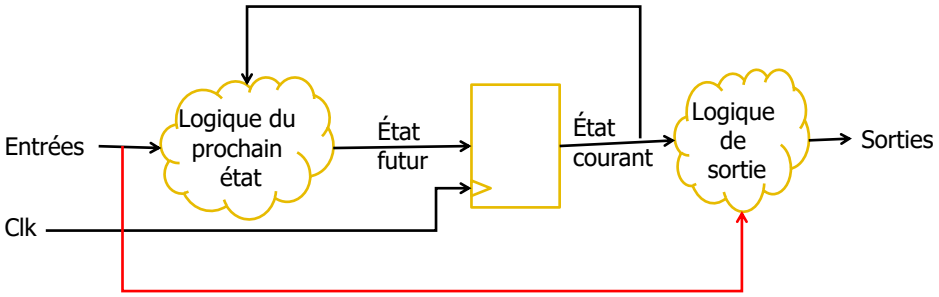
C10 Machine à état (MAE) / Finite State Machine (FSM)

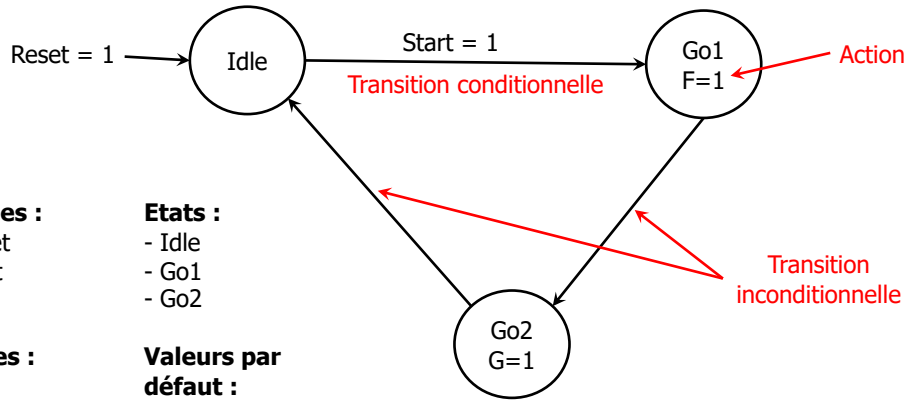
Yann DOUZE

Machine de Moore et Mealy



- Machine de Moore→ les sorties ne dépendent que de l'état courant.
- Machine de Mealy → les sorties dépendent de l'état courant et des entrées.

Diagramme d'état



Entrées :
- Reset
- Start
- Clk

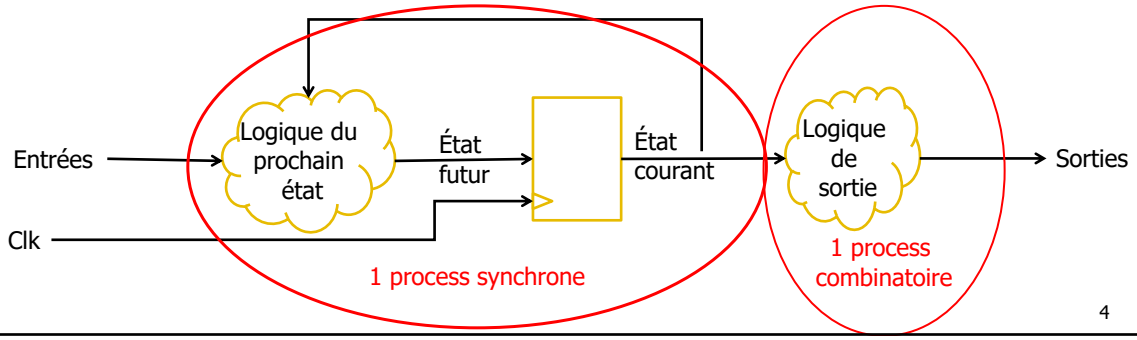
Etats :
- Idle
- Go1
- Go2

Sorties :
- F
- G

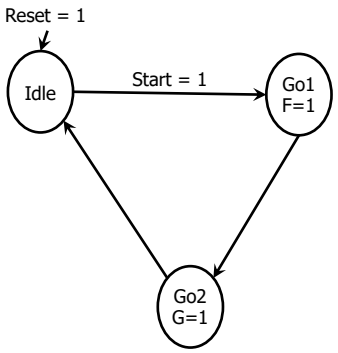
Valeurs par défaut :
- F = 0
- G = 0

Description VHDL : Machine de Moore (à éviter, moins performante)

- 2 process
  - 1 process synchrone pour déterminer le prochain état en fonction de l'état courant et des entrées.
  - 1 process combinatoire qui permet de déterminer les sorties en fonction de l'état courant.



Description VHDL : Machine de Moore (à éviter, moins performante)

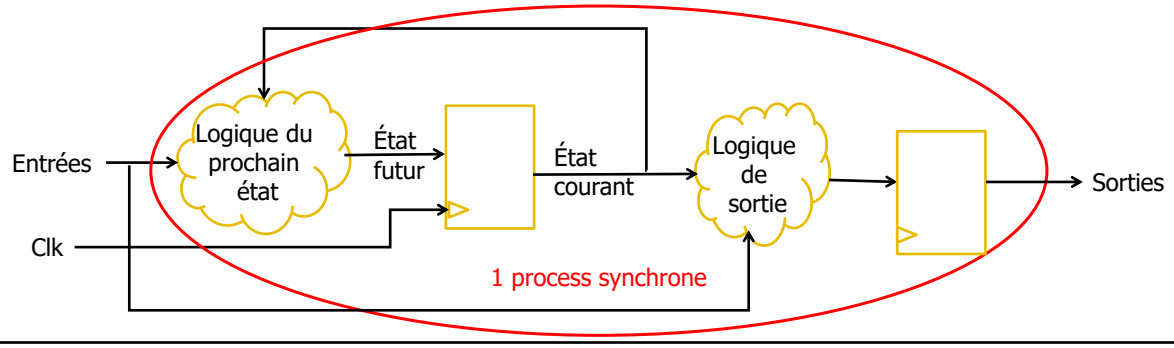



```
architecture FSM of EXEMPLE is
   type StateType is (Idle, Go1, Go2);
   Signal State : StateType;
begin
process(Clk, Reset)
   if Reset = '1' then
      State <= Idle;
   elsif Rising_edge(Clk) then
      case State is
      when Idle =>
         if Start = '1' then
            State <= Go1;
         end if;
      when Go1 =>
         State <= Go2;
      when Go2 =>
         State <= Idle;
      end case;
   end if;
end process;
```

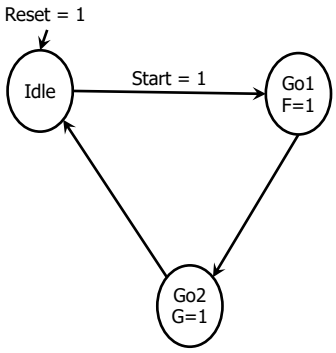
```
output : process(State)
begin
   F <= '0';
   G <= '0';
   if State = Go1 then
      F <= '1';
   elsif State = Go2 then
      G <= '1';
   end if;
end process;
end architecture;
```


Description VHDL : Machine de Mealy resynchronisé.

- 1 seul process synchrone qui permet en même temps de :
  - Déterminer la logique du prochain état en fonction de l'état courant
  - Déterminer l'état des sorties en fonction de l'état courant et des entrées

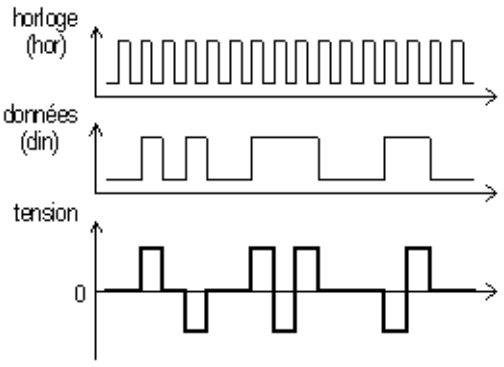


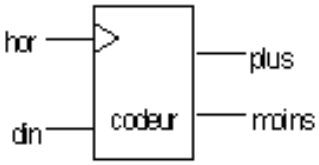
Description VHDL : Machine de Mealy




```
architecture FSM of EXEMPLE is
  type StateType is (Idle, Go1, Go2);
  Signal State : StateType;
begin
process(Clk, Reset)
begin
  if Reset = '1' then
    State <= Idle;
    G <= '0';
    F <= '0';
  elsif Rising_edge(Clk) then
    case State is
    when Idle =>
      if Start = '1' then
        State <= Go1;
        F <= '1';
      end if;
    when Go1 =>
      State <= Go2;
      G <= '1';
      F <= '0';
    when Go2 =>
      State <= Idle;
      G <= '0';
    end case;
  end if;
end process;
end architecture;
```


Codeur AMI (Alternate Mark Inversion)





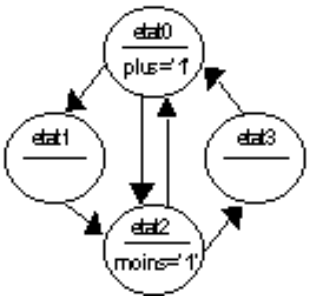
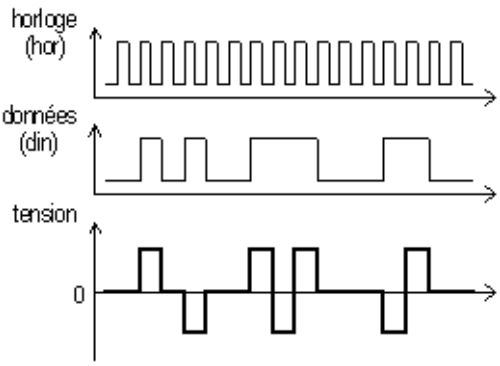
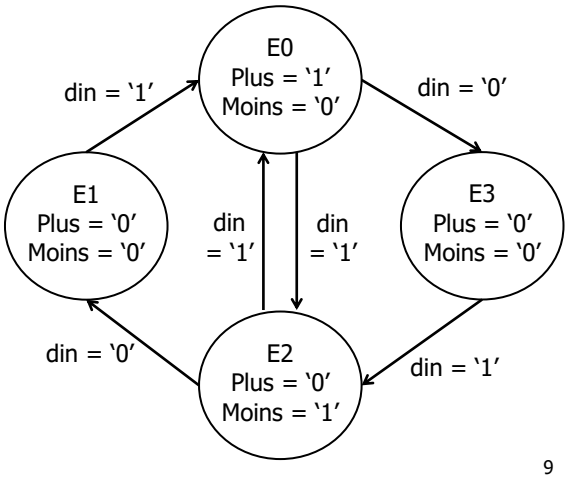
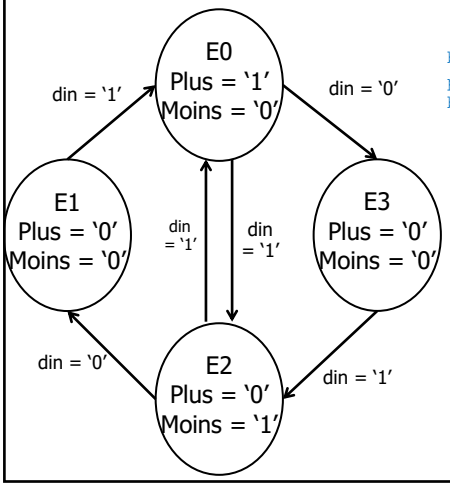


Diagramme d'état du codeur AMI





Code VHDL du codeur AMI




```
architecture MAE of AMI is
   type StateType is (E0, E1, E2, E3);
   Signal State : StateType;
begin
process(Clk, Reset)
begin
   if Reset = '1' then
      State <= E1;
      plus <= '0';
      moins <= '0';
   elsif Rising_edge(Clk) then
      case State is
      when E0 =>if Din = '0' then
                   State <= E3;
                   plus <= '0';
                   moins <= '0';
                elsif Din = '1' then
                   State <= E2;
                   moins <= '1';
                   plus <= '0';
                end if;
      when E1 =>if Din = '1' then
                   State <= E0;
                   moins <= '0';
                   plus <= '1';
                end if;
      when E2 =>if Din = '1' then
                   State <= E0;
                   plus <= '1';
                   moins <= '0';
                elsif Din = '0' then
                   State <= E1;
                   plus <= '0';
                   moins <= '0';
                end if;
      when E3 =>if Din = '1' then
                   State <= E2;
                   plus <= '0';
                   moins <= '1';
                end if;
      end case;
   end if;
end process;
end architecture;
```