# Simulateur de netlists

Mathieu Fehr

## 1 Utilisation

Pour des raisons d'efficacité, le programme n'est pas un simulateur mais un compilateur de netlist.

Le compilateur est déjà sous forme d'executable, mais il est toujours possible de lire le code source situé dans le dossier src, et de compiler ce code source via la commande "make all".
Le programme se nomme "compilateur_netlist" et s'execute par la commande "./compilateur_netlist file.net output" pour compiler la netlist contenue dans le fichier file.net, et créer le programme output qui va simuler la netlist, et output.cpp qui va être le fichier source c++ de l'executable.

L'execution de output se lance sans paramètres, et le programme demande au fur et a mesure les valeurs des variables d'entrées, les fichiers dans lesquels sont stockés les ROM, et le nombre d'itérations du circuit à effectuer.
Un fichier contant une ROM est un fichier contenant des entiers écrits en binaire et de même taille, avec un entier écrit par ligne, sans espaces (rom1 et rom2 situés dans le dossier tests sont des exemples de tels fichiers).
Lors de la demande de la valeur d'une variable, il est spécifié la taille de celle-ci (c'est à dire si c'est un fil ou une nappe de fils contenant x fils). Il faut entrer la valeur en binaire. Il faut noter qu'il n'est pas nécessaire de noter les 0 des poids les plus forts (donner la valeur 0001 aura le même effet que donner la valeur 01), et si plus de chiffres sont données que prévu, seuls les bits les plus faibles seront lus.
Il faut aussi savoir qu'il a été pris pour convention que le bit de poids faible correspond au fil numéro 0 dans une nappe de fils. Le test du n-adder renvoit donc le résultat dans l'ordre opposé.

## 2 Spécifications

Plusieurs conventions ont été prises dans la lecture des netlists, et il convient donc de les préciser.

### 2.1 Conventions sur les nappes de fils

Une nappe de fil de taille 1 est un fil.
Toutes les opérations booléennes (comme l'opération AND ou OR) ont été étendues aux nappes de fils, en effectuant des opérations bits à bits.

### 2.2 Conventions sur les RAM, les ROM et les registres

Premièrement, l'écriture sur la RAM et sur les registres s'effectue à la fin d'un cycle de calcul. La lecture sur une RAM et sur un registre donne alors la valeur qu'elle possédait au cycle précédent.
De ce fait, la RAM et les registres sont initialisés à la valeur 0 avant la simulation du circuit.

Un circuit peut avoir plusieurs RAM ainsi que plusieurs ROM, et chaque adressage est lié à une RAM et une ROM distinctes. Ainsi, la lecture de la ROM se faisant avec un adressage sur 2 bits se situe sur une autre ROM que lors de la lecture de la ROM avec un adressage sur 3 bits.

Lorsqu'un fichier ROM est donné au simulateur, si le nombre d'élements du fichier est inférieur à sa capacité maximale (vis à vis de l'adressage), alors les autres éléments auront la valeur 0 attribué.

# 3 Structure du programme

Le programme est enitèrement écrit en c++11.
Il est composé de deux parties majeures :

- Le parser associé au lexer, qui va lire un fichier .net et trier les données.
- Le compilateur, qui va écrire un fichier c++ à partir des données, puis qui va le compiler à l'aide de gcc pour produire un programme permettant de simuler la netlist.

Le programme est divisé en plusieurs classes.

- Var et ses dérivées représentent un fil (qui peut être constante (comme 1 où 0)) ou une nappe de fils.
- Expression et ses dérivées représentent les portes logiques du circuit, ainsi que les autres opérations utilisées dans les netlists (comme CONCAT, ou encore MUX).
- Program contient les données structurées de la netlist, et compile le fichier c++.
- Parser contient le parser, qui va récupérer les données depuis le fichier codant la netlist
- Graph permet de créer un graphe et de le trier topologiquement.

La compilation d'une netlist se fait en plusieurs étapes.

## 3.1 Le parsing

Premièrement, un lexer et un parser sont utilisés pour lire le fichier .net contenant la netlist.

Le lexer utilisé provient de la bibliothèque lexertl et le parser est situé dans les fichiers Parser.cpp et Parser.hpp du dossier parser.
Les tokens du lexer permettent de créer un objet de la classe Program, en créant un à un les objets des classes Expression et Var.

## 3.2 Le tri des instructions

Ensuite, les instructions de la netlist sont triés dans le programme (dans la classe Program), pour que la valeur de sortie des portes logiques ne soient calculées après que les entrées sont calculées.

Le programme renvoie une erreur si une boucle apparait dans le circuit, sauf si une telle boucle contient une opération sur un registre, une RAM, ou une ROM.
Le tri se fait via un tri topologique, grace a la classe Graph.

## 3.3 Le compilateur

Finalement, la classe Program se charge de compiler les instructions en c++.

Chaque fil et nappe de fils est stocké dans une variable qui est soit un bool, soit un entier non signé, qui ne peut pas dépasser 64 bits.
Le fichier c++ est ensuite compilé en un executable via le compilateur gcc, en utilisant l'optimisation -03 du compilateur.