Architecture des Circuits : Cahier d'Exercices – TP 3 (4h)

Table des matières

Séparation contrôle-données : le multiplieur binaire

Préambule

ATTENTION : ce TP [1] est difficile ! Les très (trop ?) nombreuses questions sont là pour vous guider. N'allez ni trop vite ni trop lentement et assurez-vous que vous comprenez toujours ce que vous faites. Dans le cas contraire, demandez tout de suite à un.e enseignant.e (ou à votre voisin.e, ça nous fait des vacances ;) mais n'avancez pas plus loin avant d'être sûr.e de vous !

1 Introduction

Jusqu'à maintenant, en TD et en TP, nous avons étudié des circuits relativement simples : circuits combinatoires, registres et mémoires et circuits séquentiels (FSM). Au cours de cette dernière séance de TP, nous allons intégrer tous ces éléments pour réaliser des *Machines à États Algorithmiques* (*Algorithmic State Machines* – ASM). Une ASM est une machine à états finis (*Finate State Machine* – FSM) qui utilise un circuit séquentiel (l'automate de contrôle ou "contrôleur") pour coordonner une série d'opérations réalisées par un ensemble d'unités fonctionnelles (compteurs, registres, ALU, ...). Cet ensemble d'unités fonctionnelles est appelé le *Chemin de données* (*datapath*). Cette série d'opérations implémente un algorithme (d'où le nom de la machine). Le gros intérêt des ASM par rapport aux FSM est qu'elles permettent de réaliser des FSM comportant un très grand nombre d'états mais en utilisant un contrôleur comportant relativement peu d'états (un grand nombre d'états étant gérés dans le chemin de données). Ainsi, un chronomètre, tel que celui que vous allez réaliser ci-dessous, comporte un très grand nombre d'états – toutes les valeurs différentes affichées chronomètre sont des états différents. Son implémentation sous la forme d'une FSM est donc théoriquement possible mais pratiquement impossible ! Avec une ASM, on va pouvoir diviser le chronomètre en deux éléments séquentiels : un compteur, comportant un très grand nombre d'états mais dont le comportement séquentiel est suffisamment simple pour être décrit directement – c'est-à-dire sans utiliser une FSM – et un contrôleur qui concentrera la complexité du chronomètre sous la forme d'une petite FSM dont le comportement compliqué sera décrit par une FSM ... Les ASM sont la dernière étape de l'architecture des circuits ... Au-delà, nous ne parlerons plus de circuit mais nous commencerons à pouvoir parler d'ordinateur ...
Pour contrôler le chemin de données, votre contrôleur va émettre des *commandes* qui vont impacter le comportement du chemin de données en indiquant, par exemple, quel registre est autorisé à enregistrer, comment les démultiplexeurs ou les multiplexeurs orientent l'information ou quelle opération va être réalisée par une ALU. En retour, le chemin de données va émettre des *rapports* qui vont être utilisés par l'automate de contrôle comme des entrées. Les deux entités (contrôleur et chemin de données) peuvent recevoir et émettre des messages depuis/vers l'extérieur et ils sont impérativement séquencés par la même horloge (ils sont synchrones).
Le choix des éléments relevant du contrôleur et de ceux relevant du chemin de données n'est pas toujours simple. Grossièrement, on peut considérer – même si ce n'est pas un cas général – que le datapath va traiter toutes les opérations sur des nombres (et produire de rapport binaires). Le contrôleur, lui, ne traite généralement que des informations binaires afin de limiter son nombre d'états. Une autre façon de considérer la répartition – même si là encore ce n'est pas une règle absolue – est que, pour un algorithme écrit en pseudocode, le datapath va realiser les opérations tandis que le contrôleur va s'occuper de leur enchainement ...

1. Dont plusieurs éléments sont inspirés d'une séance de travaux pratiques de D. Capson, à l'université McMaster.

Au cours de ce TP, vous allez réaliser deux ASM. La première est très simple et ne devrait pas vous demander trop de temps (le chronomètre, section 2). La seconde est plus complexe : il s'agira de réaliser un multiplieur binaire (sur 4 bits). Elle est présentée section 3 et devrait vous demander environ 3h ...

2 Une ASM simple : le chronomètre – 1h maximum

Vous allez implémenter le chronomètre 16 bits vu en cours (au tableau ; ne le cherchez pas dans les slides, il n'y est pas ...). Pour mémoire, il dispose d'un seul bouton. Une pression sur le bouton déclenche le chronomètre ; une seconde pression l'arrête et affiche le temps écoulé ; une troisième pression remet le chronomètre à zéro et le rend disponible pour le prochain comptage.
Votre chronomètre va donc implémenter un algorithme simple. En pseudocode cet algorithme est :

```
répéter
     tant_que BOUTON == 0
          mettre le chronomètre à zéro et transmettre le résultat
     fin_tant_que
     tant_que BOUTON == 0
          compter et transmettre le résultat
     fin_tant_que
     tant_que BOUTON == 0
          transmettre le résultat
     fin_tant_que
fin_répeter
```

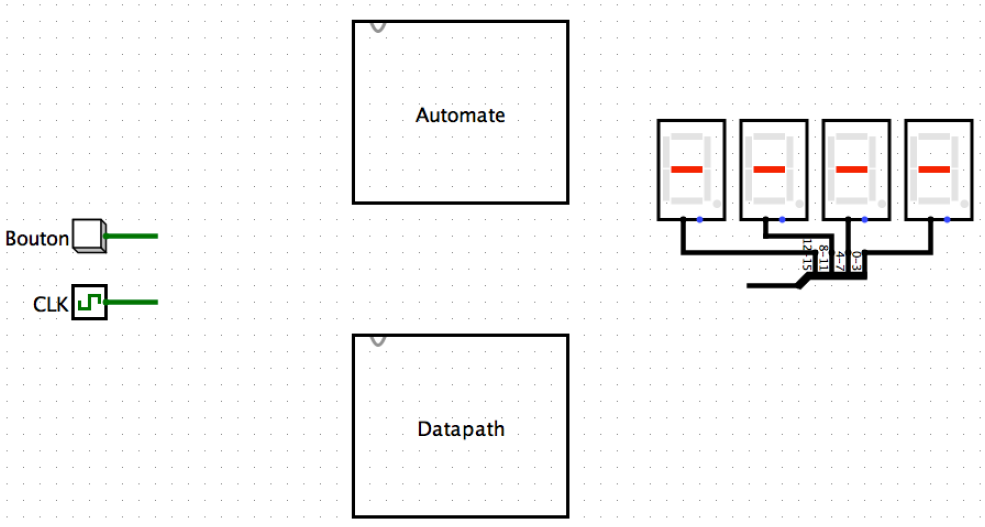
QUESTION 1 ► Où est passée l'information cruciale `BOUTON == 1` ("le bouton est appuyé") dans cet algorithme ?

QUESTION 2 ► Sachant que le chemin de données va s'occuper des opérations touchant aux données numériques et que l'automate va s'occuper de l'enchainement de ces opérations, quels sont, dans l'algorithme, les actions qui vont être réalisées par le chemin de données et celles qui vont être réalisées par l'automate ?

— Actions du chemin de données :

— Action de l'automate :

QUESTION 3 ► Pour que le chemin de données puisse enchaîner ses opérations, il faut que l'automate lui envoie des `Commandes`. Combien y en a-t-il ? Lesquelles ? Compléter le schéma ci-dessous pour indiquer les signaux échangés entre l'automate et le chemin de données. Branchez aussi les entrées-sorties (`Bouton`, `CLK` et l'afficheur) de la façon qui vous paraît le plus adéquate.



2.1 Le chemin de données

QUESTION 4 ► Le chemin de données de ce chronomètre est un simple compteur composé d'un additionneur 16 bits, d'un registre 16 bits et d'une sortie 16 bits (et rien d'autre !). Implémentez ce compteur sous LogiSim.

QUESTION 5 ► Ce chemin de données est commandé par deux signaux (voir question 3). Ajoutez deux entrées à votre chemin de données et branchez-les de façon adéquate ...

QUESTION 6 ► Quel est le nombre d'états de votre chemin de données ? Si vous ne comprenez pas la question, relisez la section 1 !

2.2 L'automate de contrôle

QUESTION 7 ► Votre ASM (et donc votre automate) devra réagir à l'état de son entrée `BOUTON`. Relisez la question 1 ; ça vous met la puce à l'oreille ?

QUESTION 8 ► À partir du signal `BOUTON`, votre automate devra envoyer la "bonne" séquence de signaux au chemin de données (dans la suite, nous appellerons ces signaux `GO` et `RAZ`). Combien votre automate a-t-il d'états ? Dessinez-le.

QUESTION 9 ► Implémentez-le en utilisant le codage "One-Hot-Coding". Branchez votre automate sur votre chemin de données pour obtenir votre ASM ; branchez votre ASM sur un "Button" (LogiSim, menu "Input/Output), sur une "Probe" (LogiSim, menu "Wiring" – vous pouvez utiliser les mêmes afficheurs que dans le schéma de la question 3 mais vous n'êtes pas obligés ...) et, bien sûr, sur une horloge. Secouez un peu, testez ... Bravo, vous venez de réaliser votre première ASM ...

QUESTION 10 ► Votre ASM implémente une FSM ... Quel est le nombre d'états de celle-ci ? Si vous ne comprenez pas la question, relisez encore une fois la section 1 !

Faite valider votre chronomètre par un enseignant avant de passer à la suite.

3 Une ASM un peu plus compliquée : le multiplieur – 3h

Le chronomètre réalisé précédemment restait relativement simple, en particulier parce que dans le cas du chronomètre, l'automate de contrôle ne reçoit pas de rapports de la part du chemin de données ... Nous allons maintenant passer à une ASM un peu plus compliquée (à peine !) en implémentant un *multiplieur entier 4 bits*. La multiplication est un élément fondamental dans de très nombreux algorithmes mais son

implémentation est un peu plus compliquée que celle, par exemple, d'une addition. Par conséquent les circuits numériques disposent généralement de circuits dédiés à la multiplication. Une des problématiques centrale est alors de réaliser des circuits de multiplication qui soient rapides.
L'option la plus évidente, puisque la multiplication est une opération *combinatoire* serait de l'implémenter sous la forme d'un circuit combinatoire ... c'est possible mais cela donne des circuits très complexes dont le *chemin critique* va être assez long (pourquoi est-ce un problème ?). Du coup, une approche alternative va être de décomposer la multiplication en une série d'opérations combinatoires effectuées *les unes après les autres*. En d'autres termes, nous allons remplacer un calcul combinatoire compliqué par un calcul *séquentiel* que nous espérons simple (et donc rapide !). Une approche évidente est de décomposer la multiplication en une série d'additions ($6 \times 4 = 6 + 6 + 6 + 6$) mais on comprend aisément que ce n'est pas très efficace (essayez 36597×76986 !). De plus, cet algorithme ne permet pas de garantir le temps de calcul (sauf à se baser sur le temps le plus long !) puisque le nombre d'opérations à effectuer dépend de la valeur du *multiplicateur* (mais pas de celle du *multiplicande* ... pourquoi ?). Du coup, dès l'école primaire vous avez appris une méthode "papier-crayon" plus efficace que la décomposition sous la forme d'une somme. C'est de cette méthode (qui n'est autre qu'un algorithme) que nous allons nous inspirer pour implémenter un multiplicateur efficace [2] sous la forme d'une ASM ...

2. réellement efficace même si ce n'est pas forcément la meilleure solution ... d'autres approches sont possibles, telles que les "Look-Up Tables" dans lesquelles le multiplicande et le multiplieur sont utilisés pour accéder à une mémoire dans laquelle a été préalablement écrite la solution. Cette approche, gourmande en mémoire (au fait, prenez le cas d'un multiplieur 4 bits ... quelle taille mémoire serait nécessaire pour stocker une Look-Up Table ? mais très efficace en temps est souvent utilisée pour des algorithmes complexes lorsqu'ils doivent être calculés en un temps limite incompatible avec leur complexité ...

3.1 En décimal

Puisque nous allons nous inspirer de l'approche "papier-crayon" de l'école primaire, il est utile de commencer par se poser la question des fondements algorithmiques de cette méthode. Pour celà, nous allons commencer par nous pencher sur la multiplication entière décimale sur 4 digits (dans tout le TP, on ne considérera que des entiers naturels ...).

QUESTION 11 ► Si notre multiplicande et notre multiplieur ont 4 digits chacun, combien de digits sont nécessaires pour stocker le résultat ? quel le plus grand résultat possible ? Justifiez vos réponses (au moins dans vos têtes !)

QUESTION 12 ► Posez (en décimal) la multiplication suivante. Ce faisant, essayez de porter une attention particulière aux opérations nécessaires, identifiez en particulier les opérations de `produit partiel`, de `décalage` et de `somme`. L'idée est d'essayer d'identifier l'*algorithme* que vous utilisez :

		1	3	2	5	(multiplicande)
	×	9	2	8	2	(multiplieur)
						(premier produit partiel)
						(deuxième produit partiel et décalage à gauche)
						(troisième produit partiel et décalage à gauche)
+						(quatrième produit partiel et décalage à gauche)
						(somme finale)

Cette méthode de calcul de l'addition utilise un algorithme simple mais finalement assez efficace. En pseudocode ça ressemble à un truc du style (`Bi` étant le $i^{\text{ième}}$ digit de `B`) :

```
A = Multiplicande
B = Multiplieur
S = 0
pour i = 0 à 3 faire
     extraire Bi de B
     C = A*Bi   // produit partiel
     décaler C à gauche de i digits
     Tab[i] = C
pour k = 0 à 3 faire // somme finale
     S = S + Tab[k]
```

QUESTION 13 ► Classiquement lorsque vous avez appris à poser les multiplications, "on" vous a fait faire les produits partiels d'abord et une seule grosse addition à la fin. Reprenez la multiplication ci-dessus et convainquez-vous qu'en vertu de l'associativité de l'addition, on peut remplacer la somme finale par des sommes partielles effectuées à chaque étape. Reposez-là ainsi ci-dessous ...

		1	3	2	5	(multiplicande)
	×	9	2	8	2	(multiplieur)
						(premier produit partiel)
+						(deuxième produit partiel et décalage à gauche)
						(première somme partielle)
+						(troisième produit partiel et décalage à gauche)
						(deuxième somme partielle)
+						(quatrième produit partiel et décalage à gauche)
						(troisième somme partielle – mais finale)

QUESTION 14 ► Réécrivez l'algorithme sous cette nouvelle forme (toujours en pseudocode). Quel est l'avantage de ce nouvel algorithme sur le précédent ?

3.2 En binaire

Comme nous l'avons vu en TD pour l'addition, le fait que la numérotation décimale et la numérotation binaire utilisent toutes deux le principe du *codage de position* permet de réutiliser directement en binaire de nombreuses méthodes connues en décimal. C'est la cas ici et, comme nous allons le voir tout de suite, l'algorithme "papier-crayon de l'école primaire" fonctionne très bien en binaire. De plus, comme nous le verrons juste après, le passage du décimal au binaire va nous permettre de simplifier encore l'algorithme, donc (*i.*) de simplifier l'ASM et (*ii.*) d'accélérer le calcul !

QUESTION 15 ► Si le multiplicande et le multiplieur sont tous les deux des nombres binaires entiers, positifs, codés sur 4 bits, combien de bits sont nécessaires pour stocker le résultat ? quel le plus grand résultat possible ? Justifiez vos réponses (au moins dans vos têtes !)

QUESTION 16 ► Posez (en binaire) la multiplication suivante et vérifiez votre résultat. L'idée est, là encore, d'essayer d'identifier l'*algorithme* que vous utilisez :

		1	1	0	0	(multiplicande)
	×	1	0	1	1	(multiplieur)
						(premier produit partiel)
						(deuxième produit partiel et décalage à gauche)
						(troisième produit partiel et décalage à gauche)
+						(quatrième produit partiel et décalage à gauche)
						(somme finale)

QUESTION 17 ► Regardez bien votre réponse : le passage en binaire permet de simplifier l'algorithme de multiplication (par rapport à l'algorithme utilisé en décimal). En effet, il n'y a plus vraiment de produit partiel. Pourquoi ? Par quelle opération (plus simple) peut-on le remplacer ?

QUESTION 18 ► Ecrivez en pseudocode l'algorithme de la multiplication binaire en tenant compte de la simplification suggérée à la question précédente

QUESTION 19 ► La simplification de l'algorithme que nous avons utilisée en décimal peut être facilement "portée" en binaire. Réécrivez votre algorithme pour en tenir compte. Pour vous guidez, complétez la multiplication ci-dessous.

		1	1	0	0	(multiplicande)
	×	1	0	1	1	(multiplieur)
						(premier test)
+						(deuxième test et décalage à gauche)
						(première somme partielle)
+						(troisième test et décalage à gauche)
						(deuxième somme partielle)
+						(quatrième test et décalage à gauche)
						(troisième somme partielle – mais finale)

QUESTION 20 ► (Question difficile mais importante ...prenez-le temps de bien comprendre la réponse !) En supposant donc qu'on effectue des sommes partielles, reprenez – encore ! – votre multiplication et convainquez-vous qu'on pourrait très bien remplacer les décalages à gauche par des décalages à droite. Pour cela, posez votre multiplication de cette façon ci-dessous (portez une attention particulière à la retenue propagée à gauche lors des additions – le *Carry Out* d'un additionneur binaire).
Quel est l'avantage du décalage à droite (si vous ne le voyez pas, cherchez encore, il est assez énorme) ? Si c'est si avantageux, pourquoi ne vous l'a-t-on pas appris comme ça à l'école ?

	1	1	0	0	
×	1	0	1	1	
	→				
+					
→					
+					
	→				
+					
→					

QUESTION 21 ► Ecrivez le pseudocode d'un algorithme de multiplication binaire basé sur ces principes. On se rappellera que pour extraire les bits `Bi` successifs, il suffit de décaler `B` à droite à chaque tour de boucle et d'extraire `B0`. Vous pouvez simplifier l'écriture en utilisant les notations suivantes S_{MSQ} et S_{LSQ} ("*S* Most Significant Quartet" et "*S* Least Significant Quartet"). Si vous ne comprenez pas ce qui précède, demandez des explications à quelqu'un ...

Faites impérativement vérifier votre algorithme à un enseignant avant de passer à la suite.

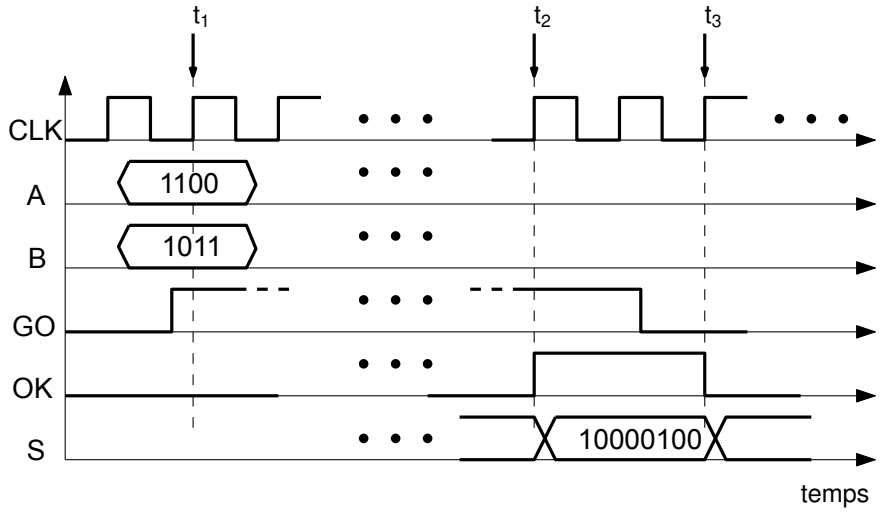
À ce stade, vous disposez d'un algorithme suffisamment simple pour envisager une implémentation numérique sous la forme d'une ASM ... Y'a plus qu'à ...

3.3 une ASM pour la multiplication binaire

Nous allons maintenant construire l'implémentation matérielle de l'algorithme proposé dans la question 21, sous la forme d'une ASM. Comme notre algorithme implémente une fonction combinatoire sous la forme d"une fonction séquentielle, il lui faudra plusieurs cycles d'horloge pour calculer le résultat. Cela va nous imposer de prévoir un *protocole de communication* avec le "monde extérieur" (un "handshake") afin de permettre (1) à l'ASM de savoir quand elle doit calculer – et sur quelles valeurs – et (2) de permettre au monde extérieur de savoir quand le résultat sera valide ...

3.3.1 Protocole de communication

Le protocole de communication avec notre diviseur est illustré dans le chronogramme ci-dessous, et expliqué dans le texte qui suit.



Pour demander une multiplication, le client doit écrire `A` (le multiplicande) et `B` (le multiplieur) sur les entrées correspondantes, positionner `GO` à 1 et attendre le prochain top d'horloge (instant t_1 sur le chronogramme). L'ASM déroule alors l'algorithme et calcule le résultat (on suppose que A et B sont maintenues pendant toute la durée du calcul).

Lorsqu'elle a terminé son calcul, l'ASM positionne sa sortie `OK` à 1 (instant t_2) pour indiquer au client qu'il peut lire le résultat sur la sortie `S` (d'autres valeurs intermédiaires ont peut-être été visibles sur `S` entre-temps, mais elles ne sont pas significatives tant que `OK` n'est pas à 1).

L'ASM garde `OK` à 1 jusqu'à ce que le signal `GO` soit retombé à zéro. La sortie `S` est valide pendant ce temps-là. Lorsque `GO` est de nouveau à zéro (instant t_3) l'ASM retourne dans son état initial et attend une nouvelle requête. (Bien sûr le circuit ne réagit pas immédiatement lorsque `GO` passe à zéro, mais seulement lors du top d'horloge suivant ... Évidemment m'sieur puisqu'il est synchrone !)

Note : Même si le client ne maintient pas `GO` à 1 pendant tout le calcul, et que `GO` est déjà à zéro au moment de t_2, notre circuit garde sa sortie valide pendant au moins un cycle d'horloge. Autrement dit, il y a toujours au moins un cycle d'écart entre les instants t_2 et t_3.

3.3.2 Organisation générale du mulpliplieur

Votre ASM va implémenter l'algorithme proposé à la question 21. En (très) gros, le contrôleur va dérouler l'algorithme et le chemin de données va effectuer les opérations de l'algorithme. Nous allons pour le moment nous intéresser aux

QUESTION 22 ► Connaissant le protocole de communication, donnez la liste des entrées-sorties de votre de votre multiplieur :

— Du monde extérieur vers votre ASM (entrées) :

— De votre ASM vers le monde extérieur (sorties) :

QUESTION 23 ► Connaissant votre algorithme et le protocoles de communication de votre multiplieur, donnez la liste des signaux échangés entre le chemin de données, l'automate de contrôle et le monde extérieur :

— Du monde extérieur vers le chemin de données :

— Du chemin de données vers le monde extérieur :

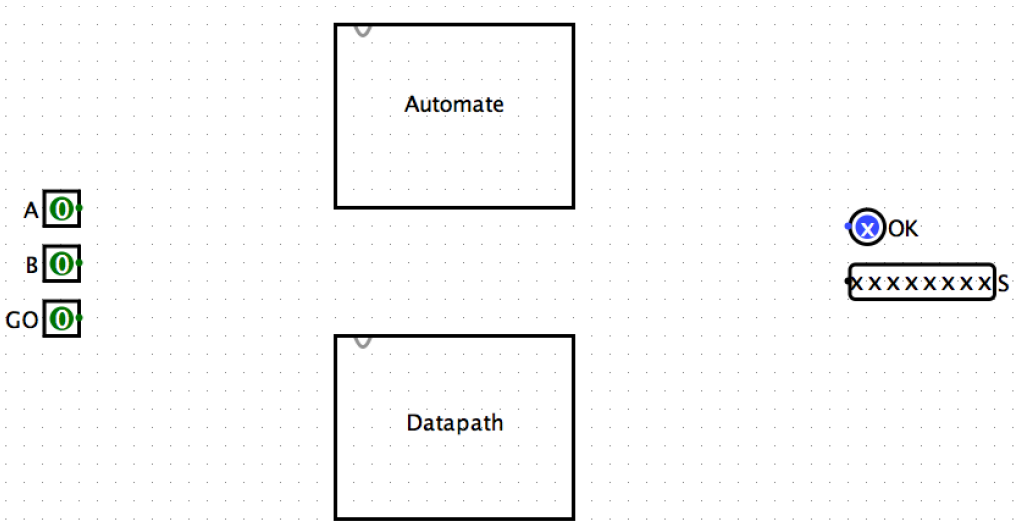
— Du monde extérieur vers l'automate de contrôle :

— De l'automate de contrôle vers le monde extérieur :

— De l'automate de contrôle vers le chemin de données (les *commandes*) :

— Du chemin de données vers l'automate de contrôle (les *rapports*) :

Complétez le schéma ci-dessous pour faire apparaître tous ces signaux.



3.3.3 Le chemin de données (*datapath*)

QUESTION 24 ► Sous LogiSim, implémentez un *registre à décalage*[3] de 4 bits en utilisant des D Flip-Flop. Il comprendra une entrée binaire `B_IN` (le bit entrant à gauche), une sortie binaire `B_OUT` (bit sortant à droite) et une sortie 4 bits `DATA_OUT` (contenu du registre ; c'est donc un SIPO). On rajoutera les commande `CLEAR`, pour remettre le registre à zéro, et `SHIFT` (si `SHIFT` est inactif, votre registre conserve la valeur précédente et n'effectue donc pas de décalage – mais la sortie `DATA_OUT` reste valide). Testez votre registre.
Note : la question suivante est facultative. Si vous êtes un peu courts en temps, utilisez le registre à décalage de LogiSim. Il comporte toutes les fonctions nécessaires (menu `Memory→Shift Register`).

QUESTION 25 ► Reprenez votre registre à décalage et rajoutez-lui une *commande de chargement* (votre registre devient de fait un `PIPO`). Pour ce faire, vous lui rajouterez en entrée un mot de 4 bits (pour la valeur à charger) et une entrée binaire `LOAD` (quand `LOAD` est activé, votre registre effectue un chargement, sans le décaler). Testez votre registre.

QUESTION 26 ► Question difficile ; n'hésitez pas à vous faire aider ! Dessinez votre datapath sous la forme d'un schéma-bloc. Pour vous aider un peu, sachez qu'il devrait contenir trois registres à décalage avec commande de chargement, un additionneur 4 bits, un registre 1 bit et un compteur (pour compter le nombre de tours de boucle dans votre algorithme) ... N'oubliez pas les signaux échangés avec l'automate de contrôle (commande et comptes-rendus). Faites impérativement valider votre solution par un enseignant avant de passer à la suite !

QUESTION 27 ► Ca ne saute pas nécessairement aux yeux mais il est possible de fusionner deux de vos trois registres à décalage. Une petite piste : `B` sort à droite de "son" registre au fur et à mesure que `S` "rentre dans le sien" ...

QUESTION 28 ► En utilisant deux régistre à décalage, un additionneur 4 bits, un compteur et un registre FlipFlop de un bit, implémentez votre datapath (vous pouvez prendre vos modules ou ceux de LogiSim. ATTENTION : pour le FlipFlop 1 bit, il est impératif de prendre celui que vous avez réalisé au 1er TP car le reset de ce registre est commandé par l'automate – commande `CLEAR_CARRY` et doit donc être synchrone. En cas de doute, prenez le FlipFlop corrigé disponible sur Moodle).
Testez-le en déroulant un cas de test à la main.
Le tableau ci-dessous vous permet de remplir l'état des différents registres et signaux au cours du test (une ligne correspondant à un cycle d'horloge ; la partie gauche du tableau correspond aux entrées au début du cycle d'horloge – avant le front montant de l'horloge, la partie droite à l'état du datapath à la fin du cycle – après le front montant de l'horloge). Remplissez-le au fur et à mesure du test. On suppose qu'à $t = 0$ les entrées du circuit valent : `A` = 1100, `B` = 1011 et on rappelle que S_{MSQ} et S_{LSQ} correspondent respectivement aux quartets de poids fort ("M pour Most") et faible ("L pour Least") de `S`). Registre_Carry indique le contenu du registre 1 bit destiné à stocker la retenue sortante de l'addition. Pourquoi a-t-on besoin de la stocker ?

t	INIT	ADD	SHR	CLEAR CARRY	Registre Carry	S_{MSQ}	S_{LSQ}	B0	Valeur Compteur	FINI
0	1	0	0	1	0	0000	1011	1	000	0
1										
2										
3										
4										
5										
6										
7										

3. https://fr.wikipedia.org/wiki/Registre_%C3%A0_d%C3%A9calage

3.3.4 L'automate de contrôle

QUESTION 29 ► Quel est l'alphabet d'entrée de l'automate de contrôle de votre ASM ? Quel est son alphabet de sortie ?

— $I = \{$

— $O = \{$

QUESTION 30 ► En vous aidant du tableau ci-dessus, dessinez l'automate de contrôle de votre ASM. Il devra permettre d'initialiser le datapath, de dérouler l'algorithme de calcul de la multiplication (test de `B0`, addition, décalage), de tester la fin du calcul (test de `FINI`) et de signaler que le résultat est disponible (tant que `GO` = 1).
Faites impérativement valider votre automate par un.e. enseignant.e avant de passer à la suite ...

QUESTION 31 ► Implémentez votre automate sous LogiSim ...

3.3.5 Intégration

QUESTION 32 ► Branchez votre automate sur votre datapath, secouez un peu, testez (commencez par le tester avec une horloge lente – voire manuelle – pour bien vérifier ; testez-le ensuite avec l'horloge la plus rapide de LogiSim ... Classe non ?) ...

Ouf, c'est fini, vous venez de venir à bout des TPs d'AC ; félicitations !