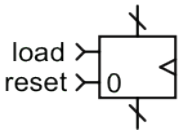
ES102/PC7 : énoncé

1) Bascule D numérique en boucle

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

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 ;-)



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.

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

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 : $(d\times2+0)\times2+d$. Quelle est la formule correspondante pour la multiplication par 12 ? Quelle conséquence sur la durée du calcul ?

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*.

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 ?

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

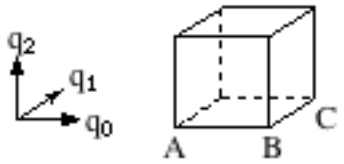
2d) Concernant la loi de sortie , exprimer seulement la sortie *sel2* en fonction des bits d'état.

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.

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

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.

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

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 ?

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…