

# FSM avec chemin de données (FSMD)

(c) E.Deknevel 2018

comment concevoir des machines séquentielles pour lesquels le nombre d'états est important et permet difficilement une 'approche par les états?

## Limite de la synthèse logique (combinatoire)

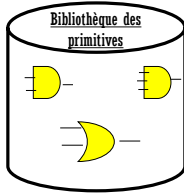
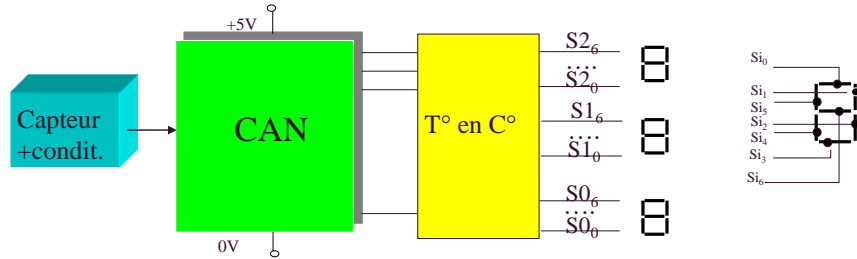
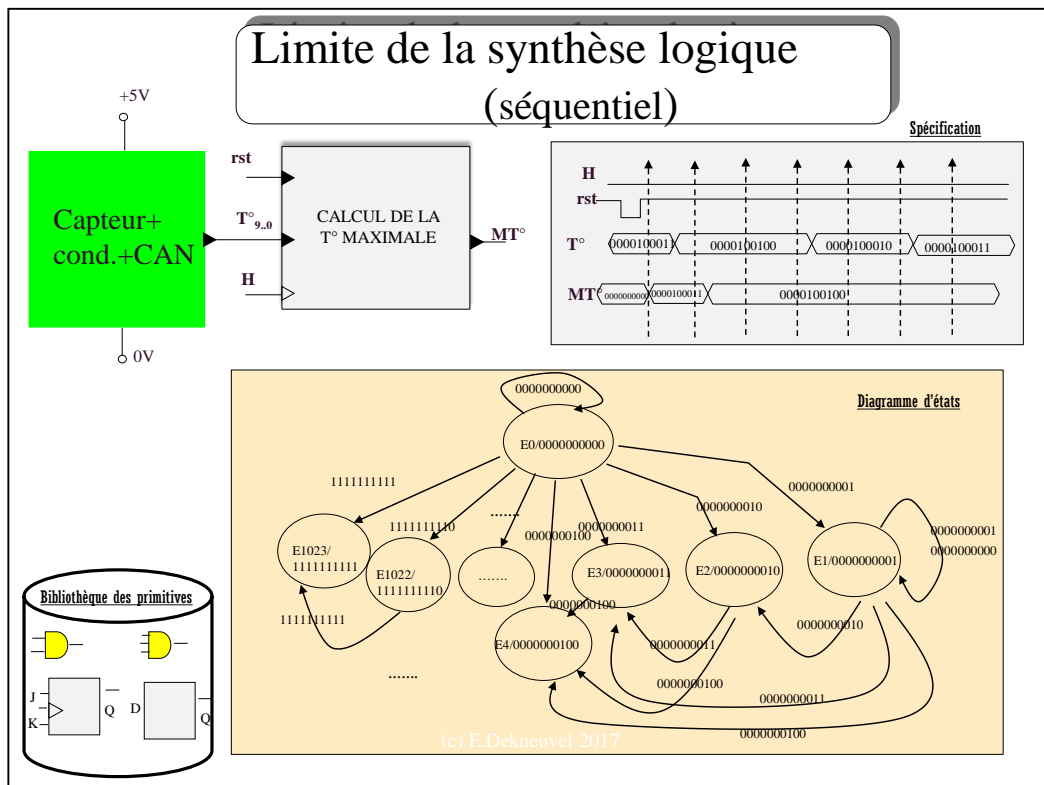


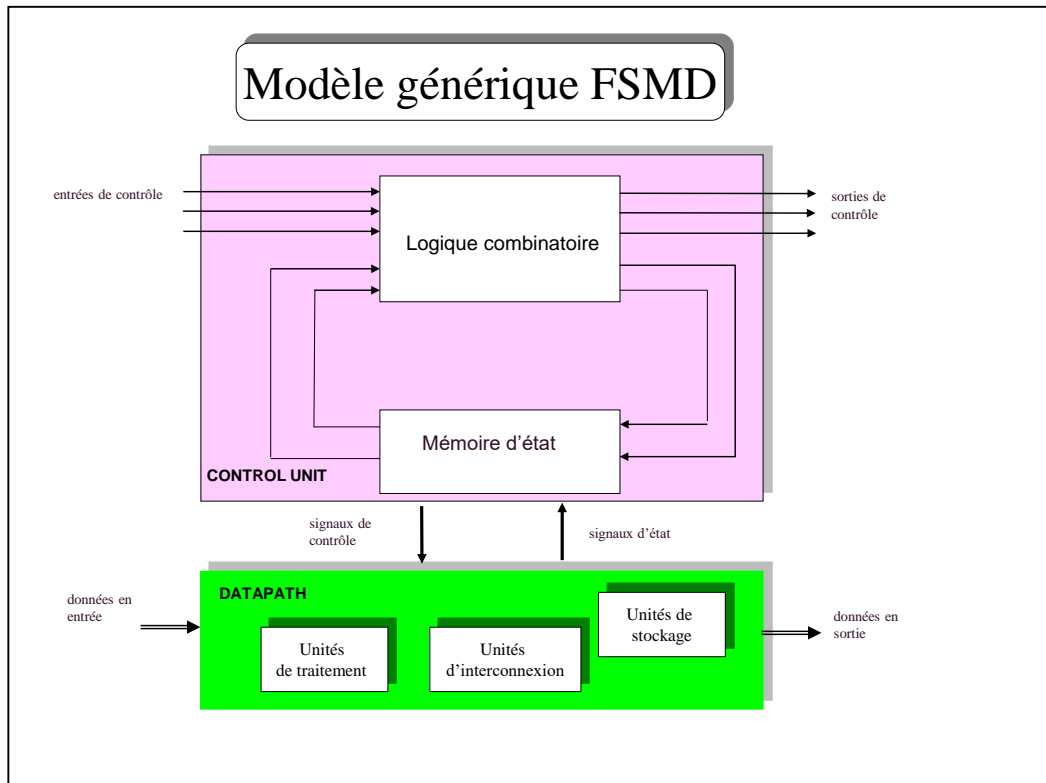
Table de vérité

10 variables booléennes										21 fonctions booléennes																				
T <sub>9</sub>	T <sub>8</sub>	T <sub>7</sub>	T <sub>6</sub>	T <sub>5</sub>	T <sub>4</sub>	T <sub>3</sub>	T <sub>2</sub>	T <sub>1</sub>	T <sub>0</sub>	S <sub>2</sub> <sub>6</sub>	S <sub>2</sub> <sub>5</sub>	S <sub>2</sub> <sub>4</sub>	S <sub>2</sub> <sub>3</sub>	S <sub>2</sub> <sub>2</sub>	S <sub>2</sub> <sub>1</sub>	S <sub>2</sub> <sub>0</sub>	S <sub>1</sub> <sub>6</sub>	S <sub>1</sub> <sub>5</sub>	S <sub>1</sub> <sub>4</sub>	S <sub>1</sub> <sub>3</sub>	S <sub>1</sub> <sub>2</sub>	S <sub>1</sub> <sub>1</sub>	S <sub>1</sub> <sub>0</sub>	S <sub>0</sub> <sub>6</sub>	S <sub>0</sub> <sub>5</sub>	S <sub>0</sub> <sub>4</sub>	S <sub>0</sub> <sub>3</sub>	S <sub>0</sub> <sub>2</sub>	S <sub>0</sub> <sub>1</sub>	S <sub>0</sub> <sub>0</sub>
0	0	0	0	0	0	0	0	0	0	1 0 0 0 0 0 0 0 <sub>7seg</sub>							1 0 0 0 0 0 0 0 <sub>7seg</sub>							1 0 0 0 0 0 0 0 <sub>7seg</sub>						
1	0	1	1	0	0	1	0	1	1	0 1 0 0 1 0 0 2 <sub>7seg</sub>							1 1 1 1 0 0 0 7 <sub>7seg</sub>							0 1 0 0 0 0 0 9 <sub>7seg</sub>						
715 <sub>10</sub>																														
1	1	1	1	1	1	1	1	1	1	0 1 1 0 0 0 0 3 <sub>7seg</sub>							0 1 0 0 0 0 0 9 <sub>7seg</sub>							0 1 0 0 0 0 0 9 <sub>7seg</sub>						

pour des problèmes complexes tels que l'affichage en 7 segments de la valeur de  $T^{\circ}$ , il devient complexe d'utiliser l'approche de synthèse logique, le nombre de combinaisons de sortie étant important. Il faut en effet multiplier le code en sortie du can par 0.0390625, et prendre le code 7 segments de chaque chiffre décimal pour déterminer les valeurs binaires de chaque fonction en sortie.



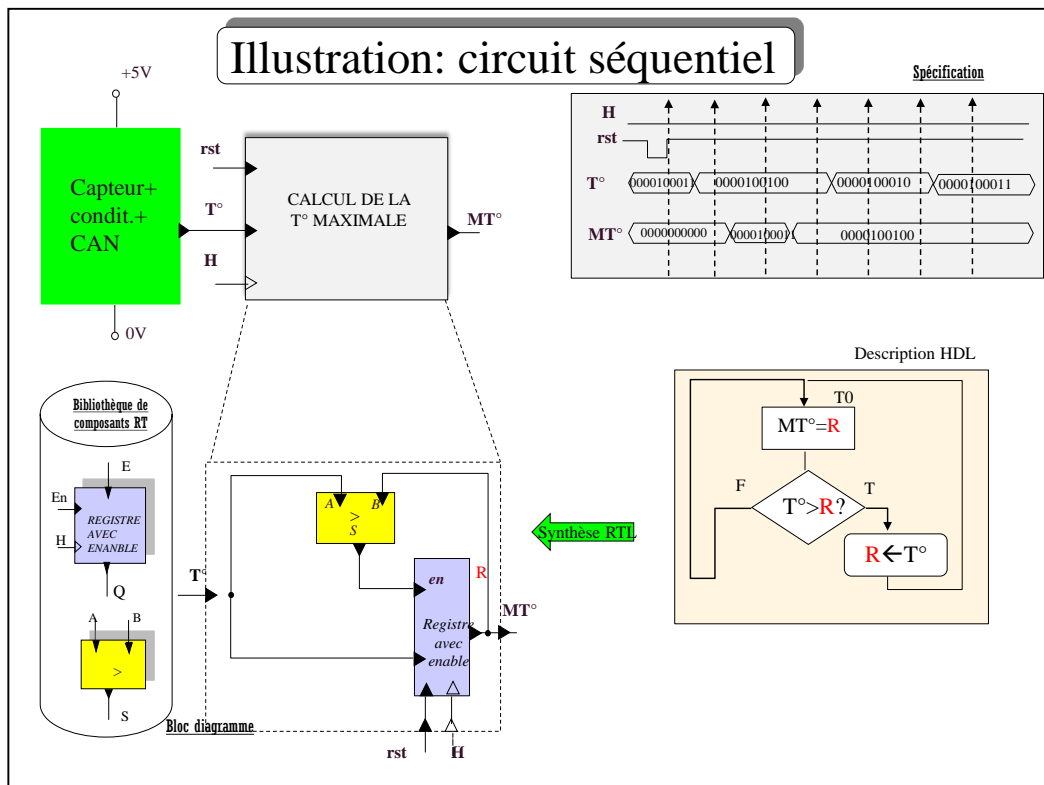
Dans le cas de systèmes complexes, l'approche par les états développée dans le chapitre « automates à états finis » qui consiste à énumérer toutes les combinaisons d'états suivants/sorties pour chaque combinaison d'entrée et d'état courant devient quasi-infaisable. Ici, on a pas moins de  $2^{10}$  états et un nombre de transitions considérable.



L'implémentation de fonctions complexes nécessite généralement une architecture résultant typiquement au plus haut niveau de l'association de deux blocs fondamentaux:

- \* Un chemin de données (datapath) qui consiste en une logique de traitement et une collection de registres pour exécuter les micro-opérations
- \* une unité de contrôle (control unit) qui détermine l'instant d'exécution des micro-opérations au travers d'équations de contrôle des signaux des unités du chemin de données (et non les équations d'entrées des FF) en fonction des entrées externes, de signaux d'état du chemin de données et d'un état interne.

On notera que les solutions à base de FSMD sont généralement moins performantes et plus couteuses que les solutions à base de FSM. C'est essentiellement la complexité de la description et de la synthèse qui motivera le choix de ce type d'architecture qu'on appelle encore architecture RTL



L'utilisation d'une approche de description de type transfert de registre ici (encore appelée approche orientée bit) consiste à décrire des opérations sur la variables (  $R$  sur 10 bits ici) plutôt que des transitions d'états. Comme pour les FSMs, le contrôle peut être exprimé de manière équivalente sous forme tabulaire plutôt que sous forme graphique.

Cette approche permet de solutionner la complexité des circuits pour lesquels des traitements doivent être effectués et pour lesquels l'approche par les états est mal adaptée.

# FSM avec chemin de données (FSMD)

*A. Blocs combinatoires et séquentiels*

*B. Description transfert de registres (HDL)*

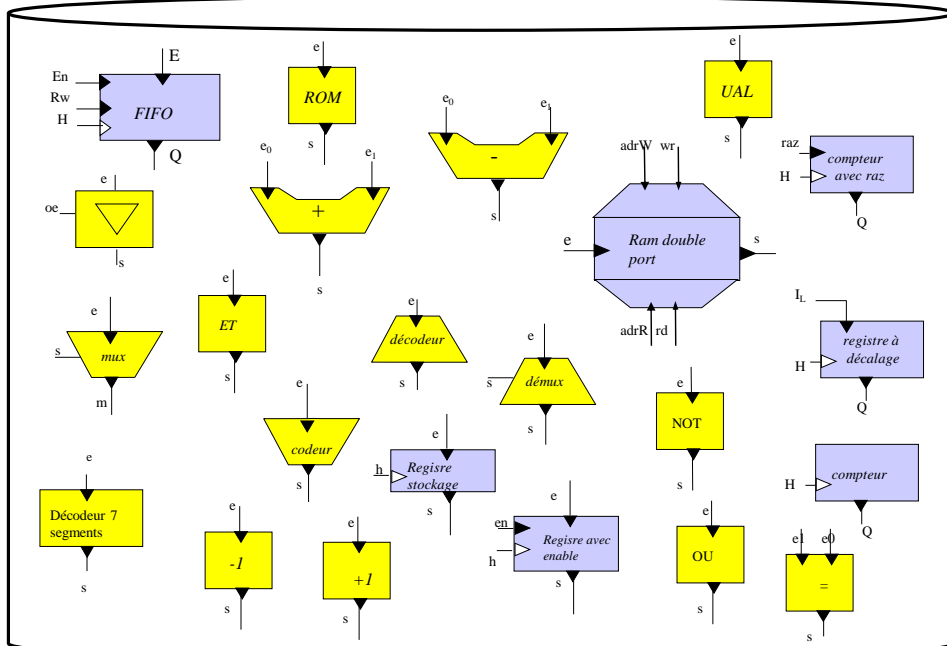
*C. Conception des FSMD*

*D. Registres complexes*

*E. Le microprocesseur*

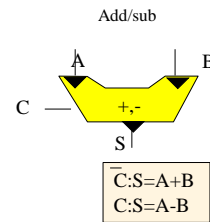
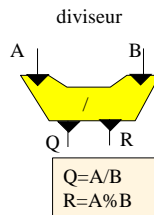
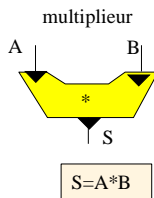
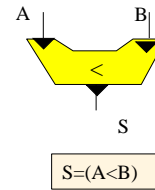
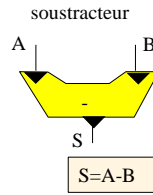
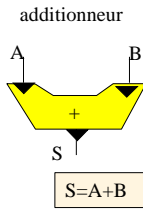
Quel formalisme pour exprimer les micro-opérations de traitement des variables et pour exprimer le flot de contrôle (le séquençement) de ces micro-opérations?

## Composants du niveau RT



De même qu'il existe de nombreux blocs combinatoires standards (en jaune), Il existe également de nombreux blocs fonctionnels standards (liste non exhaustive ci-dessus) séquentiels (en bleu) intervenant dans de nombreuses conception de circuits complexes et implémentant des fonctions de base en groupant plusieurs bits ensemble. On parle de logique structurée car ces groupes de bits concourent à la réalisation d'une même opération. Certains composants peuvent être représentés par des symboles spéciaux permettant leur identification rapide dans un schéma.

## Blocs combinatoires(opérateurs)



Les bibliothèques des fabricants de composants offrent une grande variété d'opérateurs implémentant des opérations arithmétiques et logiques sous formes d'opérateurs élémentaires ou d'opérateurs multifonctions (un ou plusieurs signaux de contrôle sélectionne l'opération)



## Blocs combinatoires (contrôle)

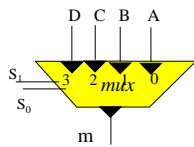


Table de vérité

S <sub>1</sub>	S <sub>0</sub>	M
0	0	A
0	1	B
1	0	C
1	1	D

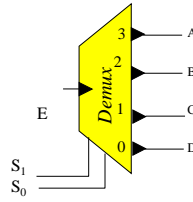


Table de vérité

S <sub>1</sub>	S <sub>0</sub>	A	B	C	D
0	0	0	0	0	E
0	1	0	0	E	0
1	0	0	E	0	0
1	1	E	0	0	0

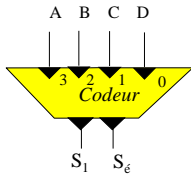


Table de vérité

A	B	C	D	S <sub>1</sub>	S <sub>0</sub>
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

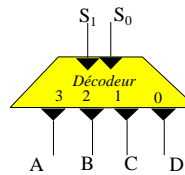
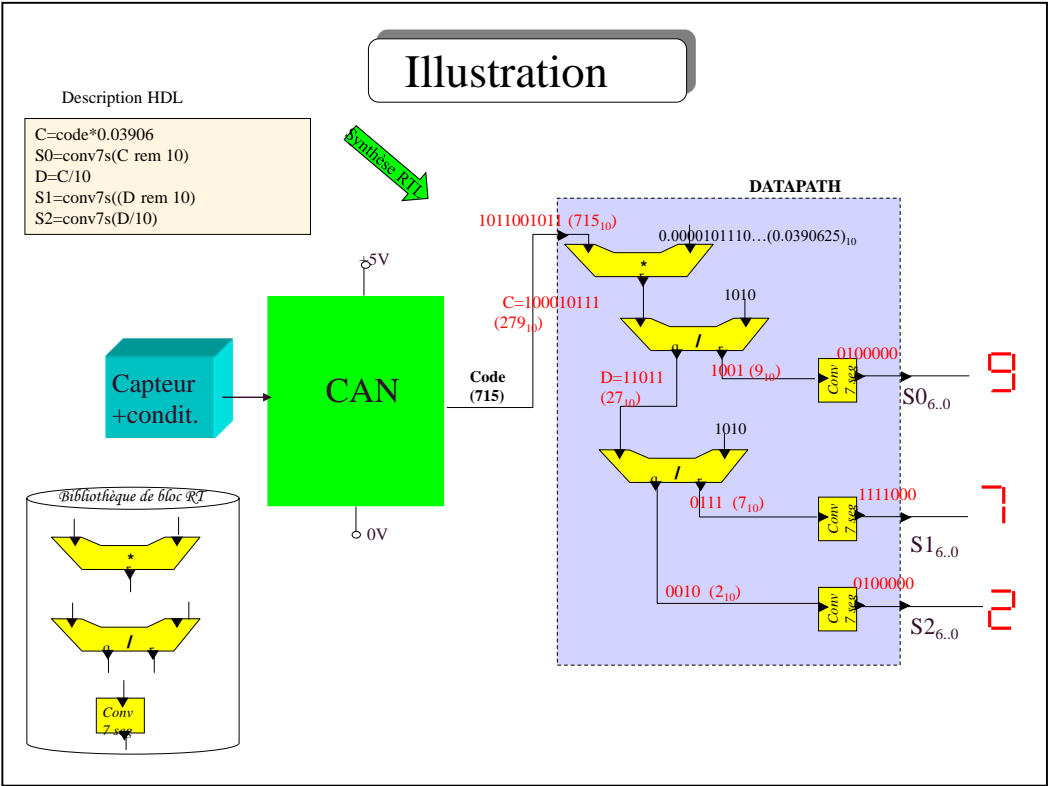
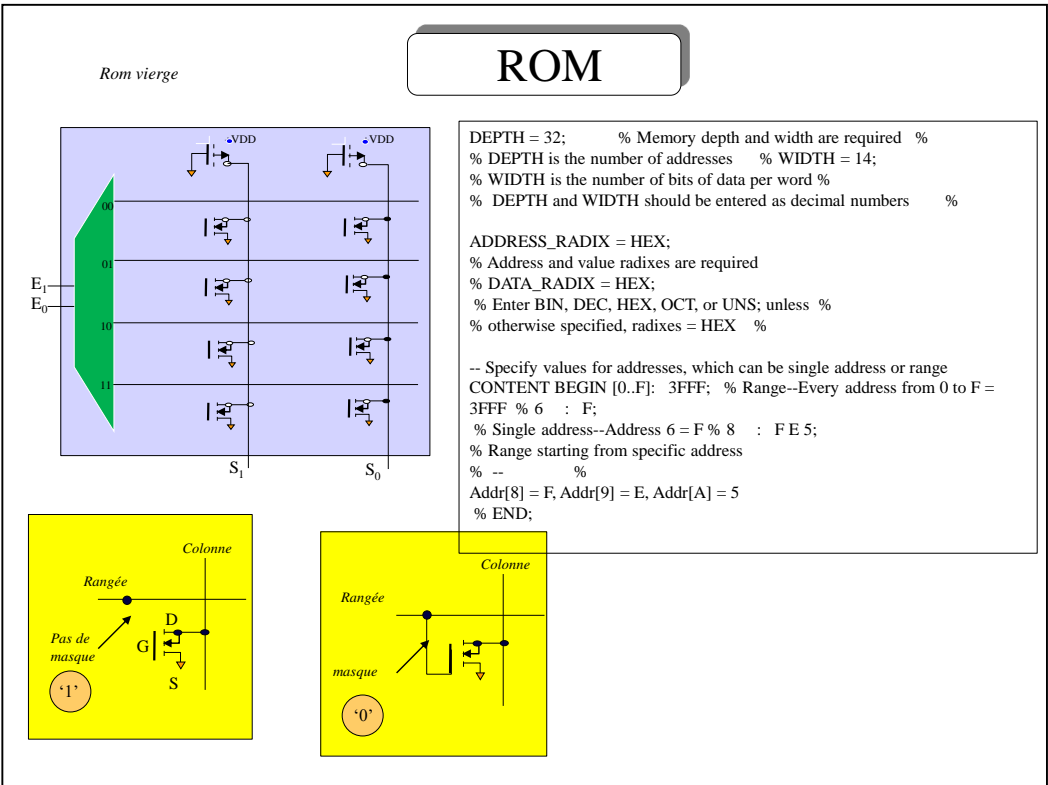


Table de vérité

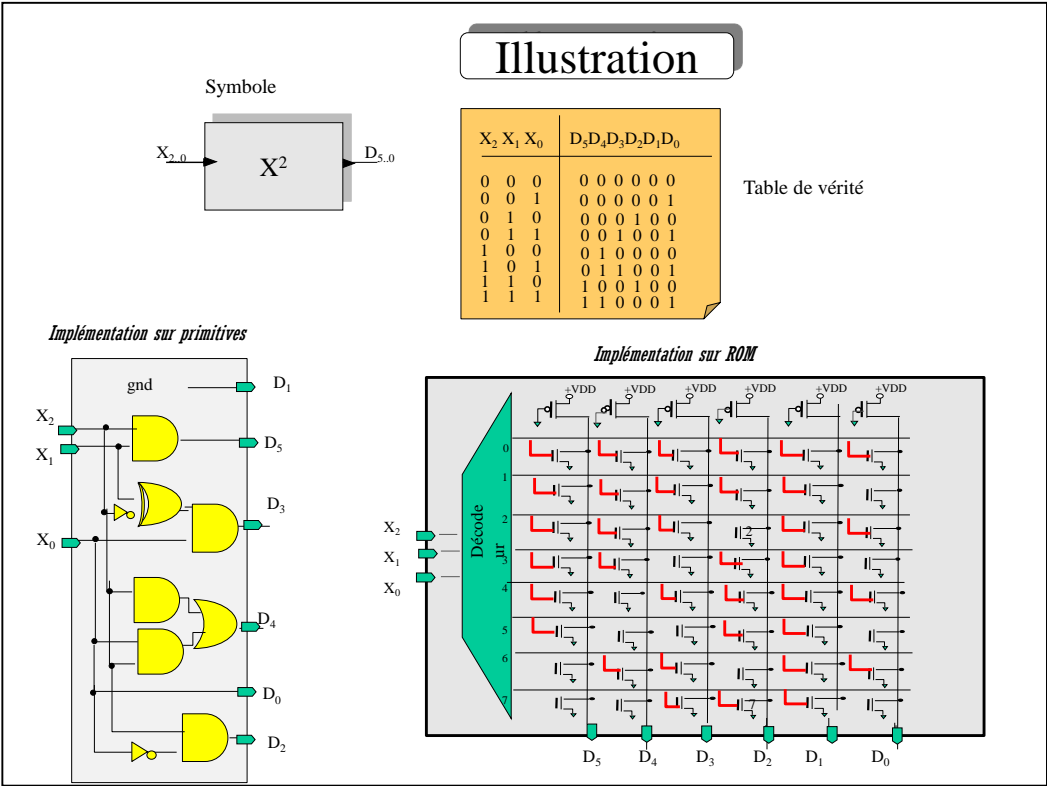
S <sub>1</sub>	S <sub>0</sub>	A	B	C	D
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Ces blocs combinatoires permettent d'implémenter des structures de contrôle (if..., switch,...)

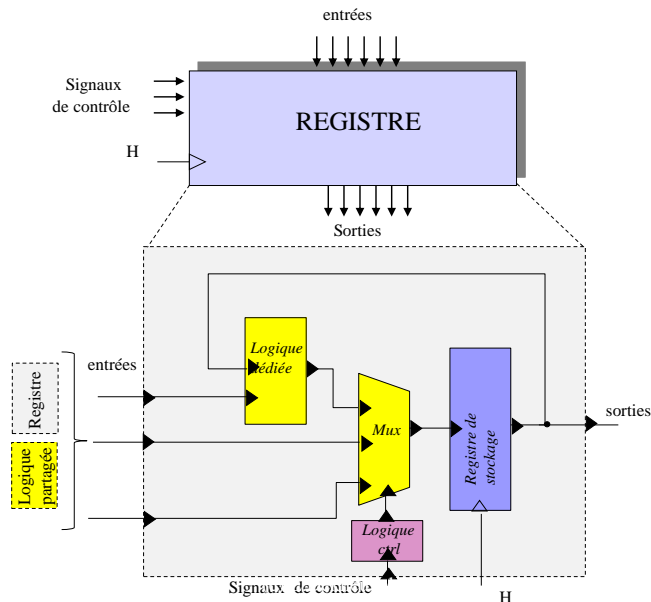




Une ROM est structure complexe associant un décodeur à des rangées de transistors qui réalisent un codeur spécifique. Si une rangée est à 1 en sortie du décodeur, la présence d'un masque entre la rangée et la grille positionne le drain à 0.



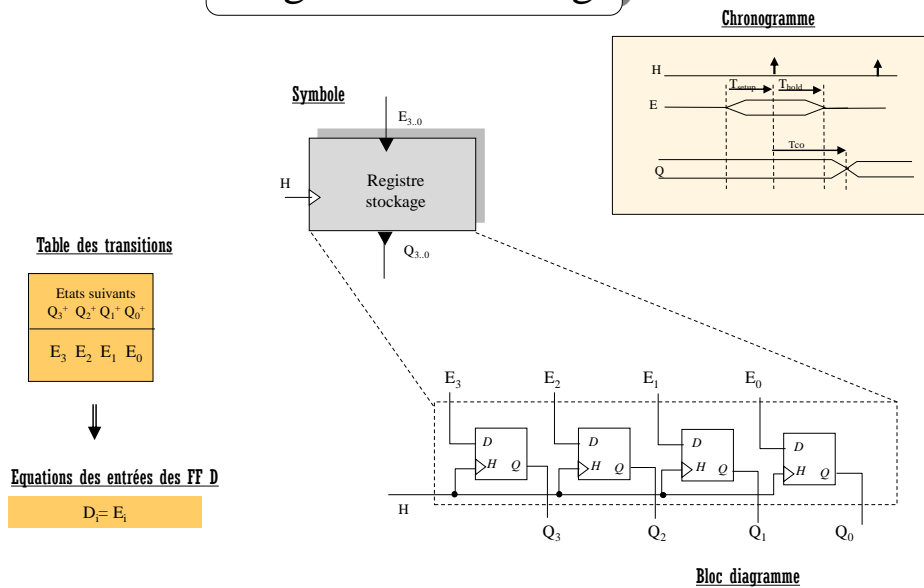
## Modèle général d'un registre



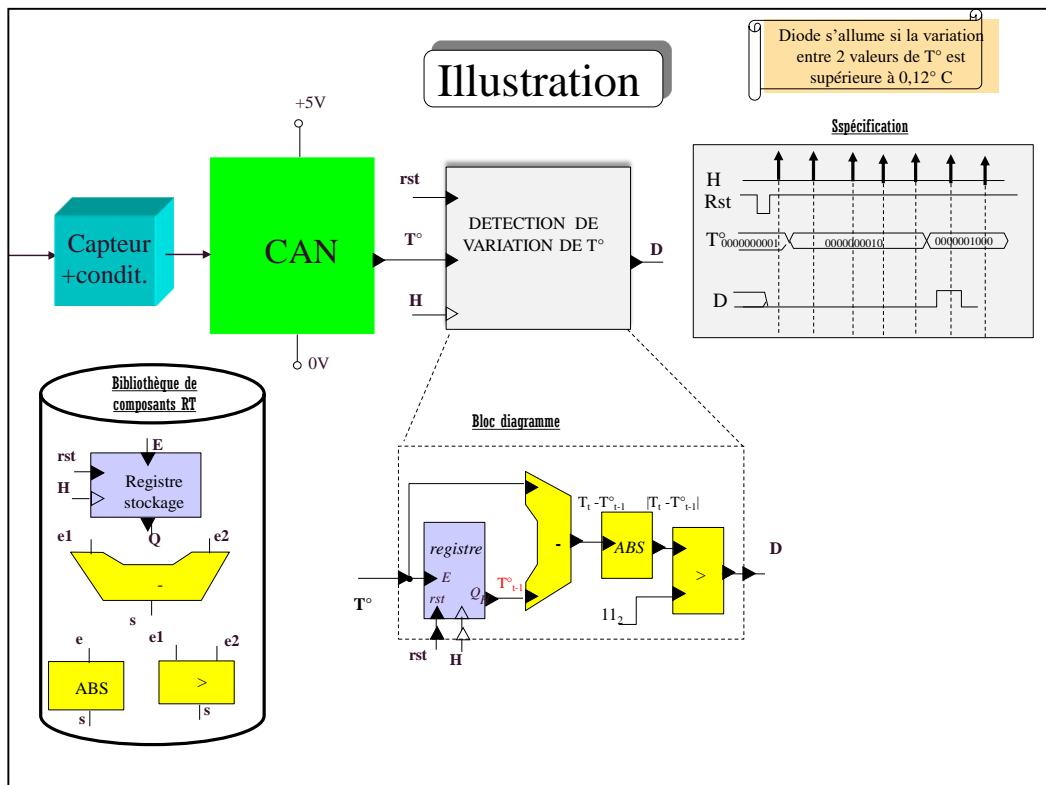
D'une manière générale, tout bloc séquentiel implémentant une ou plusieurs opérations sur une variable est qualifié de registre. Lorsque la logique réalisant les micro-opérations est dédiée au registre, on considère qu'elle fait partie intégrante du registre (ce qui permet des optimisations). Les données sources peuvent provenir d'une logique partagée avec d'autres registres ou directement d'un ou plusieurs autres registres.

On notera qu'en général les entrées de contrôle sont séparées pour chaque opération.

# Registre de stockage



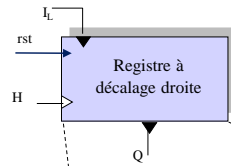
Dans sa forme élémentaire, réalise une opération de transfert inconditionnel d'une donnée d'entrée E dans Q. Une table d'opération (fonctionnelle) ou une description en langage RTL permet de décrire la valeur affectée au registre, une fois l'horloge appliquée. Cette valeur peut être une valeur constante (exprimée en binaire, hexa, décimal), une variable (E ici), ou le résultat d'une opération. Un chronogramme peut permettre de préciser des propriétés temporelles.



Le registre de stockage couplé avec quelques composants combinatoires permet d'implémenter des traitements plus ou moins complexes tel qu'une variation de  $T^\circ$  ( $1^\circ$  environ ici). Seule la température à l'instant d'horloge précédent es nécessaire pour effectuer le traitement requis.

## Registre à décalage

Symbole



Chronogramme

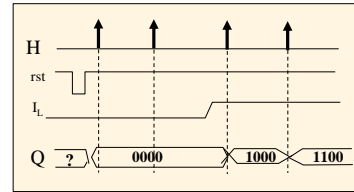


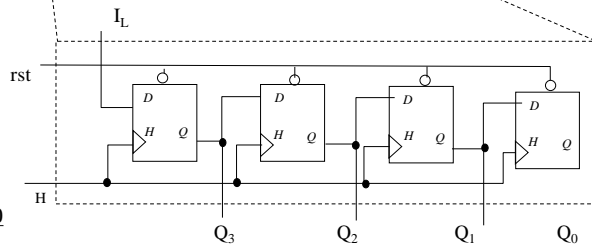
Table des transitions

Etats suivants
$Q_3^+ Q_2^+ Q_1^+ Q_0^+$
$I_L Q_3 Q_2 Q_1$

Equations des entrées des FF D

$$D_3 = I_L$$

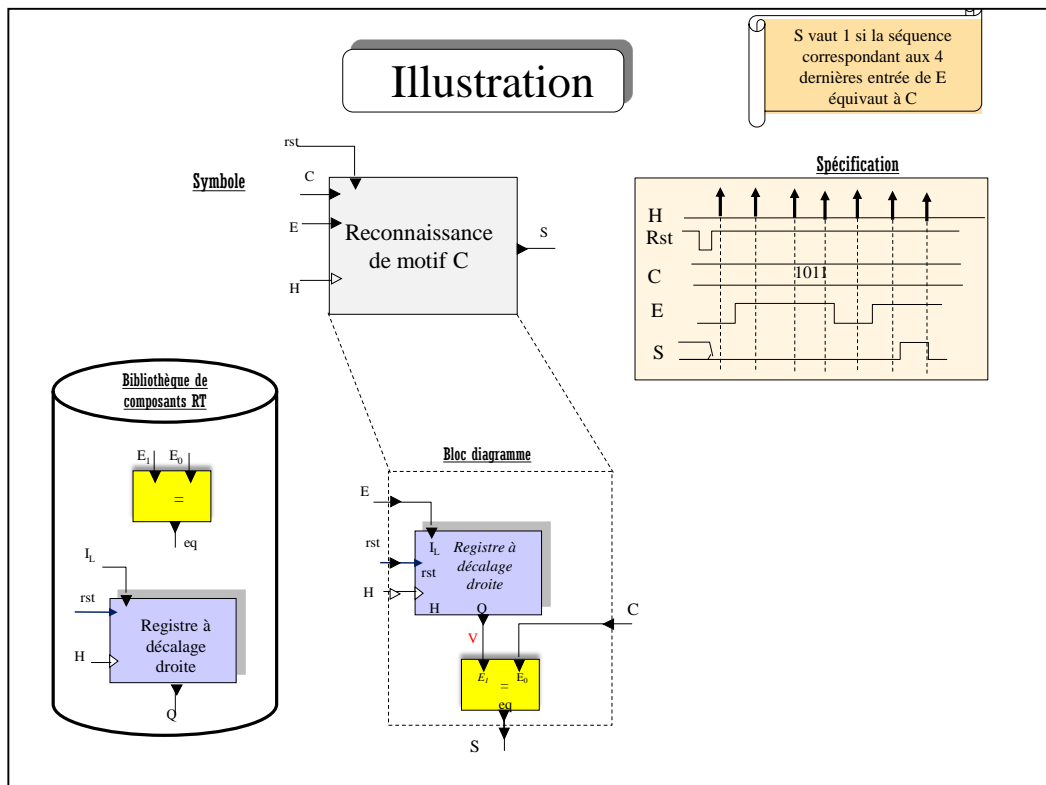
$$D_i = Q_{i+1} \text{ pour } i=2..0$$



Bloc diagramme

Les registres intègrent souvent une logique de transformation des données stockées. décaler la valeur courante d'un registre d'une ou plusieurs positions vers la droite ou vers la gauche est également une opération classique de traitement de données.





On a vu comment détecter une séquence d'entrée à partir d'une machine à états finis. Il est également possible d'obtenir un résultat similaire en associant un registre à décalage droit et un comparateur d'égalité. Le registre à décalage permet de mémoriser l'information sur les 4 derniers fronts d'horloge.

# Registre avec enable

Symbole

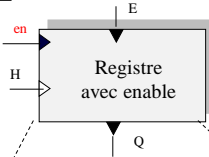
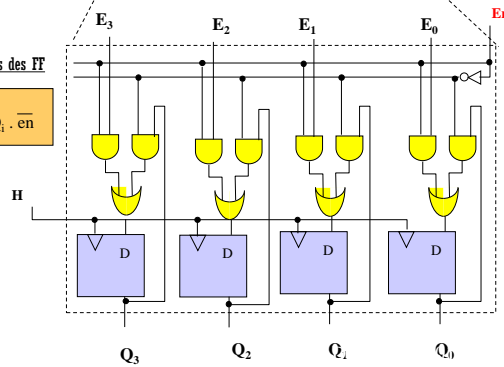


Table des transitions

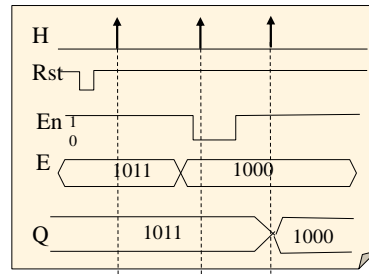
en	$Q_3^+ Q_2^+ Q_1^+ Q_0^+$
0	$Q_3 Q_2 Q_1 Q_0$
1	$E_3 E_2 E_1 E_0$

Equations des entrées des FF

$$D_i = E_i \cdot en + Q_i \cdot \overline{en}$$

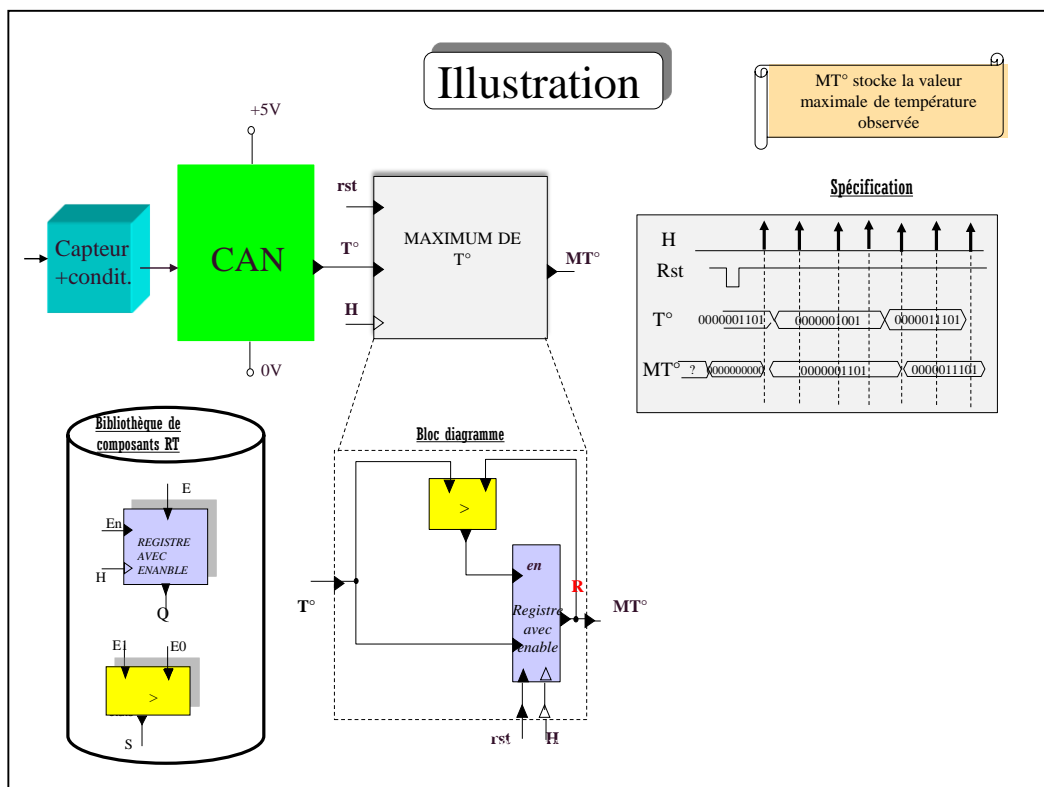


Spécification



Bloc diagramme

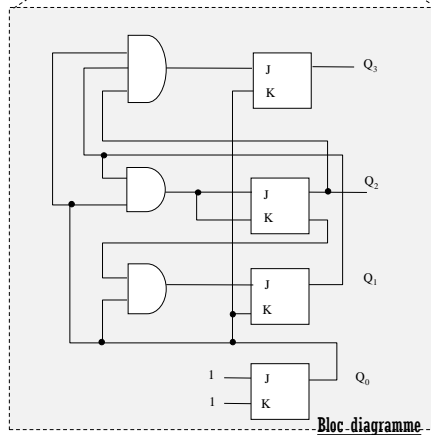
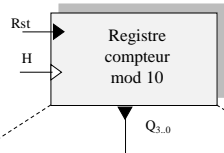
un registre peut ne pas avoir à effectuer une opération de chargement, de décalage ou d'incrément/décroissement à chaque cycle d'horloge. Charger ou conserver la valeur en mémoire peut donc être conditionnée à la valeur du signal *en*.



Dans cette FSMD, on recherche la valeur maximale de température observée à partir de l'instant d'application du *rst*. Il est à nouveau nécessaire d'introduire un registre pour mémoriser la valeur maximale observée jusqu'à l'instant courant. Vu qu'on ne mémorise plus systématiquement les valeurs de  $T^\circ$ , un registre avec enable est utilisé.

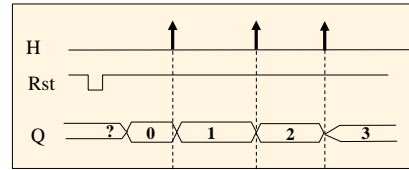
# Registre compteur

**Symbole**



**Bloc-diagramme**

**Spécification**



**Table des états**

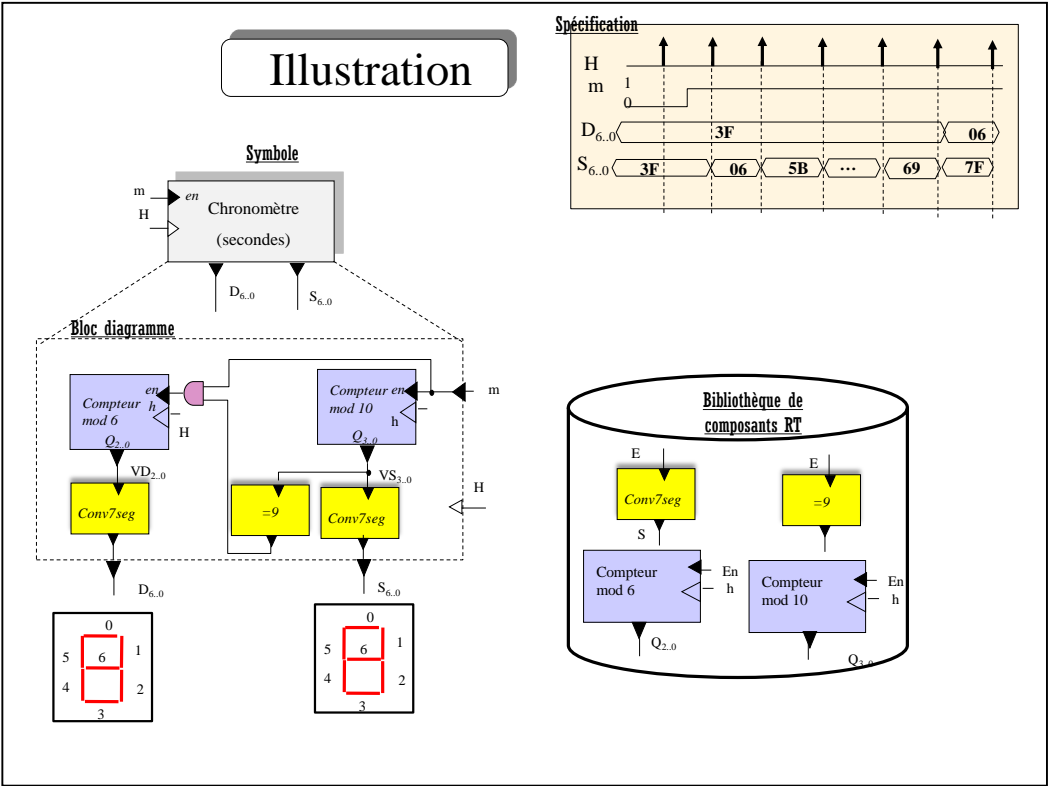
Y	Y <sup>+</sup>	Z,Z,Z,Z,Z <sub>0</sub>
E0	E1	0000
E1	E2	0001
E2	E3	0010
E3	E4	0011
E4	E5	0100
E5	E6	0101
E6	E7	0110
E7	E8	0111
E8	E9	1000
E9	E0	1001

**Equations des FF**

$$J_3 = Y_2 Y_1 Y_0 \quad J_2 = Y_1 Y_0 \quad J_1 = \overline{Y_3} Y_0$$

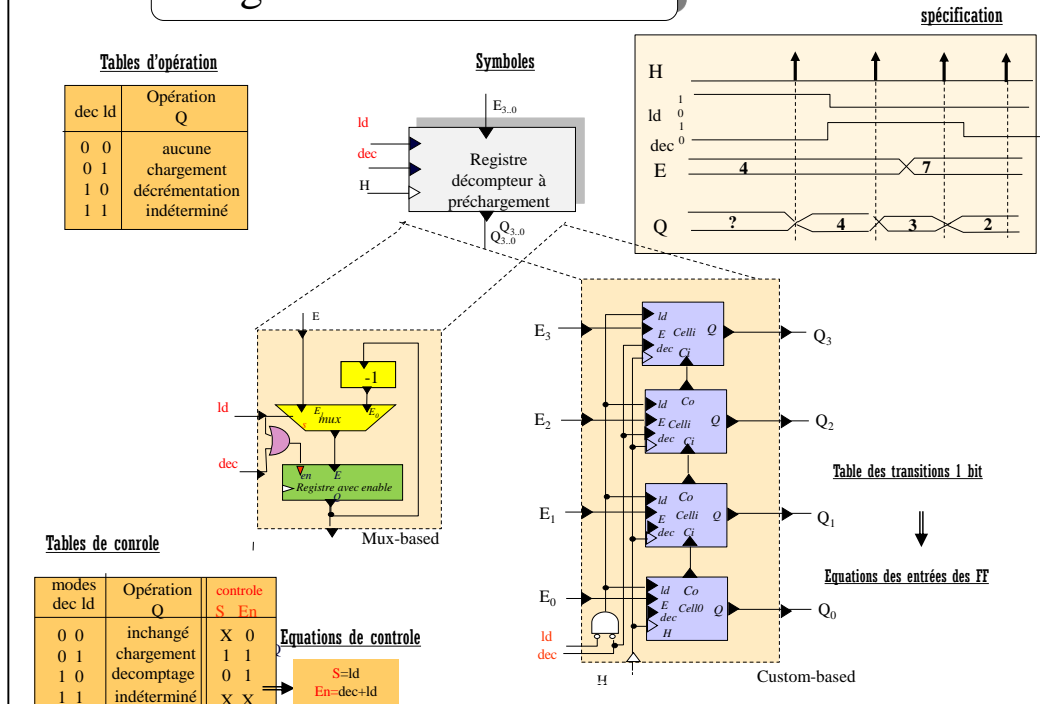
$$K_3 = Y_0 \quad K_2 = Y_1 Y_0 \quad K_1 = Y_0$$

Les registres peuvent aussi intégrer une logique de transformation des données stockées. Le registre est alors à la fois source et destinataire de l'opération. Ici, il s'agit d'un registre compteur.

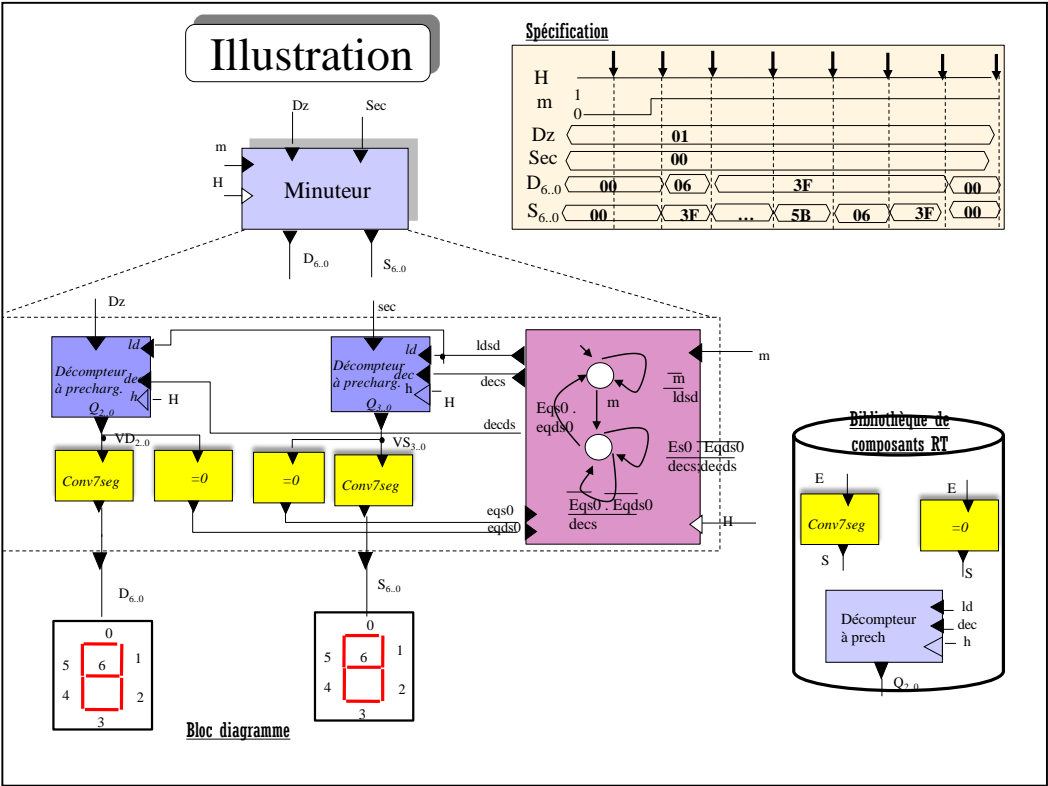


Un chronomètre des secondes peut être conçu aisément à partir de deux compteurs mod 10 et mod 6 connectés par un signal interne de retenue et dont les sorties sont converties en une représentation 7 segments.

# Registres multi-fonctions



Les registres peuvent implémenter plusieurs opérations alternatives pilotées par des entrées de contrôle. Ici le décompteur à préchargement peut réaliser de manière exclusive), une opération de chargement, une opération de décomptage ou ne rien faire. Les registres complexes peuvent implémenter plusieurs opérations alternatives pilotées par des entrées de contrôle. Ici le décompteur à préchargement peut réaliser de manière exclusive), une opération de chargement, une opération de décomptage ou ne rien faire. Comme pour les blocs combinatoires, les blocs séquentiels sont souvent réalisables de plusieurs façons: en top down à partir de blocs plus élémentaires ou en bottom à partir des primitives logiques (logique aléatoires) ou de cellules (logique structurée).



Un chronomètre des secondes peut être conçu aisément à partir de deux compteurs mod 10 et mod 6 connectés par un signal interne de retenue et dont les sorties sont converties en une représentation 7 segments.

# FSM avec chemin de données (FSMD)

- A. Blocs combinatoires et séquentiels*
- B. Description transfert de registres(HDL)***
- C. Conception des FSMD*
- D. Registres complexes*
- E. Le microprocesseur*

(c) E.Deknevel 2018

Un Langage de description de matériel (HDL) doit permettre de définir le fonctionnement d'un circuit en termes de micro-opérations (opérations réalisées sur cycles d'horloge). On parle de langage de description de matériel (par opposition à un langage de programmation).



# Les variables

## Identificateurs de registre et mémoires

*Une lettre suivie d'un ou deux chiffres ou lettres*

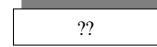
Désignation d'un bit ou d'une tranche de bits,  
d'un élément d'un registre ou collection registres

*Identificateur( $n^{\circ}$ bit),*

*identificateur( $n^{\circ}$ bit ..  $n^{\circ}$ bit)*

*Identificateur(adr)*

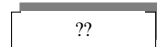
R1



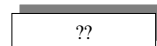
R2



R3



R4

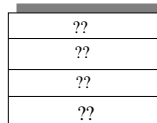


M[0]

M[1]

M[2]

M[3]



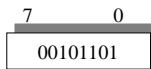
Les variables sont référencées dans la notation RTL par un identificateur d'une lettre suivie d'un ou deux chiffres ou lettres (1D incorrect par exemple). Par défaut, l'identificateur désigne l'ensemble des bits d'une variables. Dans le cas contraire, un sous-ensemble (un simple bit ou une tranche désignés par des constantes) peut être précisé entre parenthèse. La variable est alors considérée comme un vecteur de bits. Un tableau peut permettre de regrouper une collection de variables dont le  $n^{\circ}$  (l'adresse) est précisé entre crochets ou parenthèses. L'adresse peut alors être fixée à l'aide d'une autre variable.


## Micro-opérations de chargement

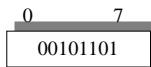
**Transfert de registre ou mémoire:**  
*destinataire  $\leftarrow$  source*

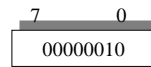
**Transfert avec concaténation:**  
*Destinataire  $\leftarrow \{source1, source2\}$   
 ou  $source1 \parallel source2$*

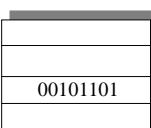
$R1 \leftarrow 00101101$   
 $R2 \leftarrow R1(0)$   
 $R3 \leftarrow R1$   
 $R4 \leftarrow 0000 \parallel R1(7..4)$   
 $M[R4] \leftarrow R3$

R1 

R2 

R3 

R4 

M[0]   
 M[1]  
 M[2]  
 M[3]

Un transfert vers une variable peut s'effectuer à partir d'une donnée constante ou à partir d'une stockée dans une variable. La source n'est pas modifiée durant le transfert. Dans le cas d'un transfert partiel d'une variable registre, il est utile de préciser l'emplacement des poids forts et des poids faibles avec deux possibilités:

1. le poids fort est à droite :  $R1(0)=0$
2. le poids faible est à gauche:  $R1(0)=1$

L'opérateur de concaténation permet de simplifier l'écriture en combinant dans l'exemple les 2 opérations  $R4(7..4) \leftarrow 0000$  et  $R4(3..0) \leftarrow R1(7..4)$  en une seule affectation.

## Micro-opérations logiques

**ET logique bit à bit**

*AND*

**OU logique bit à bit**

*OR*

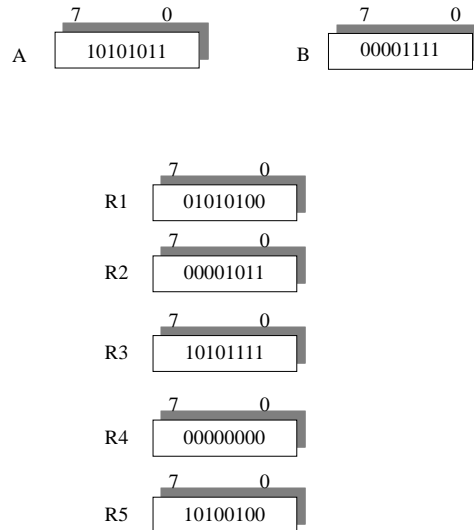
**Ou exclusif bit à bit**

*XOR*

**Complément à un**

*NOT*

$R1 \leftarrow \text{not } A$   
 $R2 \leftarrow A \text{ and } B$   
 $R3 \leftarrow A \text{ or } B$   
 $R4 \leftarrow A \text{ xor } A$   
 $R5 \leftarrow A \text{ xor } B$



Ces micro opérations considèrent chaque bit du registre de manière isolée et traite cet élément comme une variable binaire. Remarque: dans le cas du AND, B peut être considéré comme un masque forçant les 4 bits de poids fort de R2 à 0. Dans le cas du OR, B peut être considéré comme un masque forçant les 4 bits de poids faible de R3 à 1

Un XOR avec deux registres identiques équivaut à un RAZ du registre destinataire. Sinon,  $A \text{ xor } B$  complémente les bits de poids faible de R5

## Micro-opérations arithmétiques

### Addition arithmétique

+

### Soustraction

-

### Incrémentation

+1

### Décrémentation

-1

### Négation (complément à 2)

Not X+1

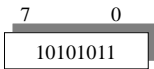
$R1 \leftarrow B-1$

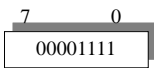
$R2 \leftarrow B+1$

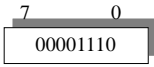
$R3 \leftarrow A+B$

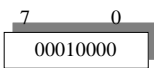
$R4 \leftarrow \text{not } B+1$

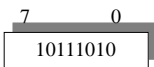
$R5 \leftarrow A+R4$

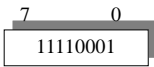
A  -85

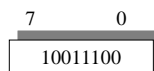
B  15

R1 

R2 

R3 

R4  -15

R5  -100

**Micro-opérations arithmétiques:** opérations utilisées dans de nombreuses conceptions de circuits numériques (filtres, asservissement)

Remarque: si des implémentations combinatoires de composants multiplication et division (plutôt que comme une séquence de microopérations), on peut utiliser des opérateurs arithmétiques \* et /.

On notera pour la dernière microopération que la retenue éventuelle est négligée

## Micro-opérations décalage

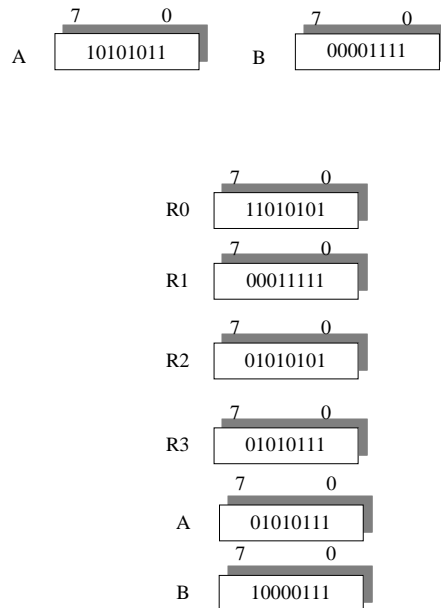
**Décalage à droite:**

*sr*

**Décalage à gauche:**

*sl*

$R0 \leftarrow sr A$   
 $R1 \leftarrow sl B$   
 $R2 \leftarrow 0 \parallel A(7..1)$   
 $R3 \leftarrow A(6..0) \parallel 1$   
 $A \leftarrow A(6..0) \parallel A(7)$   
 $B \leftarrow B(0) \parallel B(7..1)$



Les micro-opérations de décalage très utiles pour exprimer des transfert de données en série. Ces opérations de décalage peuvent également avantageusement remplacer des multiplications par multiples de 2 (dec gauche) ou des / par multiples de 2 (dec droite). Les deux dernières microop sont des décalages circulaires gauches (A) et droite (B).

## Opérateurs relationnels

=	égal à
<	Strictement inférieur
>	Strictement supérieur
≠	différent de
≥	supérieur ou égal
≤	inférieur ou égal

Composition d'opérateurs: et ou non ( )

$Z \leftarrow (I=2)$   
 $N \leftarrow (I \leq 0)$

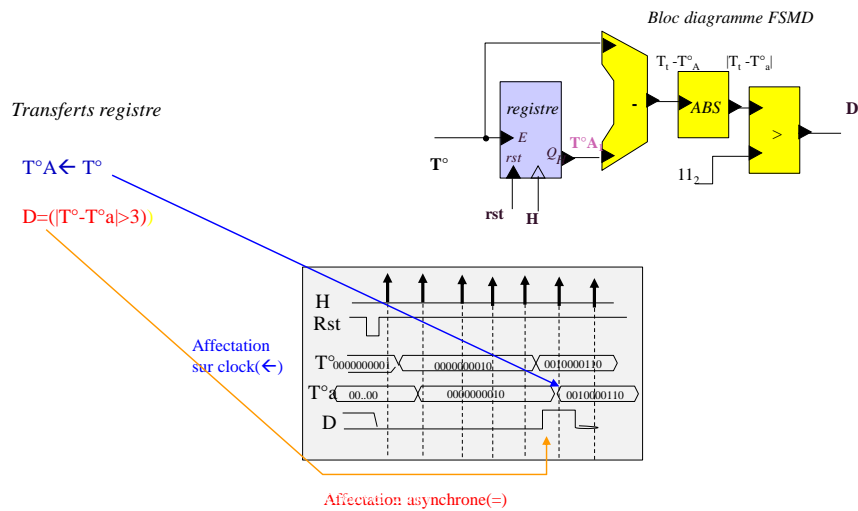
I      7      0  
10101011

Z      0

N      1

**Opérateurs relationnels:** permettent la description de conditions plus ou moins complexes. Comme dans les langages de programmation conventionnel, les opérateurs peuvent être combinés au moyen d'opérateurs ET,OR,NON. On peut également affecter le résultat d'un test à un registre et placer ce registre dans la boîte conditionnelle.

## Transfert synchrone vs asynchrone



Le transfert dans une variable est considéré comme immédiat (combinatoire) lorsqu'il est spécifié par le symbole  $=$ . Le transfert s'effectue au front d'horloge lorsqu'on utilise une flèche.

## Transferts conditionnels

cond:  $\text{Reg}_x \leftarrow f(\text{Reg}_1, \text{Reg}_2, \dots, \text{Reg}_n)$

Transferts simple

chargement

arithmétiques

logiques

décalage

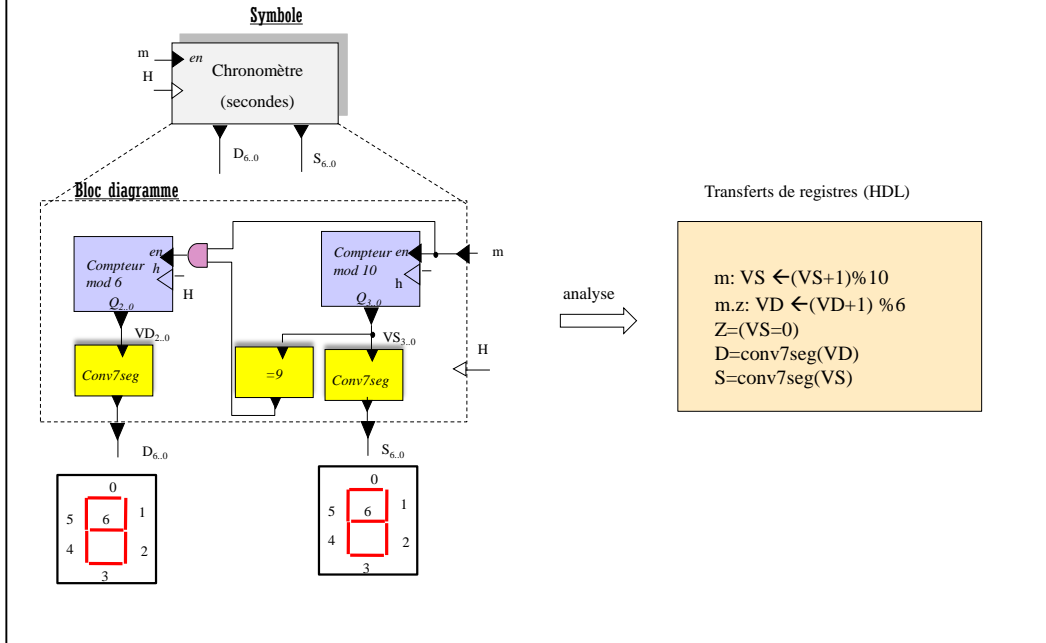
cond:  $R1 \leftarrow A+B \ // \ R2 \leftarrow C-D$

Transferts simultanés

Les transferts peuvent être conditionnels et être assujettis à des conditions sur des variables booléennes. Ces conditions sont exprimées avant ces transferts.



## Illustration



Un chronomètre des secondes peut être conçu aisément à partir de deux compteurs mod 10 et mod 6 connectés par un signal interne de retenue et dont les sorties sont converties en une représentation 7 segments.