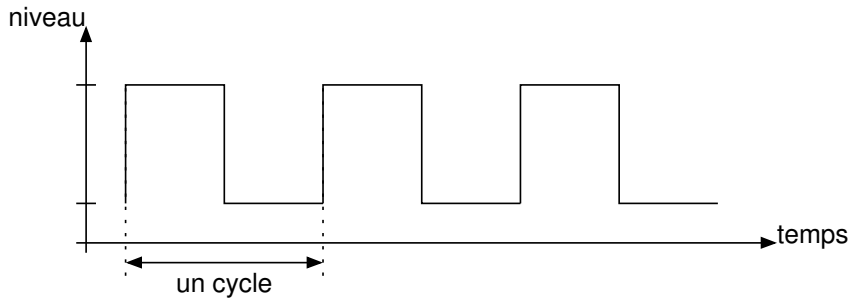
Circuits logiques séquentiels

Introduction

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

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

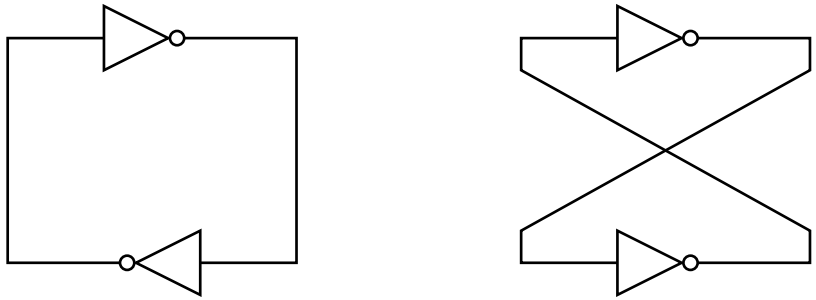
Pour respecter les relations temporelles requises, un circuit séquentiel reçoit un signal d'horloge :



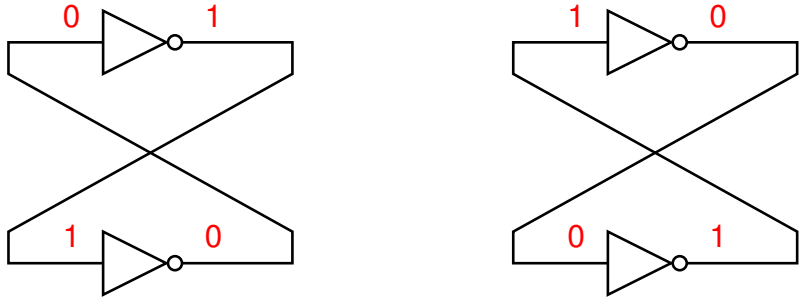
Ce signal permet de synchroniser des événements, soit sur le front montant, soit sur le front descendant.

Plan

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) :

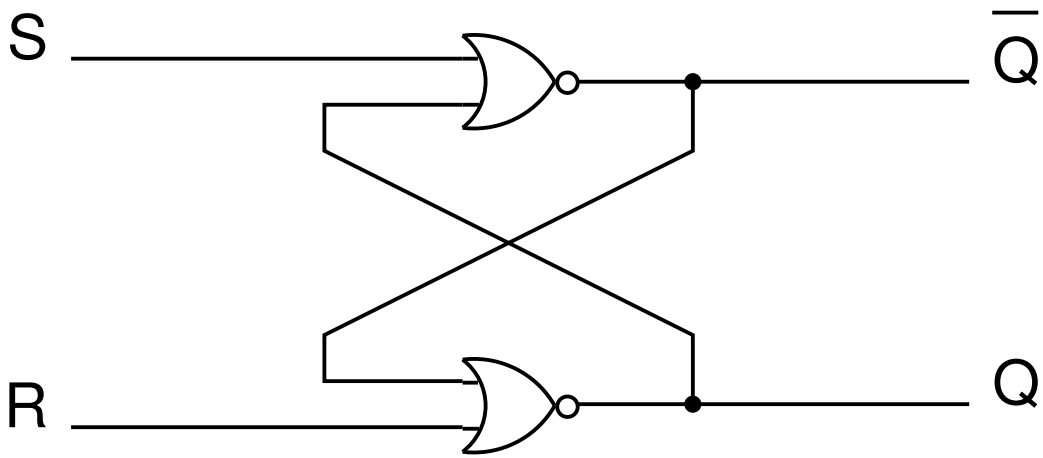


Ce circuit présente deux états stables :



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

Bascule RS



En résumé :

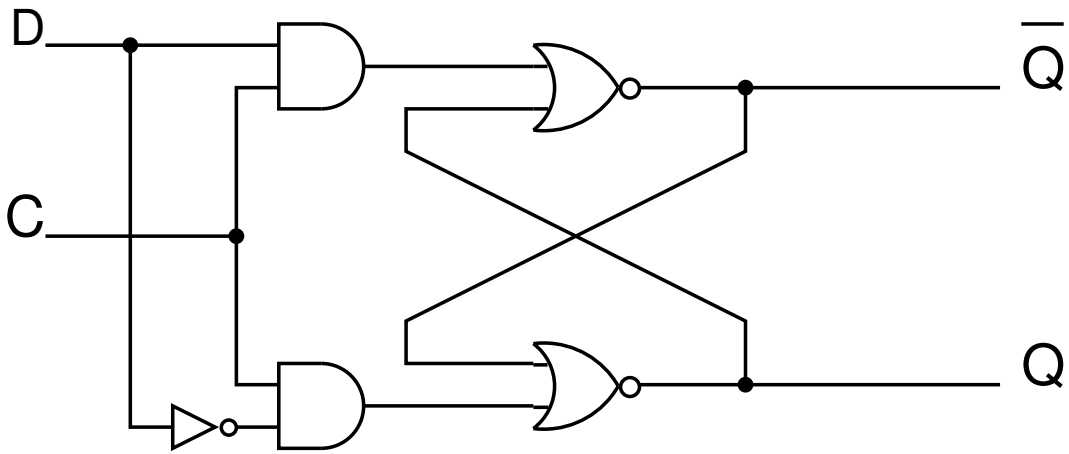
- si S (Set) prend momentanément la valeur 1, la bascule prend l'état Q=1 ;
- si R (Reset) passe momentanément à 1, la bascule prend l'état Q=0 ;

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, il faut éviter d'avoir R = S = ⎍ ...

Verrou (*latch*)

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

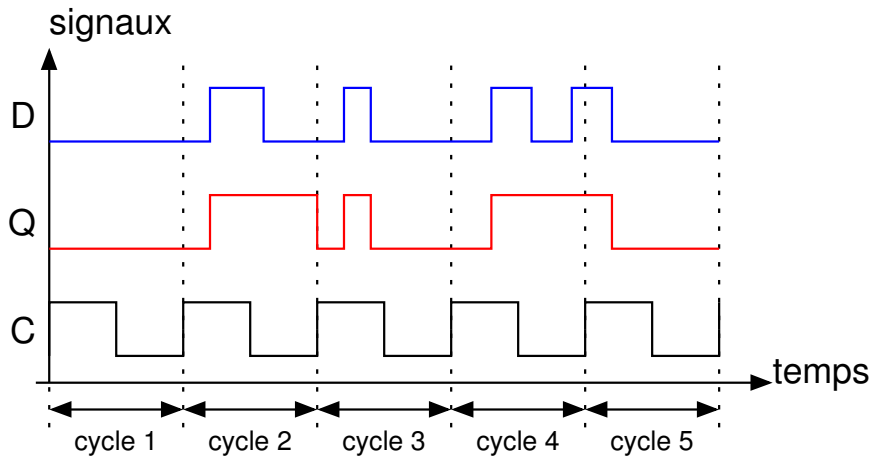


On obtient un *verrou* commandé par le signal C :

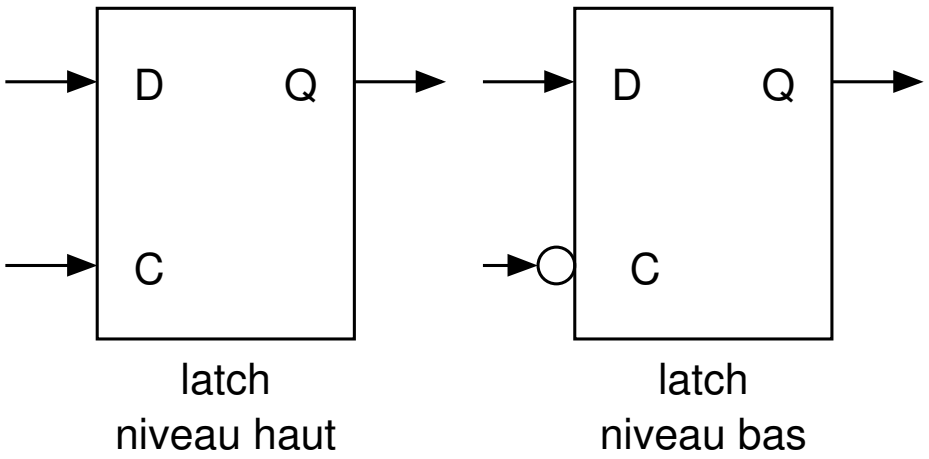
- si D=1 et que la commande C=1, la bascule passe à l'état Q=1,
- si D=0 et que la commande C=1, la bascule passe à l'état Q=0,
- quand C=0, le verrou ne peut plus changer d'état.

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.

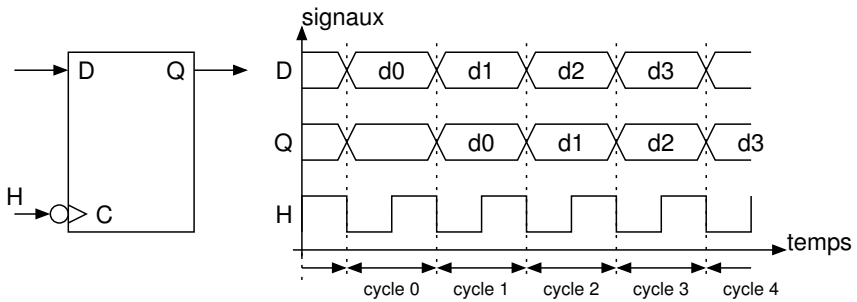


Bascule (*flip-flop*)

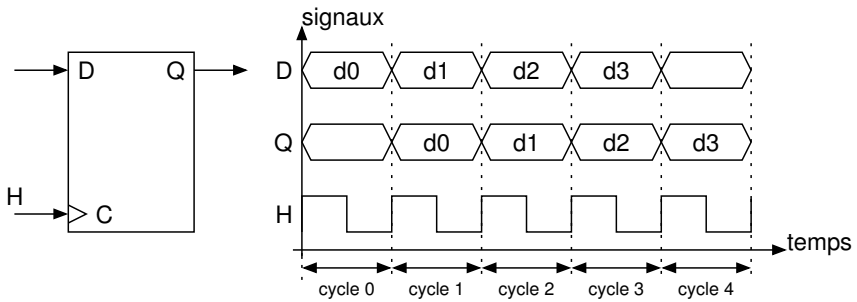
La mémorisation se produit à la fin d'un cycle, et le bit mémorisé est maintenu sur la sortie de la *bascule* pendant tout le cycle suivant.

On peut mettre au point deux types de bascules.

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

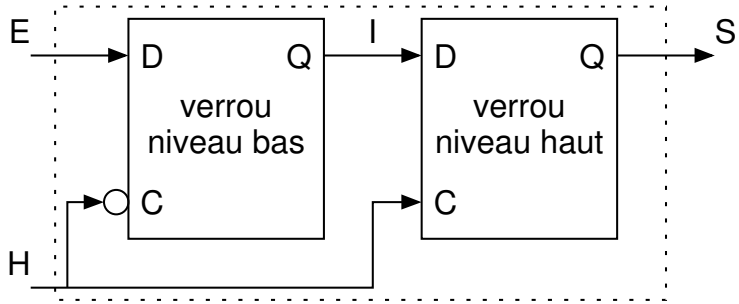


- Bascule régie par le *front montant de l'horloge* :

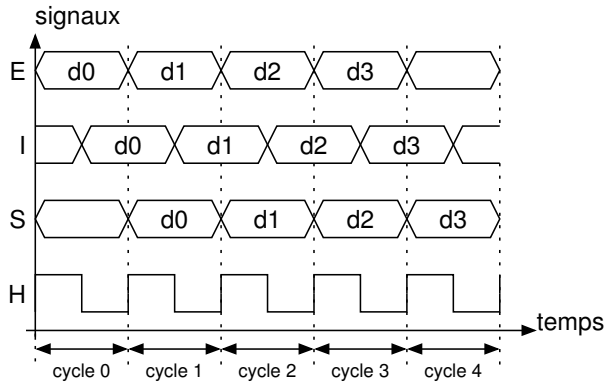


On peut raisonner de la même manière avec les deux types de bascules : il suffit d'aligner correctement la définition des cycles d'horloge.

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



La bascule mémorise l'entrée E sur le *front montant de l'horloge* H :



On travaillera par la suite avec des bascules régies par le front montant.

Rem : Avec en série un verrou régi par le niveau haut, puis un autre régi par le niveau bas, on a une bascule mémorisant sur le front descendant de l'horloge.

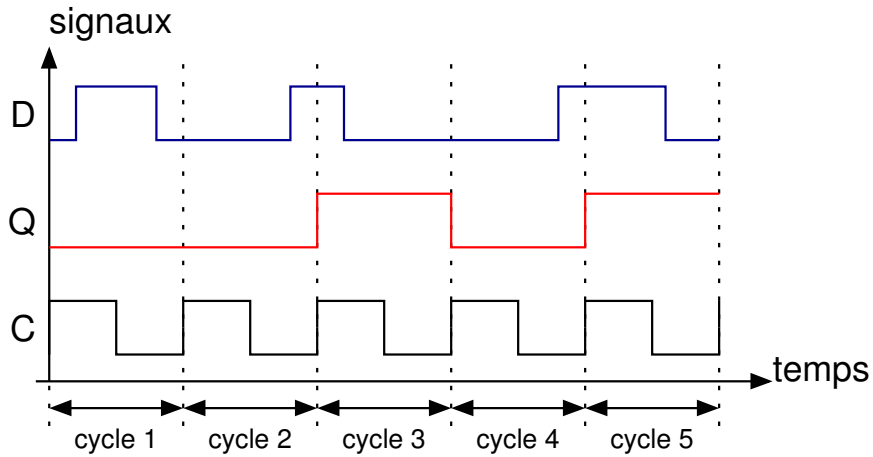
Registres

Nos registres sont fabriqués à partir de bascules régies par un front d'horloge.

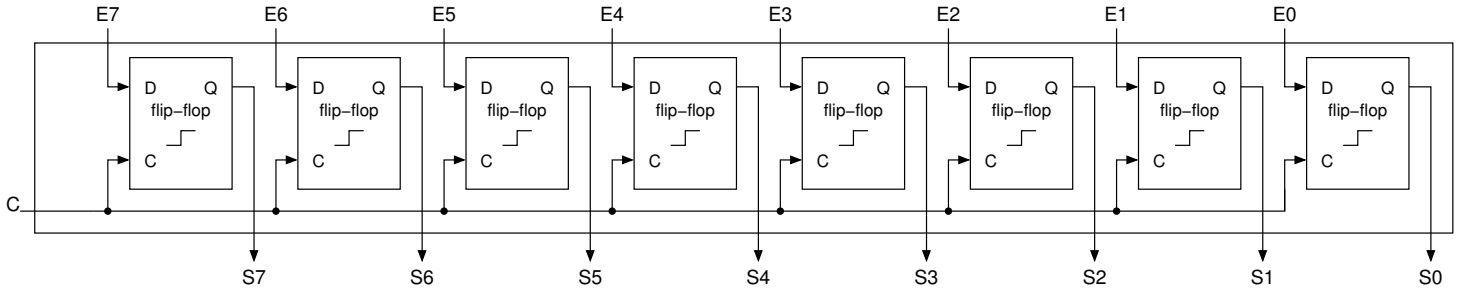
Cette convention fixe la manière d'utiliser les bascules :

- 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.

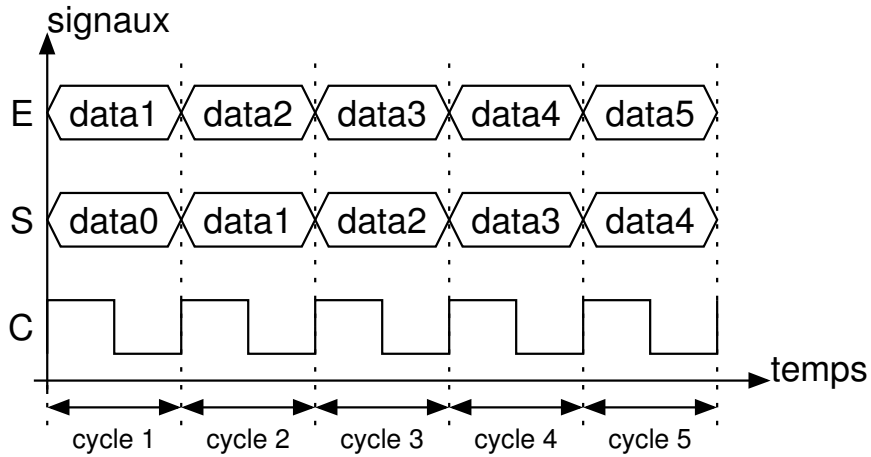
Le résultat calculé au cycle i est typiquement la sortie d'un circuit combinatoire.



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 :

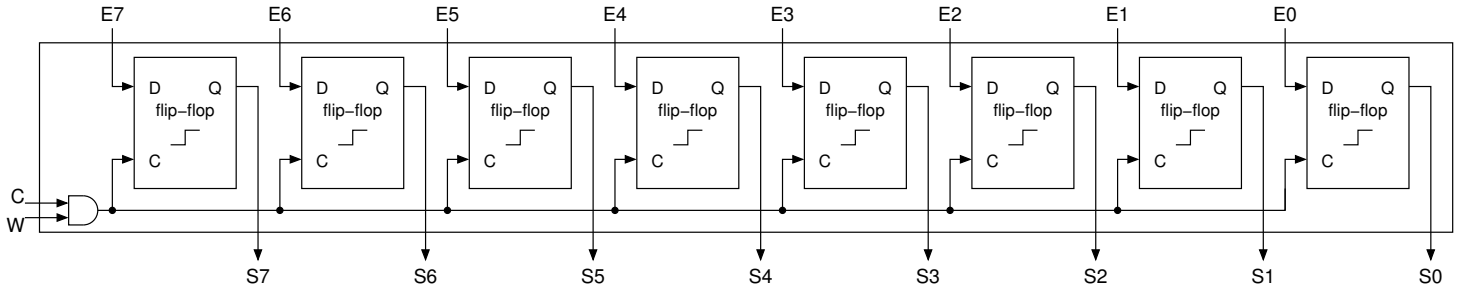


On obtient le type de chronogramme suivant :

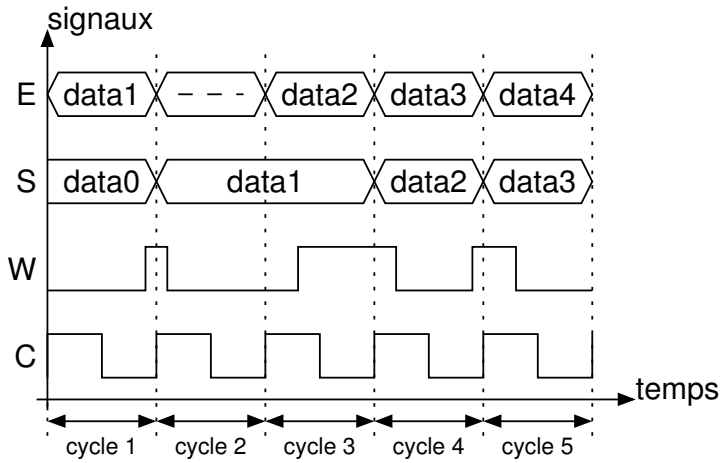


A chaque fin de cycle, l'entrée E est stockée et est disponible au cycle suivant.

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.

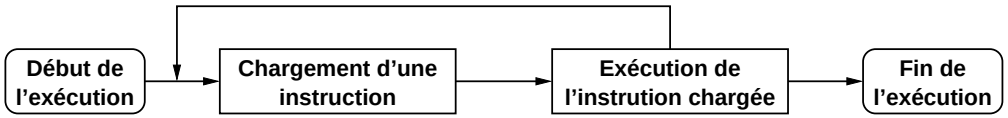
Plan

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...
- Un processeur passe par les étapes du cycle d'instruction.



Un *automate fini* est une abstraction du *comportement d'un circuit séquentiel* : c'est le pendant d'une fonction booléenne pour un circuit combinatoire.

Cette abstraction est *a priori* asynchrone (l'automate réagit à des événements), mais on s'intéressera 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 $S \times E \to S$;
- $s_0 \in S$ est un état spécial appelé *état initial*.

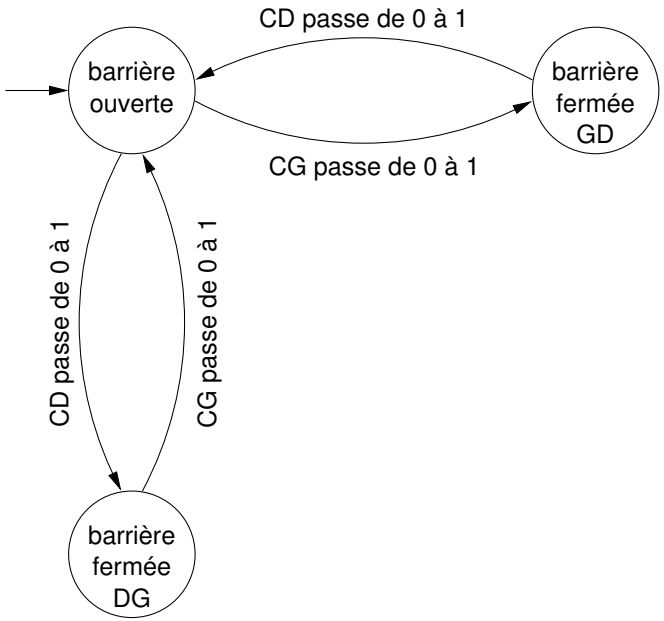
Graphiquement, on fait des dessins avec

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

On suppose que les trains peuvent se croiser au passage à niveaux :

- 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.

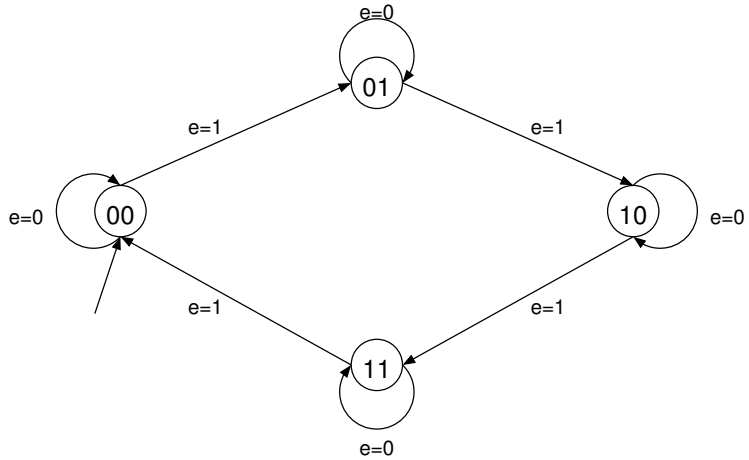
Un *automate fini synchrone* est un quadruplet (S, I, T, s_0) où

- S est un ensemble fini d'états (*States*) ;
- I est un ensemble fini d'entrées (*Inputs*) ;
- T est une *fonction de transition* de $S \times I \to S$;
- $s_0 \in S$ 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 : on construit un compteur 2 bits modulo 4, qui s'incrémente 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.

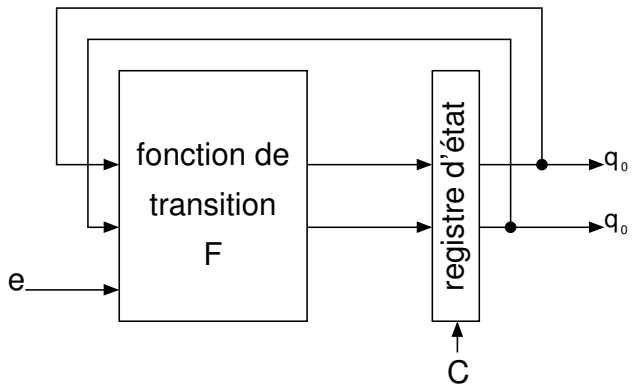


q_1	q_0	e	$F_1(q_1, q_0, e)$	$F_0(q_1, q_0, e)$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

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 $F(q_1, q_0, e)$.

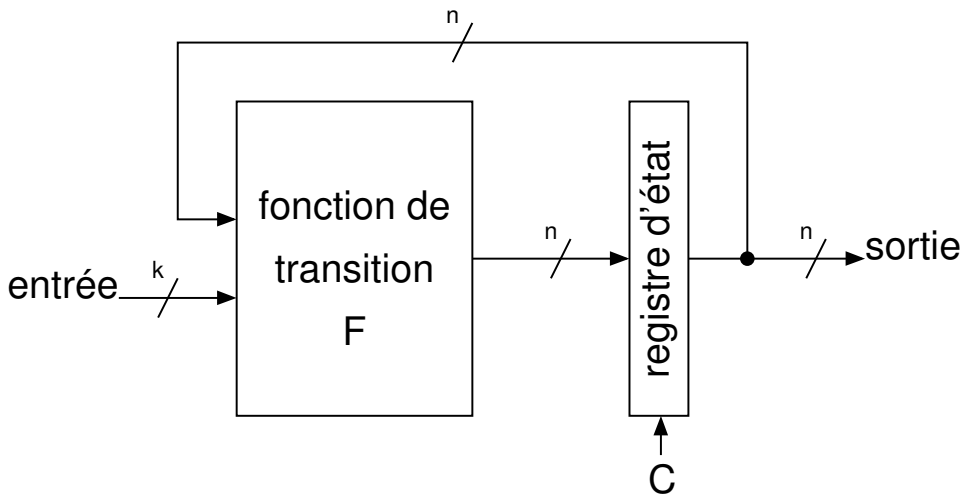
On pourra donc implanter facilement l'automate demandé :



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 combinatoire, qui prend en entrée :
  - reçoit les n bits du registre d'état, et k signaux codant l'entrée courante ;
  - ressort l'état suivant, qui sera stocké par le registre d'état en fin de cycle.

Un tel automate peut être implanté par le circuit suivant :



Si on est capable de décrire formellement un automate fini synchrone, on arrivera toujours à déduire de cette description un circuit séquentiel pour le réaliser.

Il existe d'autres types d'automates que ceux décrits jusqu'à présent : dans un *automate de Moore*, la sortie n'est pas simplement l'état de courant de l'automate, mais une fonction (combinatoire) de cet état courant.

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

Banc de registres

Un *banc de registre* 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.

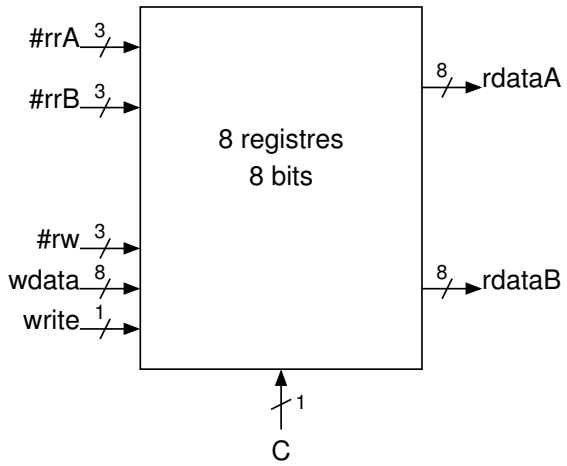
Le banc de registre maintient en permanence sur son port de lecture `rdataX` la donnée contenue dans le registre dont l'indice est spécifié sur l'entrée `#rrX`.

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`.

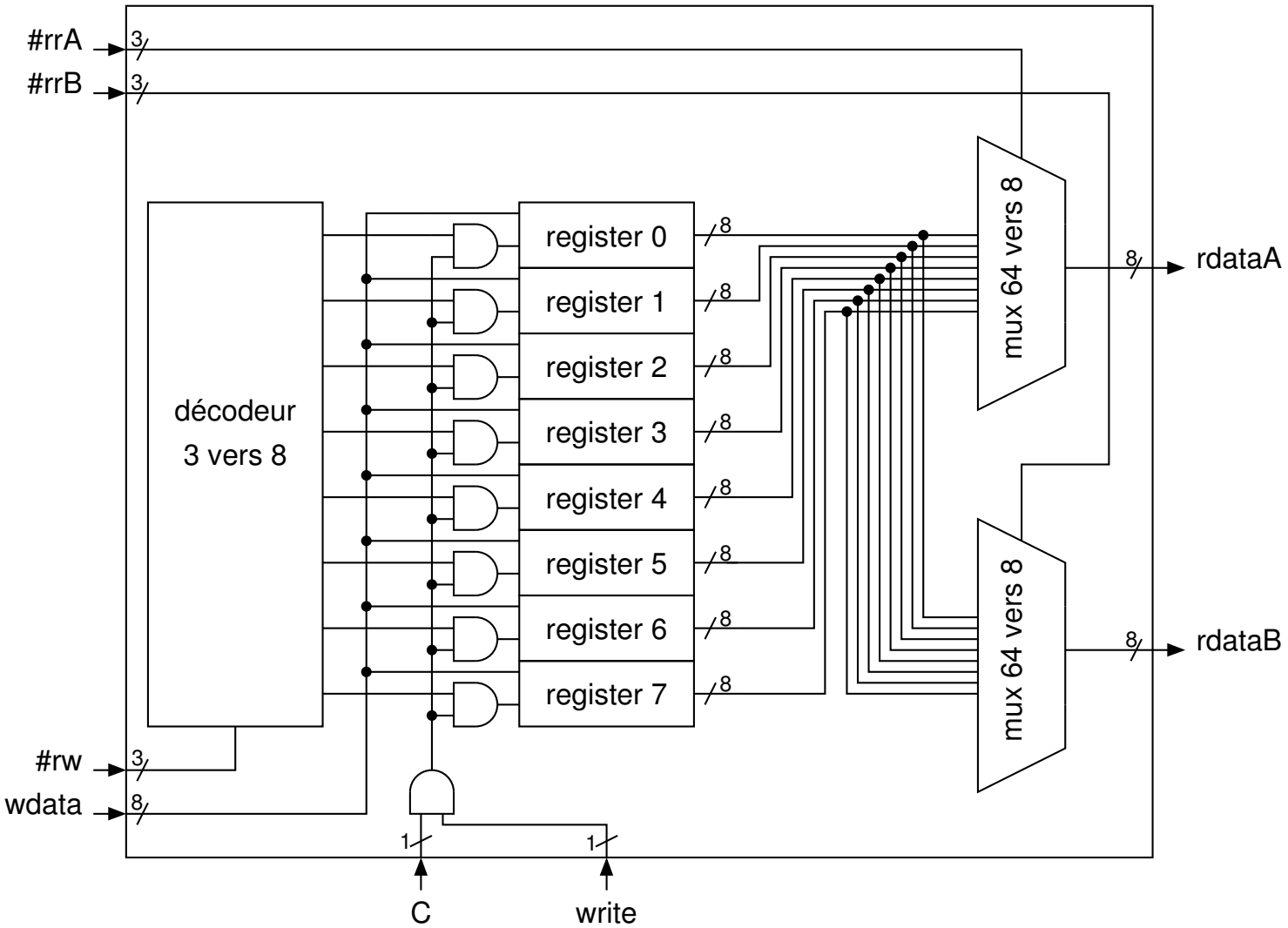
Si sur un front montant de l'horloge `C` le signal `write` est activé, la donnée présente sur le port d'écriture `wdata` est placée dans le registre `#rw`.

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`.


#rrA
#rrB
3
3
décodeur
3 vers 8
register 0
register 1
register 2
register 3
register 4
register 5
register 6
register 7
8
mux 64 vers 8
mux 64 vers 8
rdataA
rdataB
#rw
wdata
C
write
1

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 verrous : 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 condensateurs, dont il faut rafraîchir la charge à intervalles de temps réguliers : ils 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 (≈Mio),
- la DRAM compose la mémoire centrale (≈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 :

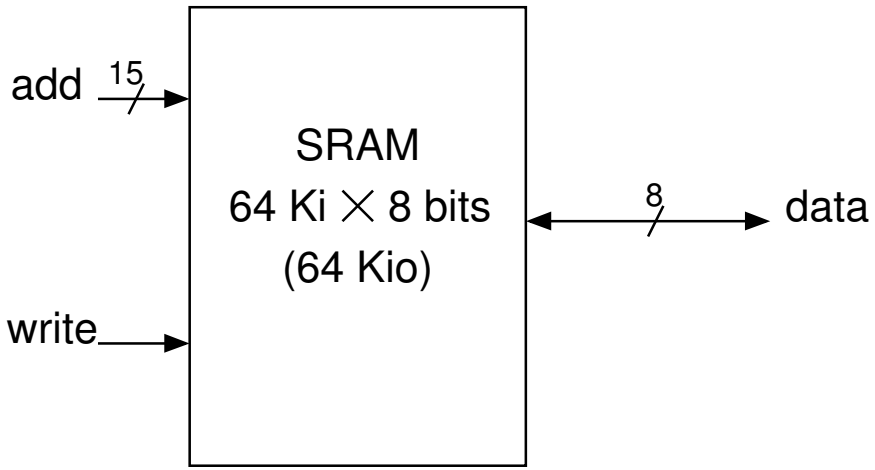
- *SDRAM* (*Synchronous DRAM*) : opérations sur un front d'horloge.
- *DDR SDRAM* (*Double Data Rate SDRAM*) : sur les deux fronts d'horloge.

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 `data` pour l'écriture ou la lecture d'un mot de k bits en mémoire,
- un signal `write` qui doit être activé pour l'écriture depuis `data`,
- si `write` est à 0, alors on est en lecture, et la donnée est lue sur `data`.



Conclusion

Dans le cours précédent, nous avons fait des rappels sur les *circuits logiques combinatoires*, dont les sorties ne dépendent que des entrées. Ces circuits permettent d'effectuer les calculs pour les instructions arithmétiques ou logiques.

Nous avons présenté les *circuits séquentiels*, dans lesquels les sorties dépendent de l'état courant et des entrées. Ces circuits permettent de mémoriser des données, et de gérer l'exécution des instructions dans un processeur.

Les deux types de circuits seront utilisés dans les TP consacrés à la mise au point d'un processeur implantant l'architecture LC3.