ELECINF102 : Processeurs et Architectures Numériques

Logique séquentielle

Guillaume Duc
guillaume.duc@telecom-paris.fr

Plan

La bascule D

La logique séquentielle synchrone

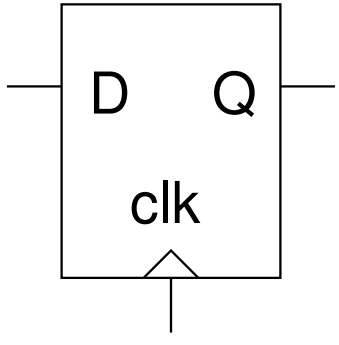
Applications

En SystemVerilog

La bascule D

Élément de base de la logique séquentielle

- Bascule D, flipflop, dff, registre . . .
- Une entrée particulière : l'horloge **clk**
- L'horloge est symbolisée par un triangle
- Fonctionnement
  - À chaque front montant de l'horloge **clk** (passage de 0 $\rightarrow$ 1) l'entrée **D** est copiée (échantillonnée, mémorisée) sur la sortie **Q**
  - Entre deux fronts d'horloge, la sortie **Q** ne change pas



La bascule D

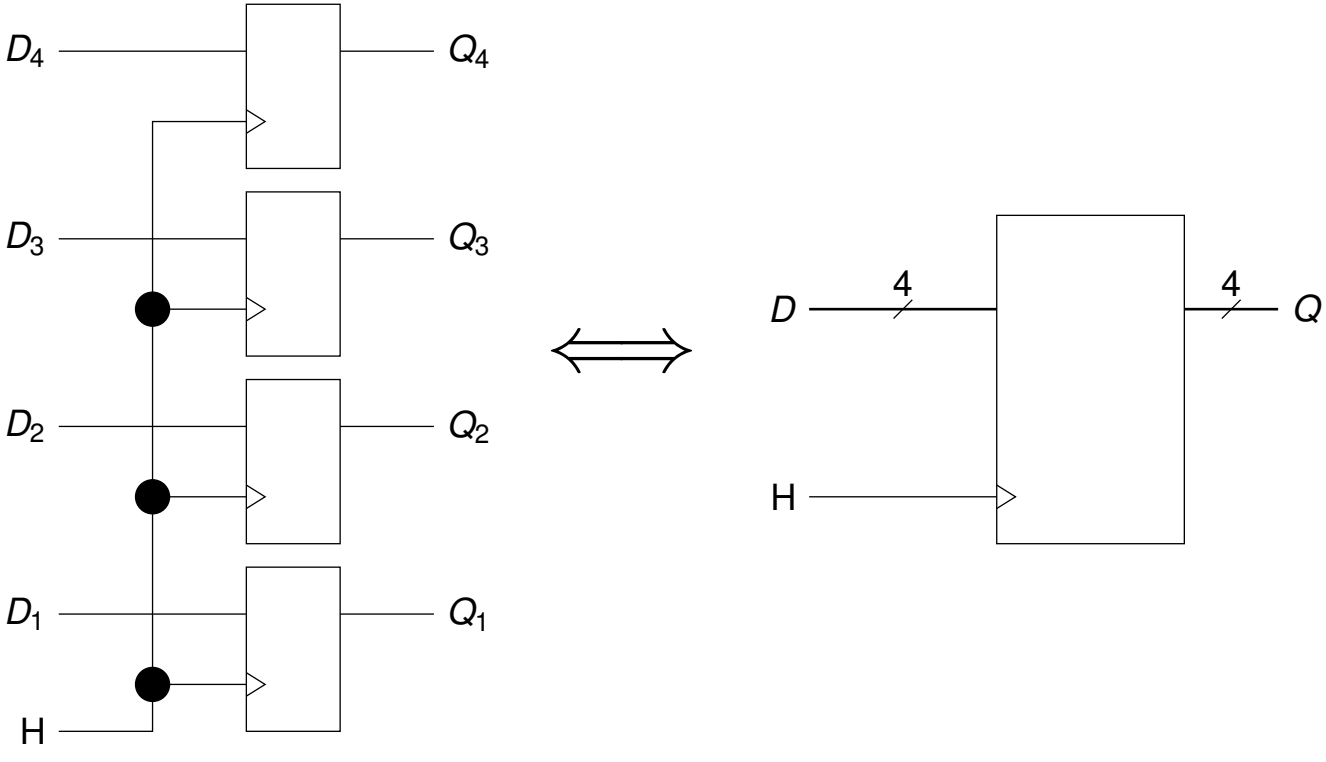
Table de vérité de la bascule D

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

La bascule D

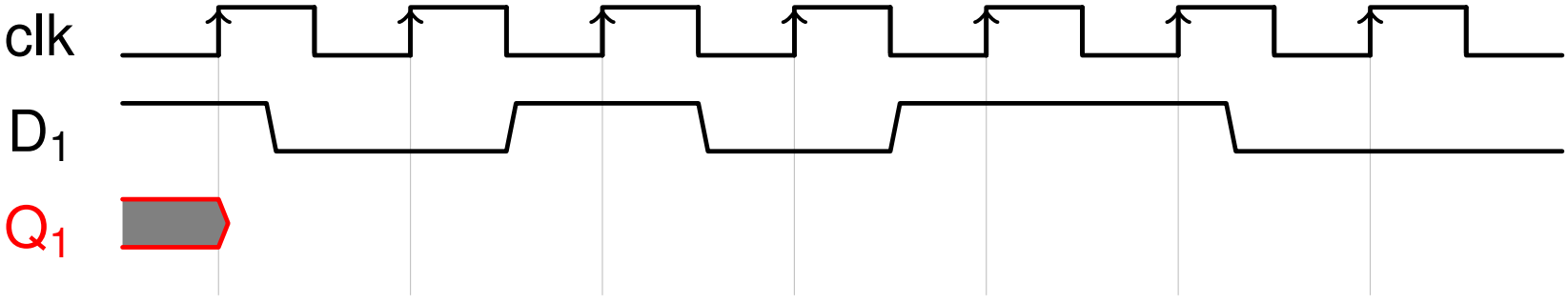
Le registre

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



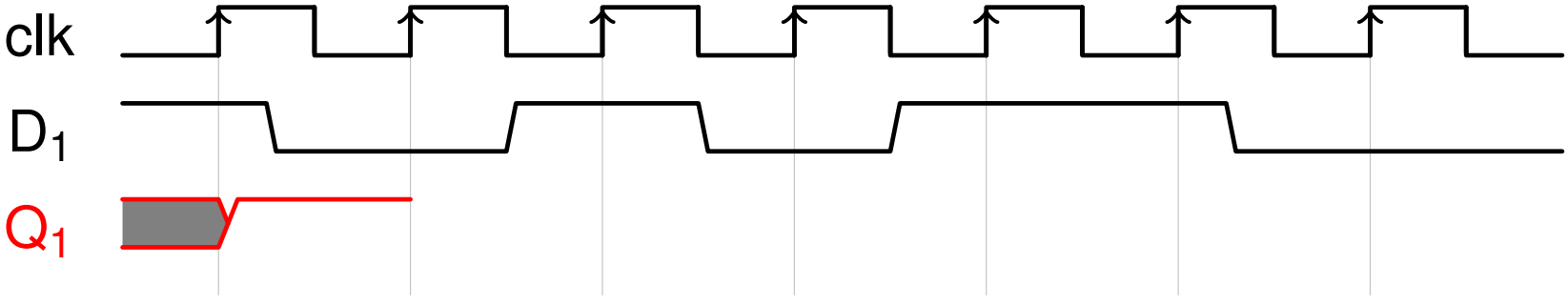
La bascule D

Exemple



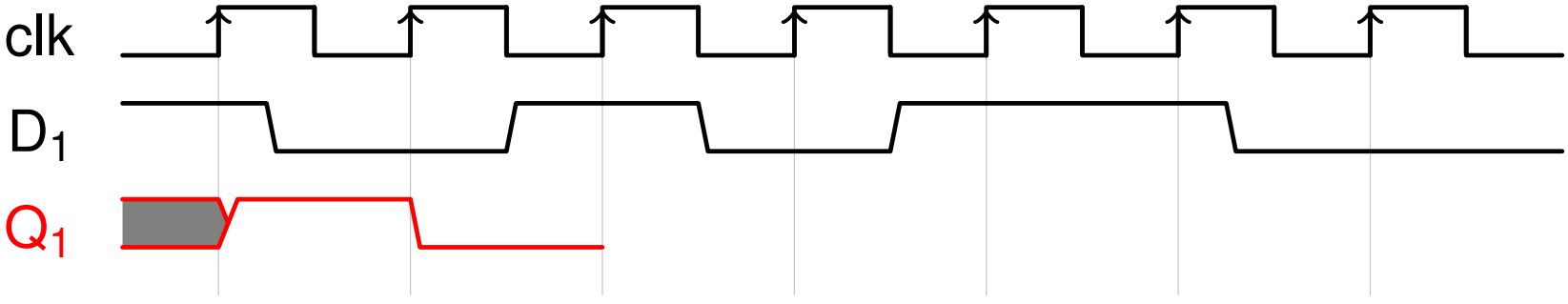
La bascule D

Exemple



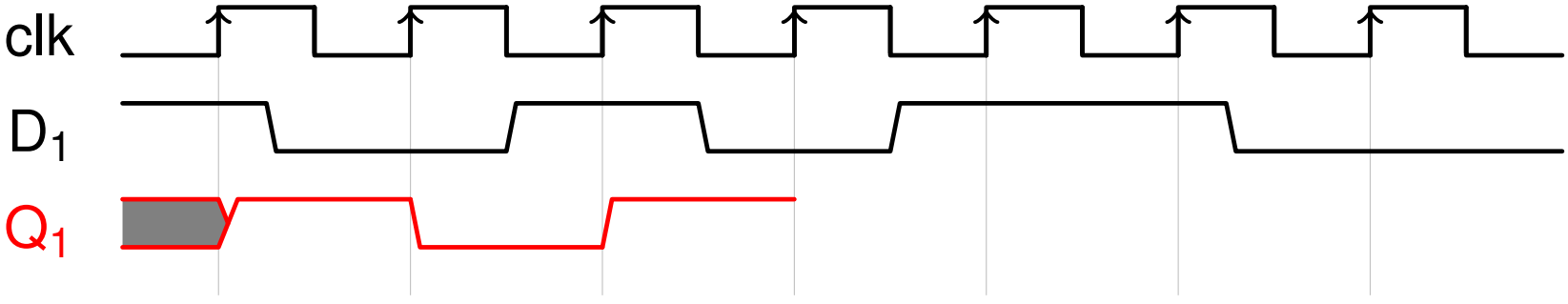
La bascule D

Exemple



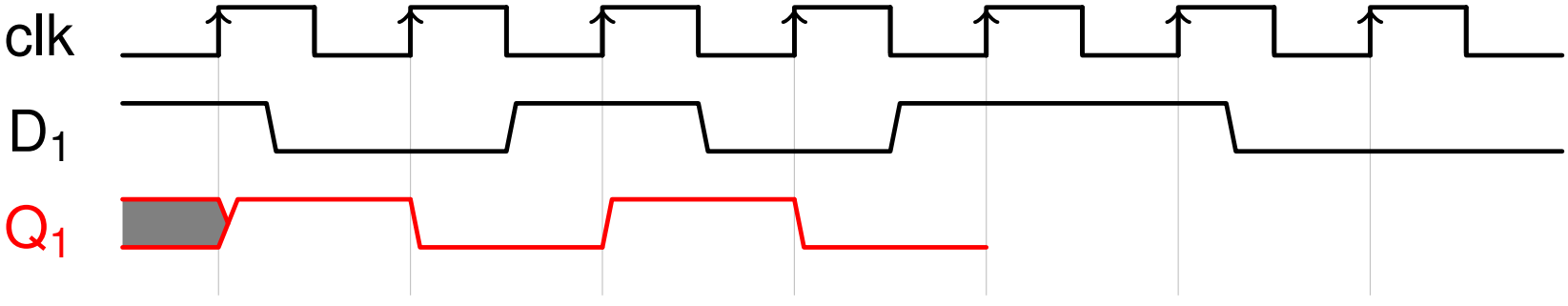
La bascule D

Exemple



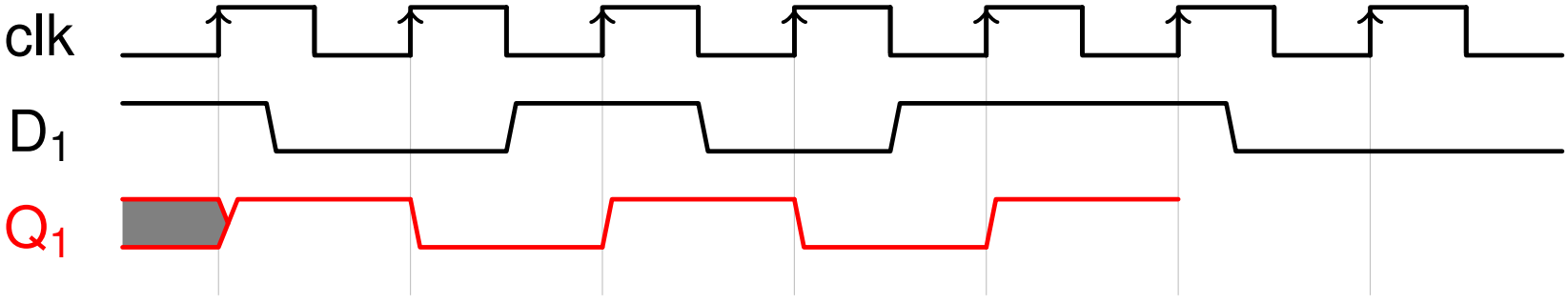
La bascule D

Exemple



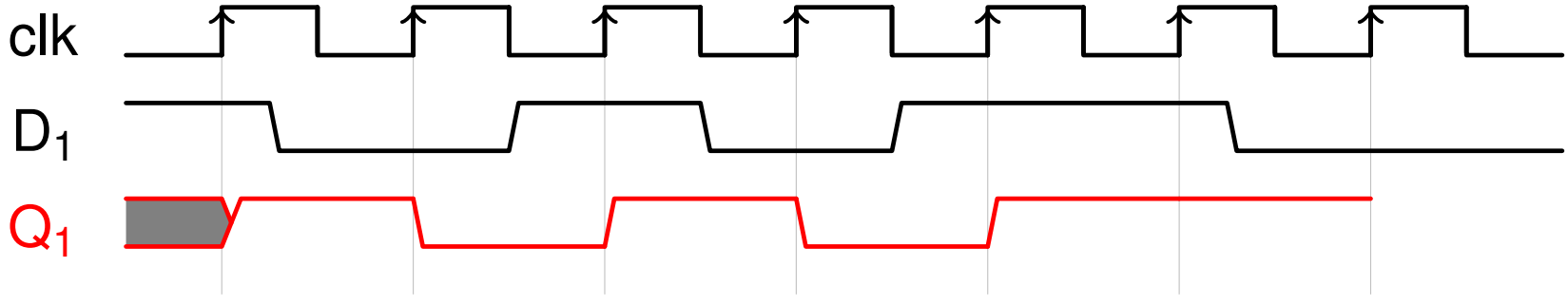
La bascule D

Exemple



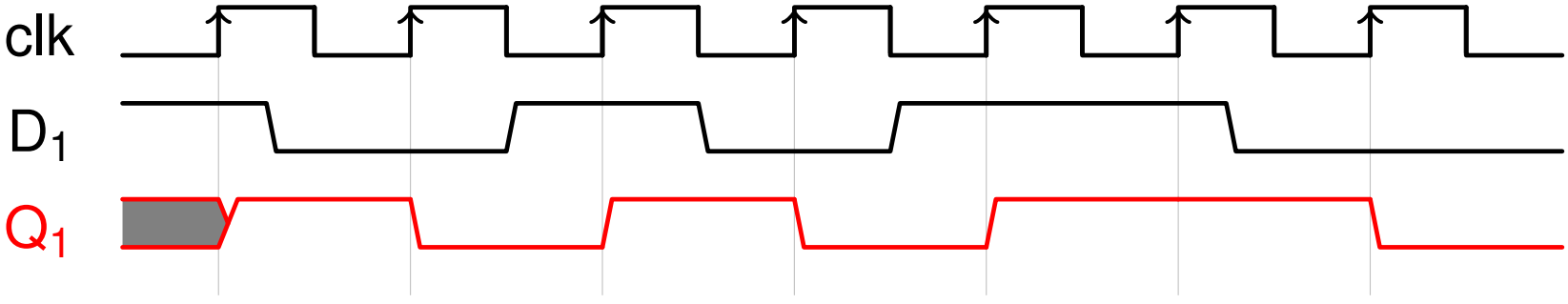
La bascule D

Exemple



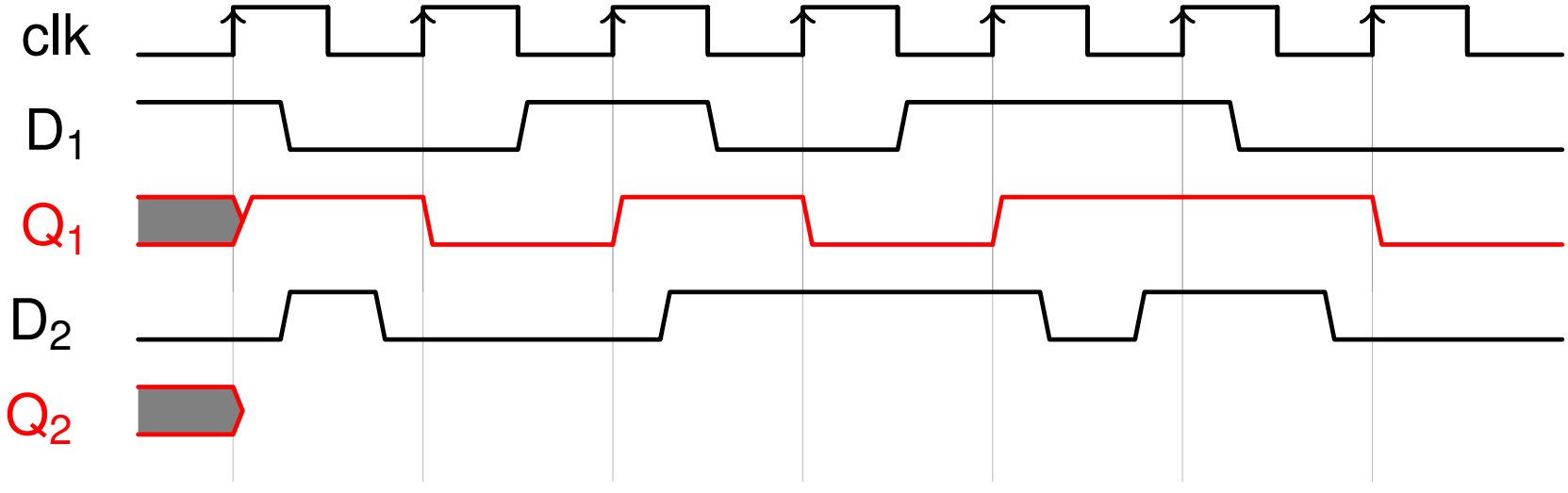
La bascule D

Exemple



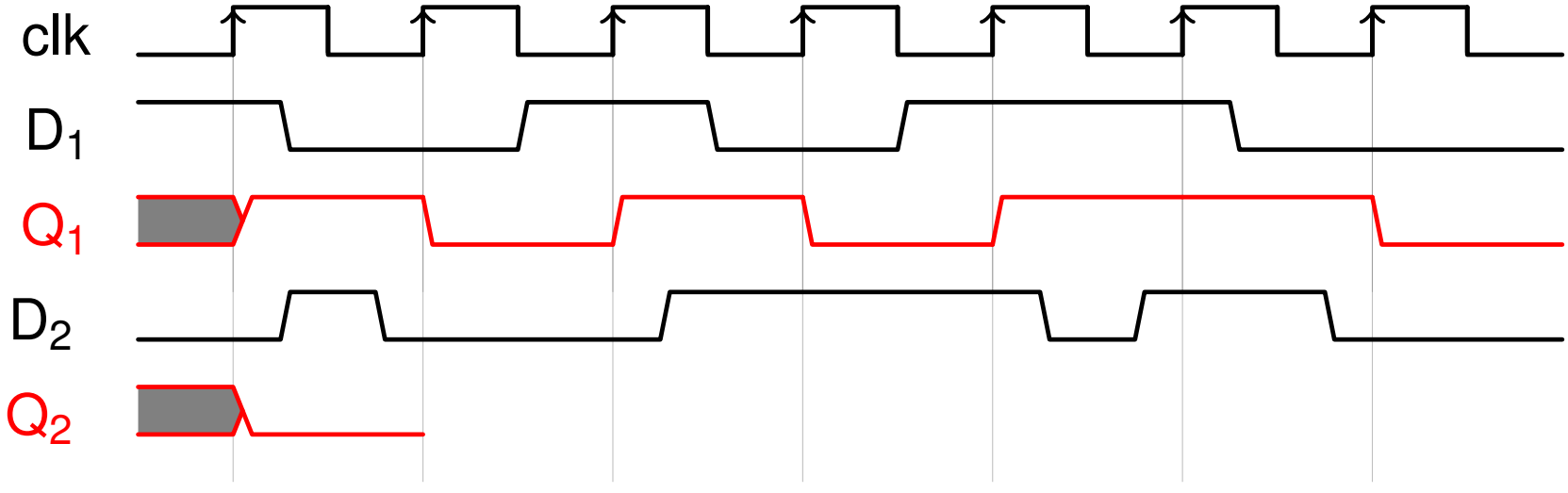
La bascule D

Exemple



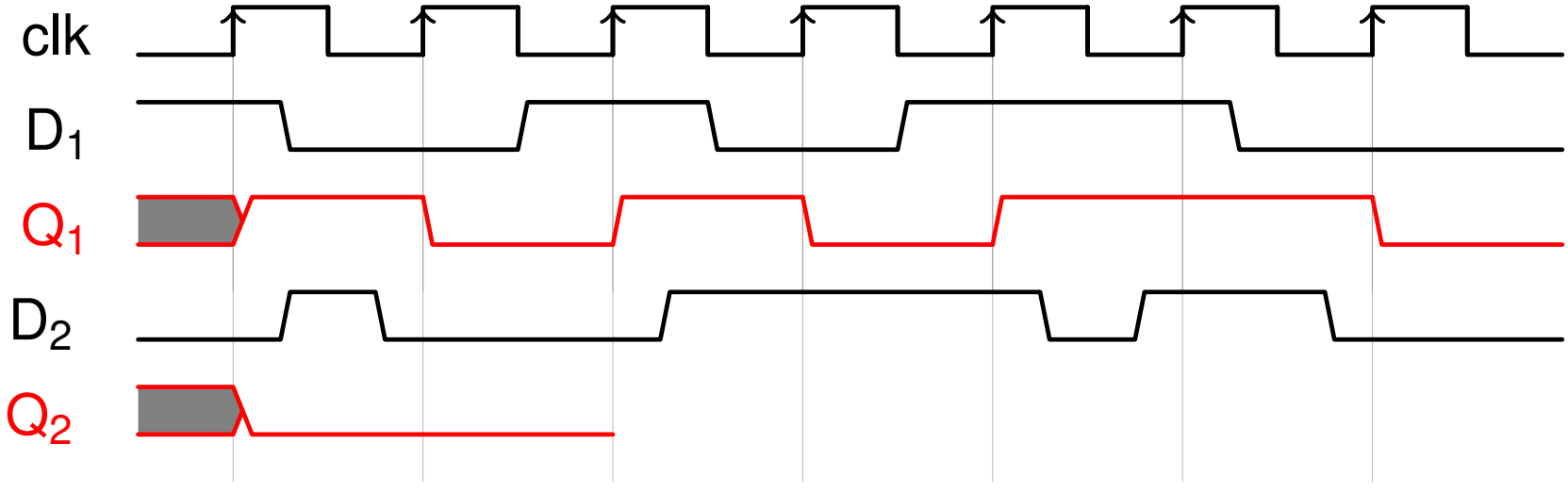
La bascule D

Exemple



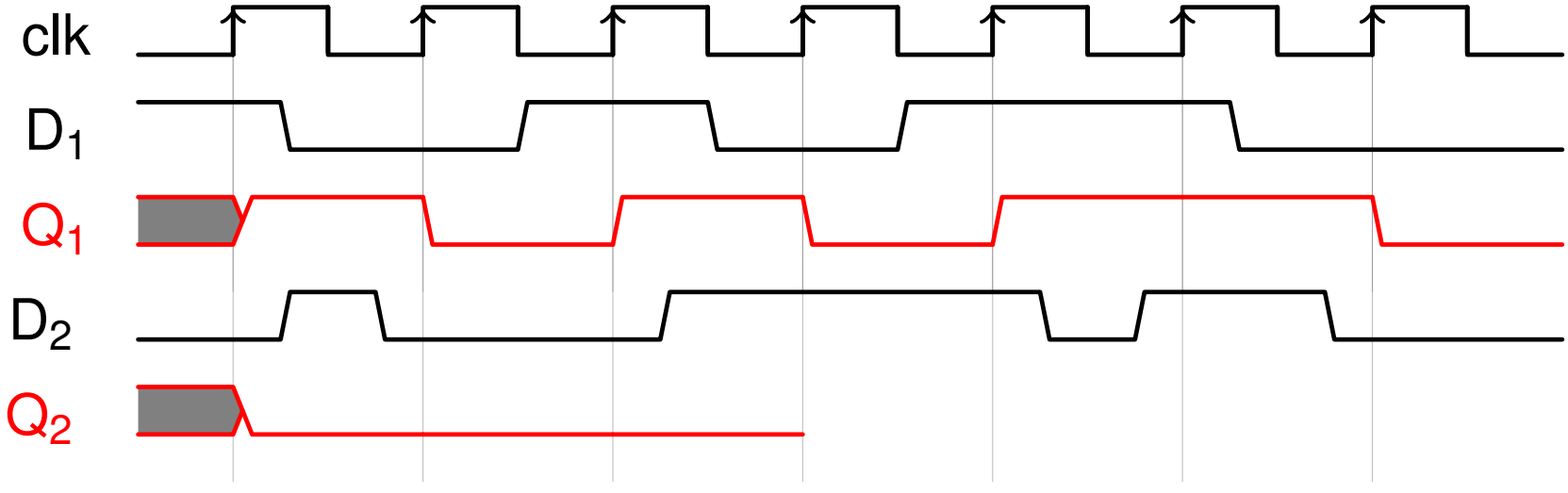
La bascule D

Exemple



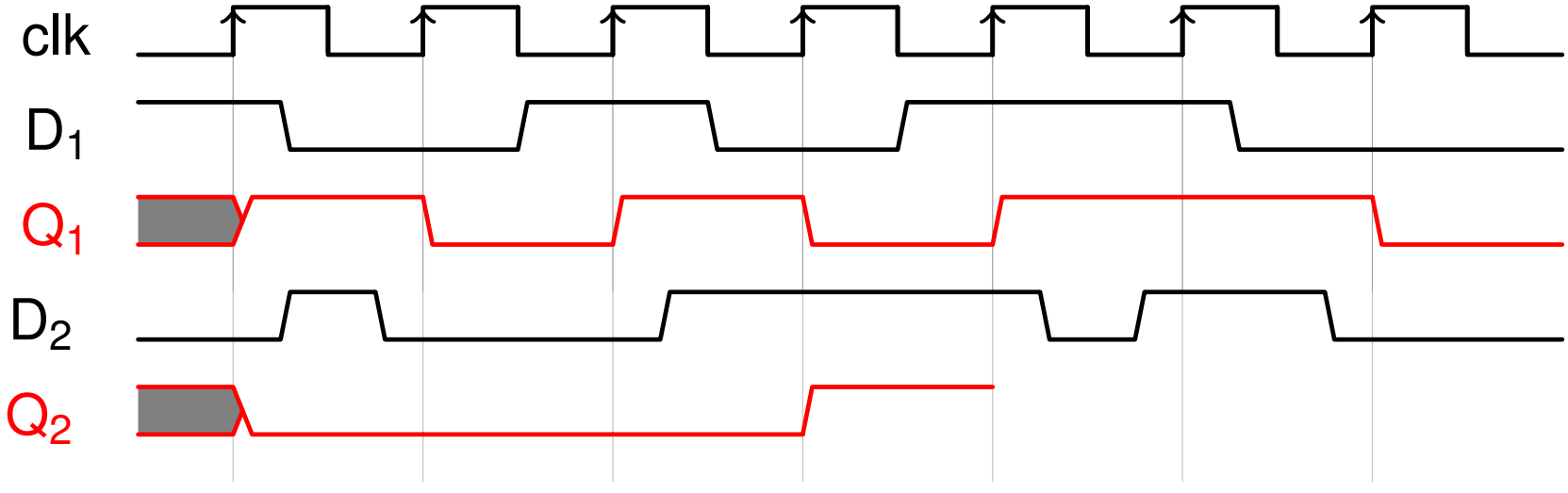
La bascule D

Exemple



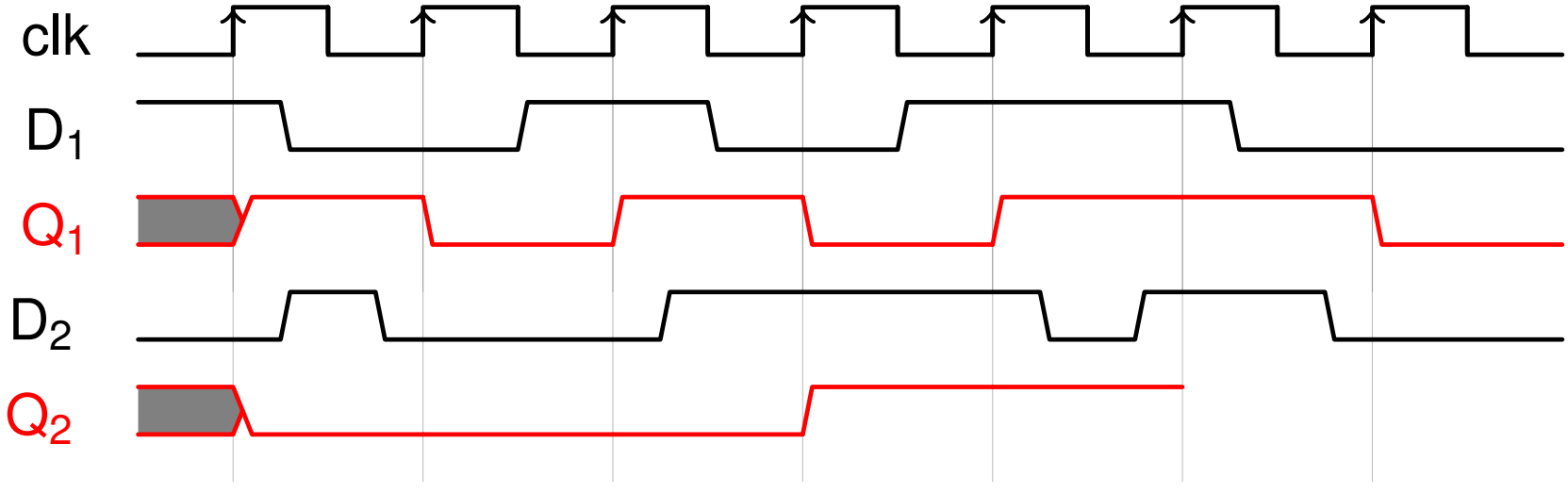
La bascule D

Exemple



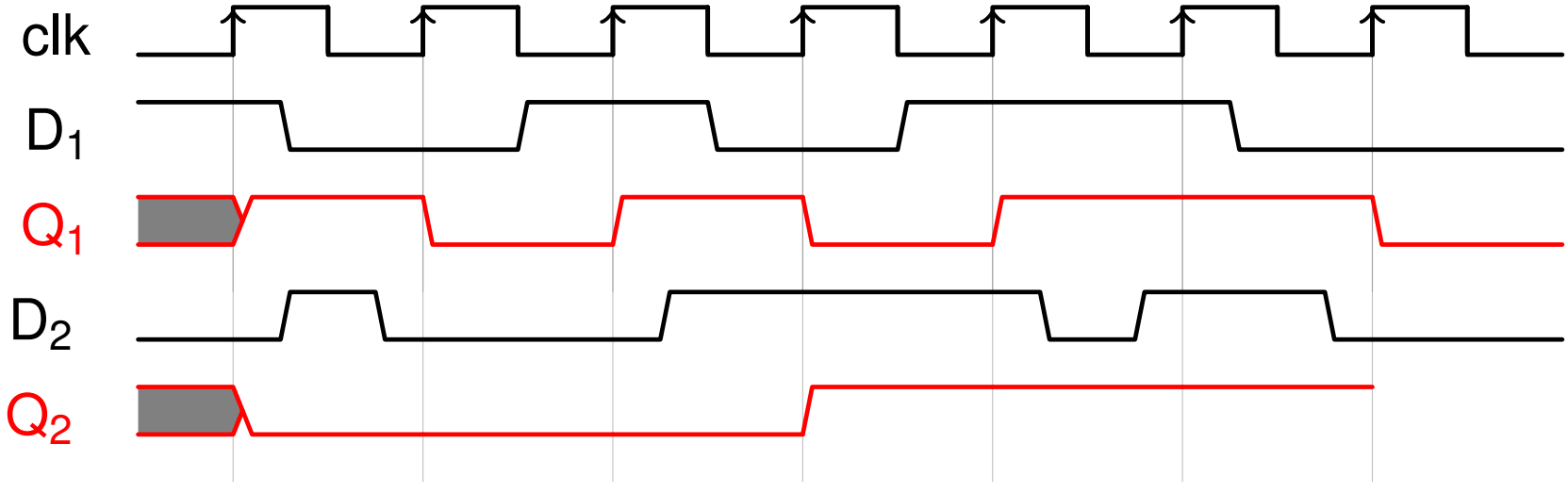
La bascule D

Exemple



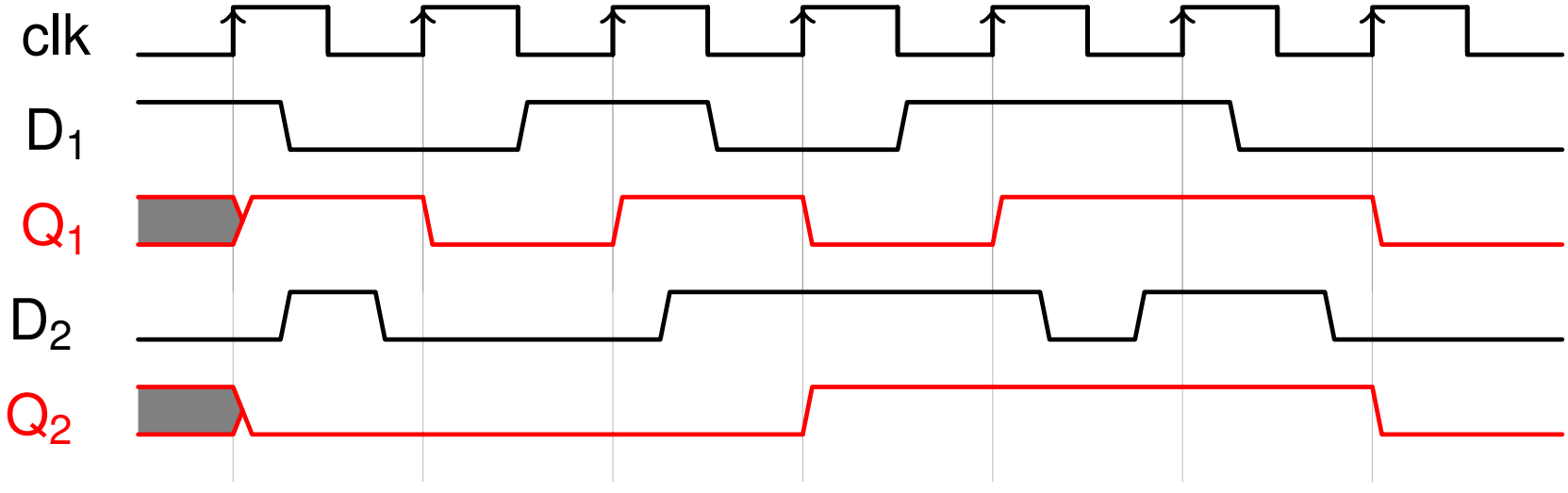
La bascule D

Exemple



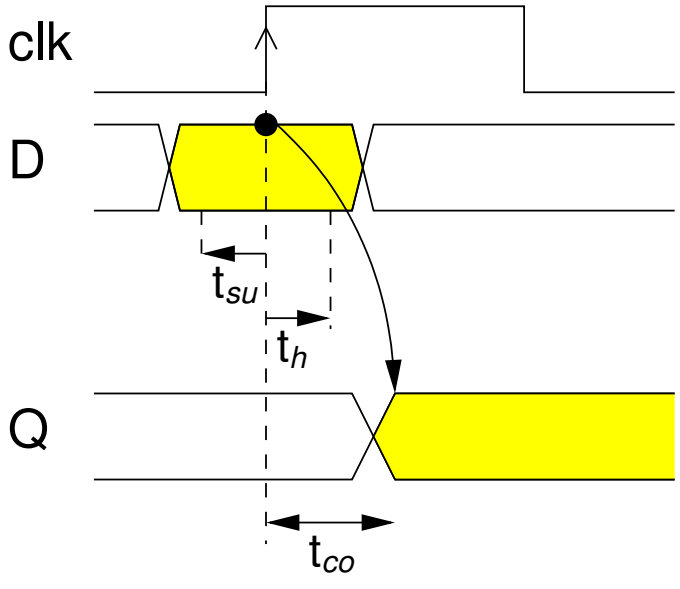
La bascule D

Exemple



La bascule D

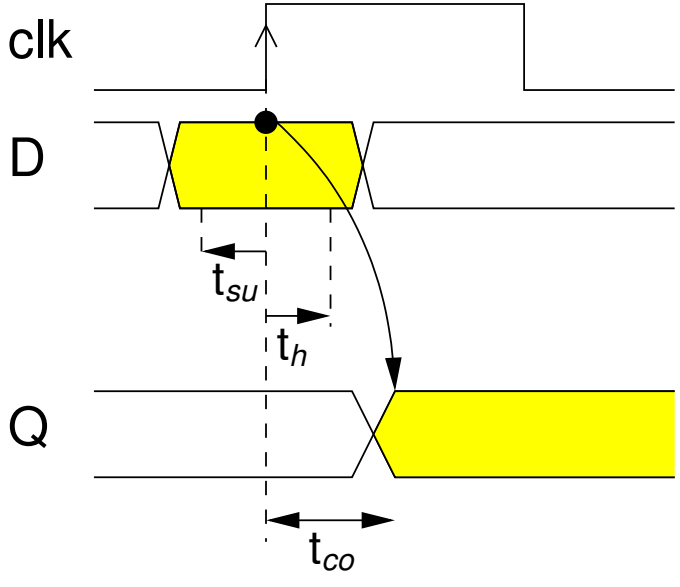
Contraintes temporelles



- 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*)
- Si ces temps ne sont pas respectés, le fonctionnement de la bascule n'est plus garanti
- La copie de l'entrée sur la sortie se fait avec un retard de t_{co} (*clock to output*)

La bascule D

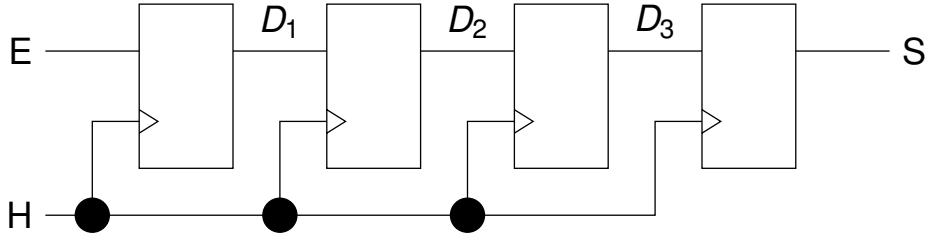
Contraintes temporelles

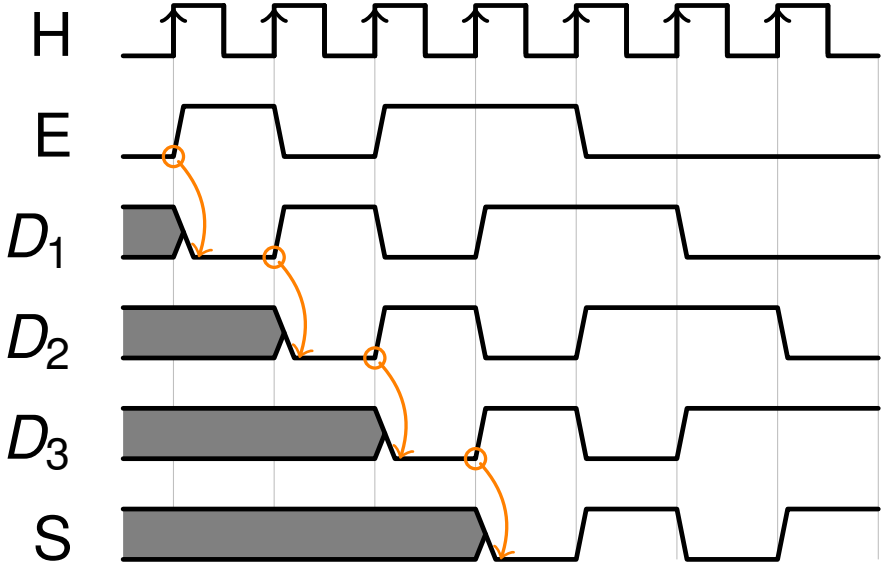


- t_{su} : temps de pré–positionnement
- t_h : temps de maintien
- t_{co} : temps de propagation

Exemples d'applications

Registre à décalage





Plan

La bascule D

La logique séquentielle synchrone

Applications

En SystemVerilog

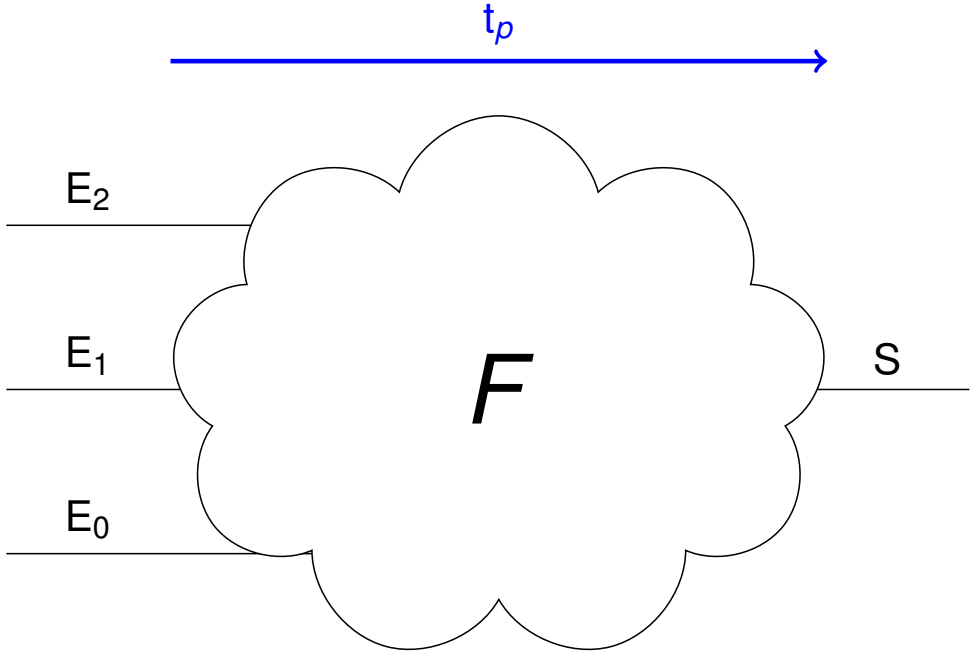
La logique combinatoire

Rappels

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

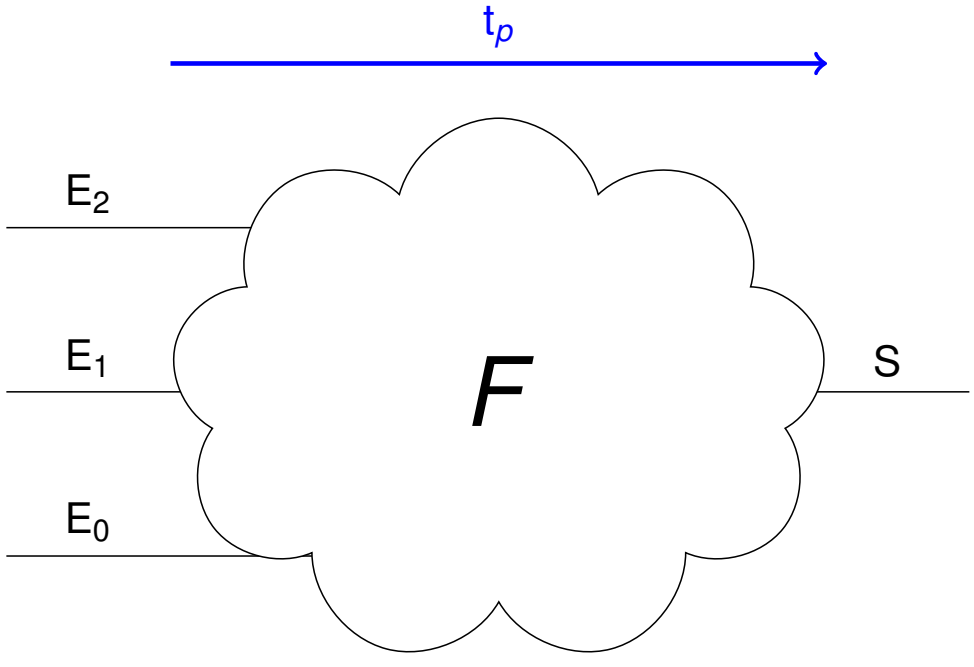
La logique séquentielle

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



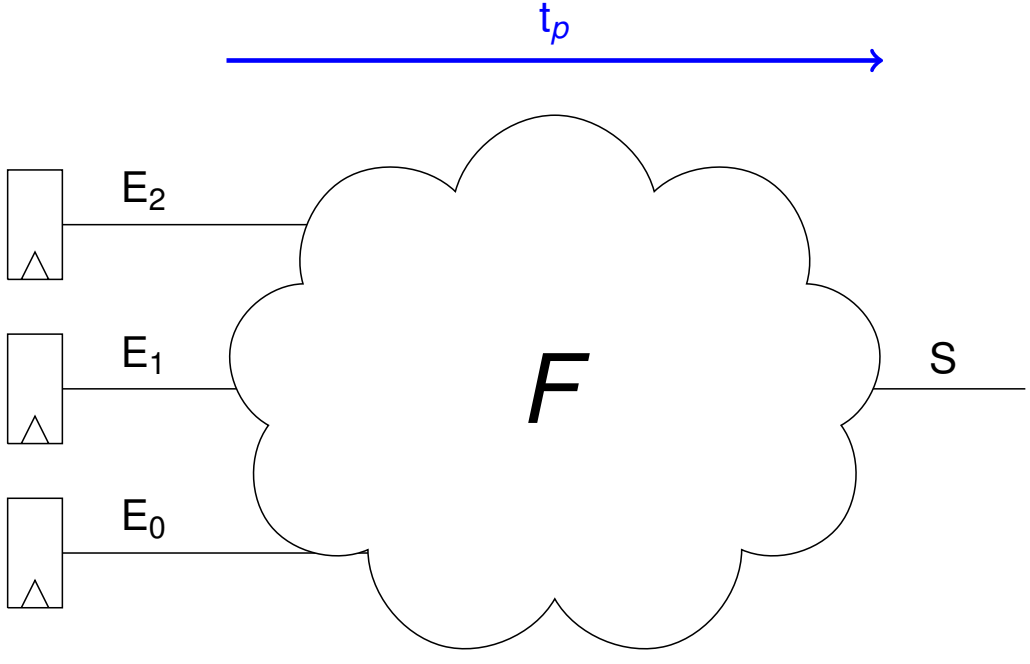
La logique séquentielle

- On maintient les entrées stables au moins pendant t_p



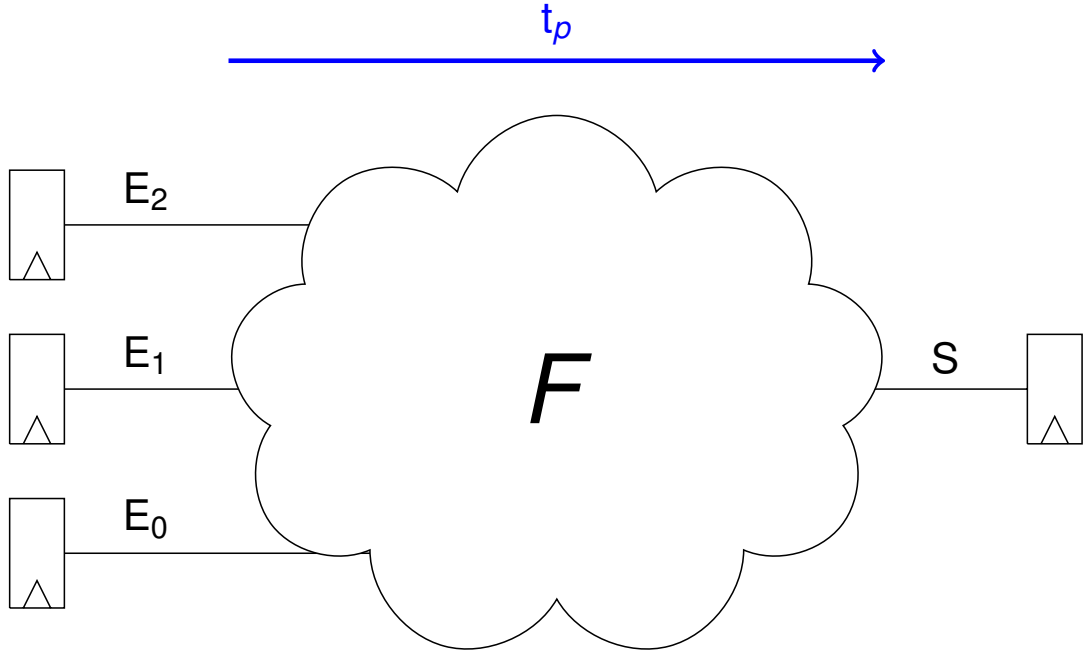
La logique séquentielle

- On maintient les entrées stables au moins pendant t_p
- En mémorisant les entrées



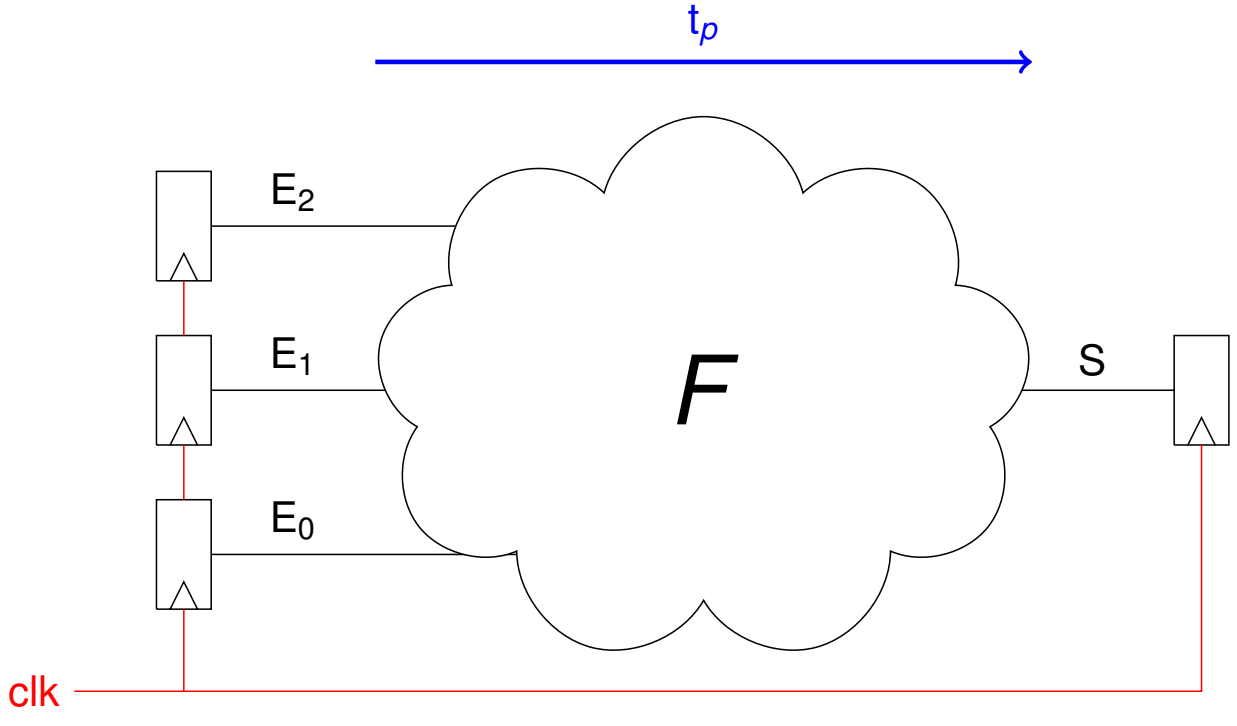
La logique séquentielle

- Quand le résultat est prêt (après au moins t_p) :
  - La sortie peut être mémorisée (pour être utilisée ailleurs)
  - On peut au même instant modifier les entrées

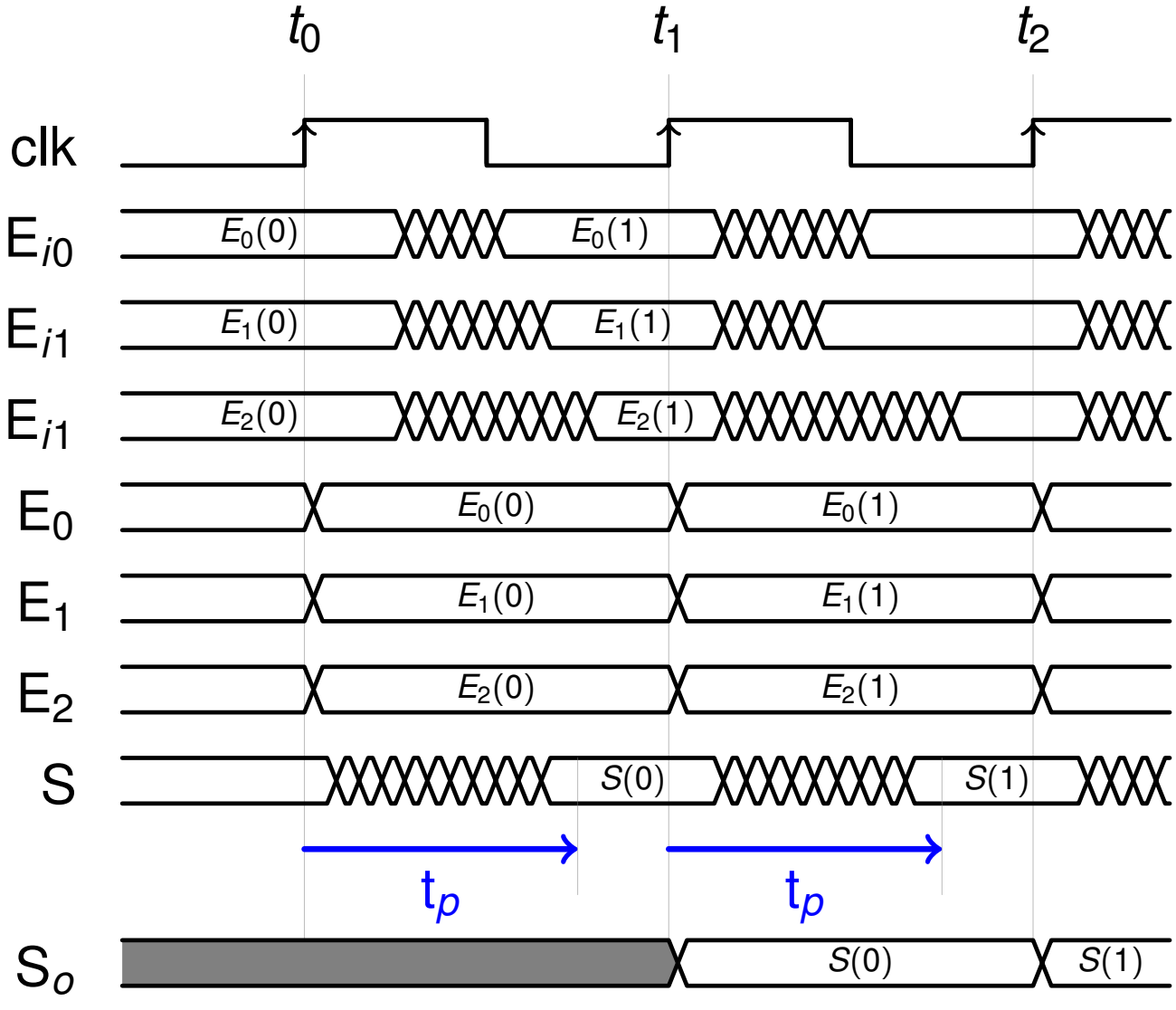


La logique séquentielle

- On utilise donc le même signal de synchronisation
  - La même horloge



La logique séquentielle



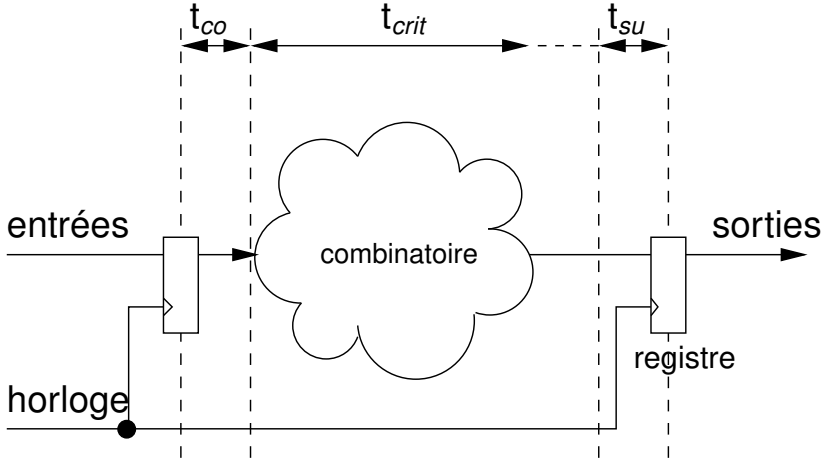
La logique séquentielle synchrone

Règles

- 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

La logique séquentielle

Fréquence de fonctionnement

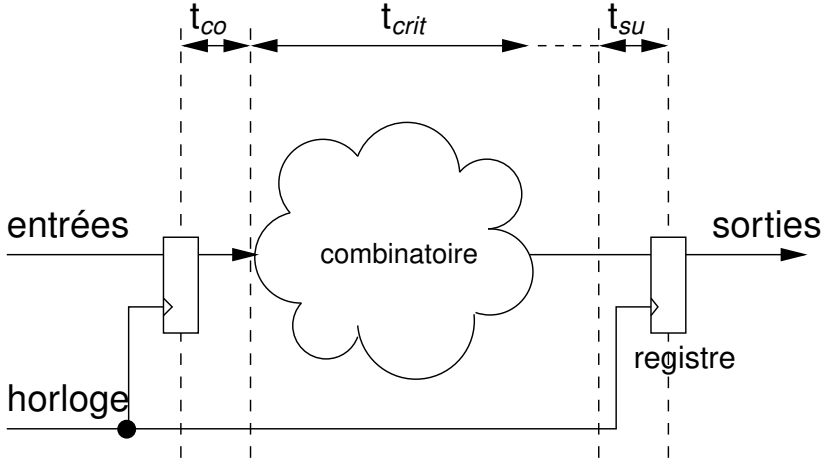


$$T_{clk} > t_{co} + t_{crit} + t_{su}$$

- t_{crit} est le temps du chemin critique c'est-à-dire le temps de propagation du chemin combinatoire le plus long

La logique séquentielle

Fréquence de fonctionnement

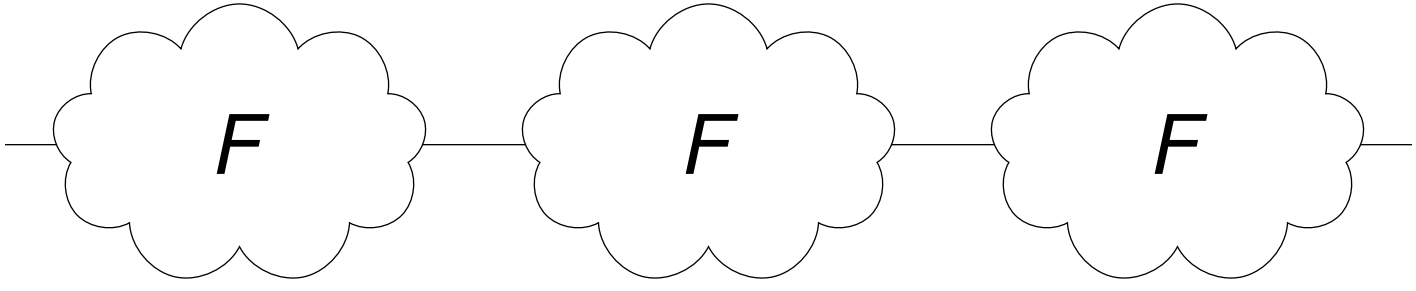


$$F_{max} = \frac{1}{t_{co} + t_{crit} + t_{su}}$$

- t_{crit} est le temps du chemin critique c'est-à-dire le temps de propagation du chemin combinatoire le plus long

Séquencer les traitements

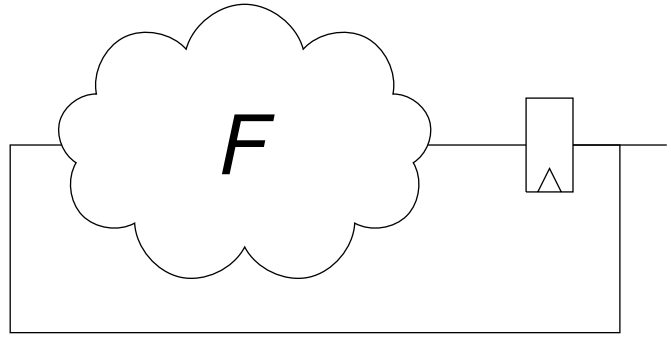
Répéter plusieurs fois le même calcul



- 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

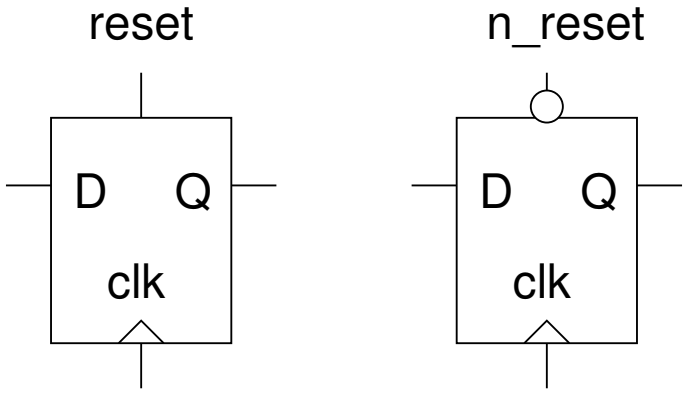
La valeur initiale ?

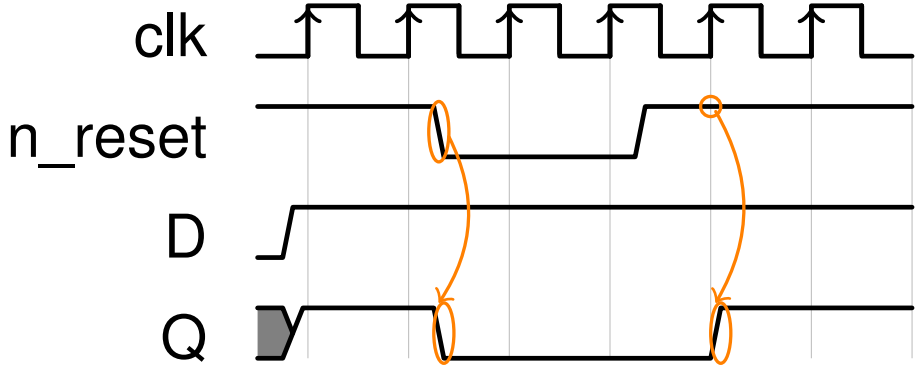
- Un signal extérieur est utilisé pour forcer l'état des bascules
  - À zéro : *reset*
  - À un : *preset*
- Sa valeur peut être prise en compte
  - Uniquement lors d'un front montant d'horloge (*synchrone*)
  - En permanence (*asynchrone*)
- Son effet (remise à zéro) peut se produire
  - Lorsqu'il vaut 1 (*actif à l'état haut*)
  - Lorsqu'il vaut 0 (*actif à l'état bas*)

Séquencer les traitements

Le reset asynchrone

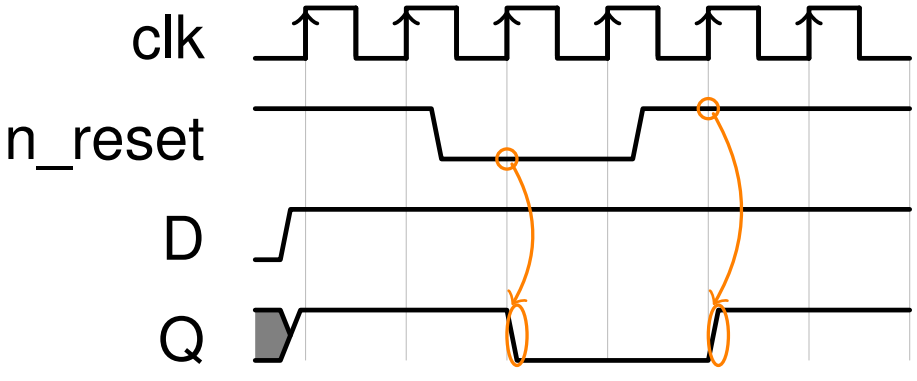
- Un signal connecté directement aux bascules
- Son action est indépendante de l'horloge
- En général global pour tout le circuit





Séquencer les traitements

Le reset synchrone



- Vient de la logique combinatoire qui précède les bascules
- Son action n'est effective qu'au front de l'horloge
- C'est une remise à zéro fonctionnelle

Séquencer les traitements

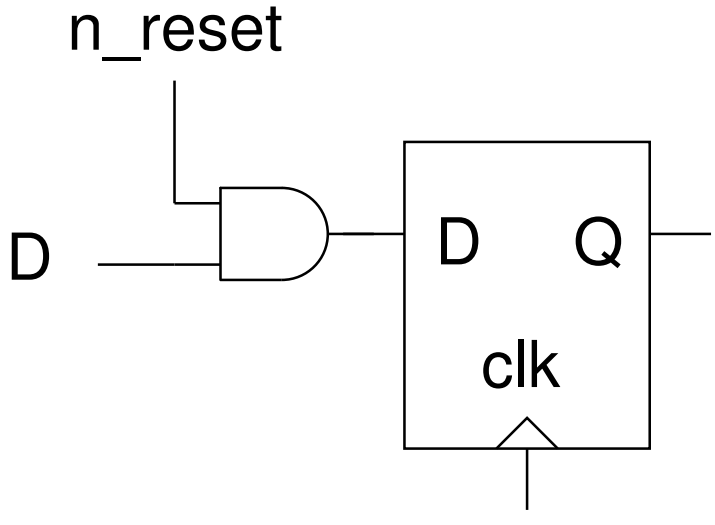
Le reset synchrone

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

Séquencer les traitements

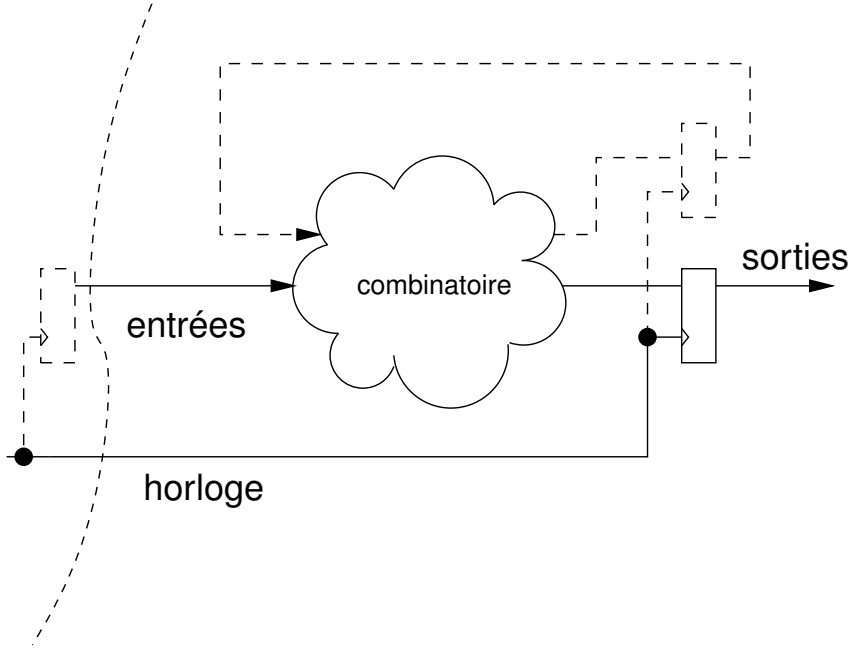
Le reset synchrone

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



Logique séquentielle synchrone

Résumons



- Utilisation de bascules D synchrones pour mémoriser les variables internes et les sorties
- La sortie d'une fonction séquentielle dépend de ses entrées et de son état précédent
- L'état initial doit être forcé par un signal extérieur

Plan

La bascule D

La logique séquentielle synchrone

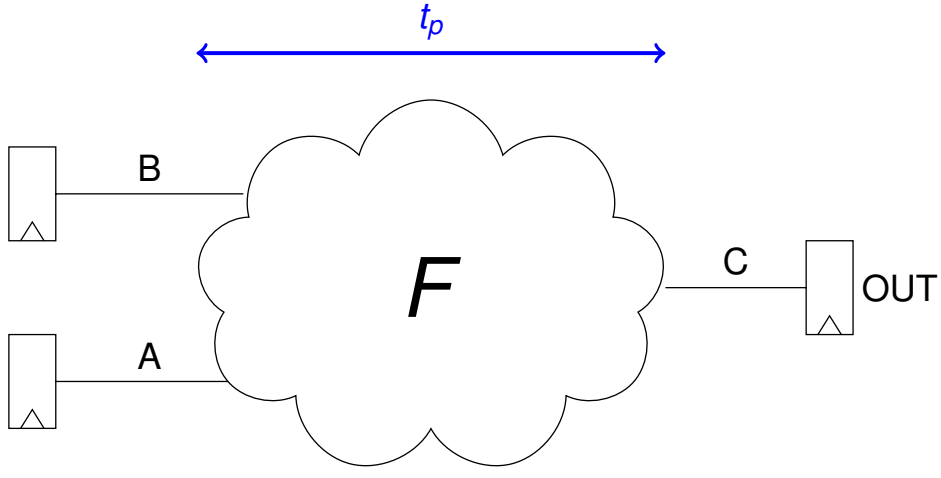
Applications

En SystemVerilog

Exemples d'applications

Le Pipeline

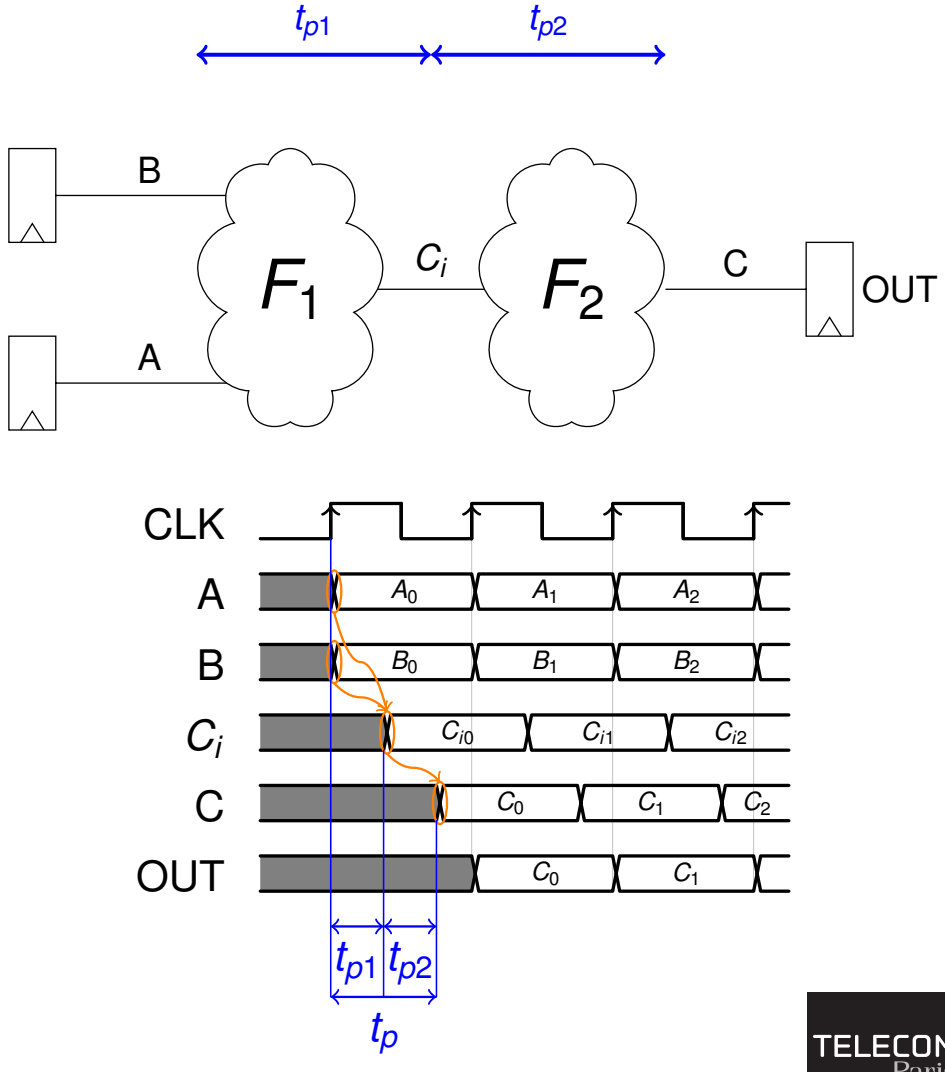
- Une fonction combinatoire F de temps de propagation t_p
- Les entrées sorties peuvent être synchrones tant que $t_p < T_{CLK}$
  - Où T_{CLK} est la période d'horloge



Exemples d'applications

Le Pipeline

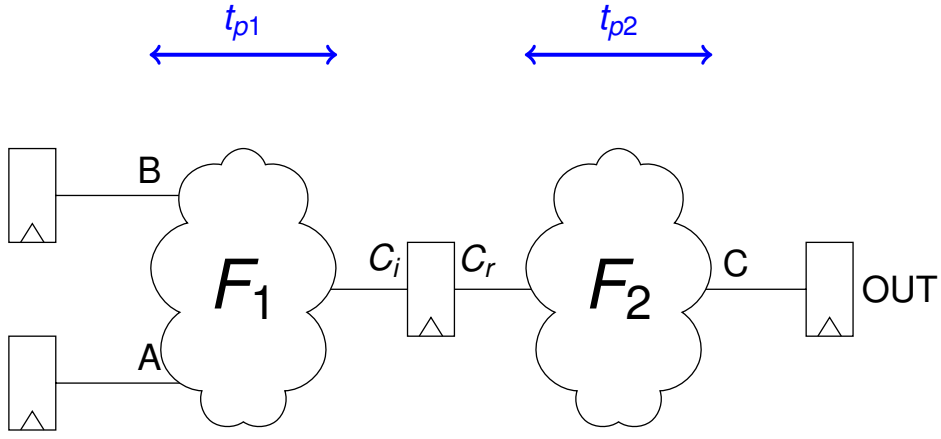
- On décompose F en deux fonctions combinatoires F_1 et F_2 de temps de propagation t_{p1} et t_{p2}
  - On s'arrange pour avoir $t_{p1} < t_p$ et $t_{p2} < t_p$
- Les entrées sorties peuvent être synchrones tant que $t_{p1} + t_{p2} < T_{CLK}$

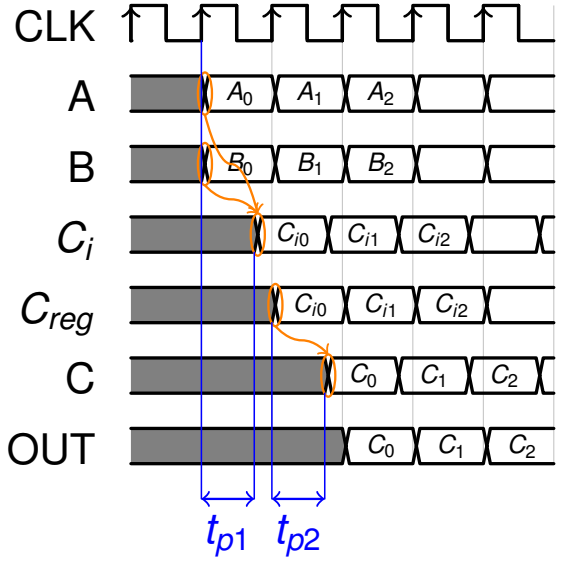


Exemples d'applications

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 (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

Plan

La bascule D

La logique séquentielle synchrone

Applications

En SystemVerilog

La bascule D

- Fonctionnement
  - Toujours, à chaque montant front d'horloge, copier l'entrée sur la sortie
  - Le reste du temps, la sortie ne change pas

La bascule D

- Fonctionnement
  - Toujours, à chaque montant front d'horloge, copier l'entrée sur la sortie
  - Le reste du temps, la sortie ne change pas

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

La bascule D

La remise à zéro asynchrone actif à l'état haut

- Toujours, à chaque front montant d'horloge *ou* si le reset devient actif (front montant du signal)
  - Si le reset est actif (1), forcer la sortie à zéro
  - Sinon, copier l'entrée sur la sortie
- Le reste du temps, la sortie ne change pas

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

La bascule D

La remise à zéro asynchrone actif à l'état bas

- Toujours, à chaque front montant d'horloge *ou* si le reset devient actif (front descendant du signal)
  - Si le reset est actif (0), forcer la sortie à zéro
  - Sinon, copier l'entrée sur la sortie
- Le reste du temps, la sortie ne change pas

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

La bascule D

La remise à zéro synchrone actif à l'état haut

- Toujours, à chaque front montant d'horloge
  - Si le reset est actif (1), forcer la sortie à zéro
  - Sinon, copier l'entrée sur la sortie
- Le reste du temps, la sortie ne change pas

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

La bascule D

La remise à zéro synchrone actif à l'état bas

- Toujours, à chaque front montant d'horloge
  - Si le reset est actif (0), forcer la sortie à zéro
  - Sinon, copier l'entrée sur la sortie
- Le reste du temps, la sortie ne change pas

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

Un registre de 4 bits

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

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

Un compteur module 16

```
logic[3:0] q;


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

- Remarque : on ne connait pas l'état initial
  - Il faudrait ajouter un signal de reset (voir exemples précédents)

Un registre à décalage

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 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