TELECOM ParisTech

INSTITUT Mines-Télécom

ELECINF 102 : Processeurs et Architectures Numériques

Automates finis: Réalisation matérielle de séquenceurs

Yves Mathieu
yves.mathieu@telecom-paristech.fr

Automates finis

Vocabulaire

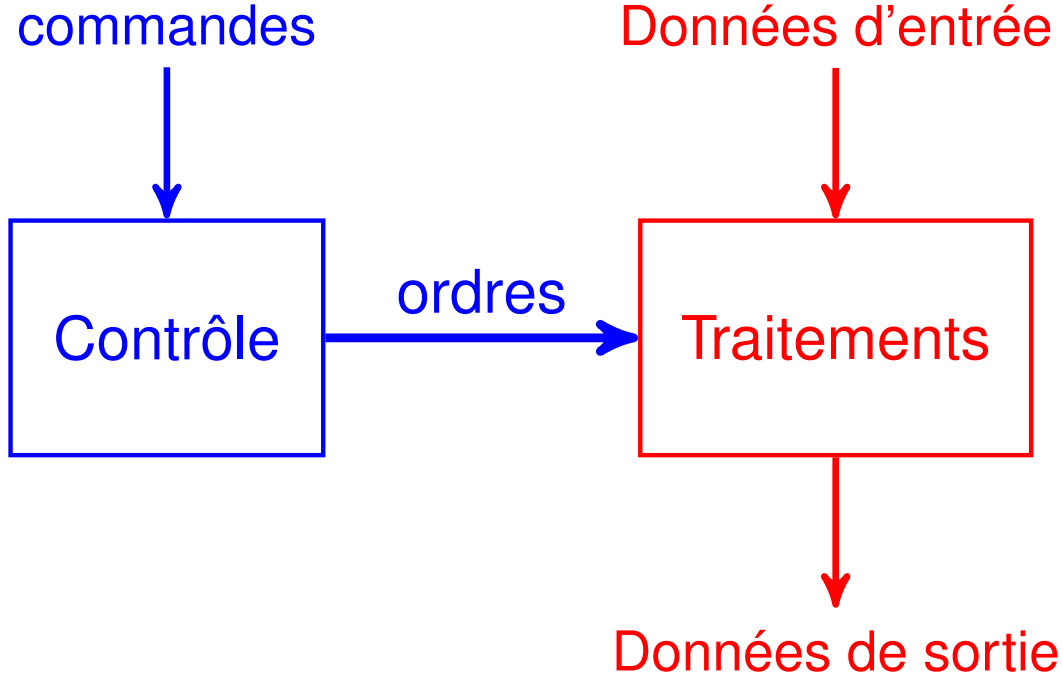
- Automates finis
- Machines à états finis (MAE)
- Finite–State Machines (FSM)
- Ce qui est fini, c'est le nombre d'états

Automates finis

Pourquoi ?

Dans une architecture, on distingue traditionellement

- Le **traitement** proprement dit des données
- le **contrôle** de ce traitement.



Automates finis

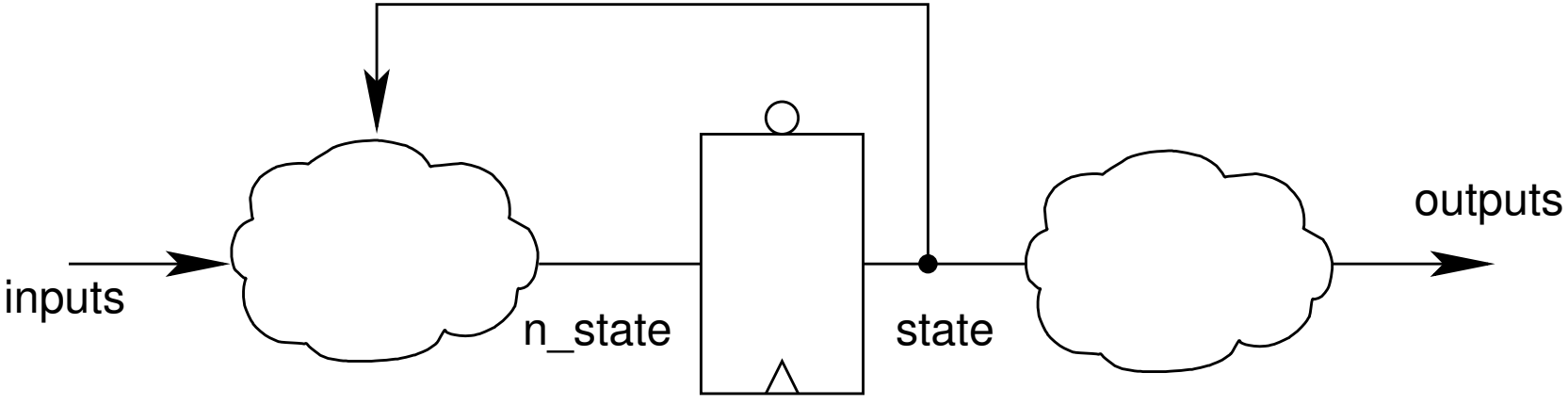
Les automates sont une solution à la réalisation des blocs de **contrôle**.

- Un automate fini :
  - Définit l'**état** d'un système
  - Définit les actions effectuées dans un état déterminé.
  - Définit le passage d'un état à l'autre (les **transitions**)

Automates finis en matériel

- Mémoriser l'état $\rightarrow$ **logique synchrone**
  - Un registre pour mémoriser l'état
  - La taille du registre est liée aux nombres d'états
  - La réinitialisation du registre permet de maitriser l'état de départ
- En fonction de l'**état courant** calculer combinatoirement les sorties (les **ordres**)
- En fonction de l'**état courant** et des entrées (les **commandes**) calculer combinatoirement l'état futur

Automates finis en matériel



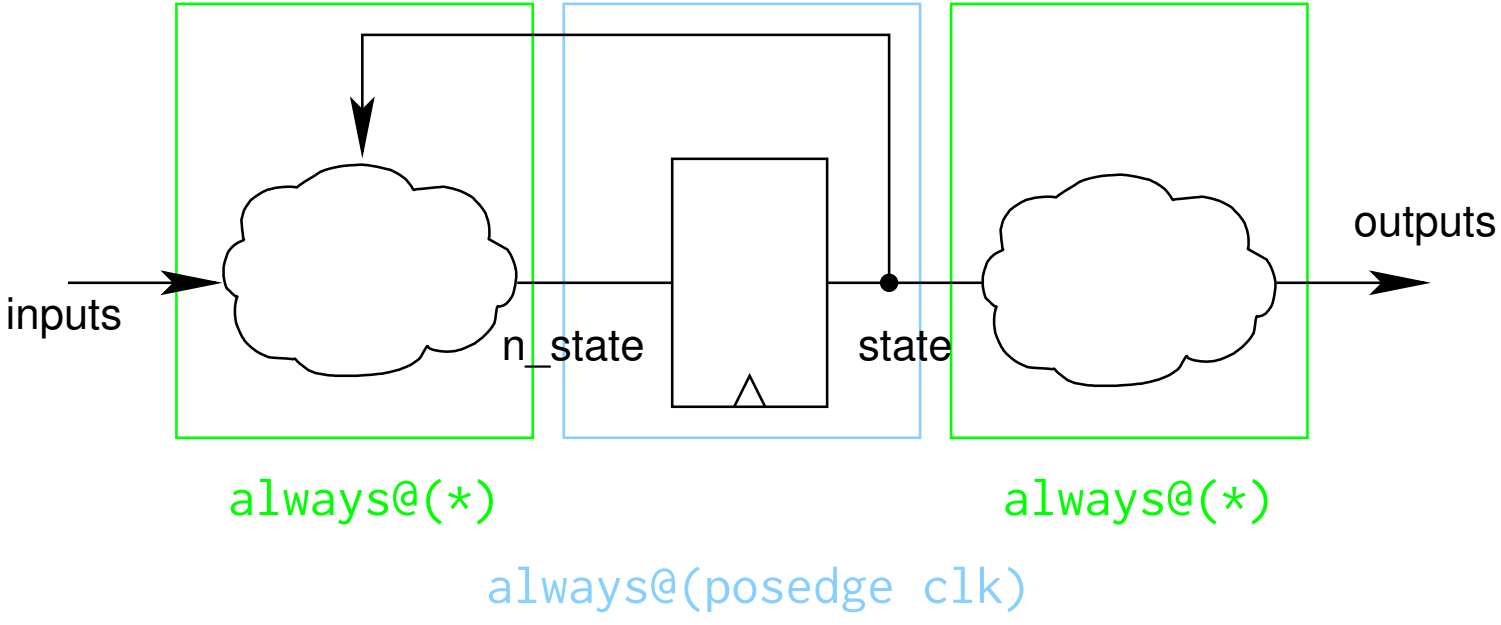
- La sortie du registre représente l'état courant.
- L'entrée du registre représente l'état futur.
- Au front d'horloge, l'état courant est mis à jour.

Cette architecture est appelée **machine à états de Moore**

Codage en SystemVerilog

Trois blocs

- Un bloc séquentiel pour sauvegarder l'état.
- Deux blocs combinatoires :
  - Calcul de l'état futur.
  - Calcul des sorties.



Représentation en SystemVerilog

Trois blocs

Déclaration d'un type **enum** pour avoir un code lisible.

```
enum logic[1:0] {INIT, S0, S1} state, n_state ;
```

Calcul de l'état futur

```
always @( *)
begin
   // par défaut on reste
   // dans l'état courant
  n_state <= state ;
  case (state)
   INIT: if (cond0)
            n_state <= S0;
   S0  : if (cond1)
            n_state <= S1;
   S1  : if (cond2)
            n_state <= INIT;
  endcase
end
```

Stockage de l'état courant

```
always @(posedge clk)
if (reset)
    state <= INIT ;
else
    state <= n_state ;
```

Calcul des sorties

```
always @( *)
    output1 <= f(state);

always @( *)
    output2 <= g(state);

always @( *)
    output3 <= ...
...
```

Avant de coder...

Une méthode générique de conception

- On trace un graphe
  - C'est un graphe orienté
  - Chaque état est représenté par un cercle
  - On précise l'état initial par un double cercle (après un reset)
  - On précise pour chaque état la valeur des sorties
  - Les états sont liés par des flèches qui représentent les transitions
  - La condition de chaque transition doit être précisée
  - Ces conditions dépendent des entrées

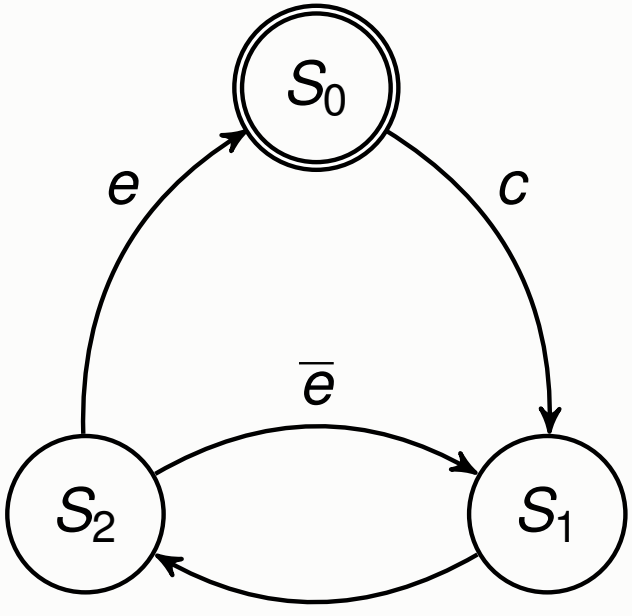
Comment ?

Une méthode générique de conception

Automates finis

Une méthode générique de conception

Exemple



- L'état initial est S_0
- Le passage de S_0 à S_1 ne se fait que si la condition c est vraie
- Le transition de S_1 à S_2 est inconditionnelle (toujours vraie)
- À partir de S_2 on passe à S_0 ou S_1 en fonction de e

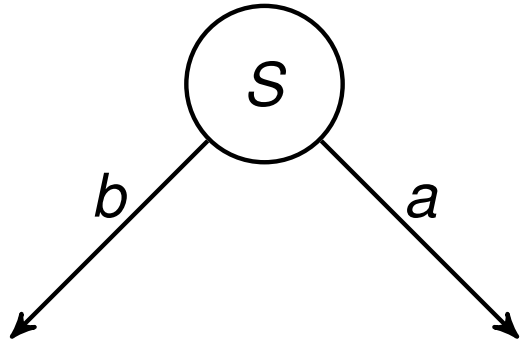
REMARQUE : Compte tenu de la technique de codage SystemVerilog, on ne représentera que les transitions où l'état futur est différent de l'état courant.

Automates finis

Règles de construction

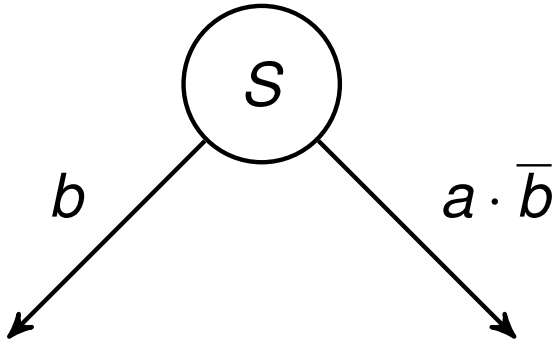
Graphe non contradictoire

Deux transitions avec des conditions contradictoires ne doivent pas partir du même état.



Graphe non contradictoire

Deux transitions avec des conditions contradictoires ne doivent pas partir du même état.



Une variante

La machine de Mealy

- La sortie peut aussi dépendre directement des entrées