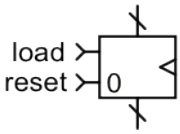
ES102/PC7 : énoncé et corrigé

1) Bascule D numérique en boucle

1a) Rappeler la différence (de structure et d'utilisation) entre bascule D numérique et registre.

Voir CM7/P4 : un registre est une bascule D numérique rebouclée par un multiplexeur numérique commandé par *load*. Alors que la bascule D numérique recharge son contenu à chaque top d'horloge, le registre ne le fait que lorsque *load*=1. L'intérêt d'un registre est donc de pouvoir conserver une donnée pendant plusieurs périodes d'horloge, pour utilisation ultérieure.



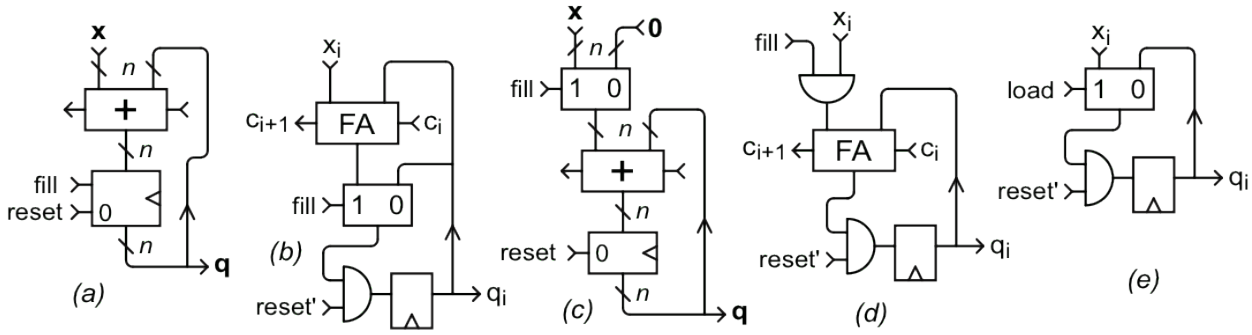
1b) Dessiner la tranche d'un registre équipé d'une remise à 0 *reset* prioritaire sur *load*, qu'on représentera comme montré ci-contre et qu'on appellera *registre resettable* (affranglais ;-)

Comme déjà vu (fin PC6 et CM7), il faut insérer une porte AND avec *reset'* en entrée, comme montré ci-dessous en figure (e), tout à droite. Cette porte AND peut être vue comme la simplification d'un MUX commandé par *reset* et laissant rentrer 0 si reset=1. L'insertion est faite juste à l'entrée de la bascule, pour que cette remise à 0 soit prioritaire ; en aucun cas juste derrière, sinon la sortie remise à 0 serait de type Mealy.

En fin de PC6, on a pu réaliser un *compteur* d'événements en utilisant une retenue entrante comme signal d'entrée binaire x. Ici, on souhaite encore *tenir des comptes*, mais avec un signal d'entrée x désormais entier, et dont certaines valeurs seulement vont être *prises en compte*.

1c) Associer un additionneur à un *registre resettable* pour que, à chaque top d'horloge, celui-ci augmente son contenu de la valeur de l'entrée entière x, mais seulement lorsqu'un signal de commande *fill* est à 1. Puis dessiner la tranche, au niveau portes.

Utiliser le signal *fill* comme entrée *load* sur le registre permet d'obtenir le comportement souhaité par un simple rebouclage via un additionneur numérique, comme montré en (a) ci-dessous. La tranche s'en déduit immédiatement en (b). On n'expose pas l'intérieur du FA, porte complexe désormais bien connue.



1d) En fait, n'y aurait-il pas une solution un peu moins coûteuse ?

Ce qu'on réalise est une somme récursive entière. C'est un système dynamique discret dont la loi d'évolution est $\mathbf{q}^+ = \mathit{reset'}\cdot(\mathbf{q} + \mathit{fill}\cdot\mathbf{x})$. Dans cette équation, on multiplie des entiers en gras par des booléens en italique (ce qui est parfaitement) licite. Or, dans la tranche, ces multiplications peuvent se faire avec des portes AND, sans utiliser de multiplexeurs, permettant une économie en transistors. L'implantation naturelle de cette équation d'évolution conduit à utiliser non pas un registre, mais une simple bascule D numérique précédée d'une remise à 0.

On représente ce couple sous la forme d'une *bascule resettable* improvisée sur la figure (c). Quant à la multiplication par fill, on la laisse – faute de meilleure notation - sous forme de MUX au niveau numérique, ce qui correspond en fait à la loi d'évolution suivante, employant une notation du langage C : $\mathbf{q}^+ = reset' \cdot [\mathbf{q} + (\text{fill? } \mathbf{x} : \mathbf{0})]$. Par contre, on passe bien à des portes AND dans la tranche de la figure (d), réalisant effectivement une économie par rapport à la figure (b).

2) Piloter la calculette primitive du CM7 pour multiplier par 12

En CM7/P9 a été introduit un chemin de données (CD) élémentaire à deux registres, baptisé « calculette primitive » et en P10 un *programme* pour y calculer la multiplication par 5 sur une donnée entière. Une unité de commande UC×5 a ensuite été conçue pour faire exécuter ce programme au CD. On souhaite désormais y réaliser des multiplications par 12. Il faut donc concevoir une nouvelle unité de commande UC×12, en suivant la méthodologie de synthèse introduite au CM6 et expérimentée en CM7/P14-15-17.

2a) Les premières « instructions » appliquées par l'UC au CD installent la donnée *d* dans R2, puis dans R1. Ensuite le programme de multiplication par 5 peut être représenté par la formule suivante : (d×2+0)×2+d. Quelle est la formule correspondante pour la multiplication par 12 ? Quelle conséquence sur la durée du calcul ?

Il faut 4 bits pour exprimer l'entier 12, contre 3 pour l'entier 5. D'où un niveau de parenthèses en plus : [(d×2+d)×2+0]×2+0. C'est donc un cycle (une période) d'horloge de plus où l'on doit exploiter les ressources de calcul de notre calculette.

2b) Première étape de la méthodologie de synthèse, dessiner le diagramme d'état de la nouvelle unité de commande, baptisée UC×12, en ne faisant apparaître qu'un seul signal de sortie : *sel2*.

Par rapport au diagramme du cours correspondant à la multiplication par 5, il suffit d'insérer un nouvel état F entre E et A, reliés par des transitions inconditionnelles.

La sortie *sel2* , qui commande l'ajout ou pas de *d* dans la formule précédente, vaut 1 en C et D, puis 0 en E et F. Elle est indifférente en A et B.

Pour la *deuxième* étape de la méthodologie de synthèse, on reste pour l'instant sur le principe d'encodage « naïf » adopté en cours pour UC×5 : chaque état est numéroté suivant son ordre d'apparition dans le diagramme d'état, qui est cyclique, en commençant par 0 pour A. Ces numéros sont codés en tant qu'entiers non signés sur les 3 bits q_2, q_1 et q_0 (du MSB au LSB).

2c) Quels sont les bits d'état de UC×12 et quelles ressources matérielles mobiliseront-ils ?

Les bits d'état sont justement ceux qui servent à coder l'état courant du petit système séquentiel UC×12. Ce sont donc évidemment les 3 bits q_2, q_1 et q_0 ici. Chacun sera porté par une bascule D dans l'implantation à venir.

On aborde maintenant la *troisième* et dernière étape de la synthèse, vers l'implantation.

2d) Concernant la loi de sortie , exprimer seulement la sortie *sel2* en fonction des bits d'état.

Les indéterminations offrent deux possibilités de FDM : sel2=q_1 ou sel2=q_2'.

2e) Elaborer le tableau de Karnaugh fonctionnel vectoriel (*tout en un*) fournissant l'état futur en fonction de l'état courant et de l'entrée *start*. Puis exprimer la loi/fonction de transition.

état présent q_0 q_1 q_2 / état futur	q_1 = 00	q_1 = 01	q_1 = 11	q_1 = 10
q_0 = 0, start = 0	A / 0 / 0 / A	C / 1 / 1 / D	*1* / *1* / *1*	E / 1 / 1 / F
q_0 = 0, start = 1	B / 0 / 1 / C	D / 0 / 0 / E	*0* / *0* / *1*	F / 0 / 0 / A
q_0 = 1, start = 0	B / 0 / 1 / C	D / 0 / 0 / E	*0* / *0* / *1*	F / 0 / 0 / A
q_0 = 1, start = 1	A / 1 / 0 / B	C / 1 / 1 / D	*1* / *1* / *1*	E / 1 / 1 / F

q_1 ; q_2

état présent q_0 q_1 q_2 / état futur	q_1 = 00	q_1 = 01	q_1 = 11	q_1 = 10
q_0 = 0, start = 0	A / 0 / 0 / A	*0* / *0* / *1*	E / 0 / 1 / F	F / 0 / 0 / A
q_0 = 0, start = 1	B / 1 / 1 / C	C / 1 / 1 / D	D / 0 / 1 / E	*0* / *1* / *0*
q_0 = 1, start = 0	B / 1 / 1 / C	C / 1 / 1 / D	D / 0 / 1 / E	*0* / *1* / *0*
q_0 = 1, start = 1	A / 1 / 0 / B	*1* / *0* / *1*	E / 0 / 1 / F	F / 0 / 0 / A

q_1 ; q_2

$\mathbf{q_0^+} = q_0'\text{start} + q_0'q_1 + q_0'q_2 = q_0'(\text{start}+q_1+q_2)$

$q_1^+ = q_2'q_1'q_0 + q_1q_0'$

$q_2^+ = q_2q_0' + q_1q_0$

$\mathbf{q_0^+} = q_2'q_0 + q_2'\text{start} = q_2'(q_0+\text{start})$

$q_1^+ = q_0$

$q_2^+ = q_1$

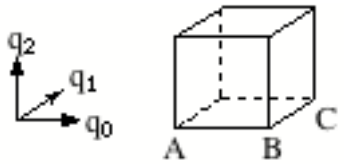
Voir ci-dessus à gauche (*le côté droit concerne une question ultérieure*). Les 0 ou 1 en italique correspondent aux cases indéterminées, car inutilisées par le diagramme d'état. Ils sont donc choisis pour simplifier l'expression de la loi de transition.

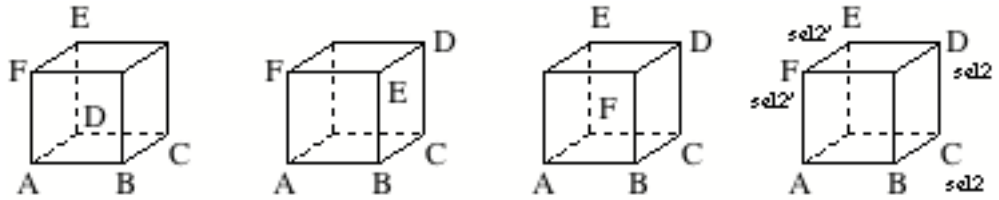
2f) Si l'automate se trouve par hasard initialisé dans l'état $q_2q_1q_0$=110, que se passe-t-il ?

Cet état se trouve hors du diagramme : il n'est donc pas censé être utilisé. Vu la loi de transition adoptée sur les cases indéterminées, il transite successivement vers 111 (également hors diagramme), puis vers E, F et enfin A, aux tops d'horloge suivants. Il y a donc 4 cycles pendant lesquels la calculette fait n'importe quoi. Ensuite, elle se met à marcher correctement.
Est-ce un problème ? En tout cas, le problème est réel car on ne sait pas dans quel état s'initialise une bascule D lorsqu'on la met sous tension. Est-ce grave ? Tout dépend de l'application : il y a des appareils dont on tolère qu'ils aient à « chauffer » un peu avant de fonctionner correctement. Pour les autres, on ajoutera un signal d'entrée pour commander une initialisation dans l'état A (c'est-à-dire remise à 0 des 3 bits d'état) et on lui fera subir une impulsion avant d'utiliser la calculette.

Au lieu d'achever la *troisième* étape de la synthèse, en implantant fonctions de transition et de sortie, on revient à la *deuxième*, où l'on souhaite désormais utiliser un encodage d'états adjacent, espérant une loi de transition plus simple.

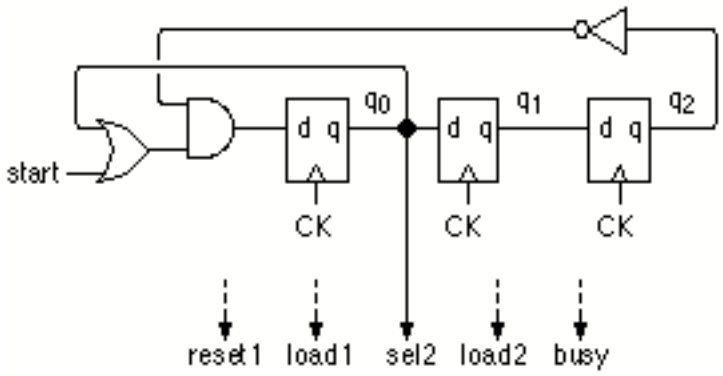
2g) Comme illustré sur la figure 3D ci-contre, on impose le code $q_2q_1q_0$ des 3 premiers états du cycle : 000 pour A, 001 pour B et 011 pour C (en fait, aux symétries du 3-cube près, ce choix n'en est pas un). Quels sont les différents encodages adjacents respectant cette contrainte ?





2h) Choisir celui qui présente une symétrie centrale, puis réaliser complètement la troisième étape de la synthèse. En termes plus précis, établir le tableau et la loi d'évolution qui lui correspondent, en profitant au maximum des cas indéterminés, puis implanter.

Le seul encodage présentant une symétrie centrale est celui le plus à droite ci-dessus. Le tableau de Karnaugh tout-en-un correspondant est présenté une page plus haut, à droite. Après minimisation, il apparaît une remarquable simplification de la loi de transition : l'encodage adjacent fonctionne à merveille ici (la symétrie centrale y est pour quelque chose).

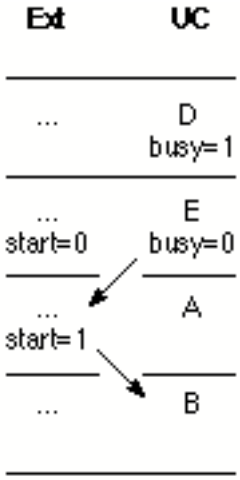


Cette loi de transition est en fait proche de celle d'un registre à décalage, d'où le schéma ci-dessous dans lequel les bascules sont arrangées horizontalement plutôt que verticalement comme il est d'usage. Seule la sortie *sel2* est précisée ici. Comme les autres, elle a changé avec le nouvel encodage : elle vaut désormais q_0 (voir la figure d'avant, à droite).

2i) Faire fonctionner la logique de transition, pour vérifier qu'elle décrit bien un cycle à 6 états.

Sans difficulté.

2j) Expédié en cours, le cas de *busy* mérite plus d'attention. Cette sortie de UC×5 comme de UC×12 sert d'entrée à un système séquentiel synchrone externe, cadencé lui aussi par CK, qui recourt au service de multiplication fourni par le couple UC + CD. Lorsque le signal *busy* retombe à 0, c'est un message adressé par l'UC à ce système externe pour lui indiquer qu'il peut demander une nouvelle multiplication s'il le souhaite. Il le dira en mettant *start* à 1. En cas de nombreuses multiplications à réaliser en série, il convient d'envoyer ce message dès que possible pour permettre de les enchaîner sans délai. Les valeurs affectées à *busy* sur le diagramme d'état de UC×5 vu en CM7 répondent-elles à cette exigence d'efficacité ?



Voilà qui soulève la question des relations temporelles entre circuits séquentiels synchrones. Il a été montré en cours (CM7/P22) que le CD a une période d'horloge de retard sur l'UC. Plus précisément, les valeurs de sortie correspondant à l'état courant de l'UC, qui apparaissent au sein de l'UC lors de la période d'horloge courante, ne sont prises en compte par le CD qu'au prochain top d'horloge et n'y auront donc d'effet qu'au cours de la période d'horloge suivante. Pour les mêmes raisons, il faut une période d'horloge à l'UC et au système externe pour pouvoir agir l'un sur l'autre.

Enchaîner les multiplications au plus vite suppose que l'on ne reste pas plus d'un cycle (une période d'horloge) dans l'état A lorsqu'on passe d'une multiplication à la suivante. Pour cela, le système externe doit mettre *start* à 1 aussitôt l'UC dans l'état A – c'est-à-dire dans la même période – afin qu'elle passe à l'état B au prochain top d'horloge, comme illustré ci-contre. Mais pour cela, il faut que le système externe soit prévenu, par le retour à 0 de *busy*, lors de la période d'horloge précédente, lorsque l'UC est dans l'état précédant A dans le diagramme d'état, c'est-à-dire E (pour UC×5). Le diagramme d'état de UC×5 présenté en CM7 doit donc être modifié, en mettant *busy* à 0 dans l'état E.

A titre subsidiaire, on note que la cadence des multiplications pourrait encore être accrue si l'on pouvait sauter de E en B directement, court-circuitant A. Il « suffit » pour cela de mettre en

place des transitions conditionnelles partant de E et allant vers A ou B selon *start*. Mais alors il faut aussi revoir la loi d'évolution…

A l'issue de cet exercice, il est évident que la réalisation purement matérielle d'unités de commande manque de souplesse. Il serait plus commode de pouvoir stocker dans une mémoire les valeurs des signaux de commande correspondant à chaque instruction. Ces valeurs seraient alors plus aisément modifiables pour obtenir différents « programmes ». Le séquencement lui-même pourrait-il être mis en mémoire ? Ces aspirations sont celles qui ont conduit à la conception des microprocesseurs, dernière étape du cours ES102. Mais un microprocesseur aura lui aussi besoin d'unités de commande, plus complexes…