Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

Chapitre 4
Circuits séquentiels

Yann Thorimbert

Chapitres du cours

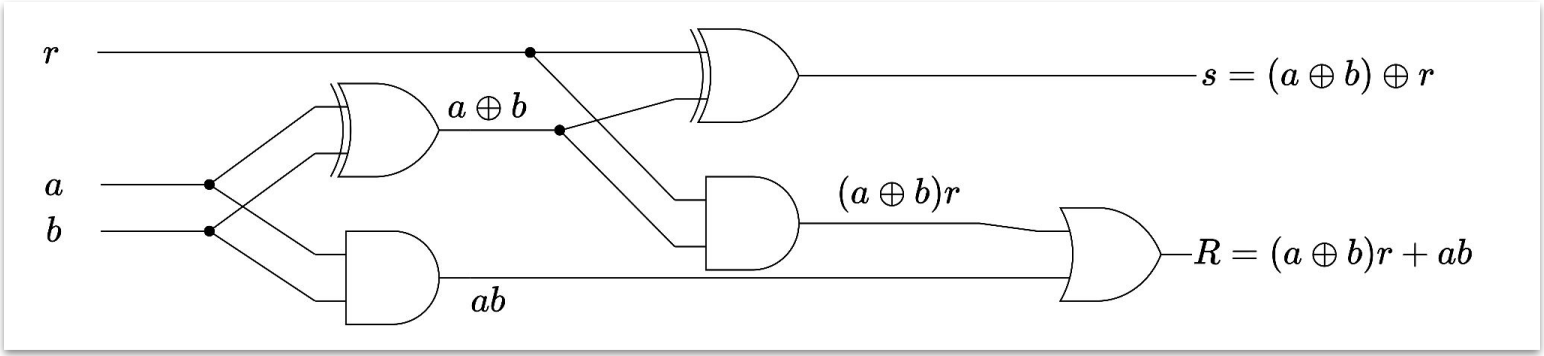
1. Origines des ordinateurs et des réseaux informatiques
2. Codage des nombres
3. Codage des médias
4. **Circuits logiques ← (Partie 2)**
5. Architecture des ordinateurs
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Rappel sur les circuits logiques

- Nous avons vu comment des circuits électroniques pouvaient simuler des portes logiques, et donc une table de vérité.

- Nous avons vu l'exemple des circuits nommés multiplexeur, additionneur, soustracteur.

Circuit additionneur



Mémorisation de valeur | **Que cherche-t-on à faire ?**

- Toutes sortes de données doivent être mémorisées par la machine afin de mener à bien l'exécution d'un algorithme.

- Exemple de code Python pour connaître la valeur de n! (factorielle).

```
n = 6 #pour l'exemple
res = 1
while n > 1:
    res = res * n
    n = n - 1
print(res) #affiche 720 = 6*5*4*2*1
```

Mémorisation de valeur | **Que cherche-t-on à faire ?**

Exemple de code pour connaitre la valeur de n!

```
n = 6 #pour l'exemple
res = 1
while n > 1:
    res = res * n
    n = n - 1
print(res) #affiche 720
```

Itération	**n**	**res**
1	6	1
2	5	6
3	4	30
4	3	120
5	2	360
6	1	720

Mémorisation de valeur | **Qu’est-ce que mémoriser ?**

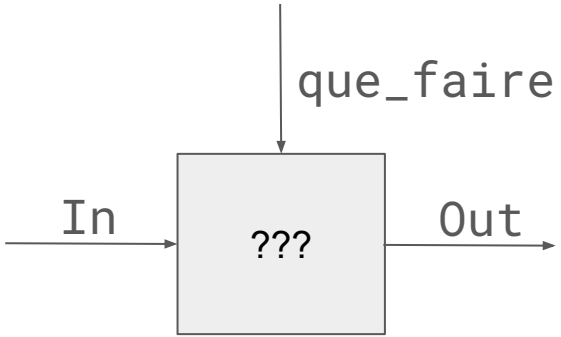
Plus précisément, on veut que :

- Si l’on décide que la valeur d’un bit est 1, ce dernier prenne la valeur 1 (peu importe sa valeur précédente).
- Si l’on décide que la valeur d’un bit est 0, ce dernier prenne la valeur 0 (peu importe sa valeur précédente).
- Si l’on ne décide rien, la valeur du bit est conservée.

Mémorisation de valeur | **Qu’est-ce que mémoriser ?**

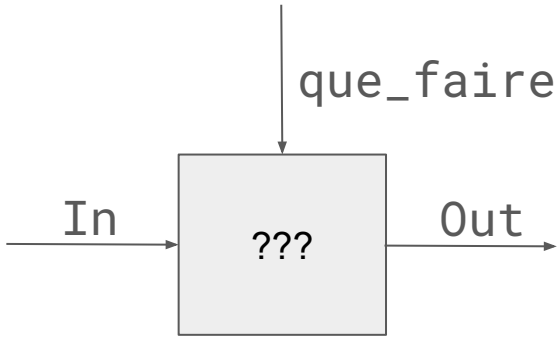
Résumé sous forme de code :

```
if que_faire == "Set":
    Out = 1
elif que_faire == "Reset":
    Out  = 0
else:
    Out = In
```



Mémorisation de valeur | **Qu’est-ce que mémoriser ?**

Ne reste plus qu’à trouver le circuit logique.

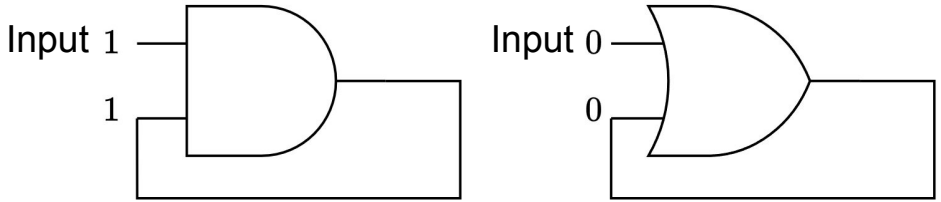


On peut déjà le nommer : **bascule set-reset**.

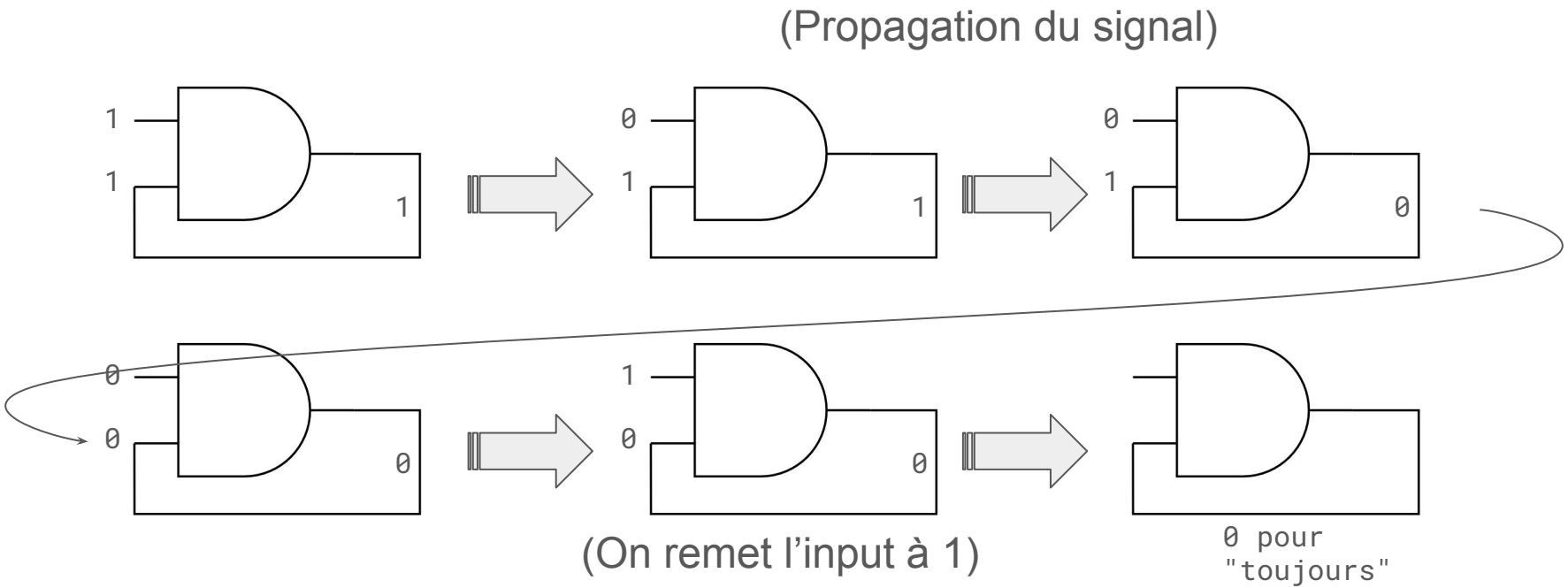
Circuit retenant 1 bit

- Idée : il faut **garder** une valeur. Utilisons donc un circuit qui se "réinjecte" sa propre valeur !

- Dans le cas de AND et de OR, que se passe-t-il si l'on change la valeur d'input ?

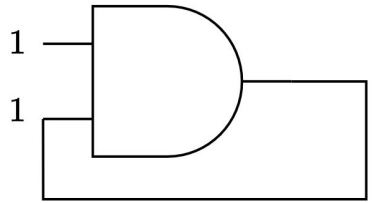


Circuit retenant 1 bit | **Évolution temporelle**



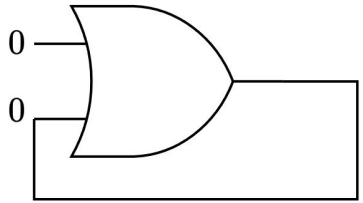
Circuit retenant 1 bit

Ce circuit ne peut basculer qu'une fois et ne retenir que le `0`.



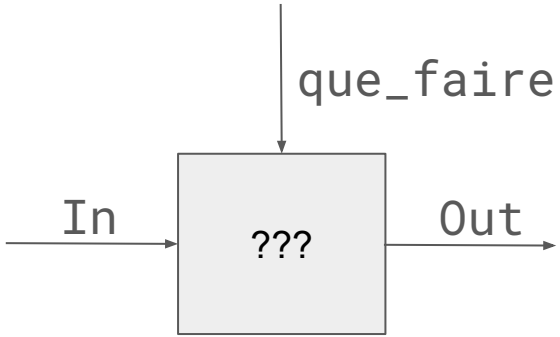
Circuit retenant 1 bit

Ce circuit ne peut basculer qu'une fois et ne retenir que le 1.



Retour au problème initial

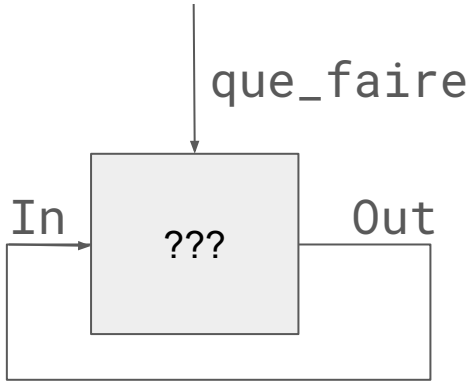
```
if que_faire == "Set":
    Out = 1
elif que_faire == "Reset":
    Out  = 0
else:
    Out = In
```



⇒ Écrivons la **table de vérité** !

Retour au problème initial

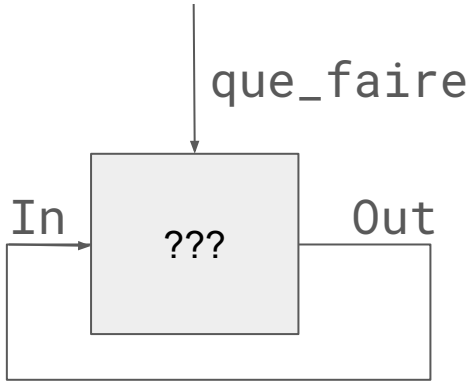
```
if que_faire == "Set":
    Out = 1
elif que_faire == "Reset":
    Out  = 0
else:
    Out = In
```



⇒ Écrivons la **table de vérité** !

Retour au problème initial

```
if que_faire == "Set": #Cas1
    Out = 1
elif que_faire == "Reset": #Cas2
    Out  = 0
else: #Cas3
    Out = In
```



⇒ Écrivons la **table de vérité** !

`que_faire` peut prendre 3 valeurs.

Doit être écrit sur 2 bits. Nous les nommerons S et R.

Bascule Set-Reset | **Table de vérité**

S	R	In	Out

Bascule Set-Reset | **Table de vérité**

Mémorisation

S	R	In	Out
0	0	0	0
0	0	1	1

Bascule Set-Reset | **Table de vérité**

Reset

S	R	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0

Bascule Set-Reset | **Table de vérité**

S	R	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

Set

Bascule Set-Reset | **Table de vérité**

S	R	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	?

Indéfini

Bascule Set-Reset | **Expression logique**

`Out =!S!RIn + S!R!In + S!RIn` (forme canonique)

S	R	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

Bascule Set-Reset | **Expression logique**

```
Out =!S!RIn + S!R!In + S!RIn      (forme canonique)
    =!R(!SIn + S!In + SIn)
    =!R(S + In)
```

S	R	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

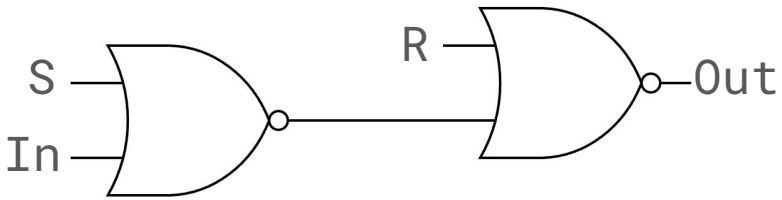
Bascule Set-Reset | **Expression logique**

```
Out =!S!RIn + S!R!In + S!RIn      (forme canonique)
    =!R(!SIn + S!In + SIn)
    =!R(S + In)
    =!R !!(S + In)
    =!(R + !(S + In))
    = NOR(R, NOR(S, In))
```

S	R	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

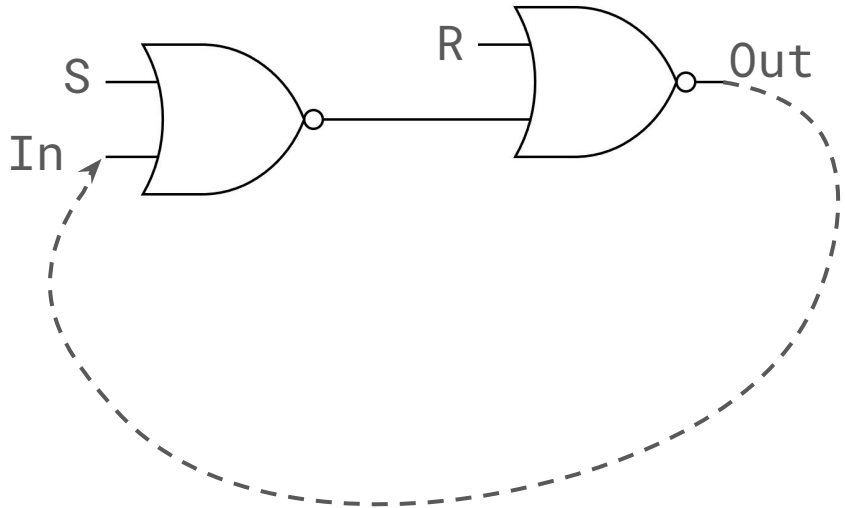
Bascule Set-Reset | **Circuit logique**

```
Out =!S!RIn + S!R!In + S!RIn      (forme canonique)
    =!R(!SIn + S!In + SIn)
    =!R(S + In)
    =!R !!(S + In)
    =!(R + !(S + In))
    = NOR(R, NOR(S, In))
```



Bascule Set-Reset | **Circuit logique**

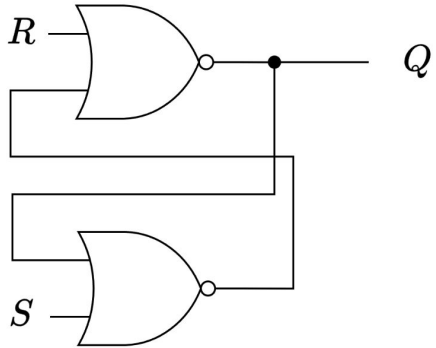
N’oublions pas que $Out_t = In_{t+1}$

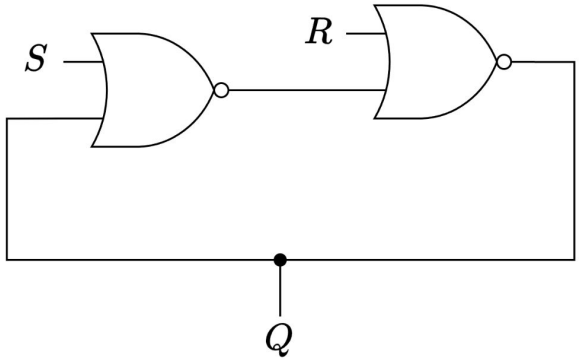


Bascule Set-Reset | **Circuit logique**

N’oublions pas que $\texttt{Out}_t = \texttt{In}_{t+1}$

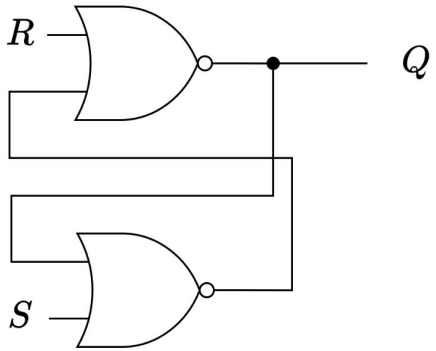
Deux représentations équivalentes :





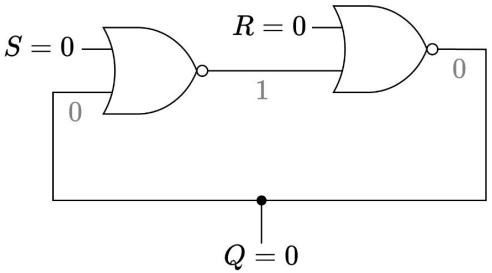
Bascule set-reset

Aussi nommé "verrou" (*latch*), car **capture** un état.

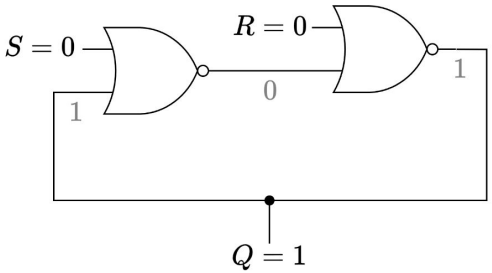


Bascule Set-Reset | **Les quatre cas valides**

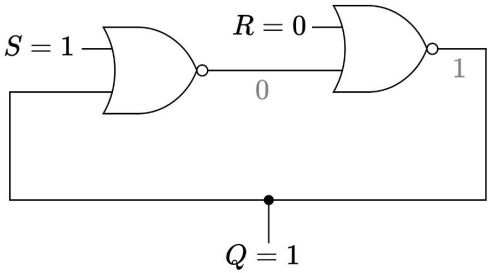
Memorisation de la valeur 0



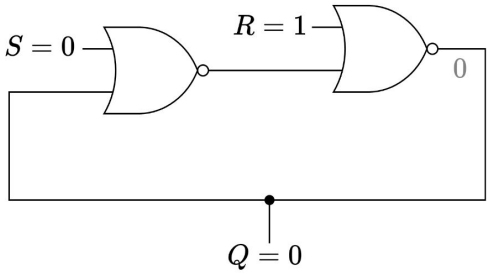
Memorisation de la valeur 1



Set



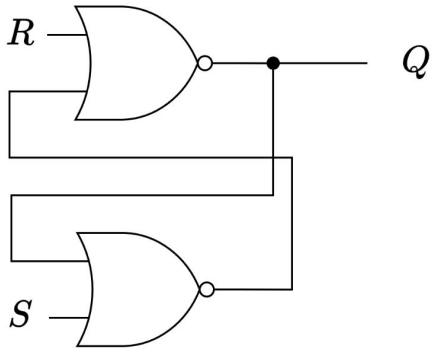
Reset



Historique des valeurs

R = 0 et S = 0 ⇒ Q = ?

La réponse dépend de l'état au temps t précédent.



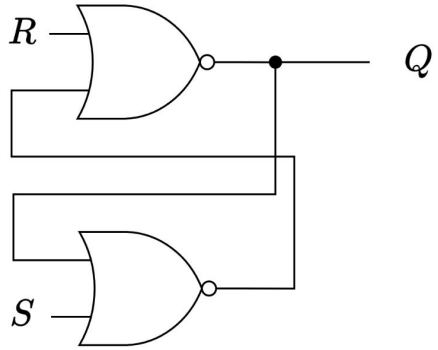
Historique des valeurs

La réponse dépend de l'état au temps t précédent.

Si $R_{t-1} = 0$ et $S_{t-1} = 1$, alors $Q_t = 1$.

Si $R_{t-1} = 1$ et $S_{t-1} = 0$, alors $Q_t = 0$.

Circuit dont l'output dépend de la **séquence** des inputs : "circuit séquentiel".

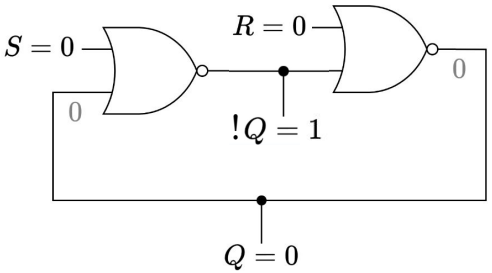


Le problème des valeurs interdites

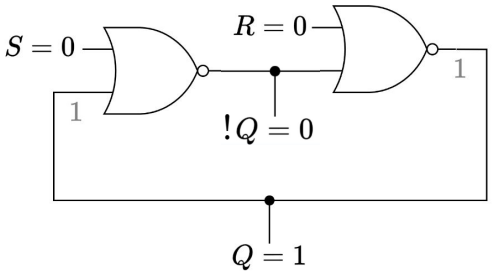
- R = S = 1 ne signifient rien d'un point de vue logique.
- D'un point de vue électronique :
  - Premièrement, quand on revient à R = S = 0, l'état dépend du temps de propagation du signal (tout à fait indésirable, peu prédictible) ; est-ce R = 0 ou S = 0 qui atteint en premier la porte concernée ?
  - Deuxièmement, Q et son complément ne sont plus cohérents (cf. diapo suivante).

Bascule Set-Reset | **Les quatre cas valides**

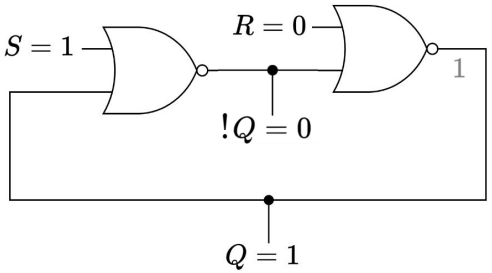
Memorisation de la valeur 0



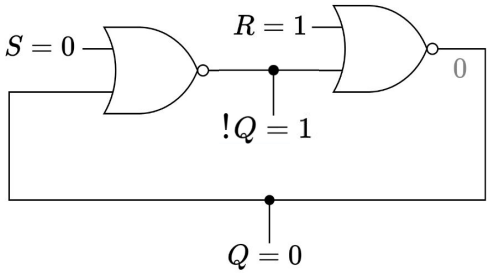
Memorisation de la valeur 1



Set



Reset



Retour au problème de propagation du signal

- Circuits physiques : le signal ne s'établit pas instantanément partout dans le circuit.
- Phénomène d'autant plus important que le signal doit traverser beaucoup de portes.
- Comment laisser un certain temps d'attente aux différents circuits de la machine ?

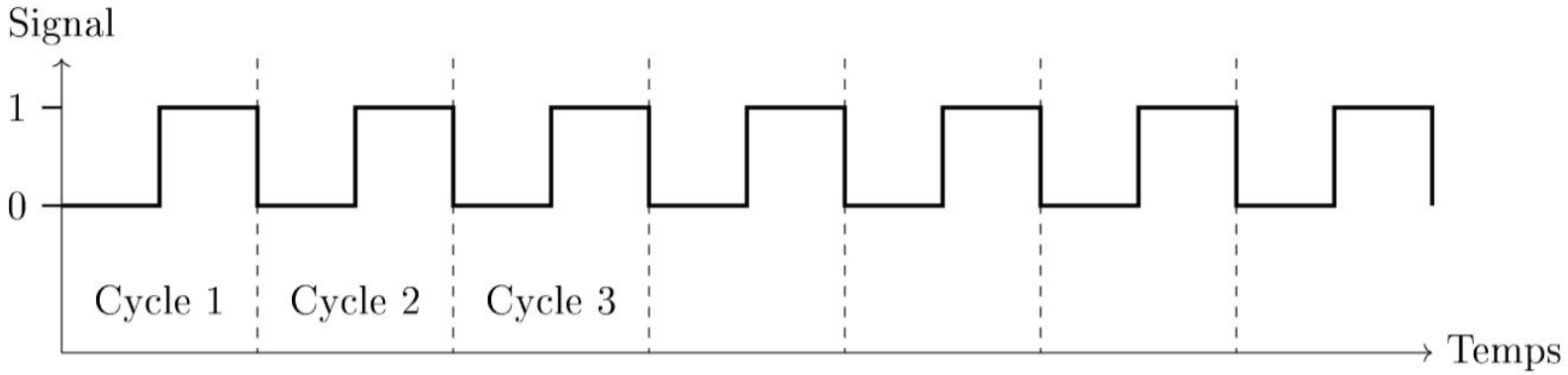
Cycles d’horloge

- Comment laisser un certain temps d’attente aux différents circuits de la machine ?
- Solution possible : battre la mesure avec une **horloge**.
- Le temps de stabilisation de tous les circuits doit être inférieur à la durée entre deux **pulsations** de l’horloge.
- C’est ce qui détermine la durée du **cycle d’horloge**. Chaque cycle comprend une période de signal bas et une période de signal haut.

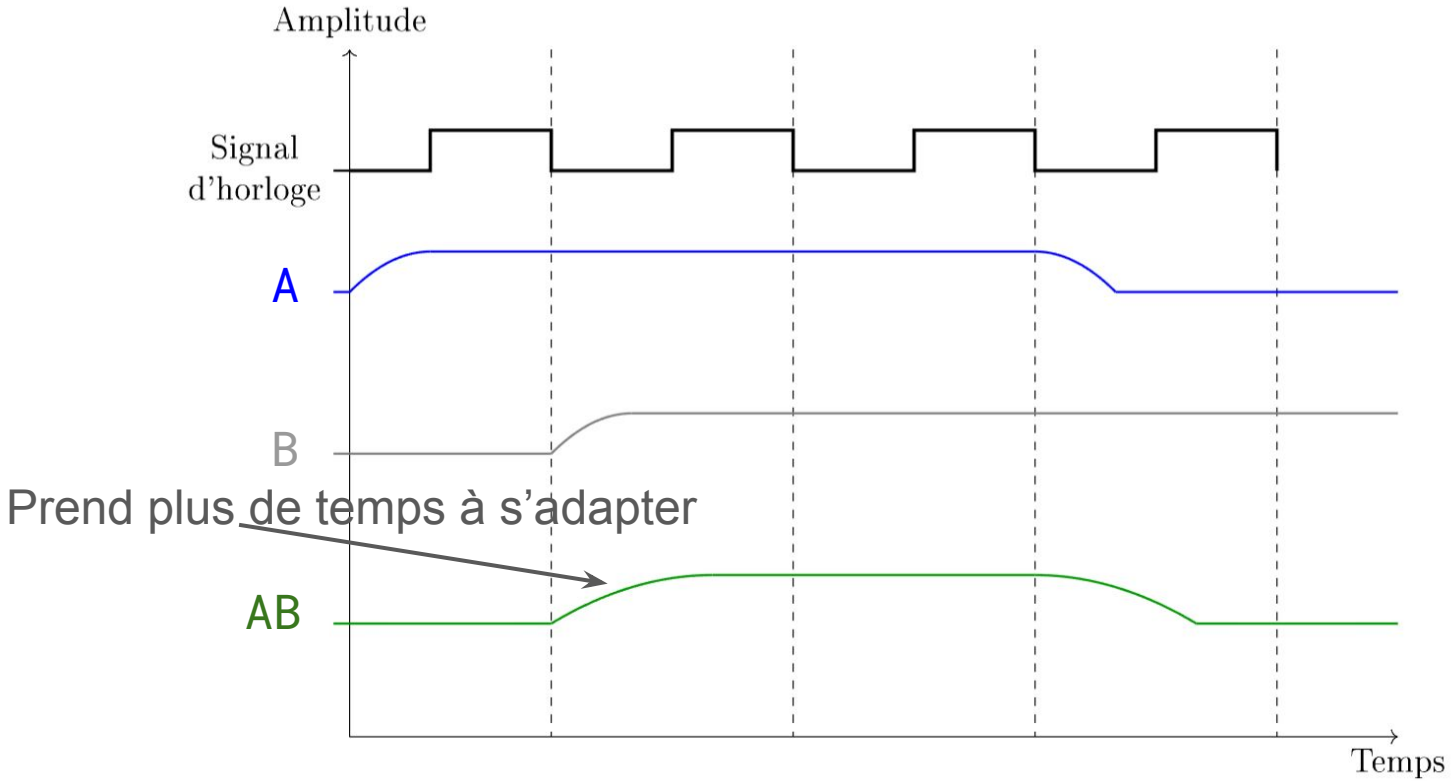
Cycles d’horloge

- Les éléments synchronisés sur l’horloge se nomment les **éléments d’état**.

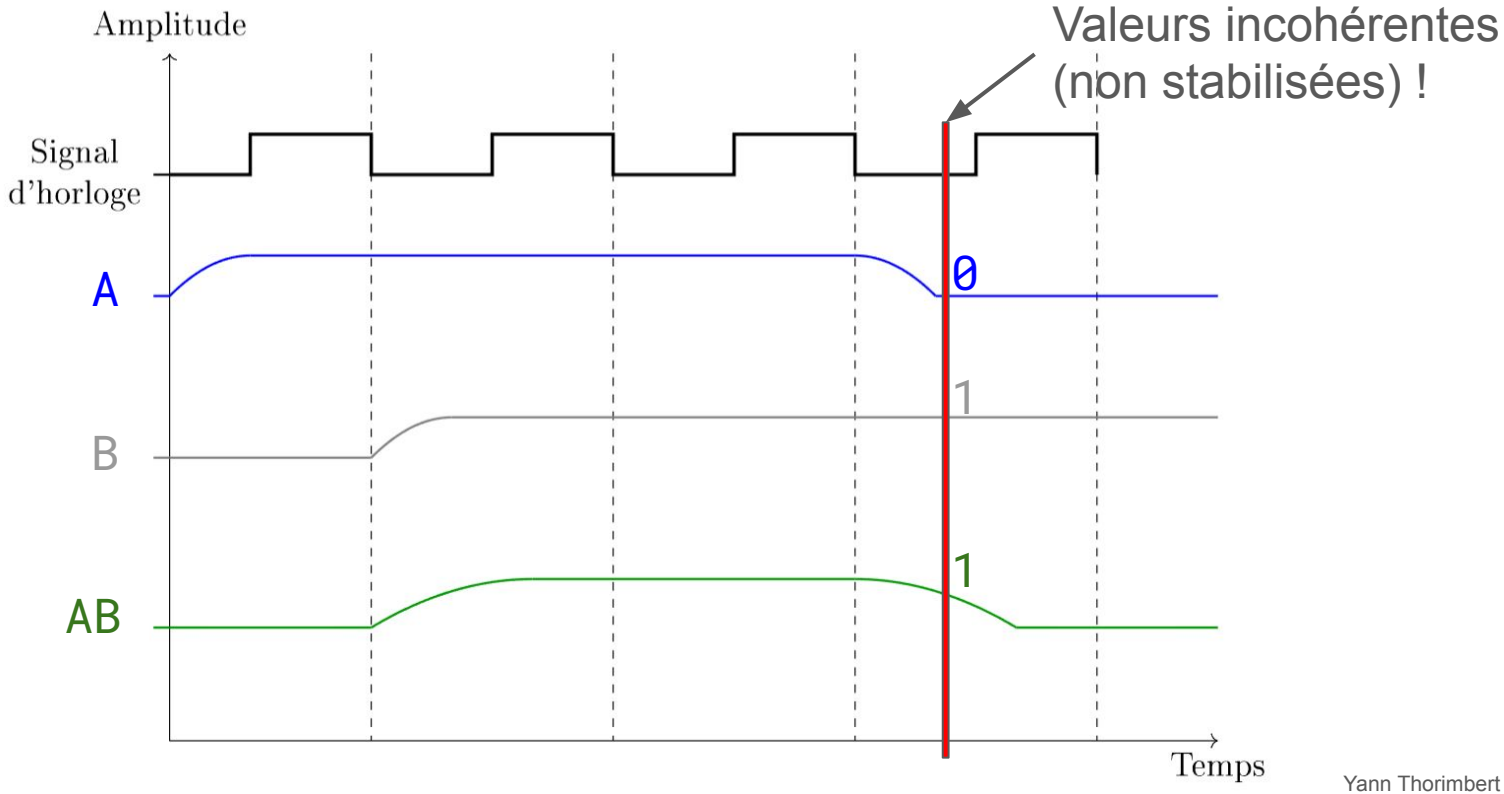
- L’état du système ne peut changer qu’en certains instants précis.
  Exemple :



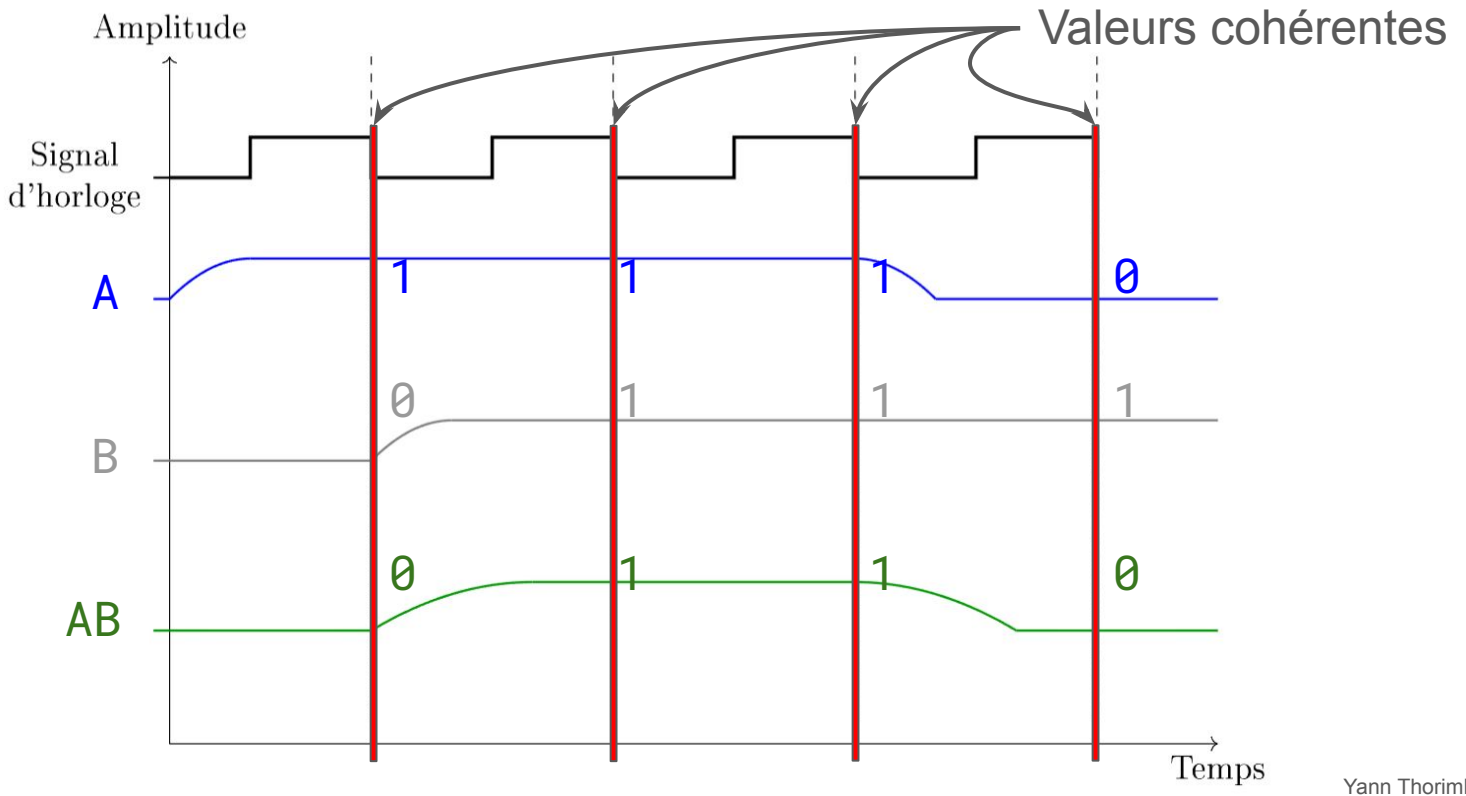
Cycles d’horloge



Cycles d'horloge



Cycles d'horloge



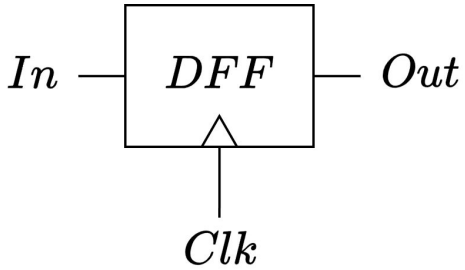
Problème de la bascule set-reset

- L'output de la bascule ne dépend pas de l'état (horloge). La bascule est **asynchrone**.

- On veut une bascule **synchrone**, afin de pouvoir garantir que son état change conformément aux pulsations de l'horloge.

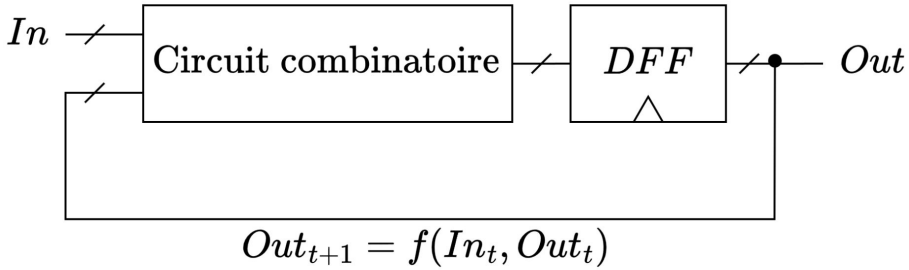
Bascule synchrone

- Une bascule synchrone "bloque" la valeur tant que le signal d'horloge n'est pas reçu.
- En anglais : **delay flip-flop** (DFF).
- Idée :
  - Une ligne reçoit la donnée à mémoriser In.
  - Une ligne reçoit le signal d'horloge Clk.
  - Le signal de sortie Out n'est mis à jour qu'au signal d'horloge montant.
  - Tout le reste du temps, entretient sa valeur à mémoriser.

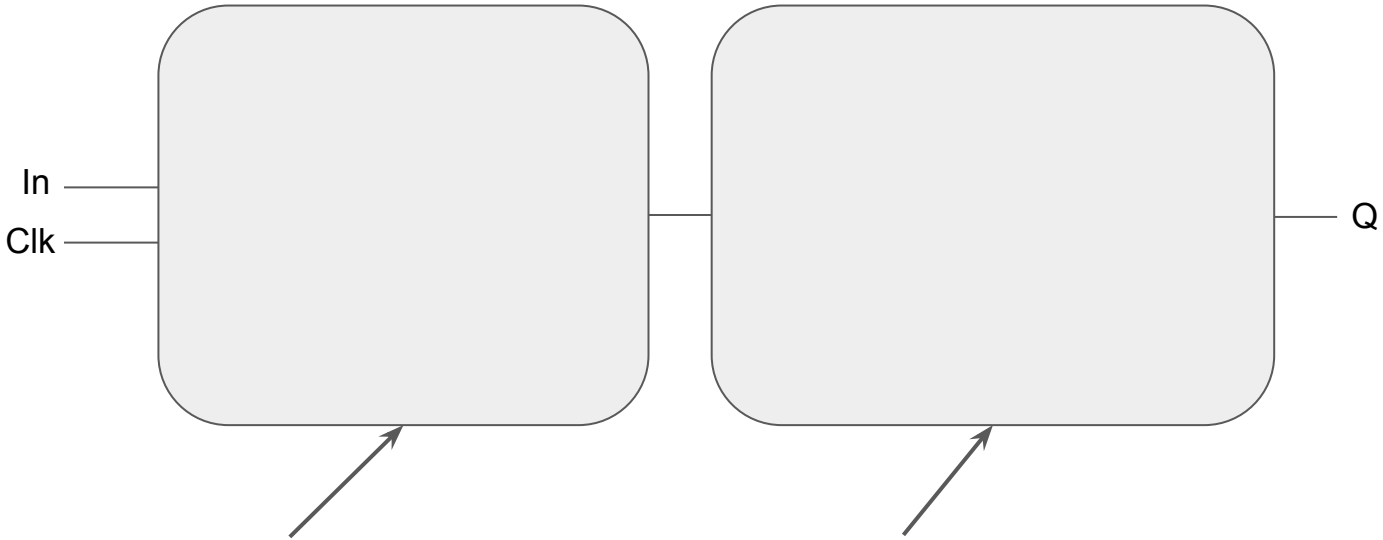


Delay Flip-Flop

Permet de synchroniser un circuit logique combinatoire sur l'horloge.

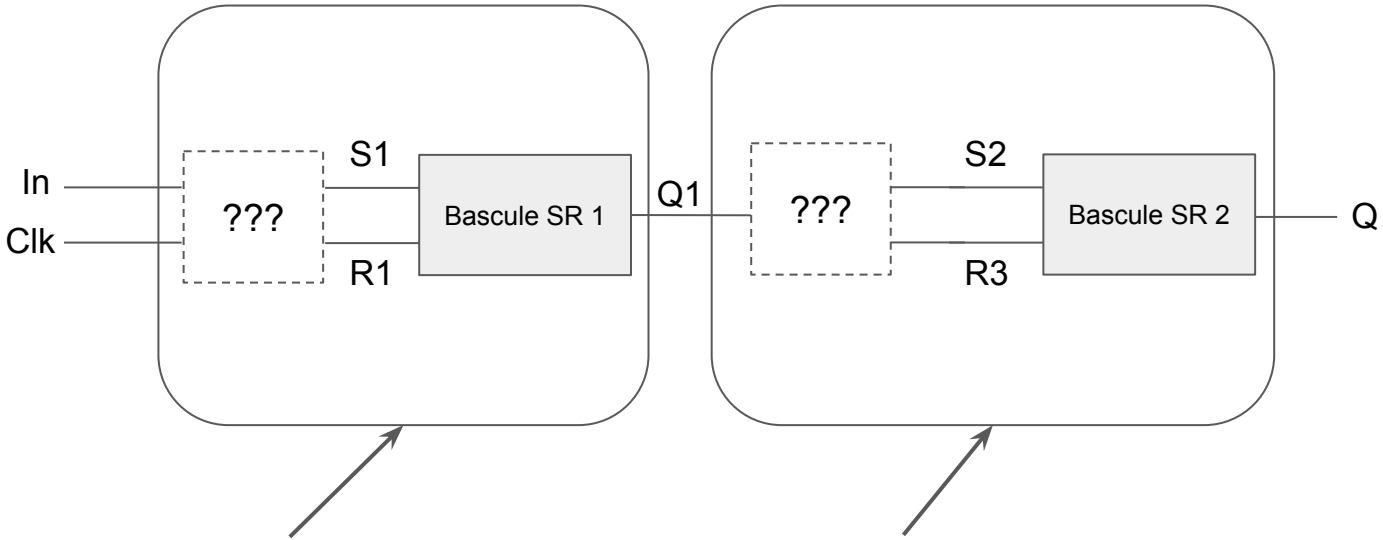


Delay flip-flop | **Idée du circuit**



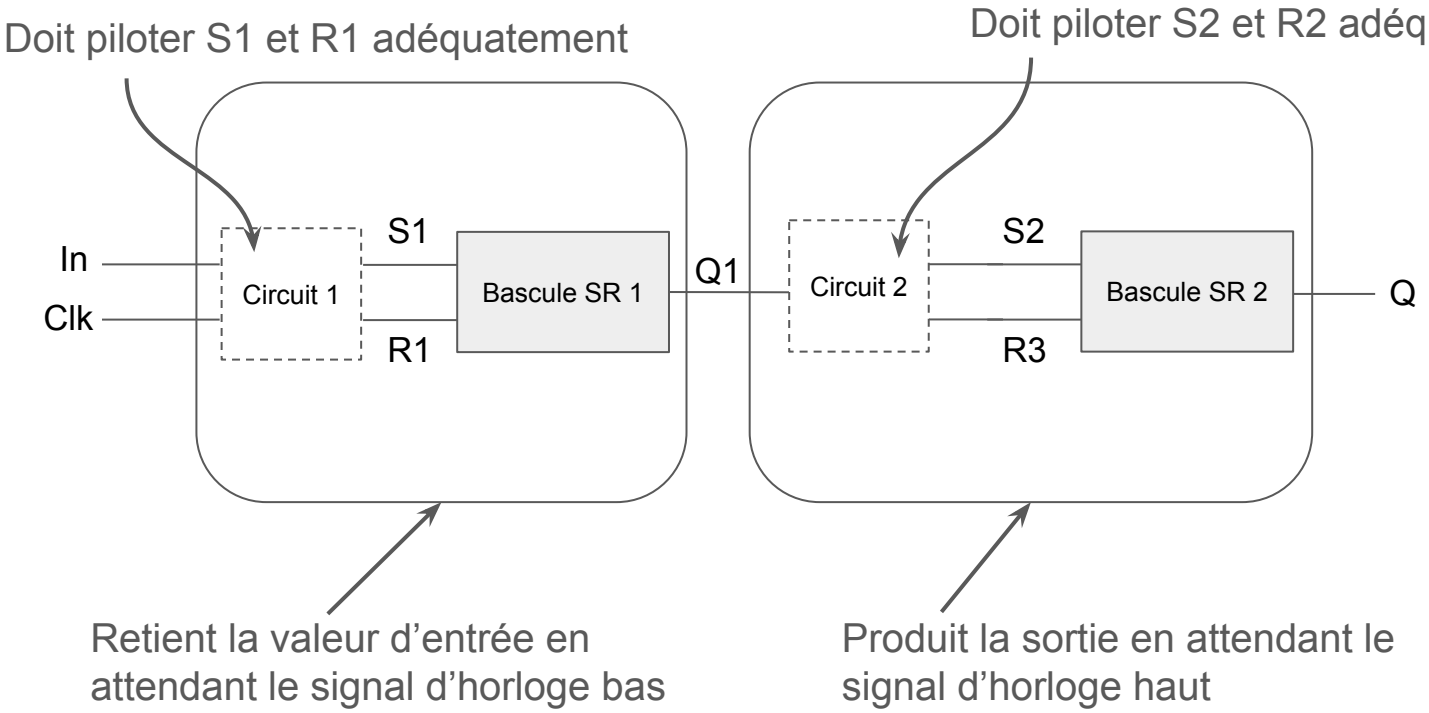
(Diapositive hors champ)

Delay flip-flop | **Idée du circuit**



(Diapositive hors champ)

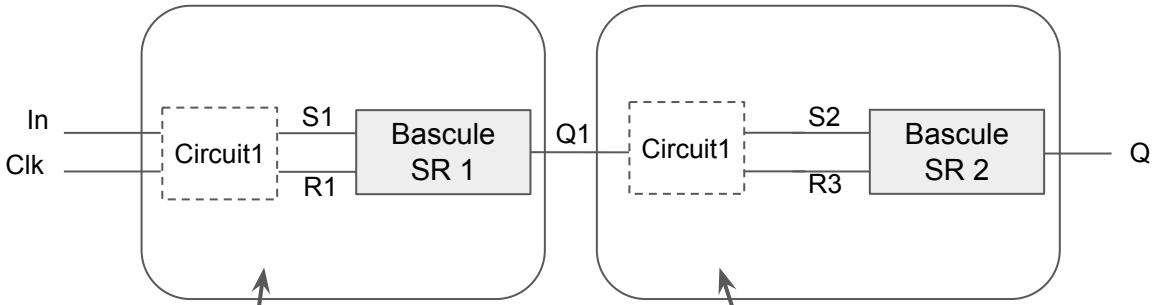
Delay flip-flop | **Idée du circuit**



(Diapositive hors champ)

Delay flip-flop | **Idée du circuit**

```
if signal bas:
    bascule 1 accepte valeur IN et produit valeur Q1
    bascule 2 conserve valeur Q
elif signal haut:
    bascule 1 conserve valeur Q1
    bascule 2 accepte valeur Q1 et produit valeur Q
```

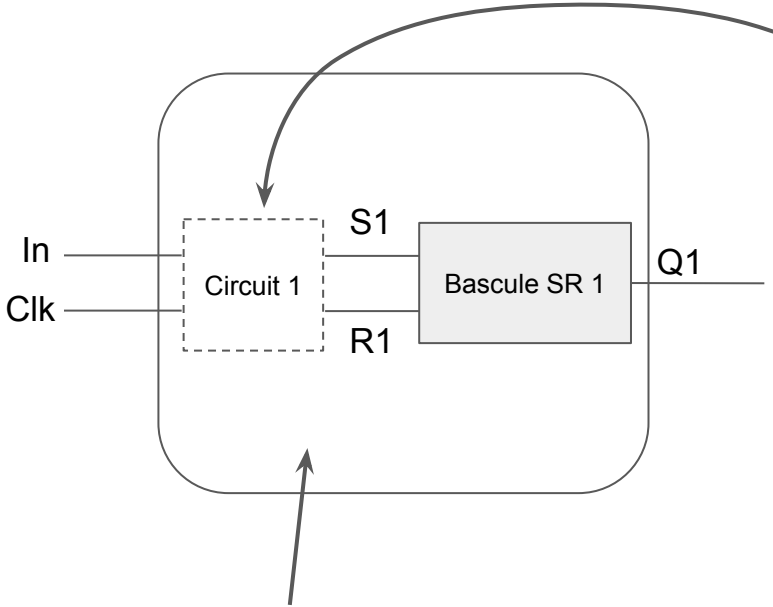


Retient la valeur d'entrée en attendant le signal d'horloge bas

Produit la sortie en attendant le signal d'horloge haut

(Diapositive hors champ)

Delay flip-flop | **Idée du circuit**

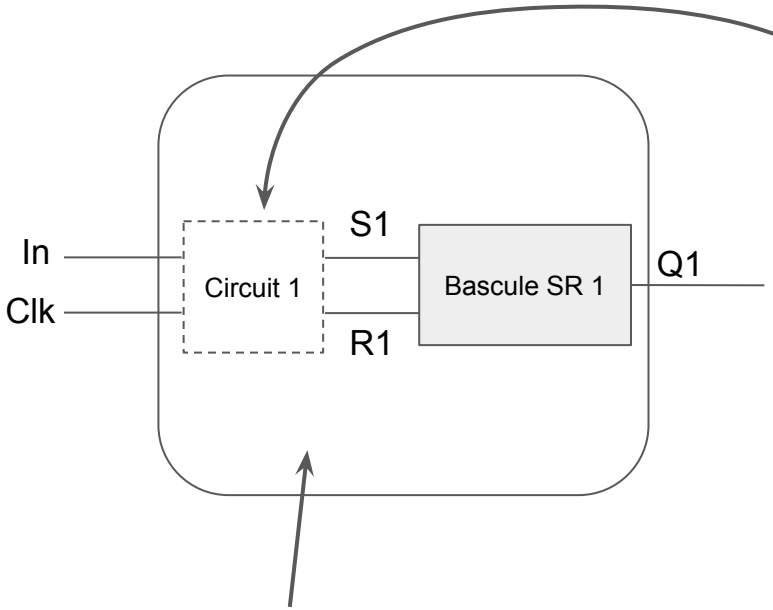



```
#Circuit 1
if Clk == 0:
    if In:
        S1 = 1
        R1 = 0
    else:
        S1 = 0
        R1 = 1
else:
    S1 = 0
    R1 = 0
```

Retient la valeur d'entrée en attendant le signal d'horloge bas

(Diapositive hors champ)

Delay flip-flop | **Idée du circuit**




```
#Circuit 1
if Clk == 0:
    if In:
        S1 = 1
        R1 = 0
    else:
        S1 = 0
        R1 = 1
else:
    S1 = 0
    R1 = 0
```

Signal bas : accepte l'input

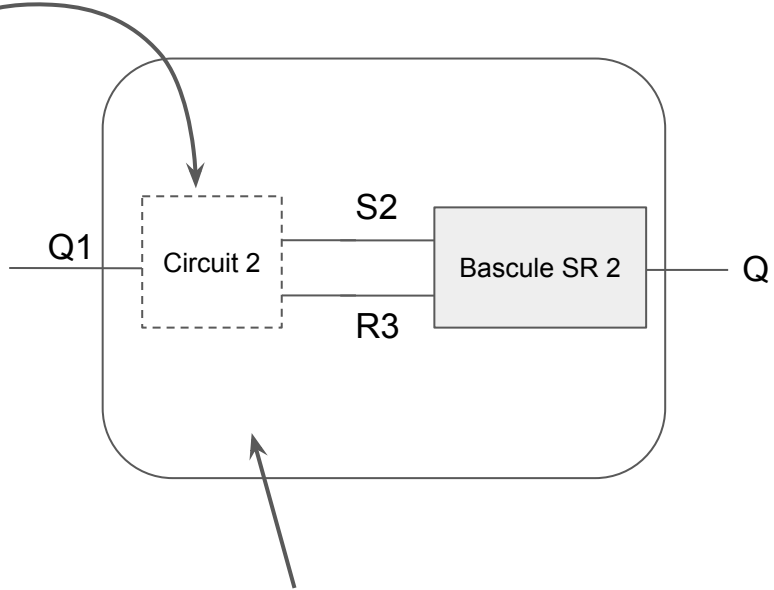
Signal haut : retient simplement la valeur

Retient la valeur d'entrée en attendant le signal d'horloge bas

(Diapositive hors champ)

Delay flip-flop | **Idée du circuit**

```
#Circuit 2
if Clk == 1:
    if Q1:
        S2 = 1
        R2 = 0
    else:
        S2 = 0
        R2 = 1
else:
    S2 = 0
    R2 = 0
```



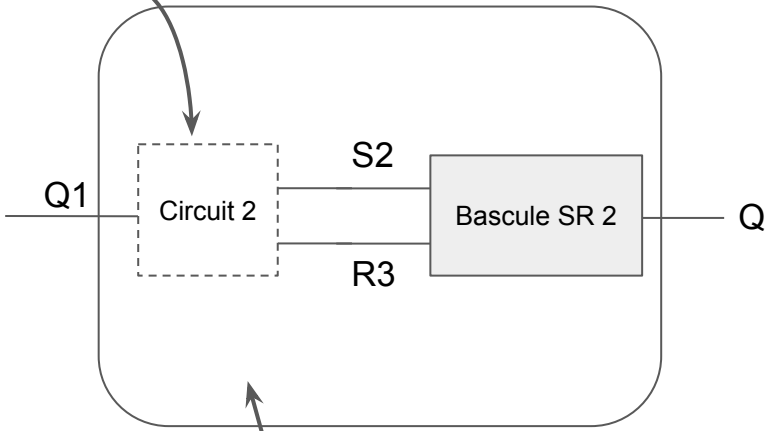
(Diapositive hors champ)

Delay flip-flop | **Idée du circuit**

Signal haut :
accepte l'input

```
#Circuit 2
if Clk == 1:
    if Q1:
        S2 = 1
        R2 = 0
    else:
        S2 = 0
        R2 = 1
else:
    S2 = 0
    R2 = 0
```

Signal bas : retient valeur



Produit la sortie en attendant le signal d'horloge

(Diapositive hors champ)

Delay flip-flop | **Tables de vérité des circuits 1 et 2**

```
#Circuit 1
if Clk == 0:
    if In:
        S1 = 1
        R1 = 0
    else:
        S1 = 0
        R1 = 1
else:
    S1 = 0
    R1 = 0
```

Clk	In	S_1	R_1	Q_1	S_2	R_2
0	0	0	1	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	1
1	1	0	0	1	1	0

```
#Circuit 2
if Clk == 1:
    if Q1:
        S2 = 1
        R2 = 0
    else:
        S2 = 0
        R2 = 1
else:
    S2 = 0
    R2 = 0
```

(Diapositive hors champ)

Delay flip-flop | **Tables de vérité des circuits 1 et 2**

```
#Circuit 1
if Clk == 0:
    if In:
        S1 = 1
        R1 = 0
    else:
        S1 = 0
        R1 = 1
else:
    S1 = 0
    R1 = 0
```

Clk	In	S_1	R_1	Q_1	S_2	R_2
0	0	0	1	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	1
1	1	0	0	1	1	0

$S1 = !Clk \cdot In$

$R1 = !Clk \cdot !In$

$S2 = Clk \cdot Q1$

$R2 = Clk \cdot !Q1$

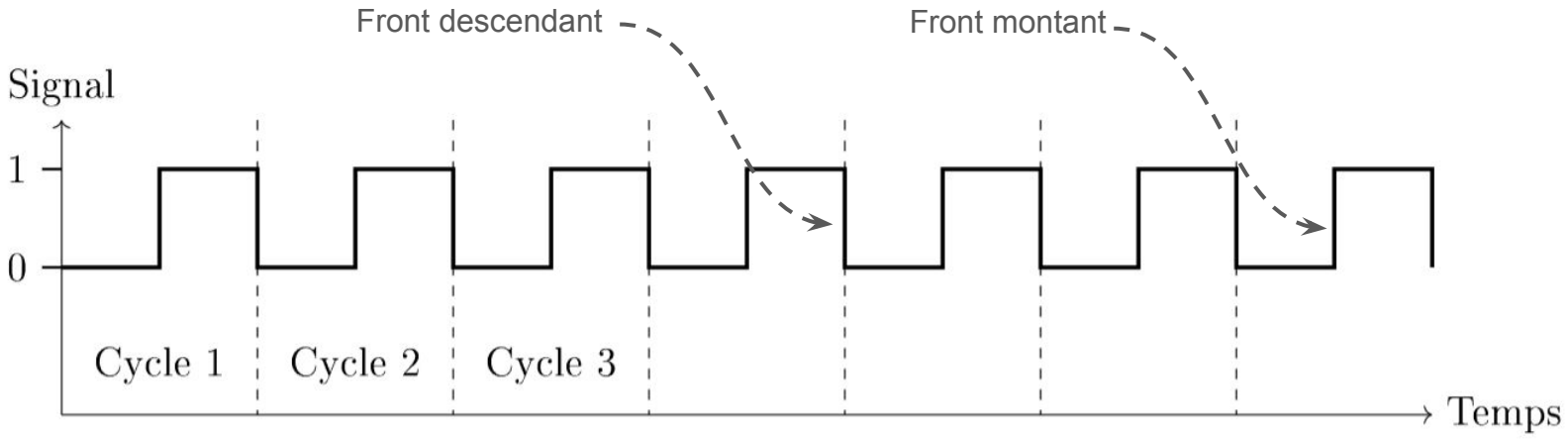
```
#Circuit 2
if Clk == 1:
    if Q1:
        S2 = 1
        R2 = 0
    else:
        S2 = 0
        R2 = 1
else:
    S2 = 0
    R2 = 0
```

Nous connaissons maintenant l'expression logique de tous les éléments d'un DFF. Diagramme complet non traité dans ce cours.

(Diapositive hors champ)

Comment détecter les fronts d'horloge ?

- Comment détecter le moment où le signal passe d'une valeur à l'autre ?
- Nécessaire pour que Clk code un "signal haut" ou un "signal bas".

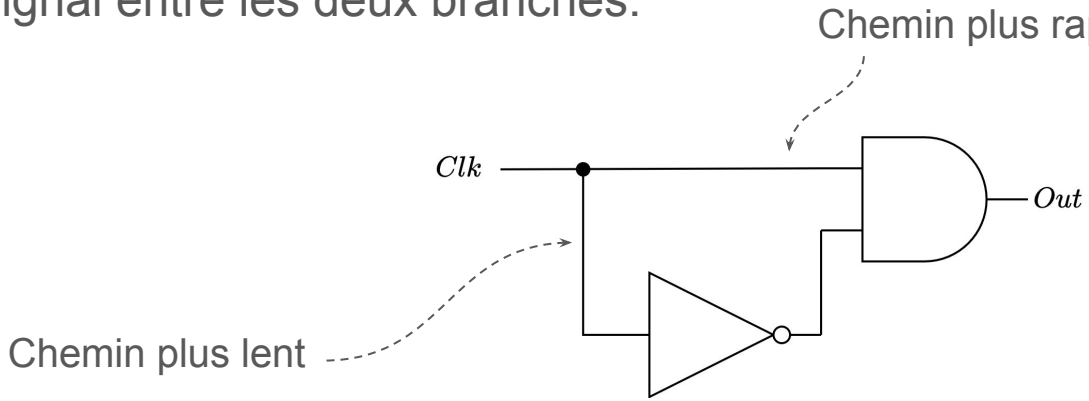


(Diapositive hors champ)

Comment détecter les fronts d'horloge ?

- Comment détecter le moment où le signal passe d'une valeur à l'autre ?

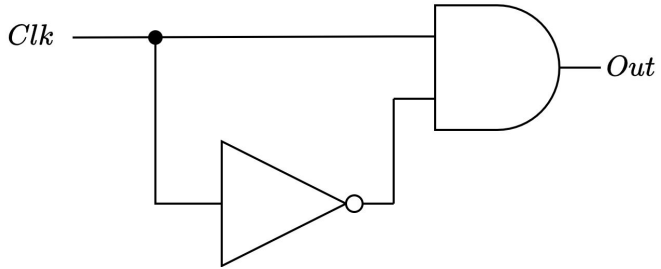
- Idée de base : exploitation d'une différence dans le temps de propagation du signal entre les deux branches.



(Diapositive hors champ)

Comment détecter les fronts d'horloge ?

Si le signal se propageait instantanément, `Out` aurait toujours la valeur `0`.



Mais le signal ne se propage pas instantanément, et durant un bref instant `Out = 1`.

(Diapositive hors champ)

Les tables de Karnaugh

Complément à la méthode des mintermes/maxtermes

(Diapositive hors champ)

Complément à la méthode des mintermes/maxtermes

- Passer d'une table de vérité à une expression booléenne : écrire toutes les expressions valant 1 et les combiner par des OU logique, puis simplifier le tout (méthode des mintermes).

- Ne donne pas nécessairement une expression optimale.

- Fastidieux si beaucoup de variables.

- La méthode de Karnaugh en revanche est assez efficace, en particulier pour les expressions de 3 ou 4 variables.

(Diapositive hors champ)

Complément à la méthode des mintermes/maxtermes

- Table de vérité : tableau à "une" entrée (variables côte-à-côte):

a	b	c	Out
0	0	0	0
0	0	0	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

```
Out = a!b!c + a!bc + abc

    = ... ?
```

- Table de Karnaugh : tableau à double entrée.
  - Les entrées sont des expressions.
  - D'une ligne à l'autre, une seule variable change de valeur par expression.
  - Seul l'output figure dans les cases.

(Diapositive hors champ)

Complément à la méthode des mintermes/maxtermes

Exemple de construction :

Table de Karnaugh

	ab	a!b	!a!b	!ab
c				
!c				

Table de vérité

a	b	c	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Quand ab = 1 et !c = 1, que vaut Out ?

(Diapositive hors champ)

Complément à la méthode des mintermes/maxtermes

Exemple de construction :

Table de Karnaugh

	ab	a!b	!a!b	!ab
c	1	1	0	0
!c	0	1	0	0

équivalent

Table de vérité

a	b	c	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- D'une ligne à l'autre, un seul bit change de valeur par expression.
- Seul l'output figure dans les cases.

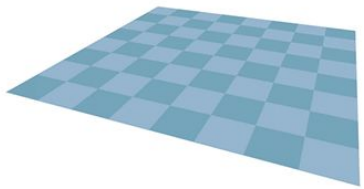
(Diapositive hors champ)

Complément à la méthode des mintermes/maxtermes

Exemple de construction :

Table de Karnaugh

	ab	a!b	!a!b	!ab
c	1	1	0	0
!c	0	1	0	0



À présent, construisons des "blocs" rectangulaires ne contenant que des 1 :

- Tous les 1 doivent être dans un bloc au moins.
- On veut le moins de blocs possibles.
- Un bloc doit contenir un nombre de cases qui est une puissance de 2.
- Les blocs peuvent être périodiques (connectés toroïdalement).

(Diapositive hors champ)

Complément à la méthode des mintermes/maxtermes

Exemple de construction :

Table de Karnaugh

	ab	a!b	!a!b	!ab
c	1	1	0	0
!c	0	1	0	0

À présent, construisons des "blocs" rectangulaires ne contenant que des 1 :

- Tous les 1 doivent être dans un bloc au moins.
- On veut le moins de blocs possibles.
- Un bloc doit contenir un nombre de cases qui est une puissance de 2.
- Les blocs peuvent être périodiques (connectés toroïdalement).

(Diapositive hors champ)

Complément à la méthode des mintermes/maxtermes

Exemple de construction :

Table de Karnaugh

	ab	a!b	!a!b	!ab
c	1	1	0	0
!c	0	1	0	0

Enfin, repérons les variables **redondantes** au sein des blocs.

Ici, b est redondant au sein du groupe bleu.

Ici, c est redondant au sein du groupe rouge.

On en conclut que `Out = ac + a!b = a(c + !b)`

Complément à la méthode des mintermes/maxtermes

Vérification de `Out = a(c + !b)`

a	b	c	Out
0	0	0	0
0	0	0	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

```
Out = a!b!c + a!bc + abc
    = a(!b!c + !bc + bc)
    = a(!b(!c + c) + bc)
    = a(!b + bc)
    = a(!b + c)
```