Unité: Systèmes logiques (SysLog1)
Conception de systèmes numériques (CSN)

MSS simples
(Machines Séquentielles Synchrones)

Version pour SysLog2

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

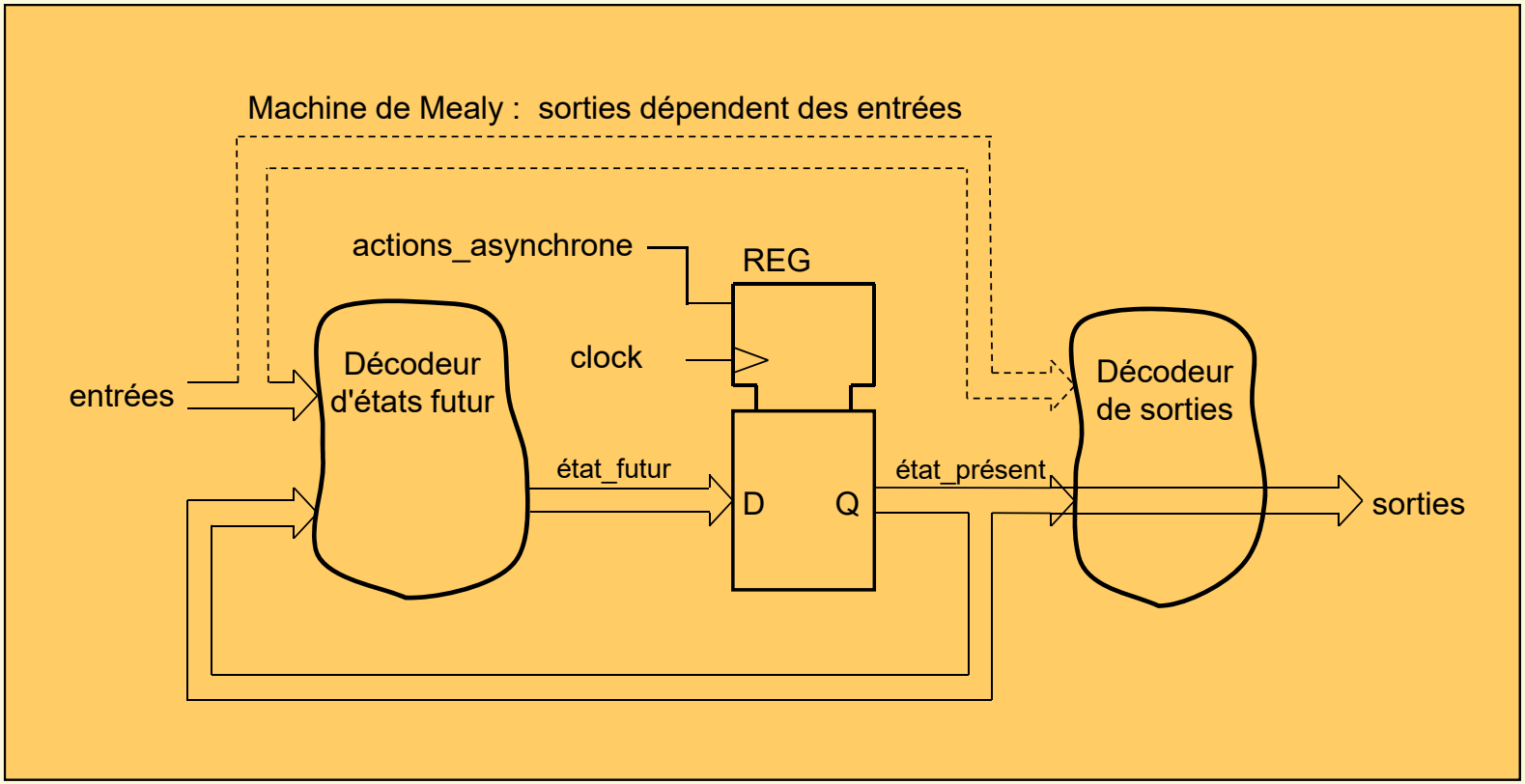
Etienne Messerli
& collègues de l'institut REDS

Novembre 2021 (modifié avril 2022)

Décomposition d'une MSS
(Machine Séquentielle Synchrone)

- Ramener les MSS (machines séquentiels synchrones) à :
  des systèmes combinatoires
  \+ des variables internes (mémoires)
- Eléments 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



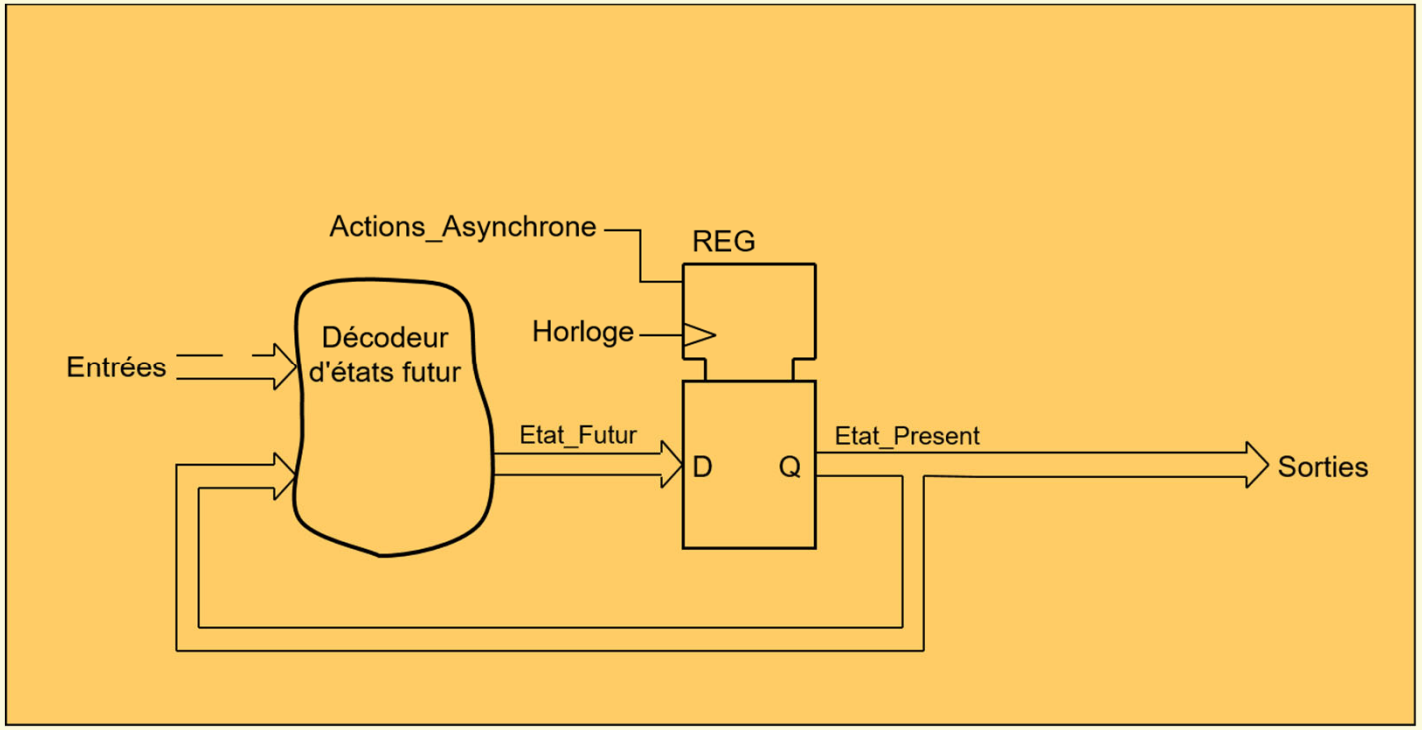
Types de machine d'état (MSS simples)

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

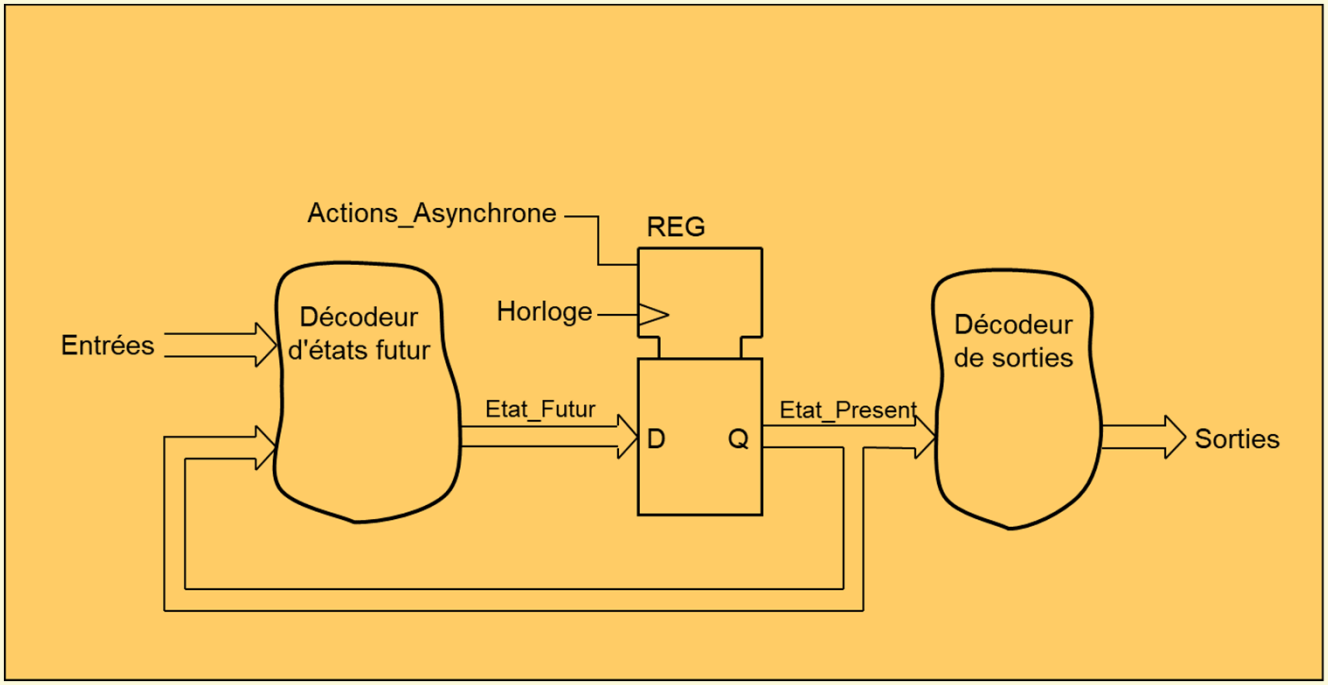
MSS simple de MEDVEDEV

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



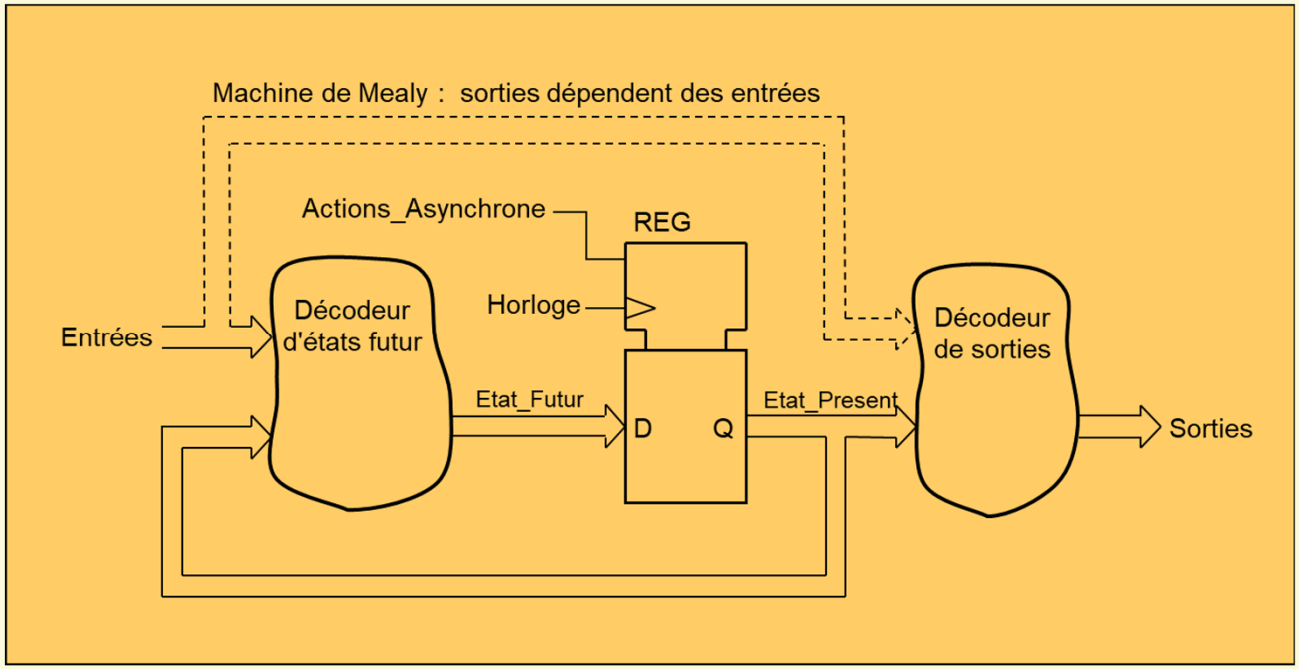
MSS simple de MOORE

- 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



Spécification MSS

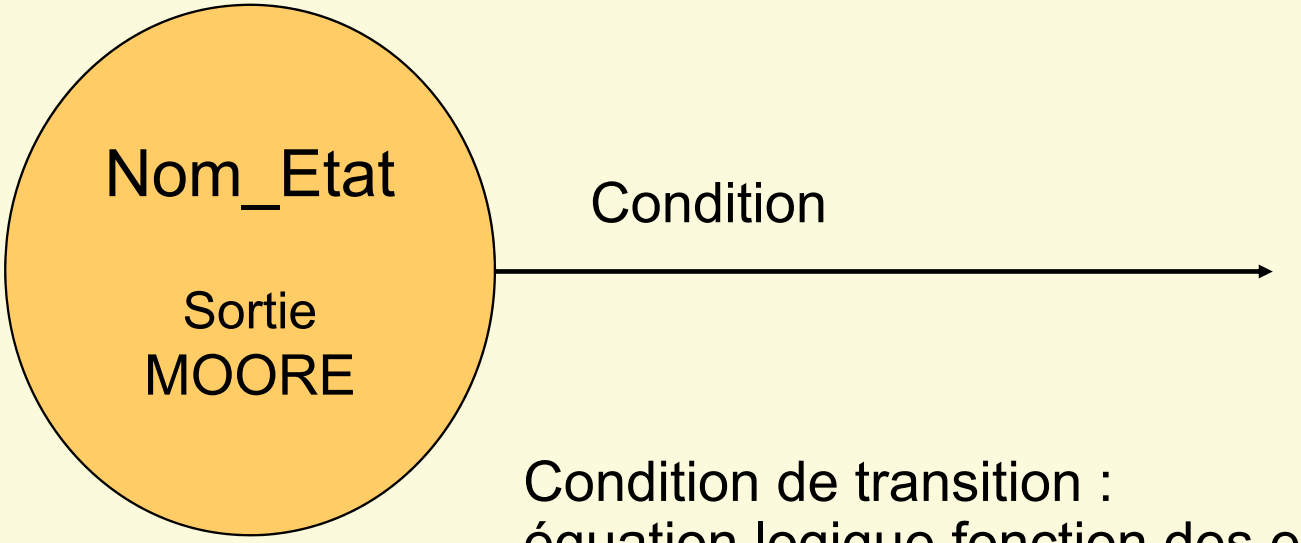
- Chronogramme :
  - présente une seule séquence
  - vue partielle
  - grand nombre de chronogramme
- Graphe des états :
  - méthode graphique
  - permet de représenter tous les "chemins" possibles
  - bien adapté pour des MSS simples

Graphe des états ...

- Constitué d'un 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

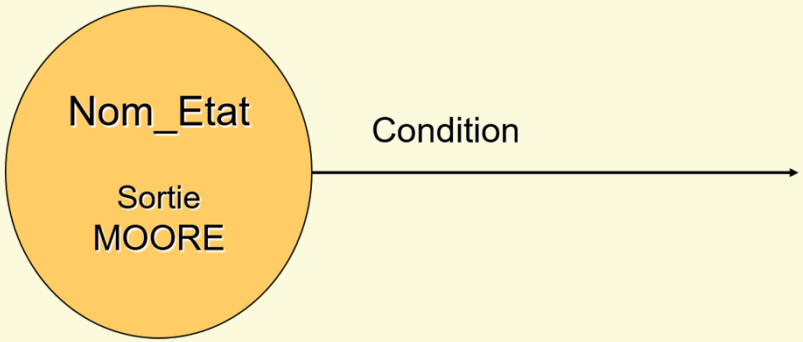
Convention de dessin Moore :

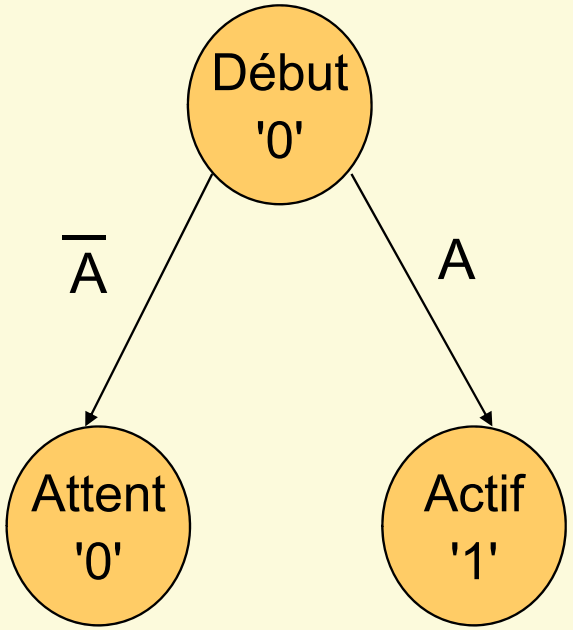


Condition de transition :
équation logique fonction des entrées

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





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:

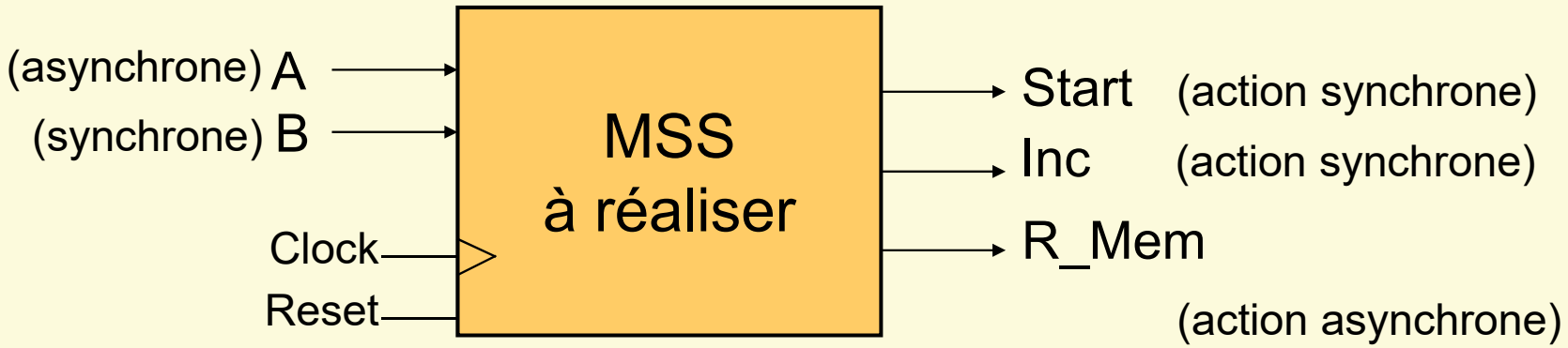
- Appliquer les règles présentées ci-après en suivant une des méthodes indiquées ci-dessus
- Puis: **compléter** le graphe en respectant la règle 1 (ci-après)

Règles construction graphe des états

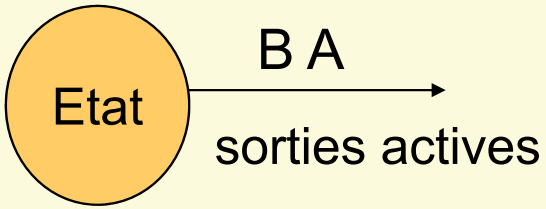
IMPORTANT

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

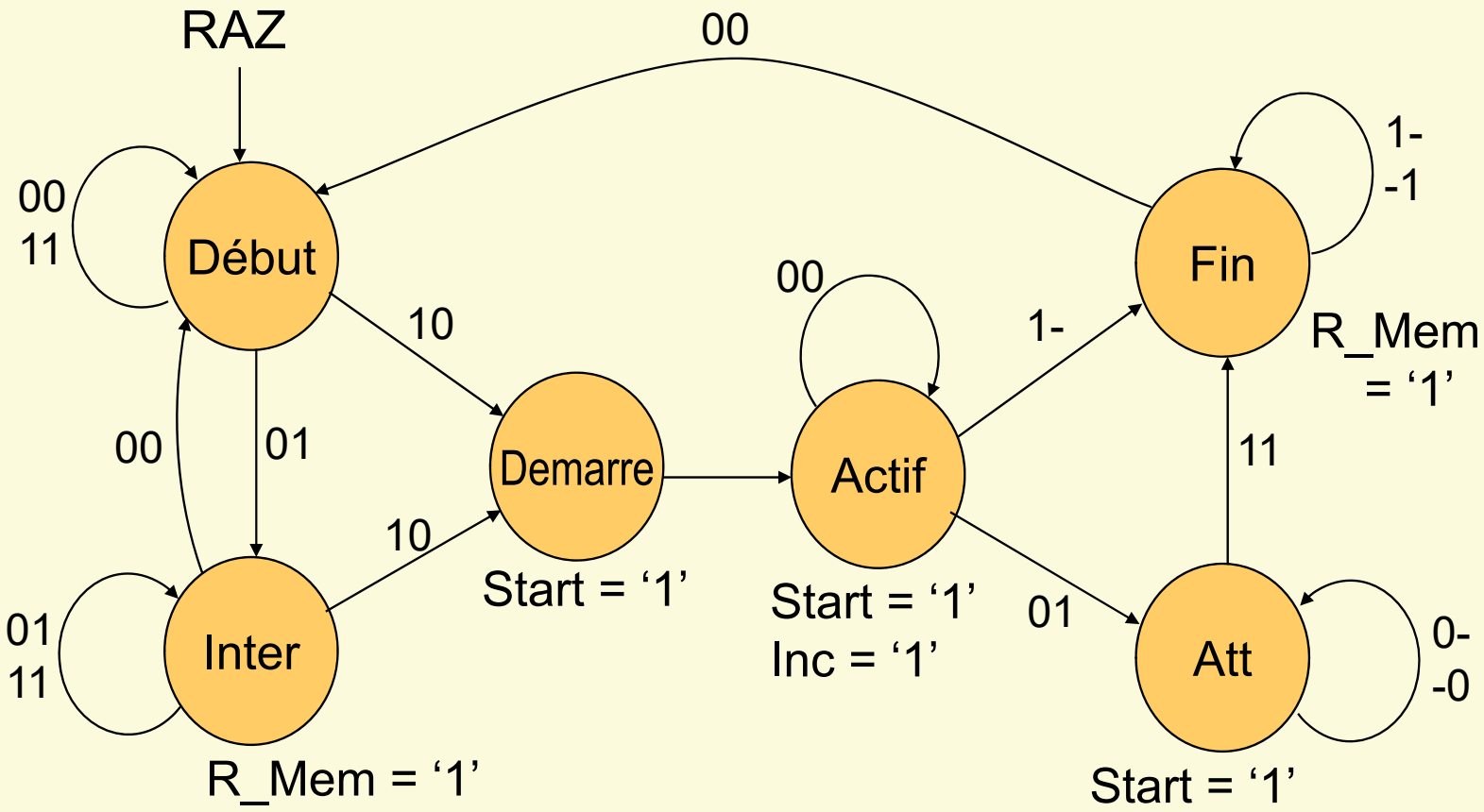
Exemple de MSS…



Convention pour le graphe:



… exemple de MSS… graphe des états



... exemple de MSS : contraintes

Contrainte pour le codage:

- Doit garantir un fonctionnement correct de la MSS

Traitement des entrées :

- A : asynchrone => à synchroniser avec DFF
- B : synchrone => ok

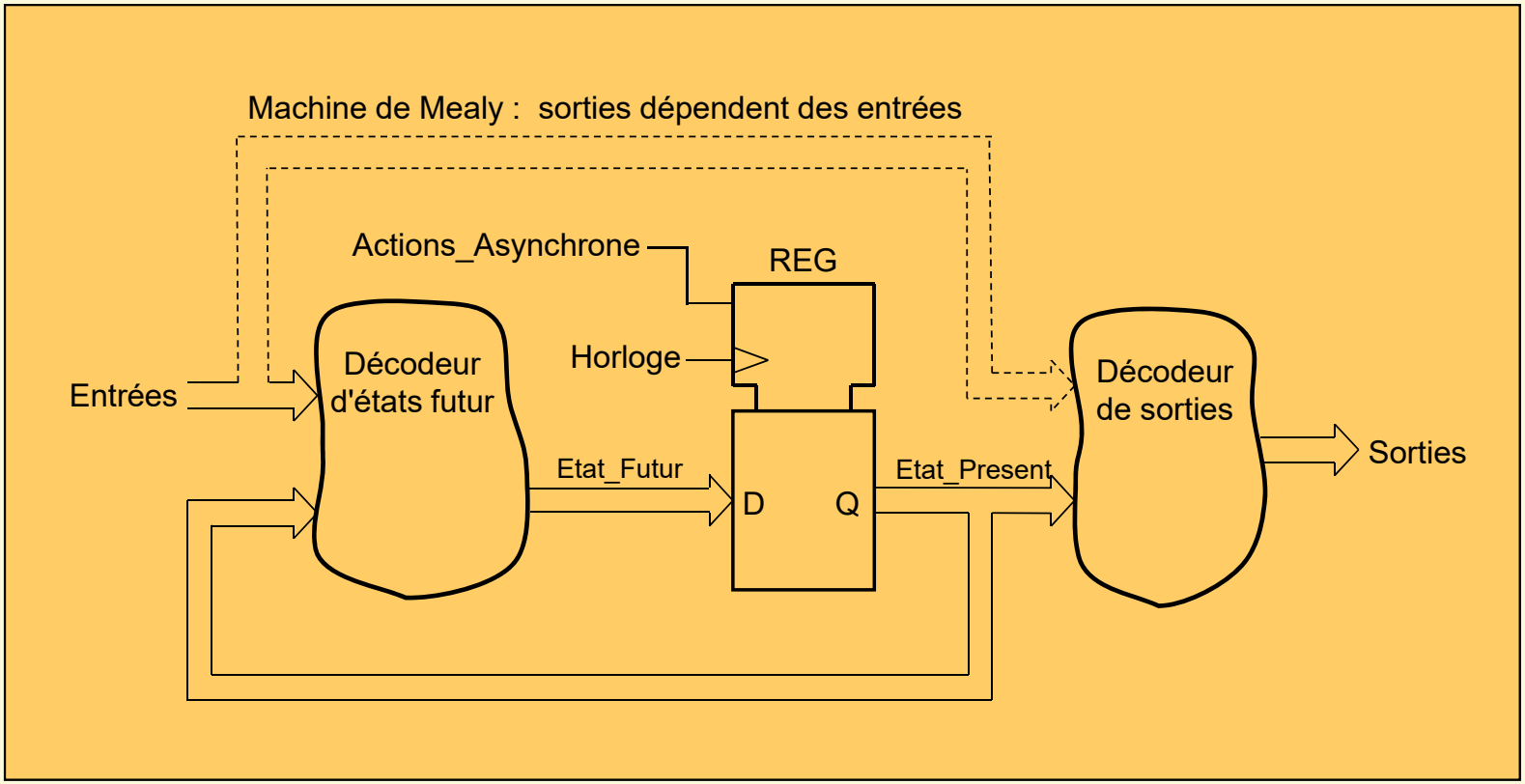
Traitement des sorties :

- Start & Inc : action synchrone => aléas acceptables.
- R_Mem : action asynchrone (type reset) => pas d'aléas
  doit correspondre à un bit d'état (sortie de flip-flop)

Description VHDL machine d'état …

- Comment décrire une machine d'état en VHDL ?
  - Suivre la décomposition d'une MSS simple
    - Permet d'assurer une synthèse fiable
    - Bonne lisibilité
  - Décodeur d'états futur
    - Instruction when … else pas adaptée !
    - Utilisation instruction *process* avec instructions *case / if*
      - permet de décrire aisément le graphe des états

Rappel: schéma bloc d'une MSS simple



… description VHDL machine d'état …

- Décodeur d'états futurs spécifié par le graphe des états
  - Evolution depuis chaque état dépend état présent
    - Utilisation instruction *case*
  - Evolution depuis un état dépende des entrées
    - Utilisation instruction *if*
- Cas générale d'une MSS: Mealy:
  - Décodeur de sorties dépend des mêmes entrées que le décodeurs de sorties
  - Fusionner pour la description en VHDL les 2 décodeurs (futur & sortie)

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 defaut
    sortie_A <= '0'; sortie_B <= '0'; --valeur par defaut
```

… 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

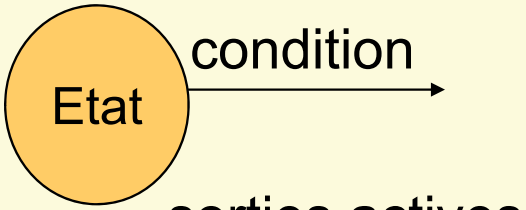
```
  end process;

  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;
```

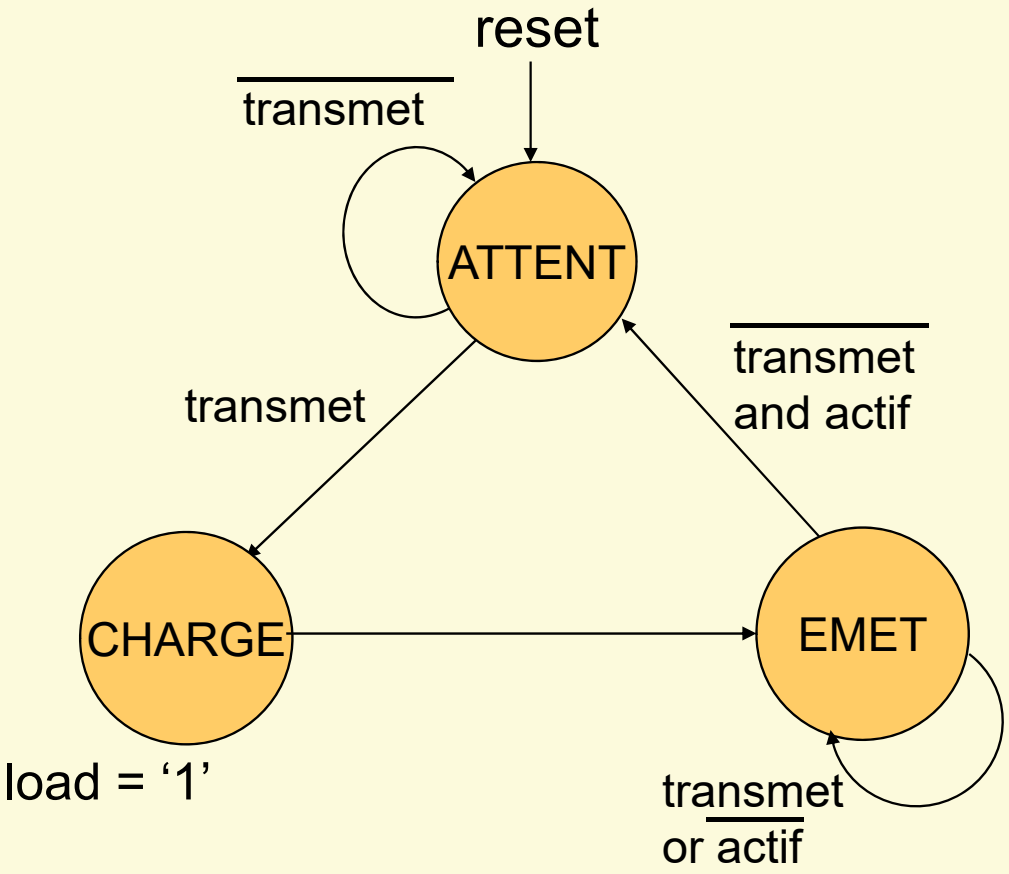
Exemple de machine d'état …

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 defaut
  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 maitrisé

```
…
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
…
```