HDL et SystemVerilog

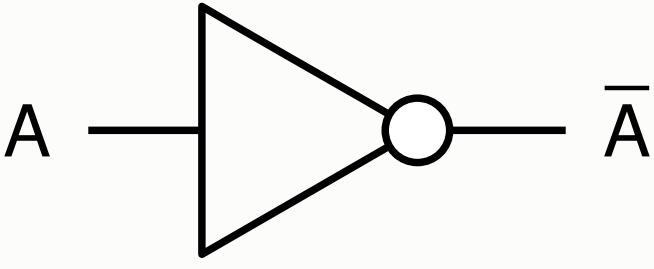
Introduction aux langages de description du matériel

Tarik Graba
tarik.graba@telecom-paristech.fr
Année universitaire 2024/2025

Plan

Représentation de fonctions numériques

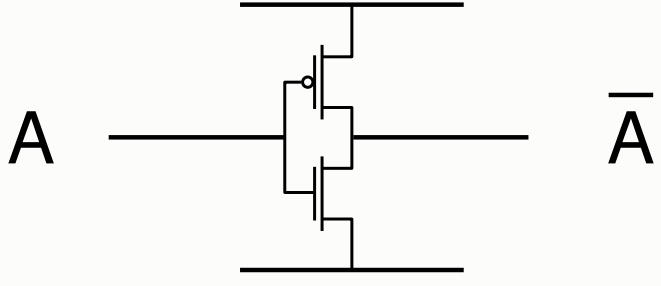
Schéma d'une fonction booléenne



Inverseur

Représentation de fonctions numériques

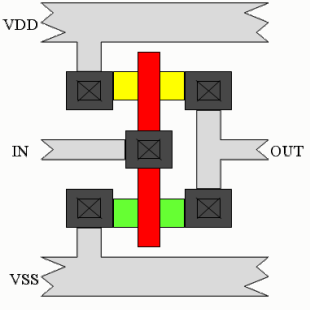
Schéma des transistors



Inverseur CMOS

Représentation de fonctions numériques

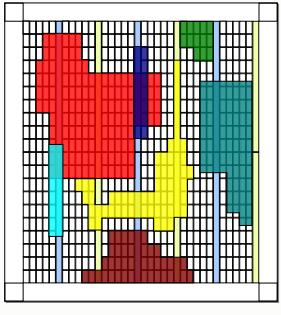
Dessin des couches physiques



Inverseur CMOS intégré

Représentation de fonctions numériques

Et dans le FPGA ?



Programmer des cellules

Représentation de fonctions numériques

Comment représenter ?

- Dessins/schémas ?
  - Pas facile…
- Équations ?
  - Comment représenter la structure physique ?
  - Comment représenter le temps ?
- Autres ?

Quel niveau de représentation ?

- Logique ? Structurel ? Physique ?

Représentation de fonctions numériques

Autres niveaux d'abstraction ?

Augmenter la productivité

- Abstraire
- Réutiliser

Quel niveau de représentation ?

- Algorithme ?
- autres…

HDL

HDL

- **HDL : H**arware **D**escription **L**anguage.
- Langage informatique de description du matériel.

HDL

Ces langages doivent permettre deux choses

- Concevoir/Réaliser
  - Implémenter
  - Fabriquer
- Modéliser/Simuler
  - Tester la fonctionnalité

HDL : Une représentation textuelle

Qui permet :

de représenter la structure

L'inverseur

```
not(nA, A)
```

une porte logique avec une entrée et une sortie

HDL : Une représentation textuelle

Qui permet :

de représenter son comportement

L'inverseur

```
nA = !A
```

son comportement sous forme d'une équation

HDL : Une représentation textuelle

Qui permet :

de représenter son comportement

L'inverseur

```
if(A) nA = 0 else nA = 1
```

son comportement par une séquence (fonctionnellement)

HDL : Les plus

Automatisation

- **EDA : E**lectronic **D**esign **A**utomation.
- Conception électronique automatisée.
- Utilisation de l'outil informatique pour générer les autres représentations.

Abstraction et productivité

- S'abstraire de la cible technologique.
- Utiliser des représentations de plus haut niveau.
  - Ne pas se limiter à des équations logiques.

Plan

Le «RTL»

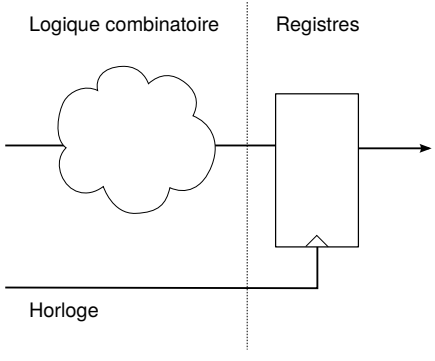
Représentation des états en logique synchrone

RTL

- **RTL : R**egister **T**ransfer **L**evel.
- Le niveau « transfert entre registres ».

Le «RTL»

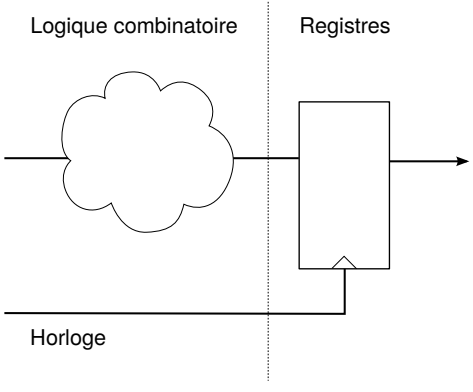
Représentation des états en logique synchrone



Un registre (ou une bascule) est un élément mémorisant dont le changement d'état est déclenché par un signal d'horloge.

Le «RTL»

Représentation des états en logique synchrone



- Une horloge explicite !
- Il faut décrire ce qui ce passe à chaque coup d'horloge.

Les algorithmes doivent être transformés pour exprimer ce qui se passe à chaque cycle d'horloge.

Le «RTL»

Représentation des états en logique synchrone

1- L'algorithme

```
int i;

for (i = 0; i< 10; i++)
{
   ...
}

...
// puis on utlise i
```

- pas de notion de temps ou d'horloge

Le «RTL»

Représentation des états en logique synchrone

2- L'algorithme + hypothèses d'architecture

```
int i; // <- valeur signée sur 32 bits

for (i = 0; i< 10; i++)   // <--- une itération par cycle?
{
   ...
}

...
// puis on utlise i // <- et ainsi de suite
```

- Faire des hypothèses sur l'architecture

3- Attribuer des ressources

- Un registre pour stocker `i` .
  - Qui change à chaque front d'horloge.
  - Suffisamment grand (32 ou 4 bits ?)
- Un additionneur (ou incrémenteur).
- Un comparateur.

3- Description RTL

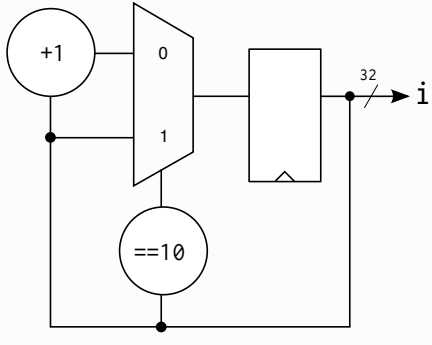
```
i[32]
next_i[32]
sum[33]
cond[1]

// logique combinatoire
cond = i<10
sum  = i + 1
next_i = cond? sum[31:0] : i

// registres
@(clk) i = next_i
```

- Décrire la logique combinatoire.
- Décrire l'évolution des registres.

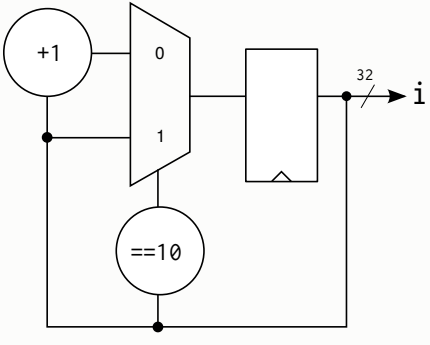
4- Synthèse automatique

```
i[32]
next_i[32]
sum[33]
cond[1]

// logique combinatoire
cond = i<10
sum  = i + 1
next_i = cond? sum[31:0] : i

// registres
@(clk) i = next_i
```

→



4`- Synthèse automatique (netlist)



→

```
// représentation structurelle
inc   inc_i( i,is );
mux   mux_i( i_next,i,is );
cmp10 cmp_i( cond,i );
reg32 reg_i( i,i_next );
```

Puis descendre vers des primitives technologiques

Pourquoi le niveau RTL ?

- Suffisamment haut niveau pour représenter «simplement» tout système numérique synchrone.
  - chemin de données
  - contrôle, MAE
  - …
- Abstraction de la technologie.
- Il existe depuis longtemps des outils automatiques fiables pour la synthèse
  - Synthèse logique RTL -> Structurel
  - Synthèse physique Structurel -> Technologique

Le «RTL»

Simplifier la description du comportement

Permettre d'utiliser des constructions de plus haut niveau dans une description RTL.

```
i[32]
next_i[32]
sum[33]
cond[1]

// logique combinatoire
cond = i<10
sum  = i + 1
next_i = cond? sum[31:0] : i

// registres
@(clk) i = next_i
```

→

```
i[32]
next_i[32]

// logique combinatoire
@_ if(i<10) next_i = i+1
   else     next_i = i

// registres
@(clk) i = next_i
```

ou même :

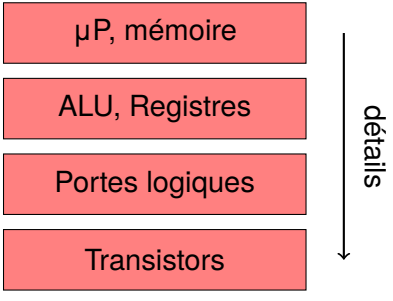
```
i[32]

// logique synchrone
@(clk) if(i<10) i = i+1
```

Note : Notion de processus et d'affectations concurrente plus tard dans le cours.

Plan

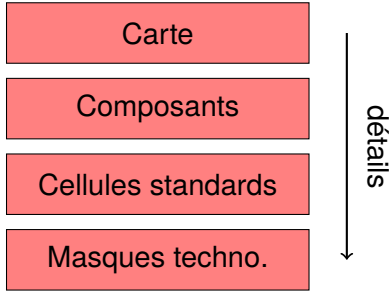
Description structurelle

- Des composants
- Des connexions
- On parle de «netlist»
- …
- Conception par assemblage



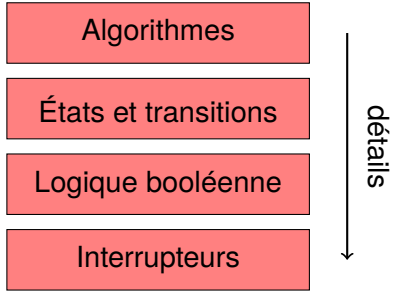
Description physique

- Les matériaux
- Les dimensions
- …
- Nécessaires pour la fabrication



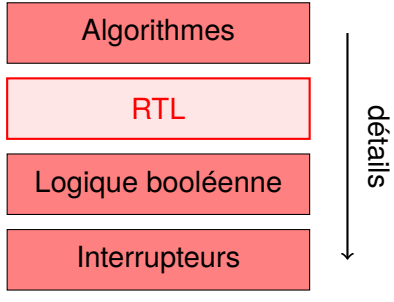
Description comportementale

- Décrire la fonction réalisée.
- …
- C'est ici qu'on retrouve le RTL



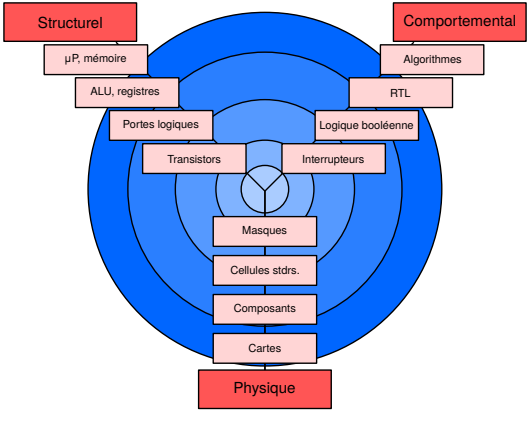
Description comportementale

- Décrire la fonction réalisée.
- …
- C'est ici qu'on retrouve le RTL



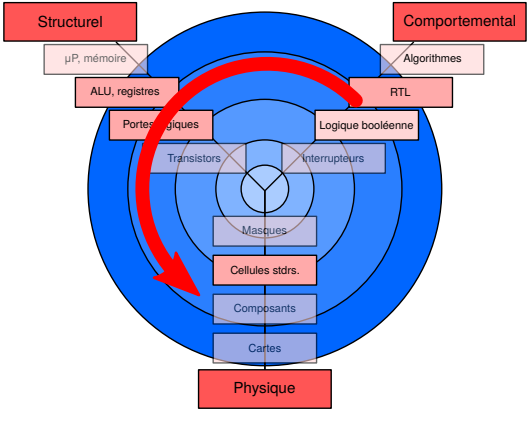
Niveaux de description

Équivalence mais différentes finalités



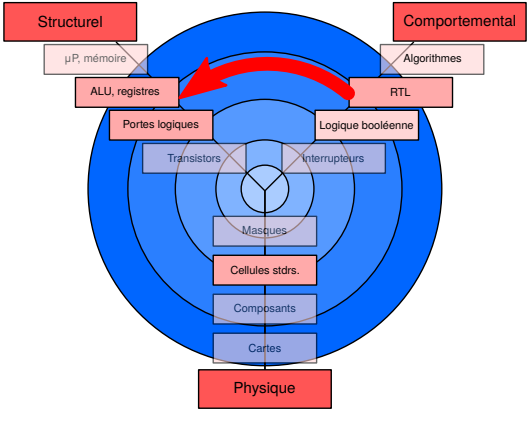
Niveaux de description

Synthèse automatique



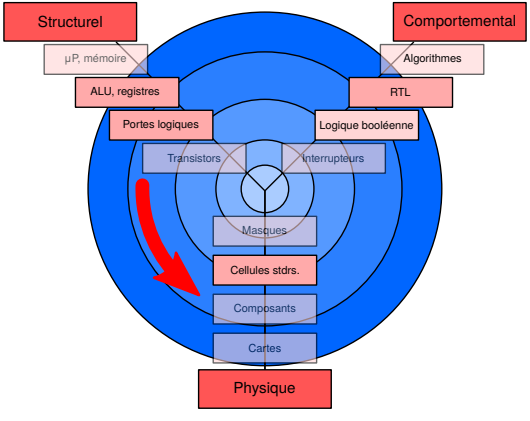
Passage d'une représentation comportementale à la fabrication/programmation

Niveaux de description
Synthèse automatique



Une étape de synthèse logique générique

Niveaux de description
Synthèse automatique



Une étape de synthèse physique spécifique

Niveaux de description
et les FPGAs ?

- La structure des FPGA est figée (matrice de cellules et interconnexion)
- Les cellules technologiques des FPGA sont programmable
- L'étape de synthèse physique peut être vue comme :
  - Structurel -> Cellules du FPGA
  - Placement des cellules dans la matrice et choix des routes
  - Génération du fichier de programmation (*bitstream*)

Plan

Simuler le comportement du matériel

Simuler efficacement une représentation RTL

- D'un côté,
  - le matériel est intrinsèquement parallèle,
  - tous les composants d'un circuit sont actifs et fonctionnent en même temps,
  - le nombre d'éléments peut être nombreux.
- D'un autre côté,
  - les simulations sont exécutées sur des machines séquentielles (le processeur de votre PC) ;
  - il nous est plus simple de décrire des séquences.

Simuler le parallélisme

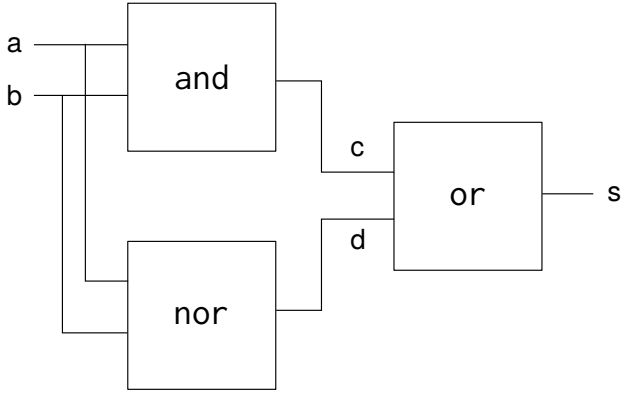
- Comment simuler efficacement du matériel ?
- On ne veut pas simuler ce qui se passe physiquement ! Uniquement le comportement (logique/arithmétique) à un niveau RTL.

Simuler le parallélisme

Exemple : de la logique combinatoire

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x



Simuler le parallélisme
Pour de la logique combinatoire

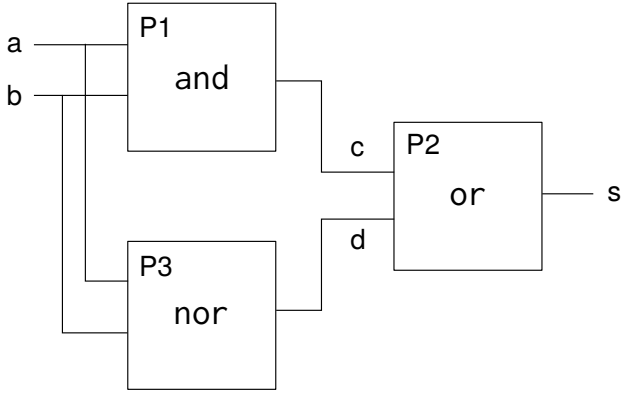
- Une fonction par bloc combinatoire
  - qu'on appellera **processus**
  - les instructions dans un processus sont exécutées séquentiellement
- On agit sur des variables globales vues par tous les processus
- On exécute les processus dans un ordre quelconque
- On re-exécuter tant qu'il y a des changements
- Ne fonctionne plus s'il y a une boucle

Simuler le parallélisme

Exemple : de la logique combinatoire

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x

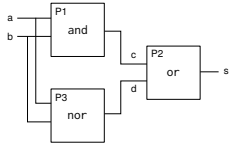


Simuler le parallélisme

Pour de la logique combinatoire

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x



	0				1				2				3
		P1	P2	P3		P1	P2	P3		P1	P2	P3	
c	x	0	–	–	0	0	–	–	0	0	–	–	0
d	x	–	–	0	0	–	–	0	0	–	–	0	0
s	x	–	x	–	x	–	0	–	0	–	0	–	0

Simuler le parallélisme
Le temps symbolique

- Chaque cycle d'exécution du simulateur est appelé **temps symbolique** (ou delta Δ).
- Ça ne représene pas un temps «physique' !
- Tant qu'un processus a besoin d'être exécuté, on est dans le même delta.

Simuler le parallélisme

Le temps symbolique

					fin				fin				fin
	0ns	0ns			0ns	0ns			0ns	0ns			0ns
	init	Δ_1	$\rightarrow$		Δ_1	Δ_2	$\rightarrow$		Δ_2	Δ_3	$\rightarrow$		Δ_3
		P1	P2	P3		P1	P2	P3		P1	P2	P3	
c	x	0	–	–	0	0	–	–	0	0	–	–	0
d	x	–	–	0	0	–	–	0	0	–	–	0	0
s	x	–	x	–	x	–	0	–	0	–	0	–	0

Simulation événementielle
évènements et liste de sensibilité

Pour accélérer la simulation, il ne faut exécuter que les processus dont les entrées ont changé.

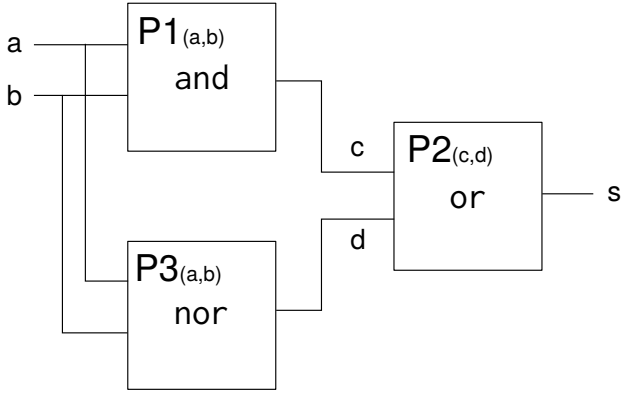
On ajoute alors deux notions :

- Les évènements sur les entrées
  - Si une entrée change
- La liste de sensibilité
  - La liste des évènements qui nécessitent l'exécution d'un processus

Simulation événementielle
évènements et liste de sensibilité

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x

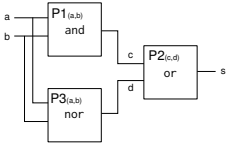


Simulation événementielle

évènements et liste de sensibilité

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x



				fin		fin
	0ns	0ns		0ns	0ns	0ns
	init	Δ_1	$\rightarrow$	Δ_1	Δ_2	Δ_2
		P1	P3		P2	
c	x	0	-	0	-	0
d	x	-	0	0	-	0
s	x	-	-	x	0	0

On a réduit le nombre d'itérations de simulation et de fonctions exécutées.

Simulation événementielle
La gestion du temps

Le simulateur maintient un compteur pour le **temps physique** indépendant du temps symboliques.
À un évènement sont attachés 2 informations de temps :

- Le temps symbolique Δ
  - ou le cycle de simulation
- Le temps physique
  - celui qu'on voudrait observer

Ceci permet d'ordonner les événements dans **l'échéancier** du simulateur ou de préciser le temps physique auquel il doit être pris en compte.

On parle de **notification**.

Simulation événementielle
Exemple

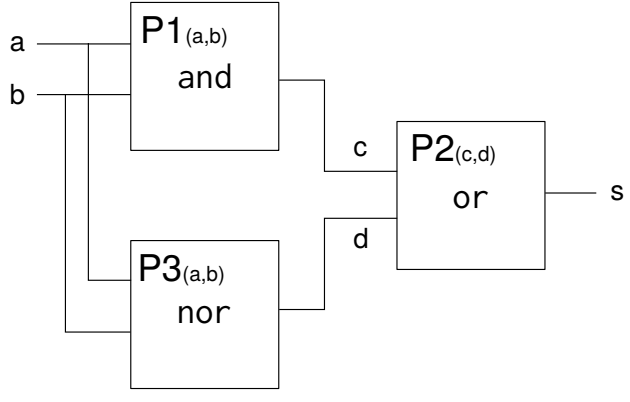
- a = x
- b = x
- c = x
- d = x
- s = x

@t=0ns

- a = 1
- b = 0

@t=10ns

- a = 1
- b = 1



Simulation événementielle

Exemple

- a = x
- b = x
- c = x
- d = x
- s = x

@t=0ns

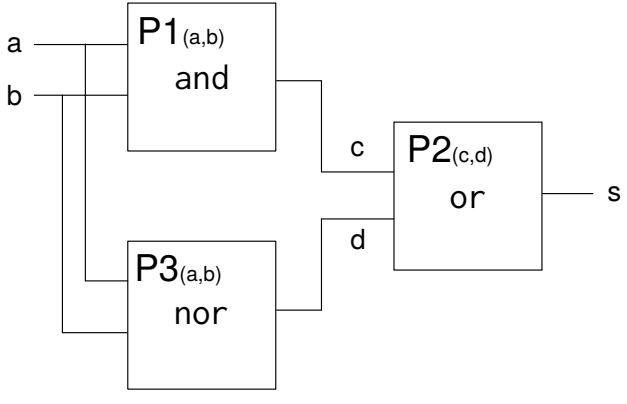
- a = 1
- b = 0

@t=10ns

- a = 1
- b = 1

@

P0



Simulation événementielle
le temps physique

		fin			fin		fin		fin			fin		fin
		0ns			0ns	0ns	0ns	→	10ns	10ns		10ns	10ns	10ns
	init	Δ_0	Δ_1	→	Δ_1	Δ_2	Δ_2		Δ_0	Δ_1	→	Δ_1	Δ_2	Δ_2
		P0	P1	P3		P2			P0	P1	P3		P2	
a	x	1	–	–	1	–	1		1	–	–	1	–	1
b	x	0	–	–	0	–	0		1	–	–	1	–	1
c	x	x	0	–	0	–	0		–	1	–	1	–	1
d	x	x	–	0	0	–	0		–	–	0	0	–	0
s	x	x	–	–	x	0	0		–	–	–	0	1	1

Le temps physique avance quand il n'y a plus d'évènements déclenchant un processus.

Simulation événementielle
Fonctionnement des processus

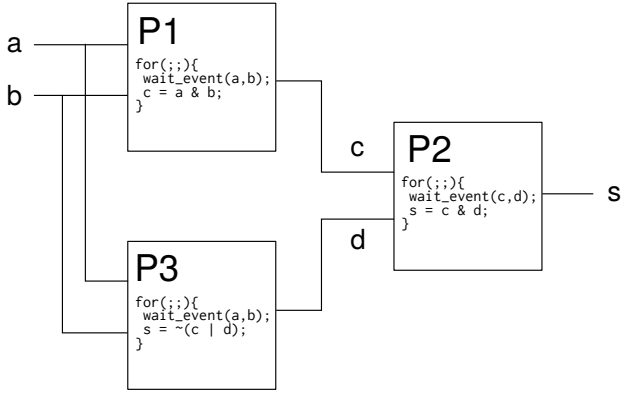
Les processus sont en pratique des **boucles infinies**.

Ils sont en attente d'un événement :

- Implicite due à la liste de sensibilité
- Explicite à l'aide d'instructions de synchronisation

Simulation événementielle
Fonctionnement des processus

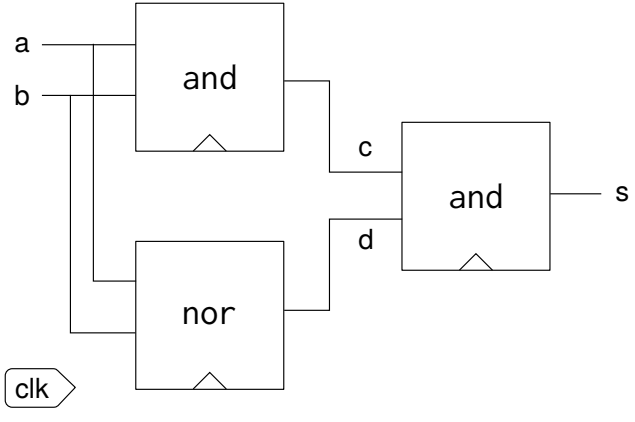
```
P0  for(;;){
    wait_for(0,ns);
    a = 1;
    b = 0;
    wait_for(10,ns);
    a = 1;
    b = 1;
    wait_forever();
    }
```



Simulation événementielle
et logique séquentielle ?

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x



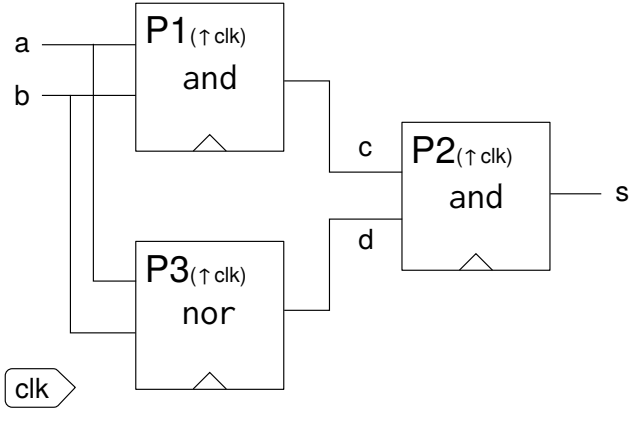
Simulation événementielle
et logique séquentielle ?

- Un évènement unique sur une entrée particulière, l'horloge.

Simulation événementielle
et logique séquentielle ?

Initialement :

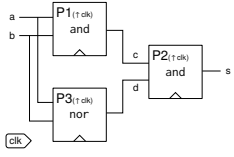
- a = 1
- b = 0
- c = x
- d = x
- s = x



Simulation événementielle
et logique séquentielle ?

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x

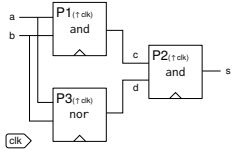


							fin					fin
		0ns	→	↑ clk				→	↑ clk			
	init	Δ_0		Δ_1	→	→	Δ_1		Δ_1	→	→	Δ_1
		P0		P1	P2	P3			P1	P2	P3	
a	x	1		–	–	–	–		–	–	–	–
b	x	0		–	–	–	–		–	–	–	–
c	x	x		0	–	–	0		0	–	–	0
d	x	x		–	–	0	0		–	–	0	0
s	x	x		–	0	–	0		–	0	–	0

Simulation événementielle
et logique séquentielle ?

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x



							fin					fin
		0ns	→	↑ clk				→	↑ clk			
	init	Δ_0		Δ_1	→	→	Δ_1		Δ_1	→	→	Δ_1
		P0		P2	P3	P1			P2	P3	P1	
a	x	1		–	–	–	1		–	–	–	1
b	x	0		–	–	–	0		–	–	–	0
c	x	x		–	–	0	0		–	–	0	0
d	x	x		–	0	–	0		–	0	–	0
s	x	x		x	–	–	x		0	–	–	0

Simulation événementielle
et logique séquentielle ?

Problème

- Tous les processus sont déclenchés en même temps.
- En modifiant instantanément les sorties, le résultat n'est pas tout le temps le même, il dépend de l'ordre d'exécution.

Simulation événementielle
Les signaux

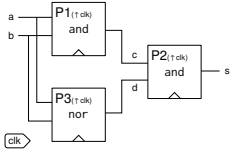
- Un **signal** est une structure de données contenant :
  - la valeur **courante** avant le Δ
  - la valeur **future** qu'il aura après le Δ
- Une affectation sur un signal ne modifie que sa valeur future
- Lire un signal renvoie sa valeur courante
- À la fin du Δ la valeur du signal est mise à jour
  - ie. la valeur future remplace la valeur courante

La valeur d'signal est maintenue tant que tous les processus actifs à un «Δ» n'ont pas été exécutés. On parle d'**affectations différées**.

Simulation événementielle
et logique séquentielle ?

Initialement :

- a = 1
- b = 0
- c = x
- d = x
- s = x



		m.à.j				m.à.j				m.à.j
	0ns	→	↑ clk			→	↑ clk			
init	Δ_0		Δ_1	→	→	Δ_1	Δ_1	→	→	Δ_1
	P0		P1	P2	P3		P1	P2	P3	
a (x,x)	(x,1)	(1,1)	-	-	-	-	-	-	-	-
b (x,x)	(x,0)	(0,0)	-	-	-	-	-	-	-	-
c (x,x)	(x,x)	(x,x)	(x,0)	-	-	(0,0)	(0,0)	-	-	(0,0)
d (x,x)	(x,x)	(x,x)	-	-	(x,0)	(0,0)	-	-	(0,0)	(0,0)
s (x,x)	(x,x)	(x,x)	-	(x,x)	-	(x,x)	-	(x,0)	-	(0,0)

Simulation événementielle (SPOILER)

Les affectations en Verilog/SystemVerilog

En Verilog/SystemVerilog, il n'y a pas de différence de déclaration entre une variable et un signal. C'est le symbole utilisé pour l'affectation qui permet de faire la différence :

- `a <= b` : affectation différée (donc signal)
- `a = b` : affectation immédiate (donc variable)

Simulation événementielle (SPOILER)

Les affectations en Verilog/SystemVerilog

a <= b : affectation différée

Doivent être utilisées pour modéliser de la logique séquentielle de façon deterministe.

a = b : affectation immédiate

Peuvent être utilisées pour modéliser de la logique combinatoire.

Simuler le parallélisme
Résumons

- Décrire sous la forme d'un ensemble de fonctions : **les processus**
- La communication entre ces processus se faisant en modifiant des variables globales de type **signal** pour garantir le déterminisme.
- On définit la liste des **évènements** sur les **signaux** qui nécessitent de relancer des processus.

Simuler le parallélisme
principes

1. Exécuter les processus dans un ordre arbitraire.
2. Si on demande à modifier la valeur d'un signal, mémoriser sa valeur **future**.
   Les variables sont modifiées immédiatement.
3. Quand tous les processus ont été exécutés (à la fin du Δ), modifier vraiment la valeur des signaux.
4. Refaire les étapes 1,2,3 si on a déclenché un nouvel évènement.
5. S'il n'y a aucun évenement, alors, faire avaner le temps physique

Simulation événementielle

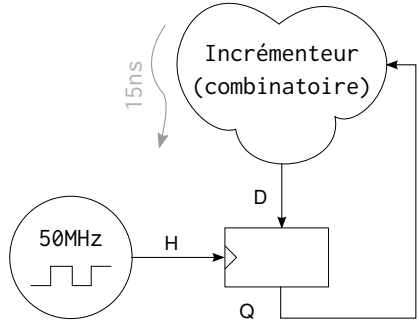
Exercice à faire

On veut simuler un système composé :

- un générateur d'horloge,
- un registre,
- un incrémenteur.

Avec

- 50MHz de fréquence d'horloge,
- 15ns de temps de propagation.



Simulation événementielle

Exercice à faire

Simulez «à la main» la description suivante :

Notation :

`@(x)`	:	attente d'un évènement sur x
`#10`	:	processus stoppé pour 10ns
`x <= y`	:	affectation différée à la fin du Δ
`x <= #10 y`	:	affectation différée de 10ns

P1

```
H <= 0;
#10;
H <= 1;
#10;
```

P2

```
@(H);
if H ==1
 Q <= D;
```

P3

```
@(Q);
D <= #15 Q+1;
```

À mettre dans votre dépôt git dans un dossier **`ExoSim`**!