Modélisation et simulation VHDL semaine 4

Objectif(s)

- ✯ Un premier objectif est de décrire (en `VHDL`) et simuler un petit circuit dont le schéma est fourni.
- ✯ Dans un second temps, vous allez décrire et simuler un second circuit dont seulement une spécification assez abstraite vous est fournie.

Pour simuler vos circuits, vous allez utiliser l'outil **ghdl** et le visualiseur de chronogrames **gtkwave**. Ces deux outils sont sous licence **GPL** et peuvent donc être utilisés librement par vous.

Exercice(s)

Exercice 1 – Circuit Addaccu

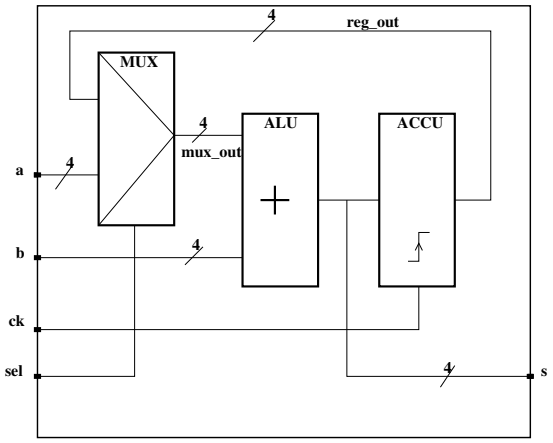

FIGURE 1 –
Schéma du circuit addaccu.

Question 1

Ecrivez le fichier `.vhdl` correspondant à ce circuit en partant des informations fournies en cours. Cette description doit être de type "zéro-délai" (pas de "clause After").

Vous allez pouvoir vérifier la syntaxe et **analyser** votre fichier grâce à la commande suivante :

```
> ghdl -a -v addacu.vhdl
```

Si l'analyse de votre circuit a réussi vous devez avoir obtenu un fichier `addaccu.o` correspondant au résultat de la compilation.

Question 2

Une fois la description comportementale compilée avec succès (sans aucune erreur), pour valider votre description

vous devez écrire un autre fichier VHDL pour tester votre circuit. Dans ce fichier que l'on appelle courament *test bench*, vous allez d'une part instancier votre votre circuit à tester et d'autre part écrire un process dont le rôle va être de fournir des valeurs pour les entrées du circuit à tester et éventuellement vérifier la valeur des sorties.

Votre circuit addaccu étant séquentiel vous allez notament devoir générer un signal d'horloge.

Compilez votre fichier *test bench* comme vous l'avez déjà fait avec votre circuit addaccu.vhdl puis réalisez l'élaboration de votre projet.

Commencez par quelques test simples de votre circuit.

```
ghdl -a -v addaccu_tb.vhdl
ghdl -e -v addaccu_tb
```

Simulez votre circuit.

```
ghdl -r addaccu_tb --vcd=addaccu.vcd
```

Visualisez avec gtkwave le résultat de votre simulation.

Question 3

Modifiez votre fichier *test bench* pour effectuer un test exhaustif de votre circuit, rendez-le auto-testant (ne plus visualiser les résultats avec gtkwave.

Exercice 2 – ALU pour l'ARM

Comme vous allez le voir dans les semaines à venir on simplifier cette ALU en restreignant ses fonctionnalités à 4 opérations de base : ADD avec retenue, AND, OR et XOR.

Cette *ALU* recoit ses deux opérandes codés sur 32 bits et fournit un résultat sur 32 bits. Elle reçoit également en entrée une retenue (cin) et fournit en sortie les 4 *flags* : Z, N, V et Cout. Cette *ALU* est exclusivement combinatoire, voici son interface :

```
entity Alu is
    port ( op1       : in Std_Logic_Vector(31 downto 0);
           op2       : in Std_Logic_Vector(31 downto 0);
           cin       : in Std_Logic;

           cmd_add   : in Std_Logic;
           cmd_and   : in Std_Logic;
           cmd_or    : in Std_Logic;
           cmd_xor   : in Std_Logic;

           res       : out Std_Logic_Vector(31 downto 0);
           cout      : out Std_Logic;
           z         : out Std_Logic;
           n         : out Std_Logic;
           v         : out Std_Logic;

           vdd       : in bit;
           vss       : in bit);
end Alu;
```

Les entrées cmd_ permettent de spécifier l'opération à effectuer, une seule d'entre elles ne doit être égale à 1, les autres doivent être égales à 0.

Question 1

Ecrivez le modèle .vhdl de l'*Alu*.

Question 2

Ecrivez un *test bench* permetant de tester cette *Alu*

