
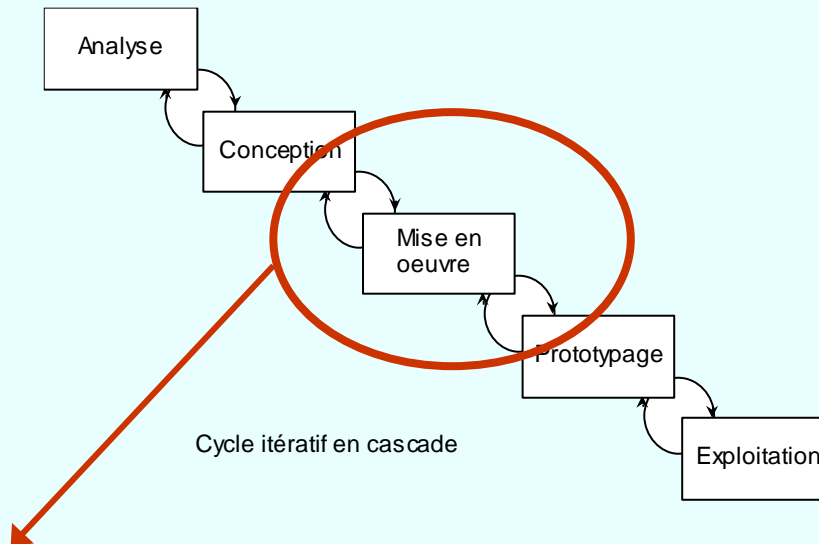


# "Systèmes à événements discrets"

- 
- I • Rappels de logique combinatoire et séquentielle
  - II • Les systèmes séquentiels logiques
  - III • Mise en œuvre de systèmes séquentiels logiques
  - IV • Systèmes à évolutions simultanées (sem2)

### III.1. Qu'est ce que la mise en œuvre ?

#### ❑ Position du problème



- Mise en œuvre = choix d'une cible + choix d'un codage + Synthèse logique + réalisation
- Codage = Définition du nombre  $n$  de VI + affectation d'un code binaire à chacun des  $r$  états
- Synthèse logique = détermination d'une représentation "algébrique" du système

## III.2. Codage d'une machine à états

### ❑ Codage par minimisation du nombre de variables internes

- Le nombre  $n$  de bits du code des états est minimal ( $\Rightarrow$  minimisation du coût de mise en œuvre / complexification des équations)

### ❑ Cas du mode pulsé : méthode de codage et exemple

- Choisir un code de  $n$  bits tel que  $2^n \geq$  (nombre  $r$  d'états du système)
- Affecter à chaque état un code binaire de  $n$  bits (les états doivent avoir des codes différents)

### ❑ Cas du mode fondamental : le problème des courses critiques

- Illustration

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

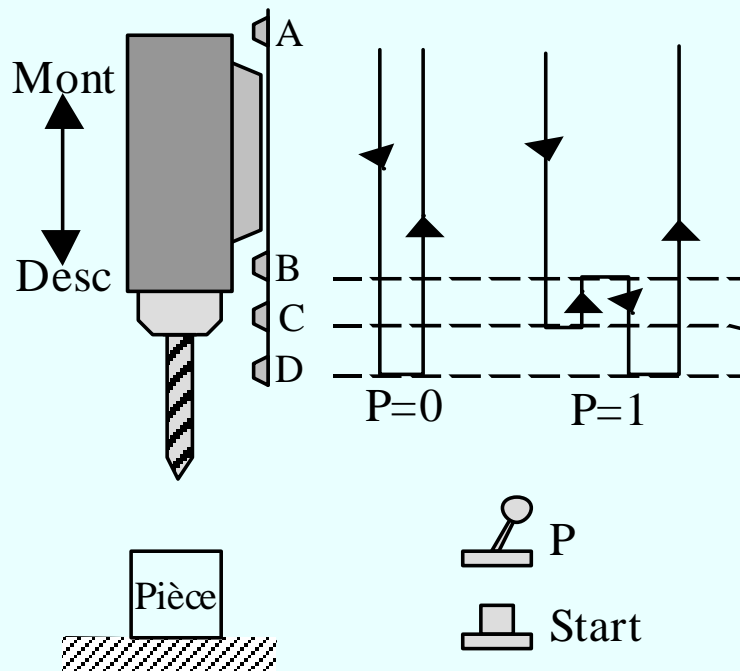
- Une course est critique si elle ne permet pas de se stabiliser sur l'état désiré

### III.1. Qu'est ce que la mise en œuvre ?

#### ❑ Exemples utiliser pour la MEO

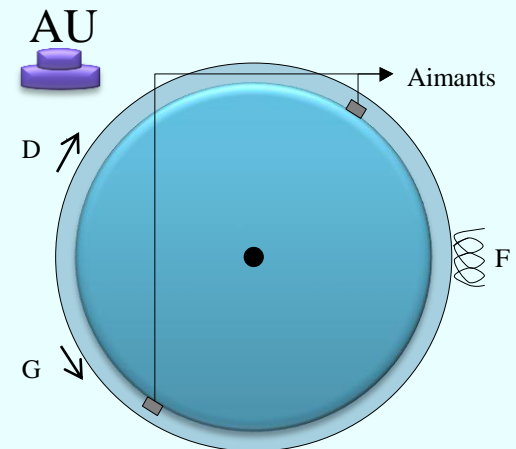
##### ➤ Mode fondamental

exemple de la machine à percer



##### ➤ Mode pulsé

Malaxeur avec arrêt d'urgence



- a) 1 tour à droite / 1 tour à gauche
- b) Si appui sur AU, on interrompt le mouvement. Si nouvel appui sur AU, on le relance là ou on l'avait interrompu.

## III.2. Codage d'une machine à états

### ❑ Cas du mode fondamental (suite)

#### ➤ Il faut éviter les courses critiques

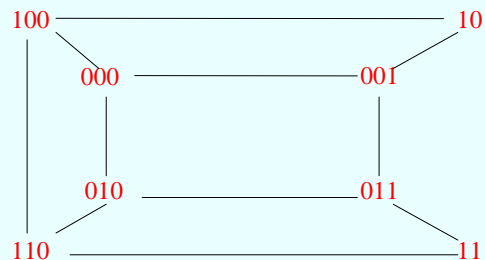
- car non adjacence des codes => indéterminisme de comportement
- Une condition suffisante pour éliminer les courses critiques :

**Deux états voisins doivent posséder des codes adjacents**

#### ➤ Une méthode saine de codage

- Choisir un code de  $n$  bits tel que :
  - $2^n \geq (\text{nombre } r \text{ d'états du système})$  **ET**
  - $n \geq \text{Max}(\text{contraintes d'adjacence entre états})$
- Affecter un code de  $n$  bits à chaque état en respectant les contraintes d'adjacence

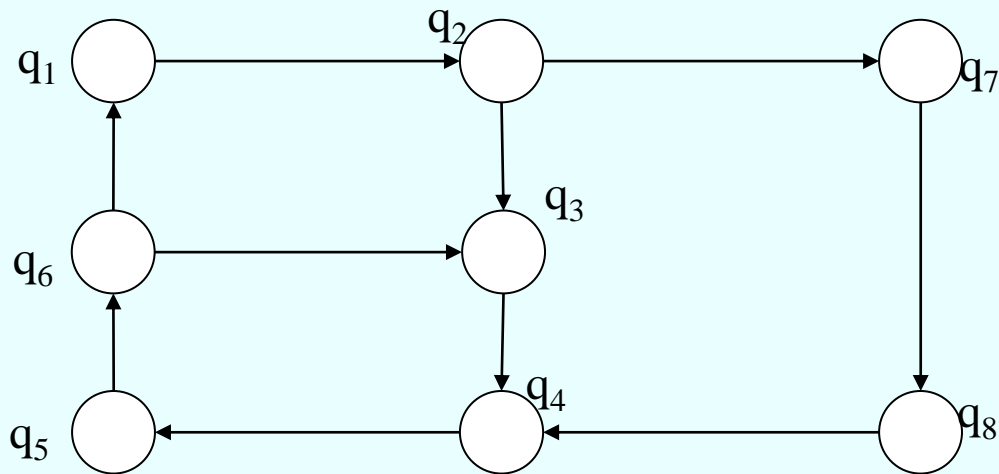
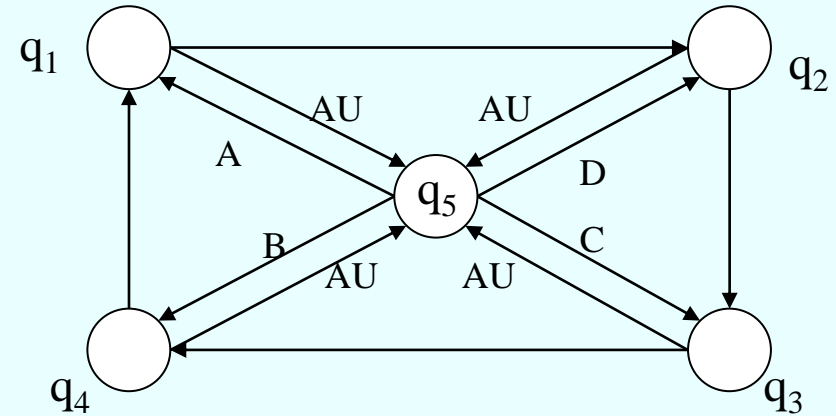
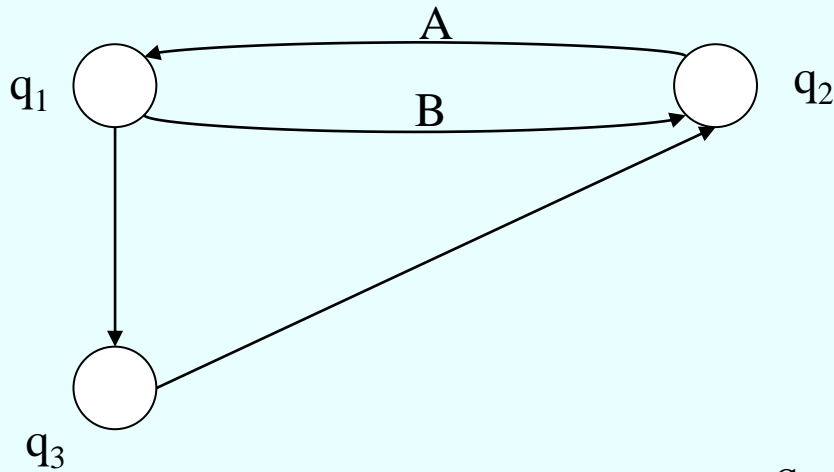
#### ➤ Graphe d'adjacence



## III.2. Codage d'une machine à états

### ❑ Cas du mode fondamental (suite)

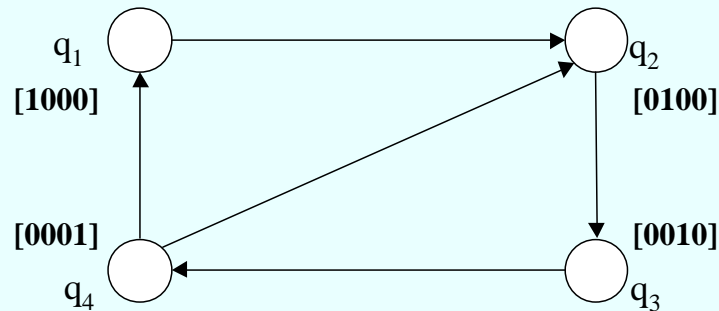
➤ Exemples : Proposer un codage pour les machines à états suivantes



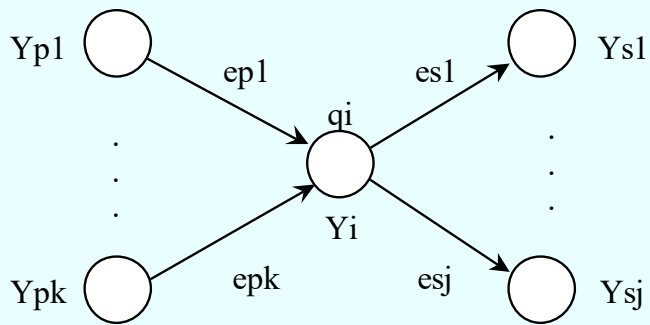
## III.2. Codage d'une machine à états

### ❑ Codage 1 parmi n

- Le nombre  $n$  de bits de  $\underline{Y}$  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 d'état actif)



- La déduction des équations est plus simple



$y_i = \text{terme d'excitation} + \text{terme de maintien}$

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

## III.2. Codage d'une machine à états

### □ Cas du mode pulsé

- Équations :

Terme de MAZ :

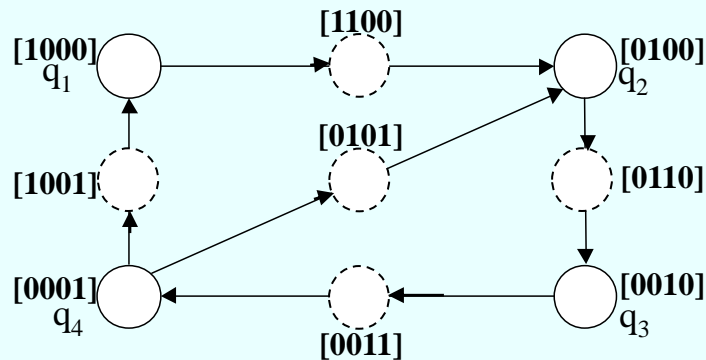
$$\sum_{l=1}^j e_{sl} \Rightarrow$$

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

### □ Cas du mode fondamental

- Le codage 1 parmi n n'est pas adjacent !!!
- Conventionnellement, on impose une course qui consiste à mettre à 1 la VI correspondant à l'état suivant puis, à éteindre la VI correspondant à l'état présent.

- Exemple :



- Équations :

Terme de MAZ :

$$\sum_{l=1}^j Y_{sl}$$

=>

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



## III.3. Mise en œuvre sur cible matérielle

### ❑ Introduction

#### ➤ Intérêts

- vitesse
- coût
- encombrement

#### ➤ 3 types de mise en œuvre :

- Mise en œuvre par circuits combinatoires (ET, OU, XOR, NAND, ...)
- Mise en œuvre par bascules (D,JK,RS)
- Mise en œuvre par PLD (EPROM, PAL, CPLD, FPGA)

#### ➤ Une entrée supplémentaire indispensable : le signal d'initialisation (*init*)

### III.3. Mise en œuvre sur cible matérielle

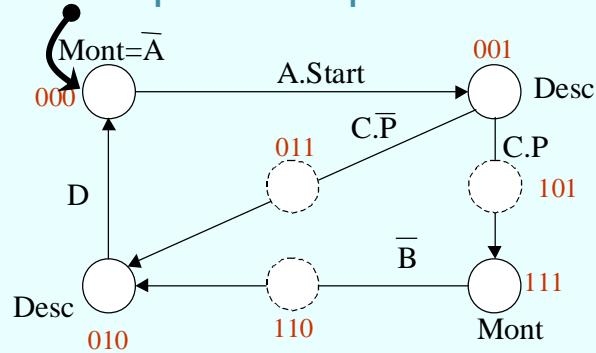
#### ❑ Utilisation de circuits combinatoires : le rebouclage direct

- Type de mise en œuvre adapté au mode fondamental seulement
- Portes combinatoires = Base de toutes les mises en œuvre matérielles
- 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 du nombre de VI)

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

### III.3. Mise en œuvre sur cible matérielle

#### ➤ Exemple de la perceuse : minimisation du nombre de VI



Y1 \ Y2Y3	00	01	11	10
0				
1				

y1

Y1 \ Y2Y3	00	01	11	10
0				
1				

y3

Y1 \ Y2Y3	00	01	11	10
0				
1				

y2

Y1 \ Y2Y3	00	01	11	10
0				
1				

Mont 1

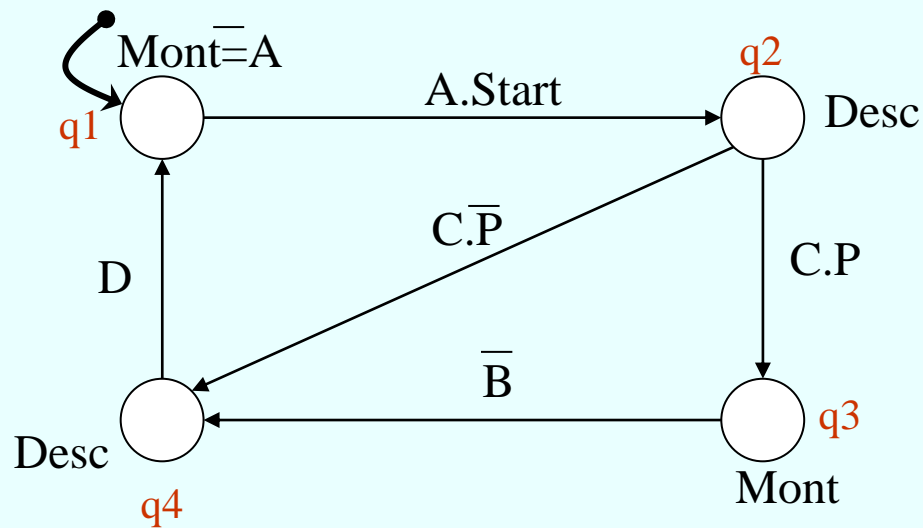
Y1 \ Y2Y3	00	01	11	10
0				
1				

Desc

### III.3. Mise en œuvre sur cible matérielle

➤ Exemple de la perceuse : mise en œuvre 1 parmi n

- Dédution directe des équations à partir de la représentation graphique



$y1 =$

$y4 =$

$y2 =$

$\text{Mont} =$

$y3 =$

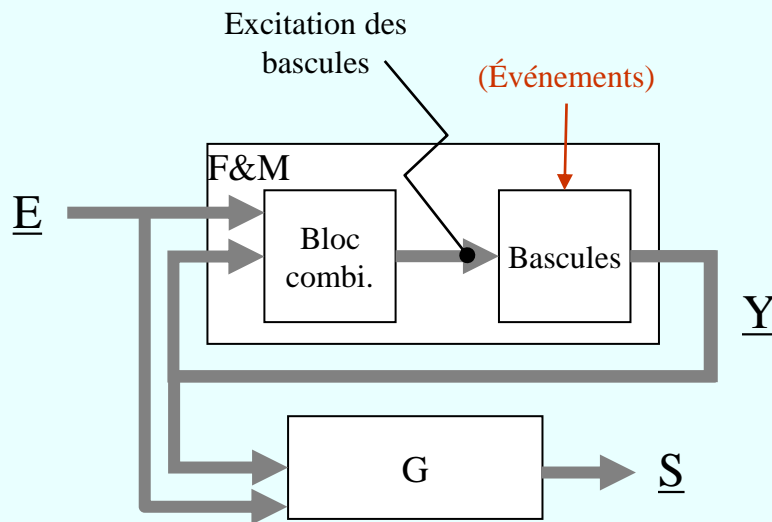
$\text{Desc} =$

### III.3. Mise en œuvre sur cible matérielle

#### □ Utilisation de bascules

- Bascules D, JK, T => Mode pulsé
  - Bascules RS => Mode fondamental
  - Une bascule  $\equiv$  une VI
  - Utilisation des entrées asynchrones *Set* et *Clear* des bascules pour l'initialisation (entrée *INIT*)
  - Principe
- Choisir le type de bascule en fonction du mode de fonctionnement

#### □ Méthode

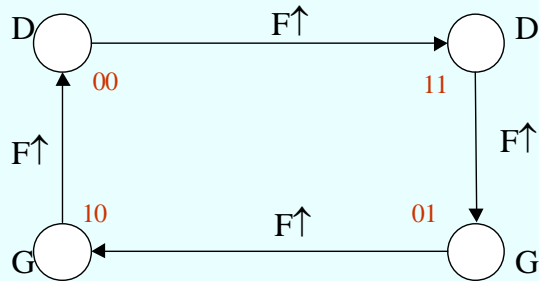


- *Coder*
- *Construire les tables d'excitation des bascules (en utilisant la table d'évolution de la bascule choisie)*
- *Construire les TKVI des sorties*
- *Déduire les équations d'excitation des bascules et celles des sorties*
- *Déduire le logigramme*

### III.3. Mise en œuvre sur cible matérielle

#### ➤ Cas du mode pulsé

- Exemple : le malaxeur sans arrêt d'urgence (bascules JK) : minimisation du nb 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 événement  $F$  qu'on associe au *clock* des bascules

Y1 \ Y2	0	1
0		
1		

J1

Y1 \ Y2	0	1
0		
1		

K1

Y1 \ Y2	0	1
0		
1		

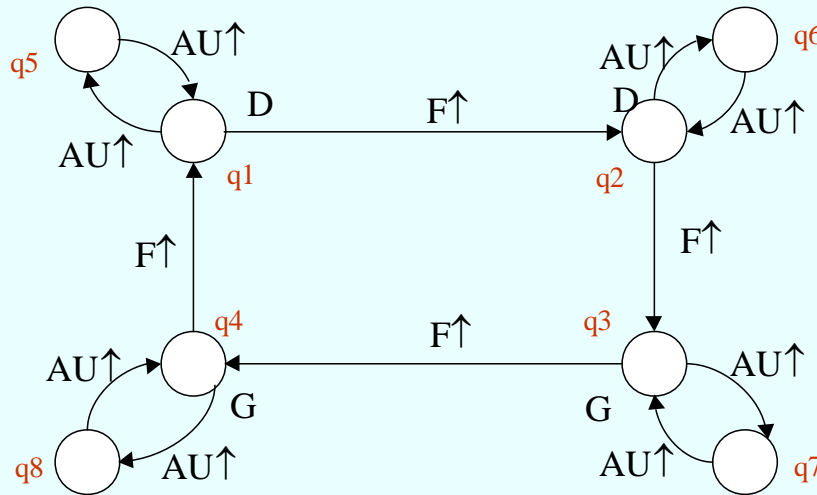
J2

Y1 \ Y2	0	1
0		
1		

K2

### III.3. Mise en œuvre sur cible matérielle

- Exemple : Malaxeur avec arrêt d'urgence - Réalisation 1 parmi  $n$  avec bascules D



$$D_1 = F.\overline{AU}.Q_4 + AU.Q_5 + Q_1.0$$

$$D_2 = F.\overline{AU}.Q_1 + AU.Q_6$$

$$D_3 = F.\overline{AU}.Q_2 + AU.Q_7$$

$$D_4 = F.\overline{AU}.Q_3 + AU.Q_8$$

$$D_5 = AU.Q_1 + Q_5.\overline{AU}$$

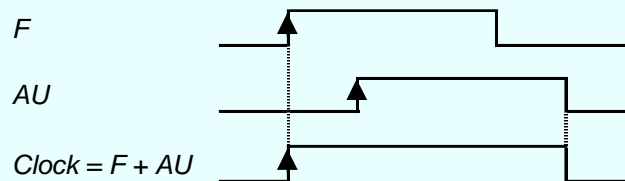
$$D_6 = AU.Q_2 + Q_6.\overline{AU}$$

$$D_7 = AU.Q_3 + Q_7.\overline{AU}$$

$$D_8 = AU.Q_4 + Q_8.\overline{AU}$$

$$Clk_i = F + AU \quad \forall i = 1 \dots 8$$

- ATTENTION : Ca ne marche que si on peut garantir que les signaux associés aux événements (ici  $F$  et  $AU$ ) ne sont pas à 1 au même instant



On "râte" un événement



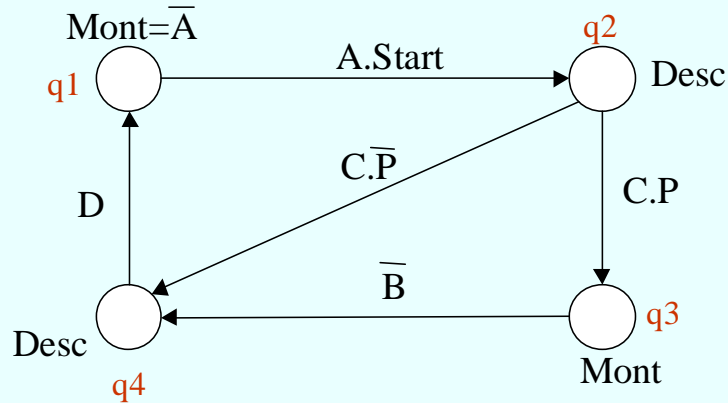
Si on était en  $q1$  le système va en  $q2$  au lieu de  $q6$

- DANS LE CAS CONTRAIRE : PASSER EN MODE FONDAMENTAL

### III.3. Mise en œuvre sur cible matérielle

#### ➤ Cas du mode fondamental

- Exemple de la perceuse : Mise en œuvre par bascule RS (1 parmi n)



R1 =

R3 =

S1 =

S3 =

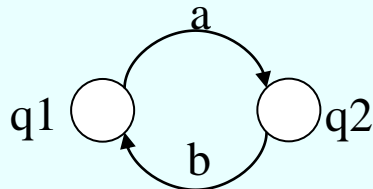
R2 =

R4 =

S2 =

S4 =

- Mise en œuvre 1 parmi n par bascules RS : le problème des boucles!



R1 =

S1 =

R2 =

S2 =



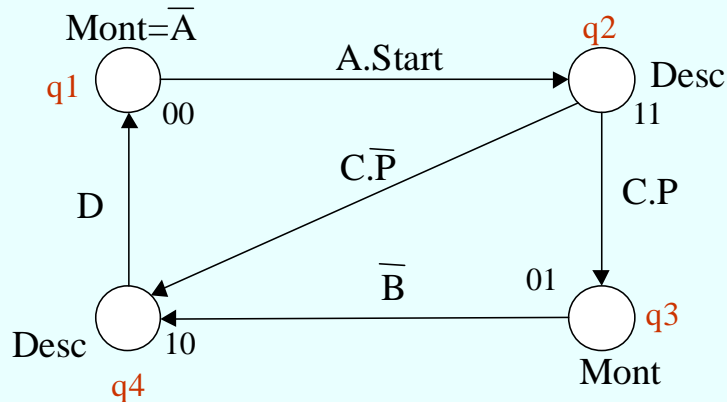
### III.3. Mise en œuvre sur cible matérielle

#### ❑ Mise en œuvre **synchronisée** d'un mode fondamental

##### ➤ Pour éviter le problème des courses critiques

- on peut ajouter une entrée  $H$  événementielle périodique  $\Rightarrow M$  = 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  $H$  doit être inférieure à celle du bloc  $F$  afin d'échantillonner  $y$  que lorsqu'il a atteint un état stable
- la vitesse de réaction du système est donc plus lente

##### ➤ Exemple : mise en œuvre par bascules D



Y1 \ Y2	0	1	
0			
1			D1

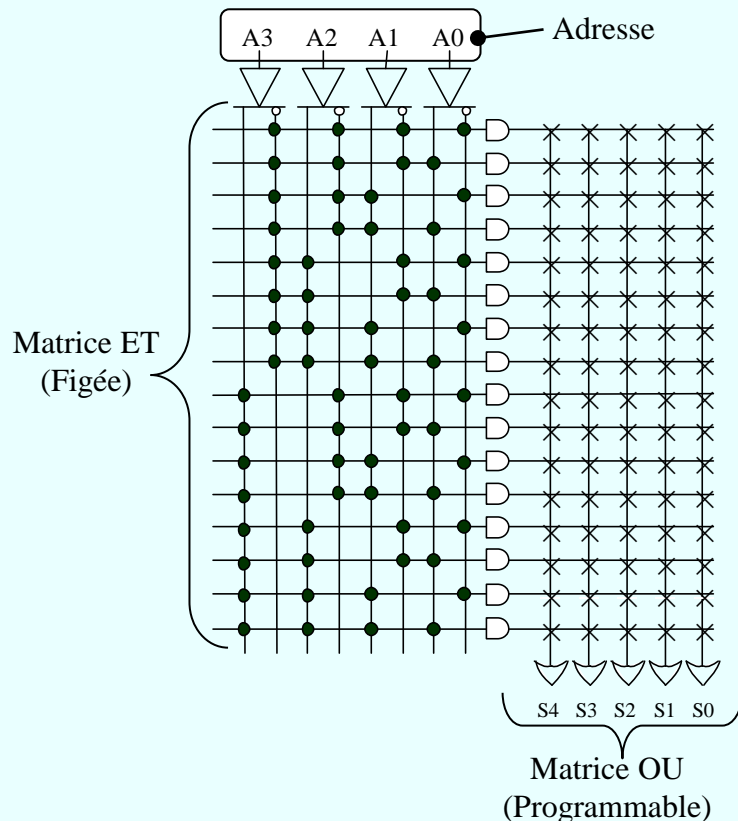
Y1 \ Y2	0	1	
0			
1			D2

### III.3. Mise en œuvre sur cible matérielle

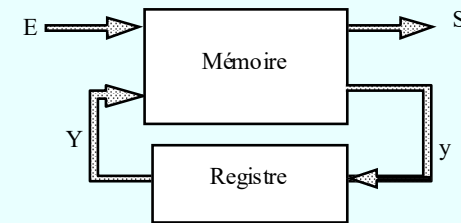
#### ❑ Utilisation de circuits logiques programmables : EEPROM

##### ➤ Architecture

- 4 bits adresse/5 bits mémoire



##### ➤ 1er Principe



- Aléas sur les entrées



Mise en œuvre synchrone seulement

- Taille mémoire

Mot Adresse	E1 .. Ep	Y1..Yn
Mot Mémoire	Y1..Yn	S1..Sm

$$2^{n+p} \times (n + m)$$



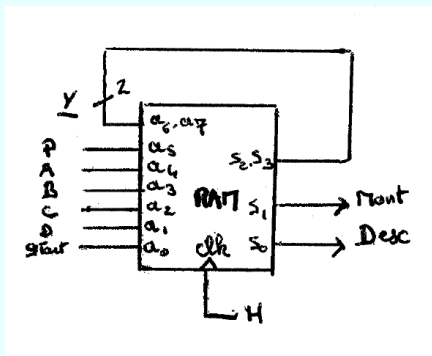
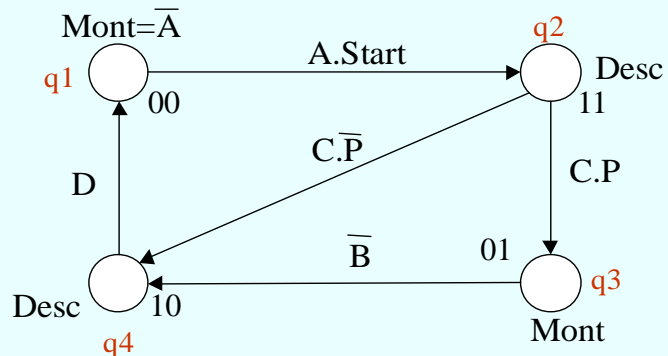
La mémoire croît exponentiellement avec le nombre d'entrées

### III.3. Mise en œuvre sur cible matérielle

#### ➤ Méthode

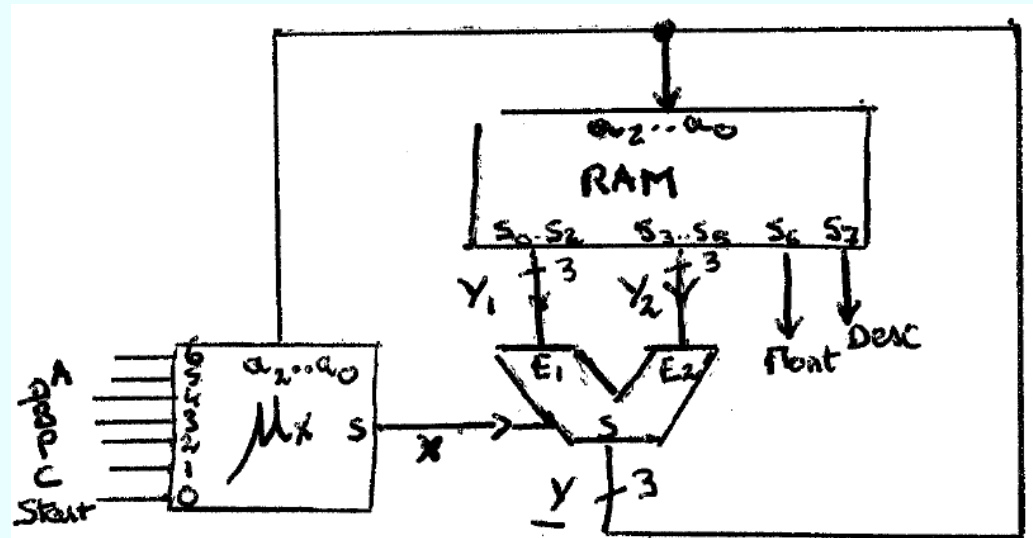
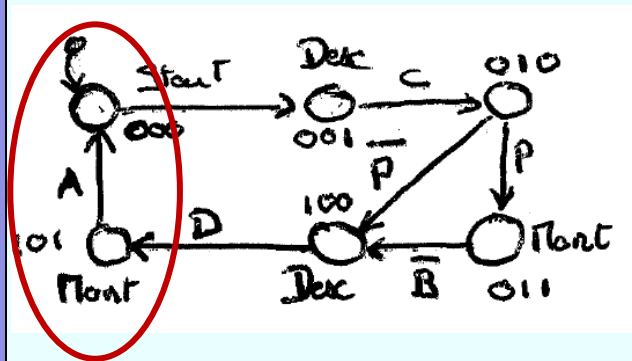
- Coder
- Faire le plan mémoire
- Éventuellement, ajouter une combinatoire sur les sorties

#### ➤ Exemple Perceuse (mise en œuvre synchrone d'un mode fondamental)



Net Adresse							Net memoire			
$y_1 y_2$	P	A	B	C	D	Start	$y_1 y_2$	Mont	Desc	
00	-	0	-	-	-	-	00	0	1	
00	-	1	-	-	-	0	00	0	0	
00	-	1	-	-	-	1	1	1	0	
01	-	-	1	-	-	-	0	1	1	
01	-	-	0	-	-	-	1	0	1	
10	-	-	-	0	-	-	1	0	0	
10	-	-	-	1	-	-	0	0	0	
11	-	-	-	0	-	-	1	1	0	
11	0	-	-	1	-	-	1	0	0	
11	1	-	-	1	-	-	0	1	0	

### III.3. Mise en œuvre sur cible matérielle

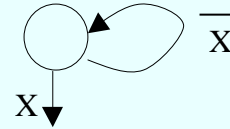
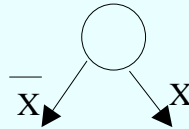


Adresse	Plout Plémoire							Plout	Desc
$Y_1 Y_2 Y_3$	$Y_{11}$	$Y_{12}$	$Y_{13}$	$Y_{21}$	$Y_{22}$	$Y_{23}$			
0 0 0	0	0	0	0	0	1	0	0	
0 0 1	0	0	1	0	1	0	0	1	
0 1 0	1	0	0	0	1	1	0	0	
0 1 1	1	0	0	0	1	1	1	0	
1 0 0	1	0	0	1	0	1	0	1	
1 0 1	1	0	1	0	0	0	1	0	

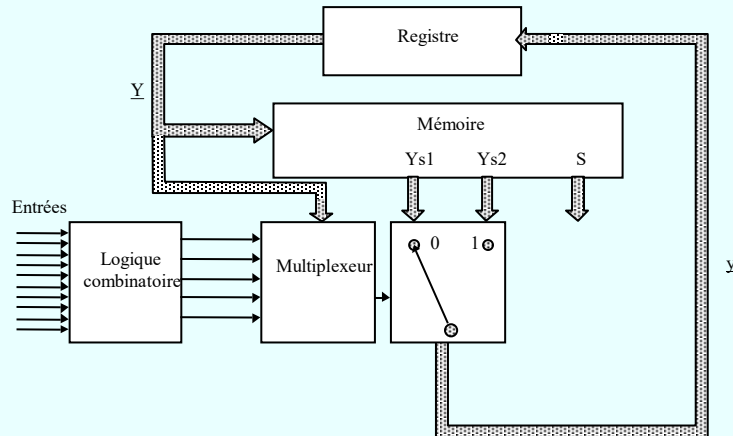
### III.3. Mise en œuvre sur cible matérielle

#### ➤ 2ème principe de mise en œuvre

- Objectif : rendre la taille mémoire indépendante du nombre d'entrées
- Hypothèse :



- Architecture



- Méthode

- *Modifier le graphe*
- *Coder les états*
- *Coder les événements (multiplexeur)*
- *Faire le plan mémoire*
- *Éventuellement, ajouter une combinatoire sur les sorties*

- Taille mémoire

- Adresse :  $Y_1 \dots Y_n$
- Mot mémoire:  $Y_{s1} \dots Y_{s1n} | Y_{s21} \dots Y_{s2n} | S$

$$(2n+m) \times 2^n$$

- Retour à l'exemple

### III.3. Mise en œuvre sur cible matérielle

#### ❑ Utilisation de circuits logiques programmables : PLA (Programmable Logic Array)

##### ➤ Architecture

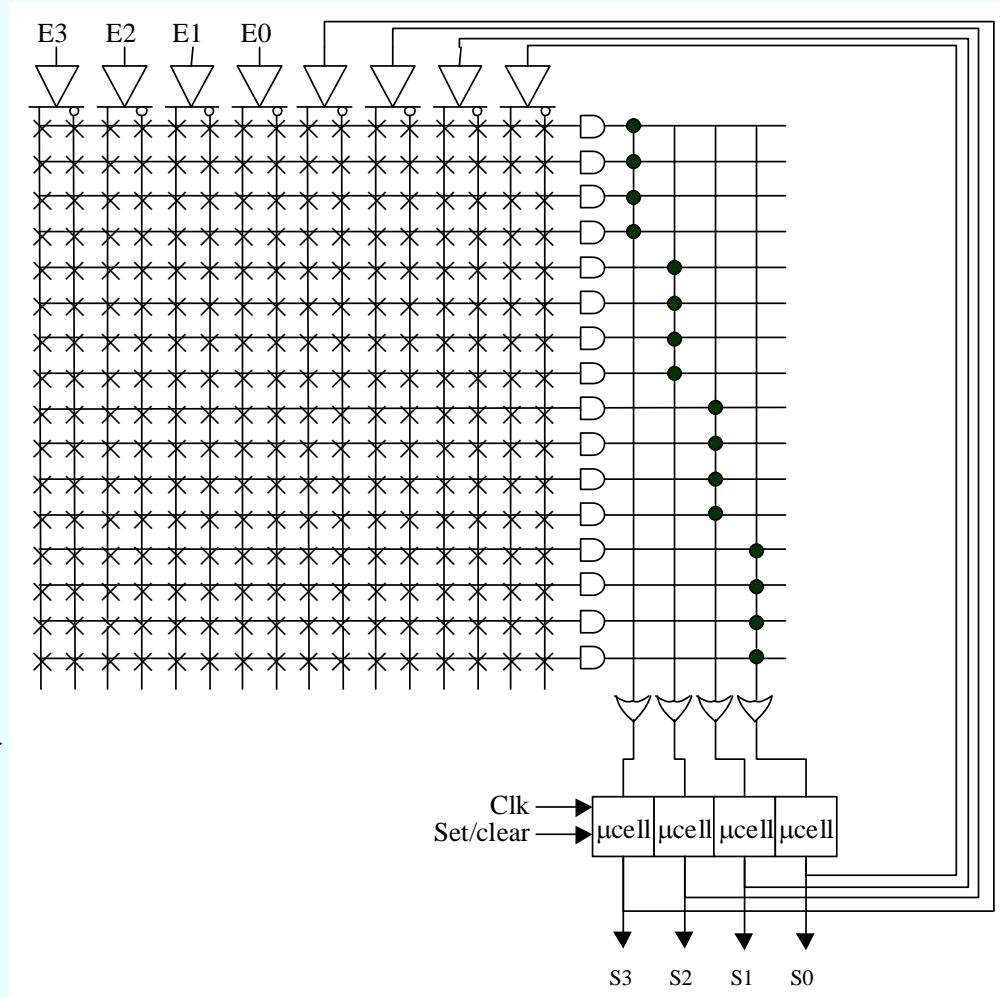
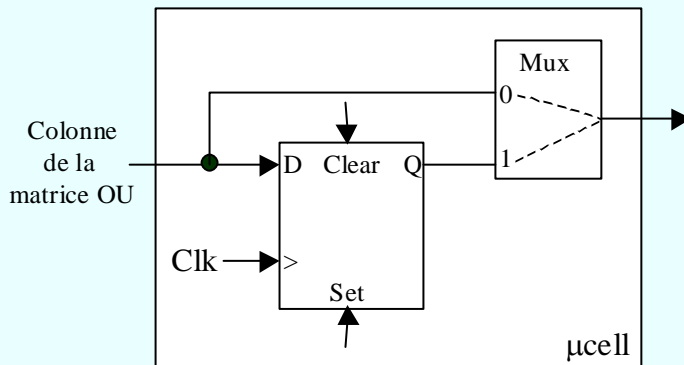
- Matrice ET programmable
- Macrocellules
- Technologie "Fuse"

##### ➤ Langage (PLDShell)

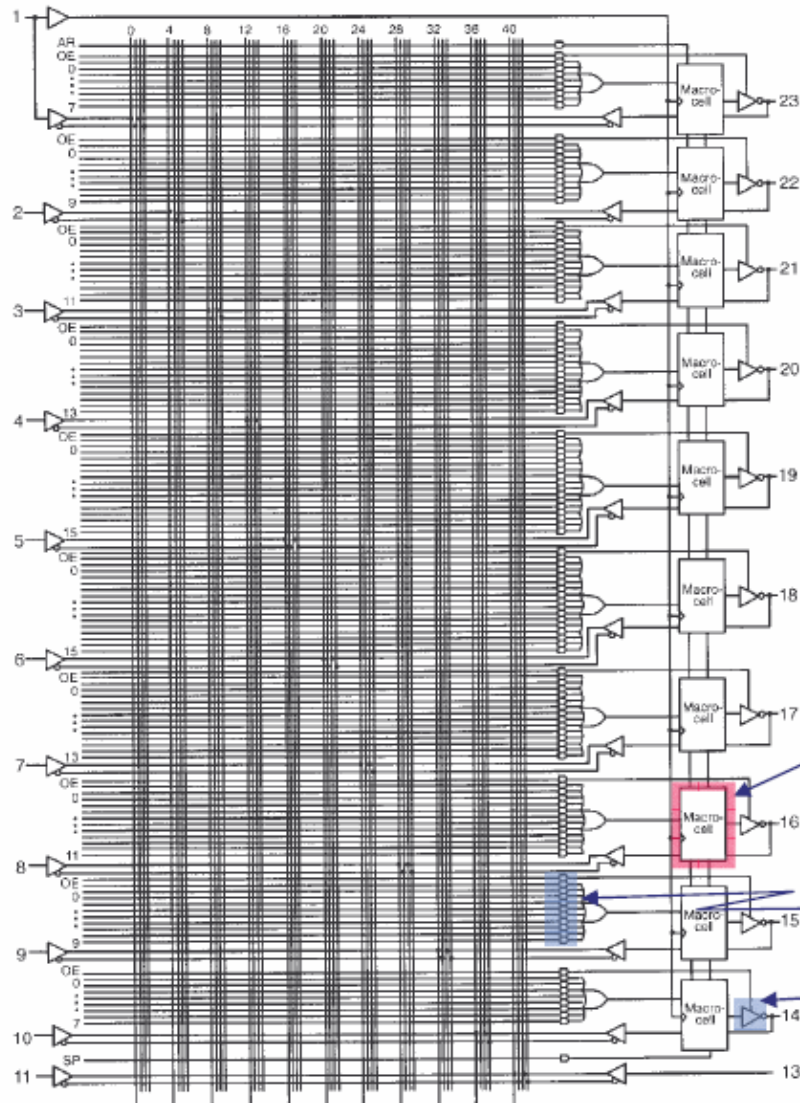
##### ➤ Simulation

##### ➤ Description

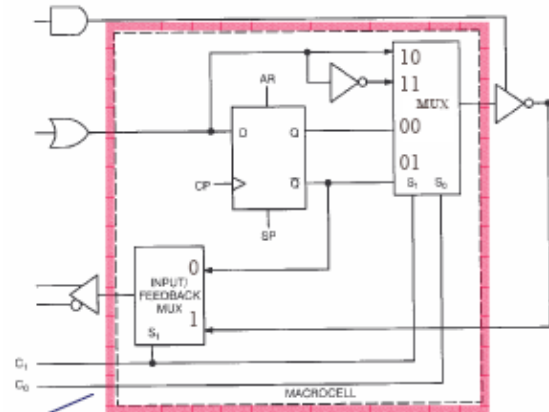
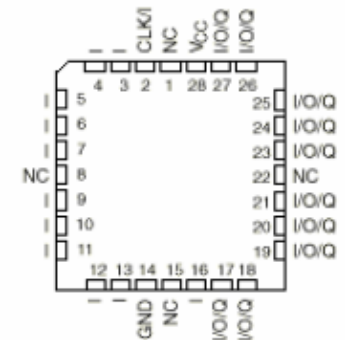
- des équations
- du graphe



### III.3. Mise en œuvre sur cible matérielle

**PLD 22V10**

- \* **22 inputs / 10 outputs**
- \* **88 to 176 product terms**
- \* **State machines, shifters, etc**



## Programmable macrocell

## Variable programmable **AND** Array

Each AND can create a product of any of the 22 inputs

### Independent three-state control

### III.3. Mise en œuvre sur cible matérielle

#### ➤ Exemple Perceuse

```
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
; Définition des sorties rebouclées
PIN 20 Y1
PIN 19 Y2
```

#### EQUATIONS

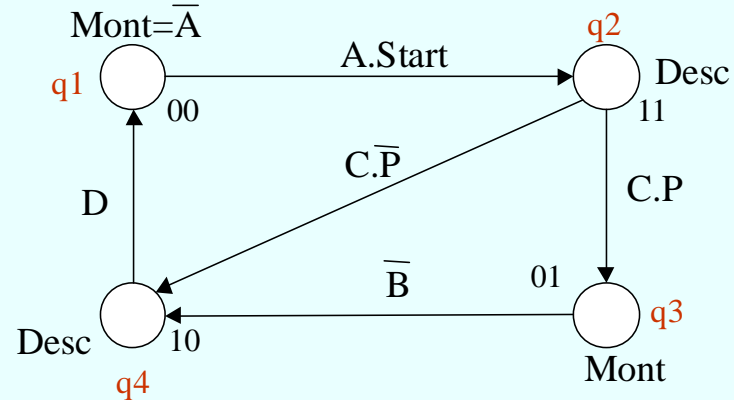
```
; Description des excitations des bascules D
Y1.D:= A*Start*/Y1*/Y2 + /B*/Y1*Y2 + Y1*Y2*/(C*P) + Y1*/Y2*/D
Y1.CLKF=H
Y1.RSTF=INIT
```

```
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)
```



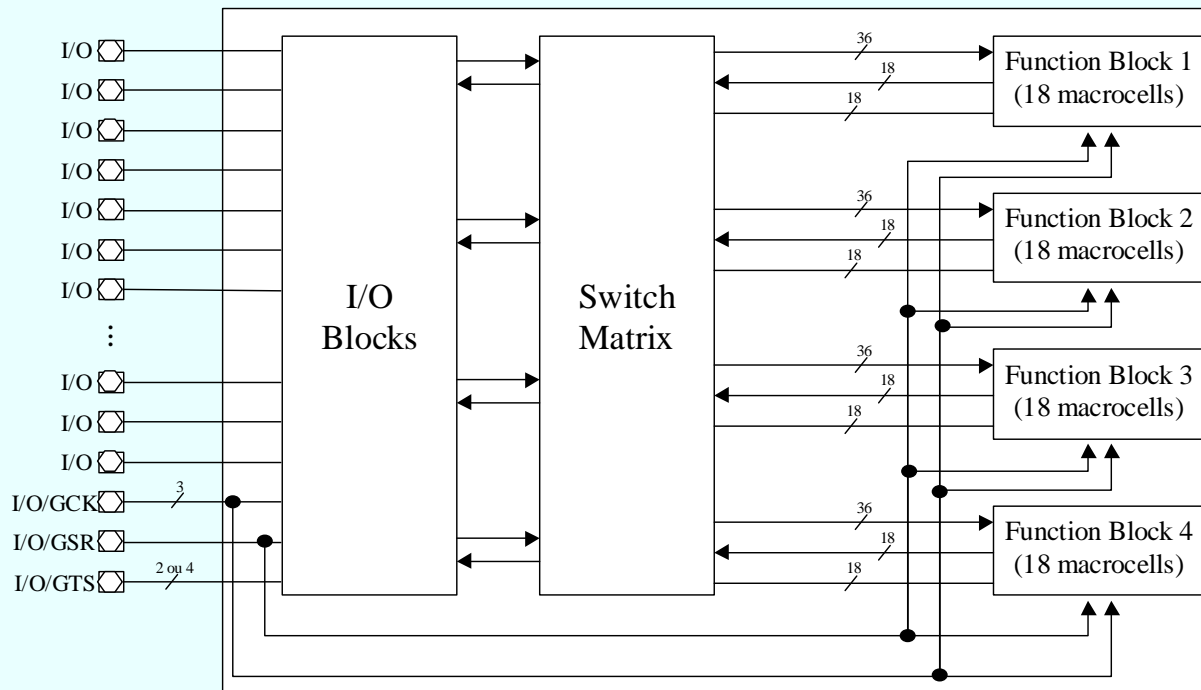


### III.3. Mise en œuvre sur cible matérielle

#### ❑ Utilisation de circuits logiques programmables : les **CPLD** (Complex Programmable Logical Device) **et FPGA** (Field Programmable Gate Array)

##### ➤ Architecture d'un CPLD

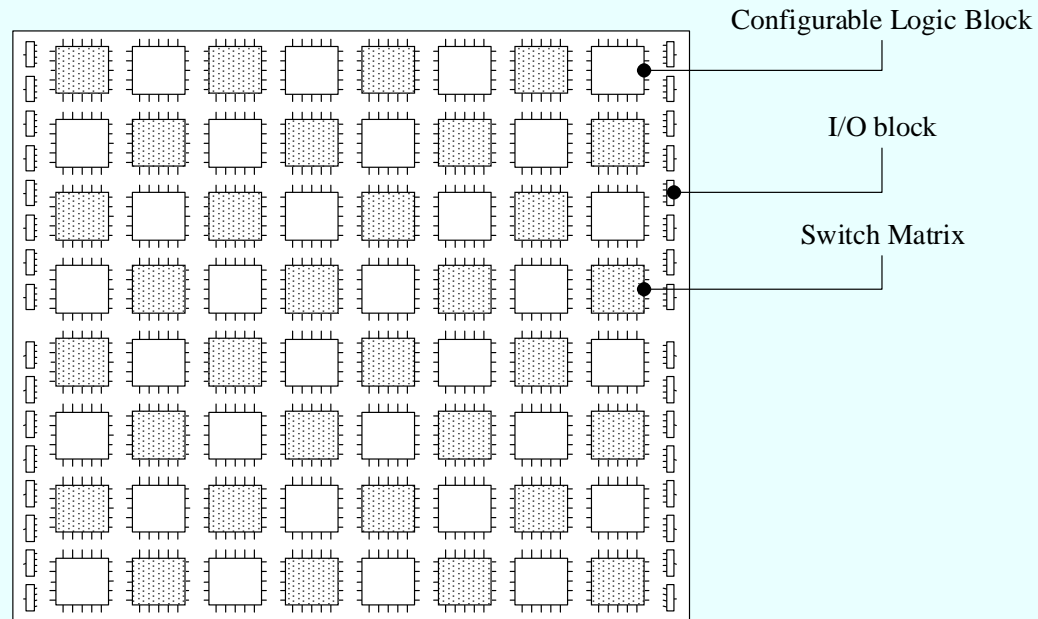
- Difficulté de créer des PLD possédant beaucoup d'entrées
  - Effet Capacitif + courant de fuite
  - Un PLD comportant 128 entrées occuperait 64 fois plus de surface qu'un PLD à 16 entrées
- Idée: Intégrer plusieurs PLDs



### III.3. Mise en œuvre sur cible matérielle

#### ➤ Architecture d'un FPGA

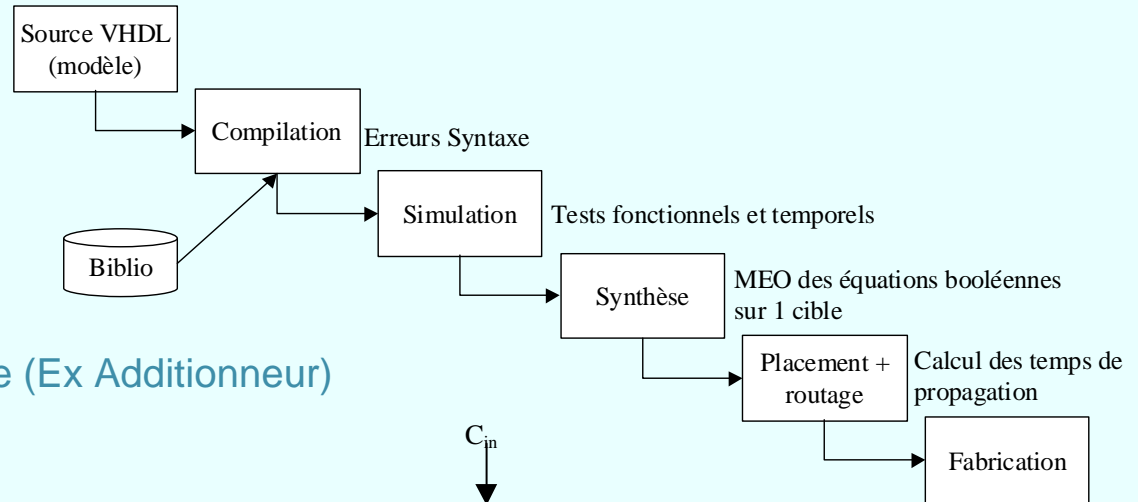
- Un constructeur origine : Xilinx
- Même idée que les CPLD mais le nombre de PLD intégré est beaucoup plus grand (plusieurs milliers contre qqes dizaines dans 1 CPLD)
- Le PLD de base (Logic Block) est plus simple (~1-2 macrocells)
- Plusieurs "Switch Matrix" et "I/O block" => Flexibilité
- Technologie "Fuse" ou "SRAM"
- Pb Routage !



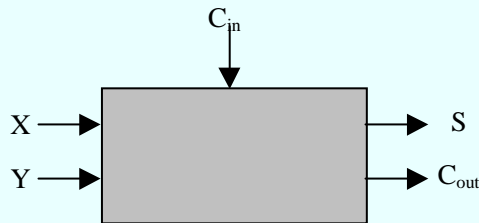
### III.3. Mise en œuvre sur cible matérielle

#### ❑ V.4.5.2. Un outil de synthèse : VHDL (VHSIC Hardware Description Language)

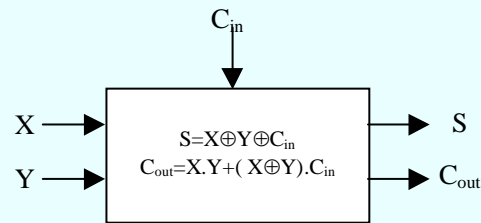
➤ 1987 : norme IEEE



➤ Vue externe / interne (Ex Additionneur)



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



```
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;
```

### III.3. Mise en œuvre sur cible matérielle

#### ➤ Instructions concurrentes

Réalisation combinatoire



Les affectations se font en parallèle

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

#### ➤ Instructions séquentielles

- Notion de PROCESS

Réalisation Synchrone



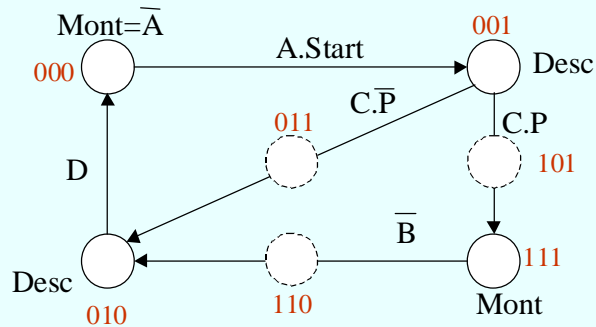
Les affectations se font en séquence et ne sont répercutées en sortie qu'à la fin du process (horloge)

```
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;
```

### III.3. Mise en œuvre sur cible matérielle

#### ➤ Mise en œuvre asynchrone de machine à états en VHDL

- Minimisation du nombre de VI



```

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;

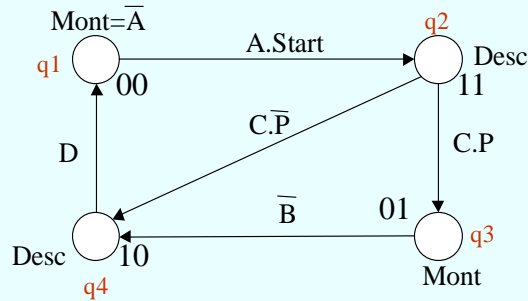
Mont <= ((Y1 and Y3) or (not A and not Y1 and not Y2 and not Y3)) and not init;

**End** MEF\_Asynchrone\_MinVI;

## III.3. Mise en œuvre sur cible matérielle

### ➤ Mise en œuvre synchrone de machine à états en VHDL

- Minimisation du nombre de VI



```

Entity CommandePerceuse is
Port
  (Start, A, B, C, D, P, Init : In std_logic;
  Mont, Desc : Out std_logic;
  -- Et en plus pour le cas synchrone :
  H : In 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;
  
```

## III.3. Mise en œuvre sur cible matérielle

### ➤ Mise en œuvre synchrone abstraite de machine à états en VHDL

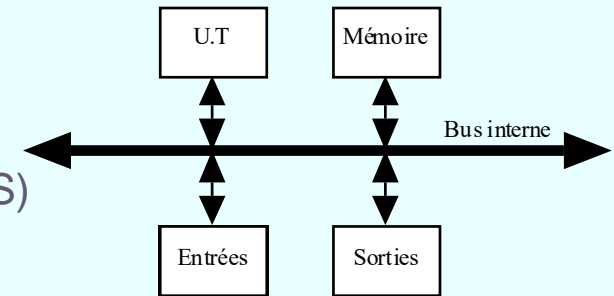
```
Architecture MEF_Synchrone_Haut_Niveau of CommandePerceuse
is
-- Déclaration des états
Type Etat is (Etat1, Etat2, Etat3, Etat4);
Signal EtatPresent, EtatSuivant : Etat;
Begin
-- Description du Bloc F
Process (A,B,C,D,Start)
Begin
Case EtatPresent is
  When Etat1 =>
    If ((A='1') and (Start='1'))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

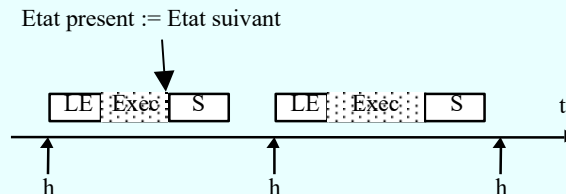
## III.4. Mise en œuvre sur cible logicielle

### ➤ Principe et chronogrammes d'exécution

- Cycle Programme
  - Lecture Entrées (LE)
  - Calcul des états suivants (EXEC)
  - Calcul des sorties (S)
- API + Microcontrôleurs + PC (avec periph. E/S)

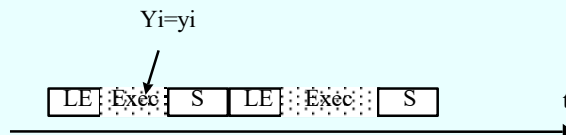


### ➤ Mise en œuvre synchrone



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...



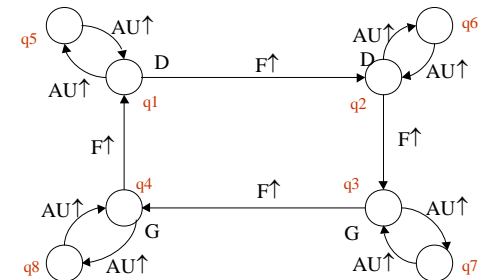
## III.4. Mise en œuvre sur cible logicielle

### ➤ Exemple – Malaxeur avec arrêt d'urgence

- Mise en œuvre synchrone - 1 parmi n (langage C - PC)

```
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;
}
```



### III.4. Mise en œuvre sur cible logicielle

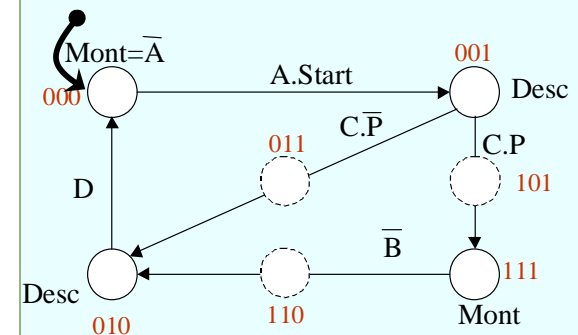
- Mise en œuvre asynchrone - Minimisation du nb de VI (langage C - PC)

```
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;
}
```

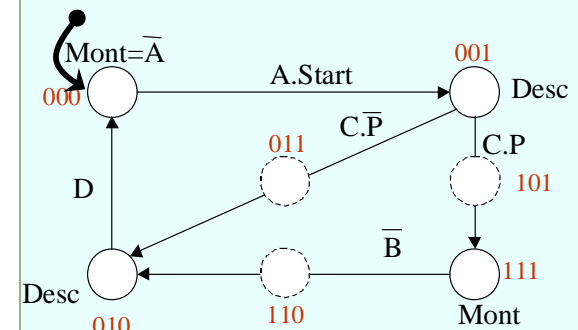
Il est plus prudent, pour éviter d'avoir à se préoccuper du problème des courses critiques, de distinguer les variables  $Y_i$  et  $Y_{i\_suivant}$  et de faire une mise en œuvre "synchrone" (l'horloge serait ici implicite  $\equiv$  Cycle programme)



## III.4. Mise en œuvre sur cible logicielle

- Mise en œuvre synchrone - Minimisation du nb de VI (langage C - PC)

```
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;
}
```



## III.4. Mise en œuvre sur cible logicielle

- Mise en œuvre synchrone – Mise en œuvre directe de la MEF

```
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;  
}
```

