Architecture, Système, Réseau 2017-2018
{alice.pellet___mary,fabrice.mouhartem}@ens-lyon.fr, florent.de-dinechin@insa-lyon.fr

ASR1, TD/TP : circuits séquentiels

On suppose que ce TP est réalisés avec *Logisim* sous Linux : vous pouvez utiliser `java -jar logisim.jar` depuis un terminal (remplacer `logisim` par le nom effectif de l'archive) pour lancer le logiciel.

Exercice 1: Verrou, flip-flop et chenillard :

1. Dans un sous-circuit `latch`, implantez un verrou à l'aide d'un multiplexeur 2 vers 1. Le verrou doit être passant sur le niveau bas de l'horloge, et verrouiller sur le niveau haut.
2. En plus du signal d'horloge, ajoutez une entrée `reset` qui, au niveau haut, force la valeur verrouillée à 0 de façon asynchrone.
3. Dans un sous-circuit `flip-flop`, implantez une bascule à l'aide de deux composants `latch`, et d'une porte de base. Votre bascule doit être régie par le front montant de l'horloge. En plus du signal d'horloge, ajoutez une entrée `reset` afin de forcer la valeur stockée à 0 de façon synchrone.
4. Dans le circuit `main`, réalisez un chenillard à 5 leds, avec une seule led allumée à la fois. Votre circuit doit comporter une horloge, ainsi qu'un bouton `start` :
   — lorsque `start` = 0 en fin d'un cycle d'horloge (front montant), le circuit est ré-initialisé (seule la première led est allumée, et le circuit est prêt à redémarrer),
   — lorsque `start` = 1 la led allumée avance dans le chenillard.
5. Les spécifications du circuit ne sont pas très précises... Faites donc un chronogrammes pour bien décrire le comportement au démarrage du circuit, puis sur une dizaine de cycles.

Exercice 2: Registre, compteur, et banc de registres :

1. En utilisant les composants flip-flop de la bibliothèque de Logisim (pour changer, disons qu'ils seront régis par le front descendant de l'horloge), construisez un circuit `reg4` réalisant un registre 4 bits. Vous doterez votre circuit d'une entrée `reset` synchrone (remise à 0 du registre au front descendant de l'horloge).
2. A l'aide de `reg4` et du d'un additionneur de la bibliothèque, construisez un compteur 4 bits `cpt4`. Votre compteur comportera une entrée `reset` synchrone (remise à 0 du compteur). Testez-le *in situ*.
3. Modifiez `cpt4` de façon à pouvoir construire un compteur 8 bits à l'aide de deux composants `cpt4`. Réalisez ce circuit dans votre circuit `main`, et testez-le.
4. En utilisant votre composant `reg4`, construisez maintenant un circuit `regfile4`, réalisant un banc de 4 registres 4 bits. Votre composant devra présenter deux ports de lecture des registres, et un port d'écriture. Réfléchissez aux signaux d'entrée et de sortie nécessaires en cours de route !

Exercice 3: Un multiplieur séquentiel :

En C par exemple, on peut implanter de la façon suivante l'algorithme de multiplication de la petite école pour multiplier deux entiers naturels `a_init` et `b_init` sur 8 bits, et obtenir le résultat sur 16 bits.

```
uint16_t mul(uint8_t a_init, uint8_t b_init) {
  uint16_t ax = a_init;
  uint8_t   b = b_init;
  uint16_t  c = 0;

  while(b != 0) {
    if(b & 0x01) c += ax; // b & 0x01 est le bit de poids faible de b
    ax <<= 1;             // décalage d'un bit à gauche
    b  >>= 1;             // décalage d'un bit à droite
  }
  return c;
}
```

Le but est d'implanter cet algorithme sous la forme d'un circuit séquentiel dans Logisim. Votre circuit prendra en entrée les entiers naturels `a_init` et `b_init` sur 8 bits, un signal `run` et un signal d'horloge. Il produira en sortie la valeur « courante » de `c`, ainsi qu'un signal `finished`, initialement à 0, et qui passera à 1 dès que le calcul est terminé.

Exercice 4: Comparaison de circuits combinatoires :

Récupérez le fichier `test_add4.circ` : ce fichier contient une tentative de réalisation d'un additionneur 4 bits (sous-circuit `add4`). Pour toutes les entrées possibles, on souhaite comparer les sorties de `add4` avec celles produites

par un additionneur de la librairie de Logisim. Si une différence existe, on pourra présumer qu'il y a une erreur dans l'implantation de `add4`.

L'idée est d'utiliser la simulation en ligne de commande pour effectuer la comparaison : parcourez la documentation de *Command Line Verification* dans le documentation de Logisim. En résumé, l'idée est de pouvoir faire tourner la simulation sans interagir avec Logisim : la simulation produit un tableau qui contient les valeurs de toutes les sorties du circuit `main`. La simulation s'arrête lors du cycle où une sortie `halt` prend la valeur 1.

1. Dans le circuit principal (`main`), on fournit une partie de l'infrastructure pour effectuer la comparaison : combien d'entrées distinctes faut-il tester pour être sûr d'avoir passé en revue toutes les entrées possibles ? Est-ce que cela correspond bien à ce qui est prévu dans `main` ?
2. Complétez le circuit `main`, puis lancer la simulation en ligne de commande ; vos commandes peuvent ressembler à :

   ```
   $ java -jar logisim.jar test_add4.circ -tty table > simu.out
   $ less simu.out
   ```

   Comment détectez vous que le sous-circuit `add4` comporte effectivement une erreur ?
3. Corrigez l'erreur qui produit le mauvais fonctionnement de `add4`. Relancez la simulation pour vous assurer que vous avez bien détecté tous les problèmes : comment faire pour vérifier que tous les tests ont eu lieu avec succès ?