Programmation IV Série 8 10 mai 2006

## Introduction

Le but de cette série est de réaliser un simulateur de circuits logiques simple. Ce simulateur est un cas particulier de simulateur basé sur des *événements discrets*.

La structure de données centrale du simulateur est *l'agenda*. L'agenda mémorise l'ensemble des *actions* qui auront lieu dans le futur, triées en fonction de leur heure de déclenchement. En plus de l'agenda, le simulateur garde le *temps courant de simulation*.

La simulation s'effectue en extrayant la première action $a$ de l'agenda, en avançant le temps de simulation jusqu'au temps de déclenchement de $a$, en déclenchant $a$ puis en recommençant jusqu'à ce que l'agenda soit vide.

Les circuits simulés par notre programme sont composés de *fils* et de *composants*. Un fil permet de transporter un signal digital, qui peut être 0 (représenté par `false`) ou 1 (représenté par `true`). Un composant est connecté à des fils d'entrée et de sortie, et la valeur qu'il place sur ses fils de sortie est une fonction des valeurs se trouvant sur ses fils d'entrée. Le changement des valeurs se trouvant sur les fils d'entrée ne produit pas instantanément un changement des valeurs sur les fils de sortie : chaque composant a un *délai* qui lui est propre.

À chaque fil est associé un ensemble de fonctions, nommées également *actions*, qui sont activées à chaque changement de la valeur transportée par le fil.

Pour cette série nous utiliserons trois composants de base : un composant NON, un composant ET et un composant OU. Tous nos circuits seront composés à partir de ces pièces de base.

## Exercice 1

Une solution partielle au problème est fournie sur la page Web du cours. Commencez par compléter la classe `Simulator` en écrivant les fonctions `next` et `afterDelay`. La fonction `next` retire la première action de l'agenda, l'effectue et met à jour le temps courant ; `afterDelay` stocke dans l'agenda une action à effectuer après un certain délai, spécifié à partir du temps courant. **Attention :** si plusieurs actions dans l'agenda ont le même temps de déclenchement, il *faut* qu'elles y soit stockées dans l'ordre dans lequel elles ont été ajoutées.

Une fois la classe `Simulator` complétée, vous pouvez la tester en simulant un circuit quelconque composé des portes NON et ET fournies.

## Exercice 2

Écrivez la méthode `orGate` dans la classe `CircuitSimulator` qui réalise une porte OU. Faites deux versions de cette porte : la première similaire à la porte ET, la

seconde définie uniquement en termes des portes ET et NON.

## Exercice 3

Définissez une fonction `demux` qui réalise un démultiplexeur à $n$ fils de contrôle (et $2^n$ sorties), tel que celui présenté à la figure 1. Pour mémoire, un démultiplexeur aiguille un signal d'entrée sur une sortie parmi $2^n$ en fonction de $n$ signaux de contrôle. La fonction `demux` a le profil suivant :

```
def demux(in: Wire, c: List[Wire], out: List[Wire]): unit
```

On admet ici que les listes de fils de contrôle (`c`) et de sortie (`out`) sont triées par index décroissant. C'est-à-dire qu'en tête de la liste `c` se trouve le fil de contrôle d'indice $n-1$ et en tête de la liste `out` se trouve le fil de sortie d'indice $2^n - 1$.

Indication : pensez récursif, et songez au fait que le démultiplexeur le plus simple qui soit comporte 0 fils de contrôle et 1 fil de sortie.

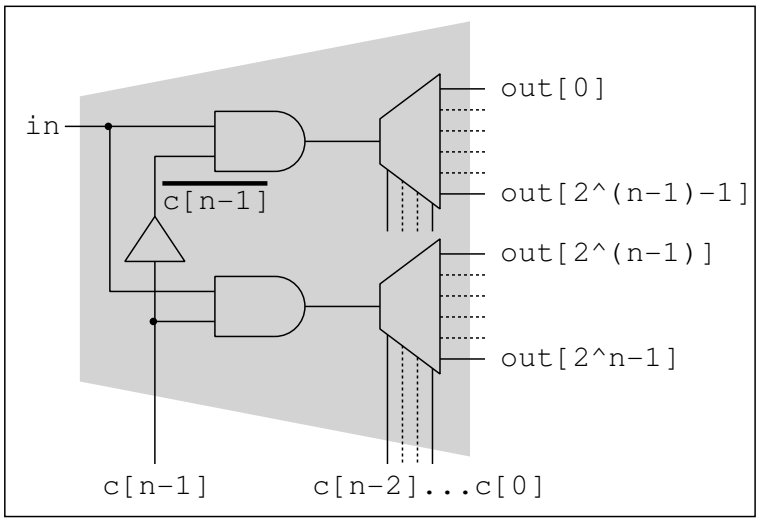


FIG. 1 – Démultiplexeur à $n$ entrées