Automates Synchrones

version 1.0

Plan

- **Modèle des automates synchrones**
- **Exemple 1 : détecteur de séquence**
- **Exemple 2 : allocateur de bus**
- **Modèle VHDL d'un automate abstrait**

Automate d'états synchrones

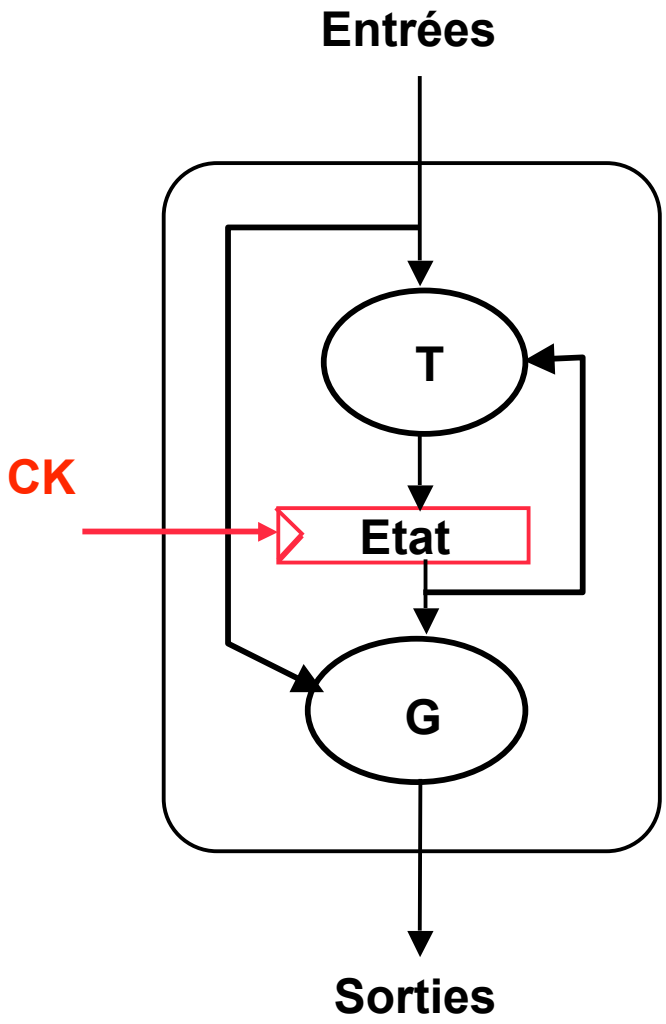
La théorie des automates est une méthode de représentation extrêmement générale du comportement d'un système matériel numérique synchrone.

Le comportement de n'importe quel système synchrone, (simple compteur 4 bits, ou microprocesseur 32 bits complet) peut - en principe - être représenté par ce modèle.

Le comportement d'un système complexe est souvent décrit en interconnectant des automates plus simples.

Il existe une méthode systématique permettant de construire le schéma en portes logiques réalisant le comportement défini par un automate abstrait.

Principe Général



Tout système numérique synchrone peut être modélisé par un automate :

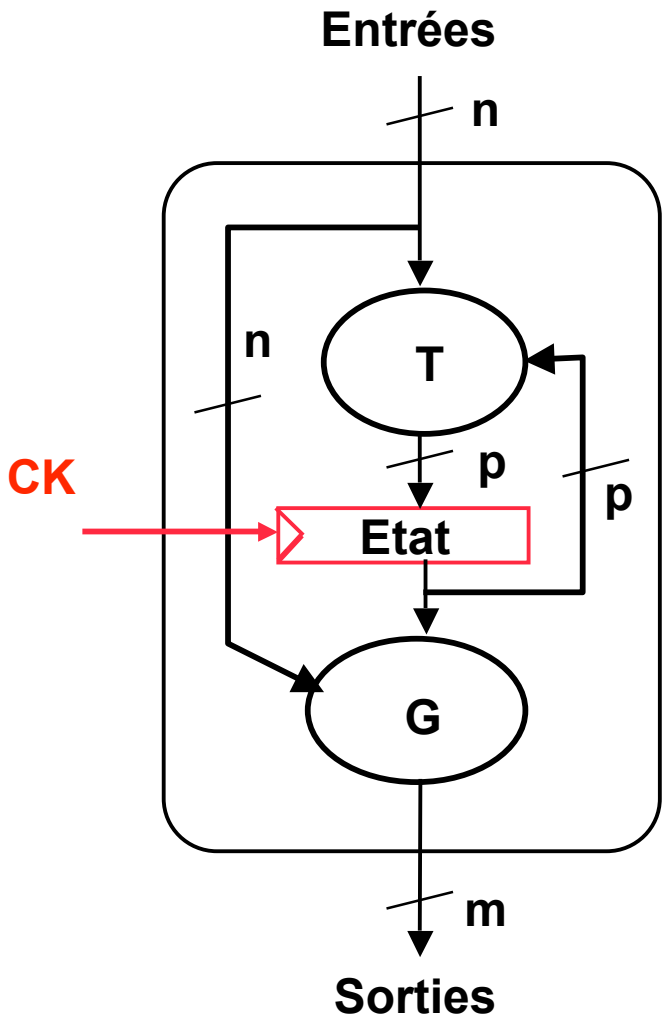
Fonction de transition :

NextEtat <= T(Etat, Entrées)

Fonction de génération :

Sorties <= G(Etat, Entrées)

Fonctions de génération et de transition



Si on a :
- n bits d'entrée Ei
- m bits de sortie Sj
- p bits mémorisés Rk

Il faut définir :

- p fonctions booléennes dépendant de (n+p) variables pour la transition

$$NR_k = T_k(E_i, R_k)$$

- m fonctions booléennes dépendant de (n+p) variables our la génération

$$S_j = \Gamma j(E_i, R_k)$$

Représentation abstraite

• Dans la définition générale d'un automate, les fonctions de transition et de génération sont définies pour :

- un ensemble de valeurs synboliques pour les entrées
- un ensemble de valeurs symboliques pour les sorties
- un ensemble de valeurs symboliques pour les états

... mais le code binaire associée à chacune de ces valeurs n'est pas défini.

• En pratique, les signaux d'entrée et de sortie sont très souvent définis au niveau Booléen : n bits d'entrée correspondent à 2^n valeurs possibles pour les entrées. Idem pour les sorties.

Seules les valeurs stockées dans les registres ne sont pas définies au niveau Booléen : le codage des états n'est pas explicite.

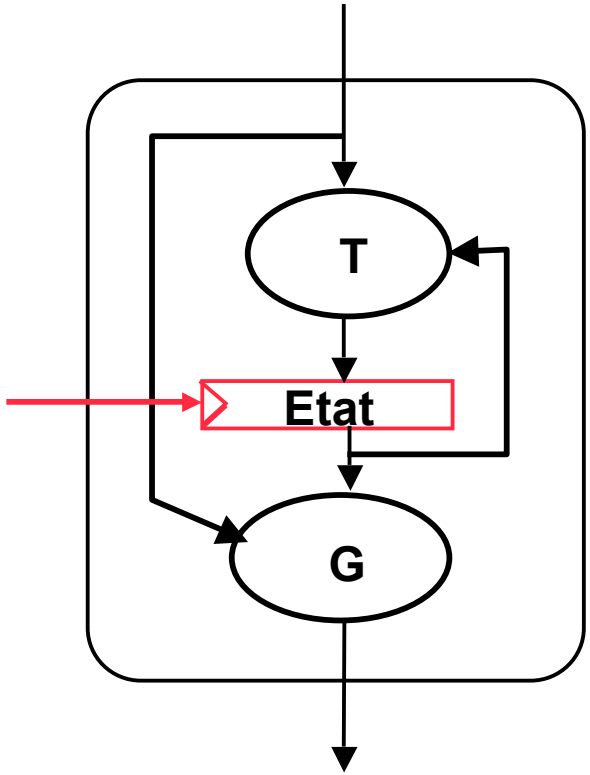
Initialisation

• Puisqu'un automate contient un état interne, il faut définir un état initial pour obtenir un comportement totalement déterministe.

• Les automates synchrones possèdent très souvent un signal d'entrée particulier (signal NRESET), qui n'agit que sur la fonction de transition : lorsque le signal NRESET est actif (état bas), on force la valeur de l'état initial dans le registre d'état lors du prochain front du signal d'horloge...

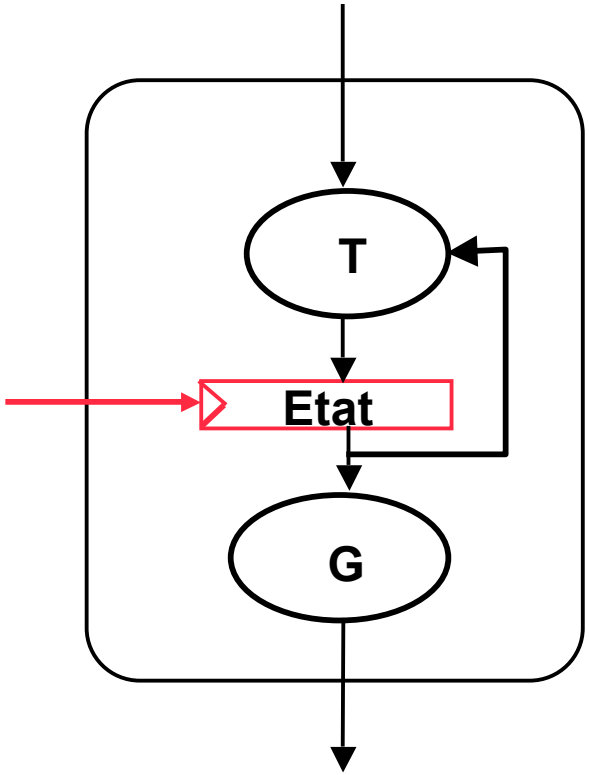
- quel que soit l'état actuel de l'automate
- quelle que soit la valeur des autres entrées.

Automates de Moore et de Mealy



Cas général :

Automate de Mealy



Cas particulier : La fonction de génération ne dépend que de l'état !

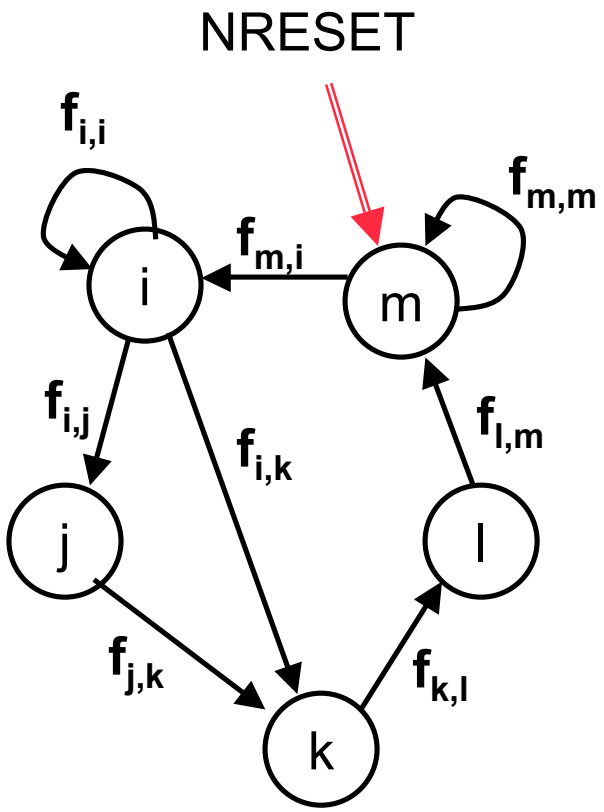
Automate de Moore

Représentation graphique d'un automate

- Les nœuds {i} représentent les états
- Les arcs (i,j) représentent les transitions

Chaque arc (i,j) est étiqueté par une expression Booléenne $\mathbf{f_{i,j}}$ ne dépendant que des signaux d'entrée, et définissant la condition de transition.

Dans le cas d'un automate de Moore, les signaux de sortie ne dépendent que de l'état : chaque nœud est donc étiqueté par la valeur des signaux de sortie.



Automate déterministe

Pour qu'un automate possède un comportement déterministe, il faut respecter les conditions suivantes :

• Existence d'un mécanisme matériel d'initialisation permettant de forcer l'automate dans un état initial connu.

• Condition d'orthogonalité : pour tout état i, et pour toute configuration des entrées, il y a un seul état successeur de l'état i.

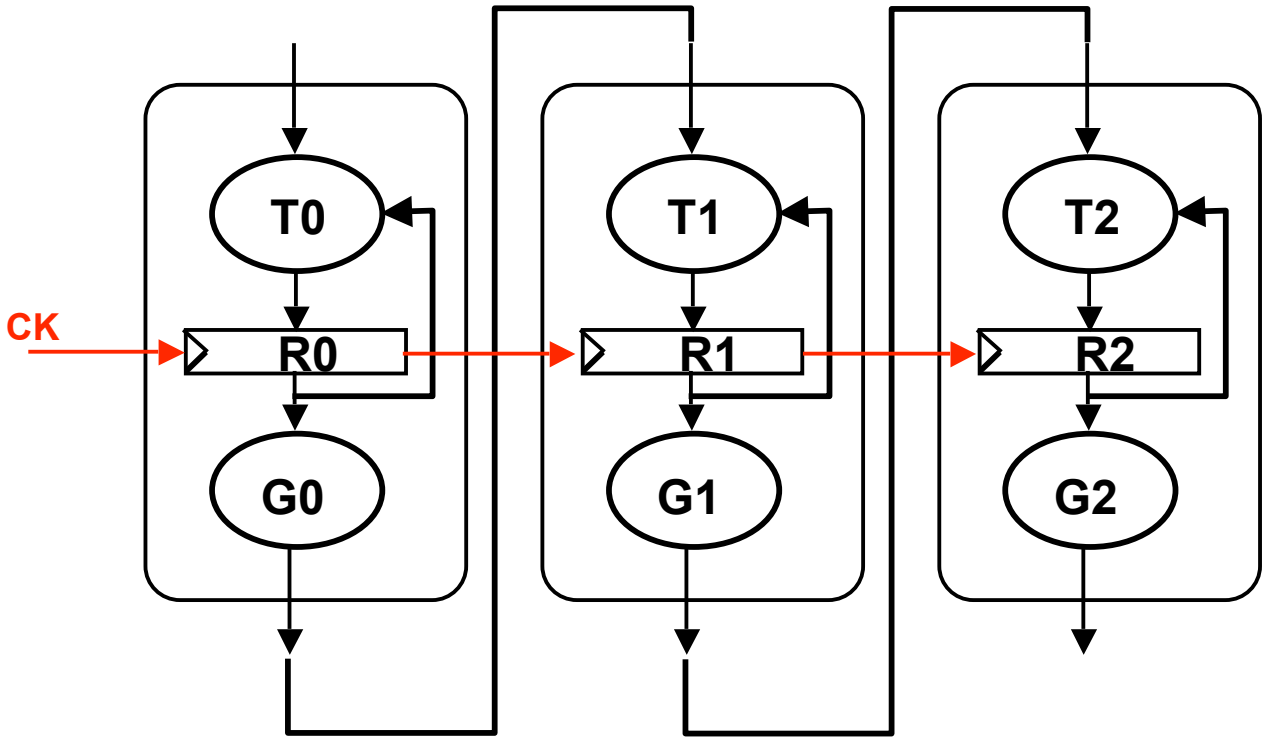
Pour tout état i, $\mathbf{f}_{i,j} . \mathbf{f}_{i,k} = 0$ si j différent de k

• Condition de complétude : pour toute configuration des entrées, il y a toujours un état successeur de l'état i.

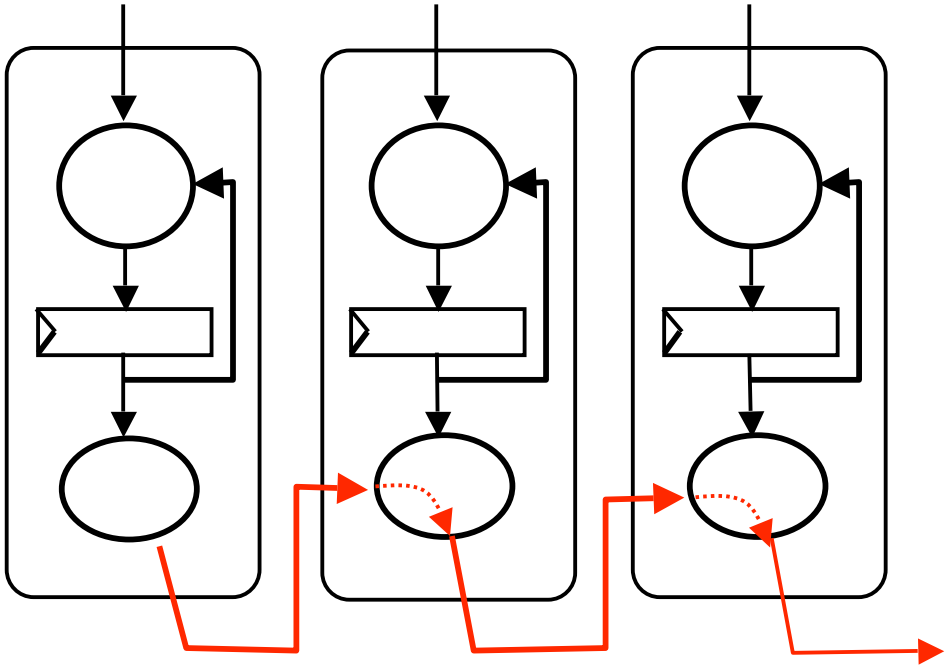
Pour tout état i, $\Sigma_j \mathbf{f}_{i,j} = 1$

Automates communicants

Un système numérique synchrone est souvent conçu comme un ensemble d'automates fonctionnant en parallèle, et communicant entre eux : les entrées d'un automate sont les sorties d'un autre automate.



Inconvénient des Automates de mealy



Dans un automate de mealy,

Il existe une dépendance combinatoire entre les entrées et les sorties !

⇒ ceci introduit des chaînes longues traversant plusieurs composants.

Il devient impossible de caractériser le comportement temporel de chaque automate indépendamment des autres.

Plan

- **Modèle des automates synchrones**
- **Exemple 1 : détecteur de séquences**
- **Exemple 2 : allocateur de bus**
- **Modèle VHDL d'un automate abstrait**

Exemple 1 : Détection d'une séquence

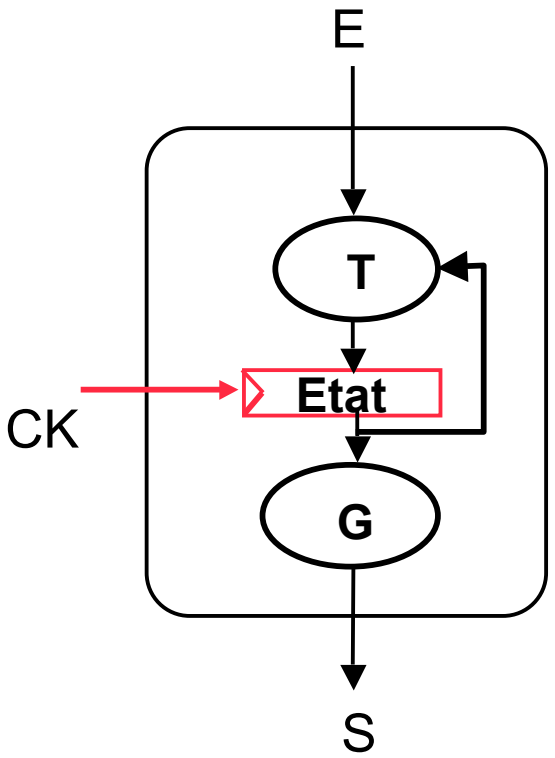
Problème à résoudre :

On cherche a réaliser un dispositif matériel possédant un seul bit d' entrée E et une seul bit de sortie S.

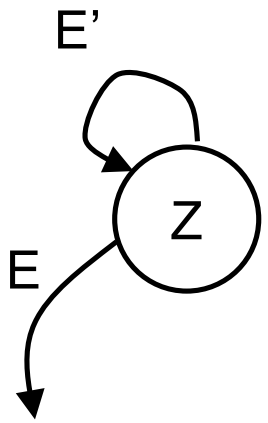
L'entrée E est échantillonnée à chaque cycle.

La sortie S passe à 1 pendant un cycle chaque fois que l'entrée E a la valeur 1 pendant 3 cycles consécutifs.

Remarque : Il n'y a pas d'entrée de type RESET !



1) représentation graphique / a

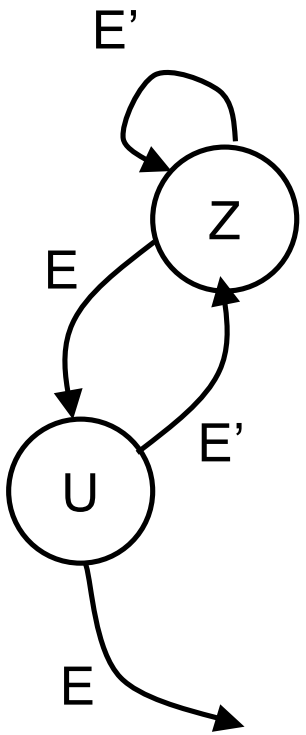


notation : E' <=> not E

Signification des états :

- Z : le dernier bit reçu valait 0

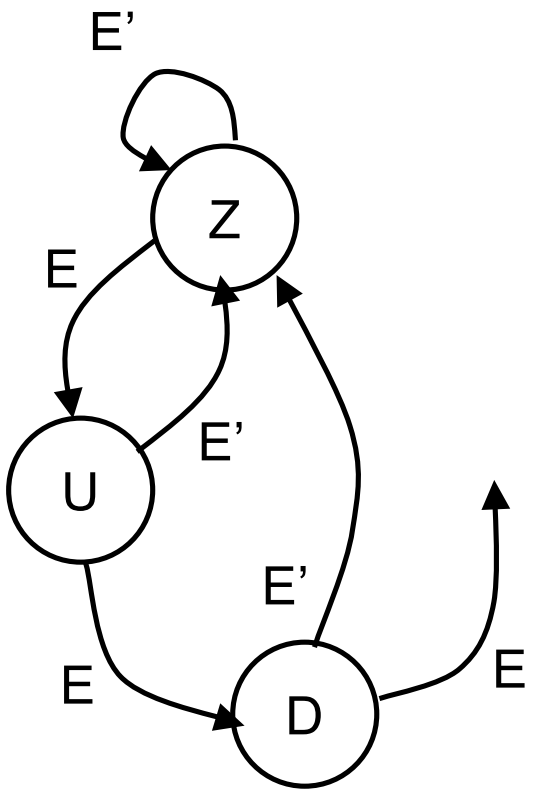
1) représentation graphique / b



Signification des états :

- Z : le dernier bit reçu valait 0
- U : les 2 derniers bits reçus valaient 0,1

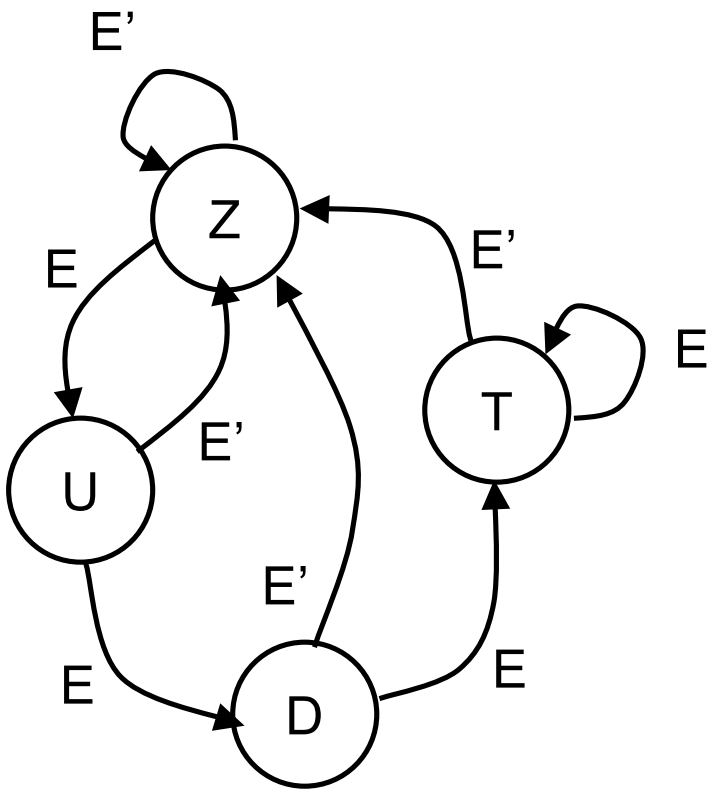
1) représentation graphique / c



Signification des états :

- Z : le dernier bit reçu valait 0
- U : les 2 derniers bits reçus valaient 0,1
- D : les 3 derniers bits reçus valaient 0,1,1

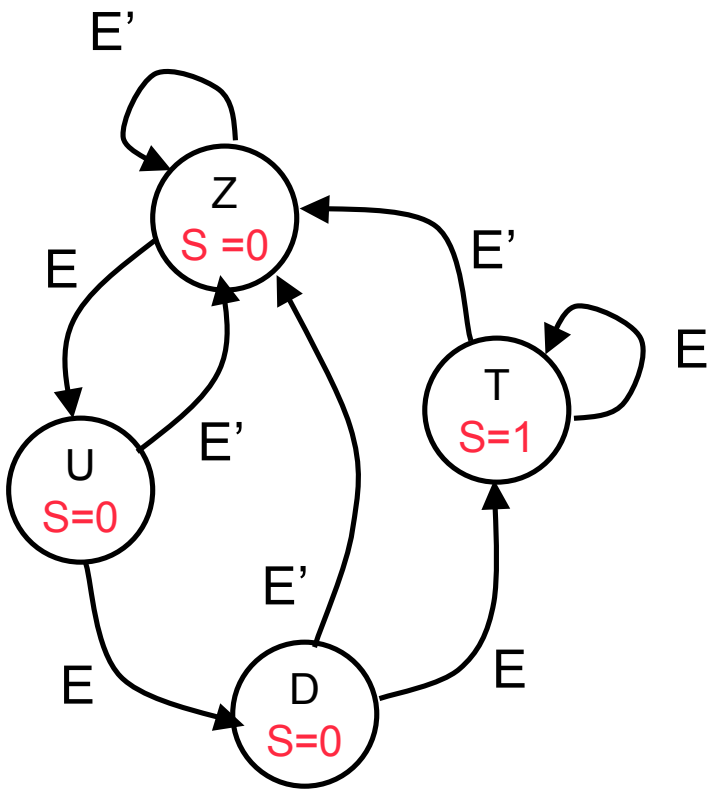
1) représentation graphique / d



Signification des états :

- Z : le dernier bit reçu valait 0
- U : les 2 derniers bits reçus valaient 0,1
- D : les 3 derniers bits reçus valaient 0,1,1
- T : les 3 derniers bits reçus valaient 1,1,1

1) représentation graphique / e



La valeur de la sortie S
ne dépend que de l'état.

2) encodage des états / a

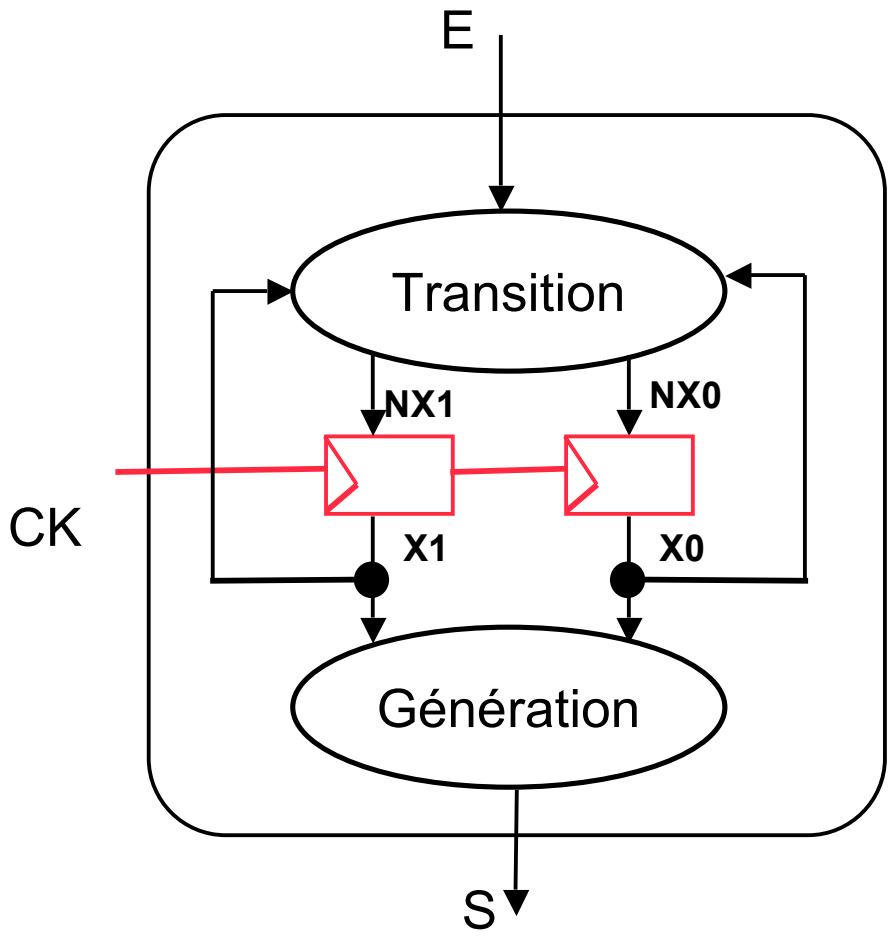
Etat	X1	X0	Y0	Y1	Y2	Y3
« Z »	0	0	1	0	0	0
« U »	0	1	0	1	0	0
« D »	1	0	0	0	1	0
« T »	1	1	0	0	0	1

code binaire (sur 2 bits)

code « one hot » (un bit par état)

Il y a plusieurs types de code possibles, et pour chaque type on peut avoir un très grand nombre d'encodages : n!
Dans la suite de l'exemple, on utilisera le code binaire ci-dessus.

2) Encodage des états / b



Deux fonctions Booléennes
pour la fonction de transition :

NX0 = T0(X1, X0, E)
NX1 = T1(X1, X0, E)

Une fonction Booléenne
pour la fonction de génération :

S = G(X1, X0)

3) table de vérité

X1	X0	E	NX1	NX0	S
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Variables d'entrée

Fonction de Transition

Fonction de Génération

4) Simplification des expressions

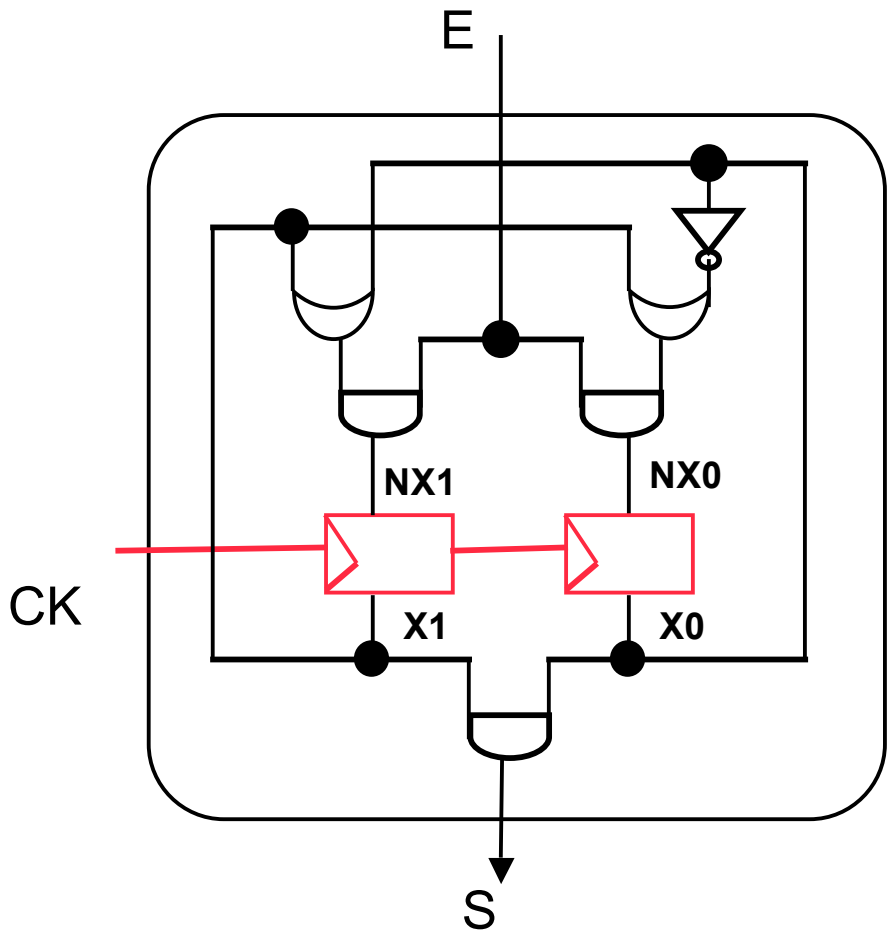
Fonction de transition :

$$NX1 = E \,.\, (X1 + X0)$$
$$NX0 = E \,.\, (X1 + X0')$$

Fonction de génération :

$$S = X1 \,.\, X0$$

5) Synthèse logique



On traduit les expressions Booléennes représentant la fonction de transition et la fonction de génération en un schéma en portes.

Pour cet exemple, il faut 3 portes « and », 2 portes « or », un inverseur et 2 bascules « D ».

Plan

- **Modèle des automates synchrones**
- **Exemple 1 : détecteur de séquences**
- **Exemple 2 : allocateur de bus**
- **Modèle VHDL d'un automate abstrait**

Exemple 2 : Allocateur de bus

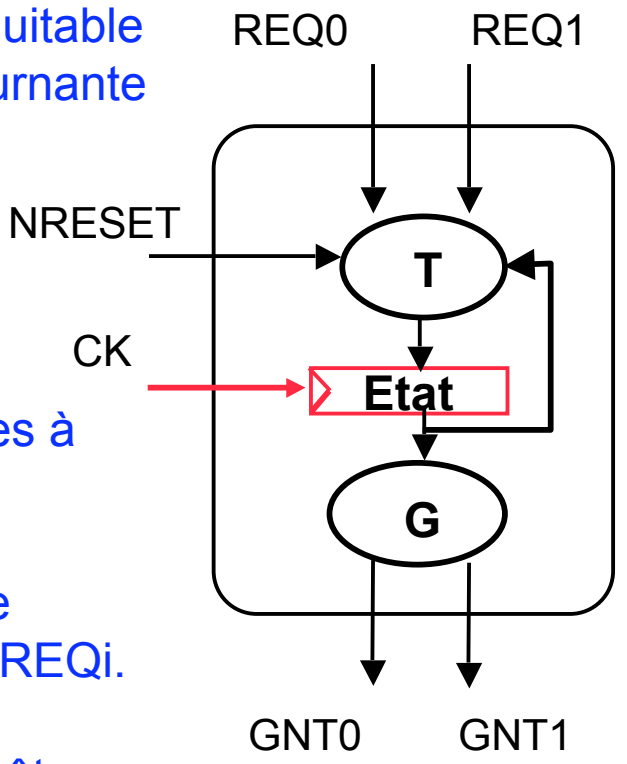
On cherche a réaliser un allocateur de bus équitable entre 2 utilisateurs (respectant une priorité tournante ou « round robin »).

Le signal NRESET initialise l'automate dans un état où le bus n'est pas alloué.

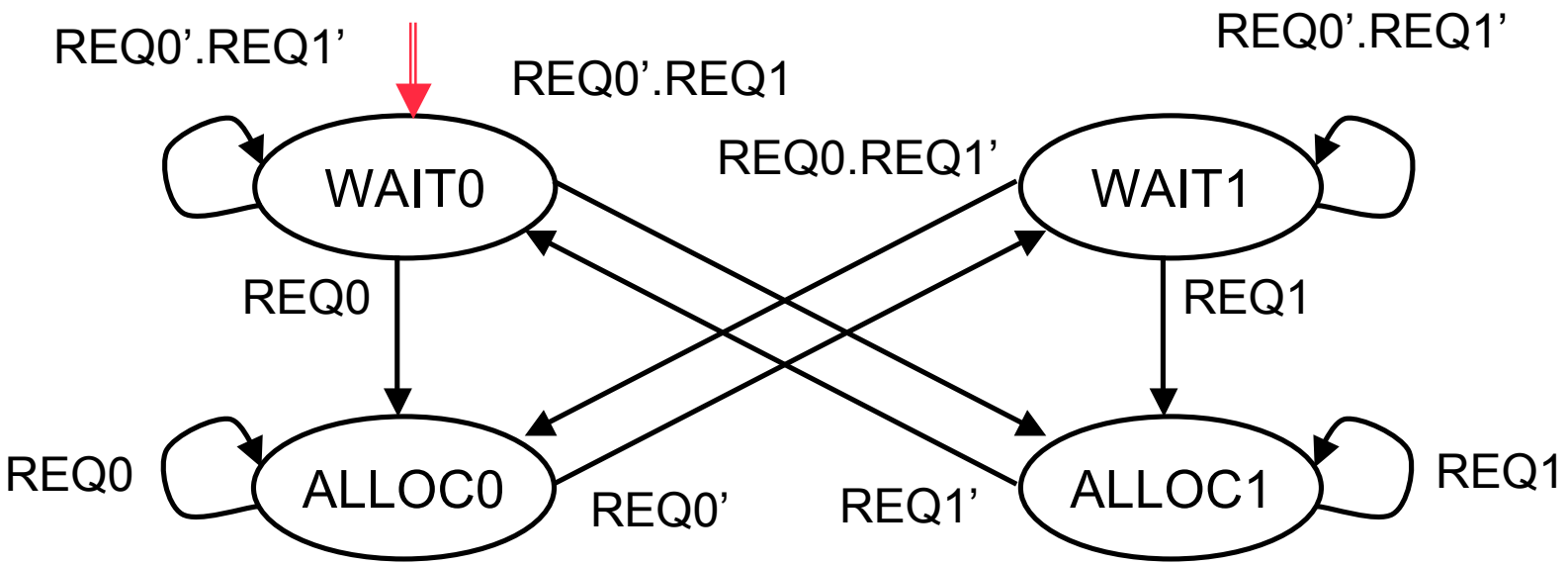
Les deux requêtes REQ0 et REQ1 sont actives à l'état haut, et indépendantes.

L' utilisateur possédant le bus signale la fin de l'utilisation en forçant la valeur 0 sur le signal REQi.

Les deux signaux GNT0 et GNT1 ne peuvent être actifs en même temps, et il y a toujours un cycle non alloué (GNT0 = GNT1 = 0) entre deux allocations



1) représentation graphique



Signification des états :

- WAIT0 : bus non alloué / utilisateur 0 prioritaire
- WAIT1 : bus non alloué / utilisateur 1 prioritaire
- ALLOC0 : bus alloué à l'utilisateur 0
- ALLOC1 : bus alloué à l'utilisateur 1

2) encodage des états

Etat	X1	X0
« WAIT0 »	0	0
« WAIT1 »	0	1
« ALLOC0 »	1	0
« ALLOC1 »	1	1

codage binaire
(sur 2 bits)

3) table de vérité (sans NRESET)

	X1	X0	REQ0	REQ1	NX1	NX0	GNT1	GNT0
	0	0	0	0	0	0	0	0
	0	0	0	1	1	1	0	0
WAIT0	0	0	1	0	1	0	0	0
	0	0	1	1	1	0	0	0
	0	1	0	0	0	1	0	0
	0	1	0	1	1	1	0	0
WAIT1	0	1	1	0	1	0	0	0
	0	1	1	1	1	1	0	0
	1	0	0	0	0	1	0	1
	1	0	0	1	0	1	0	1
ALLOC0	1	0	1	0	1	0	0	1
	1	0	1	1	1	0	0	1
	1	1	0	0	0	0	1	0
	1	1	0	1	1	1	1	0
ALLOC1	1	1	1	0	0	0	1	0
	1	1	1	1	1	1	1	0

4) simplification des expressions

Sans le NRESET:

NX1 = X1'.REQ1 + S1'.REQ0 + X0.REQ1 + X0'.REQ0
NX0 = REQ0'.REQ1 + X0.REQ1 + X1'.X0.REQ0' + X1.X0'.REQ0'
GNT0 = X1.X0
GNT1 = X1.X0'

Avec le NRESET:

NX1 = NRESET.(X1'.REQ1 + S1'.REQ0 + X0.REQ1 + X0'.REQ0)
NX0 = NRESET.(REQ0'.REQ1 + X0.REQ1 + X1'.X0.REQ0' + X1.X0'.REQ0')
GNT0 = X1.X0
GNT1 = X1.X0'

Plan

- **Modèle des automates synchrones**
- **Exemple 1 : détecteur de séquences**
- **Exemple 2 : allocateur de bus**
- **Modèle VHDL d'un automate abstrait**

Modèle VHDL allocateur : architecture fsm / a

```
entity allocateur is
-- liste des ports d'entrée-sortie
port (
      ck : in bit ;
      req0 : in bit ;
      req1 : in bit ;
      nreset : in bit ;
      gnt0 : out bit ;
      gnt1 : out bit ;
      vss : in bit ;
      vdd : in bit
      ) ;
end allocateur ;
```

Modèle VHDL allocateur : architecture fsm / b

```
architecture fsm of allocateur is

-- définition du type énuméré
type etat_type is (    wait0,
                       wait1,
                       alloc0,
                       alloc1) ;

-- déclaration des signaux
signal present : etat_type ;
signal futur : etat_type ;

-- directives pour la synthèse :
-- pragma current_state present
-- pragma next_state futur
-- pragma clock ck

begin
```

2

```
-- processus « combinatoire »
process ( present, req0, req1, nreset ) begin

-- fonction de transition
if (nreset = '0') then   futur <= wait0 ;
else
     case present is
     when wait0 =>
        if (req0)        then futur <= alloc0 ;
        elsif (req1)     then futur <= alloc1 ;
        end if ;
     when wait1 =>
        if (req1)        then futur <= alloc1 ;
        elsif (req0)     then futur <= alloc0 ;
        end if ;
     when alloc0 =>
        if (req0)= '0') then futur <= wait1 ;
        end if ;
```

3

Modèle VHDL allocateur : architecture fsm / c

```
    when alloc1 =>
        if (req1= '0') then futur <= wait0 ;
        end if ;
    end case ;
end if ;

-- fonction de génération
if (present = alloc0) then gnt0 <= '1' ;
else                       gnt0 <= '0' ;
end if ;
if (present = alloc1) then gnt1 <= '1' ;
else                       gnt1 <= '0' ;
end if ;

end process ;
```

4

```
-- processus « update »
process ( ck ) begin

if (ck = '1' and not ck' stable)
      then present = futur :
end if ;

end process ;

end fsm ;
```

5