ELECINF 102 : Processeurs et Architectures Numériques

Logique séquentielle synchrone

Tarik Graba
tarik.graba@telecom-paris.fr

Plan

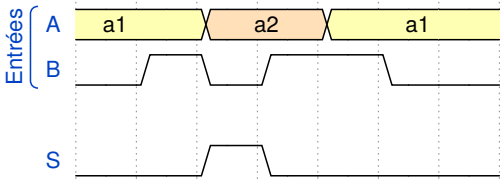
Rappel

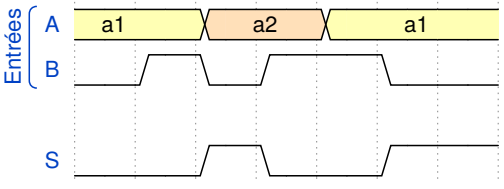
- La sortie d'une fonction ne dépend que de ses entrées
  - Pour les mêmes entrées on obtient toujours les mêmes sorties.
- Permet de construire des opérateurs :
  - Logiques
  - Arithmétiques

La fonction logique F à n entrées $E_0, ..., E_{n-1}$ s'exprime :

$$S = F(E_0, ..., E_{n-1})$$

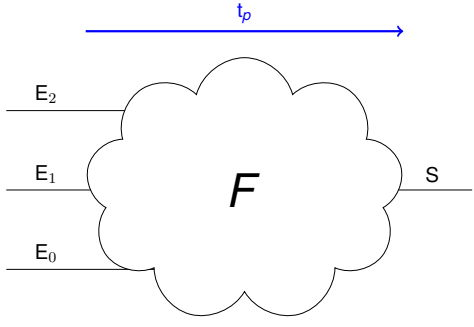
Introduction à la logique séquentielle





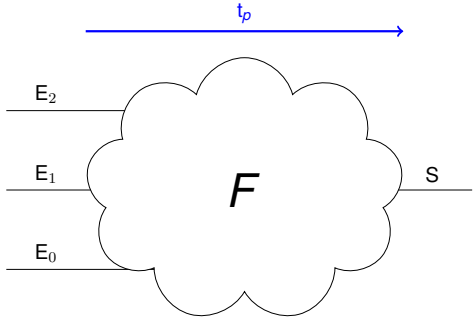
Lequel de ces chronogrammes ne peut pas être réalisé par la logique combinatoire ?

Logique séquentielle synchrone
Apprivoiser le temps de propagation



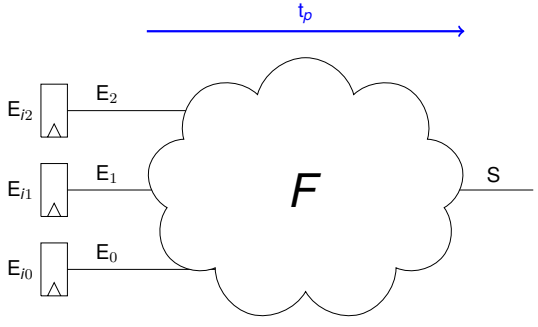
- Le temps de propagation t_p est non nul. Durant ce temps :
  - La sortie n'est pas valide !
  - On ne doit pas modifier les entrées !
- Comment enchaîner plusieurs calculs ?

Logique séquentielle synchrone
Apprivoiser le temps de propagation



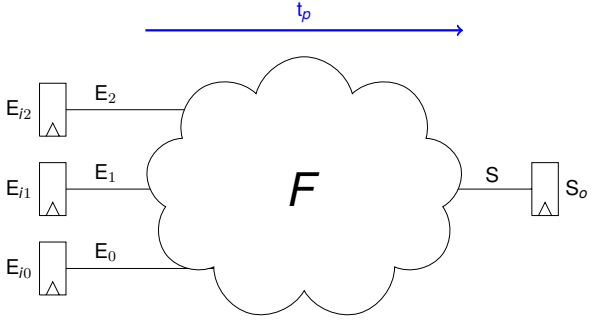
- On maintient les entrées stables au moins pendant t_p

Logique séquentielle synchrone
Apprivoiser le temps de propagation



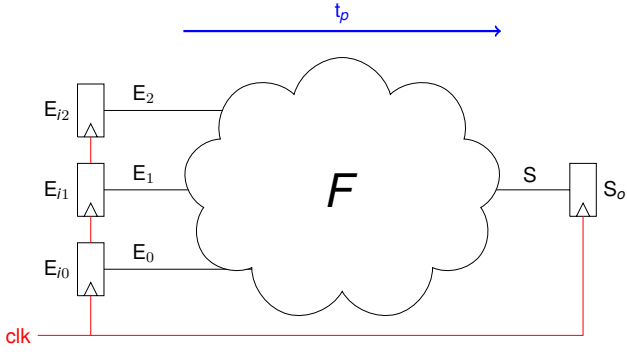
- On maintient les entrées stables au moins pendant t_p
- Rajout d'un élément mémoire :
  ⇒ mémorisation pour maintenir les entrées stables
  ⇒ échantillonnage pour la mise à jour des entrées

Logique séquentielle synchrone
Apprivoiser le temps de propagation



- Dès que le résultat est prêt :
  - La sortie est échantillonnée
  - Les entrées peuvent l'être en même temps
  - La sortie est mémorisée pour servir d'entrée à un prochain calcul

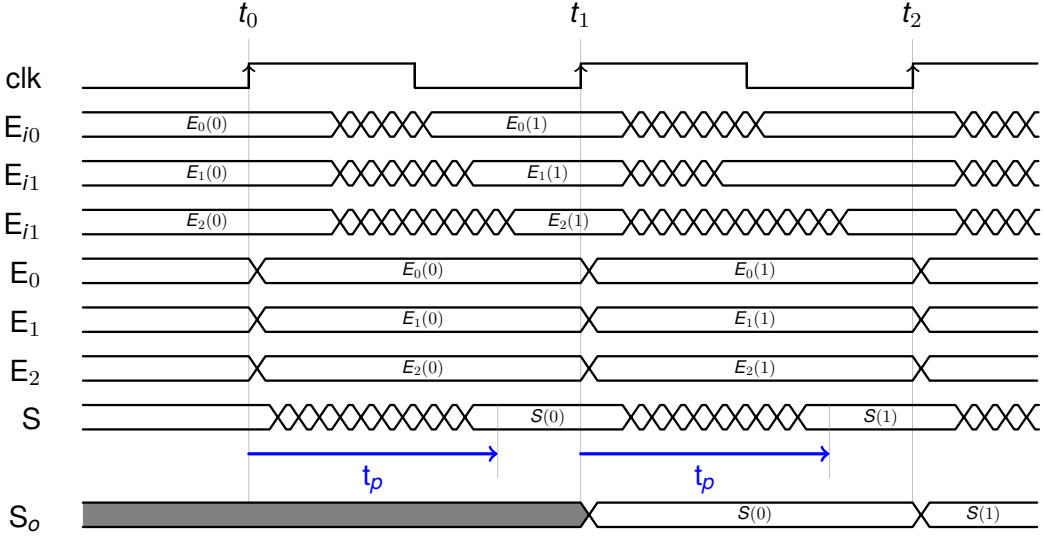
Logique séquentielle synchrone
Apprivoiser le temps de propagation



- On utilise le même signal de synchronisation pour l'échantillonnage dans les bascules :
  - **L'horloge**

Logique séquentielle synchrone

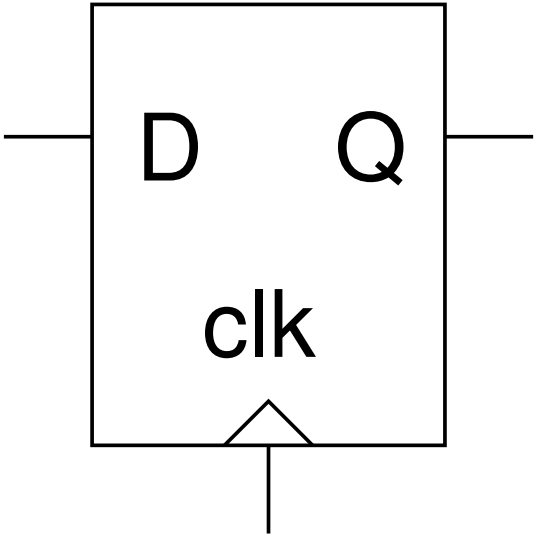
Apprivoiser le temps de propagation



Plan

La bascule D

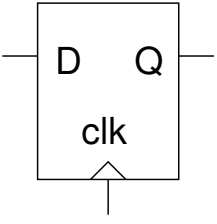
Élément de base de la logique séquentielle



La bascule D

Élément de base de la logique séquentielle

- Plusieurs dénominations :
  bascule D, flipflop, dff, registre …
- entrée D, sortie Q
- entrée d'horloge **clk**
  symbolisée par un triangle



Fonctionnement :

- A chaque front montant de l'horloge **clk** (passage de $0 \rightarrow 1$) l'entrée **D** est copiée sur la sortie **Q**.
  $\Rightarrow$ échantillonnage
- Entre deux fronts d'horloge, la sortie **Q** ne change pas.
  $\Rightarrow$ mémorisation

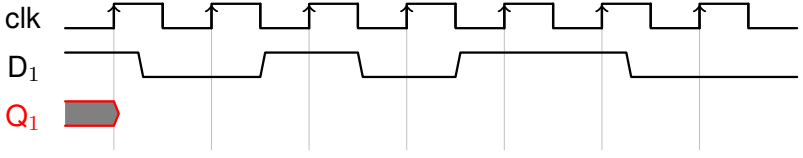
La bascule D

Table de vérité de la bascule D

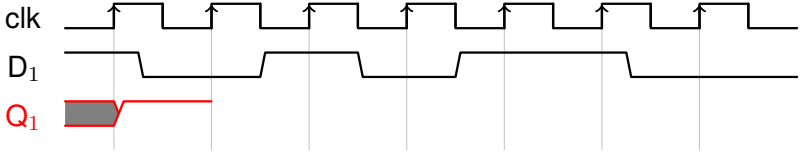
D	clk	Q		
0	$\uparrow$	0	copie de D sur Q	échantillonnage
1	$\uparrow$	1	copie de D sur Q	échantillonnage
$\times$	0	Q	Q conserve sa valeur	mémorisation
$\times$	1	Q	Q conserve sa valeur	mémorisation
$\times$	$\downarrow$	Q	Q conserve sa valeur	mémorisation

La bascule D

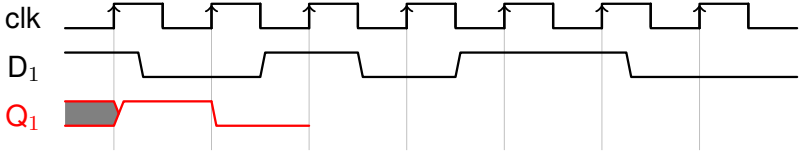
Exemple



La bascule D
Exemple

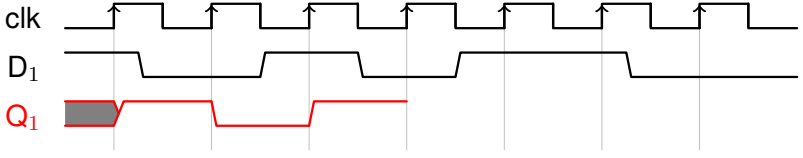


La bascule D
Exemple

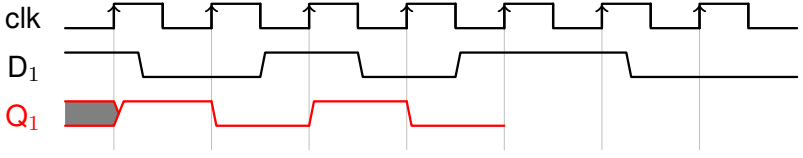


La bascule D

Exemple

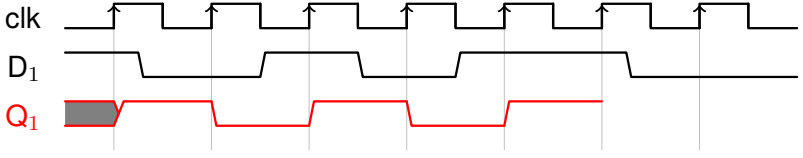


La bascule D
Exemple

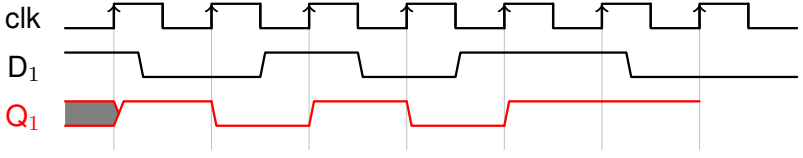


La bascule D

Exemple

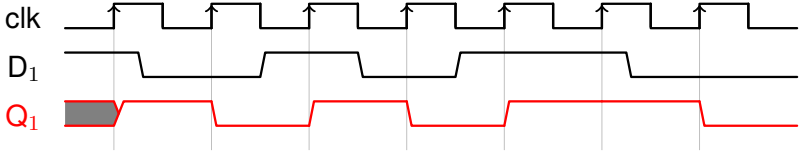


La bascule D
Exemple



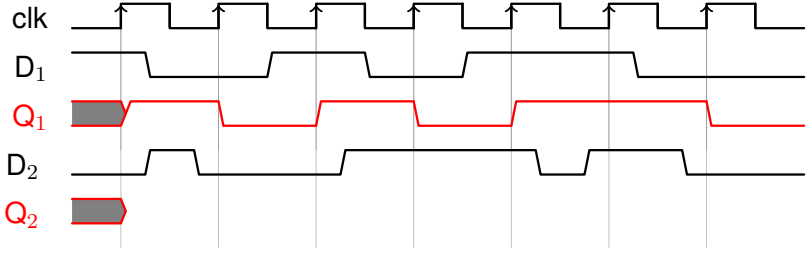
La bascule D

Exemple



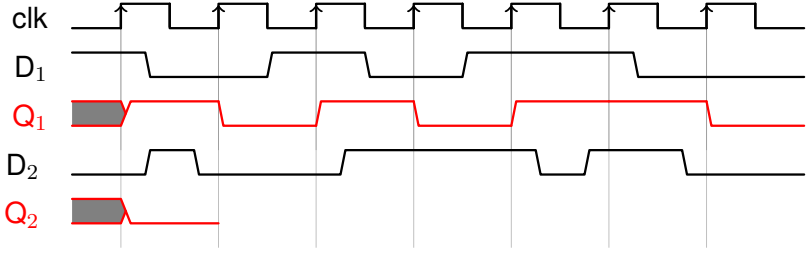
La bascule D

Exemple



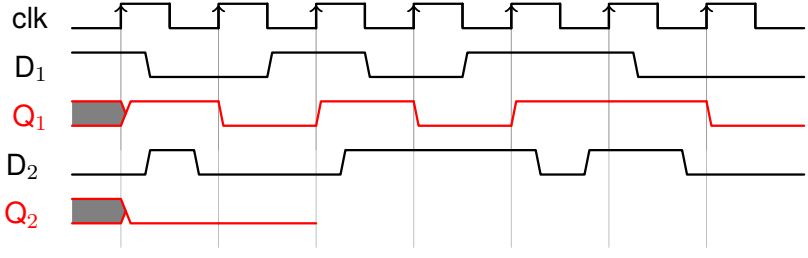
La bascule D

Exemple



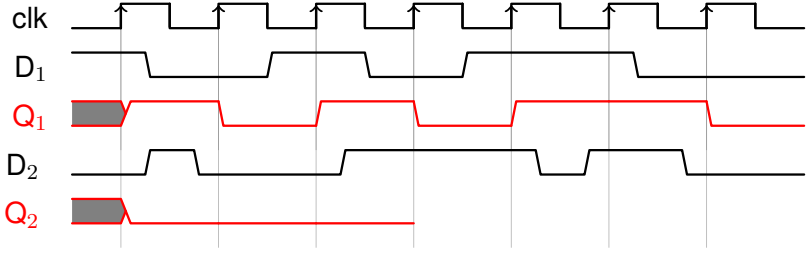
La bascule D

Exemple



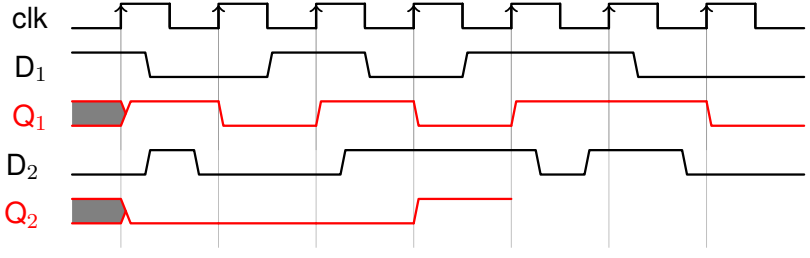
La bascule D

Exemple



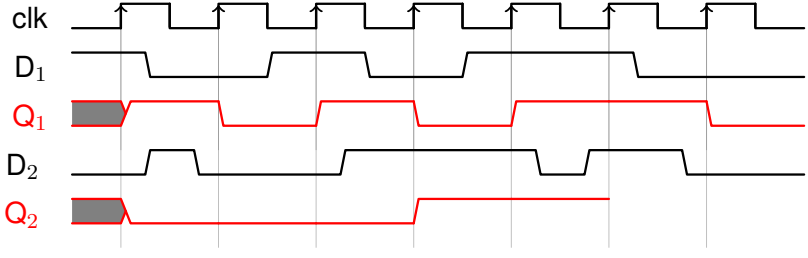
La bascule D

Exemple



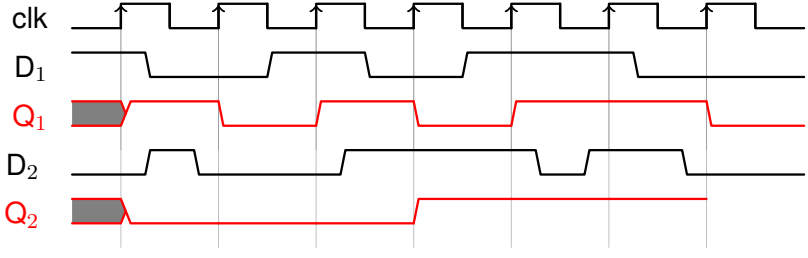
La bascule D

Exemple



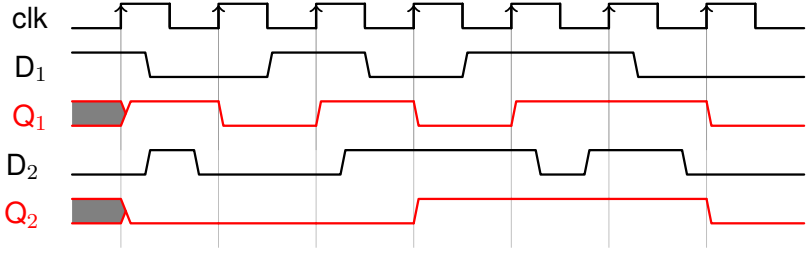
La bascule D

Exemple

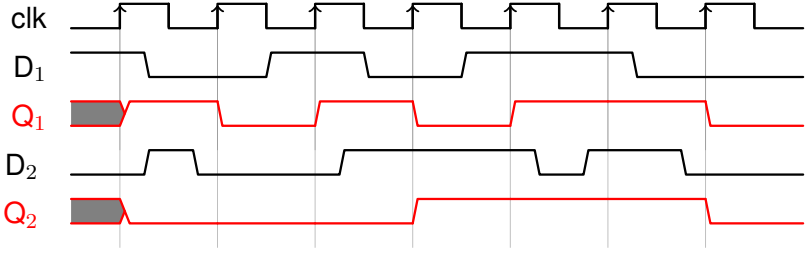


La bascule D

Exemple



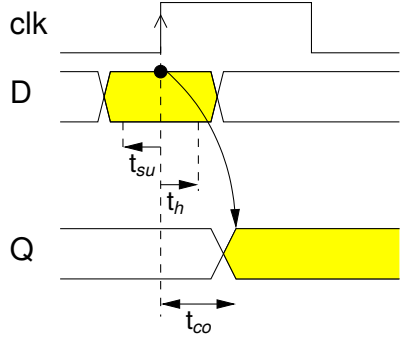
La bascule D
Exemple



- Q_1 recopie D_1 avec un cycle de retard.
- Q_2 recopie D_2 en filtrant les impulsions de durée inférieure à la période de l'horloge.

La bascule D

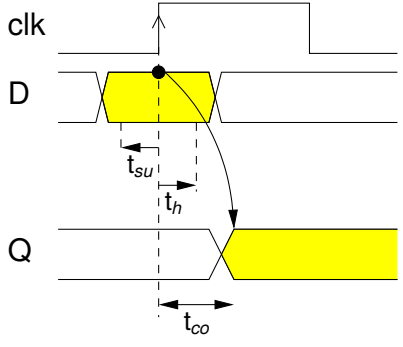
Contraintes temporelles



- La donnée doit être stable au front d'horloge :
  - avoir atteint sa valeur t_{su} avant le front d'horloge (setup).
  - cette valeur doit être maintenue t_h après le front d'horloge (hold).
- La copie de l'entrée sur la sortie se fait avec un retard de t_{co} (clock to output).

La bascule D

Contraintes temporelles



t_{su} : temps de pré--positionnement

t_h : temps de maintien

t_{co} : temps de propagation

La bascule D
en System Verilog

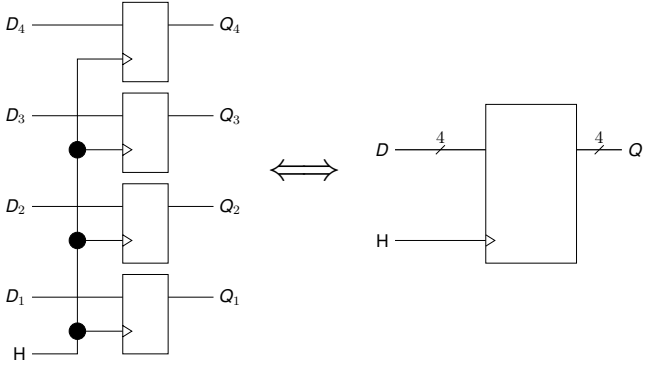
A chaque front d'horloge, copier l'entrée sur la sortie.
Sinon, la sortie ne change pas.

En SystemVerilog

```
always @(posedge clk)
   q <= d;
```

Application de la bascule D
Registre

- Un registre est un ensemble de bascules fonctionnant en parallèle.
  - Exemple : un registre 4 bits



Application de la bascule D

Registre en SystemVerilog

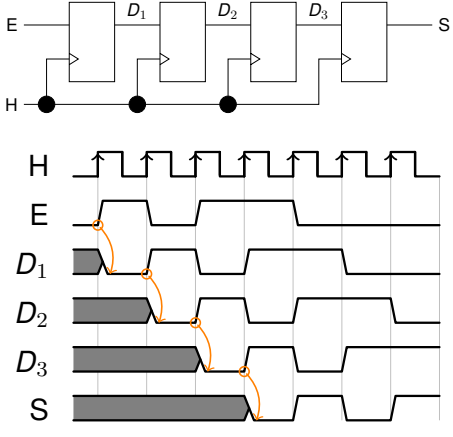
Un registre de 4 bits

```
logic[3:0] d;
...
logic[3:0] q;

always @(posedge clk)
   q <= d;
```

Application de la bascule D

Registre à décalage



- fonctionne car $t_{co} > t_h$ toujours vrai
- génère un signal retardé de n périodes d'horloge
- Parallèlisation des données : entrée série E , sorties sur les Q_i des bascules
- Sérialisation des données : entrées sur les D_i des bascules, sortie sur Q_n

Application de la bascule D

Registre à décalage en SystemVerilog

Affectations simultanées

```
logic a, b, c, d;

always @(posedge clk)
begin
   b <= a;
   c <= b;
   d <= c;
end
```

```
logic a, b, c, d;

always @(posedge clk)
begin
   d <= c;
   c <= b;
   b <= a;
end
```

Dans un bloc, entre begin et end, les affectations sont faites simultanément.
L'ordre n'a donc pas d'importance et les deux codes sont équivalents.

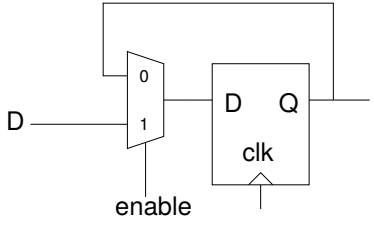
- À **droite** d'une affectation <=, l'état **avant** le front.
- À **gauche** d'une affectation <=, l'état **après** le front.

Application de la bascule D
L'enable

- Construire une bascule D avec une entrée d'activation (enable).
  - enable vaut 1 la bascule fonctionne normalement.
  - enable vaut 0 la bascule ne change pas d'état.

Application de la bascule D
L'enable

- Construire une bascule D avec une entrée d'activation (enable).
  - enable vaut 1 la bascule fonctionne normalement.
  - enable vaut 0 la bascule ne change pas d'état.



La bascule D
l'enable en System Verilog

```
always @(posedge clk)
   if (en)
      q <= d;
```

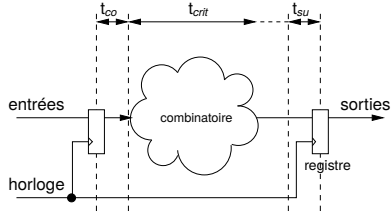
Plan

Logique séquentielle synchrone
Règles

La logique séquentielle ``synchrone''

- Tout bloc combinatoire est entouré par des registres (bascules D).
- Les registres sont synchrones
  - Ils utilisent le même signal d'horloge
  - L'horloge doit arriver en même temps
  - pas de combinatoire sur l'horloge
- La période de l'horloge dont être compatible avec le temps de propagation de la logique combinatoire

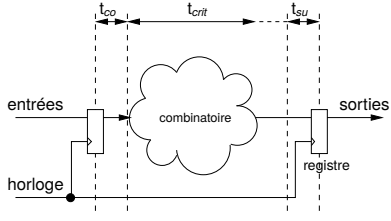
Logique séquentielle synchrone
Fréquence de fonctionnement



$$T_{clk} > t_{co} + t_{crit} + t_{su}$$

- t_{crit} est le temps du chemin critique càd le temps de propagation du chemin combinatoire le plus long

Logique séquentielle synchrone
Fréquence de fonctionnement

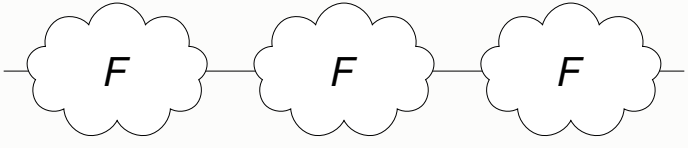


$$F_{max} = \frac{1}{t_{co} + t_{crit} + t_{su}}$$

- t_{crit} est le temps du chemin critique càd le temps de propagation du chemin combinatoire le plus long

Séquencer les traitements

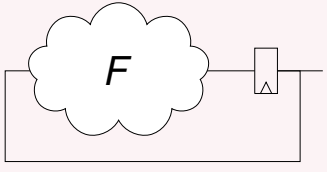
Répéter plusieurs fois le même calcul



- Comment effectuer plusieurs fois le même calcul sans dupliquer les opérateurs ?

Séquencer les traitements

Mémoriser et réutiliser le même bloc combinatoire



- Mémoriser les résultats intermédiaires.
- Reboucler sur la partie combinatoire.

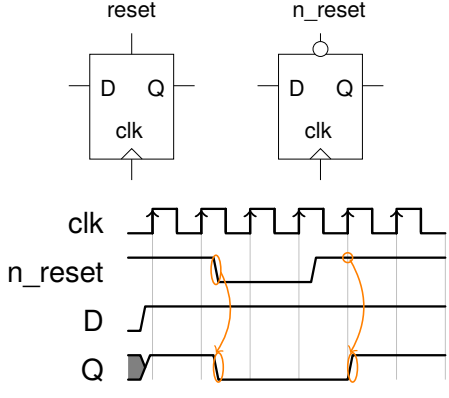
Séquencer les traitements
La valeur initiale ?

- Un signal extérieur est utilisé pour forcer l'état des bascules à zéro :

  $\Rightarrow$ Le ``reset''.
- Il peut s'agir d'une mise à ``1'', on parle alors de ``preset''.

Séquencer les traitements
Le reset asynchrone

- Reset **A**synchrone :
  $\Rightarrow$ Son action est indépendante de l'horloge.
- connecté directement aux bascules par une entrée spéciale.
- global pour tout le circuit.
- Peut être positif (actif à 1) ou négatif (actif à 0).
  convention :
  n_reset $\Rightarrow$ négatif car *n_* devant.



Séquencer les traitements
Reset asynchrone en SystemVerilog

Bascule D avec Reset asynchrone actif sur niveau haut

```
always @(posedge clk or posedge reset)
   if(reset)
      q <= 1'b0;
   else
      q <= d;
```

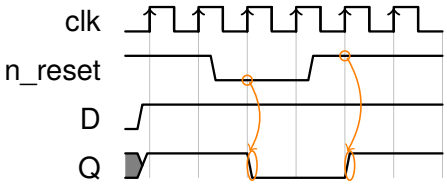
Séquencer les traitements

Reset asynchrone en SystemVerilog

Bascule D avec Reset asynchrone actif sur niveau bas

```
always @(posedge clk or negedge nreset)
   if(!nreset)
      q <= 1'b0;
   else
      q <= d;
```

Séquencer les traitements
Le reset synchrone



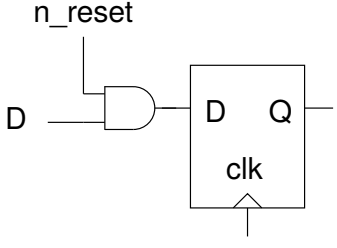
- Reset synchrone :
  $\Rightarrow$ son action n'est effective qu'au front de l'horloge.
- Vient de la logique qui précède les bascules.
- C'est une remise à zéro fonctionnelle.

Séquencer les traitements
Le reset synchrone

- Comment construire un reset synchrone en utilisant les portes logiques de base ?

Séquencer les traitements
Le reset synchrone

- Comment construire un reset synchrone en utilisant les portes logiques de base ?



Séquencer les traitements
reset synchrone en SystemVerilog

Bascule D avec Reset synchrone actif sur niveau haut

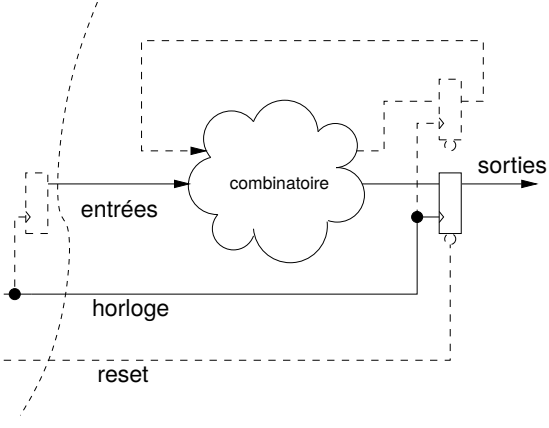
```
always @(posedge clk)
   if(reset)
      q <= 1'b0;
   else
      q <= d;
```

Séquencer les traitements
reset synchrone en SystemVerilog

Bascule D avec Reset synchrone actif sur niveau bas

```
always @(posedge clk)
   if(!nreset)
      q <= 1'b0;
   else
      q <= d;
```

Logique séquentielle synchrone
Récapitulatif



- Utilisation de bascules D sur les entrées et sorties.
- Utilisation d'une **Horloge** globale.
- La sortie dépend de ses entrées et de son état précédent.
  $S_i = F(S_{i-1}, E_0, ..., E_{n-1})$
- L'état initial peut être forcé par un signal global extérieur : le **Reset**.

Plan

Exemples d'applications
Un compteur

- Construisez un compteur ($+1$ à chaque coup d'horloge) de 0 à 255.
- Ajoutez la possibilité de le remettre à zéro
  - de façon asynchrone
  - de façon synchrone
- Ajoutez la possibilité d'interrompre/reprendre le comptage.
- Ajoutez la possibilité qu'il compte ou décompte.

Exemples d'applications
Un compteur simple en SystemVerilog

Un compteur modulo 16

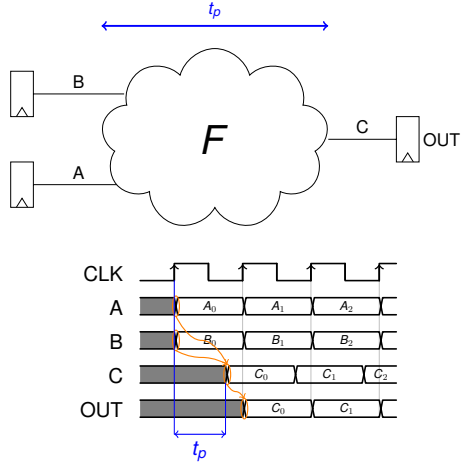
```
logic[3:0] q;

always @(posedge clk)
   q <= q + 1;
```

Comment coder l'initialisation, l'interruption et le décompatge ?

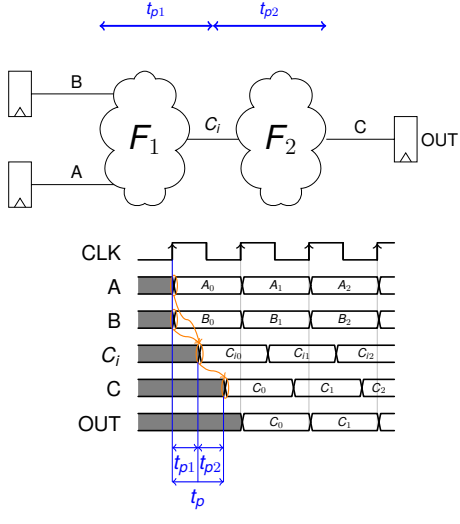
Exemples d'applications
Le Pipeline

- Une fonction combinatoire F de temps de propagation t_p
- Les entrées sorties peuvent être synchrones tant que $t_p < T_{CLK}$
  - Où T_{CLK} est la période d'horloge



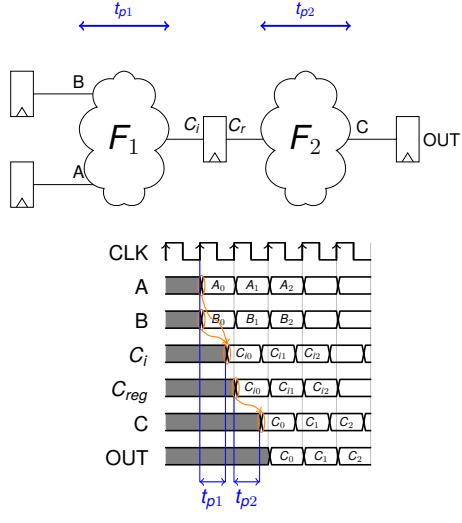
Exemples d'applications
Le Pipeline

- On décompose F en deux fonctions combinatoires F_1 et F_2 de temps de propagation respectifs t_{p1} et t_{p2}
  - On s'arrange pour avoir $t_{p1} < t_p$ et $t_{p2} < t_p$
  - Dans cet exemple $t_{p1} + t_{p2} = t_p$
- Les entrées sorties peuvent être synchrones tant que $t_{p1} + t_{p2} < T_{CLK}$



Exemples d'applications
Le Pipeline

- On peut échantillonner la sortie de la fonction F_1
- Les entrées sorties peuvent être synchrones tant que $t_{p1} < T_{CLK}$ et $t_{p2} < T_{CLK}$
  - On peut donc réduire la période de l'horloge
  - Ou augmenter la fréquence



Exemples d'applications
Le Pipeline

- Le pipeline permet d'augmenter la fréquence de fonctionnement.
- On a augmenté la taille du circuit (Ajout des bascules, modification de la partie combinatoire).
- On a augmenté la latence initiale.