N7_SN_1A

Architecture des ordinateurs - Semestre 5

TD3-TP&TP4

Algorithmes cablés

Nous nous intéressons à l'algorithme de tri par sélection, présenté ci-dessous :

```
Pour I de 1 à N-1 Faire
    Calculer Max et Indice_Max de Tab(I..N)
    Echanger Tab(I) et Tab(Indice_Max)
Fin Pour
```

Et nous allons essayer de l'implanter dans un circuit logique.

Commençons par adapter l'algorithme de manière à le rendre plus proche des notions manipulées dans les couches basses : adresse, etc. On notera donc Tab(ad1..adn) le tableau dont les éléments sont stockés aux adresses Ad1, Ad2, … Adn ; et on changera la structure de contrôle de l'algorithme :

```
Ad_courante <- Ad1
Tant que Ad_courante <= Adn Faire
        Calculer Max et Ad_Max de Tab(Ad_courante..Adn)
        Echanger Tab(Ad_courante) et Tab(Ad_Max)
        Ad_courante <- Ad_Courante + 1
Fin TantQue
```

1- Calcul de Max et de Ad_Max de Tab(Ad1..Ad2)

Nous allons nous intéresser dans un premier temps au calcul de Max et de Ad_Max, et nous adopterons l'algorithme suivant :

```
Ad_courante <- Ad1
Max <- Tab(Ad_courante)
Ad_Max <- Ad_courante
Ad_courante <- Ad_courante + 1
Tant Que Ad_Couante <= Ad2 Faire
    Si Tab(Ad_courante) > Max Alors
        Max <- Tab(Ad_Courante)
        Ad_Max <- Ad_courante
    Fin Si
    Ad_courante <- Ad_courante + 1
Fin TantQue
```

Analysons cet algorithme et listons les éléments dont nous avons besoin :

- Nous utiliserons une mémoire RAM avec des adresses codées sur 8 bits (256 mots), et des mots codés sur 32 bits.

- Pour conserver Ad_Max, nous aurons besoin d'un registre 8 bits (reg8 fait en TP2)
- Pour conserver le Max, nous aurons besoin d'un registre 32 bits (extension de reg8)
- Pour gérer Ad_Courante, nous avons besoin d'un circuit qui permet :
  - d'initialiser Ad_Courante avec Ad1
  - d'incrémenter Ad_Courante
  - de s'arrêter lorsque Ad_courante atteint la valeur Ad2

Il s'agit d'un compteur que l'on peut initialiser, et que l'on peut arrêter. En s'inspirant du module count_init4 fait en TP2, on réalisera en TP3 le module :
count8_b1_b2(rst, clk, count, init, b1[7..0], b2[7..0] : c[7..0]) qui :

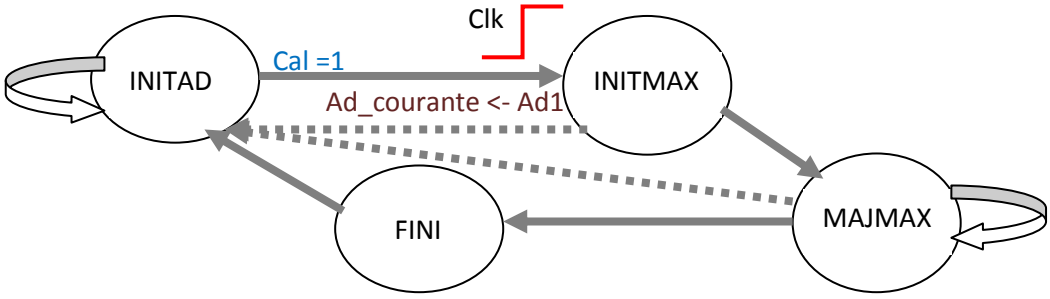
- est initialisé avec b1 lorsque init=1 et count=0
- est incrémenté de 1 lorsque init=0, count=1 et c[7..0]<b2[7..0]

On utilisera des bascule T sans entrée « enabled »

L'exécution de l'algorithme passe par plusieurs étapes successives dans le temps, et nécessite par conséquent d'utiliser un circuit séquentiel qui permet de séparer les différentes actions dans le temps et de gérer leur séquencement. Il s'agit donc d'un circuit séquentiel composé d'un certain nombre d'états, et qui fonctionne selon le schéma suivant :

- A chaque top d'horloge, une transition s'effectue selon les conditions en cours, soit sur le même état soit vers un état différent,
- A chaque top d'horloge, donc à chaque transition, des opérations sont effectuées : les entrées de ces opérations ont préparées avant le top d'horloge, et le résultat est validé (enregistré) sur le top d'horloge

Pour notre algorithme de calcul du Max, notre circuit peut être représenté par le graphe à états suivant (les noms des états seront majuscule pour faire la différence avec les autres signaux) :



- A l'entrée dans l'algorithme, on se trouve dans un état que l'on appellera « INITAD», et on y restera tant que l'ordre de calcul n'est pas donné (on utilisera une entrée appelée « cal »)
- Lorsque « cal » est mis à 1, on quitte l'état « INITAD » pour passer dans un nouvel l'état « INITMAX », en préparant l'initialisation du compteur d'adresse Ad_Courante <- Ad1
- Au prochain top d'horloge, on quitte l'état « INITMAX » pour passer dans l'état « MAJMAX », en préparant les opérations suivantes :
  Max <- Tab(Ad_Courante) ; Ad_Max <- Ad_Courante; Ad_courante <- Ad_Courante + 1
  Il est important ici de noter que les 3 opérations ne sont pas interdépendantes, sinon elles ne pourraient pas être exécutées et validées sur un même cycle d'horloge.
- On boucle dans l'état « MAJMAX » tant que Ad_courante <= Ad2 (on utilisera un signal appelé « finTab » produit par un circuit ucmp8 fait en TP1. A chaque cycle d'horloge, on réalisera les opérations suivantes :

Si Elément_courant > Max Alors Max <- Elément_courant & Ad_Max <- Ad_courante

- Lorsque Ad_courante atteint Ad2 (signal finTab=1), on passe dans un nouvel état qu'on appellera « FINI » et on y restera jusqu'à ce que l'entrée cal soit remise à 0. Dans ce cas on passe de nouveau à l'état « INITAD » pour recommencer un nouveau cycle de calcul dès que « cal » sera mis à 1. Le circuit qui utilisera le module Cal_max peut enchaîner plusieurs calculs en se synchronisant à la l'aide de la sortie « FINI » de Cal_max et de la remise à 0 (j'ai bien pris connaissance du résultat) puis à 1 de cal (je lance un nouveau calcul).

A- Circuit etats_cal_max

Pour gérer le séquencement décrit dessus, on construira et testera un module qui gère les transitions entre les différents états :
etats_cal_max(rst, clk, cal, finTab : INITAD, INITMAX, MAJMAX, FINI)
On peut résumer les conditions de passage entre états dans le tableau suivant :

Etat	INITAD	INITMAX	MAJMAX	FINI
Condition d'entrée ou de maintien	/cal	INITAD*cal	INITMAX* cal + MAJMAX *cal */fintAB	MAJMAX*cal*finTab + FINI*cal

Et pour simplifier cette réalisation, on utilisera une bascule par état (bascule D). On peut donc écrire :
INIT := /cal on clk set when rst // set pour que INIT soit initialisé à 1 au démarrage
On n'utilisera pas l'entré « enable » de la bascule (par défaut = 1)

Compléter le module etats_cal_max. Il sera testé en TP3.

B- Circuit cal_max

Compléter le module :
module cal_max(rst, clk, cal, ad1[7..0], ad2[7..0], data[31..0] : adCour[7..0], max[31..0], adMax[7..0], InitAd, InitMax, CalMax, Fini)
    etats_cal_max(rst, clk, cal, finTab : InitAd, InitMax, CalMax, Fini)
    count8_b1_b2(rst, clk, encount, ……, ad1[7..0], ad2[7..0] : adCour[7..0])
    ucmp8(……, ……. : supad, eqad)
    reg32(rst, clk, enMax, …… : max[31..0])
    reg8(rst, clk, enMax, …… : adMax[7..0])
    ucmp32(……, …… : supmax, eqmax)
end module

Pour travailler de façon incrémentale, simplifier les tests, et pouvoir réutiliser le module « cal_max » dans le circuit de tri, on n'y intègre pas la mémoire. L'entrée « data » indique la valeur de l'élément courant supposée lue en mémoire, à l'adresse fournie sur la sortie « adCour ». Il sera couplé avec un module RAM, à l'étape suivante, pour que le calcul soit fait sur un tableau enregistré en mémoire.

C- Module max_tab

Pour réaliser le module max_tab, on utilisera une mémoire RAM, module prédéfini, qui a pour interface : $ram_aread_swrite(clk, write, ad[7..0], data_in[31..0] : data_out[31..0])
Et qui fonctionne de la façon suivante :

- Lecture asynchrone : la sortie data_out indique à tout moment le contenu du mot mémoire présent à l'adresse ad
- Ecriture synchrone : la donnée data_in est enregsitrée dans le mot mémoire à l'adresse ad sur le front d'horloge clk lorsque write=1

Réaliser le module max_tab (rst, clk, cal, ad1[7..0], ad2[7..0] : max[31..0], adMax[7..0], initAd, initMax, majMax, Fini)

Il sera testé en TP3. Pour initialiser la ram on utilisera un fichier selon le format suivant (adresse sur la première colonne, donnée sur la seconde) :

```
[
        {
          "00000000" : "0x00000011",
          "00000001" : "0x00000082",
          "00000010" : "0x00000093",
          "00000011" : "0x00000064",
          "00000100" : "0x00000055",
          "00000101" : "0x00000123",
          "00000110" : "0x00000077",
          "00000111" : "0x00000028",
          "00001000" : "0x00000219"
        }
]
```

2- Tri de Tab(Ad1..Adn)

La construction du module de tri suit la même démarche :

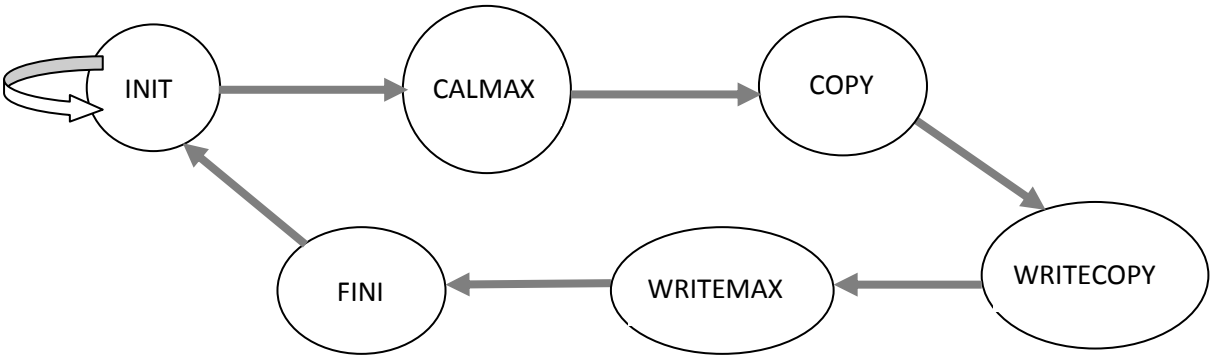
- réalisation d'un module etats_tri_tab (6 états)
- utilisation du module max_tab fait plus haut et d'une RAM.

Reprenons l'algorithme principal :

```
Ad_courante <- Ad1
Tant que Ad_courante < Adn Faire
        Calculer Max et Ad_Max de Tab(Ad_courante..Adn)
        Echanger Tab(Ad_courante) et Tab(Ad_Max)
        Ad_courante <- Ad_Courante + 1
Fin TantQue
```

Et listons les éléments dont nous aurons besoin :

- Max et Ad_Max seront fournis par le module cal_max
- Pour gérer Ad_Courante, nous utiliserons, comme pour cal_max, un module count8_b1_b2(rst, clk, count, init, b1[7..0], b2[7..0] : c[7..0])
- L'exécution de l'algorithme passe par plusieurs étapes successives dans le temps, et nécessite par conséquent d'utiliser un circuit séquentiel que l'on peut représenter avec le graphe dessous (transitions incomplètes) :



- A l'entrée dans l'algorithme, on se trouve dans un état que l'on appellera « INIT », et on y restera tant que l'ordre de calcul n'est pas donné (on utilisera une entrée appelée « cal »)
- Lorsque « cal » est mis à 1, on quitte l'état « INIT » pour passer dans un nouvel état que l'on appellera « CALMAX », en préparant l'opération d'initialisation Ad_courante <- Ad1
- On boucle dans l'état « CALMAX » tant que la recherche du max n'est pas terminée : la sortie Fini du module cal_max nous l'indique
- Quand Fini=1, on transite automatiquement vers un état appelé « COPY », en préparant l'opération qui permettra de copier Tab(Ad_courante) dans un registre 32 bits.
- Depuis l'état « COPY », on transite automatiquement vers un nouvel état nommé « WRITECOPY », en préparant l'opération permettant d'enregistrer l'élément précédemment sauvegardé à la place du max (Tab(adMax))
- Depuis l'état « WRITECOPY », on transite automatiquement vers un nouvel état nommé « WRITEMAX », en préparant l'opération permettant d'enregistrer le max dans Tab(Ad_courante)
- Depuis l'état « WRITEMAX », deux cas de figure se présentent :
- Si Ad_courante < Adn, alors on transite vers l'état « CALMAX », en incrémentant Ad_Courante, et on repart sur un nouveau cycle de calcul du max
- Sinon, on transite vers un état appelé « FINI » (car le tri est terminé), et on y reste tant que cal reste égal à 1
- Dès que cal est remis 0, on repart dans l'état initial « INIT».

Compléter le graphe d'états dessus.

A- Circuit etats_tri_tab

Pour réaliser le module etats_tri_tab(rst, clk, cal, finMax, finTab : INIT, CALMAX, COPY, WRITECOPY, WRITEMAX, FINI), commençons par résumer les conditions de transition entre les différents états :

Etat	INIT	CALMAX	COPY	WRITECOPY	WRITEMAX	FINI
Condition d'entrée ou de maintien	/cal					

Réaliser et tester le circuit en TP4

B- Circuit tri_tab

On va réaliser et tester le circuit :
module tri_tab (rst, clk, cal, ad1[7..0], ad2[7..0] : etat[5..0], calMax, etat_max[3..0], finMax, max[31..0], adMax[7..0], adCour[7..0], adRam[7..0], wRam, inRam[31..0], outRam[31..0])

qui effectue le tri décroissant d'un tableau stocké en mémoire entre l'adresse ad1 à l'adresse ad2 (comprises), selon l'algorithme décrit ci-dessus.

- Cal = 0, on peut lire le contenu de la Ram à l'adresse indiqué dans ad1
- Cal =1, on lance le tri
- La sortie etat[5..0] indique de droite à gauche les états INIT, CALMAX, COPY, WRITECOPY, WRITEMAX et FINI : le tri est terminé lorsque etat[5]=FINI=1
- Les autres sorties sont mises pour aider « débugguer », et voir ce qui se passe

On utilisera les modules suivants :

- count8_b1_b2 pour gérer adCour (initialisation et incrémentation)
  count8_b1_b2(rst, clk, ……, ……, ad1[7..0], ad2[7..0] : adCour[7..0])
- Cal_max pour le calcul du max du sous-tableau courant Tab(adCour..ad2)
  cal_max(rst, clk, ……., ……, ……, …… : adElem[7..0], max[31..0], adMax[7..0], InitAd, InitMax, MajMax, finMax)
- Etats_tri_tab pour gérer le séquencement du circuit
  etats_tri_tab(rst, clk, ……, ……., …… : INIT, CALMAX, COPY, WRITECOPY, WRITEMAX, FINI)
- Un reg32 pour stocker l'élément couant Tab(adCour) qui sera échangé avec le max courant
  reg32(rst, clk, …… , …… : regCopy[31..0])
- Une RAM $ram_aread_swrite(clk, wRam, adRam[7..0], inRam[31..0] : outRam[31..0])

Complétez les signaux manquants et leur équation logique quand cela est nécessaire.
A finir et tester en TP4.

Pour que le module respecte fidèlement l'algorithme dessus et ne fasse pas un tour en trop, il faut y ajouter un autre module fait en TP1.