VHDL Avancé 1 : Machines à états en VHDL

Clément Foucher

▶ **To cite this version:**

Clément Foucher. VHDL Avancé 1 : Machines à états en VHDL. Licence. VHDL Avancé, IUT Paul Sabatier, Toulouse, France. 2024. hal-04399300

HAL Id: hal-04399300
https://hal.science/hal-04399300

Submitted on 17 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - ShareAlike 4.0 International License

VHDL

MACHINES À ÉTATS EN VHDL

© 2024 Clément Foucher

Cours distribué sous licence libre

BUT GEII Toulouse S6 ESE

Rappels : Machines à états

UNE MACHINE À ÉTATS EN ÉLECTRONIQUE NUMÉRIQUE ?

REPRÉSENTATION D'UNE MAE EN VHDL

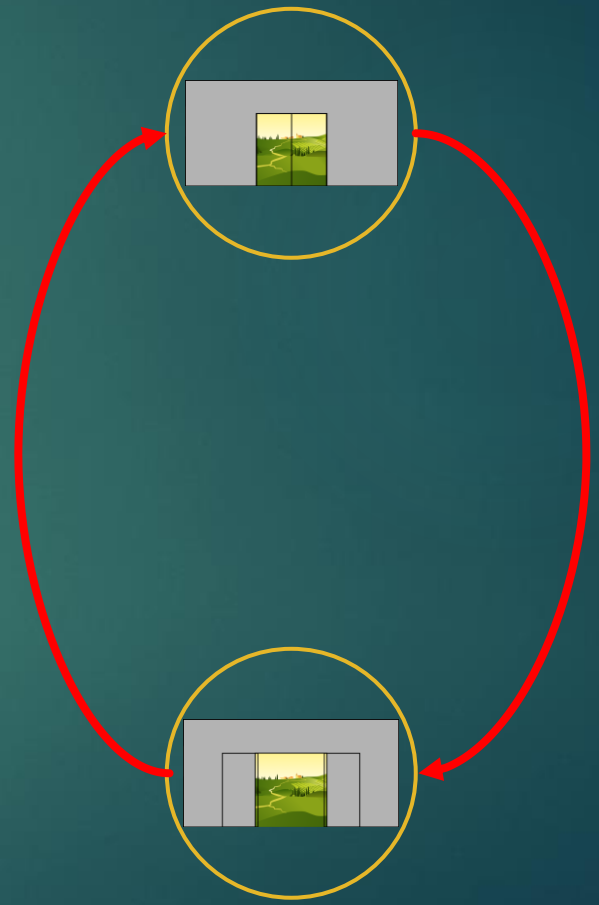
BLOC D'ÉVOLUTION DE L'ÉTAT

CALCUL DES ACTIONS

EXERCICE

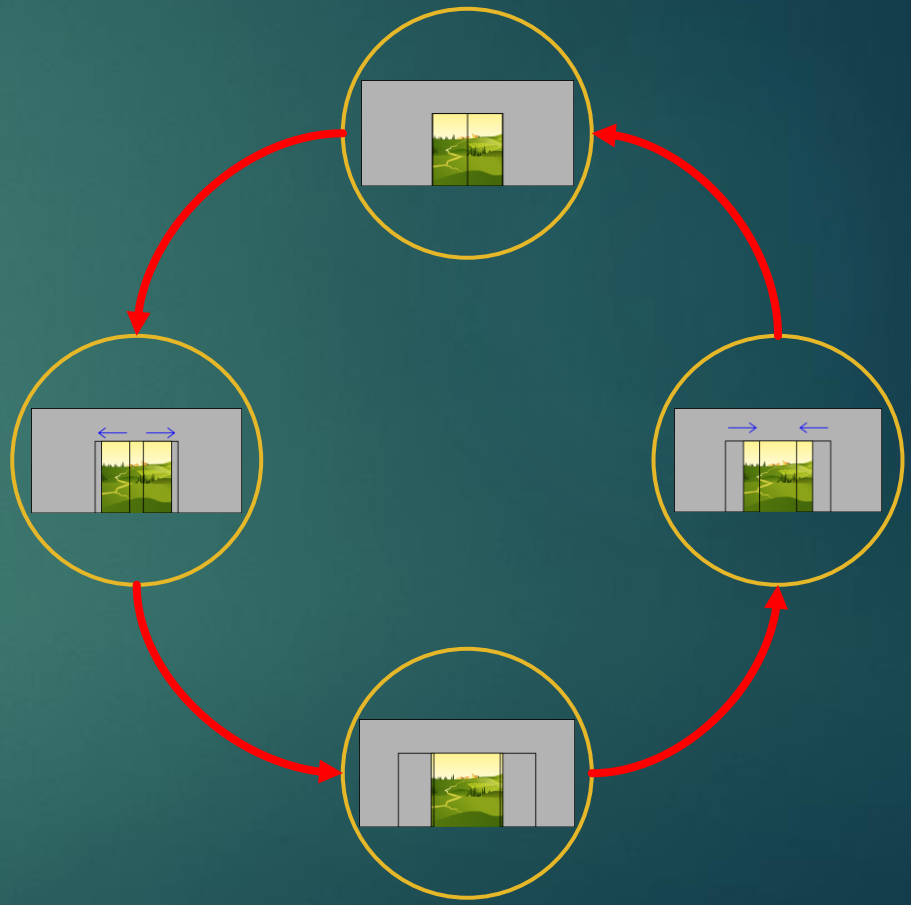
Principe de base d'une machine à états

- Problématique : comment représenter un système de manière simple ?
- Solution : représenter le système sur la base de ses états possibles
- Exemple : système d'ouverture d'une porte coulissante
  - La porte peut être ouverte ou fermée (état)
  - La porte peut changer d'état (transition)



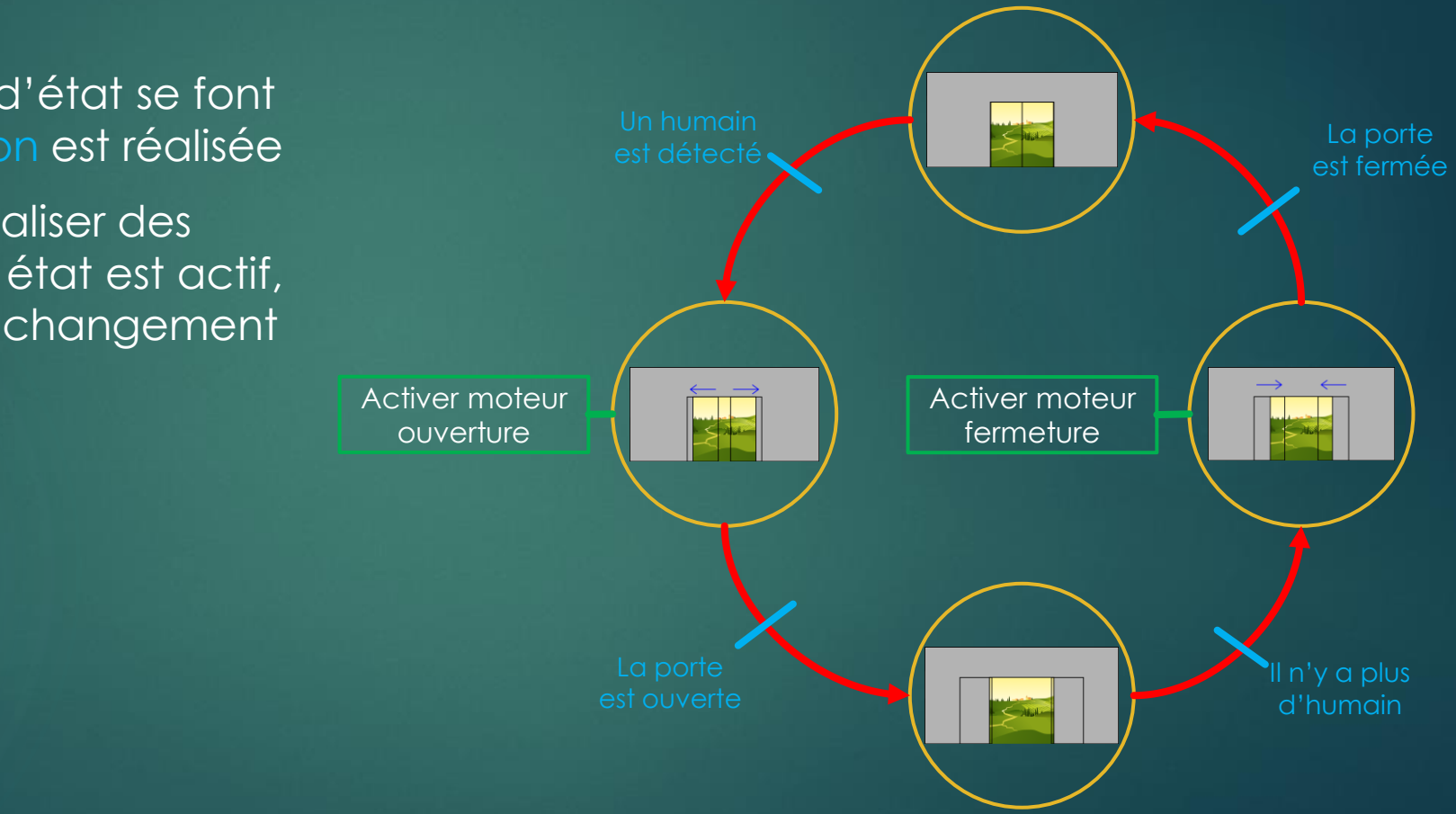
Principe de base d'une machine à états

- Problématique : comment représenter un système de manière simple ?
- Solution : représenter le système sur la base de ses états possibles
- Exemple : système d'ouverture d'une porte coulissante
  - La porte peut être ouverte ou fermée (état)
  - La porte peut changer d'état (transition)
- Un état n'est pas forcément « statique » :
  - Il y a des états intermédiaires avec la porte s'ouvrant ou se fermant



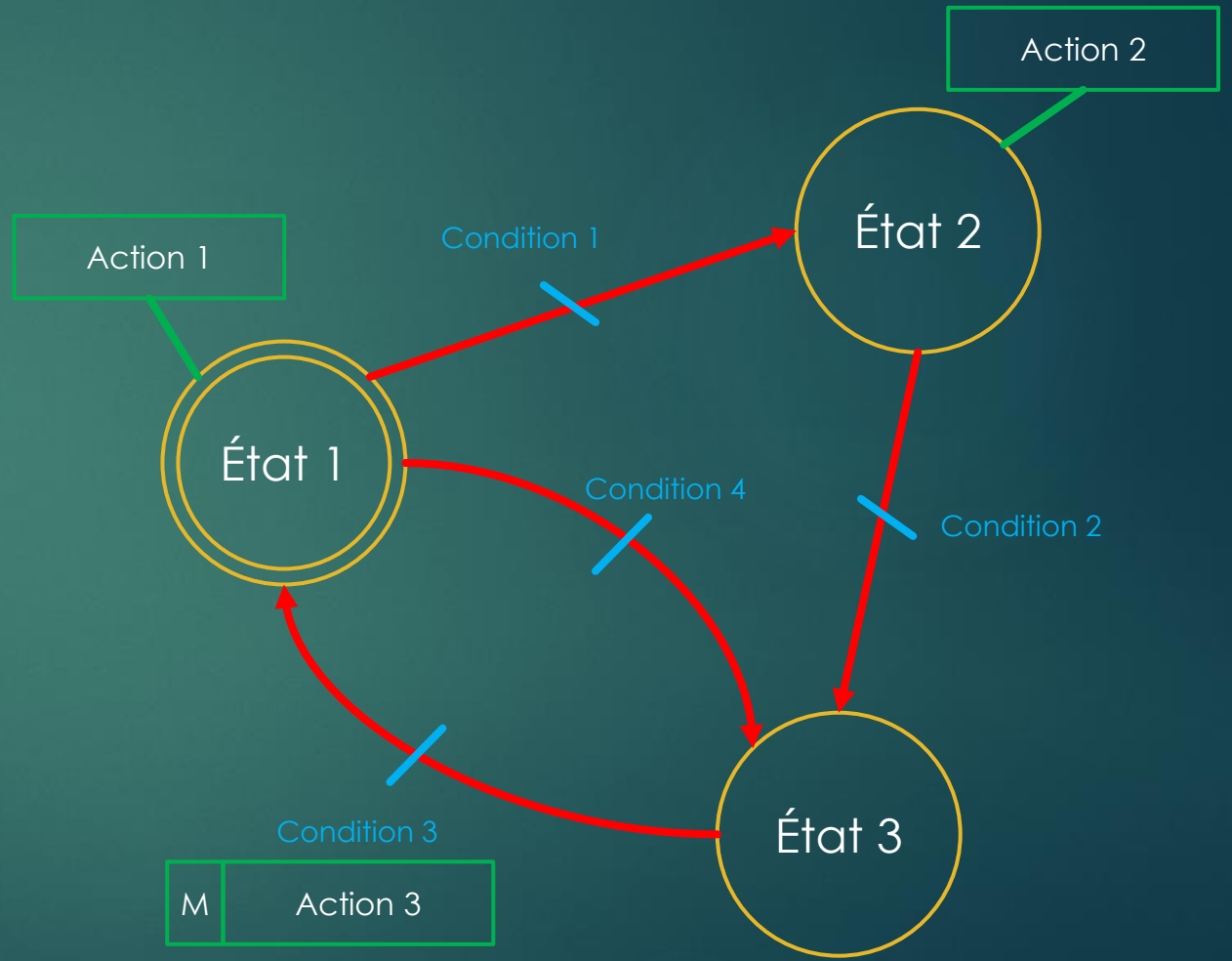
Principe de base d'une machine à états

- Les changements d'état se font lorsqu'une condition est réalisée
- Il est possible de réaliser des actions tant qu'un état est actif, ou au moment du changement d'état



Représentation d'une machine à états

- Ce principe est particulièrement adapté aux systèmes de commande, pour lesquels des capteurs peuvent être utilisés pour les conditions, et des actionneurs pour les actions
- Règles de construction d'une MAE :
  - Un état est représenté par un cercle
    - L'état initial par un double cercle
  - Une transition est représentée par une flèche
  - Une condition est représentée par du texte placé sur une barre placée sur une transition
  - Une action est représentée par du texte placé dans un rectangle placé sur une transition ou un état
    - Si l'action s'applique à une variable mémorisée, on l'indique avec un « M »



Règles d'une machine à états

- États
  - À tout instant, un état et un seul est actif, tous les autres étant inactifs
    - On peut donc dire par abus de langage qu'a tout moment, la MAE « est » dans un état (celui qui est actif)
  - L'un des états doit être défini comme « initial », c'est-à dire que c'est lui qui sera activé au démarrage ou en cas de réinitialisation du système
- Transitions
  - Une transition doit être dessinée entre 2 états : un état « source » et un état « cible »
    - Il est possible d'avoir une transition dont l'état source et l'état cible sont confondus
  - Toute transition doit être associée à une et une seule condition
- Conditions
  - Une condition s'exprime sous la forme d'une opération booléenne dont le résultat est soit « vrai » soit « faux »
  - Les conditions associées à plusieurs transitions ayant le même état source doivent être mutuellement exclusives
- Actions
  - Une action se présente sous la forme d'une affectation de variable
  - Une action peut être placée sur une transition : elle est alors ponctuelle (active brièvement, au moment du changement d'état)
  - Une action peut être placée sur un état : elle est alors maintenue (active tant que l'état est actif, se désactive lorsque l'on sort de l'état)

Variables

- 3 types de variables :
  - Variable d'entrée
    - Peut être lue, ne peut pas être modifiée
  - Variable interne
    - Peut être lue et modifiée
  - Variable de sortie
    - Peut être modifiée, ne peut pas être lue

- Une variable modifiable peut être mémorisée ou non mémorisée
  - Une variable mémorisée a une valeur initiale, et conserve sa nouvelle valeur après affectation
  - Une variable non mémorisée a une valeur par défaut et reprend cette valeur lorsque l'affectation est terminée

RAPPELS : MACHINES À ÉTATS

Une machine à états en électronique numérique ?

REPRÉSENTATION D'UNE MAE EN VHDL

BLOC D'ÉVOLUTION DE L'ÉTAT

CALCUL DES ACTIONS

EXERCICE

Comment réaliser une MAE avec des concepts numériques ?

- En électronique numérique, il faut réfléchir en termes de cycles d'horloges
- Changement d'état
  - Si une condition pour le changement d'état est vraie, alors le changement aura lieu au prochain front d'horloge
    - Il faut donc « préparer » le prochain état pour qu'il soit prêt à être modifié au prochain front d'horloge
    - Il est donc nécessaire que le calcul du prochain état puisse être fait en dehors des fronts, pour être prêt au prochain front
    - Le calcul doit donc être fait de manière combinatoire, et mémorisé dans un registre au front d'horloge

Comment réaliser une MAE avec des concepts numériques ?

- En électronique numérique, il faut réfléchir en termes de cycles d'horloges
- Actions sur état
  - Les actions sur état doivent être réalisées dès que l'on est dans l'état
  - Or, la mémorisation de l'état se fait sur front d'horloge
  - Les actions sur état doivent donc s'activer juste après la mémorisation de l'état, qui a lieu immédiatement après le front d'horloge
    - Deux solutions imaginables
      - Actions séquentielles : seront réalisées au prochain front d'horloge
      - Actions combinatoires : seront réalisées immédiatement après le changement d'état
  - Les actions sur état séquentielles auront donc un cycle de retard sur le changement d'état
    - Dans la mesure du possible, les actions sur état doivent donc être calculées de manière combinatoire

Comment réaliser une MAE avec des concepts numériques ?

- En électronique numérique, il faut réfléchir en termes de cycles d'horloges
- Actions sur transition
  - Les actions sur transition doivent être réalisées *au moment* du changement d'état
  - Si il y a changement d'état, l'action sur transition associée doit donc être active sur le front où la transition se produit
  - L'action doit donc avoir été préparée *avant* le front pour pouvoir être active au moment du front
    - Dans la mesure du possible, les actions sur transition doivent donc être calculées de manière combinatoire

Cas des actions mémorisées

- Les actions mémorisées sont un cas particulier
- Une mémoire est forcément synchrone
  - Sauf à utiliser des *latches*, concept à bannir
- Donc une action mémorisée est forcément synchrone, et changera donc de valeur *après* un front d'horloge
- Sauf à utiliser des constructions complexes, une action mémorisée aura donc un cycle de retard sur une action non mémorisée
  - Une action sur transition mémorisée se produira *après* le front d'horloge, contrairement aux autres actions sur transition
  - Une action sur état mémorisée ne se déclenchera qu'un cycle *après* que l'on ait changé d'état

Machines de Moore et de Mealy

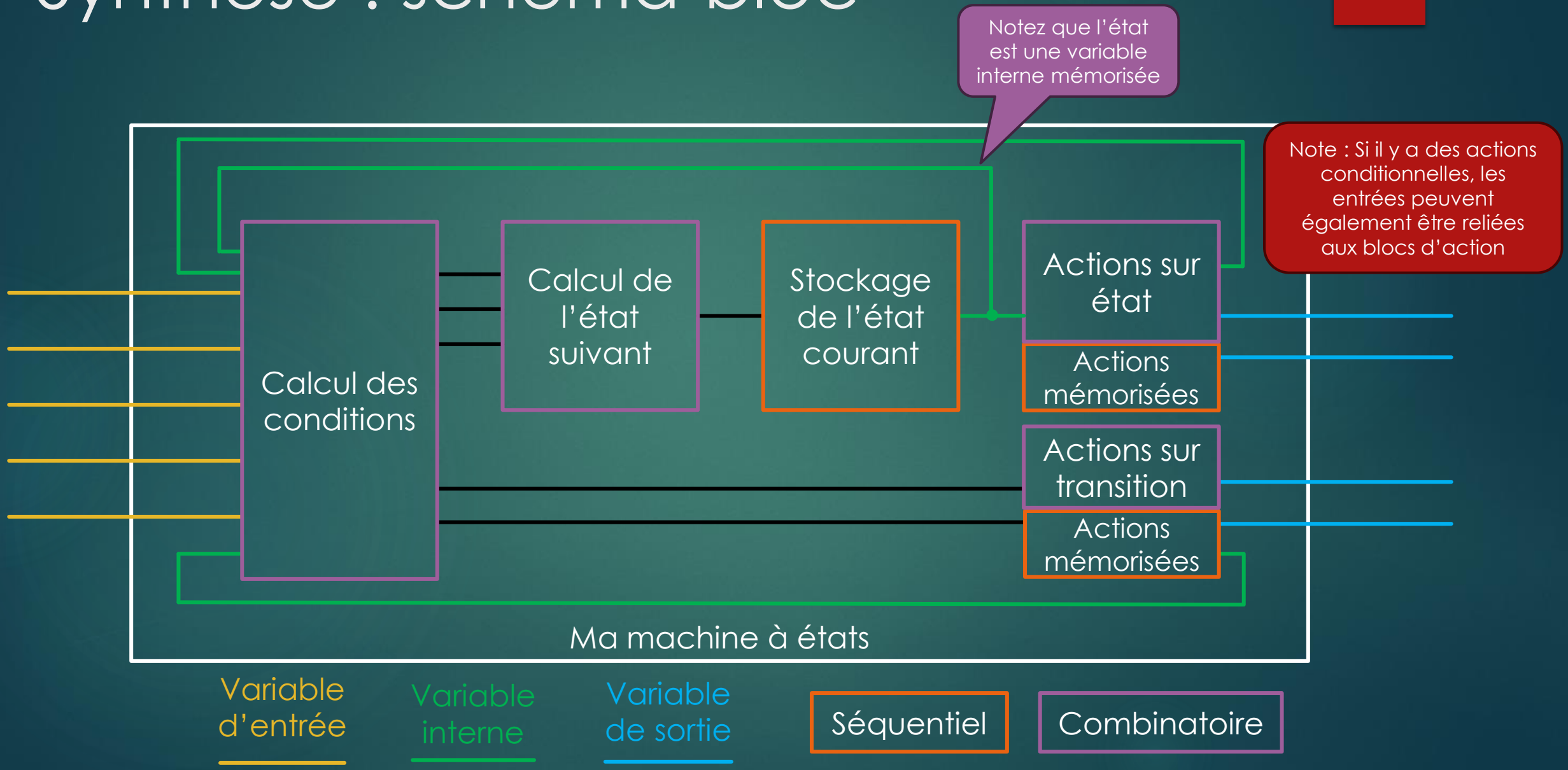
L'icône d'information indique que le contenu de ce slide est à titre informatif, il ne vous est pas demandé de le retenir

- Historiquement, il existe deux grands type de machines à états
  - Dans une machine de Moore, les sorties sont définies en fonction de l'état
    - Uniquement des actions sur état, donc
  - Dans une machine de Mealy, les sorties sont définies en fonction de l'état et des entrées
    - Uniquement des actions sur transition, donc
    - Notez que rester dans le même état peut être vu comme une transition dans laquelle l'état cible est identique à l'état source
- Dans le deux cas, les variables sont non mémorisées, et les actions réalisées de manière asynchrone
- La vision présentée précédemment combine donc les deux : machine de Mealy pour les actions sur transition, machine de Moore pour les actions sur état
  - On y ajoute également la notion de variable mémorisée
- Mais attention ! Les actions sur transition pouvant évoluer de manière asynchrone, il faut s'assurer qu'à l'extérieur de la MAE, les sorties ne soient pas rebouclées sur les entrées de manière combinatoire, sinon attention aux dégâts…
  - Ce cas reste rare, il est donc relativement sûr d'utiliser ce type de machine

Pour aller plus loin…
Moore machine
Mealy machine

Information icon4.svg © srittau – Public domain

Synthèse : schéma-bloc



RAPPELS : MACHINES À ÉTATS

UNE MACHINE À ÉTATS EN ÉLECTRONIQUE NUMÉRIQUE ?

Représentation d'une MAE en VHDL

BLOC D'ÉVOLUTION DE L'ÉTAT

CALCUL DES ACTIONS

EXERCICE

Représenter une MAE en VHDL

- Pour rappel, en VHDL :
  - Un bloc séquentiel est *obligatoirement* représenté par un process synchrone
  - Un bloc combinatoire peut être représenté par du code concurrent ou un process asynchrone
- Rappel liste de sensibilité
  - Un process synchrone a dans sa liste de sensibilité *uniquement* l'horloge et le reset
  - Un process asynchrone a dans sa liste de sensibilité *tous* les signaux dont la valeur est lue dans le process

Point de syntaxe VHDL : les types personnalisés

- Il est possible de définir des types avec la syntaxe suivante :

```
type monType is (valeur1, valeur2, valeur3);
```

  - Équivalent du « typedef enum » en C/C++
- Une fois le type défini, il est possible de créer des signaux utilisant ce type
- Par exemple :

```
architecture ar of compo is
    type monType is (ouvert, ferme);
    signal monSignal : monType;
begin
    monSignal <= ouvert when condition else ferme;
end architecture;
```

Équivalence des notions en VHDL

- État
  - Un état sera défini par un signal de type personnalisé
- Variable
  - Une variable sera définie par un signal
    - Signal « in » dans l'entity pour une variable d'entrée
    - Signal interne pour une variable interne
    - Signal « out » dans l'entity pour une variable de sortie

RAPPELS : MACHINES À ÉTATS

UNE MACHINE À ÉTATS EN ÉLECTRONIQUE NUMÉRIQUE ?

REPRÉSENTATION D'UNE MAE EN VHDL

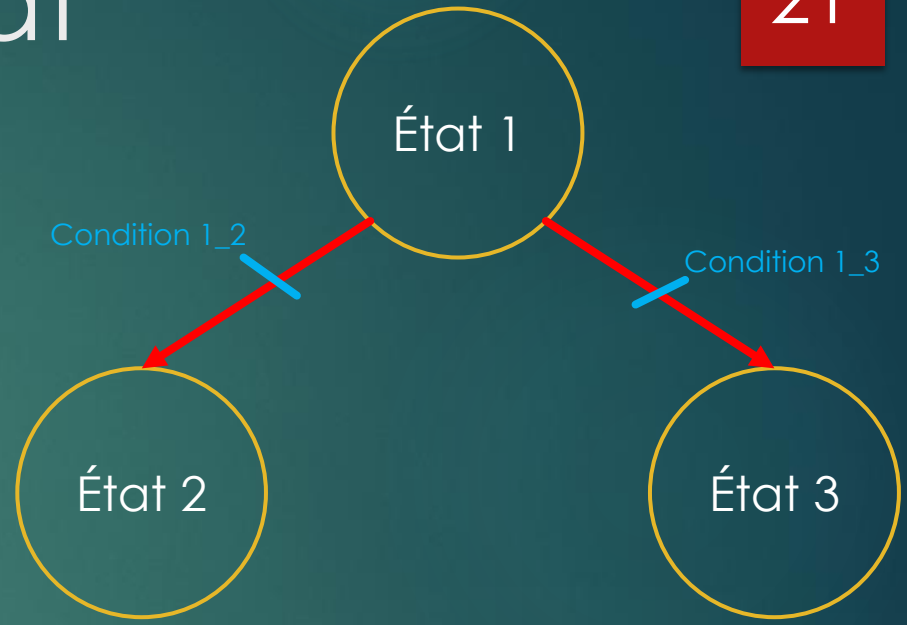
Bloc d'évolution de l'état

CALCUL DES ACTIONS

EXERCICE

Bloc d'évolution de l'état

- Le prochain état est défini par
  - L'état actuel
  - Les valeurs des conditions associées aux transitions sortant de l'état
- Si l'état 1 est actif, alors
  - Si la condition 1_2 est vraie, on passe à l'état 2
  - Si la condition 1_3 est vraie, on passe à l'état 3
  - Sinon, on reste dans l'état 1
- Solution naïve de codage

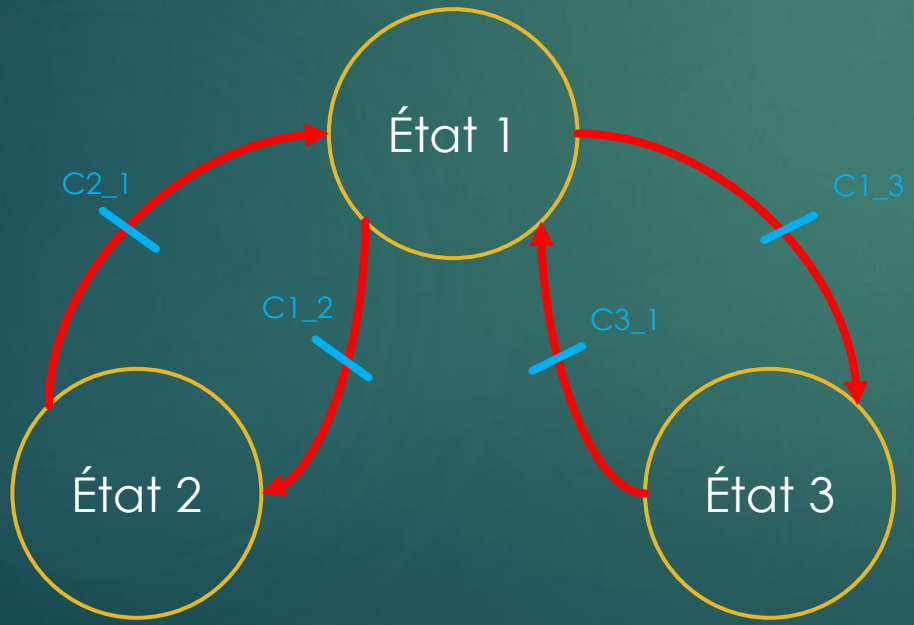



```
if etat_courant = etat_1 then
    if condition_1_2 then
        etat_courant <= etat_2;
    elsif condition_1_3 then
        etat_courant <= etat_3;
    end if;
end if;
```

Rappel : dans un process synchrone, si on ne modifie pas la valeur d'un signal, alors elle conserve sa valeur précédente

Bloc d'évolution de l'état : Exemple

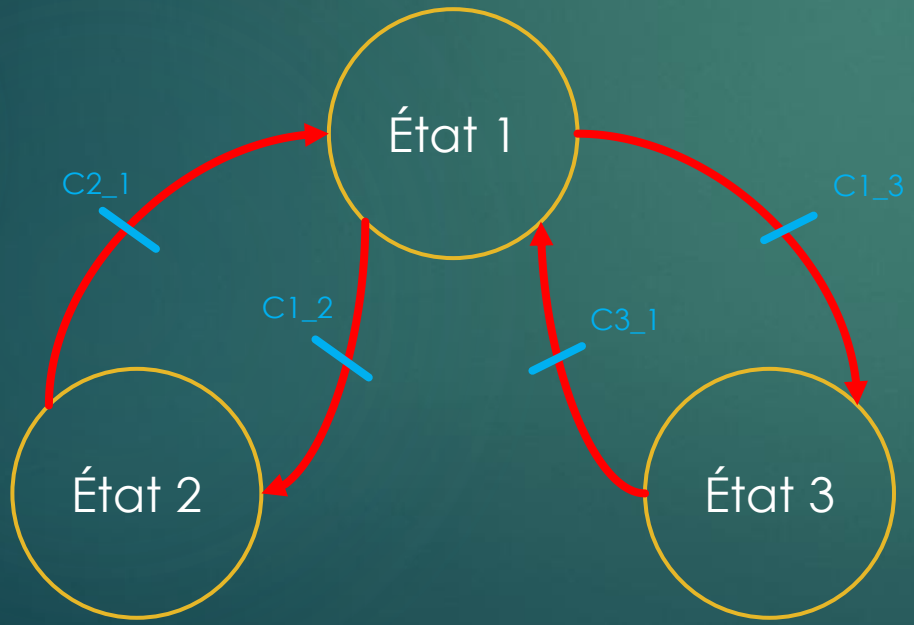
- Exemple de codage complet avec un `if` pour la MAE ci-dessous




```
if etat_courant = etat_1 then
    if C1_2 = '1' then
        etat_courant <= etat_2;
    elsif C1_3 = '1' then
        etat_courant <= etat_3;
    end if;
elsif etat_courant = etat_2 then
    if C2_1 = '1' then
        etat_courant <= etat_1;
    end if;
elsif etat_courant = etat_3 then
    if C3_1 = '1' then
        etat_courant <= etat_1;
    end if;
end if;
```

Bloc d'évolution de l'état : Version améliorée

- Utiliser un case permet d'éviter de répéter `etat_courant`



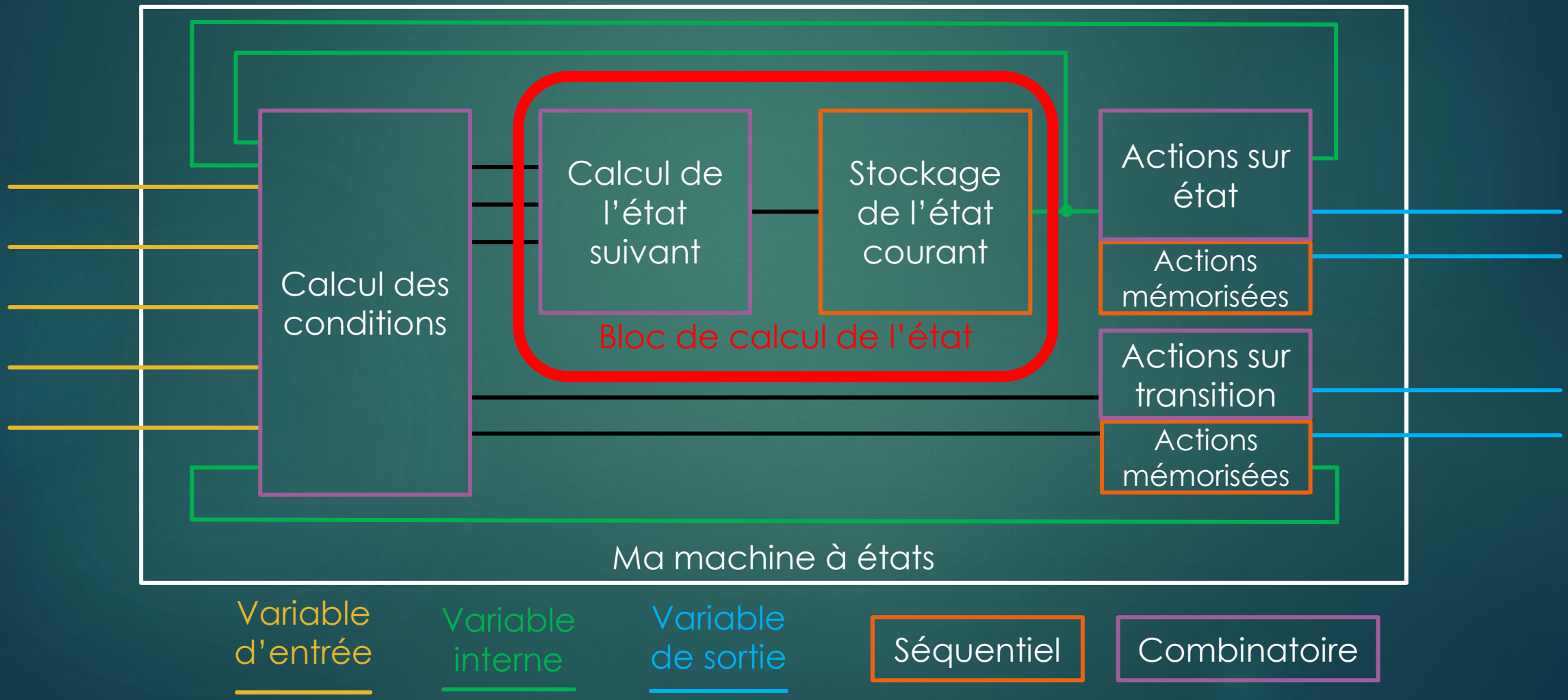

```
case etat_courant is
    when etat_1 =>
        if C1_2 = '1' then
            etat_courant <= etat_2;
        elsif C1_3 = '1' then
            etat_courant <= etat_3;
        end if;
    when etat_2 =>
        if C2_1 = '1' then
            etat_courant <= etat_1;
        end if;
    when etat_3 =>
        if C3_1 = '1' then
            etat_courant <= etat_1;
        end if;
end case;
```

Bloc d'évolution de l'état : Process complet

- On place l'ensemble dans un process synchrone
- Rappels
  - La liste de sensibilité d'un process synchrone contient *uniquement* l'horloge et le reset
  - Le reset est prioritaire, et doit donc être testé en premier

```
process(clk, reset)
begin
    if reset = '1' then
        etat_courant <= etat_initial;
    elsif rising_edge(clk) then
        case etat_courant is
            when etat_1 =>
                if C1_2 = '1' then
                    etat_courant <= etat_2;
                elsif C1_3 = '1' then
                    etat_courant <= etat_3;
                end if;
            when etat_2 =>
                if C2_1 = '1' then
                    etat_courant <= etat_1;
                end if;
            when etat_3 =>
                if C3_1 = '1' then
                    etat_courant <= etat_1;
                end if;
        end case;
    end if;
end process;
```

Synthèse : schéma-bloc



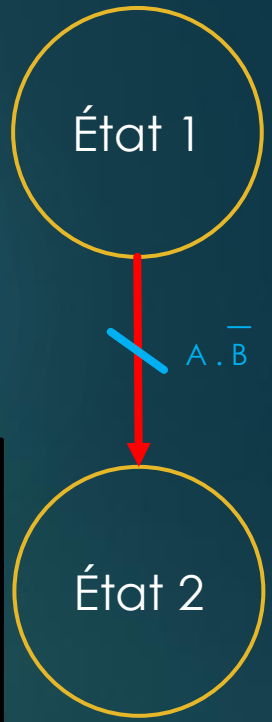
Comment coder les conditions ?

- Soit on crée un signal pour chaque condition, comme dans les slides précédents
  - Avantages
    - Code mutualisé avec les actions sur transition (voir plus loin)
    - Lisibilité accrue du code

```
architecture ar of mon_compo is
     signal C1_2 : std_logic;
begin

     C1_2  <= A and not B;
     process( clk, reset )
     begin
          […]
          case etat_courant is
               when etat_1 =>
                    if C1_2 = '1' then
                         etat_suivant <= etat_2;
                    end if;
          end case;
          […]
     end process;
end architecture;
```

- Soit on exprime les conditions directement dans le process
  - Avantage : moins de signaux à déclarer
  - Si pas d'actions sur transition, plus compact

```
architecture ar of mon_compo is

begin


     process( clk, reset )
     begin
          […]
          case etat_courant is
               when etat_1 =>
                    if A = '1' and B = '0' then
                         etat_suivant <= etat_2;
                    end if;
          end case;
          […]
     end process;
end architecture;
```



RAPPELS : MACHINES À ÉTATS

UNE MACHINE À ÉTATS EN ÉLECTRONIQUE NUMÉRIQUE ?

REPRÉSENTATION D'UNE MAE EN VHDL

BLOC D'ÉVOLUTION DE L'ÉTAT

Calcul des actions

EXERCICE

Différentes méthodes pour le calcul des actions

- Les actions mémorisées et non mémorisées sont forcément traitées séparément
  - Actions mémorisées dans un process synchrone
  - Actions non mémorisées dans un process asynchrone ou avec du code concurrent
- Les actions sur transition et sur état peuvent être traitées conjointement ou de manière séparée

Actions non mémorisées sur état

- En utilisant un process

```
process( etat_courant )
begin
    mon_action1 <= valeur_par_defaut1;
    mon_action2 <= valeur_par_defaut2;

    case etat_courant is
        when etat_1 =>
            mon_action1 <= valeur_1;
        when etat_2 =>
            mon_action1 <= valeur_2;
            mon_action2 <= valeur_3;
    end case;
end process;
```

- En utilisant du code concurrent

```
with etat_courant select mon_action1 <=
     valeur_1            when etat_1
     valeur_2            when etat_2,
     valeur_par_defaut1 when others;

with etat_courant select mon_action2 <=
     valeur_3            when etat_2,
     valeur_par_defaut2 when others;
```

Attention, avec du code concurrent, on traite action par action, et non état par état

Actions non mémorisées sur transition

- En utilisant un process

```
process( etat_courant, C1_2, C2_1 )
begin
    mon_action1 <= valeur_par_defaut1;
    mon_action2 <= valeur_par_defaut2;

    case etat_courant is
        when etat_1 =>
            if C1_2 = '1' then
                mon_action1 <= valeur_1;
            end if;
        when etat_2 =>
            if C2_1 = '1' then
                mon_action2 <= valeur_2;
            end if;
    end case;
end process;
```

- En utilisant du code concurrent

```
mon_action1 <=
    valeur_1 when etat_courant = etat_1 and C1_2 = '1' else
    valeur_par_defaut1;

mon_action2 <=
    valeur_2 when etat_courant = etat_2 and C2_1 = '1' else
    valeur_par_defaut2;
```

Nécessite d'avoir défini des signaux intermédiaires pour les conditions, sinon il faut mettre la condition complète

Actions non mémorisées en un seul bloc

- Il suffit de combiner les deux
- En utilisant un process
  - Faire un seul process, qui contient un seul case
  - Les actions sur état apparaissent directement après le `when`
  - Les actions sur transition sont en plus protégées par un `if`
  - Attention à la liste de sensibilité !

Actions mémorisées

- Les actions mémorisées sont forcément gérées dans un process synchrone
- Comme pour les actions non mémorisées, on peut également faire deux process indépendants pour les actions sur état et sur transition

```
process( clk, reset )
begin

    if reset = '1' then
        mon_action1 <= valeur_initiale1;
        mon_action2 <= valeur_initiale2;
    elsif rising_edge(clk) then
        case etat_courant is
            when etat_1 =>
                -- Action sur état
                mon_action1 <= valeur_1;
            when etat_2 =>
                -- Action sur transition
                if C2_1 = '1' then
                    mon_action2 <= valeur_2;
                end if;
        end case;
    end if;

end process;
```

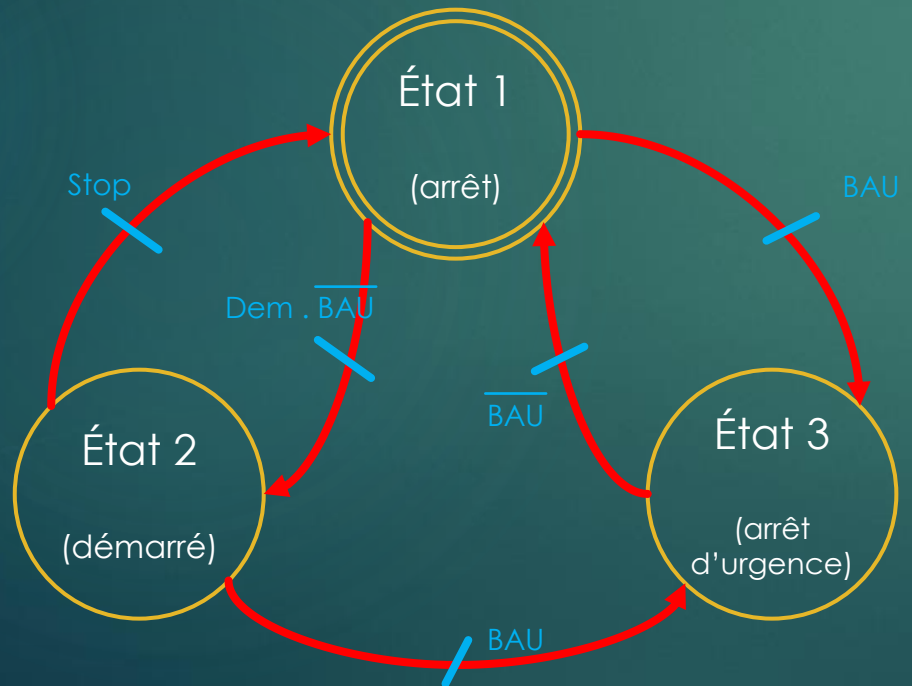
Actions conditionnelles

- Et les actions conditionnelles dans tout ça ?
- Il suffit d'ajouter la condition de l'action
- Dans un process
  - Ajouter un `if` pour les actions sur état
  - Ajouter la condition au `if` existant pour les actions sur transition
- En combinatoire
  - Il faut forcément utiliser un `when … else`, même pour les actions sur état

Exercice

Exercice d'application

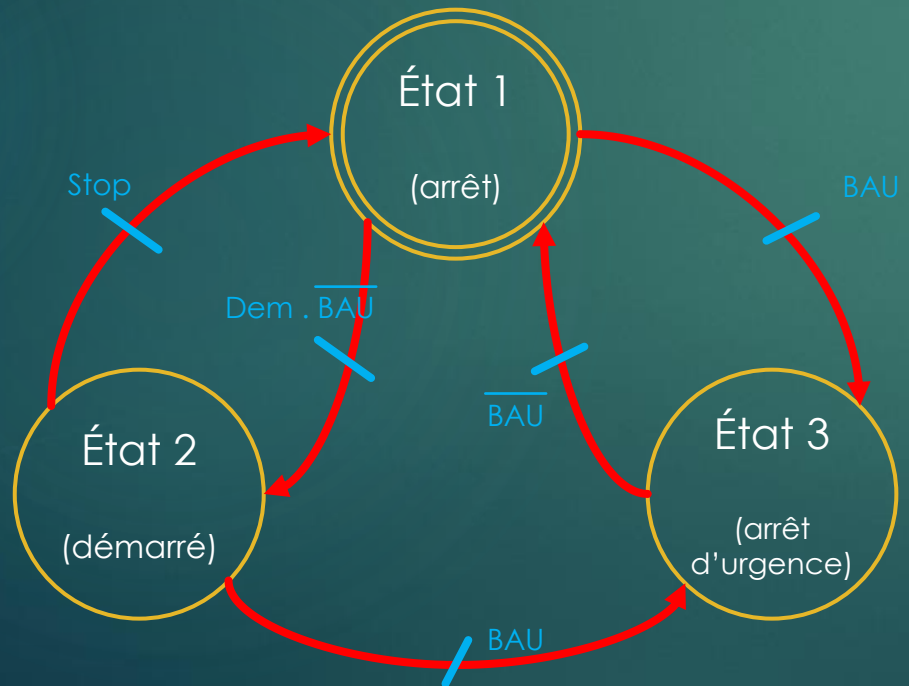
- 1) Décrire le type état, le signal état et les conditions




```
architecture ar of ma_mae is

    type etat_t is (etat_1, etat_2, etat_3);
    signal etat_courant : etat_t;

    signal C1_2 : std_logic;
    signal C1_3 : std_logic;
    signal C2_1 : std_logic;
    signal C2_3 : std_logic;
    signal C3_1 : std_logic;


begin

    C1_2  <= Dem and not BAU;
    C1_3  <= BAU;
    C2_1  <= Stop;
    C2_3  <= BAU;
    C3_1  <= not BAU;

end architecture;
```

Exercice d'application

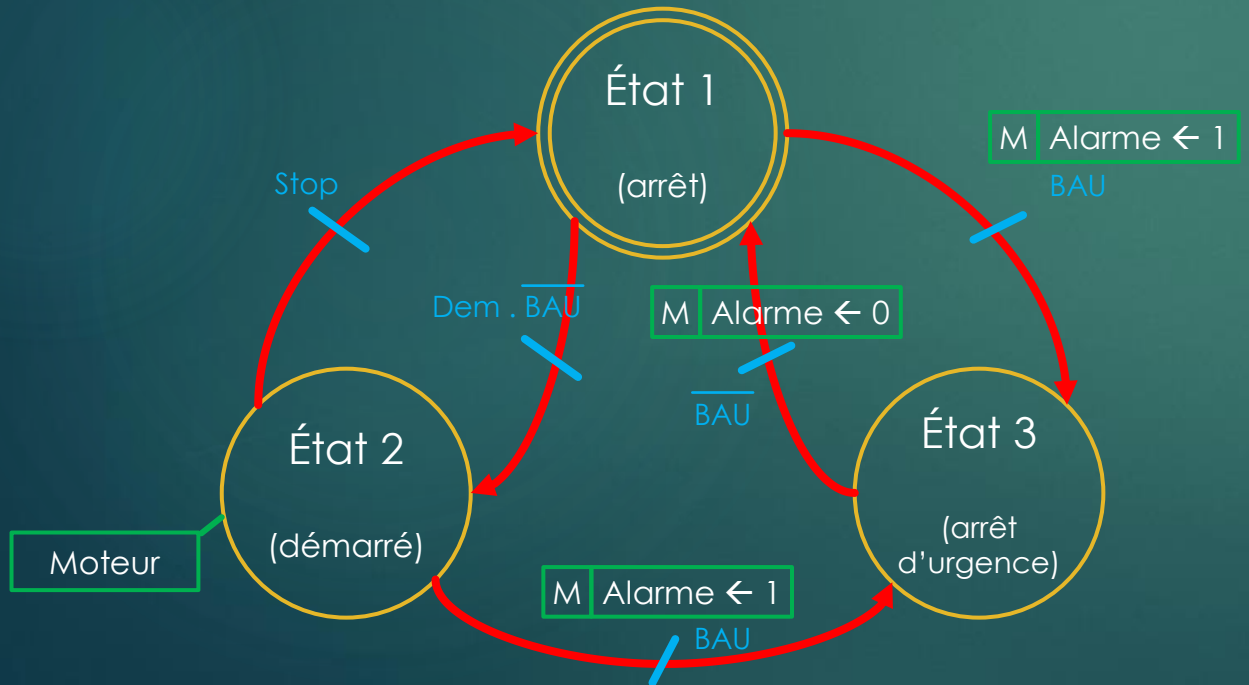
- 1) Décrire le type état, le signal état et les conditions
- 2) Décrire le bloc d'évolution de l'état




```
process(clk, reset)
begin
     if reset = '1' then
          etat_courant <= etat_1;
     elsif rising_edge(clk) then
          case etat_courant is
               when etat_1 =>
                    if C1_2 = '1' then
                         etat_courant <= etat_2;
                    elsif C1_3 = '1' then
                         etat_courant <= etat_3;
                    end if;
               when etat_2 =>
                    if C2_1 = '1' then
                         etat_courant <= etat_1;
                    elsif C2_3 = '1' then
                         etat_courant <= etat_3;
                    end if;
               when etat_3 =>
                    if C3_1 = '1' then
                         etat_courant <= etat_1;
                    end if;
          end case;
     end if;
end process;
```

Exercice d'application

- 1) Décrire le type état, le signal état et les conditions
- 2) Décrire le bloc d'évolution de l'état
- 3) Décrire les actions en utilisant des process




```
process(etat_courant)
Begin
     moteur <= '0';
     case etat_courant is
          when etat_2 =>
               moteur <= '1';
          end case;
     end if;
end process;
```

Attention, il faut penser à indiquer à côté du graphe d'états les valeurs initiales et valeurs par défaut

```
process(clk, reset)
begin
     if reset = '1' then
          alarme <= '0';
     elsif rising_edge(clk) then
          case etat_courant is
               when etat_1 =>
                    if C1_3 = '1' then
                         alarme <= '1';
                    end if;
               when etat_2 =>
                    if C2_3 = '1' then
                         alarme <= '1';
                    end if;
               when etat_3 =>
                    if C3_1 = '1' then
                         alarme <= '0';
                    end if;
          end case;
     end if;
end process;
```

L'icône d'information indique que le contenu de ces slides est à titre informatif, il ne vous est pas demandé de le retenir

Bonus

Information icon4.svg © srittau – Public domain

Variante en utilisant des booléens

```
architecture ar of ma_mae is
    signal C1_2 : boolean;
    signal C1_3 : boolean;
    signal C2_1 : boolean;
    signal C2_3 : boolean;
    signal C3_1 : boolean;
begin

    C1_2  <= true when Dem = '1' and BAU = '0' else false;
    C1_3  <= true when BAU = '1'               else false;
    C2_1  <= true when Stop = '1'              else false;
    C2_3  <= true when BAU = '1'               else false;
    C3_1  <= true when BAU = '0'               else false;

end architecture;
```

```
process(clk, reset)
begin
     if reset = '1' then
          etat_courant <= etat_1;
     elsif rising_edge(clk) then
          case etat_courant is
               when etat_1 =>
                    if C1_2 then
                         etat_courant <= etat_2;
                    elsif C1_3 then
                         etat_courant <= etat_3;
                    end if;
               when etat_2 =>
                    if C2_1 then
                         etat_courant <= etat_1;
                    elsif C2_3 then
                         etat_courant <= etat_3;
                    end if;
               when etat_3 =>
                    if C3_1 then
                         etat_courant <= etat_1;
                    end if;
          end case;
     end if;
end process;
```

Information icon4.svg © srittau – Public domain

Notion d'état total

- Connaître l'état actif d'une machine à état suffit-il à décrire complètement l'état de la machine ?
  - Oui… si il n'y a aucune variable mémorisée
- Dans le cas où il y a des variables mémorisées, on parle d'« état total » pour décrire la totalité de l'état de la machine, qui contient
  - L'état courant
  - Les valeurs courantes de toutes les variables mémorisées
- Voilà pourquoi c'est parfois un abus de langage de parler de « l'état de la MAE » ou de dire qu'une MAE « est dans un état »

Information icon4.svg © srittau – Public domain

Pour aller plus loin…

- One-process vs two-process vs three-process state machine
- Moore machine
- Mealy machine

Information icon4.svg © srittau – Public domain