

N7_SN_1A

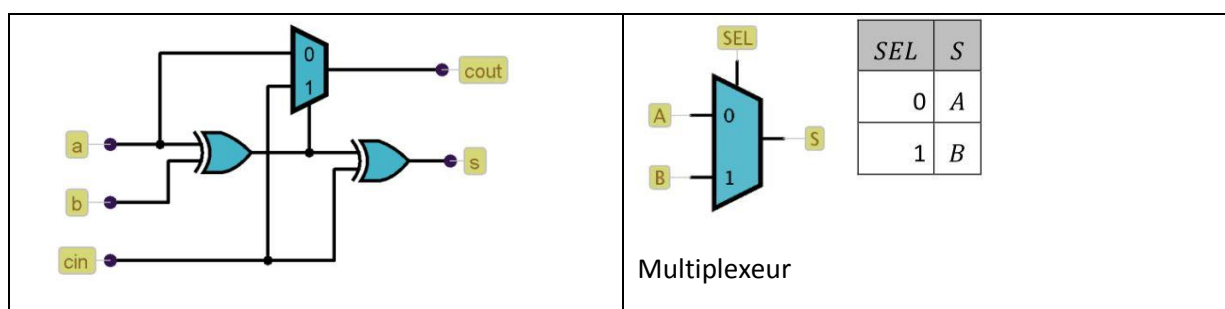
Architecture des ordinateurs - Semestre 5


TP1 :

Les circuits des questions 1 à 6 sont nécessaires pour les TP's suivants. Ils doivent être terminés, éventuellement en dehors des séances de TP.

Afin de pouvoir utiliser les tests automatiques, il faut utiliser les noms des modules et des signaux fournis dans le sujet en respectant la casse.

1- Soit le circuit ci-dessous. En déduire les équations de s et de cout.



La porte  réalise un ou exclusif entre les entrées a et b : $ouex = a * b + a / b$

Sur la plateforme « shdl.fr », écrire et tester le module fulladder.

```
module fulladder (a, b, cin : s, cout)
```

```
    s = ...
```

```
    cout = ...
```

```
end module
```

Bien vérifier qu'il réalise bien l'addition binaire entre les entrées a, b et cin.

2- Ecrire et tester le module adder8 (a[7..0], b[7..0], cin : s[7..0], cout), en utilisant le module fulladder testé dans l'étape précédente.

```
module adder8 (a[7..0], b[7..0], cin : s[7..0], cout)
```

```
    fulladder (a[0], b[0], cin : s[0], c0);
```

```
    ...
```

3- Ecrire et tester le module adder32(a[31..0], b[31..0], cin : s[31..0], cout)

4- Ecrire et tester le module addsub32(a[31..0], b[31..0], sub : s[31..0], C, V), qui, en utilisant **un seul module adder32**, permet de :

- réaliser $s[31..0] = a[31..0] + b[31..0]$ lorsque sub=0
- réaliser $s[31..0] = a[31..0] - b[31..0]$ lorsque sub=1
- et de positionner les indicateurs C et V en fonction du résultat obtenu

Rappel :

- C représente la retenue finale pour l'addition ou l'emprunt final pour la soustraction
- V représente le débordement sur le bit de signe


Tester les cas suivants, en interprétant le résultat (juste ou faux et pourquoi, à l'aide de V et de C) pour a et b non signés, et a et b signés. En déduire une règle générale qui permet d'interpréter le résultat en fonction de C et de V. Enregistrer les réponses dans un fichier portant le nom addsub32_resultats.

					<u>Interprétation du résultat</u> (OK / KO)	
a	b	C	V	S	<u>non signé</u>	<u>signé</u>
0111....1111	+ 0000.....0001	0	1	1000...0000	OK : $<2^{32}$	KO : Pos + Pos \rightarrow Neg
1111....1111	+ 1000.....0000	1	1			
1111....1111	+ 1111.....1111	1	0			
0111....1111	- 1000.....0000	1	1			
0100....0000	- 1111.....1111	1	0			
1000....0000	- 0111.....1111	0	1			

Nombre signés : résultat Faux si ?

Nombre non signés : résultat Faux si ?



En cliquant sur le bouton  (en haut à droite de la fenêtre de simulation), on peut charger un fichier de test, et tester de façon plus rapide et plus complète un circuit. Tester addsub32 en utilisant le fichier de test fourni « addsub32.tst », à copier depuis moodle dans un dossier local.

5- On aura besoin pour la suite de circuits appelés décodeurs, et qui permettent, pour un vecteur de N signaux (entrée), de générer 2^N sorties exclusives (une seule vraie à la fois) correspondant au code binaire (valeur) représenté par le vecteur en entrée.

- Décrire en shdl et tester le circuit decoder2to4(e[1..0] : s[3..0]), qui permettra de générer : s[0]=1 lorsque e[1..0]=00, s[1]=1 lorsque e[1..0]=01, ...
- En vous aidant de la table de vérité, trouver le lien entre un decoder3to8 et un decoder2to4 et en déduire la description shdl de decoder3to8 en utilisant un seul decoder2to4. Tester decoder3to8 en utilisant le fichier de test « decoder3to8.tst » fourni.
- Suivre le même raisonnement pour générer les modules decoder4to16, decoder5to32, et decoder6to64. Tester en utilisant les fichiers de test fournis.

6- Ecrire et tester le module ucmp1(a, b : sup, equ) qui effectue la comparaison entre les entrées a et b et fournit en sortie sup=1 si a>b et equ=1 si a=b.

Ecrire et tester progressivement les modules ucmp2, ucmp4, et ucmp8, qui effectuent la comparaison entre nombres non signés codés, respectivement, sur 2, 4 ou 8 bits.

7- En utilisant un module ucmp32, écrire et tester le module cmp32(a[3..0], b[3..0] : usup, ssup, equ) qui effectue la comparaison entre les entrées a et b et fournit en sortie usup=1 si a>b (non signés), ssup=1 si a>b (signés) et equ=1 si a=b.

8- Peut-on réaliser le module cmp32 en utilisant le module addsub32 ? Si oui, proposer une solution.