Systèmes numériques : de l'algorithme aux circuits

Logique Séquentielle

Hadrien Barral (prenom.nom@ens.psl.eu)

Version : 1.0

Circuit combinatoire

Rappel

Circuit combinatoire

- La sortie ne dépend que de l'état actuel des entrées.
- Utilisé pour construire e.g. des opérateurs logiques et arithmétiques.

$$\forall t, s(t) = F(e_0(t), e_1(t), ..., e_n(t))$$

Mais c'est un peu limité...

Circuit combinatoire vs circuit séquentiel

Rappel

Fonctions séquentielles

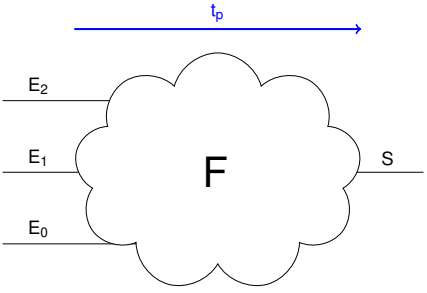
La sortie dépend de l'état actuel des entrées et de leur passé.

$$\begin{aligned} s(t) = F(&e_0(t), e_1(t), \dots, e_n(t),\\ &e_0(t-1), e_1(t-1), \dots, e_n(t-1),\\ &\dots,\\ &e_0(t-\dots), e_1(t-\dots), \dots, e_n(t-\dots)) \end{aligned}$$

- Comment enchaîner plusieurs calculs ?
- Comment stocker l'état interne ?

Comment enchaîner plusieurs calculs

- Le temps de propagation des portes logiques (t_p) est strictement supérieur à zéro.
- Durant l'évaluation de F :
  - La (les) sortie(s) ne sont pas valides
  - Les entrées doivent rester stables, i.e. on ne peut les modifier

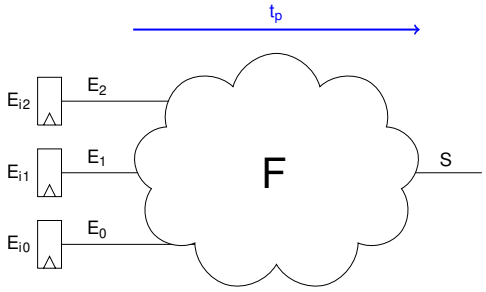


Comment enchaîner plusieurs calculs

Solution pour les entrées

Deux solution pour les entrées :

- On maintient les entrées stables pendant au moins t_p.
- On échantillonne les entrées.

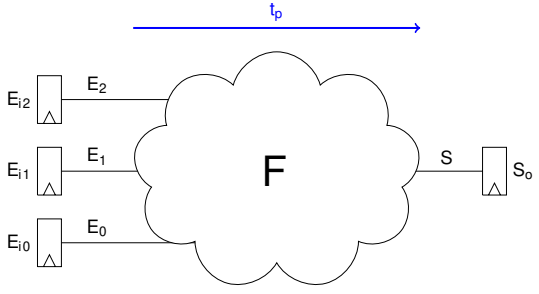


Comment enchaîner plusieurs calculs

Solution pour les sorties

De manière similaire, on peut échantillonner les sorties (qui serviront d'entrées pour le bloc suivant).
Ainsi, dès que le résultat est stable :

- On échantillonne les sorties.
- On peut dès lors re-échantillonner les entrées pour pour faire le prochain calcul.

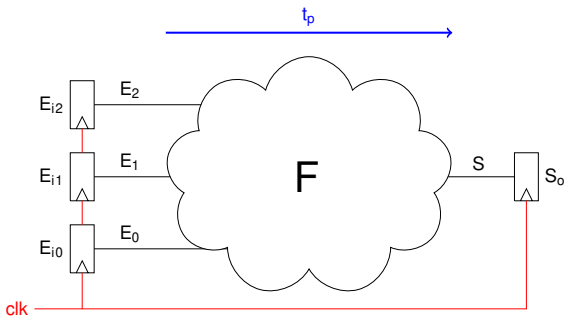


Comment enchaîner plusieurs calculs

L'horloge

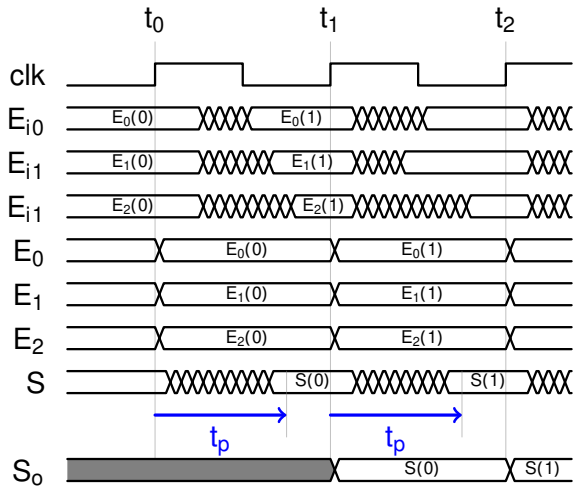
On peut utiliser le même signal de synchronisation pour les entrées et les sorties :

- i.e. la même horloge (`clk`)



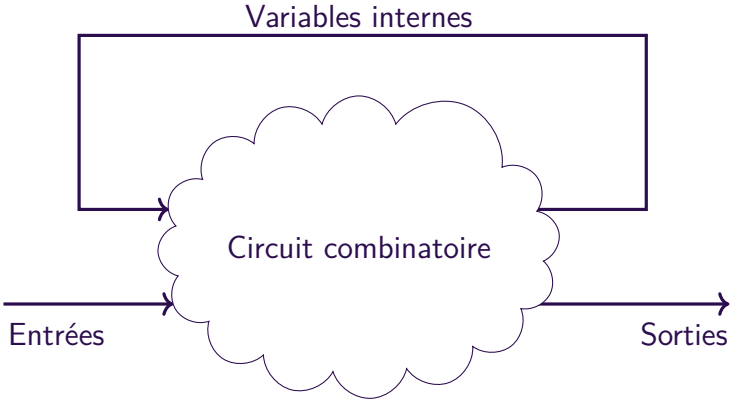
Comment enchaîner plusieurs calculs

Exemple



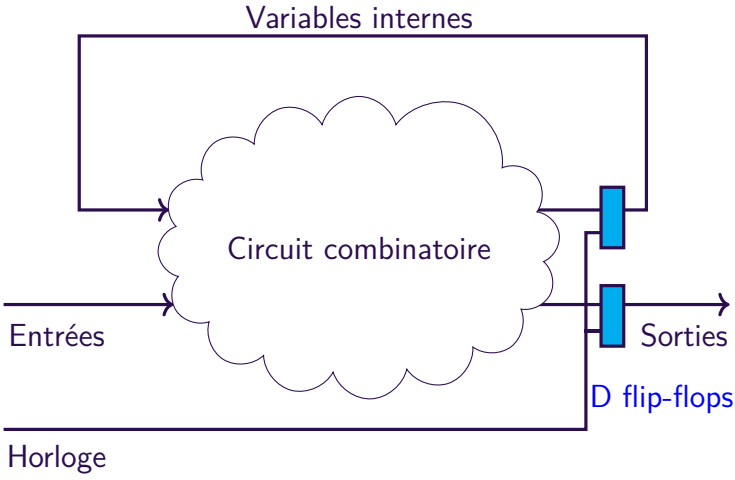
Circuit séquentiel

Les variables internes



Circuit séquentiel

Avec horloge (circuit synchrone)



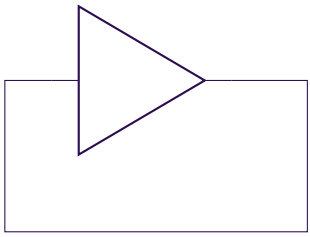
Circuit séquentiel

Types

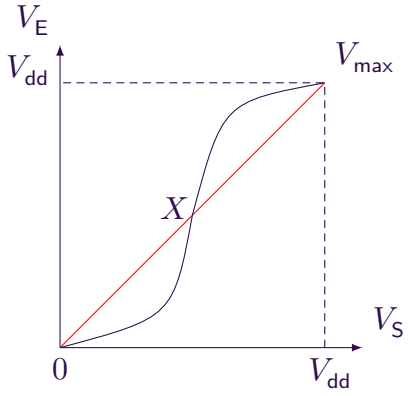
- Circuit monophasé (synchrone, une seule horloge)
- Circuit polyphasé (plusieurs horloges, e.g. microprocesseur)
- Circuit asynchrone

Mémorisation

Via amplificateur



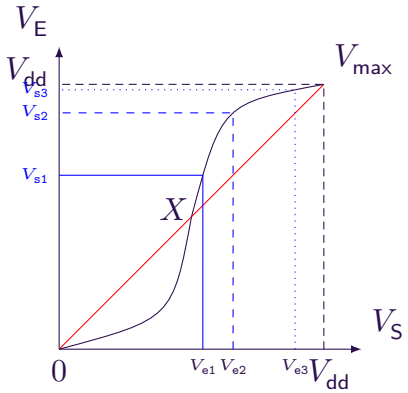
Bit de mémoire basé sur un amplificateur rebouclé



Fonction de transfert de l'amplificateur

Mémorisation

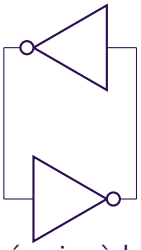
Convergence



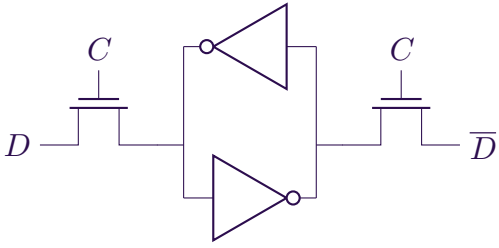
Convergence vers un état stable (sauf si on part de X).

Mémorisation

NOT-NOT



Un bit de mémoire à base de deux NOT.



Cellule SRAM (static RAM)

Problème : comment éviter le court-circuit ?

Latches à portes

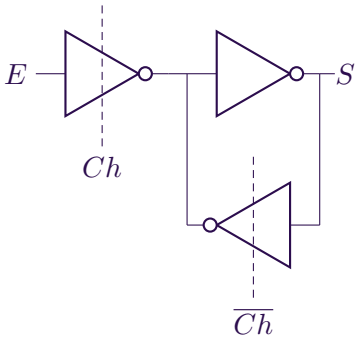


Schéma logique

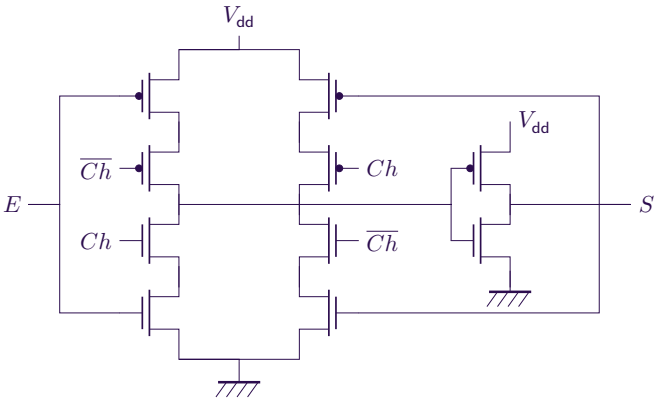


Schéma électrique

Latches à résistances

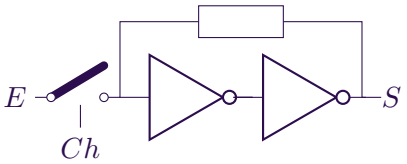


Schéma logique

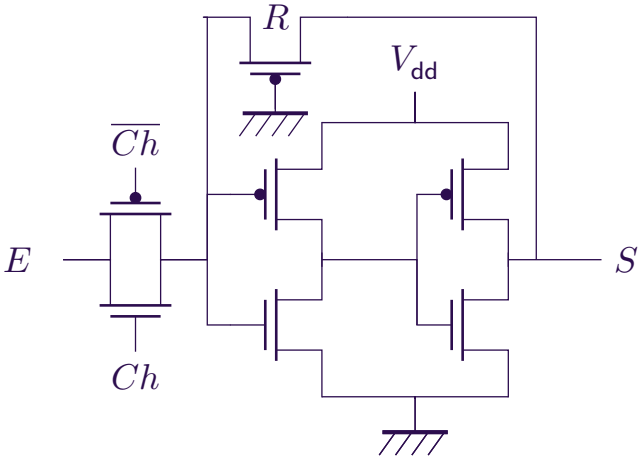


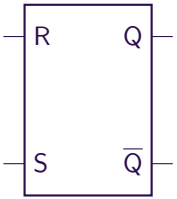
Schéma électrique

Flip-flops v. latches

- Latch (verrou) : les sorties dépendent uniquement du niveau logique des entrées
- Flip-flop (bascule) : les sorties sont mises à jour sur des fronts montants ou descendants
- ~~Crocs (crocs)~~ : ~~les sorties sont d'une mode douteuse~~

SR latch (verrou RS)

- La plus simple du répertoire
- `R` pour *reset* ; `S` pour *set* ;



S	R	Q	Remarque
0	0	Q	Mémoire
0	1	0	Reset
1	0	1	Set
1	1	-	État invalide

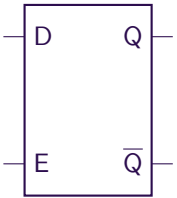
Exercice

Comment faire un SR latch avec 2 portes `NOR` ? Et 2 portes `NAND` ?

(Pour les exercices de cette section, on utilisera directement des portes logiques, et non des transistors.)

D latch (verrou D)

- une entrée de contrôle : `E`
- une entrée de données : `D`



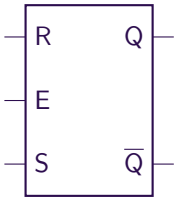
E	D	Q	Remarque
0	-	Q	Mémoire
1	d	d	Écriture

Exercice

Comment faire un D latch en partant d'un SR latch ? Monter qu'on peut faire mieux avec le SR latch à base de `NAND`.

Gated SR latch (verrou RSH/RST)

- La plus simple du répertoire
- R pour *reset* ; S pour *set* ;



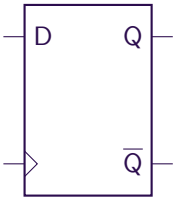
E	S	R	Q	Remarque
0	-	-	Q	Mémoire
1	0	0	Q	Mémoire
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	-	État invalide

Exercice

Comment faire un gated SR latch avec un SR latch ?

D flip-flop (bascule D)

- Similaire au D latch, mais mémorise sur un front montant.
- C'est une bascule, donc le fonctionnement est synchrone à une entrée d'horloge.
  - → E devient donc l'horloge, souvent représentée par un $\triangleright$
  - → On peut maintenant noter Q_0, Q_1, ..., Q_i, ... la valeur au i-ième tick d'horloge



$\triangleright$	D	Q	Remarque
0,1, $\searrow$	-	Q	Mémoire
$\nearrow$	d	d	Écriture

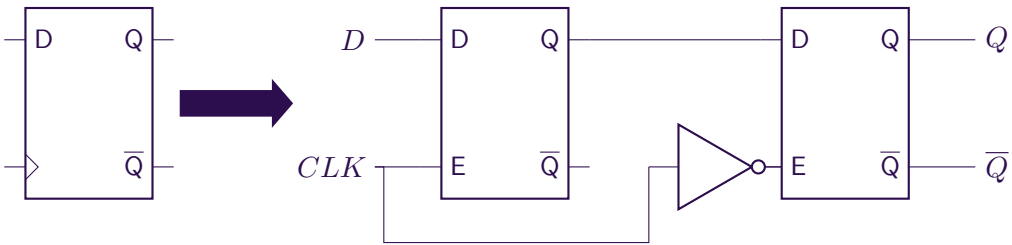
Équation : $Q_n = D_{n-1}$

D flip-flop (bascule D)

Réalisation

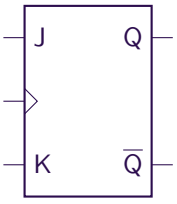
Exercice

Comment faire un D flip-flop avec des D latches ?



JK flip-flop (bascule JK)

- À partir du gated SR latch, on peut former un autre type de bascule.



J	K	Q_n	Remarque
0	0	Q_{n-1}	Mémoire
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}_{n-1}$	Inversion

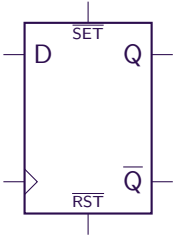
Équation : $Q_n = J\overline{Q}_{n-1} + \overline{K}Q_{n-1}$

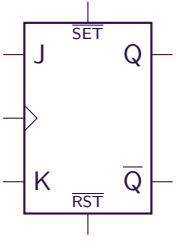
Exercice

Comment faire un JK flip-flop avec un SR latch ?

Flip-flop avec Set/Reset

- Il existe des version des flip-flops avec en supplément un Set/Reset asynchrone.





En pratique

Amusez-vous avec https://www.falstad.com/circuit/circuitjs.html
(`Circuits -> Sequential Logic -> Flip-flops -> …`)

Autres latches/flip-flops

Pour aller plus loin

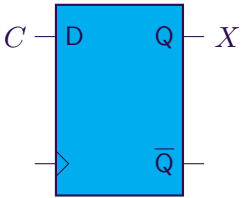
Si ça vous intéresse, il existe aussi :

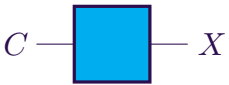
- JK latch
- C-element
- T flip-flop
- Schmitt trigger (attention, c'est de l'électronique)
- ...

Registre

À quoi correspond le `REG` des netlists (cf TP) ?

```
x = REG c
```





Compteurs

Au tableau

Registres à décalage (*shift registers*)

Au tableau

SRAM / DRAM / ROM

Au tableau

Dans les netlists, on vous fournit directement des briques `RAM` et `ROM`. Comment les construire ?

Exercice

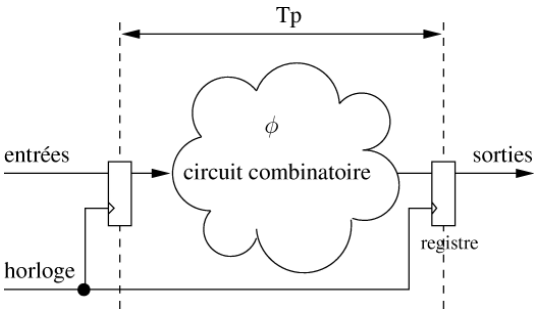
Quelle est la meilleure complexité possible pour le temps d'accès à une mémoire (en fonction de sa capacité) ?

Début de calcul

On veut effectuer un calcul ϕ. Paramètres à prendre en compte :

- f : fréquence d'arrivée (débit) des données à traiter par ϕ (les entrées)
- T_h : période de l'horloge
- T_p : temps de propagation du chemin critique du circuit effectuant ϕ

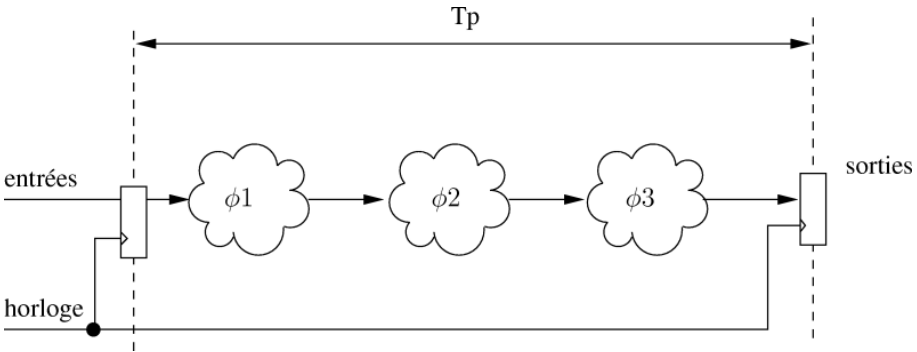
Condition suffisante : $T_p < T_h = 1/f$.



Pipelining

Décomposition

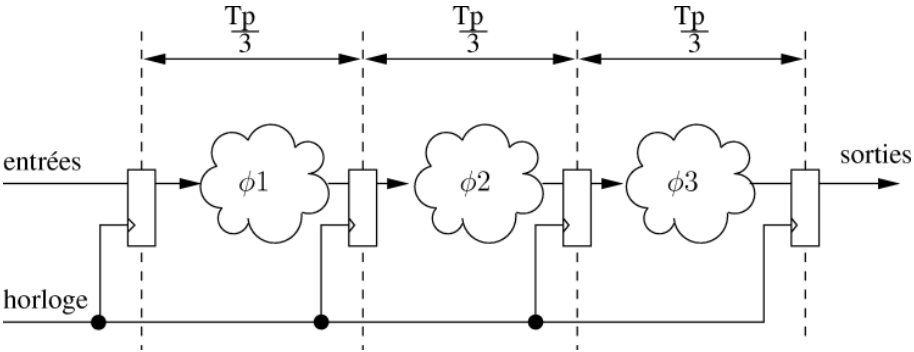
- Que faire si T_p est trop long ?
  1. Décomposer en sous fonctions
  2.



Pipelining

Décomposition

- Que faire si T_p est trop long ?
  1. Décomposer en sous fonctions
  2. Rajouter des registres

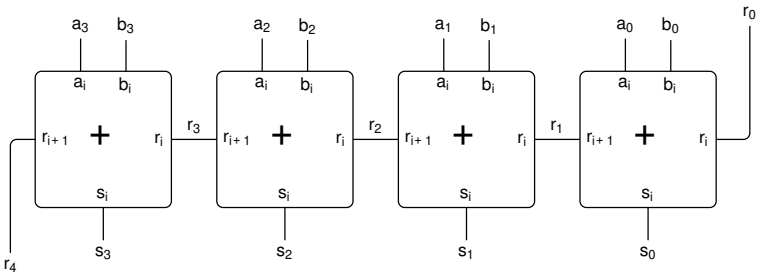


Pipelining

Conséquences & Limite

- Conséquences :
  - On respecte maintenant les contraintes de débit sur les entrées
  - La latence passe de 1 à 3
  - Le retard absolu de ϕ ne change pas forcément
- Limites :
  - Pas forcément simple de décomposer en sous-fonctions équiréparties
    $\rightarrow$ Ainsi, pour N étages de *pipeline*, on gagne en pratique un débit inférieur à N.
  - Les registres sont à prendre en compte dans le calcul du chemin critique

Exercice



Exercice

1. Quelle est la latence de cet additionneur 4-bit (en nombre de full-adders) ?
2. Comment améliorer le débit par *pipelining* ?
3. Quelle est alors la nouvelle latence ?

Codage des *state-machines* (automates)

Au tableau

Retiming

Retiming

- But : optimiser le chemin critique **sans** augmenter la latence.
  → Donc, pas comme le pipelining.
- Voir le papier *Retiming Synchronous Circuitry* de Charles E. Leiserson & James B. Saxe (1983)
  → Les figures ci-après sont tirées du papier.

Retiming : un premier exemple

Corrélateur : spécification

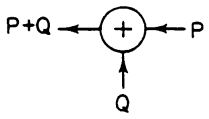
In order to illustrate retiming, consider the problem of designing a digital correlator. The correlator takes a stream of bits $x_0, x_1, x_2, \ldots$ as input and compares it with a fixed-length pattern $a_0, a_1, \ldots, a_k$. After receiving each input x_i $(i \geq k)$, the correlator produces as output the number of matches

$$y_i = \sum_{j=0}^{k} \delta\left(x_{i-j}, a_j\right) \ ,$$

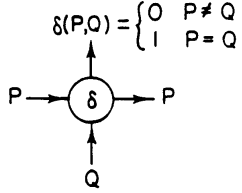
where δ is the comparison function

$$\delta(x, y) = \begin{cases} 1, & \text{if } x = y; \\ 0, & \text{otherwise.} \end{cases}$$

[1]Recall that one eptosecond (esec) equals one one-zillionth of a second.



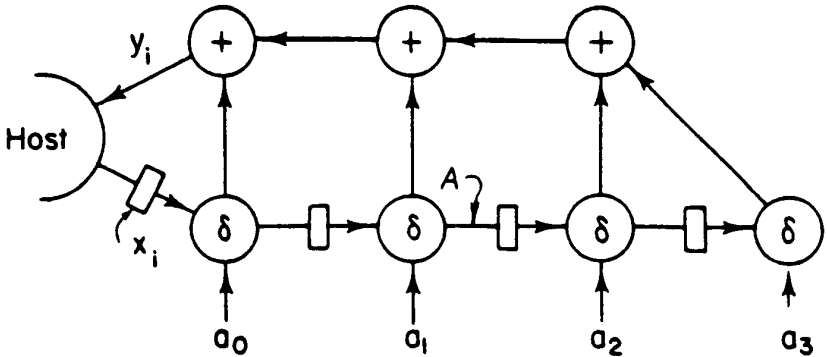
Propagation delay : 7 esec



Propagation delay : 3 esec

Retiming : un premier exemple

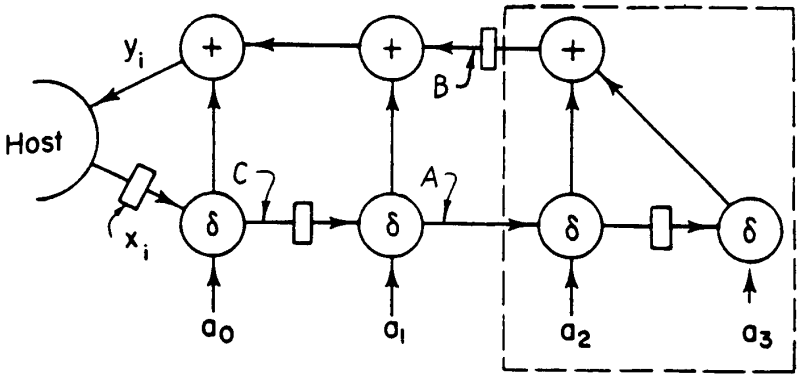
Corrélateur : circuit naïf



Longueur du chemin critique ? Comment faire mieux ?

Retiming : un premier exemple

Corrélateur : un 'meilleur' circuit

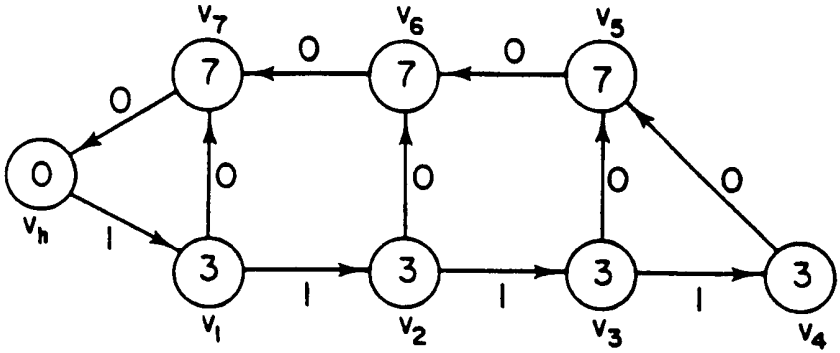


Longueur du chemin critique ? Le circuit est-il équivalent ?

Retiming

Algorithme (1/2)

Comment calculer la fréquence maximale pour un circuit donné ?



Retiming

Algorithme (2/2)

- Définir un *retiming* sur les sommets : $r : V \to \mathbb{Z}$
- Transformer le problème en LP
- Dichotomiser sur la fréquence maximale et utiliser Bellman-Ford pour déterminer si le *retiming* peut être valide

Retiming

Limites

Le retiming rend plus difficile :

- la vérification
- le debug
- la synthèse

Retiming

Et après ?

Pour aller plus loin, cf cours de Chee Wee Liu :
https://twins.ee.nctu.edu.tw/courses/vsp_11_summer/lecture/VSP%20Lecture%20PDF/VSP-lec01-1-pipelining%20&%20retiming.pdf

Autre approches possibles pour améliorer la *vitesse* des calculs :

- pipelining (ORLY ?)
- parallel processing (dual du pipelining)
- clock skewing