Travaux pratiques d'architecture des ordinateurs

TP3: circuits séquentiels

Rappel des notions abordées (polycopié, chapitre IV)

- Circuit séquentiel = les sorties dépendent des entrées et de **l'état interne**, et impliquent un rebouclage des signaux internes
- Le **graphe d'états** décrit complétement les **transitions** d'un circuit séquentiel

MOORE MEALY

- La **table de transitions** est la forme tabulaire du graphe d'états, elle permet de trouver les états équivalents
- L'état interne est **instancié** dans une suite de bits appelée **vecteur d'état**, dont la taille dépend du nombre d'états. Par exemple 6 états différents →vecteur d'état de 3 bits
- Si une horloge est responsable des changements d'états, le circuit est dit **synchrone**. Les circuits synchrones sont simples à concevoir car tous les bits du vecteur d'états changent simultanément au front d'horloge
- Les **bascules** sont des mémoires 1 bit qui forment les éléments du vecteur d'état d'un circuit synchrone. Il y en a de 3 types : D, T, JK. Elles sont équivalentes, mais adaptées à des contextes différents. Equations SHDL :
  - `x := d; x.rst = reset; x.clk = h; // bascule D`
  - `x := /t*x + t*/x; x.rst = reset; x.clk = h; // bascule T`
  - `x := /k*x + j*/x; x.rst = reset; x.clk = h; // bascule JK`

1. Utilisation typique de la bascule D : registre

La bascule D est particulièrement adaptée aux situations dans lesquelles une donnée doit être mémorisée telle quelle.

Concevoir dans cette optique un circuit qui mémorise en permanence la valeur la plus élevée d'un capteur. Il aura l'interface suivant :

```
module maximum(rst, h, val[3..0] : max[3..0])
```

La valeur du capteur arrive en `val[3..0]`; à chaque front de l'horloge `h`, le circuit doit modifier si nécessaire et mémoriser la valeur maximum, disponible sur `max[3..0]`. L'horloge est supposée suffisamment rapide pour suivre les changements de la température.

2. Utilisation typique de la bascule T : compteur

La bascule T est particulièrement adaptée aux situations qui se décrivent en termes d'inversion, par exemple les algorithmes de comptage et de décomptage.

a. Compteur standard

Concevoir des circuits de comptage 2 et 4 bits d'interface suivant (on ne cherchera pas à les faire de façon modulaire) ; le compteur ne s'incrémente au front d'horloge que si `en` = 1 :

```
module count2(rst, h, en : s[1..0])
module count4(rst, h, en : s[3..0])
```

b. Compteur de montre

Un compteur encore plus simple peut être réalisé uniquement avec des bascules T, *sans aucune logique combinatoire supplémentaire*, mais en violant la règle qui prescrit que toutes les bascules doivent partager la même horloge. Concevoir un tel compteur sur 4 bits, d'interface :

```
module clock4(rst, h : s[3..0])
```

Ce compteur évolue aux fronts *descendants* de h. Tester `s` avec les entrées-sorties à distance.

Chainer ensuite 7 de ces compteurs pour former un compteur de montre 28 bits d'interface :

```
module clock28(rst, h : s[27..0])
```

Ecrire ensuite un module de test qui incorpore une instance de clock28 et qui :

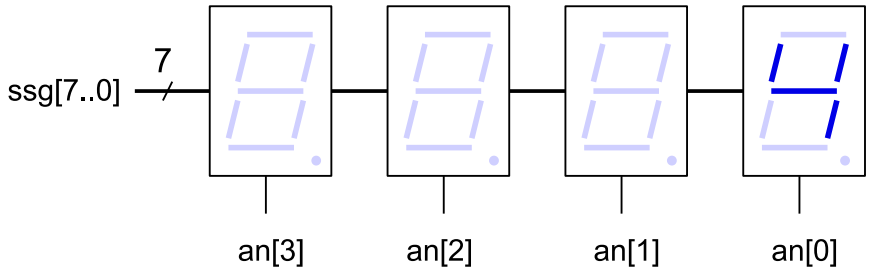
- Utilise pour `h` l'horloge interne `mclk` à 50 MHz, et force `rst` à 0
- affiche sur `ld[7..0]` les 8 bits de poids forts de `s[27..20]`

Etablir la formule de la fréquence du signal carré présent sur la sortie `s[i]`

3. Conception d'un circuit séquentiel à l'aide d'un compteur de phases

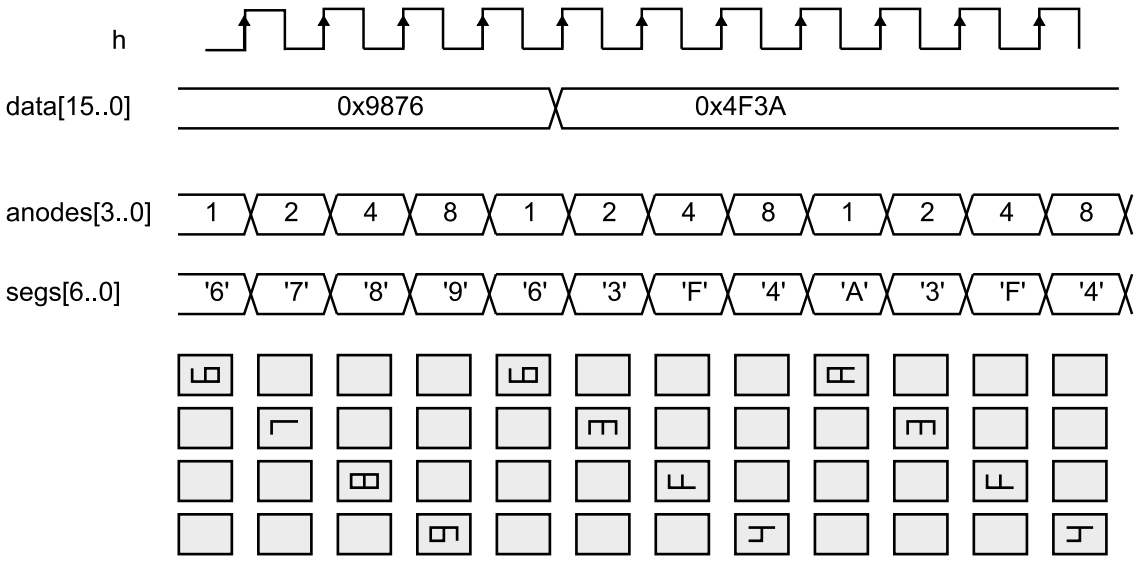
Certains circuits séquentiels suivent une séquence de phases. L'utilisation d'un compteur suivi d'un décodeur de phase est alors la méthode de conception la plus directe.

On va réaliser par cette méthode le multiplexage dans le temps des 4 afficheurs 7-segments. On rappelle le câblage des afficheurs, avec les cathodes `ssg[7..0]` toutes reliées entre elles pour chacun des 4 afficheurs afin de minimiser le nombre de signaux :



L'inconvénient, c'est qu'on ne peut alors afficher que le même chiffre sur tous les afficheurs à un moment donné ! Un multiplexage temporel est donc nécessaire pour donner l'illusion optique d'un chiffre différent sur chaque afficheur.

A chaque pas de temps, on active un seul afficheur (en agissant sur les anodes), et on place sur les cathodes `ssg[7..0]` le décodage des 4 bits de donnée qu'on souhaite afficher sur cet afficheur. En faisant circuler assez vite l'afficheur actif, on donne l'illusion d'un affichage simultané sur tous les afficheurs. Le chronogramme suivant illustre le processus :



En utilisant un compteur, un décodeur et le décodeur 7-segments `dec7segsH` du TP1, concevoir le module qui réalise ce multiplexage :

```
module affmux(rst, h, data[15..0] : anodes[3..0], segs[6..0])
```

Tester avec les entrée-sorties à distance.

Tester ensuite sur les afficheurs réels, avec `data[15..0]` en entrées-sorties à distance, et une horloge prise sur un des 8 bits de poids fort (selon la vitesse voulue) d'un compteur de montre `clock28` cadencé par `mclk`. On n'oubliera pas de tenir compte de l'inversion des anodes et des segments.

4. Mémoire RAM ; implémentation d'un buffer FIFO

Le module prédéfini suivant implémente une mémoire dual-port de 16 mots de 8 bits :

```
 rams_dual_asyn_read16x8(clk, en, wrAddr[3..0], rdAddr[3..0],
din[7..0], doutWr[7..0], doutRd[7..0])
```

Cette mémoire est asynchrone en lecture, c'est-à-dire qu'elle fournit sur `doutRd[7..0]` le contenu d'une case mémoire dès que l'adresse de lecture est placée sur `rdAddr[3..0]`, sans attendre de front d'horloge. Elle fournit également de façon asynchrone dans `doutWr[7..0]` le contenu de la case mémoire d'adresse `wrAddr[3..0]`.

Elle est synchrone en écriture, c'est-à-dire que le contenu de la case mémoire d'adresse `wrAddr[3..0]` est affecté de la valeur `din[7..0]`, non pas immédiatement, mais au front montant de l'horloge `clk`, et à la condition que `en=1`.

Créer un petit module de test et faire des essais de lecture et d'écriture de cette mémoire en utilisant les entrées/sorties à distance.

On souhaite implémenter un buffer FIFO de 16 mots de 8 bits, d'interface :

```
 module fifo16x8(rst, clk, put, get, din[7..0], dout[7..0],
empty, full)
```

- `empty` indique que le buffer est vide, `full` indique qu'il est plein
- Pour ajouter un mot dans le buffer, on le présente sur `din[7..0]`, on met put à 1 et on envoie un front d'horloge. La commande est inopérante si `full` = 1
- Quand le buffer n'est pas vide, `dout[7..0]` contient le plus ancien mot ajouté qui n'a pas encore été retiré
- Pour retirer un mot du buffer, on met `get` à 1 et on envoie un front d'horloge. La commande est inopérante si `empty` = 1
- On peut lire et écrire durant le même cycle d'horloge

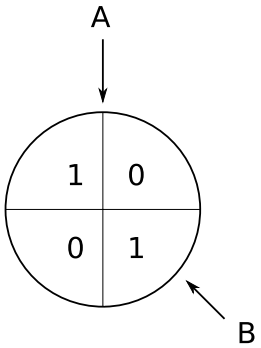
Les mots du buffer seront stockés dans une ram dual-port et 2 compteurs de 4 bits fourniront l'adresse courante de lecture `rdAddr[3..0]` et l'adresse courante d'écriture `wrAddr[3..0]`

- Dessiner le schéma reliant ensemble les compteurs et la mémoire
- Produire le signal `lastFull` qui indique que le pointeur de lecture est sur le point de rattraper le pointeur d'écriture, le signal `lastEmpty` qui indique que le pointeur d'écriture est sur le point de rattraper le pointeur de lecture.
- Dessiner le graphe d'état du système, avec le vecteur d'entrées `(get, put, lastFull, lastEmpty)` et le vecteur de sorties `(empty, full)`

Implémenter l'ensemble et le tester avec les entrées-sorties à distance.

5. Sens de rotation d'un disque

On considère un disque comportant des secteurs colorés en alternance, et comportant 2 capteurs décalés :



Lorsque le disque tourne dans le sens 0, la séquence sur A, B est : 01, 11, 10, 00, 01, etc. Lorsqu'il tourne dans le sens 1 : 01, 00, 10, 11, 01, etc.

- Dessiner le graphe de Moore de ce circuit
- Le convertir en graphe de Mealy
- Dessiner la table de transition et la simplifier
- Synthétiser le circuit avec l'interface suivant :

```
rotation(rst, clk, a, b : sens)
```

L'horloge sera supposée suffisamment rapide pour ne rater aucun événement.

Notation:

- Bascule D et maximum
- Bascule T et compteurs
- Conception à l'aide d'un compteur de phases & affichage multiplexé
- RAM & FIFO
- Sens de rotation d'un disque