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

ASR1, TD2 : circuits combinatoires

On suppose que ce TP est réalisés sous Linux [1]. Créez un répertoire pour le TP, puis ajoutez dans votre navigateur un favori vers la page de Logisim : `http://www.cburch.com/logisim/`. Télécharger depuis cette page l'archive `.jar` de Logisim dans votre répertoire de TP : vous utiliserez la commande `java -jar archive.jar` depuis un terminal (remplacer `archive` par le nom de l'archive) pour lancer le logiciel.

Exercice 1: Introduction à Logisim :

Commencez par parcourir rapidement le tutorial fourni dans l'aide : il vous est d'ailleurs recommandé de vous référer fréquemment à l'aide de Logisim, qui est très bien faite! Commencez ensuite cet exercice dans un nouveau fichier, `mux.circ` par exemple.

1.1) Dans le circuit `main`, réalisez un multiplexer 2 bits vers 1 bits, et testez le. Vous utiliserez pour cela des portes `AND` à 2 entrées, en inversant certaines entrées, ainsi qu'une porte `OR`.

1.2) Encapsulez votre multiplexeur 2 vers 1 dans un sous-circuit que vous appellerez `Mux21`. Instanciez ensuite `Mux21` dans `main`, et testez le.

1.3) En utilisant des portes `AND` à 3 entrées, ainsi qu'une grosse porte `OR` à 8 entrées, réalisez dans un sous-circuit `Dec38` un décodeur 3 bits vers 8 bits .

1.4) En utilisant votre sous-circuit `Dec38`, réalisez dans `main` un multiplexeur 8 bits vers 1 bit. Utilisez le composant `Splitter` pour pouvoir spécifier l'entrée sélectionnée à l'aide d'un vecteur sur 3 bits.

1.5) Réalisez un autre multiplexeur 8 bits vers 1 bit, mais cette fois ci en utilisant comme brique de base votre multiplexeur 2 bits vers 1 bits.

Exercice 2: Une unité arithmétique et logique :

Commencez cet exercice dans un nouveau fichier, disons `alu.circ`. Le but est de construire en Logisim une petite (ALU *Arithmetic and Logic Unit*) 8 bits, capable d'effectuer la conjonction et la négation bit-à-bit, ainsi que l'addition et la soustraction en complément à deux.

2.1) Rappelez la construction de l'additionneur à partir de cellules *full adder*. Ecrivez la table de vérité du *full adder*.

2.2) Construisez, à base de portes de bases, un sous-circuit `FA` qui implémente le *full adder*, en suivant stupidement la méthodologie automatique vue en cours.

2.3) Créez un sous-circuit `addsub`, contenant un additioneur/soustracteur en complément à deux sur 8 bits. Le circuit :
- prend en entrée deux entiers `a` et `b`, ainsi qu'une entrée `sub`
- produit en sortie `s`, et un indicateur de dépassement de capacité `dep`.

En l'absence de dépassement de capacité, `s` prend pour valeur `a` + `b` si `sub` = 0, `a` − `b` si `sub` = 1. L'indicateur `dep` prend pour valeur 1 en cas de dépassement de capacité, 0 sinon.

2.4) Réalisez ensuite l'ALU dans un sous-circuit `ALU`. Le circuit :
- prend en entrée deux entiers `a` et `b`, et qu'une entrée de contrôle `c` ;
- produit en sortie `r`, et un indicateur de dépassement de capacité `dep`.

Les valeurs produites dépendent de `c` :
- si `c` = (00), alors `r` = $\overline{\texttt{a}}$, et `dep` = 0,
- si `c` = (01), alors `r` = a&b, et `dep` = 0,
- si `c` = (10), alors `r` = `a` + `b` et `dep` indique les dépassements,
- si `c` = (11), alors `r` = `a` − `b` et `dep` indique les dépassements,

Utilisez votre `addsub`, et les composants fournis par la bibliothèque de Logisim (`Splitter`,...). Vous pouvez aussi utiliser l'outil `Analyze`, via le menu `Projet`, pour générer automatiquement de petits circuits combinatoires à partir d'une expression logique (utile pour l'indicateur de dépassement de capacité, même si ce n'est pas indispensable...).

1. Pour la Debian et ses dérivées il y a même un paquet `logisim`.

Exercice 3: Retenue anticipée : En cours, vous avez (peut-être) vu comment presque diviser par 2 le délai d'une addition 32 bits. Sinon, retrouvez-le.

3.1) En faisant une récurrence sur cette idée, quelles sont les complexités en temps et en espace (nombre de portes) de l'addition de n bits ? Et pour l'additionneur naïf ?

3.2) Réalisez en Logisim un additionneur 8 bits à propagation de retenue anticipée en utilisant des additionneurs 4 bits de la librairie.

3.3) Sur le même principe, réalisez ensuite un additionneur 16 bits.

Exercice 4: Multiplication binaire :

4.1) Posez une multiplication en binaire. Observez qu'on peut faire toutes les multiplications bit à bit en parallèle. Essayez d'en déduire les complexités en temps et en espace de la multiplication binaire.