

# MSS simples

## (Machines Séquentielles Synchrones)



# Pourquoi des systèmes séquentiels

- L'évolution de certains systèmes dépend des entrées **ET** de l'historique des événements.
- Pour une même combinaison des entrées l'état des sorties peut être différent
  - ⇒ Ce n'est plus un système **univoque** (Se dit d'une relation logique entre deux objets qui ne s'exerce que dans un sens)

Nécessaire de mémoriser l'historique du système :

**éléments mémoires indispensables**

# Types de machines séquentielles

- Machines séquentielles synchrones MSS:
  - évolution du comportement défini par un signal particulier, nommé généralement horloge
- Machines séquentielles asynchrones MSA:
  - évolution du comportement défini par le changement d'une des entrées
  - Il n'y a aucun signal qui cadence l'évolution

**Nous ne traiterons pas les MSA dans le cadre de ce cours**

# Machines séquentielles synchrones MSS

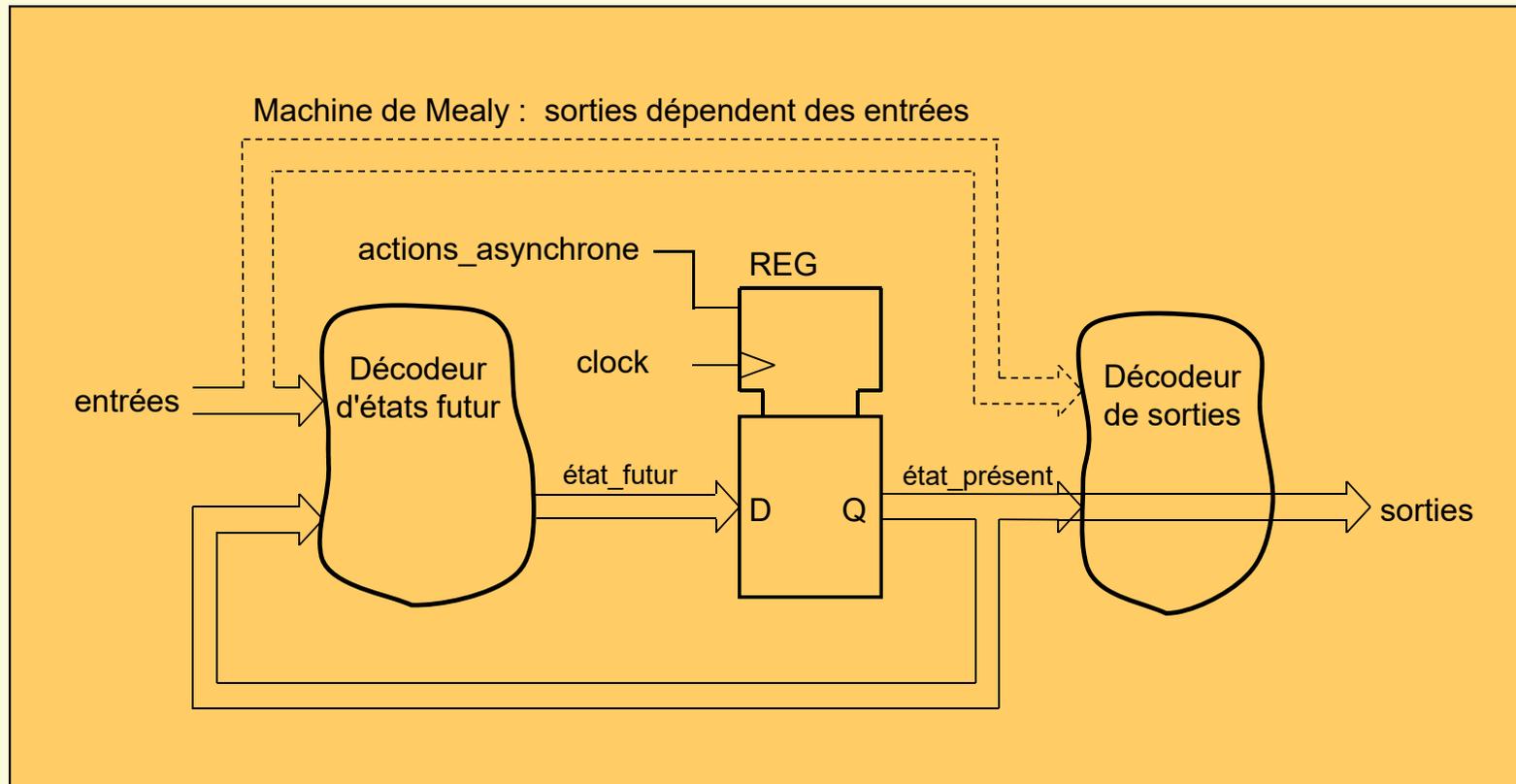
- Afin de connaître l'évolution du système une mémorisation est nécessaire.
- L'état du système est stocké dans les variables (bits) d'état du système.
- Ces variables, à un instant donné, contiennent toutes les informations nécessaires au calcul du comportement futur du système.
- L'état futur du système est fonction de son état courant et des entrées
  - état futur =  $F(\text{état présent}, \text{entrées})$ .
- Un système séquentiel synchrone voit son état synchronisé par un signal dit d' «horloge».
- Une MSS est aussi appelée machine d'état.

# Décomposition d'une MSS

## (Machine Séquentielle Synchrone)

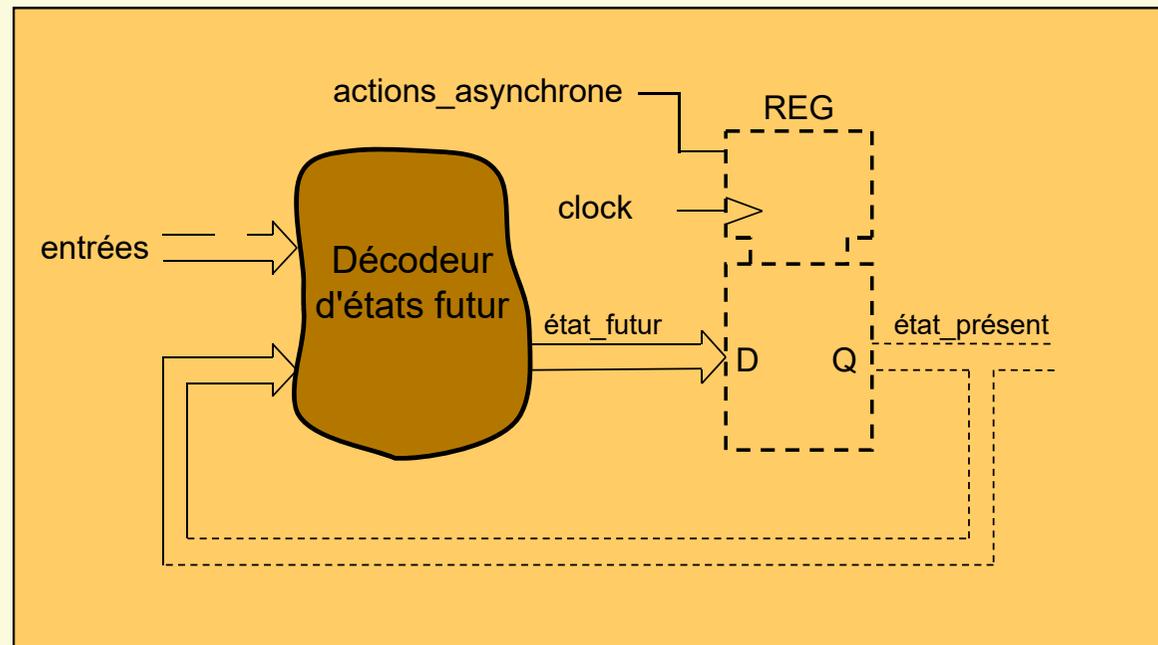
- Ramener les MSS (machines séquentiels synchrones) à :
  - des systèmes combinatoires
  - + des variables internes (mémoires)
- Éléments mémoires constitués par des flip-flops, nommés :  
**Bits d'état**
- Les flip-flops répondent à la définition énoncée au chapitre précédent

# Schéma bloc d'une MSS simple



# Décodeur d'état futur :

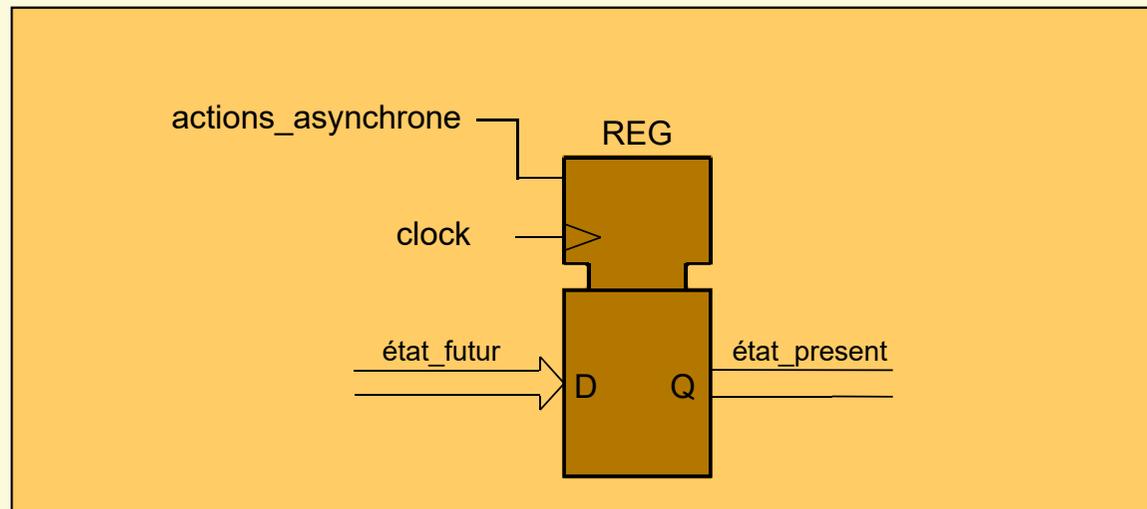
- Système logique combinatoire (SLC) générant `état_futur` en fonction de :
  - signaux d'entrée & état de la machine `état_présent`



# Etat de la MSS / Bits d'état

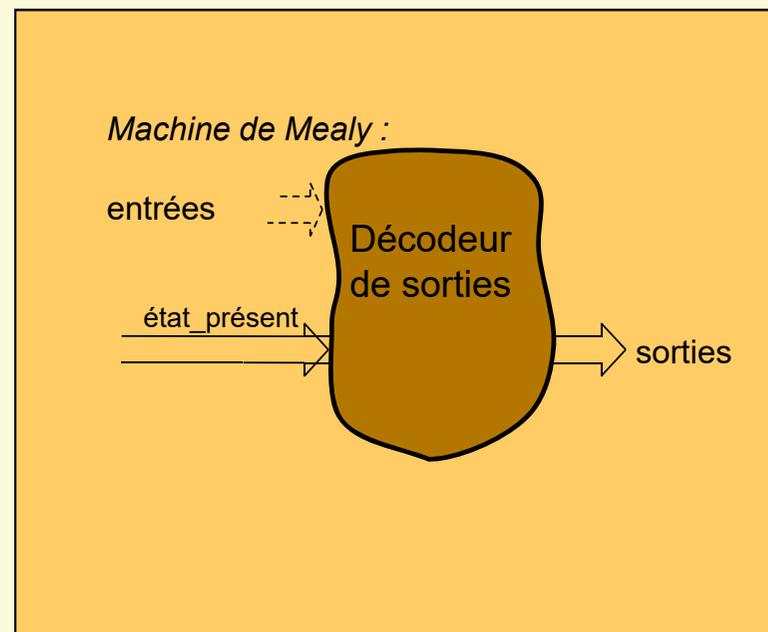
Bits d'état: `état_present`

- Mémorisation de l'état présent
- Registre synchrone constitué de flip-flops D



# Décodeur de sorties

- Système logique combinatoire (SLC) générant l'état des sorties en fonction de :
  - état de la machine `état_présent` & signaux d'entrée (Mealy)



# MSS simple: principe

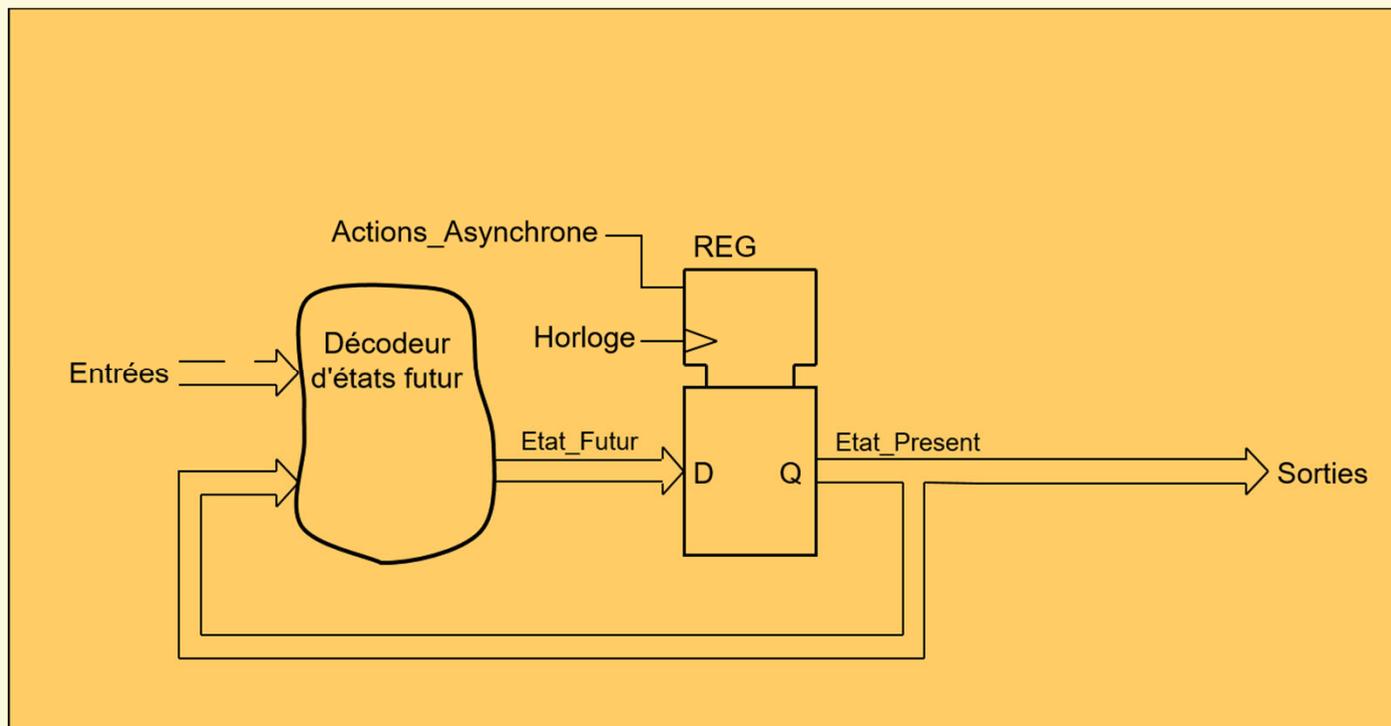
- Principe de fonctionnement général:
  - Le système fait évoluer les entrées
  - L'état interne évolue selon le changement des entrées
  - Les sorties dépendent de l'état interne
    - Sauf Mealy: les sorties dépendent aussi des entrées (changement immédiat)

## Trois types de MSS simples

- MEDVEDEV
  - Sorties correspond directement aux bits d'états
- MOORE
  - Sorties dépendent uniquement de l'état présent
- MEALY
  - Sorties dépendent de l'état présent **et** des entrées

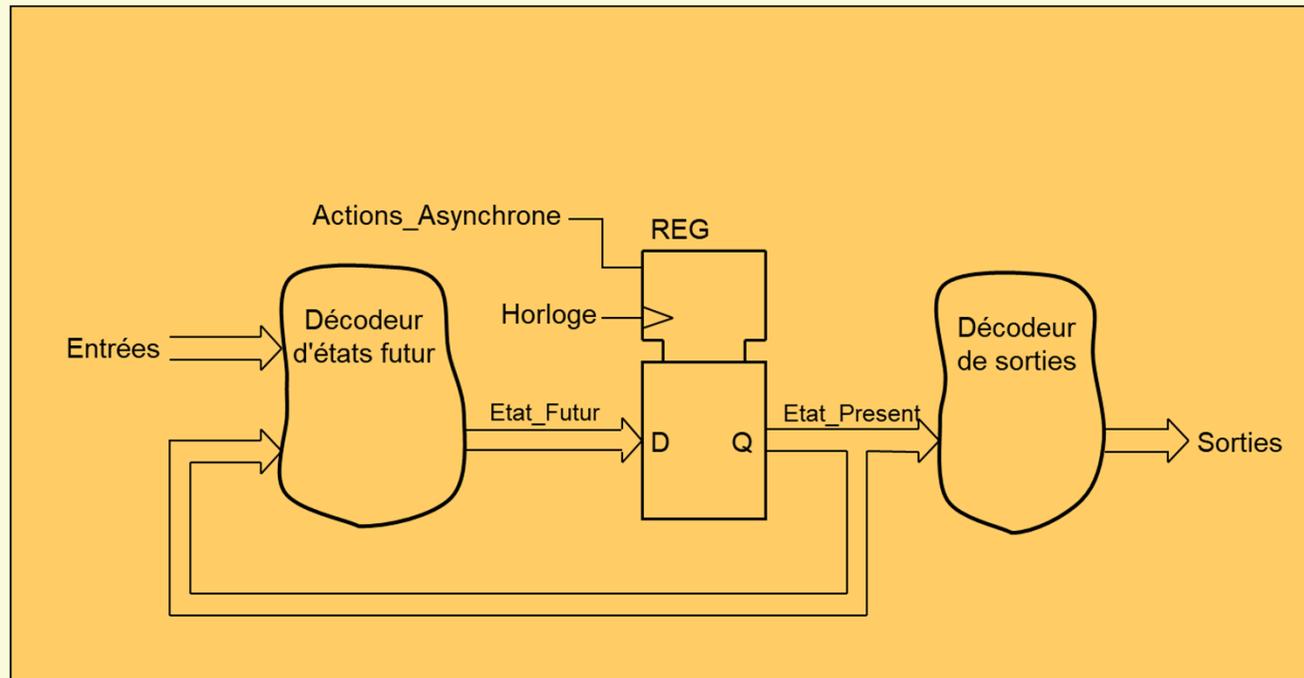
# MSS simple de MEDVEDEV

- Les sorties sont égales aux bits d'états
  - Sorties inconditionnelles



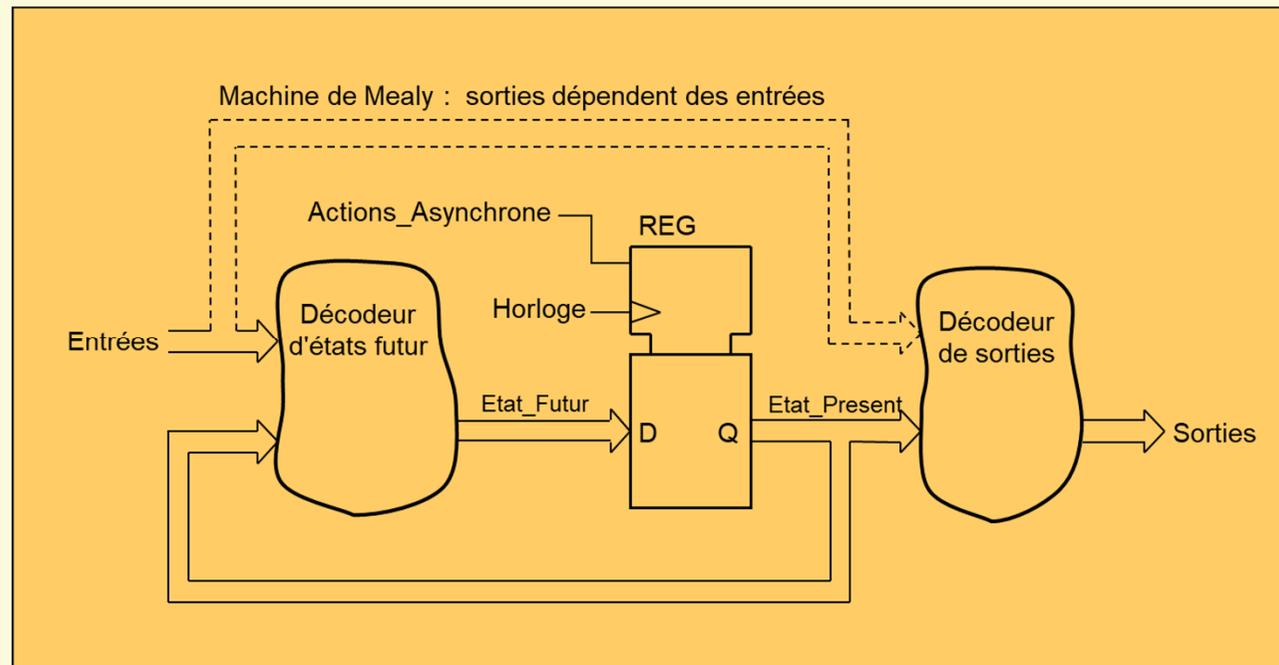
# MSS simple de MOORE

- Les sorties changent uniquement après la mise à jour de l'état interne (bits d'état)
  - Sorties inconditionnelles



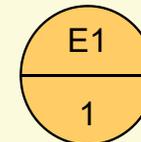
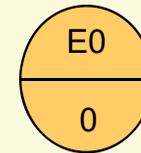
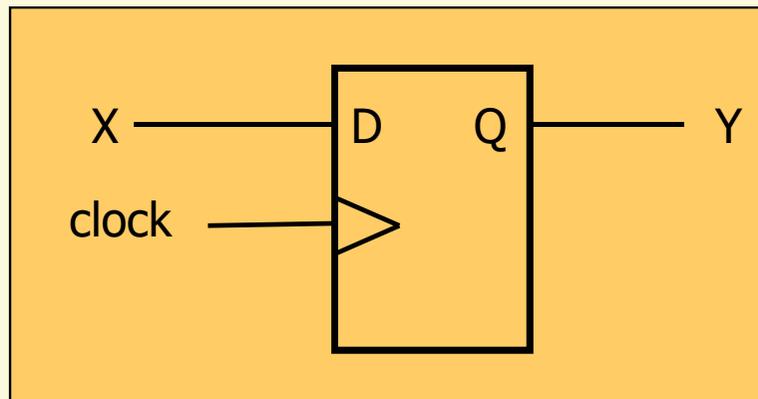
# MSS simple de MEALY

- Les sorties changent immédiatement avec les entrées et après la mise à jours de l'état interne (bits d'état)
  - sorties conditionnelles



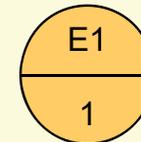
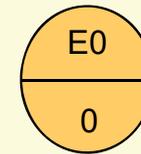
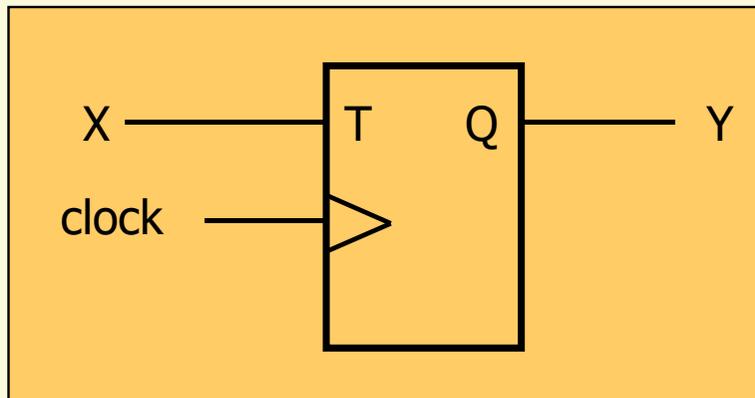
# Exemple MSS ultra simple

- Flip-flop D (DFF): une **entrée X**, une **sortie Y** et  **$2^1$  états**  
(Sortie inconditionnelle « Moore »)
  - Logique combinatoire réduite à zéro !



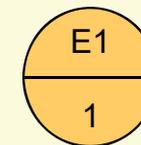
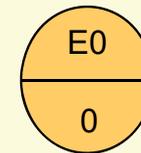
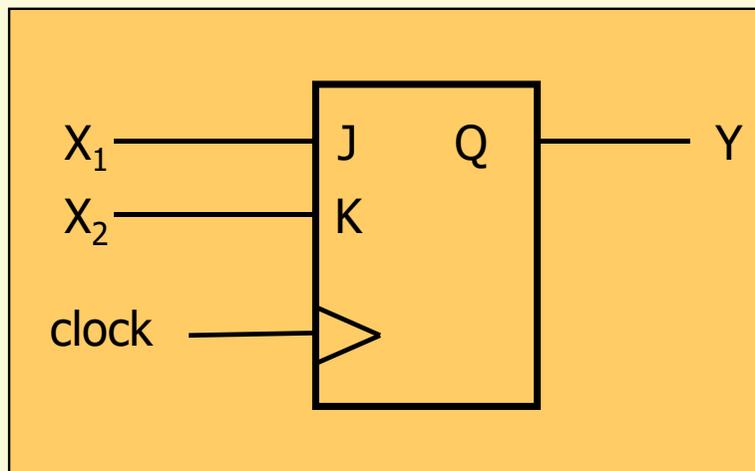
# Exemple MSS ultra simple

- Bascule T : une **entrée X**, une **sortie Y** et  **$2^1$  états**  
(Sortie inconditionnelle « Moore »)



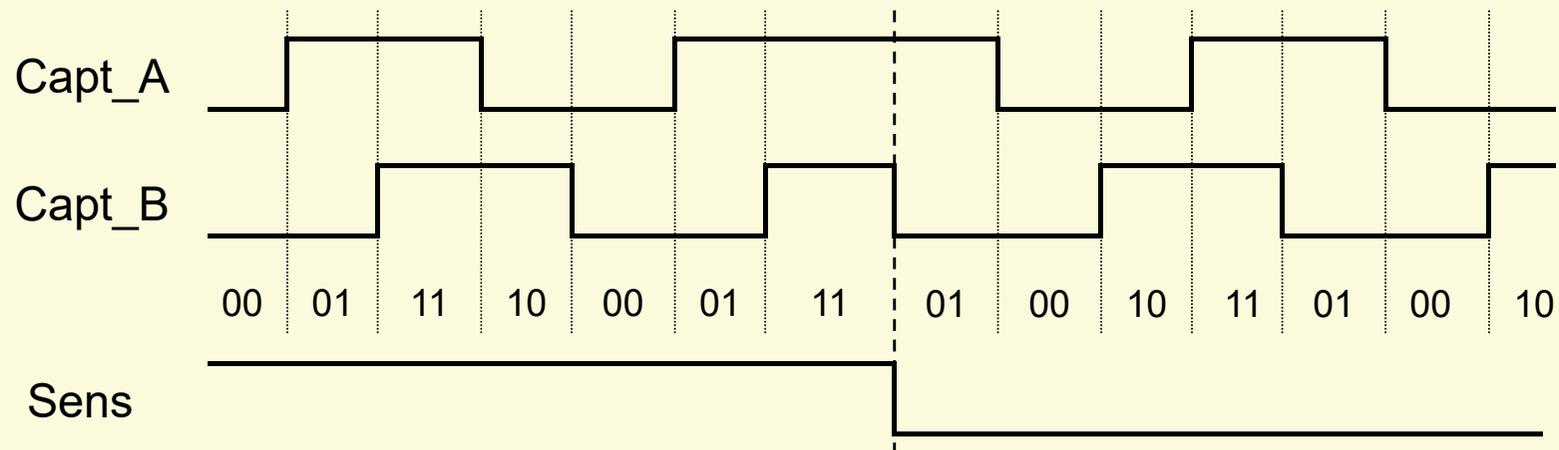
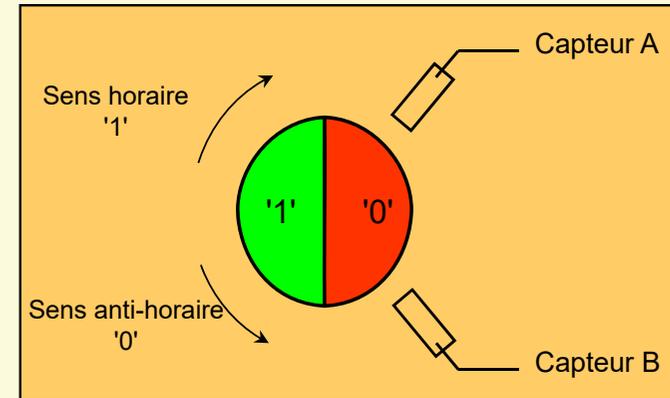
# Exemple MSS ultra simple

- Bascule JK : deux entrée  $X_1$  et  $X_2$ , une sortie  $Y$  et  $2^1$  états  
(Sortie inconditionnelle « Moore »)



# Exemple de système séquentiel

- Détecteur de sens de rotation



# Exemple de système séquentiel

---

- Animation détecteur de sens de rotation
  - [vidéo](#)

# Analyse du détecteur de sens de rotation

---

- La combinaison des entrées Capt\_A et Capt\_B ne permet pas de déterminer le sens de rotation
- La sortie Sens dépend :
  - Succession des combinaisons Capt\_A/Capt\_B
  - Nécessaire de connaître l'état précédent => mémorisation nécessaire

# Spécification MSS

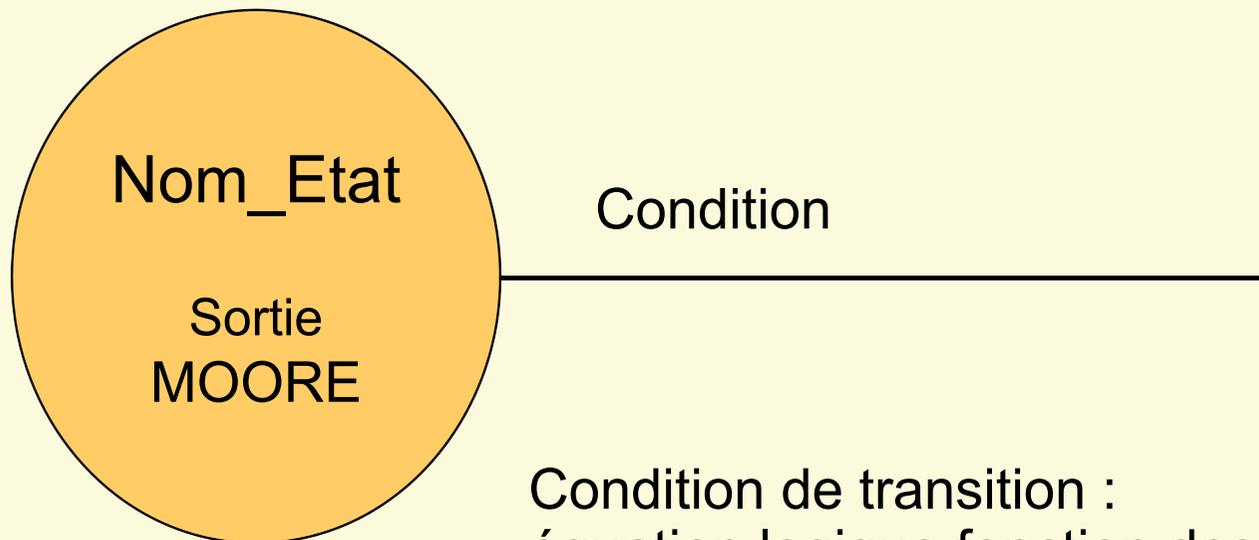
- Chronogramme :
  - présente une seule séquence
  - vue partielle
  - grand nombre de chronogramme
- Graphe des états :
  - méthode graphique
  - permet de représenter tous les "chemins" possibles
  - bien adapté pour des MSS simples

# Graphe des états ...

- Constitué d'une série de bulles reliées par des flèches (transitions):
  - une bulle représente un état distinct du système séquentiel. Elle correspond à un code des bits d'état.
  - une flèche représente une transition entre deux états. La condition, fonction des entrées, sera écrite sur chaque transition.

# ... graphe des états

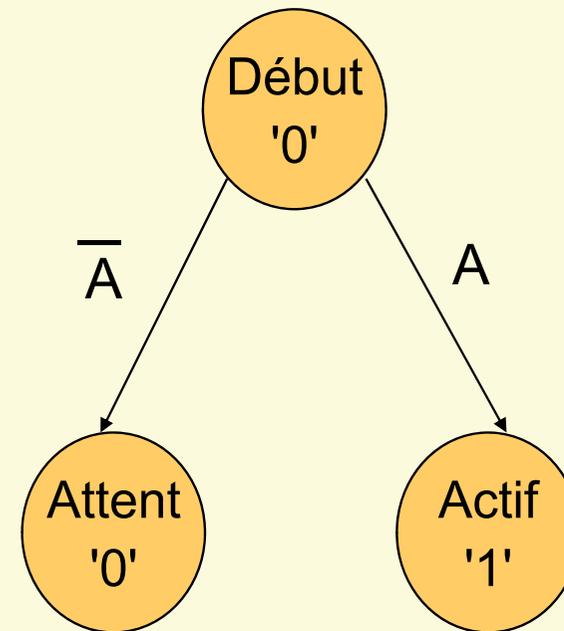
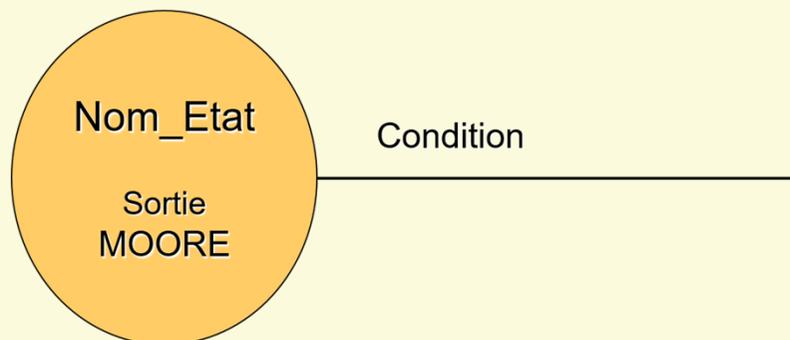
Convention de dessin Moore :



Condition de transition :  
équation logique fonction des entrées

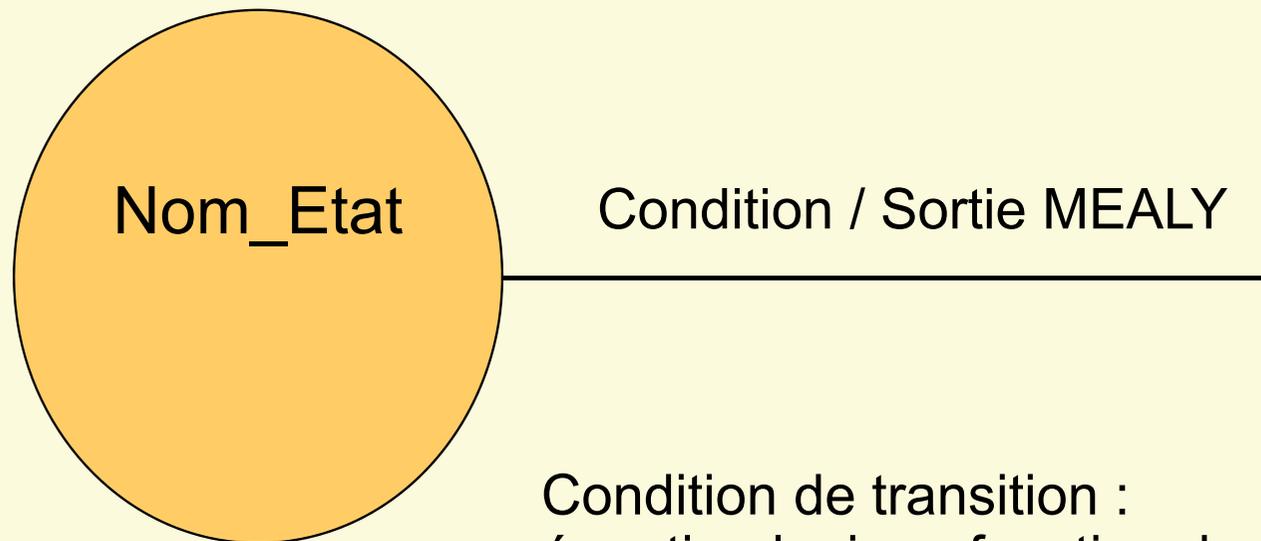
# Graphe des états pour MOORE

- Chaque état est nommé
- L'état de la sortie est indiqué dans chaque état
- Sur chaque transition la condition est indiquée



# ... graphe des états

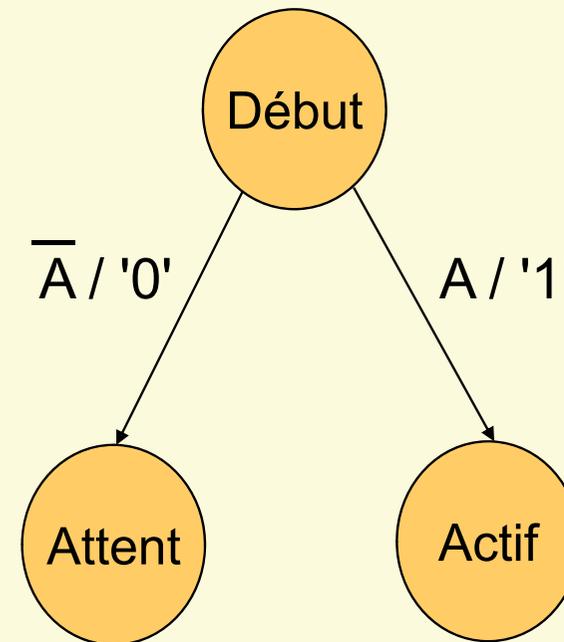
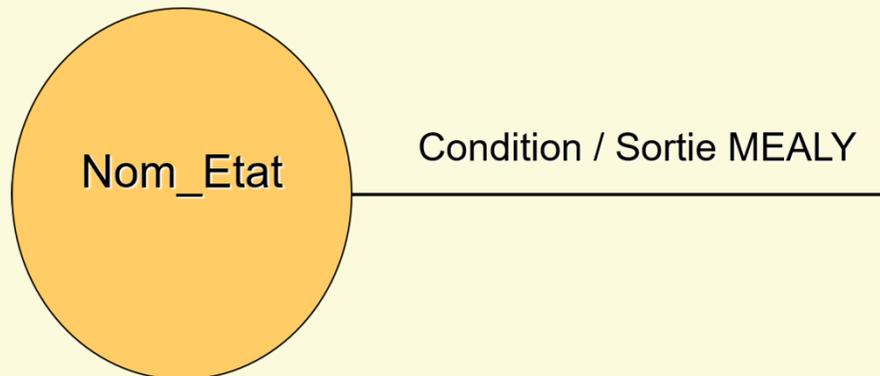
Convention de dessin Mealy :



Condition de transition :  
équation logique fonction des entrées

# Graphe des états pour MEALY

- Chaque état est nommé
- Sur chaque transition :
  - la condition est indiquée
  - l'état de la sortie est indiqué



# Conception d'un graphe des états

## Méthode de conception :

- Génération de proche en proche
- Génération depuis un chronogramme
- Génération par l'identification des états internes

## Conception d'un graphe des états:

- Appliquer les règles présentées ci-après en suivant une des méthodes indiquées ci-dessus
- Puis: **compléter** le graphe en respectant la règle 1 (ci-après)

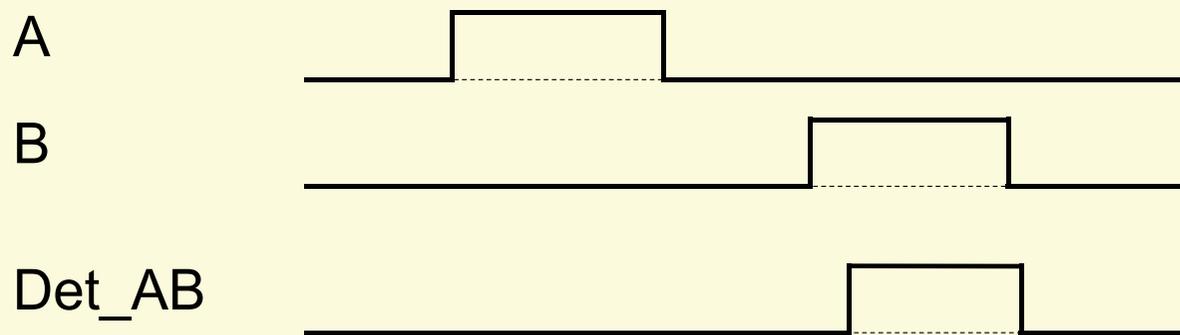
# Règles construction graphe des états

IMPORTANT

1. De chaque bulle d'état :
  - Autant de flèches que de combinaisons valides des entrées
2. Chaque changement des entrées pertinentes (nouvelle combinaison) provoque un changement d'état
  - Génère souvent trop d'états => seront simplifiés après
3. Chaque changement sur les sorties, avec les mêmes valeurs des entrées, implique un changement d'état
4. Maintien des sorties pendant un certains temps (multiple Thorloge) est réalisé par une succession d'état interne
  - 1 état = maintien de 1 Thorloge

# Exemple d'un détecteur de séquence A-B

- L'objectif est de réaliser un système qui détecte l'activation successive de deux boutons A et B
- Voici un exemple de fonctionnement



⇒ voir présentation séparée

# Table des états ...

---

- Représentation sous la forme d'un tableau du graphe des états
- C'est la liste des actions synchrones de la MSS,  
équivalent à la liste des actions synchrones d'un registre/compteur
- Utilisée pour déterminer les équations logiques du décodeur d'état futur et du décodeur de sortie

# ... table des états ...

## Structure pour une MSS de Moore

Toutes les combinaisons des entrées

Sorties  
Dépendent uniquement état présent

Etat présent	Etat futur				Sorties	
	00	01	11	10	X	Y
E0	(E0)	(E0)	E1	E2	0	0
E1	E2	E0	(E1)	E0	1	0
E2	E0	E3	E1	(E2)	0	1
...	...	....				...

Liste de tous les états présents

(...) Indique un état stable

# ... table des états

## Structure pour une MSS de Mealy

Etat présent	Etat futur f(A,B) / Sorties X Y			
	00	01	11	10
E0	(E0) / 00	(E0) / 00	E1 / 10	E2 / 01
E1	E2 / 01	E0 / 00	(E1) / 10	E0 / 00
E2	E0 / 00	E3 / 11	E1 / 10	(E2) / 01
...	...	....		

Toutes les combinaisons des entrées

Sorties  
Dépendent de l'état présent et des entrées

Liste de tous les états

(...) Indique un état stable

# Réduction MSS

Méthode pour réduire le nombre des états :

- Recherche des états identiques
  - même états futurs et mêmes sorties pour chaque combinaison des entrées
- Recherche des états compatibles
  - états futurs différents mais compatible après groupement des états compatibles;  
les sorties doivent être identique

Remarque : il y a des méthodes systématiques pour réduire une table des états.  
Pas traité dans le cadre de ce cours.

# Codage des états d'une MSS

- Jusqu'ici :
  - Etats internes désignés par un nom symbolique
- Réaliser un circuit :
  - Assigner un code binaire distinct à chaque état
- MSS comportant M états, il faudra N bits avec :  $2^N \geq M$

# Choix du code des états

- Il y a des règles obligatoires à respecter pour garantir le bon fonctionnement de la MSS.
- D'autre part il faut optimiser le codage ?
  - Dans le cas d'une machine à 9 états ( $m=9$ ), avec laquelle nous utilisons 4 bits pour le codage ( $n=4$ ), il y a 10,8 millions de possibilités de codage distincts !
  - Nous verrons des pistes pour optimiser le codage.

# Contraintes pour fonctionnement correct

Le codage doit garantir un fonctionnement **correct** de la MSS

- Entrées asynchrones :
  - **Obligatoire** de synchroniser les entrées correspondantes avec un flip-flop
- Sorties sans aléas (transitoires):
  - **Obligatoire** de générer les sorties correspondantes directement par un flip-flop
    - => la sortie doit correspondre à un bit d'état de la MSS!

# Sorties sans aléas ...

- Classification des sorties d'une MSS :
  - Action synchrone: pas de contrainte (aléas tolérés)
    - Toutes les commandes synchrone :  
Load, Shift, En, ....
  - Action asynchrone : doit être **sans aléas**
    - Reset asynchrone
  - Non recommandé !  
*Action dynamique : doit être sans aléas*
    - *Signal d'horloge*

# ... sorties sans aléas

- Solution pour obtenir une sortie sans aléas :
  - Sortie correspond à un bit d'état  
=> Solution **recommandée**
  - Sortie post-synchronisée  
Attention : sortie mise à jour avec une **période de retard**

# Choix du code des états

- Déterminer le nombre de bits d'états  $N$  :
  - $2^N \geq$  nombres d'états
  - utiliser la combinaison "0...0" pour l'état initial (reset)
- puis
  - respecter contraintes pour le codage
  - optimiser au mieux les décodeurs
  - éventuellement augmenter le nombre de bits d'état

# Optimisation du codage

Le nombre de combinaisons possibles pour le codage d'une MSS est fréquemment de plusieurs millions de combinaisons. Il est dès lors nécessaire d'avoir quelques règles pour choisir le codage binaire

## Remarque:

- Dans le cas du codage " 1-parmi-M " ("one hot"), il n'y a pas d'optimisation possible.

# Optimisation du décodeur d'états futur

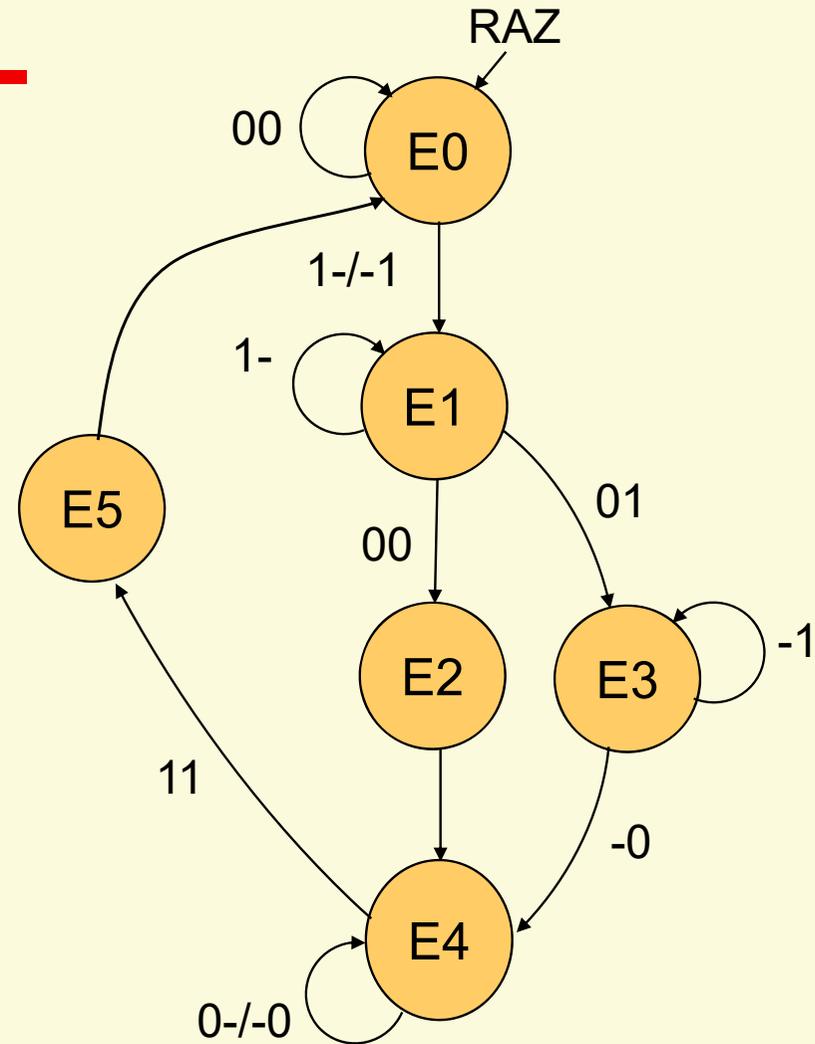
---

- Dans le cas d'un codage binaire, il est recommandé de placer les états étant liés par une ou plusieurs transitions (proches dans le graphe) dans des cases adjacentes de la table de Karnaugh

# Optimisation du décodeur d'états futur

- Exemple:

	$Y_2$	$Y_1$		
	00	01	11	10
$Y_0$				
0	E0			
1				



# Optimisation du décodeur de sorties

Pour une sortie inconditionnelle (type Moore), le codage devrait être choisi de façon à ce qu'un des bits d'états donne directement la valeur de la sortie (ou son inverse).

Proposition de simplification du décodeur de sorties:

- Utiliser un bit d'état pour chaque sortie, puis compléter avec autant de bit d'états nécessaires pour distinguer tous les états ayant les mêmes états des sorties
  - exemple s'il y a 3 états avec les même sorties il faut rajouter 2 bits d'états
  - exemple s'il y a 6 états avec les même sorties il faut rajouter 3 bits d'états
  - etc

# Exemple de codage des états

- Soit une MSS avec 6 états (E0 à E5)
  - L'état E0 est l'état initial de la MSS
  - Les entrées sont synchrones
  - Les sorties sont utilisées pour des commandes synchrones (aléas acceptables)
  - aucunes optimisations des décodeurs n'est appliquées

# exemple codage binaire

- Voici un codage des états :
  - Nous avons 6 états => il faut 3 bits ( $2^3 > 6$ )
  - Nous utilisons l'état "000" pour E0 (état initial)

Choix **arbitraire** pour les autres états :

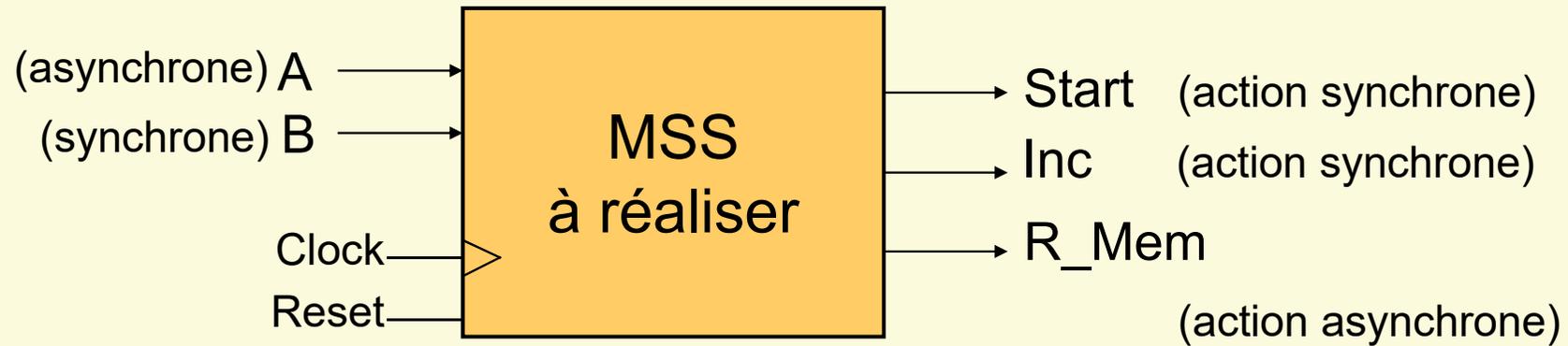
E0 = "000"	E3 = "011"
E1 = "001"	E4 = "100"
E2 = "010"	E5 = "101"

Les codes "110" et "111" sont **non utilisés**

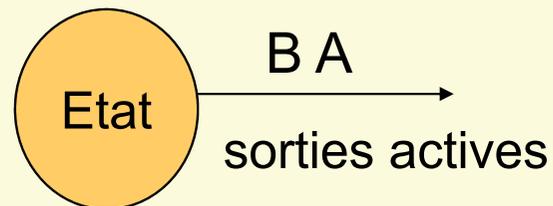
# exemple codage "One hot"

- Le codage d'état se fait sur autant de bits que d'états
  - Par exemple, 6 états => 6 bits d'état
- Un seul bit est à «1» dans le registre
  - Etat 1 => 000001
  - Etat 2 => 000010
  - Etat 3 => 000100
  - ...

# Exemple de MSS...

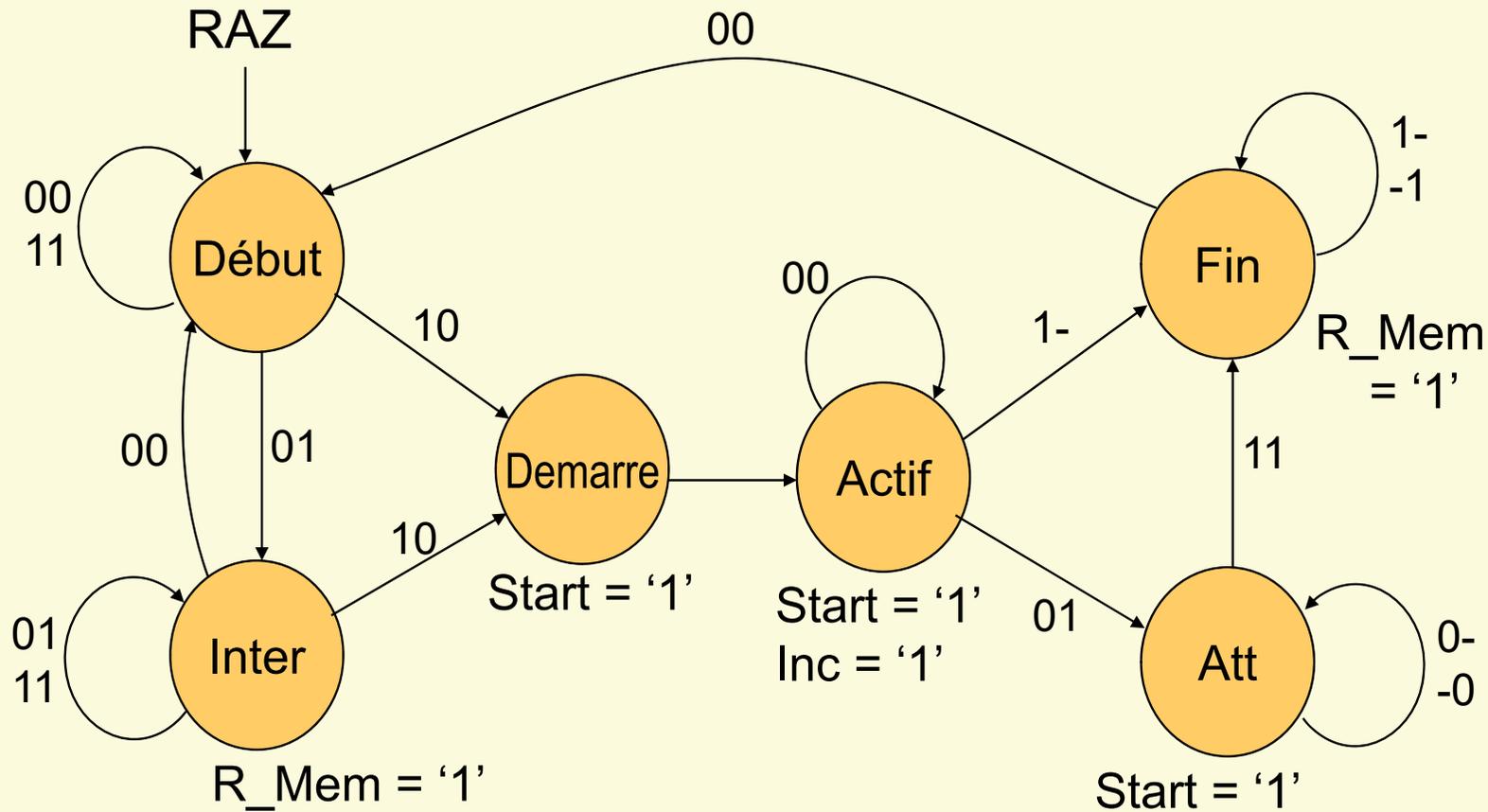


Convention pour le graphe:



# ... exemple de MSS...

# graphe des états



# ... exemple de MSS...

## Table des états de la MSS

Etat présent	Etat futur fonction (B,A)				Sorties		
	00	01	11	10	R	S	I
Debut	.....	.....	.....	.....	...	...	...
Inter	.....	.....	.....	.....	...	...	...
Demarre	.....	.....	.....	.....	...	...	...
Actif	.....	.....	.....	.....	...	...	...
Att	.....	.....	.....	.....	...	...	...
Fin	.....	.....	.....	.....	...	...	...

# ... exemple de MSS...

- Contrainte pour le codage :
  - Lister les mesures à respecter pour garantir un fonctionnement correct de la MSS

## ... exemple de MSS...

- Codage des états :
  - 6 états => 3 bits d'état nécessaire ( $Y_2 Y_1 Y_0$ )

$Y_2 \backslash Y_1$	00	01	11	10
0				
1				

# ... exemple de MSS...

Table des états avec le codage des états

Etat présent	Etat futur				fonction (B,A)		Sorties				
	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>	Y <sub>2</sub> <sup>+</sup>	Y <sub>1</sub> <sup>+</sup>	Y <sub>0</sub> <sup>+</sup>	11	10	R	S	I
Debut	0	0	0	.....	.....	.....	.....	.....	...	...	...
Inter	..	..	..	.....	.....	.....	.....	.....	...	...	...
Demarre	..	..	..	.....	.....	.....	.....	.....	...	...	...
Actif	..	..	..	.....	.....	.....	.....	.....	...	...	...
Att	..	..	..	.....	.....	.....	.....	.....	...	...	...
Fin	..	..	..	.....	.....	.....	.....	.....	...	...	...

# ... exemple de MSS

- Equations des bits d'états :
  - $Y_i^+ = F( B, A, Y_2, Y_1, Y_0)$  table de karnaugh à 5 entrées!
  
- Pour simplification table de Karnaugh à 5 et plus:
  - Simplification automatisée: <http://www.32x8.com/>

# ... exemple de MSS ...

- Equations des sorties :

	$Y_2$	$Y_1$		
	00	01	11	10
$Y_0$				
0				
1				

R

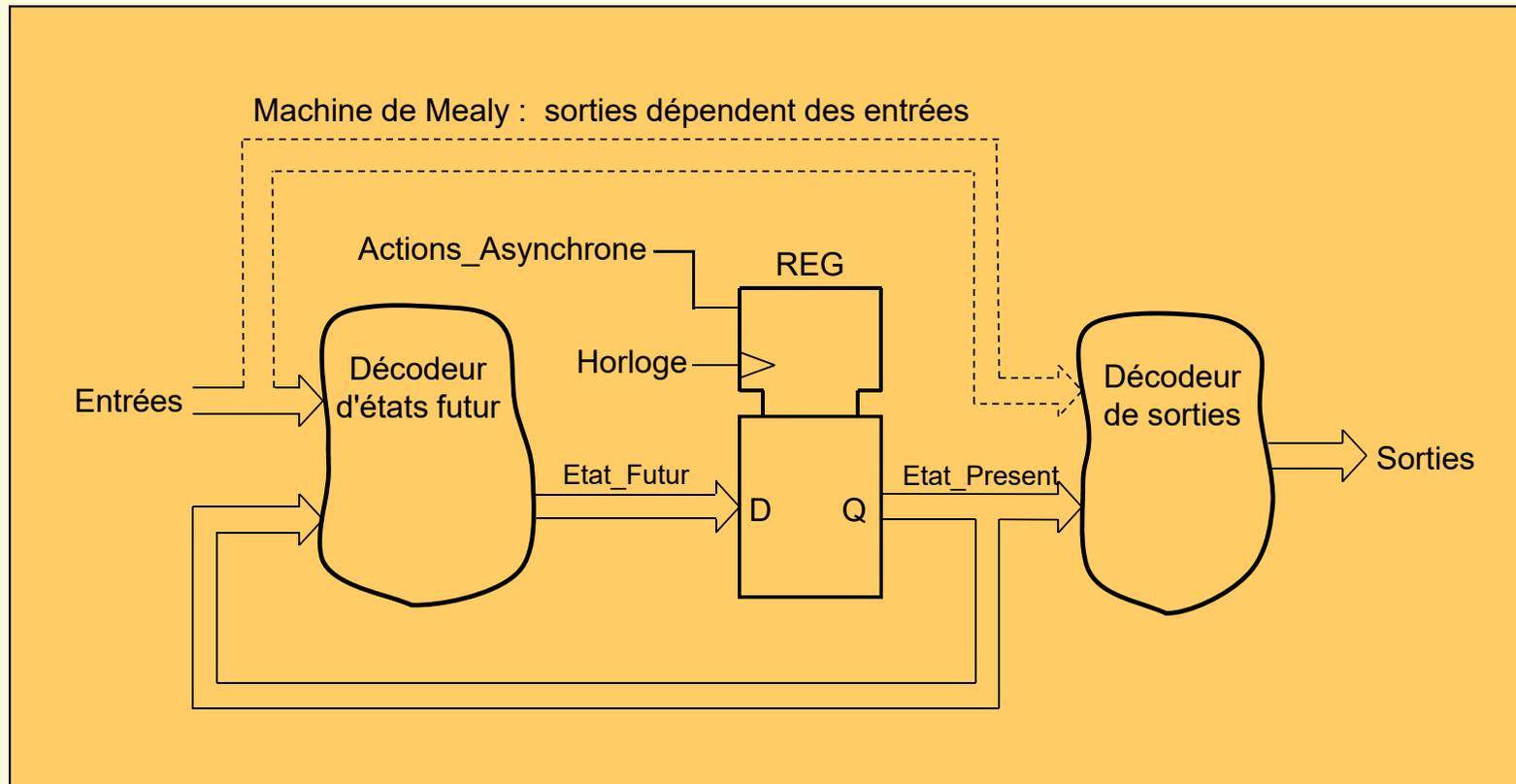
	$Y_2$	$Y_1$		
	00	01	11	10
$Y_0$				
0				
1				

S

	$Y_2$	$Y_1$		
	00	01	11	10
$Y_0$				
0				
1				

I

# Rappel: schéma bloc d'une MSS simple



# Description VHDL machine d'état ...

```
architecture m_etat of exemple is
-- Machine d'etat avec n bits d'etat
  signal etat_pres, etat_fut : std_logic_vector(n downto 0);

--Les constantes permettent de fixer le codage
--Si modification : seul les constantes sont a corriger.
  constant ETAT_0 : std_logic_vector(n downto 0) := "0..00";
  constant ETAT_1 : std_logic_vector(n downto 0) := "0..01";
  "
begin

  Fut: process (Toutes_Les_Entrees, etat_pres)
  begin
    etat_fut <= ETAT_0; --valeur par default
    sortie_A <= '0'; sortie_B <= '0'; --valeur par default
```

## ... description VHDL machine d'état ...

```
case etat_pres is
  when ETAT_0 =>
    sortie_A <= '1'; --sortie de type Moore
    if (condition) then
      etat_fut <= ETAT_0;
      sortie_B <= '1'; --sortie de type Mealy
    else
      etat_fut <= ETAT_1;
      sortie_B <= '0'; --sortie de type Mealy
    end if;
  when ETAT_1 => ...

  "

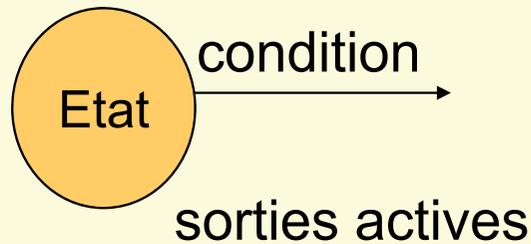
  when others =>
    sortie_A <= '0'; sortie_B <= '0';
    etat_fut <= ETAT_0;
end case;
```

## ... description VHDL machine d'état

```
end process;  
  
Mem: process (clock, reset)  
begin  
    if (reset = '1') then  
        etat_pres <= ETAT_0;  
    elsif rising_edge(clock) then  
        etat_pres <= etat_fut;  
    end if;  
end process;  
  
end m_etat;
```

# Exemple de machine d'état ...

## Convention graphe

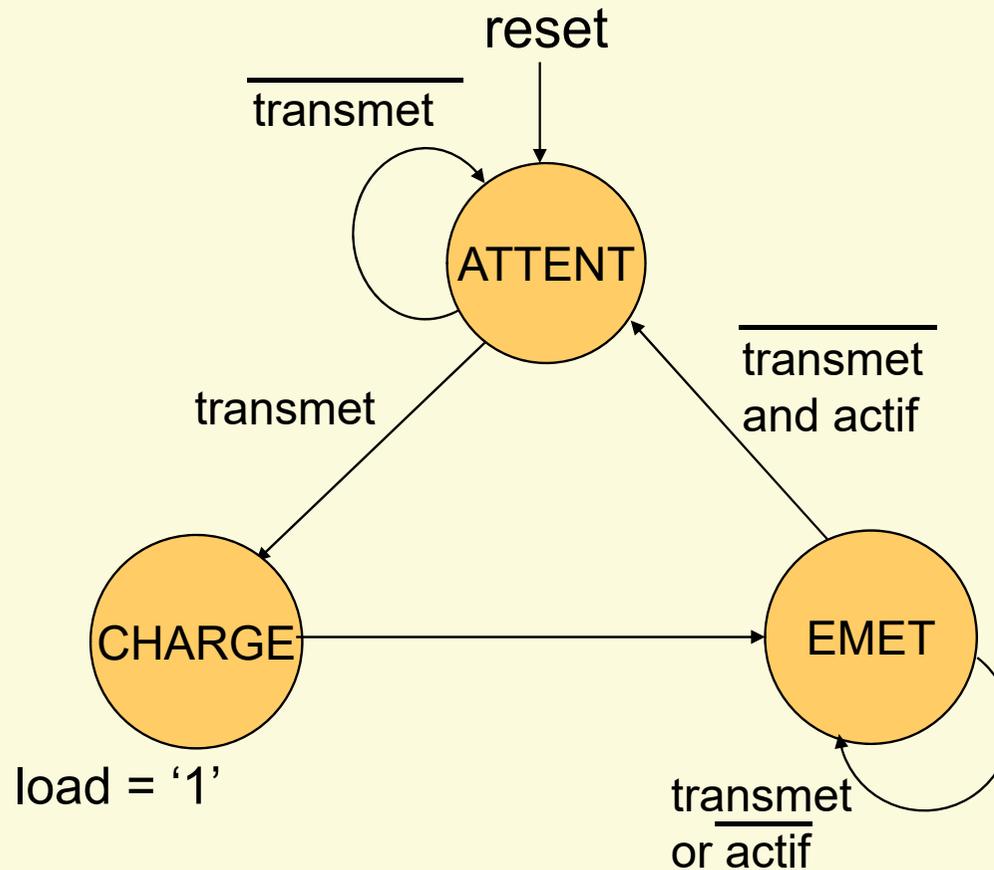


### Entrées:

- transmet, actif

### Sortie:

- load



# ... exemple de machine d'état ...

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity uc_emetteur is
  port ( transmet, actif : in  std_logic;
        clock, reset :    in  std_logic;
        load :           out std_logic);
end uc_emetteur;

architecture m_etat of uc_emetteur is

  signal etat_present, etat_futur :
          std_logic_vector(1 downto 0);

  --Les constantes permettent de fixer le codage
  --Si modification : seul les constantes sont a corriger.
  constant ATTENT : std_logic_vector(1 downto 0) := "00";
  constant CHARGE : std_logic_vector(1 downto 0) := "01";
  constant EMET   : std_logic_vector(1 downto 0) := "10";

begin
```

```
--process combinatoire, calcul etat futur
Fut:process (transmet, actif, etat_present)
begin
  etat_futur <= ATTENT; -- valeur par default
  case etat_present is
    when ATTENT =>
      if transmet = '1' then
        etat_futur <= CHARGE;
      else
        etat_futur <= ATTENT;
      end if;
    when CHARGE =>
      etat_futur <= EMET;
    when EMET =>
      if transmet = '0' and actif = '1' then
        etat_futur <= ATTENT;
      else
        etat_futur <= EMET;
      end if;
    when others =>
      etat_futur <= ATTENT;
  end case;
end process;
```

## ... exemple de machine d'état ...

```
--processus de memorisation
Mem:process (clock, reset)
begin
  if reset = '1' then
    -- reset asynchrone prioritaire
    etat_present <= ATTENT;
  elsif rising_edge(clock) then
    etat_present <= etat_futur;
  end if;
end process;

-- equations combinatoire de sortie
load <= etat_present(0); -- etat_present = CHARGE

end m_etat;
```

# ... exemple de machine d'état

- Utilisation d'un type énuméré pour les états
  - Avantage: les noms des états apparaissent lors de la simulation
  - Inconvénient: codage non maîtrisé

```
...  
architecture m_etat of uc_emetteur is  
  
  type T_etat is (ATTENT,  --attente du start  
                   CHARGE,  --chargement reg  
                   EMET     --transfert en cours  
                   );  
  
  signal etat_present, etat_futur : T_etat  
  
begin  
...  

```