6GEI228 – Systèmes Digitaux

Laboratoire #9

Division en VHDL et Conversion BCD

Hiver 2018

1. Objectifs

- Apprendre à implémenter des algorithmes simples
- S'habituer à gérer des systèmes plus costauds

2. Méthodologie

Le but de ce laboratoire est renforcer votre expérience avec les systèmes digitaux en implémentant un système plus lourd et qui demande un travail de réflexion plus poussé. Il n'y a pas de nouveaux concepts à apprendre dans ce laboratoire. La partie principale du travail sera de décortiquer l'algorithme à implémenter et de réfléchir aux différentes composantes. Tout comme avec les autres laboratoires, la première partie sera un tutoriel tandis que la deuxième partie sera le gros travail de conception. Le tutoriel consiste en la conception d'un diviseur séquentiel qui donne en sortie un quotient et un reste. Dans la deuxième partie, vous utiliserez votre diviseur pour implémenter un système qui convertit un nombre de 8 bits en un nombre BCD.

3. Théorie et tutoriel

Dans la première partie du laboratoire, il sera question d'explorer un algorithme simple pour la division. Nous savons qu'il n'est pas possible d'implémenter une division facilement en VHDL sauf quand le diviseur est une puissance de 2. Cependant, il existe une approche intuitive qui peut être utile dans plusieurs systèmes et c'est justement cette approche que nous allons explorer aujourd'hui.

L'algorithme de division que nous allons voir aujourd'hui est basé sur la soustraction successive. Quand on divise un dividende A par un diviseur B (c'est-à-dire A/B), on obtient un quotient Q et un reste R. On pourrait l'exprimer d'une autre manière :

$$A = B \cdot Q + R$$

On pourrait décomposer la multiplication :

$$A = \underbrace{B + B + \ldots + B}_{Q \text{ fois}} + R$$

Et si on l'amenait à gauche, on aurait ceci :

$$A - \underbrace{B - B - \ldots - B}_{Q \text{ fois}} = R$$

Selon l'équation précédente, on pourrait comprendre que la division peut être vue comme étant une soustraction successive. Il suffirait de soustraire le diviseur du dividende successivement jusqu'à ce que le résultat soit plus petit que le diviseur. Le nombre de fois qu'on soustrait sera le quotient Q tandis que le résultat des soustractions sera le reste.

Pour illustrer le concept, prenons par exemple le cas où nous voulons diviser 31 par 7. Pour appliquer l'algorithme, on soustrait le diviseur du dividende jusqu'à ce que le résultat soit plus petit que le diviseur :

Je soustrais une première fois
31 – 7 = 24 : Le résultat (24) est plus grand que le diviseur (7)

Je soustrais une deuxième fois
24 - 7 = 17 : Le résultat (17) est plus grand que le diviseur (7)

Je soustrais une troisième fois
17 - 7 = 10 : Le résultat (10) est plus grand que le diviseur (7)

Je soustrais une quatrième fois
10 - 7 = 3 : Le résultat (3) est plus PETIT que le diviseur (7)

On a fait la soustraction 4 fois donc le quotient est de 4. Le résultat des soustractions est 3, ce qui veut dire que le reste est 3. La vérification mathématique se fait facilement :

$$4 \times 7 + 3 = 31$$

Maintenant que l'algorithme est compris, il est possible de commencer l'implémentation. Pour ce faire, il faut commencer par déterminer les ports d'entrée et de sortie. Dans ce laboratoire, nous allons vous forcer à implémenter un système séquentiel. Les ports d'entrée et de sortie devront donc inclure *clk*, *dividende*, *diviseur*, *quotient* et *reste*.

Pour les besoins du laboratoire, nous allons aussi vouloir avoir 2 autres signaux : un signal en entrée qui nous dit QUAND commencer la division (une entrée *start* (départ), par exemple) et un signal qui indique au monde extérieur quand la division est complétée (une sortie *stop* (fini) par exemple). Ces signaux sont importants quand on veut avoir un système qui se synchronise avec le diviseur (comme on va le voir dans la deuxième partie du laboratoire).

Pour ce laboratoire, nous allons prendre des nombres de 8 bits pour le dividende, le diviseur, le quotient et le reste. Avec cette information, il est déjà possible d'écrire le code VHDL pour l'entité du bloc VHDL.

Une fois que l'entité est complétée, il est possible de commencer la conception du diviseur. La conception du diviseur dans ce document se fera avec un langage semblable au français. Ce sera à vous de transformer le code décrit ici en code VHDL.

Nous savons que la division se fait en comptant le nombre de soustractions que nous faisons. Pour ce faire, nous pouvons utiliser un compteur qui augmente de 1 à chaque fois qu'on fait une soustraction. Ceci s'exprimerait de la façon suivante :

```
resultat = resultat - diviseur
compteur = compteur + 1
```

Le problème avec ce code est que le système va continuellement incrémenter le compteur et soustraire le résultat. Pour régler ce problème, il faut mettre des conditions : « si le résultat de la soustraction est plus petit que le diviseur, j'arrête ». Ou de façon équivalente, on pourrait dire : « si le résultat est plus grand ou égal au diviseur, je fais une autre soustraction et j'incrémente le compteur ».

```
Si resultat >= diviseur
        resultat = resultat - diviseur
        compteur = compteur + 1
```

Considérons maintenant ce qui se passerait si le résultat était plus petit que le diviseur. Quand le résultat de la soustraction est plus petit que le diviseur, ça indique la fin de la division. Dans ce cas, le résultat de la soustraction serait le *reste* tandis que la valeur du compteur serait égale au *quotient*.

```
Si resultat >= diviseur
        resultat = resultat - diviseur
        compteur = compteur + 1
SINON
        quotient = compteur
        reste = resultat
```

En regardant le code, il est possible de remarquer certains problèmes. Le plus important est surement le fait que nous n'avons jamais parlé des valeurs initiales de résultat et de

compteur. La valeur du compteur devrait être 0 tandis que la valeur du résultat devrait être égale au dividende au début de chaque division. Pour ce faire, il faudrait détecter quand *start* est actif et à ce moment, initialiser les valeurs de *dividende* et *compteur*.

```
Si start = 1
        resultat = dividende
        compteur = 0
SINON Si resultat >= diviseur
        resultat = resultat - diviseur
        compteur = compteur + 1
SINON
        quotient = compteur
        reste = resultat
```

Selon les objectifs du tutoriel, il manque encore une dernière partie : il faut indiquer quand la sortie est valide et quand elle ne l'est pas. On va considérer que la sortie est valide après que la division soit complétée. Ceci se produirait seulement si *start* n'est pas 1 et si le résultat est plus petit que le diviseur. On mettrait donc :

```
Si start = 1
         resultat = dividende
        compteur = 0
        fini = 0
SINON Si resultat >= diviseur
        resultat = resultat - diviseur
        compteur = compteur + 1
        fini = 0
SINON
        quotient = compteur
        reste = resultat
        fini = 1
```

Votre travail est maintenant de traduire ce pseudo-code (également appelé LDA (pour **L**angage de **D**escription d'**A**lgorithmes, qui est une façon de décrire un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier) en code VHDL. Si vous l'aviez bien écrit, votre système devrait donner les résultats de simulation suivants :

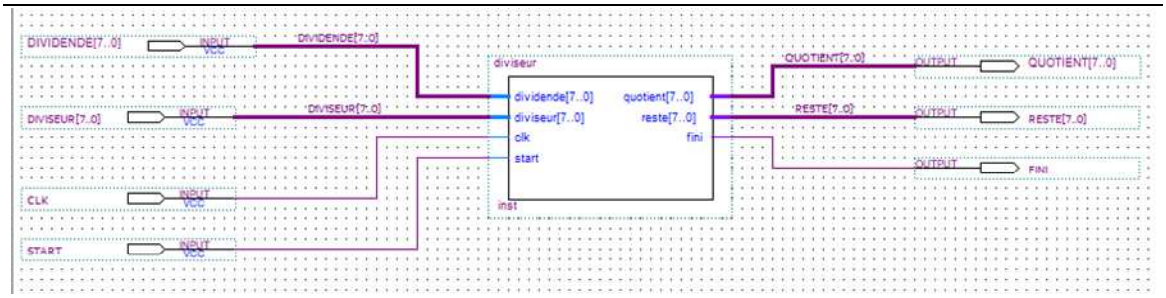

Figure 1. Schéma pour la division.

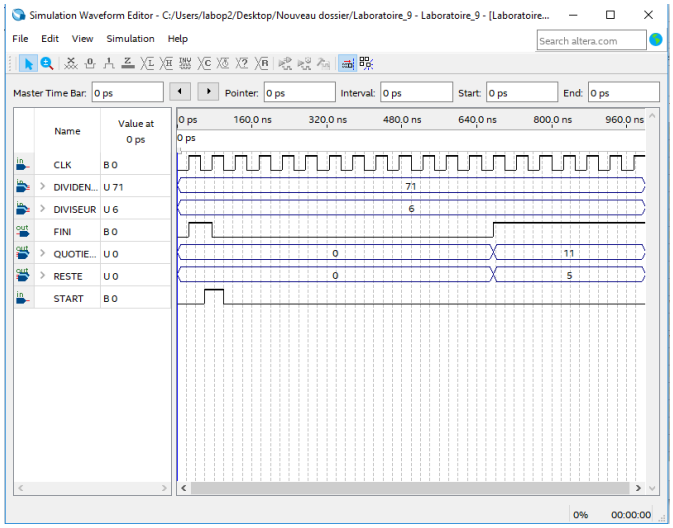

Figure 2. Simulation du diviseur.

Nous pouvons analyser les résultats pour s'assurer de bien comprendre le système. Au début de la simulation, rien ne se passe jusqu'à ce que le signal *start* monte à 1. A ce point, on fait une soustraction à chaque cycle d'horloge jusqu'à ce que le résultat soit plus petit que le diviseur. A ce point, le signal *fini* remonte à 1 pour indiquer qu'il a terminé de faire la division. Dans la simulation, on voulait diviser 23 par 5 ce qui donne effectivement un quotient de 4 et un reste de 3.

4. Travail demandé

Dans la deuxième partie du laboratoire, on va apprendre à convertir un nombre normal vers une forme BCD. Considérons le nombre 1190 en décimale que nous voulons convertir en BCD. Pour commencer, rappelons que chaque chiffre en décimale sera représenté par 4 bits allant de 0000 jusqu'à 1001 (les nombres 1010 à 1111 étant illégaux). 1190 deviendrait alors : 0001 0001 1001 0000

Pour obtenir ce format, nous proposons un algorithme utilisant la division par 10. Si nous prenions 1190 et que nous le divisions par 10, nous obtiendrons un quotient de 119 et un reste de 0. En divisant le quotient encore par 10, le quotient serait 11 et le reste serait 9.

En divisant encore le quotient par 10, le quotient donnerait 1 et le reste serait 1. Et finalement, en divisant encore par 10, le quotient serait 0 et le reste serait 1. En regroupant tous les restes, on obtiendrait 0911. En le lisant à l'envers, ça donnerait 1190.

L'algorithme serait donc de prendre le chiffre à convertir, de le diviser par 10 et de conserver le reste. Si le quotient était plus que 10, il faudrait continuer à diviser le quotient par 10. Ce deuxième reste nous donnerait notre deuxième chiffre BCD. Encore une fois, le quotient nous dirait s'il fallait continuer la division.

Pour le laboratoire, nous allons nous pencher sur l'implémentation d'un convertisseur BCD de 8 bits. C'est-à-dire que le chiffre en entrée peut avoir une valeur entre 0 et 255. La sortie devrait se retrouver sur 12 bits différents qui représentent les 3 chiffres en BCD. L'entrée de 8 bits devrait être contrôlée par les commutateurs sur la plaquette et un autre commutateur devrait être utilisé pour indiquer au système quand faire la conversion.

Une possibilité serait de faire une machine à états pour gérer les opérations. Par exemple, «Je suis à l'état initial et j'attends un signal en entrée pour me dire de faire la conversion. Quand je le reçois je passe à un état qui envoie 1 à l'entrée start du diviseur. Je passe ensuite à l'état de la division. Je reste dans cet état jusqu'à ce que je reçoive le signal du module qui me dit que c'est fini. Je passe donc…»

Soyez conscients que vous aurez surement de la difficulté à penser à un concept qui fonctionne, à générer les signaux à temps pour que le système fonctionne bien et à faire le débogage. C'est tout à fait normal. Profitez de l'occasion pour développer vos habiletés.

Bon travail