Fiche Systèmes a Evènement Discret (SED) Module 2

1 Mise en œuvre de systèmes séquentiels logiques

Position du problème : Analyse → Conception → **Mise En Œuvre (MOE)** → Prototypage → Exploitation

Une MOE c'est : le choix d'une cible, le choix d'un codage[1], la synthèse logique[2] et la réalisation.

1.1 Codage par minimisation du nombre de variables internes

Le nombre n de bits du code des états est minimal → minimisation du coût de mise en œuvre / complexification des équations

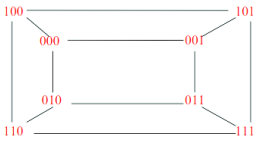
Mode pulsé : Choisir un code de n bits tel que $2^n \geq r_{syteme}$ puis affecter à chaque état un code binaire de n bits ($\neq$ pour chaque état)

Mode fondamental :

(Attention aux **courses critiques**[3])

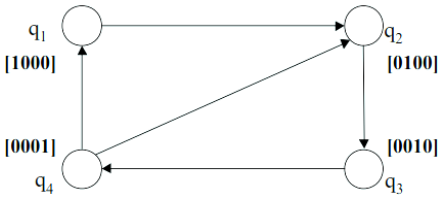
- Choisir un code de n bits tel que
  - $\boxed{2^n \geq r_{syteme}}$
  - $\boxed{n \geq \max(Adjacence\ entre\ état)}$
- Affecter un code n bit à chaque état en respectant les contraintes d'adjacence

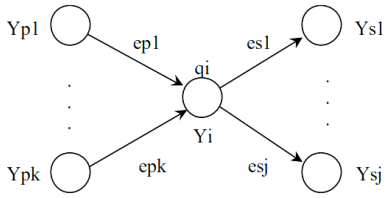
Y1Y2\ab	00	01	11	10	Z1	Z2
00	00	01	00	10	0	0
01	00	01	00	10	0	1
11	00	01	11	10	1	1
10	11	01	01	10	1	0



1.2 Codage 1 parmi n

Le nombre de bits de Y est égal au nombre d'états r du système → simplification des équations / coût de mise en œuvre plus important. Par convention, chaque code ne comporte qu'un seul bit à 1 (le rang de la composante à 1 indique le numéro de l'état actif). La déduction des équations est plus simple.





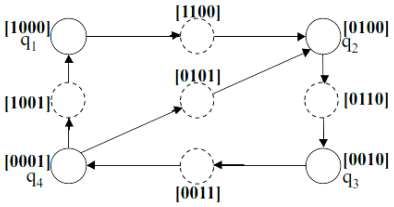
y_i = Terme d'excitation + terme de maintien

$$\boxed{y_i = \sum_{l=1}^{k} e_{pl}.Y_{pl} + Y_i.\overline{MaZ}}$$

Mode pulsé : Equations avec terme de MAZ : $\boxed{y_i = \sum_{l=1}^{j} e_{pl}.Y_{pl} + Y_i.\overline{\sum_{l=1}^{j} e_{sl}}}$

Mode fondamental : Le codage 1 parmis n n'est pas adjacent.

Conventionnellement, on impose une course qui consiste à mettre à 1 la VI correspondant a l'état suivant puis à éteindre la VI correspondant à l'état présent. Equations avec terme de MAZ : $\boxed{y_i = \sum_{l=1}^{k} e_{pl}.Y_{pl} + Y_i.\overline{\sum_{l=1}^{j} Y_{sl}}}$



2 Mise en œuvre sur cible matérielle

Intérêt : Vitesse, coût, encombrement

Types de mis en œuvre :

- Par **circuits combinatoires** (ET, OU, XOR, NAND)
- Par **Bascules** (D, JK, RS)
- Par **PLD** (EPROM, PAL, CPLD, FPGA)

Une entrée supplémentaire **indispensable** : le signal **d'initialisation** $init$

---

[1] Définition du nombre n de $VI + affectation$ d'un code binaire à chacun des r états

[2] Détermination d'une représentation « algébrique » du système

[3] Si la course ne permet pas de se stabiliser sur l'état désiré pour les éviter → 2 états voisins disposent de codes adjacents

Fiche Systèmes a Evènement Discret (SED) Module 2

2.1 Circuits combinatoires

A base de porte combinatoire, c'est un type de mise en œuvre seulement adaptée au mode de fonctionnement fondamental.

Principe

- Chaque VI et chaque sortie est mise en œuvre par un bloc combinatoire
- Les architectures électroniques (logigramme) des blocs sont déduites de la représentation algébrique
  ($y_i = F(\underline{Y}, \underline{E})$ pour $i = 1 \dots n$, $S_j = G(\underline{Y}, \underline{E})$ pour $j = 1 \dots m$)

Méthode (minimisation des VI)

- Coder avec respect des adjacences
- Construire les $TKVI$ de chaque VI **ET** de chaque Sortie
- Déduire une représentation algébrique
- Déduire les logigrammes

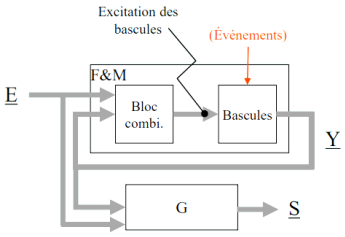
2.2 Bascules

Choisir le type de bascule en fonction du mode fonctionnement.

- Mode **pulsé → D, JK, T**
- Mode **fondamental → RS**

Une bascule = une VI

Utilisation des entrées asynchrones Set et $Clear$ des bascules pour l'initialisation $INIT$

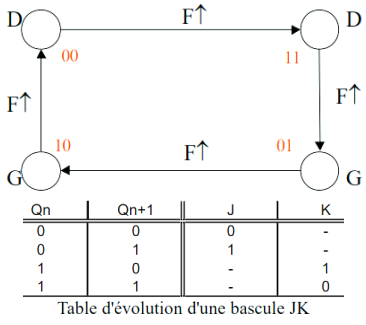


Méthode

- Coder
- Construire les tables d'excitation des bascules (en utilisant la table d'évolution de la bascule choisie)
- Construire les $TKVI$
- Déduire les équations d'excitation des bascules et celles des sorties
- Déduire le logigramme

Mode pulsé

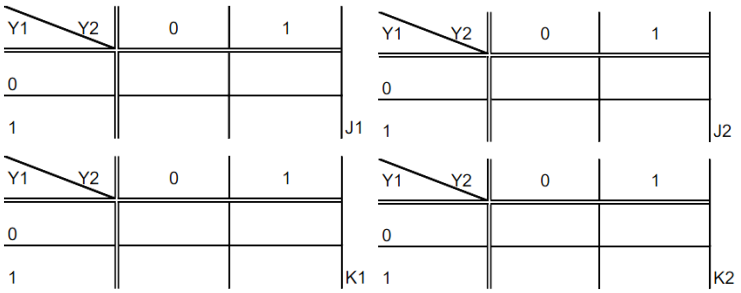
Exemple : le malaxeur **sans** arrêt d'urgence (bascules JK) : Minimisation du nombre de VI



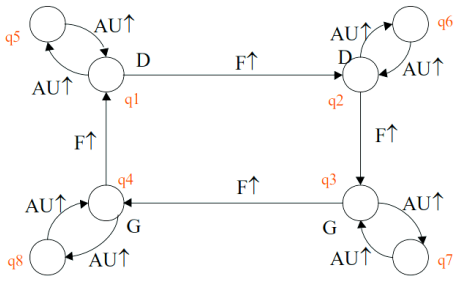
Qn	Qn+1	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

Table d'évolution d'une bascule JK

$2\ VI \rightarrow (J1, K1)(J2, K2)$. Ici, 1 seul évènement F qu'on associe au clk des bascules

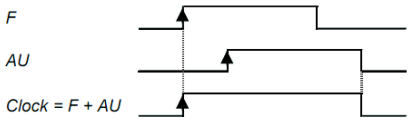


Exemple : le malaxeur **avec** arrêt d'urgence, réalisation 1 parmi n avec bascules D



$$D_1 = F.\overline{AU}.Q_4 + AU.Q_5 + Q_1.0$$
$$D_2 = F.\overline{AU}.Q_1 + AU.Q_6$$
$$D_3 = F.\overline{AU}.Q_2 + AU.Q_7$$
$$D_4 = F.\overline{AU}.Q_3 + AU.Q_8$$
$$D_5 = AU.Q_1 + Q_5.\overline{AU}$$
$$D_6 = AU.Q_2 + Q_6.\overline{AU}$$
$$D_7 = AU.Q_3 + Q_7.\overline{AU}$$
$$D_8 = AU.Q_4 + Q_8.\overline{AU}$$
$$Clk_i = F + AU \quad \forall i = 1 \cdots 8$$

Attention : Ne marche que si on peut garantir que les signaux associés aux évènements ici F et AU **ne sont pas à 1 au même instant**

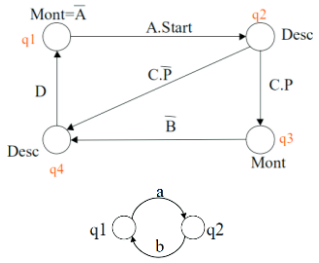


On « rate » un évènement. Si on était en q_1 le système va en q_2 au lieu de q_6

Sinon : passer en **mode fondamental**

Mode fondamental : Exemple de la perceuse

MOE via bascule RS (1 parmi n)



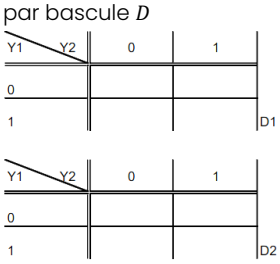
Attention aux problèmes de boucle !

MOE Synchronisé ($MOES$) d'un mode fondamental

Pour éviter le problème de courses critiques

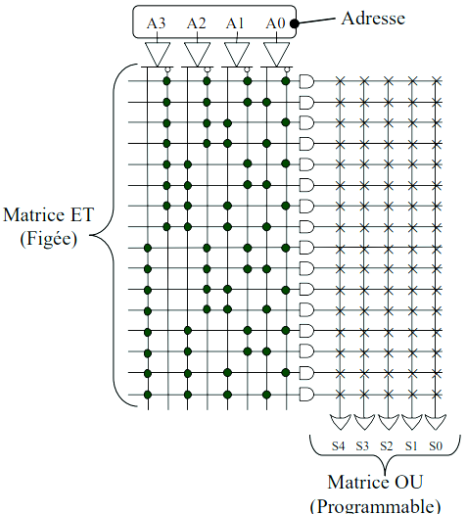
- On peut ajouter une entrée H évènementielle périodique. M = Échantillonneur bloqueur
- La fréquence de H doit être suffisamment importante pour ne pas rater des évolutions significatives du vecteur d'entrée $\underline{E}$
- La fréquence de $H < bloc\ F$ afin d'échantillonner $\underline{y}$ que lorsqu'il a atteint un état stable.
- La vitesse de réaction du système est donc plus lente

Exemple de MOE par bascule D

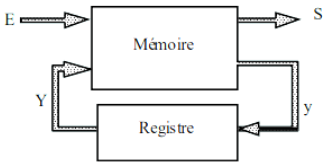


Fiche Systèmes a Evènement Discret (SED) Module 2

2.3 Utilisation de circuits logiques programmable $EEPROM$



1er Principe de MOE



Aléas sur les entrées → MOE synchrone seulement

Taille mémoire : $\boxed{2^{n+p} \times (n+m)}$

Mot Adresses $E_1 \dots E_p | Y_1 \dots Y_n$

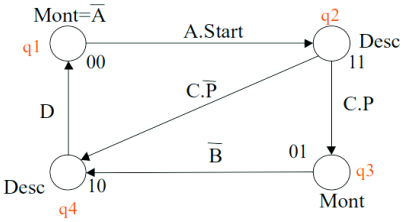
Mot Mémoire : $Y_1 \dots Y_n | S_1 \dots S_m$

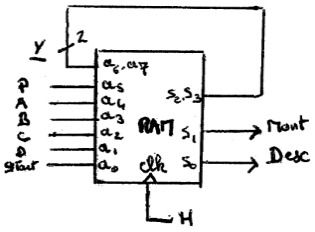
La mémoire croît exponentiellement avec le nombre d'entrées

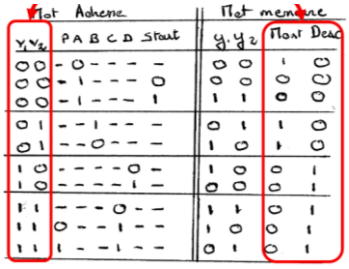
Méthode

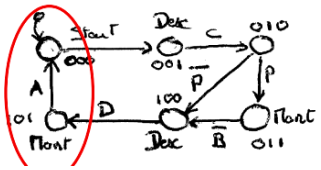
- Coder
- Faire le plan mémoire
- Eventuellement, ajouter une combinatoire sur les sorties

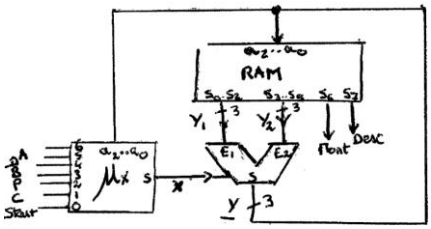
Exemple perceuse $MOES$ d'un mode fondamental

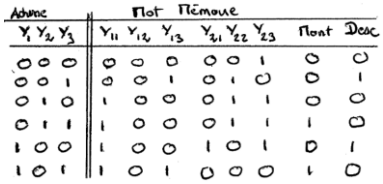








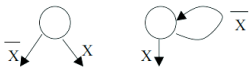




2ème Principe de MOE :

Pour rendre la teille mémoire indépendante du nombre d'entrées

Hypothèse



Taille mémoire : $\boxed{2^n \times (2n+m)}$

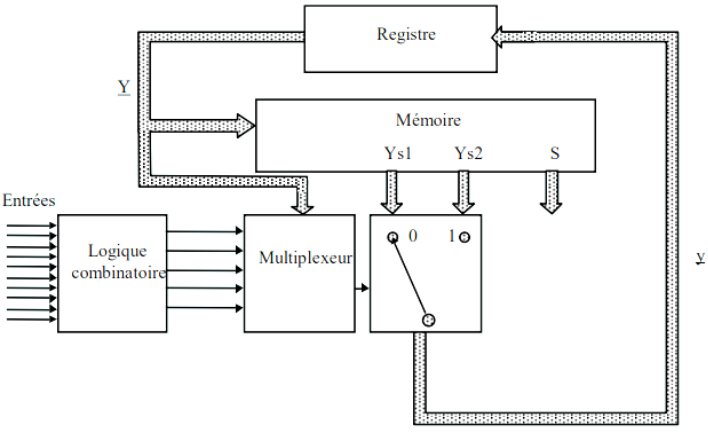
Adresses $Y_1 \dots Y_n$

Mot mémoire : $Y_{S_{11}} \dots Y_{S_{1n}} | Y_{S_{21}} \dots Y_{S_{2n}} | S$

Méthode

- Modifier le graphe
- Coder les états
- Coder les évènements multiplexeurs
- Faire le plan mémoire
- Eventuellement ajouter une combinatoire sur les sorties

Architecture

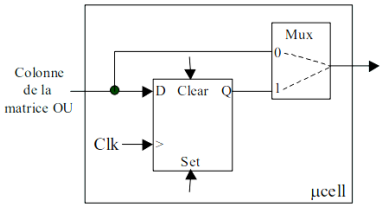


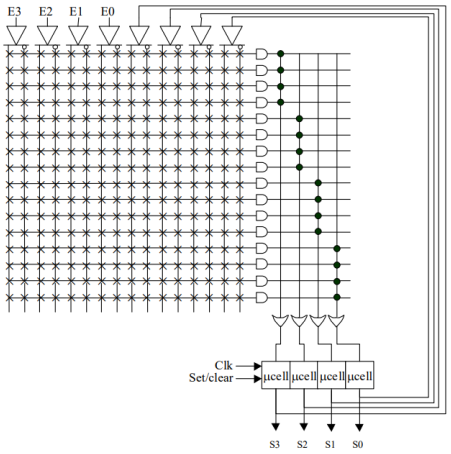
Fiche Systèmes a Evènement Discret (SED) Module 2

2.4 Utilisation de circuits logique programmable : *PLA* (Programmable Logic Array)

Architecture

- Matrice **ET** programmable
- Macrocellule
- Technologie « Fuse »
- Langage « PLDShell »
- Simulation
- Description des équations et du graphe





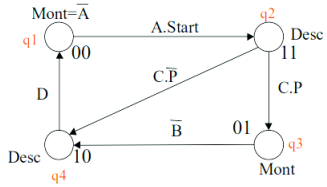
Exemple perceuse (voir MAE page précédente)

```
CHIP Perceuse PLD22V10
; Définition des entrées
PIN 1 H
PIN 2 Start
PIN 3 A
PIN 4 B
PIN 5 C
PIN 6 D
PIN 7 INIT
; Définition des sorties
PIN 22 Desc
PIN 21 Mont
```

```
EQUATIONS
; Description des excitations des bascules D
Y1.D:=A*Start*/Y1*/Y2+/B*/Y1*Y2+Y1*Y2*/(C*P)+Y1*Y2*/D
Y1.CLKF=H
Y1.RSTF=INIT

Y2.D:=A*Start*/Y1*/Y2+B*/Y1*Y2+Y1*Y2*(/C+P)
Y2.CLKF=H
Y2.RSTF=INIT

; Description des sorties
Desc=Y1*/INIT
Mont=(/Y1*(Y2+/A))*/INIT
```

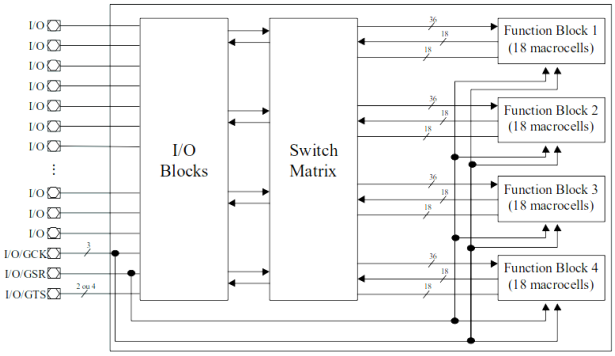



```
SIMULATION
; Description d'un scénario d'évolution des entrées
(A,B,C,D,Start,Init)
```

2.5 Utilisation de circuits logiques programmable : CPLD[4] et FPGA[5]

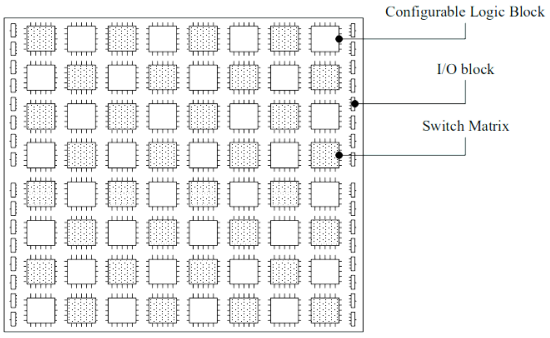
Architecture d'un CPLD

- Difficulté de créer des PLD possédants beaucoup d'entrées
- Idée : Intégrer plusieurs PLD



Architecture d'un FPGA

- Un constructeur origine : Xilinx
- Même idée que CPLD mais le nombre de PLD intégré est beaucoup plus important[6]
- PLD de base (Logic Block) est + simple (~1-2macrocells)
- Plusieurs « Switch Matrix » et « IO Block » → Flexibilité
- Technologie « Fuse » ou « SRAM »
- Pb Routage !



[4] Complex Logical Device

[5] Field Programmable Gate Array

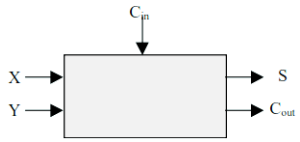
[6] Plusieurs milliers dans un FPGA contre quelques dizaines dans un CPLD

Fiche Systèmes a Evènement Discret (SED) Module 2

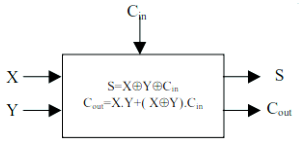
2.6 Un outil de synthèse : VHDL[7]

Norme IEEE (1987) : Source VHDL → Compilation [8] → Simulation [9] → Synthèse [10] → Placement [11]+ Routage → Fabrication

Vue externe



Vue interne



Vue externe

```
Entity adder is Port
(X, Y, Cin   : in bit;
 S,Cout      : out bit);
End Adder;
```

Vue interne

```
Architecture Data_Flow of adder is
Signal Z : bit;
Begin
Z <= X xor Y;
S <= Z xor Cin;
Cout <= (X and Y) or (Z and Cin);
End Data Flow;
```

Vue interne

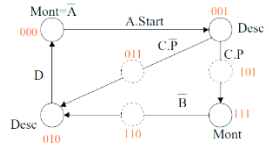
```
Architecture Comportementale of adder is
variable n: integer;
constant s_vector: Std_Logic _vector(0 to
3)="0101"
constant c_vector: Std_Logic_vector(0 to 3)="0011"
Begin Process(X,Y,Cin) -- Liste de sensibilite
N:=0;
if X='1' then N:=N+1; end if;
if Y='1' then N:=N+1; end if;
if Cin='1' then N:=N+1; end if;
S <= s_vector(N);
Cout <= c_vector(N);
End Process;
End Comportementale;
```

Instructions concurrentes :
Réalisation combinatoire → Les affectations se font en parallèle

Instructions séquentielles : Notion de PROCESS
Réalisation synchrone → Les affectations se font en séquence et ne sont répercutés en sortie qu'à la fin du process (horloge)

Exemple

MEO **asynchrone** de MAE en VHDL

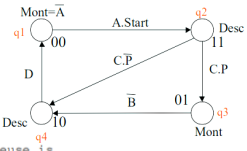



```
Entity CommandePerceuse is
Port
(Start, A, B, C, D, P, Init   : In Std_Logic;
Mont,Desc : Out Std_Logic);
End CommandePerceuse;

Architecture MEF_Asynchrone_MinVI of CommandePerceuse is
-- Déclaration des VI
Signal Y1,Y2,Y3 : std_logic;

Begin
-- Description des Blocs F et M
Y1 <= ((Y1 and Y3) or (C and P and not Y2 and Y3)) and not init;
Y2 <= ((Y2 and Y3) or (Y1 and Y3) or (Y1 and Y2) or (V and not P and Y3)) and not init;
Y3 <= ((not Y2 and Y3) or (B and Y1 and Y3) or (A and Start and not Y2)) and not init;
-- Description du Bloc G
Desc <= ((not Y1 and Y3) or (not Y1 and Y2)) and not init;
Mont <= ((Y1 and Y3) or (not A and not Y1 and not Y2 and not Y3)) and not init;
End MEF_Asynchrone_MinVI;
```

MEO **synchrone** de MAE en VHDL




```
Entity CommandePerceuse is
Port
(Start, A, B, C, D, P, Init   : In Std_Logic;
Mont,Desc : Out Std_Logic);
End CommandePerceuse;
Architecture MEF_Synchrone_MinVI of CommandePerceuse is
Signal Y1Present,Y2Present, Y1Suivant, Y2Suivant : Std_Logic;
Begin
-- Description du Bloc F
Y1Suivant <= ...;
Y2Suivant <= ...;
-- Description du bloc M
Process (H, Init)
Begin
If (Init='1') then
  Y1Present <= '0'; -- Valeur correspondant à l'état initial
  Y2Present <= '0'; -- Valeur correspondant à l'état initial
ElsIf (H'event and H='1') -- Déclenchement sur front montant
  Y1Present <= Y1Suivant;
  Y2Present <= Y2Suivant;
End If;
End Process;
-- Description du Bloc G
Desc <= ((not Y1 and Y3) or (not Y1 and Y2)) and not init;
Mont <= ((Y1 and Y3) or (not A and not Y1 and not Y2 and not Y3)) and not init;
End MEF_Synchrone_MinVI;
```

MEO **synchrone abstraite** de MAE en VHDL

```
Architecture MEF_Synchrone_Haut_Niveau of CommandePerceuse
is
-- Déclaration des états
Type Etat is (Etat1, Etat2, Etat3, Etat4);
Signal EtatPresent, EtatSuivant : Etat;
Begin
-- Description du Bloc F
Process (A,B,C,D,Start)
Begin
Case EtatPresent is
  When Etat1 =>
   If ((A='1') and (Start='1')) then EtatSuivant <= Etat2;
   else EtatSuivant <= EtatPresent;
   End If;
  When Etat2 =>
   If ((C='1') and (P='1') then EtatSuivant <= Etat3;
   Elsif ((C='1') and (P='0') then EtatSuivant <= Etat4;
   else EtatSuivant <= EtatPresent;
   End If;
  ...
   When others => EtatSuivant <= EtatPresent;
End Case;
End Process;
```

```
-- Description du bloc M
Process (H, Init)
Begin
If (Init='1') then
  EtatPresent <= Etat1; -- état initial
ElsIf (H'event and H='1') -- front montant
  EtatPresent <= EtatSuivant;
End If;
End Process;
-- Description du Bloc G
With EtatPresent select
  Mont <= (not A) and (not init) when Etat1,
          not Init when Etat3,
          0 when others;
With EtatPresent select
  Desc<= (not init) when Etat2 | Etat4,
          0 when others;
End MEF_Synchrone_Haut_Niveau;
```

[7] *VHSIC Hardware Description Language*

[8] Erreurs de Syntaxes

[9] Tests fonctionnels et temporels

[10] MEO des équations booléennes

[11] Calcul des temps de propagations

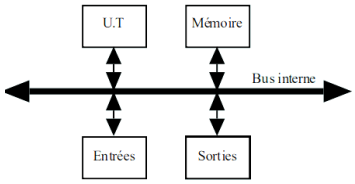
Fiche Systèmes a Evènement Discret (SED) Module 2

Principe et chronogramme d'exécution

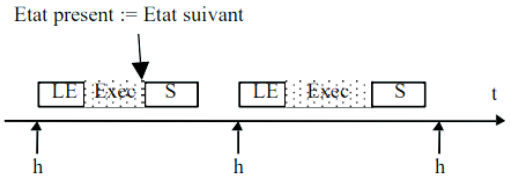
Cycle programme

- Lecture Entrée (LE)
- Calcul des états suivants ($EXEC$)
- Calcul des sorties (S)

API + Microcontrôleur + PC (avec périphériques E/S)

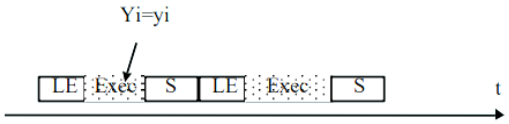


Mise en œuvre synchrone



Les entrées doivent avoir la même valeur durant la totalité du cycle programme → elles sont échantillonnées

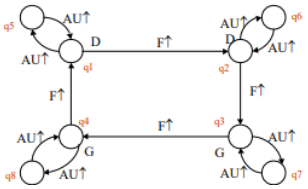
Mise en œuvre asynchrone



Si la commutation de toutes les VI a lieu à la fin du bloc $EXEC$ alors la mise en œuvre est synchrone

Exemple : Malaxeur avec arrêt d'urgence MOE **synchrone**, **1 parmi** n (Langage C)

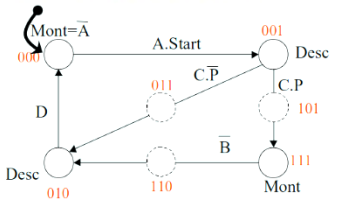
```
int Etat, Etat_Suivant;
int AU,AU_Suivant,F,F_Suivant;
int D,G;
int main(void)
{
  Etat=Etat_Suivant=0;
  AU=F=0;
  while(1)
  {
    /* Lecture de AU_Suivant et F_Suivant sur le periph
      d'entrée */
    ...
    if(AU==0 && AU_Suivant==1) // Front montant
    {
      AU=AU_Suivant;
      switch (Etat)
        {
          case 0: Etat_Suivant=1;break;
          case 1: Etat_Suivant=0;break;
          case 2: Etat_Suivant=3;break;
          case 3: Etat_Suivant=2;break;
          case 4: Etat_Suivant=5;break;
          case 5: Etat_Suivant=4;break;
          case 6: Etat_Suivant=7;break;
          case 7: Etat_Suivant=6;break;
          default: Etat_Suivant=Etat;break;
        }
    }
    else if (AU==1 && AU_Suivant==0) // Front descendant
      AU=AU_Suivant;
    Else if(F==0 && F_Suivant==1) // Front montant
        {
          F=F_Suivant;
          switch (Etat)
            {
              case 0: Etat_Suivant=2;break;
              case 2: Etat_Suivant=4;break;
              case 4: Etat_Suivant=6;break;
              case 6: Etat_Suivant=0;break;
              default: Etat_Suivant=Etat;break;
            }
        }
    else if (F==1 && F_Suivant==0) // Front descendant
        F=F_Suivant;
    Etat=Etat_Suivant;
    /* Calcul des sorties */
    G = (Etat==0) || (Etat==2);
    D = (Etat==4) || (Etat==6);
    /* Activation des sorties sur le periph de sortie */
    ...
    }
  return 1;
  }
```



Exemple : Malaxeur avec arrêt d'urgence

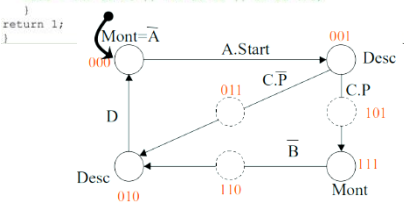
MOE **asynchrone**, **minimisation du nombre de** VI (Langage C)

```
int Y1,Y2,Y3;
int P,Start,A,B,C,D;
int Mont,Desc;
int main(void)
{
  Y1=Y2=Y3=0;
  while(1)
  {
    /* Lecture de P,Start,A,B,C,D sur le periph
      d'entrée */
    ...
    /* Calcul de l'état suivant */
    Y1= Y1 && Y3 || C && P && !Y2 && Y3;
    Y2= Y2 && Y3 || Y1 && Y3 || Y1 && Y2 || C && !P && Y3;
    Y3= !Y2 && Y3 || B && Y1 && Y3 || A && Start && !Y2;
    /* Mise à jour des sorties vers le periph de sortie */
    Mont = !Y1 && !Y2 && !Y3 && !A || Y1 && Y3;
    Desc = !Y1 && Y3 || !Y1 && Y2 || Y2 && !Y3;
    ...
  }
return 1;
}
```



MOE **synchrone**, **minimisation du nombre de** VI (Langage C)

```
int y1,Y1,y2,Y2,y3,Y3;
int P,Start,A,B,C,D;
int Mont,Desc;
int main(void)
{
  Y1=Y2=Y3=0;
  while(1)
  {
    /* Lecture de P,Start,A,B,C,D sur le periph
      d'entrée */
    /* Calcul de l'état suivant */
    y1= Y1 && Y3 || C && P && !Y2 && Y3;
    y2= Y2 && Y3 || Y1 && Y3 || Y1 && Y2 || C && !P && Y3;
    y3= !Y2 && Y3 || B && Y1 && Y3 || A && Start && !Y2;
    /* Commutation simultanée des VI */
    Y1=y1; Y2=y2;Y3=y3;
    /* Mise à jour des sorties vers le periph de sortie */
    Mont = !Y1 && !Y2 && !Y3 && !A || Y1 && Y3;
    Desc = !Y1 && Y3 || !Y1 && Y2 || Y2 && !Y3;
    }
return 1;
}
```



MOE **synchrone**, MOE directe de la MEF (Langage C)

```
int Etat_p, Etat_s;
int P,Start,A,B,C,D;
int Mont,Desc;
int main(void)
{
  Etat_p=1;
  while(1)
  {
    /* Lecture de P,Start,A,B,C,D sur le periph d'entrée */
    /* Calcul de l'état suivant */
    switch(Etat_p) {
        case 1: if(A && Start) Etat_s=2;
                else Etat_s=Etat_p; break;
        case 2: if(C && !P) Etat_s=4;
                else if(C && P) Etat_s=3;
                else Etat_s=Etat_p; break;
        case 3: if(!B) Etat_s=4;
                else Etat_s=Etat_p; break;
        case 4: if(D) Etat_s=1;
                else Etat_s=Etat_p; break;
        default: Etat_s=Etat_p;}
    /* Commutation de l'état*/
    Etat_p=Etat_s;
    /* Mise à jour des sorties vers le periph de sortie */
    Mont = (Etat_p==1) && !A || Etat_p==3;
    Desc = (Etat_p==2) || (Etat_p==4);}
return 1;
}
```