ARC TD2

Logique séquentielle, mémoire
(1 séance sur machine $\simeq$ 4h), 9 avril 2024

Construit à partir du poly de TD-TP des cours IF-AC et IF-AO

Jusqu'à présent on s'est contenté de décrire des circuits combinatoires : la valeur des sorties de votre additionneur à un instant donné t ne dépend pas du passé, mais uniquement de la valeur de ses entrées à cet instant précis.

C'est très bien, mais on ne va pas aller très loin avec ça. Très vite, on va vouloir faire des calculs qui vont nécessiter plusieurs opérations *successives* : à chaque étape, une opération combinatoire produira un résultat temporaire qui sera une opérande d'une opération à l'étape suivante. Ces étapes sont les fameux instants successifs de l'exécution de notre programme et de tels circuits sont appelés *circuits séquentiels*.

Pour pouvoir construire de tels circuits, il va nous falloir des petits circuits pour mémoriser des valeurs, *d'un instant sur l'autre*. Vous allez maintenant découvrir comment on construit de tels composants qu'on nomme *registres* (*registers* or *flip-flops* en anglais), en partant d'éléments simples appelés *verrous* (en anglais *latch*), eux-mêmes construits à partir de portes logiques. Ce chapitre aborde la construction de ces éléments.

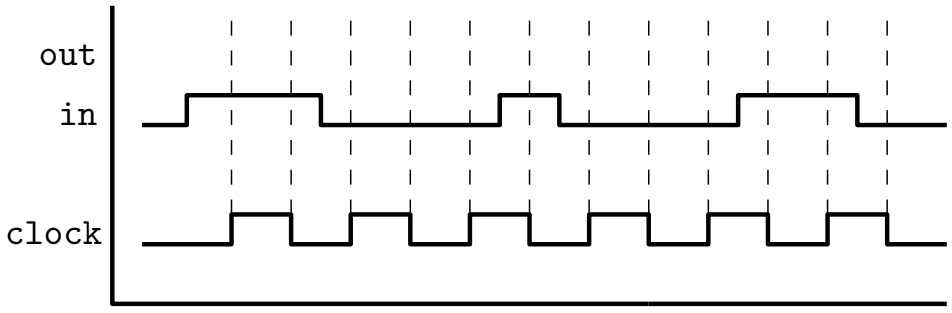
1 Registres

NB : Créez chacun des circuits demandés séparément dans Digital, notamment pour pouvoir facilement les réutiliser plus tard.

Latch et flip-flop

QUESTION 1 ▶ Construisez un verrou (latch) ainsi qu'un registre 1 bit (flip-flop) dans Digital.
Dans le poly du cours (chapitre 4.2, page 41), vous trouverez une description de ces deux composants. Vous trouverez le multiplexeur dans la bibliothèque de composants de Digital (rubrique `Plexers`).

QUESTION 2 ▶ Remplissez le chronogramme suivant pour vous assurer que vous comprenez le comportement du flip-flop.



Flip-flop avec reset

QUESTION 3 ▶ Complétez votre flip-flop afin de permettre sa remise à zéro.

Pour cela, créez un nouveau circuit qui utilise votre flip-flop précédent et possède une entrée supplémentaire *reset*. Il existe deux grandes catégories de Reset : le Reset synchrone, qui est pris en compte lors du prochain front d'horloge et le Reset asynchrone qui est pris en compte immédiatement. Ici nous vous demandons d'implémenter un Reset synchrone. Attention : l'objectif premier du Reset n'est pas de mettre à zéro la sortie de votre circuit (même si c'est ce qui va se passer) mais bien le *contenu* de votre mémoire de 1 bit.

Registre à commande de chargement

QUESTION 4 ▶ Ajoutez une commande de chargement à votre registre.

Pour cela, encapsulez votre flip-flop à reset, et étendez-le avec une entrée supplémentaire *enable* : la valeur du registre n'est modifiée pour prendre la valeur d'entrée que si *enable* est vrai. Sinon, la sortie conserve sa valeur actuelle.

Remarque : Vous venez de construire un registre complet à 1 bit, qui vous permet de conserver une information sur plusieurs cycles d'exécution et de changer cette information à la demande, grâce à la commande *enable*.

Attention Dans la suite nous utiliserons un composant *custom* disponible sur Moodle, nommé `Reg` (et sa version 8bits : `Reg8`) qui possède à la fois un reset (*clear*) et une commande de chargement (*enable*). Ce composant n'est malheureusement pas fourni par Digital de version native. **il faut donc le télécharger depuis moodle dans le repertoire ou vous faites vos design digital** puis l'acceder dans `components->custom->Reg`.

1.1 Registre à 4 bits

On ne sait pour l'instant que mémoriser 1 bit. Pour aller plus loin, il va nous falloir de quoi mémoriser des paquets de bits. En pratique, la taille des registres est très liée à d'autres éléments de l'architecture concernée : taille des bus, taille de la mémoire. Pour ce TP, nous nous limiterons à des registres 4 bits. Contrairement aux différents verrous et flip-flop que nous avons construits jusque-là, la complexité ne se situe pas dans le comportement temporel mais dans le nombre des boîtes et des connexions constituant le circuit.

QUESTION 5 ▶ Construisez un registre 4 bits en collant côte-à-côte 4 registres 1 bit (Utilisez le registre `Reg` disponible sur Moodle). Veillez à la synchronisation de l'ensemble.

2 Un petit compteur

Nous allons maintenant concevoir un circuit qui va nous permettre de compter. La valeur du compteur sera enregistrée dans un registre 8 bits. Nous compterons donc de la valeur 0 à la valeur $2^8 - 1$.

QUESTION 6 ▶ Construisez ce compteur en utilisant le registre que vous venez de construire ainsi que l'additionneur 8 bits réalisé dans le premier TD. Pour réutiliser cet additionneur il suffit de copier le fichier `.dig` de l'additionneur dans le repertoire ou vous créez le compteur.

Parce que c'est trop beau, vous pouvez maintenant lancer une simulation en choisissant *tick enabled* dans le menu *Simulate*. Vous pouvez même régler la fréquence assez haut pour vérifier rapidement que votre compteur se comporte "bien comme il faut" lorsqu'il atteint la borne sup.

3 Mémoires adressables

Lorsqu'on veut stocker beaucoup de données en même temps, utiliser des registres indépendants implique une trop grande complexité d'interconnexion (i.e. il y a trop de filasse). On invente alors (tadam) des mémoires adressables.
Dans cette partie, vous allez implémenter une telle mémoire adressable. Elle sera petite (4 mots de 4 bits), mais vous constaterez également qu'étendre ce composant à des capacités plus grandes est simple.

3.1 Construction d'une mémoire de 4 mots de 4 bits

QUESTION 7 ▶ Concevez maintenant une mémoire de 4 mots de 4 bits.

Vous avez tous les composants nécessaires. Elle sera d'une taille considérable : 16 bits en tout (2 octets) ! Pour vous aider un peu, elle aura les entrées suivantes :

— une entrée d'adresse `A`.

— un entrée de données `DI` (pour *Data In*).

— une entrée de contrôle `WE` (pour *Write Enable*) qui permet de décider quand on veut écrire la donnée présente sur le bus `DI` à l'adresse `A`.

— une entrée `reset`, pour réinitialiser la mémoire.

— une entrée `clk`, pour piloter la mise a jour de la mémoire.

Votre mémoire aura aussi une sortie de données `DO` (pour *Data Out*) sur laquelle on peut lire la valeur contenue à l'adresse `A`. Notez que vous pouvez connecter des "probes" sur les sorties des registres afin de visualiser leur contenu (menu `Components`→`I0`).

Vous utiliserez des multiplexeurs et des démultiplexeur (`components` → `plexers`) pour selectionner les données à lire ou à écrire suivant l'adresse.

QUESTION 8 ► Aller chercher dans le menu `Components->memory->RAM`, le composant qui correspond à la mémoire que vous avez faite et régler les paramêtres de ce composant pour avoir exactement la même mémoire que celle que vous avez faite.

4 Un premier automate mémoire

QUESTION 9 ► Faite un circuit qui utilise votre compteur 8 bits ainsi que le composant "RAM separated ports" configuré en 8 mots de 8 bits. Ce circuit aura comme entrée (en plus de l'horloge et du reset) un entier `Max_add` et un bit `str`. Après le reset, lorsque le bit `str` (pour "store") sera à 1, ce circuit remplira les cases mémoire de 0 à `Max_add` avec la valeur `Mem[i]=i=`. lorsque la valeur de `str` sera à 0, le circuit lira séquetiellement les case mémoires.