ASR1 – TD6 : Circuits asynchrones

{ Andreea.Chis, Matthieu.Gallet, Bogdan.Pasca } @ens-lyon.fr

23 et 24 Octobre 2008

1 Préliminaires

1.1 Sans horloge, la fête est plus folle

1. Rappelez quels sont les avantages des circuits asynchrones par rapport à leurs équivalents synchrones.

On se place ici dans le cadre de la logique asynchrone avec protocole 4-phases et encodage double-rail.

2. Malgré les apparences, la phrase précédente est en français. Que signifie-t-elle ?

1.2 La porte de Muller

On rappelle ici la table de vérité et le symbole de la porte de Muller (aussi appelée *C-element*) :

x	y	r
0	0	0
0	1	r
1	0	r
1	1	1

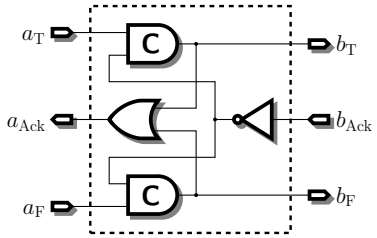
1. À quoi ce machin-là peut-il bien servir ?

2. Comment peut-on le réaliser à l'aide de portes de base ?

3. Et si l'on s'autorise des transistors en technologie CMOS, peut-on faire plus simple ?

2 Registres asynchrones

On se donne le composant suivant, appelé demi-buffer (ou *half-buffer*) :



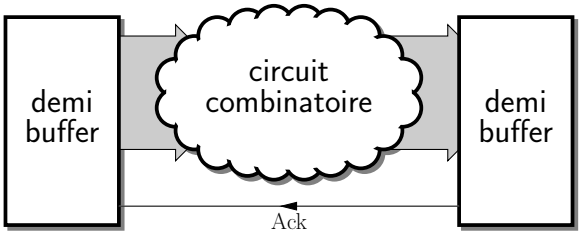
On suppose que les fils de données (a_T, a_F, b_T, b_F) sont préchargés à 0.

1. Essayez de comprendre le fonctionnement de ce demi-buffer, en supposant qu'il est connecté par son port d'entrée (a) et son port de sortie (b) à des circuit respectant le protocole 4-phases.
2. Mettez six demi-buffers bout-à-bout et observez leur comportement en mettant en entrée tour-à-tour :
   – un producteur, qui met une donnée sur (a_T, a_F) dès que l'acquittement a_{Ack} est à 0 puis l'enlève lorsque a_{Ack} passe à 1 ; ou
   – rien du tout (donnée constante à *Invalide*) ;
   et en sortie :
   – rien du tout (acquittement b_{Ack} constant à 1) ; ou
   – un consommateur, qui met son acquittement b_{Ack} dès qu'une donnée valide est présente sur (b_T, b_F), puis le repasse à 0 lorsque cette donnée redevient invalide.
3. Combien de valeurs valides peut stocker ce circuit de six demi-buffers ? Justifiez l'appellation de demi-buffer.
4. Comment construire un demi-buffer pour des valeurs sur plus d'un bit (plusieurs paires de rails de données, mais un seul acquittement) ?

3 Calculs asynchrones

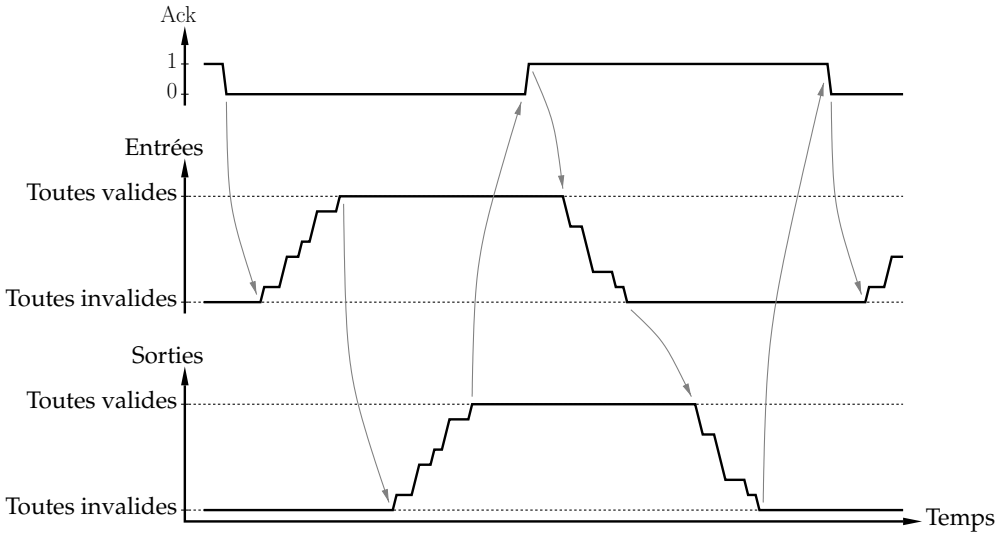
3.1 Concepts de synchronisation

On souhaite maintenant effectuer des calculs sur les données. On insère donc pour cela de la logique combinatoire entre nos étages de demi-buffers :



On distingue cependant deux types de comportement pour ces circuits combinatoires :

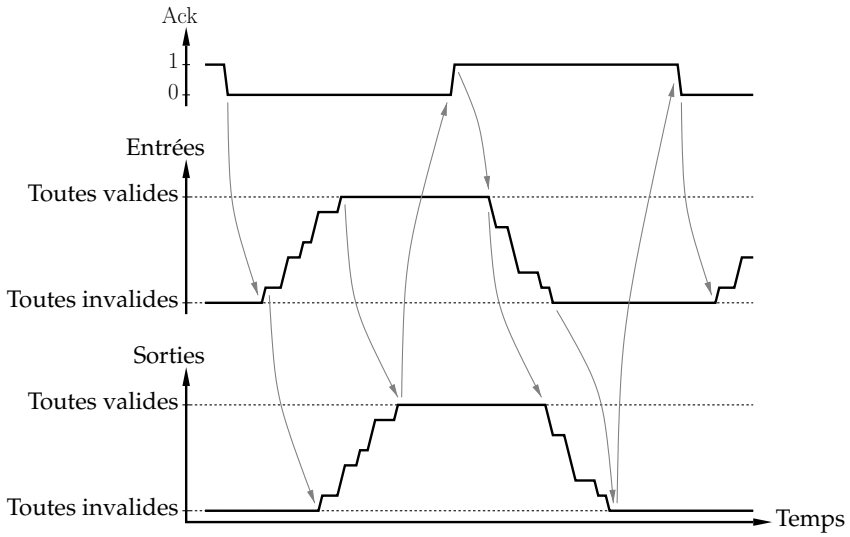
Circuits fortement indicatifs Dans ce type de circuit, les sorties ne commencent à être valides que lorsque toutes les entrées le sont. De même, lors du retour des entrées à l'état invalide, les sorties ne redeviennent invalides que lorsque toutes les entrées le sont déjà. Cela peut se voir sur le chronogramme suivant, dans lequel les flèches grises indiquent les contraintes de séquentialité temporelle :



Circuits faiblement indicatifs Ce type de circuit est sensiblement moins contraint : des sorties peuvent devenir valides dès que certaines entrées le sont, et de même, elles peuvent redevenir invalides dès que certaines entrées le sont aussi.

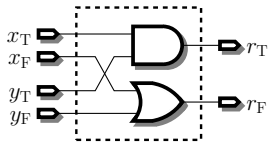
Pour un fonctionnement correct il est nécessaire que circuit ne valide pas toutes les sorties tant que toutes les entrées ne sont pas valides, et ne mette pas toutes les sorties à invalide tant que toutes les entrées ne sont pas devenues invalide.

On a alors le chronogramme type suivant :



3.2 Portes de base

1. Que dire de la porte suivante ?



2. Est-elle fortement indicative ? faiblement indicative ?
3. Quel est le problème que pose cette porte ?
4. Comment faire un ET fortement indicatif ?
5. Quel est le coût (en transistors CMOS) de votre porte ? Peut-on faire mieux ?
6. Faites de même pour les portes OU et XOR.
7. Quel est le coût de la porte NON ?

3.3 Additions

1. Utilisez la méthode DIMS (*Delay-Insensitive Minterm Synthesis*) pour construire un *half-adder* fortement indicatif.
2. De même pour un *full-adder*.
3. Assemblez tout ça pour faire un additionneur 8 bits avec propagation de retenue.
4. Quel est le délai (en nombre de portes traversées) de cet additionneur ?
5. Voyez-vous un moyen d'aller plus vite ? À quel prix ?
   Astuce : étudiez les propriétés de propagation de la retenue dans un *full-adder* en fonction des opérandes.
6. Allez-vous vraiment plus vite ?
7. Comment faire mieux ?

Pour en savoir plus, n'hésitez pas à aller à la bibliothèque emprunter le bouquin de Jens Sparsø intitulé *Principles of asynchronous circuit design – A systems perspective*. Les quelques 200 premières pages sont d'ailleurs un très bon tutoriel sur la logique asynchrone et sont de surcroît disponibles librement sur le net à l'adresse suivante :
`http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=855`