TP 4 : Mémoires (à rendre à la fin de la séance)

Rappels

La mémoire est un composant permettant de stocker des valeurs binaires. On distingue les mémoires à lecture seule dites ROM (Read-Only Memory) et à lecture/écriture dites RAM (Random-Access Memory). Le contenu d'une ROM est initialisée à l'usinage et par conséquent persistent. Au contraire, la RAM est volatile, c'est à dire que son contenu est perdu lorsqu'elle n'est plus alimentée.

Il existe des mémoires *statiques* (SRAM) et *dynamiques* (DRAM) distinguées par leur technique de fabrication. La mémoire *statique* est caractérisée par des opérations rapides et par conséquent un coût onéreux, on l'utilise donc pour implanter des caches. La mémoire *dynamique* est caractérisée par des opérations lentes et par conséquent un coût bon marché, on l'utilise donc pour implanter la mémoire principale.

Chaque élément de mémoire dynamique est formé d'un *condensateur* et d'un *transistor de commande* (cf. figure ci-dessous). La ligne A est appelée ligne de commande ou ligne d'adresse. La ligne B est la ligne de donnée sur laquelle est lu ou écrit le bit d'information. Le bit d'information est représenté par la charge du condensateur. Lorsque la ligne de commande est à 0, le condensateur est isolé de la ligne de donnée et la charge reste prisonnière du condensateur. Au contraire, lorsque la ligne de commande est à 1, on peut lire le bit en détectant la charge ou écrire un nouveau bit en forçant la ligne de donnée à une valeur.

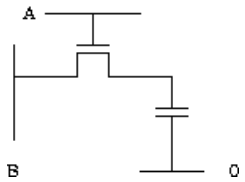

FIGURE 1 – Mémoire dynamique

L'élément de base de la mémoire s'appelle un *point-mémoire* (et correspondra pour nous à une bascule). Un regroupement de points-mémoire forme un *bloc-mémoire*, et permet de stocker un *mot* (*e.g.* un bloc de 8 bascules permet de stocker un mot de 8 bits, c'est-à-dire un octet). Dans la suite de ce TP, nous nous intéresserons à réaliser une **mémoire statique**. À la place d'un *transistor de commande* (composant analogique), nous utiliserons un **controlled-buffer** (composant logique).

Note : Un solution partielle documentée et propre sera bien évidemment préférable à une solution globale mais illisible.

La RAM

ouf, c'est pas ceux des impôts.

Le but de cet exercice est de construire un bloc-mémoire de type DRAM (Dynamique Random Access Memory), constitué par 4 mots de 4 bits chacun.

On pourra utiliser les bascules D de `Logisim`. On rappelle que celles-ci disposent :

- de l'entrée usuelle D ;
- d'une entrée *clock* (représentée par un triangle), telle que la mise à jour se fait sur **valeur haute** (1) ;
- d'une entrée *clear* qui permet d'initialiser la bascule à zéro lorsque clear vaut 1 ;
- d'une entrée *enable* qui permet la mise à jour de la bascule uniquement si enable vaut 1 ;

- des deux sorties usuelles Q et $\overline{Q}$.

1. Construire un module point-mémoire (ou registre 1 bit) à partir d'une *bascule latch D*. Le circuit sera asynchrone (c'est-à-dire sans horloge) et devra avoir en entrée :
   - D ;
   - WE permettant de choisir le mode (0=lecture/1=écriture) ;
   - CS (*chip select*) permettant d'activer (1) ou désactiver (0) le point-mémoire ;

   et en sortie :
   - Q.
2. En utilisant le stable à trois états (le *controlled-buffer)* et en réutilisant le module précédent, réaliser un registre 1 bit tel qu'une seule connexion est utilisée **à la fois pour l'entrée et pour la sortie**. L'entrée-sortie sera représentée par un pin ($Output = No, Threestate = Yes$). Avec une bonne utilisation du controlled-buffer, l'activation de CS lors du mode lecture ($WE = 0$) "forcera" en sortie la valeur en mémoire.
3. Adapter le circuit précédent pour avoir un mot de 4 bits.
4. Adapter le circuit précédent pour constituer une RAM de 4×4 bits. L'adressage se fera par des commandes extérieures, utilisées dans un module de type multiplexeur [1], qui permettra de choisir le mot à lire ou écrire parmi les quatre mots. La commande CS (*chip select*) permettra, en mode lecture, de laisser passer les sorties (celles-ci forceront ainsi la valeur des connexions communes d'entrée/sorties). Cette commande CS permet notamment de sélectionner une RAM parmi plusieurs en parallèle. Au final, le circuit devra donc comporter :
   - les quatre connexions d'entrées/sorties ;
   - les commandes d'adresses ;
   - la commande WE de lecture/écriture ;
   - la commande CS (*chip select*) qui permet de selectionner le boitîer ;

Note : `Logisim` ne permet pas d'avoir des interfaces in/out pour les modules, il s'agira donc d'adapter le même circuit à partir de la question 2.

Pile

En(e(rg({i}ze))r)

Dans cette exercice vous allez modéliser une pile. Pour cela vous devez utiliser le module RAM 8×8 de `Logisim` (8 bits d'adresse et 8 bits de donnée par adresse). Ce module comporte les 3 entrées comme dans l'exercice précédent : CS, WE ; une entrée 8 bits A qui correspond aux adresses des blocs mémoire ; une sortie 8 bits D qui représente la sortie ou l'entrée en fonction des valeurs des CS et WE.

1. Réalisez un module compteur/décompteur modulo 256 synchronisé sur une horloge. Le module doit avoir une entrée INC qui, à chaque front montant, compte lorsque $INC = 1$ et qui décompte sinon. Vous êtes encouragés à réutiliser un additionneur et un registre (sur 8 bits tous les deux) de `Logisim`.
2. Utilisez le module précédent et la RAM (asynchrone) de `Logisim` pour simuler le fonctionnement d'une pile. Le circuit comportera une seule sortie S sur 8 bits et deux entrées :
   - Empiler/Dépiler sur 1 bit
   - V sur 8 bits

   L'opération Empiler correspond à : la valeur V (définie par l'utilisateur) est écrite à l'adresse mémoire courante, puis cette adresse est incrémentée de 1.

   L'opération Dépiler correspond à : l'adresse mémoire courante est décrémentée de 1, puis la valeur à la nouvelle adresse est affichée sur la sortie S.

   Astuce : Si vous n'arrivez pas à réaliser le fonctionnement intégral, des solutions partielles réalisant exclusivement les empilements ou les dépilements seront tout de même considérées dans l'évaluation.

1. Vous pouvez ré-utiliser le multiplexeur de `Logisim`.