

Circuits logiques.

partie 2 : circuits séquentiels

1 Circuits élémentaires de mémorisation : les bascules

- Bascule RS
- Verrou (*latch*)
- Bascule (*flip-flop*)
- Registres

2 Automates

- Notion d'automate fini
- Automate fini synchrone

3 Banc de registres et RAM

- Banc de registres
- Généralités sur la RAM
- Mémoire SRAM

Introduction

Dans les circuits combinatoires la notion de temps joue un rôle « mineur » : même si tout circuit combinatoire présente un certain délai de stabilisation, les valeurs prises par les sorties du circuit ne dépendent que de celles des entrées.

Dans cette partie, nous passerons en revue certains circuits dits *séquentiels*, dans lesquels la notion de chronologie des événements joue un rôle central.

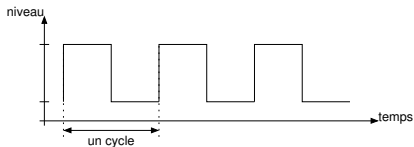
Un circuit séquentiel peut se trouver dans un nombre fini d'états, et les entrées du circuit provoquent des transitions d'un état à un autre à des instants précis.

Nous présenterons un certain nombre de circuit séquentiels de base pour la mémorisation. Nous présenterons aussi une manière assez générale de définir et de construire des circuits séquentiels : les automates finis synchrones.

Pour respecter les relations temporelles requises, de nombreux circuits numériques disposent d'une horloge.

Une horloge est un circuit qui émet un signal créneau périodique, dont les valeurs extrêmes correspondent au niveau 0 et au niveau 1.

On utilisera des signaux d'horloges de la forme :



Ce signal permet de synchroniser des événements,

- soit sur le front montant,
- soit sur le front descendant.

Plan

1 Circuits élémentaires de mémorisation : les bascules

- Bascule RS
- Verrou (*latch*)
- Bascule (*flip-flop*)
- Registres

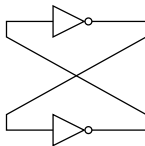
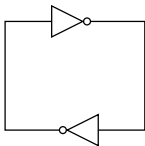
2 Automates

- Notion d'automate fini
- Automate fini synchrone

3 Banc de registres et RAM

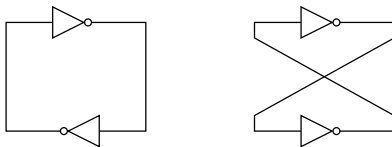
- Banc de registres
- Généralités sur la RAM
- Mémoire SRAM

L'idée de base vient du fait que l'on sait concevoir des circuits ressemblant au circuit suivant (qui n'est pas un circuit combinatoire bien formé) :

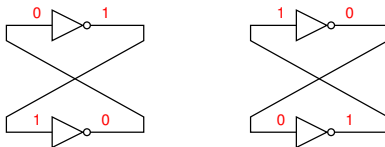


Ce circuit présente deux états stables :

L'idée de base vient du fait que l'on sait concevoir des circuits ressemblant au circuit suivant (qui n'est pas un circuit combinatoire bien formé) :



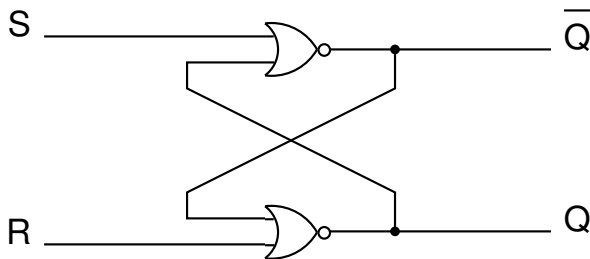
Ce circuit présente deux états stables :



Le tout est de savoir comment passer d'un état à un autre...

Bascule RS

Pour pouvoir passer d'un état à un autre, on utilise le type de montage suivant :

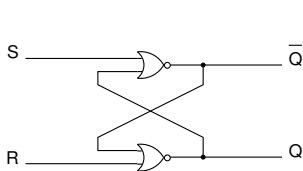


Ce circuit est appelé *bascule RS*. Il possède deux entrées :

- S pour set,
- R pour reset.

Le circuit possède deux sorties (normalement) complémentaires Q et \overline{Q} .

Contrairement à ce qui se passe dans le cas des circuits combinatoires, les valeurs de sortie ne sont pas uniquement déterminées par les variables d'entrée.

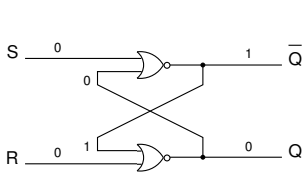


S	R	Q	Q ⁺
0	0	0	
0	0	1	
⌋	0	0	
⌋	0	1	
0	⌋	0	
0	⌋	1	
⌋	⌋	x	

On va énumérer tous les cas possibles :

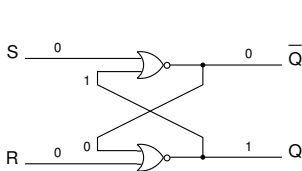
- A chaque fois, on se place dans un état possible pour S, R et Q.
- Q⁺ désigne l'état suivant de Q, *i.e.*, le prochain état stable de Q.
- le symbole ⌋ indique qu'un signal passe momentanément à 1.

Pour mémoire : $\text{NOR}(a, b) = 1$ ssi $a = b = 0$.



S	R	Q	Q ⁺
0	0	0	0
0	0	1	
1	0	0	
1	0	1	
0	1	0	
0	1	1	
1	1	x	

état stable 0



S	R	Q	Q ⁺
0	0	0	0
0	0	1	1
1	0	0	
1	0	1	
0	1	0	
0	1	1	
1	1	x	

état stable 0

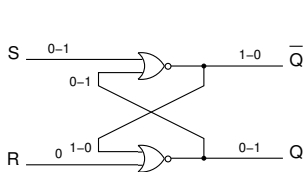
état stable 1

La bascule présente deux états stables :

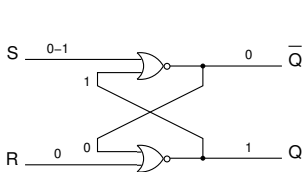
- l'état $Q = 0$,
- l'état $Q = 1$.

Quand $R = S = 0$, Q et \bar{Q} ne peuvent pas prendre la même valeur :

- si les deux valent 0, les portes NOR retournent 1 : impossible.
- si les deux valent 1, les portes NOR retournent 0 : impossible.

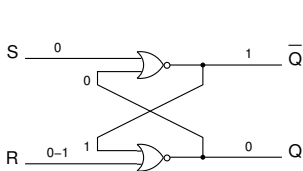


S	R	Q	Q ⁺	
0	0	0	0	état stable 0
0	0	1	1	état stable 1
1	0	0	1	set
1	0	1		
0	1	0		
0	1	1		
1	1	x		

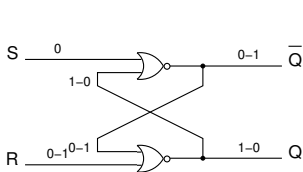


S	R	Q	Q ⁺	
0	0	0	0	état stable 0
0	0	1	1	état stable 1
1	0	0	1	set
1	0	1	1	set
0	1	0		
0	1	1		
1	1	x		

Si S prend momentanément la valeur 1, la bascule prend l'état $Q = 1$.

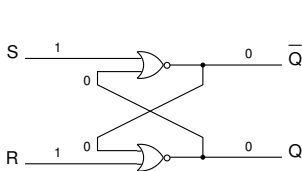


S	R	Q	Q ⁺	
0	0	0	0	état stable 0
0	0	1	1	état stable 1
1	0	0	1	set
1	0	1	1	set
0	1	0	0	
0	1	1		
1	1	x		



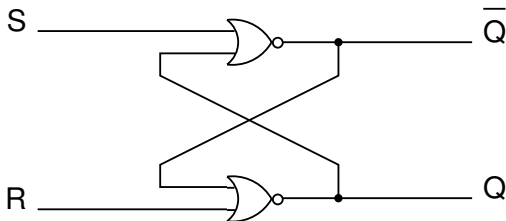
S	R	Q	Q ⁺	
0	0	0	0	état stable 0
0	0	1	1	état stable 1
1	0	0	1	set
1	0	1	1	set
0	1	0	0	reset
0	1	1	0	reset
1	1	x		

Si R prend momentanément la valeur 1, la bascule prend l'état $Q = 0$.



S	R	Q	Q ⁺	
0	0	0	0	état stable 0
0	0	1	1	état stable 1
1	0	0	1	set
1	0	1	1	set
0	1	0	0	reset
0	1	1	0	reset
1	1	x	?	indéterminé

Quand $R=S=1$, le seul état stable possible est $Q = \overline{Q} = 0$. Dès que l'un des deux signaux R ou S retombe à 0, la bascule retombe dans un état stable : lequel ? Dans ce cas, on ne sais pas conclure ; la situation devra être évitée.



En résumé :

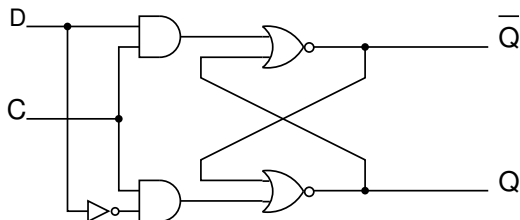
- si S (Set) prend momentanément la valeur 1, la bascule prend l'état $Q=1$;
- si R (Reset) passe momentanément à 1, la bascule prend l'état $Q=0$;

La bascule se souvient donc qui de R ou de S a été activé pour la dernière fois : c'est tout ce dont on a besoin pour mémoriser un bit.

Par contre, reste le problème du $R = S = \perp \dots$

Verrou (*latch*)

On interdit la possibilité du $S = R = 1$, et on ajoute un signal de commande C :

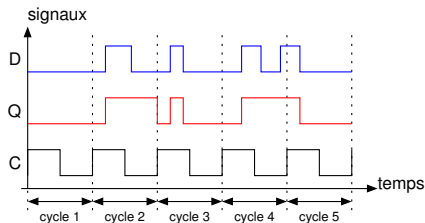


On obtient un *verrou* commandé par le signal C :

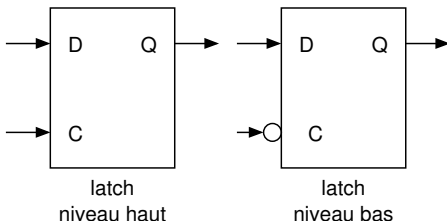
- si $D=1$ et que la commande $C=1$, la bascule passe à l'état $Q=1$,
- si $D=0$ et que la commande $C=1$, la bascule passe à l'état $Q=0$,
- quand $C=0$, le verrou ne peut plus changer d'état.

Lorsque C est activé, la valeur D est stockée : mémoire 1 bit.

Ce verrou stocke D lorsque C est à son niveau haut. Souvent C est un signal d'horloge : on dit qu'il s'agit d'un verrou *régi par le niveau haut de l'horloge*.



Il existe aussi des verrous régis par le niveau bas de l'horloge.

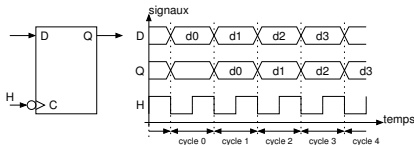


Bascule (*flip-flop*)

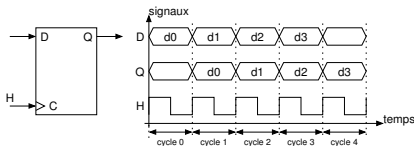
La mémorisation se produit à la fin d'un cycle, et le bit mémorisé est maintenu sur la sortie de la *bascule* pendant tout le cycle suivant.

On peut mettre au point deux types de bascules.

- Bascule régie par le *front descendant de l'horloge* :

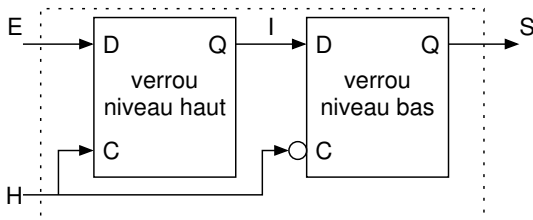


- Bascule régie par le *front montant de l'horloge* :

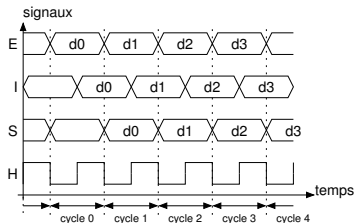


On peut raisonner de la même manière avec les deux types de bascules : il suffit d'aligner correctement la définition des cycles d'horloge.

En plaçant en série un verrou régi par le niveau haut de l'horloge, puis un autre régi par le niveau bas, on obtient une bascule :

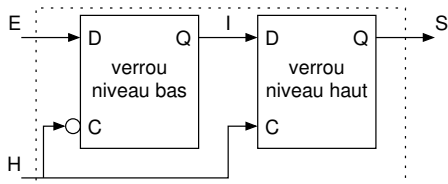


Alors, la bascule mémorise l'entrée E sur le *front descendant de l'horloge H* :

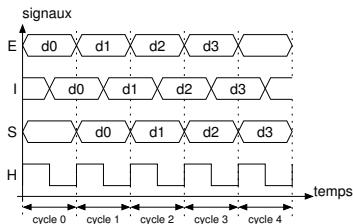


Comment faire une bascule mémorisant sur le front montant de l'horloge ?

Pour obtenir une bascule régie par le front montant, il suffit de placer en série un verrou régi par le niveau bas de l'horloge, puis un autre régi par le niveau haut :



Dans ce cas, la bascule mémorise l'entrée E sur le *front montant de l'horloge* H :



Pour fixer les idées, on supposera par la suite que l'on travaille avec des

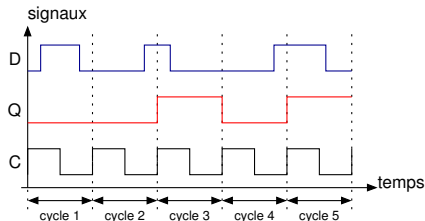
basclues régies par le front montant de l'horloge.

Registres

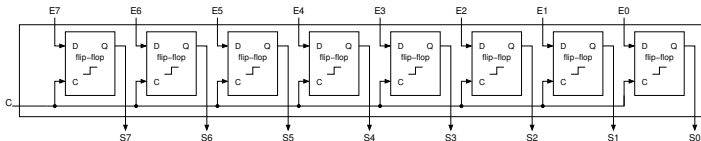
Nos registres sont fabriqués à partir de bascules régies par un front d'horloge. Cette convention fixe la manière d'utiliser les bascules :

- sur un cycle d'horloge on calcule un résultat r_i , qui est mémorisé sur l'entrée D de la bascule à la fin du cycle ;
- au cours du cycle suivant :
 - ▶ le résultat du cycle précédent r_i est disponible et constant sur la sortie Q ;
 - ▶ on calcule un résultat r_{i+1} , mémorisé à la fin du cycle.

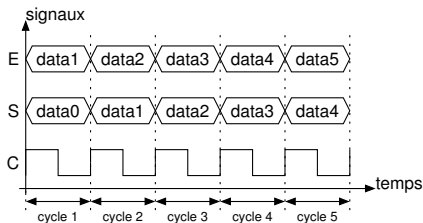
Le résultat calculé au cycle i est typiquement la sortie d'un circuit combinatoire.



Le registre le plus simple que l'on puisse imaginer consiste simplement à monter des flip-flops en parallèle, en les connectant au même signal d'horloge :

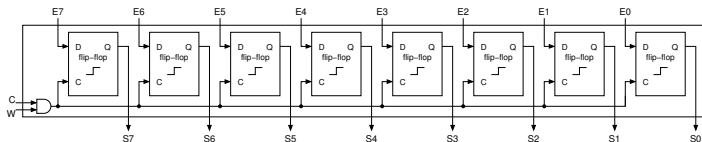


On obtient le type de chronogramme suivant :

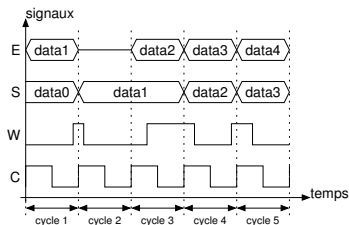


A chaque fin de cycle, l'entrée E est stockée et est disponible au cycle suivant.

Pour mémoriser une donnée sur plusieurs cycles, on ajoute un signal W :



Voici un exemple de chronogramme :



A chaque fin de cycle,

- soit $W=1$ et l'entrée E est stockée pour être disponible au cycle suivant ;
- soit $W=0$ et la sortie S conserve la même donnée au cycle suivant.

Plan

1 Circuits élémentaires de mémorisation : les bascules

- Bascule RS
- Verrou (*latch*)
- Bascule (*flip-flop*)
- Registres

2 Automates

- Notion d'automate fini
- Automate fini synchrone

3 Banc de registres et RAM

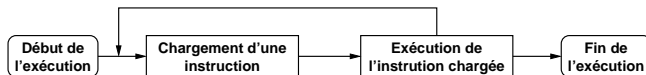
- Banc de registres
- Généralités sur la RAM
- Mémoire SRAM

Notion d'automate fini

Il faut un formalisme prenant en compte le temps dans les circuits séquentiels. . .

Exemples :

- Un passage à niveau doit réagir à l'événement « un train approche » en fermant ses barrières et en faisant passer des feux au rouge. Ensuite, il doit réagir à l'événement « le train s'éloigne » en les ouvrant. . .
- Un processeur doit passer successivement par les étapes du cycle d'instruction.



On va définir des *automates finis* comme une abstraction facile à dessiner du *comportement attendu d'un circuit séquentiel*. Le comportement, c'est le pendant d'une fonction booléenne, qui décrit un circuit logique.

Cette abstraction est *a priori* asynchrone : l'automate réagit à des événements, sans qu'il y ait nécessairement de synchronisation avec une horloge. Mais la mise au point d'automates asynchrones pose de nombreux problèmes : on sait les résoudre, mais cela sort du cadre de ce cours. . .

On s'intéressera donc rapidement à des *automates finis synchrones*, dans lesquels les événements sont synchronisés par un signal d'horloge.

Formellement, un *automate fini* est un quadruplet (S, E, T, s_0) où

- S est un ensemble fini d'états (*States*) ;
- E est un ensemble fini d'événements (*Events*) ;
- T est une *fonction de transition* de $S \times E \rightarrow S$;
- $s_0 \in S$ est un état spécial appelé *état initial*.

Graphiquement, on fait des dessins avec

- un rond par état, et une flèche pour indiquer l'état initial ;
- des flèches d'un état s_i à un état s_j chaque fois qu'il existe une transition de s_i à s_j . On étiquette la flèche avec l'événement qui déclenche la transition.

Exemple : un passage à niveau a deux capteurs (CG et CD), à 300m de chaque côté, qui sont normalement à 0, et sont à 1 lorsqu'un train pèse dessus.

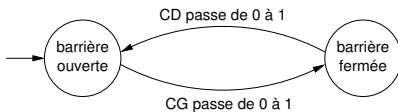
Si on considère que les trains ne peuvent passer que de gauche à droite :

- deux états : $S = \{\text{barrière ouverte, barrière fermée}\}$;
- deux événements : $E = \{\text{CG passe de 0 à 1, CD passe de 0 à 1}\}$;
- état initial : $s_0 = \text{barrière ouverte}$.

Exemple : un passage à niveau a deux capteurs (CG et CD), à 300m de chaque côté, qui sont normalement à 0, et sont à 1 lorsqu'un train pèse dessus.

Si on considère que les trains ne peuvent passer que de gauche à droite :

- deux états : $S = \{\text{barrière ouverte}, \text{barrière fermée}\}$;
- deux événements : $E = \{\text{CG passe de 0 à 1}, \text{CD passe de 0 à 1}\}$;
- état initial : $s_0 = \text{barrière ouverte}$.

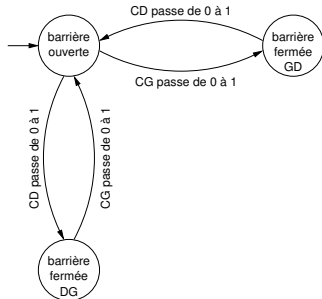


Mais un train peut en cacher un autre :

- états : $S = \{\text{barrière ouverte, barrière fermée GD, barrière fermée DG}\}$;
- événements : $E = \{\text{CG 0 à 1, CD 0 à 1}\}$;
- état initial : $s_0 = \text{barrière ouverte}$.

Mais un train peut en cacher un autre :

- états : $S = \{\text{barrière ouverte}, \text{barrière fermée GD}, \text{barrière fermée DG}\}$;
- événements : $E = \{\text{CG 0 à 1}, \text{CD 0 à 1}\}$;
- état initial : $s_0 = \text{barrière ouverte}$.



Automate fini synchrone

Dans les d'automates finis, on considère qu'un événement peut se produire à n'importe quel instant : on prend surtout en compte la relation de cause à effet.

Pour simplifier, on introduit une horloge, et on suppose que les transitions

- ne peuvent intervenir que sur un front montant de l'horloge,
- se produisent suffisamment rapidement pour qu'on puisse les considérer comme instantanées.

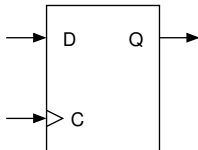
De plus, on remplace l'ensemble des événements E par un ensemble de valeurs d'entrées possibles I (*Inputs*). Un évènement est donc modélisé par le fait que l'entrée de l'automate a une certaine valeur lors d'un front montant de l'horloge.

Un *automate fini synchrone* est un quadruplet (S, I, T, s_0) où

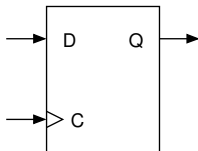
- S est un ensemble fini d'états (*States*) ;
- I est un ensemble fini d'entrées (*Inputs*) ;
- T est une *fonction de transition* de $S \times I \rightarrow S$;
- $s_0 \in S$ est un état spécial appelé *état initial* ;
- une transition se produit à chaque front montant d'une horloge.

Rem : on aurait pu se baser sur le front descendant d'une horloge ; l'important est qu'une transition se produise à chaque fin de cycle.

Exemple : Une bascule régie par le front montant de l'horloge cadre parfaitement avec le modèle d'automate synchrone décrit précédemment.

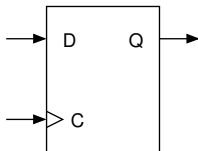


Exemple : Une bascule régée par le front montant de l'horloge cadre parfaitement avec le modèle d'automate synchrone décrit précédemment.

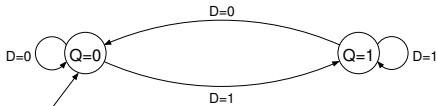


- états : $S = \{Q=0, Q=1\}$;
- entrées : $I = \{D=0, D=1\}$ (valeurs possibles pour le signal D) ;
- une transition ne peut se produire que sur un front montant de l'horloge C.

Exemple : Une bascule régée par le front montant de l'horloge cadre parfaitement avec le modèle d'automate synchrone décrit précédemment.



- états : $S = \{Q=0, Q=1\}$;
- entrées : $I = \{D=0, D=1\}$ (valeurs possibles pour le signal D) ;
- une transition ne peut se produire que sur un front montant de l'horloge C.

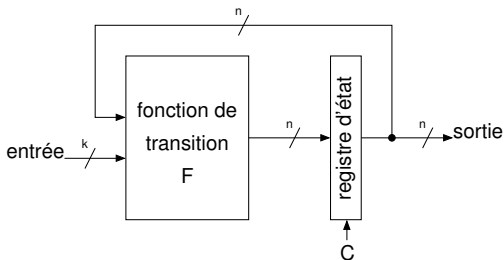


Un automate fini synchrone peut être implanté en matériel de la façon suivante.

- Un registre à n bits stocke l'état courant de l'automate.
- La fonction de transition F est réalisée par un circuit purement combinatoire. Cette fonction prend en entrée :
 - ▶ n signaux provenant du registre d'état de l'automate,
 - ▶ k signaux codant l'entrée courante de l'automate ;

Elle ressort l'état suivant de l'automate, qui sera stocké par le registre d'état sur le prochain front montant de l'horloge.

Dans un *automate de Moore*, la sortie est simplement son état courant :



Cela donne 2^n états possibles, et 2^k entrées distinctes possibles pour l'automate.

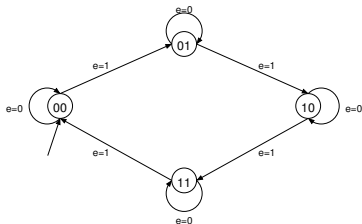
Exemple : construire un compteur 2 bits modulo 4, qui s'incrmente si son entrée $e = 1$, et qui conserve sa valeur si $e = 0$.

Exemple : construire un compteur 2 bits modulo 4, qui s'incrmente si son entrée $e = 1$, et qui conserve sa valeur si $e = 0$.

Notons $(q_1 q_0)$ le registre d'état de l'automate que l'on cherche à construire.

Exemple : construire un compteur 2 bits modulo 4, qui s'incrmente si son entrée $e = 1$, et qui conserve sa valeur si $e = 0$.

Notons $(q_1 q_0)$ le registre d'état de l'automate que l'on cherche à construire.

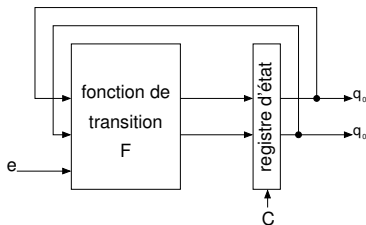


q_1	q_0	e	$F_1(q_1, q_0, e)$	$F_0(q_1, q_0, e)$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

q_1	q_0	e	$F_1(q_1, q_0, e)$	$F_0(q_1, q_0, e)$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

On trouve $F_0(q_1, q_0, e) = q_0 \oplus e$, et $F_1(q_1, q_0, e) = q_1 \oplus (q_0 e)$. Bref, on sait construire un petit circuit combinatoire qui calcule $F(q_1, q_0, e)$.

On pourra donc implanter facilement l'automate demandé :



En résumé, si on est capable de décrire formellement un automate fini synchrone, on arrivera toujours à déduire de cette description un automate de Moore pour le réaliser sous la forme d'un circuit séquentiel.

Il existe d'autres types d'automates que ceux de Moore.

- Automate de Mealy : la sortie est une fonction de l'état courant (et éventuellement de l'entrée).
- Automate micro-programmé : l'automate place successivement sur sa sortie des valeurs contenues dans une mémoire ROM, en effectuant des sauts en fonction de ses entrées.

On pourrait envisager de concevoir la totalité d'une UCT comme un automate : mais il serait impossible de s'y retrouver parmi tous les états possibles.

Par contre, on peut concevoir l'unité de contrôle de l'UCT comme un automate, chargé d'activer de façon coordonnée les différents circuits de l'UCT.

Plan

1 Circuits élémentaires de mémorisation : les bascules

- Bascule RS
- Verrou (*latch*)
- Bascule (*flip-flop*)
- Registres

2 Automates

- Notion d'automate fini
- Automate fini synchrone

3 Banc de registres et RAM

- Banc de registres
- Généralités sur la RAM
- Mémoire SRAM

Banc de registres

Un *banc de registre* ou *register file* est un circuit qui regroupe un ensemble de registres, chacun identifié par un numéro : chacun peut être lu ou écrit.

Pour permettre les lectures,

- le banc prend en entrée un certain nombre de numéros de registres `#rrX`,
- un nombre identique de *ports de lecture* `rdataX` en sortie.

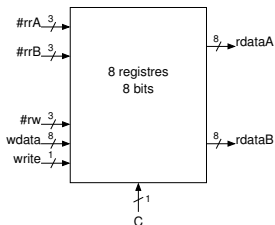
Le banc de registre maintient en permanence sur son port de lecture `rdataX` la donnée contenue dans le registre dont l'indice est spécifié sur l'entrée `#rrX`.

Pour les écritures sur le front montant de l'horloge, le banc présente en entrée

- un numéro de registre `#rw` et un *port d'écriture* `wdata`,
- un signal d'horloge `C` et un signal d'écriture `write`.

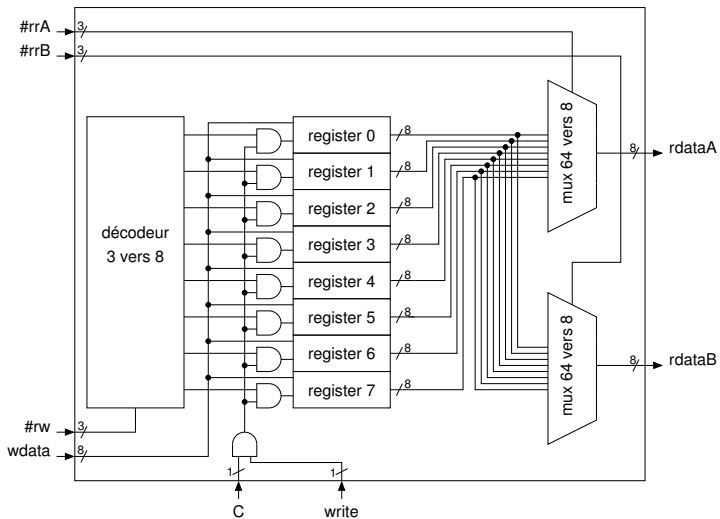
Si sur un front montant de l'horloge `C` le signal `write` est activé, la donnée présente sur le port d'écriture `wdata` est placée dans le registre `#rw`.

Voici par exemple un banc de 8 registres 8 bits, avec deux ports de lecture :



Pour implanter un tel banc de registres, on peut utiliser :

- 8 registres 8 bits ;
- 2 multiplexeurs 64 vers 8 pour sélectionner parmi les registres :
 - ▶ rdataA en fonction de #rrA,
 - ▶ rdataB en fonction de #rrB.
- un décodeur 3 vers 8 pour adresser le registre à écrire d'après #rw.



Que se passe-t-il si le même registre est lu et écrit durant un cycle d'horloge ?

Que se passe-t-il si le même registre est lu et écrit durant un cycle d'horloge ?

En d'autres termes, que ce passe t'il si, à la fin d'un cycle de l'horloge,

- le signal write est activé avec $\#rw = i$, et
- $\#rrA = i$ ou $\#rrB = i$?

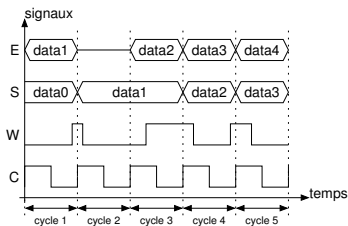
Que se passe-t-il si le même registre est lu et écrit durant un cycle d'horloge ?

En d'autres termes, que ce passe t'il si, à la fin d'un cycle de l'horloge,

- le signal write est activé avec $\#rw = i$, et
- $\#rrA = i$ ou $\#rrB = i$?

Comme l'écriture d'un registre n'intervient que sur le front montant, le registre lu sera valide jusqu'à la fin du cycle d'horloge : la valeur lue sera celle écrite lors de l'un des cycles précédents. La valeur écrite sera disponible au cycle suivant.

Bref, tout se passe comme on l'avait déjà vu pour un seul registre :



Généralités sur la RAM

Les registres et les bancs de registres fournissent les éléments de base pour la construction de petites mémoires.

Pour la construction de mémoire plus grandes, comme la mémoire centrale d'un ordinateur, on doit recourir à d'autres technologies. On distingue essentiellement deux types de mémoires RAM :

- Dans une *SRAM* (*Static RAM*), les bits sont stockés par des bascules similaires à des latches : les données stockées se conservent tant que l'ordinateur est sous tension.
- Dans une *DRAM* (*Dynamic RAM*), les bits sont stockés à l'aide de petits condensateurs, dont il faut rafraîchir la charge à intervalles de temps réguliers : les condensateurs utilisés ne conservent leur charge que pendant quelques ms.

Qu'est-ce qui distingue la SRAM et la DRAM ?

A capacité équivalente :

- le temps d'accès à une DRAM est 5 à 10 fois plus long qu'avec une SRAM,
- une SRAM présente un coût beaucoup plus élevé qu'une DRAM.

La SRAM et la DRAM jouent des rôles différents dans la hiérarchie mémoire :

- la SRAM est utilisée pour les niveaux de cache du processeur (\approx Mio),
- la DRAM compose la mémoire centrale (\approx Gio).

Certaines mémoires RAM sont asynchrones, ce qui signifie qu'elles ne sont pas cadencées par une horloge. Mais il existe aussi des DRAM synchrones :

- *SDRAM* (*Synchronous DRAM*) : opérations sur un front d'horloge.
- *DDR SDRAM* (*Double Data Rate SDRAM*) : sur les deux fronts d'horloge.

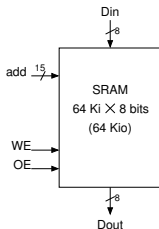
Dans la suite, on va s'intéresser brièvement à la SRAM.

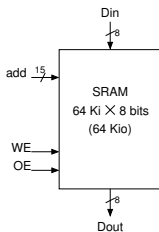
Mémoire SRAM

Une SRAM de 2^m mots de k bits de présente typiquement en entrée :

- m lignes d'adresse formant add ,
- k lignes formant Din pour l'écriture d'un mot de k bits en mémoire,
- WE (*Write Enable*) qui doit être activé pour l'écriture depuis Din ,
- OE (*Output Enable*) qui doit être activé pour la lecture sur $Dout$.

En sortie, on a k lignes formant $Dout$ pour la lecture d'un mot de k bits.

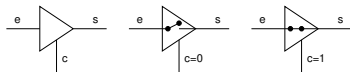




La signification des signaux WE et OE peut se résumer ainsi :

WE	OE	action sur Din	action sur Dout
0	0	Din est ignorée	Dout est déconnectée
1	0	écriture de Din	Dout est déconnectée
0	1	Din est ignorée	lecture sur Dout
1	1	écriture de Din	Dout est déconnectée

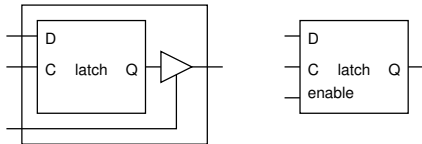
Pour réaliser un bloc de mémoire SRAM, on introduit un composant : le *buffer*.



Quand le signal c est désactivé, le buffer se comporte comme un interrupteur ouvert entre e et s . Quand c est activé, l'interrupteur se ferme.

Le buffer permet de connecter plusieurs bascules par leurs sorties Q à une même ligne en évitant les conflits (chaque bascule maintient un signal sur sa sortie).

Il existe donc des bascules D équipées d'un buffer sur leur sortie Q , commandées par un signal enable, afin d'isoler la sortie Q quand aucune lecture n'est requise.



Petit exemple d'une mémoire 4×2 bits :

