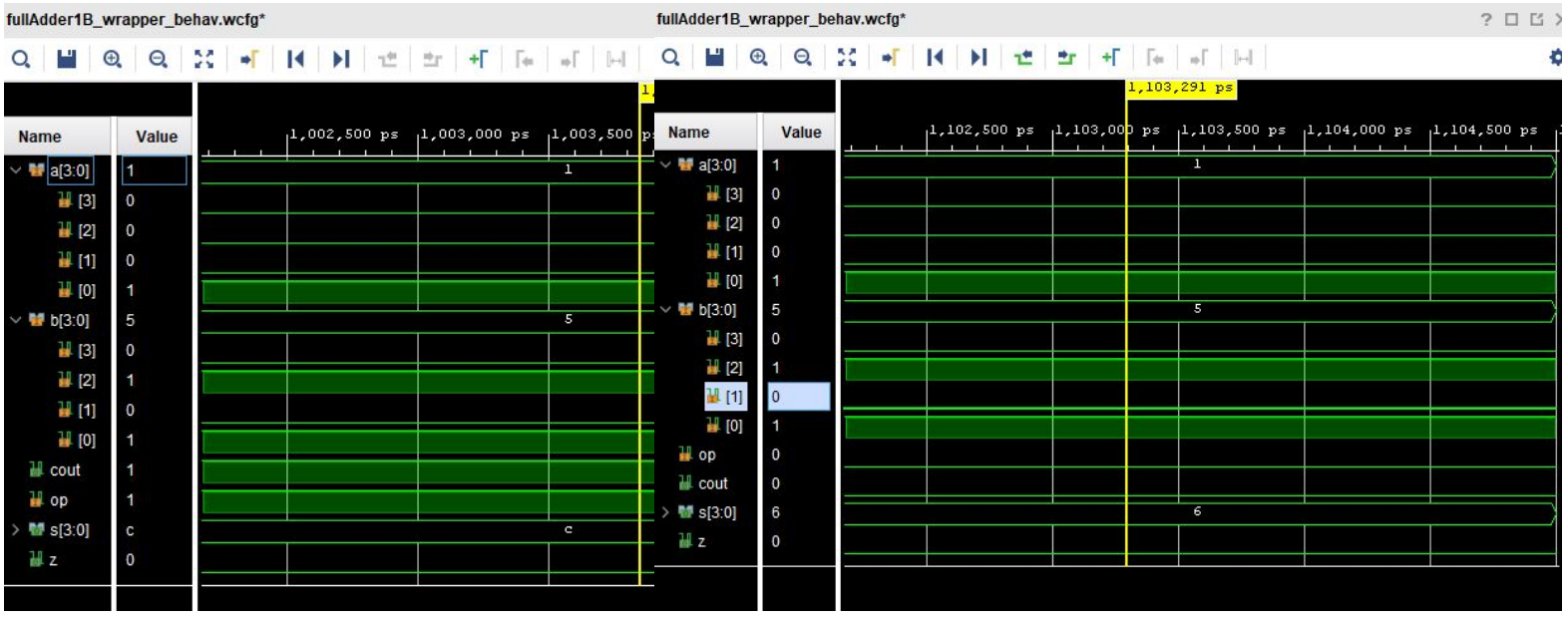
1) ***L'inversion des bits de B pour op=1*** : chaque bit de B entre avec ***op*** dans une porte *XOR*. La sortie **f** d'une porte *XOR* est : *f= bit'.op +bit.op'*. Donc pour **op =1**, si bit=0 f=1 et si bit =1 f=0.

2) ***Le fait que COUT=1 montre que, si on a une soustraction A-B, alors B>A:*** voyons l'explication en décimal: le complément à 2 d'un nombre A de 4 bit, est en décimal: 16 - Décimal A. Donc si on a 0<A<B<16 alors 16 - B +A <16. L'addition du complément à 2 de B à A ne donne pas une retenue, ce qui fait que ***"cout"*** du *FullAdder1B_4* a une valeur de 0, avec ***op=1*** dans une porte XOR, la sortie **COUT** prendra la valeur 1 aussi. Ceci dit, un **COUT**=0 lors d'une soustraction démontre qu'il y a un débordement lors de l'addition du CA2 de B à A.

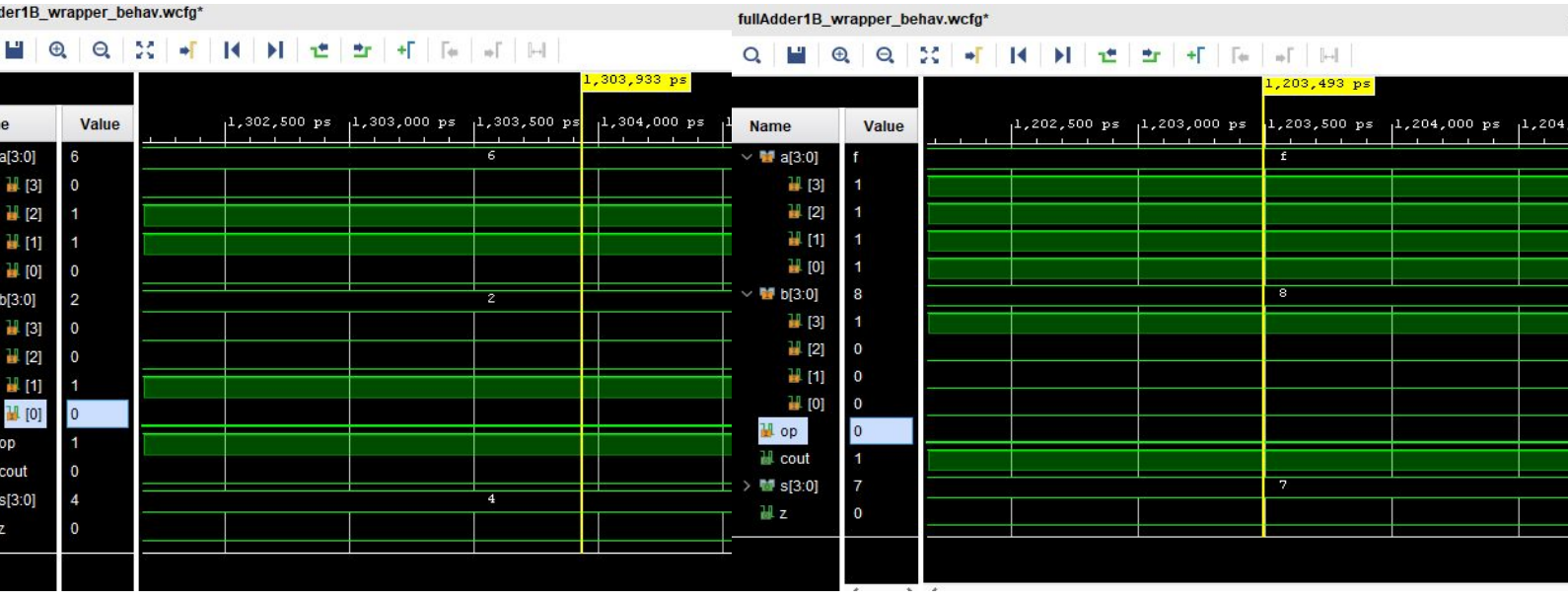
Finalement pour tester notre système (tests exhaustifs), nous avons choisi une multitude d'opérations de tous les cas possibles pour ce système: *résultat nul, débordement, soustraction avec **COUT=0**, soustraction avec **COUT=1**, etc.*

Voici dans les pages suivantes, quelques uns de nos tests exhaustifs:

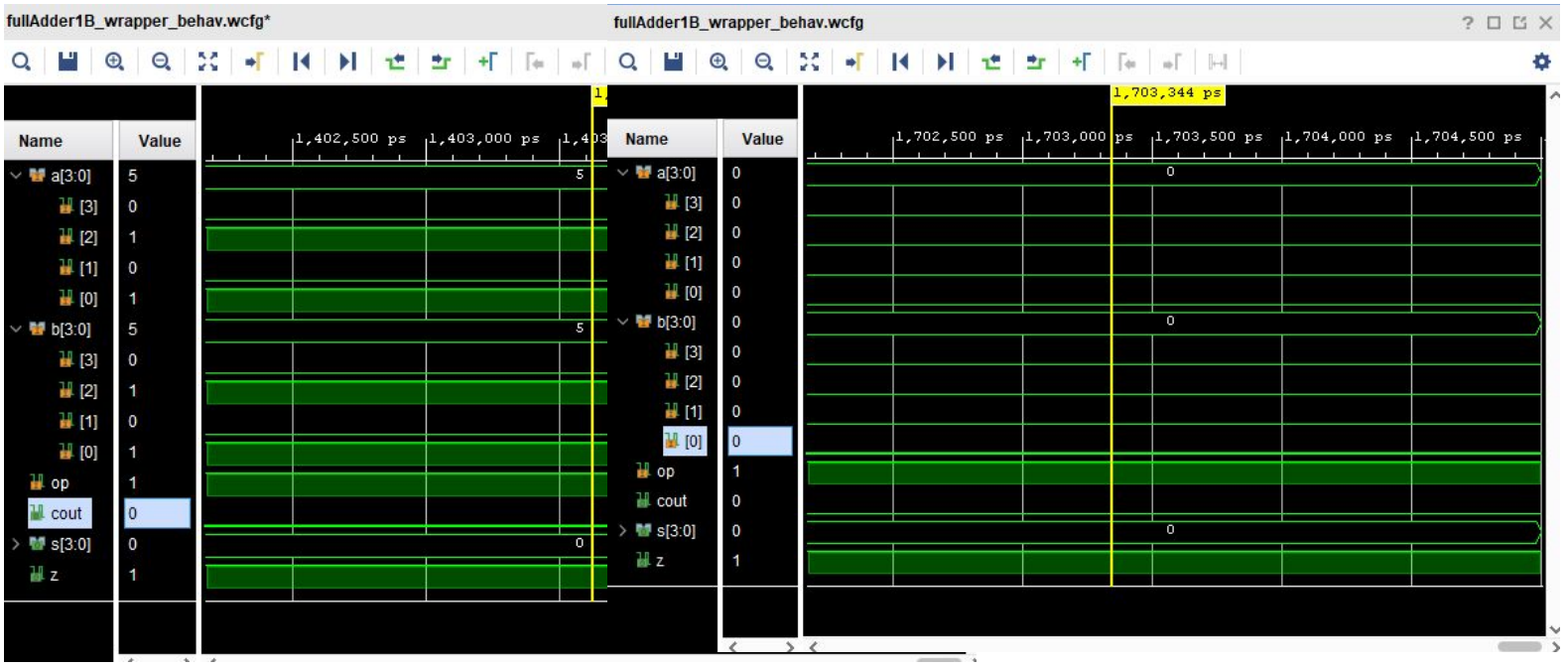
TEST NON EXHAUSTIF #1 et #6



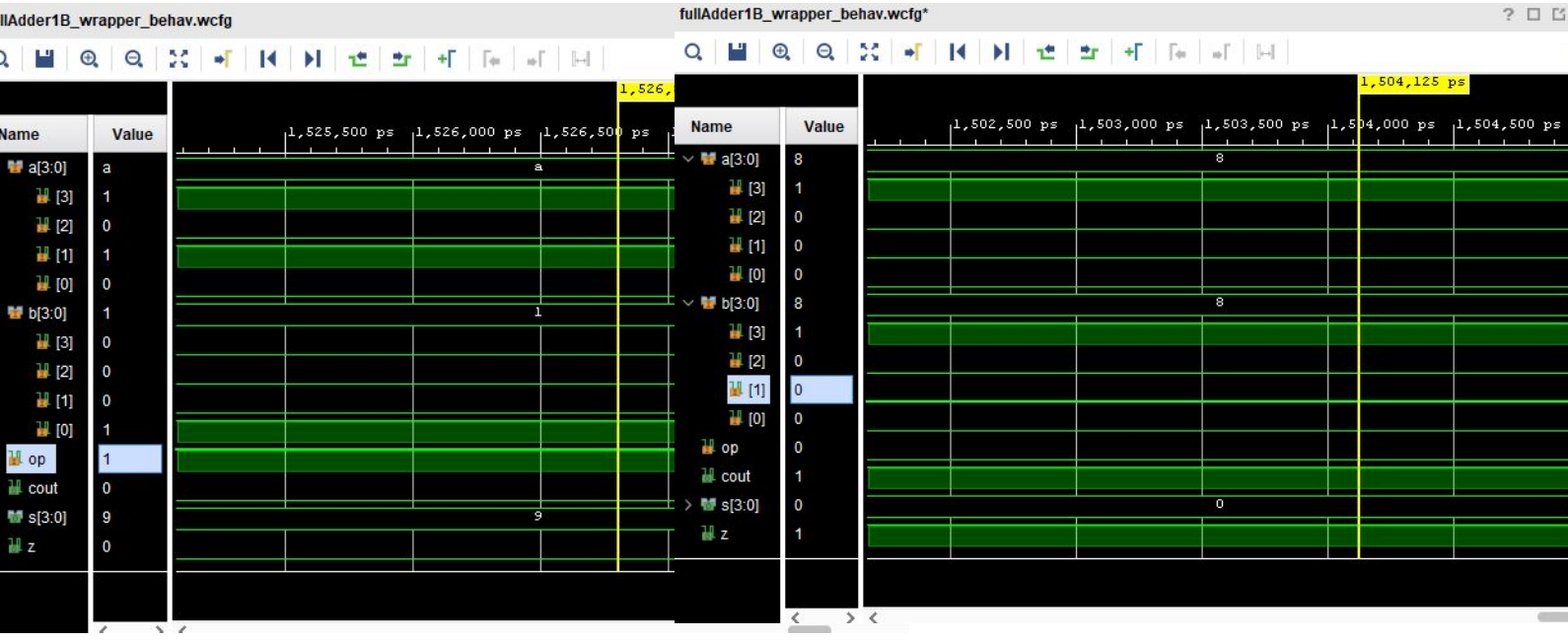
TEST NON EXHAUSTIF #5 et #2



TEST NON EXHAUSTIF #7 et #8



TEST NON EXHAUSTIF #11 et #10



3- Réponses aux questions de l'énoncé

Question 1: Nous avons réalisé le circuit additionneur sur 1 bit et remis sous format zip (lab 1). Il s'agit du *FullAdder1B*.

Question 2: Nous avons fait la simulation (text exhaustif) sur Vivado, et nous avons enregistré et remis le circuit sous format zip (lab 1).

Question 3: Nous avons aussi réalisé le circuit additionneur/soustracteur sur 4 bits et remis sous format zip (lab 1). Il s'agit du *FullAdder4B*.

Question 4:

Voici nos 13 tests non-exhaustif, avec leurs entrées, sorties et explications :

CAS	ENTRÉES	SORTIES(0x s)	EXPLICATION
#1	a: 0001 b; 0101 op:1	cout=1; s=c ; z=0	→ **0001 - 0101 = 0001 + 1010 + 1 = 1100** ● *cout=1(car a<b);* ● *s=c(hex) = 1100(bin) . CA2 de 0100 (car 1-5 =-4)* ● *z=0 :résultat non nul*
#2	a: 1111 b:1000 op:0	cout=1; s=7 ; z=0	→ **1111+1000 = 10111** ***(débordement)*** ● *cout =1 : retenue* ● *s=0111 (7 en hexadécimal, 4 bits de droite)* ● *z=0 (résultat non-nul)*
#3	a: 1111 b: 1111 op:0	cout=1; s=e ; z=0	→ **1111+1111 = 11110** ● s=1110 (E en hexadécimal); ● z=0 (résultat non nul) → **Similaire au Cas #2**
#4	a: 0010 b: 0110 op:1	cout=1 ; s=c; z=0	→ **Similaire au Cas#1**
#5	a: 0110 b: 0010 op:1	cout=0 ; s=4 ; z=0	→ **0110 - 0010= 0110+1101 +1 = 10100**. *En prenant les 4 bits à droite on aura : 0100 =4 en hexadécimal qui est 6-2 :(a-b)* ● *cout =0 car a>b.*
#6	a: 0001 b:0101 op :0	cout=0 ; s=6; z=0	→ **0001 + 0101 = 0110=6** ***(en hexadécimal)*** ● *Addition sans débordement : cout=0*
#7	a:0101 b:0101 op:1	cout=0 ; s=0 ; z=1	→ **0101 - 0101 = 0101 +1010** ● +1=10000, s=0000 donc z=1

			● cout =0 *(débordement lors de l'addition a+ ca2b)*
#8	a:0000 b:0000 op:1	cout=0 ; s=0 ; z=1	→ **0000 - 0000 = 0000+1111+1=10000** → **Similaire au Cas #7**
#9	a:0000 b: 0100 op:0	cout=0 ; s=4 ; z=0	→**Similaire au Cas #6**
#10	a:1000 b:1000 op:0	cout=1 ; s=0 ; z=1	→ **1000 + 1000 = 10000** ● s=0000 donc z=1 ● cout =1 retenue *(débordement)*
#11	a:1010 b:0001 op:1	cout=0 ; s=9 ; z=0	→ **1010 - 0001 = 1010+1110+1 =11001** ● s= 1001 = 9 ● cout =0 *(débordement a+ca2b)*
#12	a:0001 b:0010 op:1	cout=1 ; s=f ; z=0	→ **Similaire au Cas #1**

Question 5:
Pour le test exhaustif de notre additionneur/soustracteur sur 4 bits, nous avons utilisé le ***"force clock"*** et avons ajusté les entrées avec 9 périodes (en ns) différentes pour entièrement couvrir les 512 cas possibles *(2^8)*2 = 512: 16 possibilités de A * 16 possibilités de B * 2 possibilités de op)*.

Pour cela, nous avons tout d'abord choisis d'ajuster la variable ***"op"*** à notre valeur maximale, qui est 256 ns dans notre test, pour faciliter la vision du système (voir capture d'écran 1 à la **p.9**). Ainsi, d'un coté nous aurons la partie soustraction (***op=1),*** où le système fera une addition de complément à 2, et de l'autre, la partie addition (***op=0),*** où le système fera une simple addition de bits.

Pour le bus ***"a"***, nous avons respectivement ajusté les 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 ns, donc le ***0*** est à 16ns, ***1*** à 32ns, ***2*** à 64 ns, ***3*** à 128 ns. De cette manière, chaque combinaison de 9 bits sera unique parmis 512, non répétée , et occupant le même espace sur l'écran de visualisation.

De plus, dans la vue rapproché (voir capture d'écran 2 à la **p.11**), nous pouvons voir que les bus ***"a"*** et ***"b"*** prennent des valeurs correspondant à 4 bits en hexadécimal *(1,...,9,A,…,F)*, vu que dans notre cas, nous avons choisi de représenter numériquement ces bus en hexadécimal dans le simulateur.

CAPTURE 1 - Vue d'ensemble du test exhaustif

CAPTURE 2 - Vue rapproché du test exhaustif *(zoom in)*