UE Electrical Engineering
2025

UE Electrical Engineering : électronique numérique

Polycopié de cours 2/2
Logique séquentielle

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Sommaire

Chapitre 5 : Fonctions de base de la logique séquentielle

1. Introduction

Avec les circuits séquentiels, nous abordons un type nouveau de comportement par rapport à celui des circuits combinatoires, dans lequel la dimension temporelle joue un rôle fondamental.

Pour le mettre en évidence, étudions un exemple simple de circuit séquentiel. Le circuit considéré dispose d'une entrée E(t) et d'une sortie Y(t), t étant la variable temps. Sa fonction consiste à reproduire sur sa sortie la seconde impulsion dans un train de deux impulsions consécutives présenté sur son entrée. Ce comportement est illustré par le chronogramme de la figure 5.1. On néglige, pour simplifier, le temps de propagation du circuit.

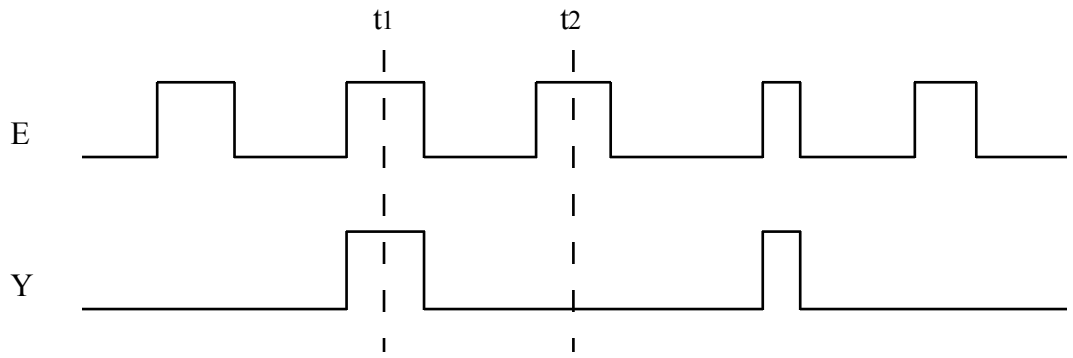


figure 5.1 : exemple de comportement d'un circuit séquentiel

On constate que, bien que $\mathrm{E}(t_1) = \mathrm{E}(t_2)$, $\mathrm{Y}(t_1) \neq \mathrm{Y}(t_2)$. Un tel comportement ne peut être engendré par un circuit combinatoire, dont, par définition, l'état de ses sorties est lié exclusivement à celui de ses entrées.

2. Modélisation d'un système séquentiel

2.1 Modèle comportemental : automate à états finis (AEF)

L'analyse du comportement présenté dans l'exemple introductif conduit à conclure qu'un circuit séquentiel dispose d'un **état interne** qui évolue en fonction des impulsions présentées sur son entrée. L'évolution de l'état interne d'un circuit est liée aux commutations des entrées. Le terme **événement** désigne une commutation d'une entrée dans la mesure où cette commutation constitue la cause de l'évolution de l'état interne du circuit séquentiel. Une **séquence d'événements** appliquée à l'entrée d'un circuit séquentiel conduit celui-ci à évoluer au sein d'un **espace fini d'états**. Ainsi, à tout moment, l'état interne du circuit reflète un historique des événements qui se sont présentés sur ses entrées. C'est ce mécanisme qui permet à un circuit séquentiel de produire des états de sortie différents en réponse à un même événement dont les occurrences se situent dans des contextes historiques

différents. Ce modèle théorique est appelé **automate à états finis** (AEF). Ce modèle est couramment utilisé dans les domaines de l'électronique numérique et de l'informatique.

Le comportement d'un AEF peut être défini par un **graphe d'états**. Ce graphe spécifie les événements régissant les **transitions** entre les différents états internes de l'automate. Chaque transition est matérialisée par un arc étiqueté par deux types d'attributs binaires : d'une part, les valeurs des entrées de l'automate associées à la transition, et d'autre part, les valeurs des sorties associées à cette même transition. Le graphe de la figure 5.2 présente une solution au problème posé par l'exemple introductif. Nous laissons le soin au lecteur de le vérifier. Pour l'y aider, signalons que l'état *B* correspond à la situation à l'instant t_2 et que l'état *D* correspond à l'instant t_1 (cf. figure 5.1).

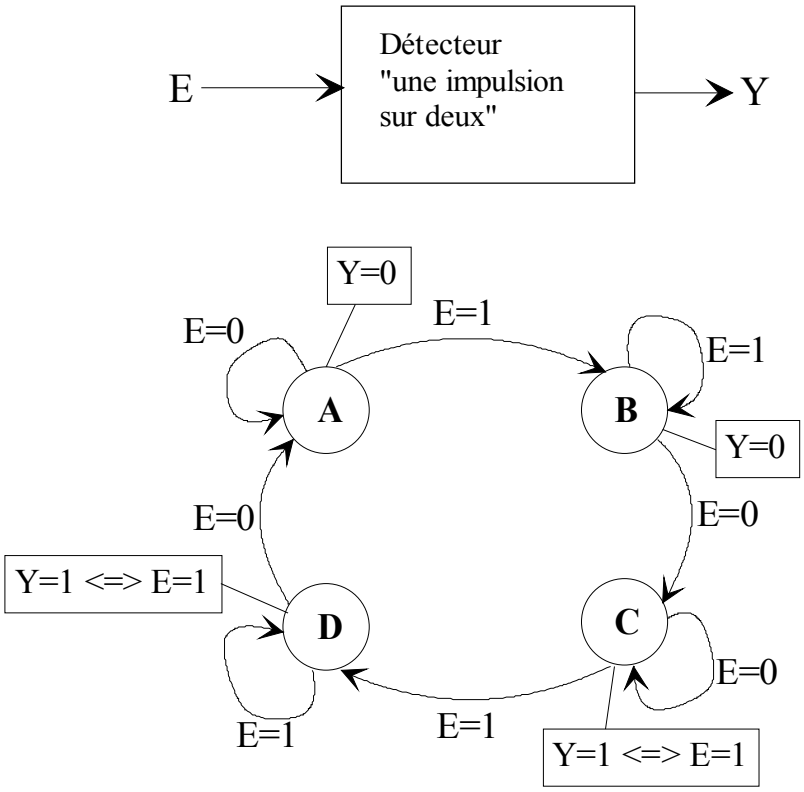


figure 5.2 : exemple de graphe d'états

D'une façon générale, dans un graphe d'états associé à un circuit séquentiel possédant *m* entrées, il y a 2^m arcs issus de chaque état, ou nœud, du graphe. Chaque arc correspond à une combinaison donnée des entrées. Si, pour un état de départ donné, seules *p* entrées parmi *m* interviennent dans les transitions issues de cet état, le nombre d'arcs représentés peut être réduit à 2^p. Dans l'étiquetage, les $m-p$ autres entrées sont marquées par un « X » à la place d'une valeur binaire. Notons enfin que le nombre d'états d'un AEF n'est pas lié au nombre de ses entrées, mais à la complexité de son comportement, autrement dit, à la longueur des séquences d'événements d'entrée qu'il reconnaît.

2.2 Modèle structurel : machine de Mealy

La réalisation d'une fonction séquentielle quelconque repose sur une structure appelée **machine de Mealy**, proposée par le mathématicien du même nom. Cette structure permet de matérialiser le comportement d'un AEF. La suite de ce paragraphe décrit la démarche permettant de passer d'une

description comportementale d'un système séquentiel (modèle d'AEF) à sa description structurelle sous forme de machine de Mealy.

L'idée de base consiste à **coder**, c'est-à-dire à matérialiser, les états de l'AEF par un vecteur, ou *n*-uplet, de variables booléennes $\mathbf{X} = (X_0, \cdots, X_{n-1})$. Les variables X_i, $i = 0 \cdots n-1$, sont appelées **variables internes**. Chaque état est codé par une et une seule combinaison de ces variables. Pour coder un AEF à *N* états il faut donc au moins *n* variables internes, si *n* est le plus petit entier vérifiant $N \leq 2^n$.

La tâche suivante consiste à caractériser les différentes transitions entre les états et à calculer les sorties de l'automate pour chaque transition. Le graphe de la figure 5.2 montre que le franchissement d'une transition inter-état (i. e. d'un arc) est conditionné par deux entités :

- d'une part, son **état courant** (ou **état présent**) **X**, représenté par une combinaison des variables internes X_i,
- d'autre part, une combinaison des variables d'entrée de l'automate.

Le nombre d'états étant fini, le problème du calcul de la condition de franchissement est dénombrable.

L'état d'arrivée d'une transition, encore appelé **état suivant** ou **état futur**, est également fonction des variables d'entrée et des variables internes. Soit G cette fonction. Si, pour une transition donnée, **X** est le vecteur représentant l'état courant, $\mathbf{X}^+$ le vecteur représentant l'état futur, et **E** le vecteur constitué des variables d'entrée de l'automate permettant le franchissement de la transition, G est une fonction combinatoire de **X** et **E** :

$$\boxed{\mathbf{X}^+ = \mathrm{G}(\mathbf{X}, \mathbf{E})} \qquad (1)$$

La fonction G est dite **fonction « état suivant »** ou **fonction d'excitation secondaire**. Cette fonction étant combinatoire, on sait la réaliser, quelle que soit sa complexité.

La lecture du graphe d'états montre que le vecteur **Y** des variables de sortie se calcule également en fonction des variables d'entrée et de l'état courant de l'automate. La fonction F permettant d'obtenir **Y** est donc également une fonction combinatoire de **X** et **E** :

$$\boxed{\mathbf{Y} = \mathrm{F}(\mathbf{X}, \mathbf{E})} \qquad (2)$$

La fonction F est dite **fonction de sortie**.

Afin de réaliser les fonctions F et G, il est nécessaire de pouvoir accéder à tout instant à l'état interne courant de la machine. Il faut, pour cela, être capable de **stocker** les variables internes. On les place, à cette fin, dans une boîte noire. Cette boîte, qui présente une entrée et une sortie par variable, a pour unique rôle de ralentir l'évolution de l'état de la machine, pour la rendre observable, en introduisant un délai temporel Δt entre sortie et entrée. Des réalisations possibles de cette boîte seront étudiées plus loin dans ce chapitre (cf. section 3). En appelant $\mathbf{X_{out}}$ le vecteur de sortie de la boîte et $\mathbf{X_{in}}$ son vecteur d'entrée, on peut caractériser cette boîte noire par :

$$\boxed{\mathbf{X_{out}}(t) = \mathbf{X_{in}}(t - \Delta t)} \qquad (3)$$

On interconnecte ensuite la boîte noire caractérisée par l'équation (2) avec les fonctions F et G de telle façon que $\mathbf{X} = \mathbf{X_{out}}(t)$ et $\mathbf{X}^+ = \mathbf{X_{in}}(t - \Delta t)$. La structure de la machine de Mealy est représentée en figure 5.3.

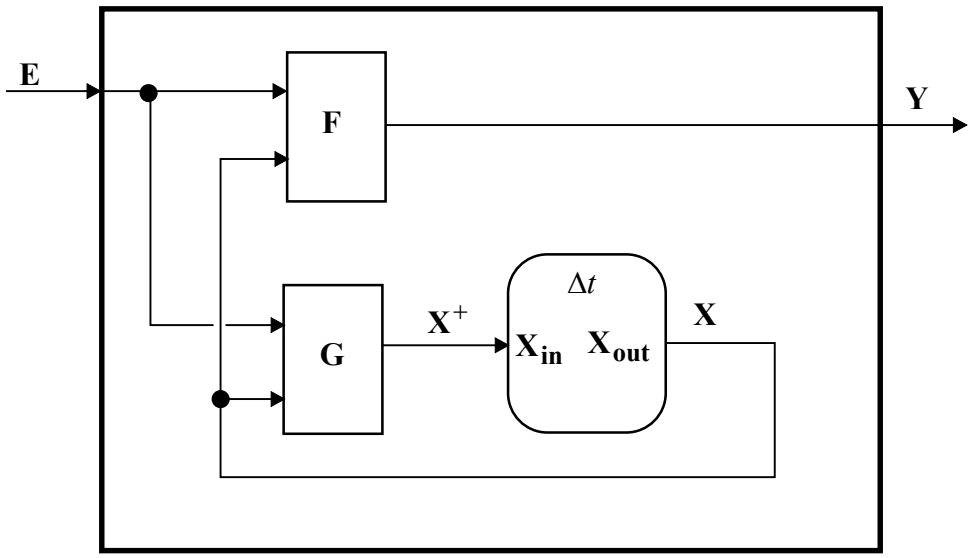


figure 5.3 : structure générale de la machine de Mealy

En résumé, le caractère séquentiel de la machine de Mealy provient de la rétroaction des sorties vers les entrées de la fonction G.

2.3 Un exemple de mise en œuvre d'un AEF par une machine de Mealy

On vient de démontrer la capacité d'une machine de Mealy à matérialiser le comportement d'un AEF. Il reste à montrer comment réaliser la machine de Mealy qui synthétise un AEF spécifié par un graphe d'états. On touche ici à un problème de méthodologie de conception des circuits et systèmes séquentiels, qui sera traité au chapitre 6. Néanmoins, à l'attention des lecteurs impatients, la figure 5.4 présente une réalisation possible des fonctions F et G à l'aide de portes logiques.

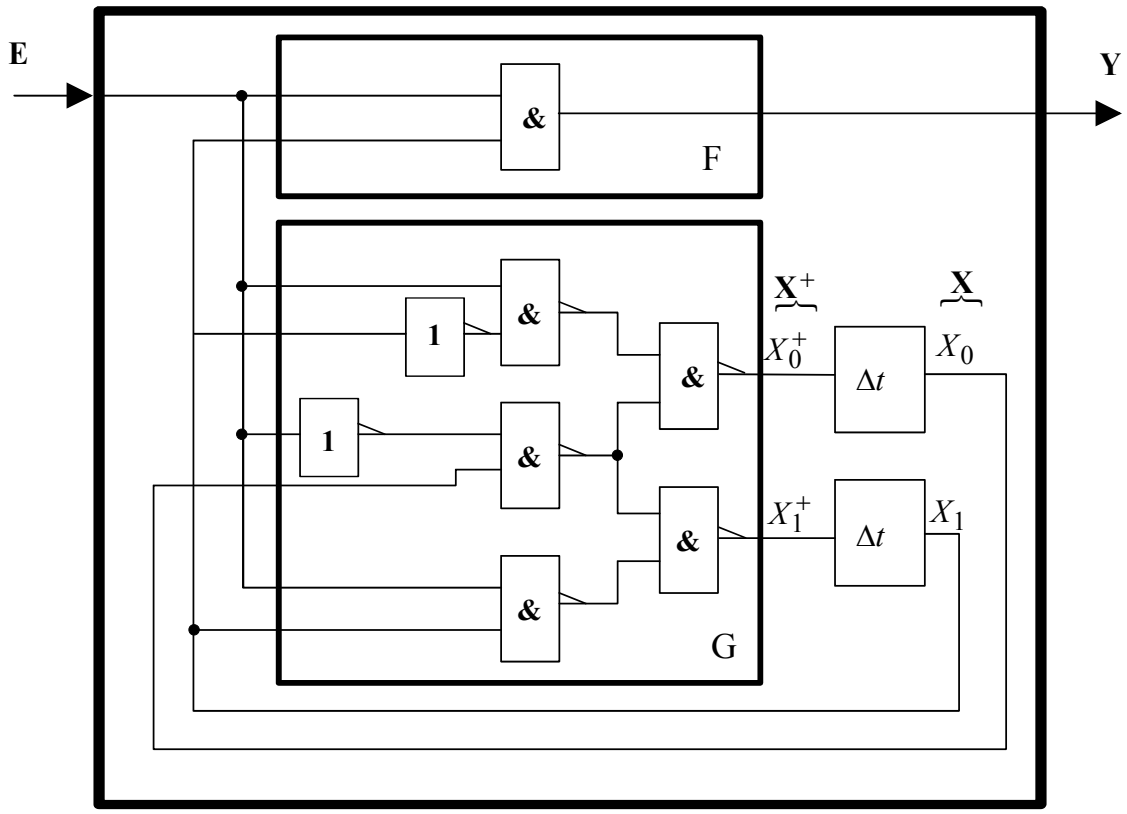


figure 5.4 : une réalisation du détecteur « une impulsion sur deux »

Ce logigramme montre que deux variables internes X_0 et X_1 ont été créées pour coder les quatre états de l'automate. Le codage choisi dans cette réalisation est donné par le tableau 5.1.

Etat	A	B	C	D
X_0X_1	00	01	11	10

tableau 5.1 : codage des états du détecteur « une impulsion sur deux »

La réalisation de la fonction de retard Δt n'est pas précisée sur la figure 5.4. En pratique, deux solutions sont envisageables :

- La première consiste à ne pas introduire de retard Δt et de prélever directement l'état courant en sortie de G. Dans le modèle de la machine de Mealy, le retard Δt est introduit pour permettre l'observation des variables internes de l'automate. En l'absence de ce retard, l'évolution de la machine se ferait à vitesse infinie, ce qui est concrètement inconcevable. En pratique, les portes utilisées pour la réalisation de la fonction G sont caractérisées par un temps de propagation (cf. chapitre 3). Ainsi, le rôle du retard Δt est joué par le temps de propagation global du circuit réalisant la fonction G. Cette solution est cependant rarement adoptée en pratique, sauf pour des applications très spécifiques, car les temps de propagation sur les différents chemins du circuit réalisant G ne sont *a priori* pas égaux. Ainsi, les variables internes X_0 et X_1 ne commutent pas de façon strictement simultanée lors des transitions entre états. On peut, par conséquent, observer des états

parasites fugitifs susceptibles d'entraîner un comportement imprédictible de l'automate. On parle alors d'**aléa de fonctionnement** (cf. § 5.2.3). Dans le cas du détecteur « une impulsion sur deux », le problème est résolu en codant les états à l'aide d'un code de Gray (cf. chapitre 1, § 2.5.2.2) : à chaque changement d'état, une seule des variables d'états change de valeur, évitant ainsi l'apparition d'états parasites. Cependant, pour un automate quelconque, il n'existe pas toujours de solution simple à ce problème.

- La solution la plus utilisée en pratique consiste à introduire explicitement en sortie de la fonction G des opérateurs permettant de rendre simultanées les commutations des variables internes. Ces opérateurs, appelés **bascules**, sont étudiés dans la section 3.

2.4 Systèmes séquentiels synchrones (versus asynchrones)

Lorsque l'on est confronté à la réalisation d'un système séquentiel complexe comme peut l'être, par exemple, un microprocesseur, il devient difficile de s'appuyer sur un modèle unique et global de machine de Mealy pour décrire et concevoir l'ensemble du système. Le système est alors en général décomposé en plusieurs sous-systèmes plus simples. Il se pose alors le problème de coordonner le fonctionnement de ces différents blocs fonctionnels pour obtenir un comportement global cohérent et fiable. On recourt, dans la grande majorité des cas, à une **synchronisation globale** du système du séquentiel.

Le principe de la synchronisation consiste à utiliser un signal, en général périodique, appelé **horloge**, pour rythmer l'évolution de tous les blocs fonctionnels composant le système. Ainsi, les instants d'occurrence de tous les événements qui commandent le système sont définis en référence à une base de temps unique. Il devient alors beaucoup plus aisé de coordonner les activités des différents blocs élémentaires. Un tel système est qualifié de **synchrone** par opposition à un système **asynchrone**, c'est-à-dire sans référence temporelle globale.

A titre d'exemple, une machine de Mealy peut présenter un comportement synchrone si on privilégie une de ses entrées, appelée dans ce cas **entrée d'horloge** ou **de synchronisation**. D'un point de vue externe, cette entrée a pour effet, selon son état, de bloquer ou d'autoriser les changements d'état de la machine.

Bien que le modèle de Mealy soit tout à fait général et permette théoriquement de décrire le fonctionnement de tout circuit séquentiel, on définit habituellement plusieurs classes de fonctions séquentielles. Leur étude est abordée, dans la suite de ce chapitre, par ordre de complexité croissante. On s'intéresse, dans un premier temps, aux fonctions séquentielles les plus élémentaires que sont les bascules, puis à des fonctions plus élaborées comme les registres (cf. section 4) et les compteurs (cf. section 5), jusqu'aux structures plus complexes des automates et des séquenceurs (cf. chapitre 6). Le cas particulier des mémoires à semi-conducteur fait l'objet d'une section à part dans ce chapitre (section 7).

3. Le point mémoire : de la bascule asynchrone à la synchronisation sur front d'horloge

La fonction la plus élémentaire que peut réaliser un circuit séquentiel est la fonction mémoire. Cette section étudie tout d'abord le point mémoire élémentaire, auquel on ajoute ensuite des commandes d'écriture. On s'intéressera respectivement au point mémoire à écriture asynchrone, puis synchrone sur niveau et enfin au problème de la synchronisation sur front.

3.1 Le point mémoire élémentaire

La fonction mémoire sous sa forme la plus élémentaire est obtenue en connectant deux inverseurs tête-bêche (figure 5.5). Le rebouclage des deux portes confère à ce circuit son caractère séquentiel (cf. modèle de Mealy, § 2.2) et permet de stocker une valeur logique. Ce circuit permet de mémoriser deux états possibles :

1. $Q = 0$ et $Q^* = 1$
2. $Q = 1$ et $Q^* = 0$

Ces deux états sont stables car le circuit est câblé de telle sorte que la propagation de ces valeurs dans les inverseurs confirme cet état (nombre pair d'inverseurs). D'autre part, d'un point de vue électrique, ce circuit offre une certaine robustesse par rapport aux perturbations. Si un bruit vient perturber le signal Q ou Q^*, le circuit conserve son état tant que ces signaux restent dans la marge de bruit des inverseurs (cf. chapitre 3, § 3.1.2.2). Ce point mémoire est également désigné par le terme **bistable**.

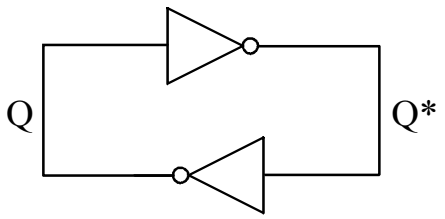


figure 5.5 : le bistable, point mémoire élémentaire

Ce bistable constitue la base de la structure des mémoires RAM statiques (cf. section 8).

Toutefois, une mémoire numérique ne présente un réel intérêt que si l'on peut modifier aisément son contenu, ce qui n'est pas le cas du bistable présenté ici. Une évolution de la structure du bistable, appelée **bascule RS**, permet de le rendre inscriptible.

3.2 La bascule RS

La bascule RS est un point mémoire dans lequel on peut écrire un 0 ou un 1 logique.

3.2.1 Structure de la bascule RS

Pour obtenir une bascule RS, on peut substituer aux inverseurs du circuit de la figure 5.5 des portes NOR, comme le montre la figure 5.6.

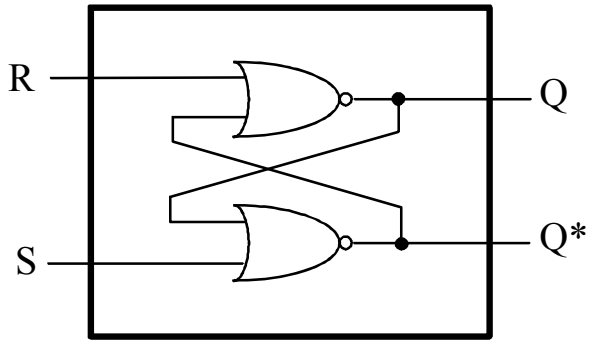


figure 5.6 : réalisation d'une bascule RS à l'aide de portes NOR

Comment fonctionne ce circuit ?

- **Mode mémoire** : Si $R = S = 0$, les entrées R et S n'ont aucune action sur le circuit. Celui-ci se comporte comme le bistable de la figure 5.5.
- **Ecriture d'un 0** : Si $R = 1$ et $S = 0$, la sortie Q est forcée à la valeur logique 0. Par propagation dans la porte du bas, la sortie Q^* est alors forcée à 1. R est l'entrée de **mise à zéro** ou de **reset**.
- **Ecriture d'un 1** : Si $R = 0$ et $S = 1$, la sortie Q^* est forcée à la valeur logique 0. Par propagation dans la porte du haut, la sortie Q est alors forcée à 1. S est appelée entrée de **mise à un** ou de **set**.
- La combinaison $R = S = 1$ est interdite car elle entraîne une mise à 0 des deux sorties, et cette configuration n'est pas mémorisable car Q et Q^* doivent avoir des valeurs logiques complémentaires.

Les entrées R et S ont une action sur l'état de la bascule lorsqu'elles sont positionnées à 1, elles sont dites **actives à 1**. On peut également réaliser une bascule RS à l'aide de portes NAND (figure 5.7). La fonctionnalité est équivalente, mais dans ce cas, les entrées R et S sont **actives à 0**.

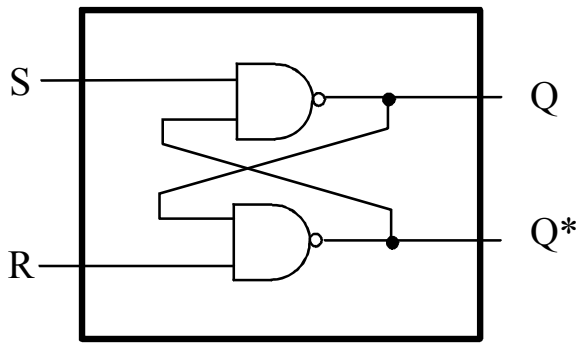


figure 5.7 : réalisation d'une bascule RS à l'aide de portes NAND

La bascule RS a un **comportement asynchrone** car une commutation sur une de ses entrées peut entraîner immédiatement la modification de l'état du circuit.

3.2.2 Analyse temporelle du comportement de la bascule RS-NOR

La figure 5.8 présente un chronogramme typique de la bascule RS réalisée avec des portes NOR. Ce chronogramme est tracé à partir de l'état initial $Q = 0$ et $Q^* = 1$, les entrées étant inactives. Les entrées *S* et *R* sont ensuite successivement activées.

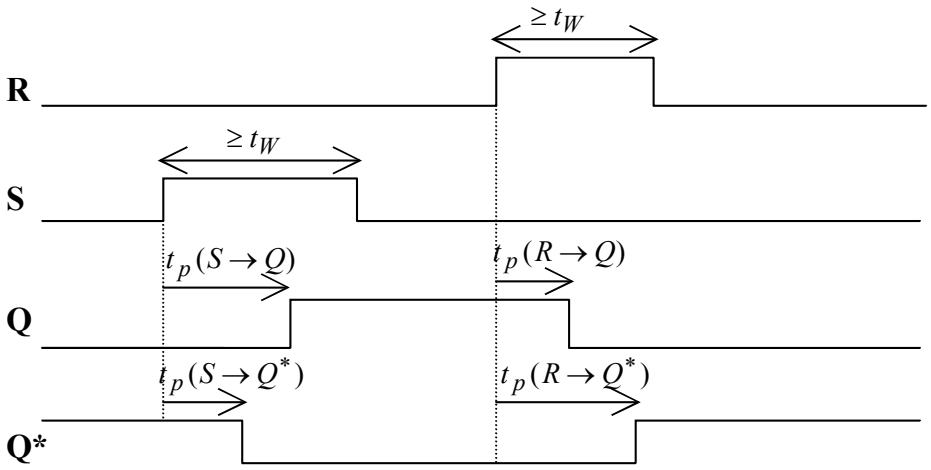


figure 5.8 : chronogramme de la bascule RS-NOR

On constate une dissymétrie dans le positionnement des sorties Q et Q^*, ce qui conduit à conclure que $Q^* = \overline{Q}$ n'est vérifiée que d'un point de vue statique et à la condition d'exclure le cas de la commande contradictoire $R = S = 1$.

Lorsque l'entrée *R* ou *S* est activée, le circuit passe par un régime transitoire avant d'arriver à un nouvel état stable. Les temps de propagation correspondants valent :

$$t_p(R \to Q) = t_p(S \to Q^*) = t_{pHL}(NOR)$$

$$t_p(S \to Q) = t_p(R \to Q^*) = t_{pHL}(NOR) + t_{pLH}(NOR)$$

Pour que la mise à zéro ou à un se passe correctement, il ne faut pas désactiver l'entrée *R* ou *S* avant que la bascule ne soit dans un état stable. Par conséquent, l'activation d'une entrée doit être maintenue pendant une durée minimale appelée **durée minimale d'impulsion**, notée t_W (Width), et

égale au temps de réaction maximal du circuit, soit $t_{pHL}(NOR) + t_{pLH}(NOR)$ dans le cas de la bascule RS-NOR.

3.2.3 Analyse statique de la bascule RS-NOR

D'un point de vue statique, c'est-à-dire en faisant abstraction du temps de propagation des portes, on peut caractériser la bascule RS-NOR par sa **table de vérité séquentielle**, encore appelée **table de transition**, ou bien encore par ses **équations séquentielles**.

Pour un circuit combinatoire, si l'on ne tient pas compte des temps de propagation, la table de vérité et les équations logiques du circuit expriment directement les sorties en fonction des entrées. Pour un circuit séquentiel, la table de transition et les équations font, de plus, intervenir l'état interne du circuit, considéré à deux instants différents : état présent et état futur (cf. machine de Mealy).

Dans le cas de la bascule RS, l'état interne de la bascule peut être représenté par la variable logique Q (ou Q^*), qui est également une sortie. Les entrées R et S et l'état Q conditionnent l'état futur de la bascule et donc les valeurs futures de Q et Q^*, notées Q^+ et Q^{*+}. Le tableau 5.2 donne la table de transition de la bascule RS-NOR.

R	S	Q^+	Q^{*+}	
0	0	Q	$\overline{Q}$	
0	1	1	0	
1	0	0	1	
1	1	0	0	combinaison interdite

tableau 5.2 : table de transition de la bascule RS-NOR

Les équations séquentielles de la bascule s'en déduisent aisément :

$$Q^+ = Q\,\overline{R}\,\overline{S} + \overline{R}\,S = (Q\,\overline{S} + S)\,\overline{R} = (Q + S)\,\overline{R}$$
$$Q^{*+} = \overline{Q}\,\overline{R}\,\overline{S} + R\,\overline{S} = (\overline{Q}\,\overline{R} + R)\,\overline{S} = (\overline{Q} + R)\,\overline{S}$$

N. B. Une analyse temporelle d'un circuit doit se faire à partir d'un schéma donnant la structure du circuit. Les équations ou la table de transition ne suffisent pas, car elles peuvent donner lieu à plusieurs logigrammes possibles.

3.3 La bascule D à verrouillage ou D latch

La bascule D Latch permet de faire un premier pas vers la synchronisation du point mémoire. Ainsi que le montre la figure 5.9, la bascule D Latch dispose de 2 entrées, comme la bascule RS, mais celles-ci jouent des rôles fort différents.

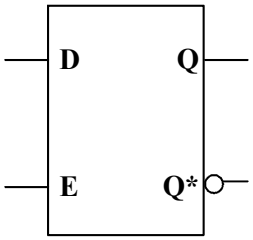


figure 5.9 : représentation symbolique de la bascule D Latch

3.3.1 Fonctionnalité de la D latch

Le fonctionnement de la bascule D latch est régi par les équations suivantes :

$$Q^{+} = ED + \overline{E}Q$$
$$Q^{*+} = E\overline{D} + \overline{E}\overline{Q}$$

Le tableau 5.3 donne la table de transition correspondante.

E	D	Q^{+}	Q^{*+}
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

tableau 5.3 : table de transition de la bascule D latch

Cette table permet de distinguer deux modes de fonctionnement de cette bascule, commandés par l'entrée E :

- Si $E = 1$, $Q^{+} = D$. Le signal présent sur l'entrée D est copié sur la sortie Q. La bascule est alors en **mode transparent** ou **mode d'acquisition**.
- Si $E = 0$, $Q^{+} = Q$. La sortie Q garde sa valeur quelle que soit la valeur du signal présent sur l'entrée D. La bascule est alors en **mode mémorisation**.

On dispose, avec la D latch, d'un moyen de définir les instants où la bascule est autorisée à évoluer, c'est le concept de synchronisation. Il s'agit ici d'une **synchronisation sur niveau** : la bascule est autorisée à évoluer (mode transparent) lorsque le signal de validation *E* est au niveau logique 1.

Les deux entrées *E* et *D* ont des rôles très différents :

- L'entrée *D* (Data) est dite **entrée de donnée** car elle fournit à la bascule la donnée à mémoriser. Cette entrée n'a aucune influence sur le mode de fonctionnement de la bascule.
- L'entrée *E* (Enable) commande le mode de fonctionnement de la bascule, c'est une **entrée de contrôle** ou de **commande**.

N. B. Il existe également des bascules D latches avec validation active sur niveau bas (figure 5.10). Dans la suite de cette section, on supposera que, sauf indication contraire, les latches étudiées sont actives sur niveau haut du signal *E*.

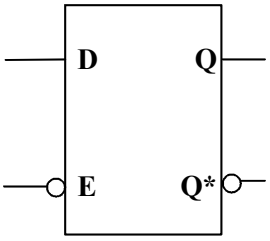


figure 5.10 : représentation symbolique d'une D latch active sur niveau bas

3.3.2 Structure de la bascule D latch

Une réalisation possible d'une bascule D latch consiste à utiliser une bascule RS-NOR et à trouver les équations $R = \mathrm{f}(E, D)$ et $S = \mathrm{g}(E, D)$. On établit, pour cela, la correspondance entre les différents modes des deux types de bascules :

- Mode mémorisation
  - D latch : $E = 0$
  - Bascule RS : $R = S = 0$
- Mise à zéro
  - D latch : $E = 1$ et $D = 0$
  - Bascule RS : $R = 1$ et $S = 0$
- Mise à un
  - D latch : $E = 1$ et $D = 1$
  - Bascule RS : $R = 0$ et $S = 1$

On en déduit :

$$R = E\overline{D}$$
$$S = ED$$

La figure 5.11 présente une réalisation possible de la bascule D latch à partir d'une bascule RS-NOR.

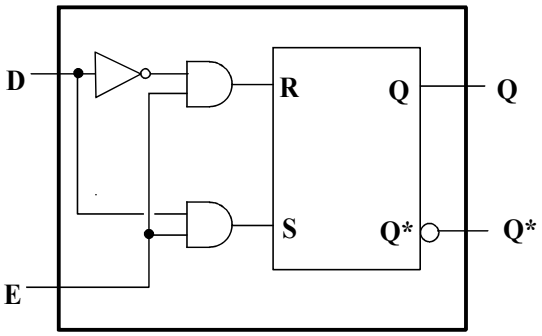


figure 5.11 : exemple de réalisation d'une D latch à partir d'une bascule RS-NOR

Cependant, en logique CMOS, ce type de structure est peu utilisé car trop encombrante. On lui préfère généralement une structure à base d'interrupteurs. En effet, l'équation $Q^+ = ED + \overline{E}Q$ montre que la bascule D latch peut être réalisée à partir d'un multiplexeur 2 vers 1 (figure 5.12).

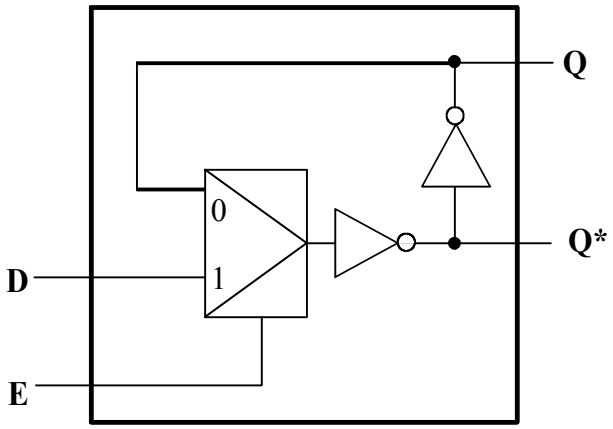


figure 5.12 : réalisation d'une bascule D latch à l'aide d'un multiplexeur

La figure 5.13 propose une structure détaillée à base d'inverseurs et d'interrupteurs de cette bascule.

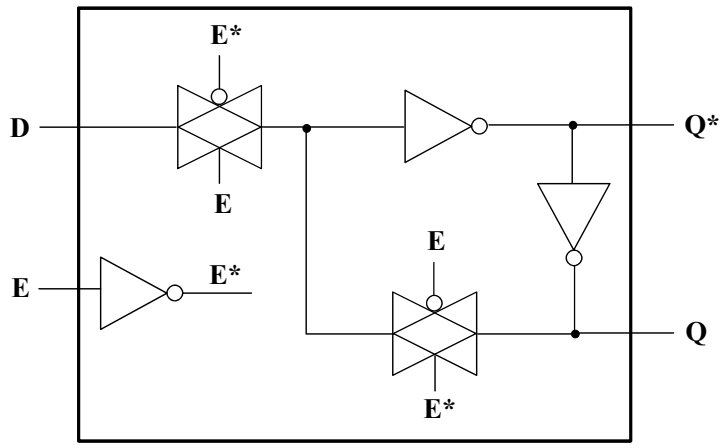


figure 5.13 : structure d'une D latch à base d'inverseurs et d'interrupteurs CMOS

N. B. Pour réaliser une latch active sur niveau bas, il suffit d'inverser les commandes E et E^* des portes de transfert dans le schéma de la figure 5.13.

3.3.3 Analyse temporelle du comportement de la D latch

La figure 5.14 présente un chronogramme typique de la D latch, avec les principaux paramètres temporels correspondants.

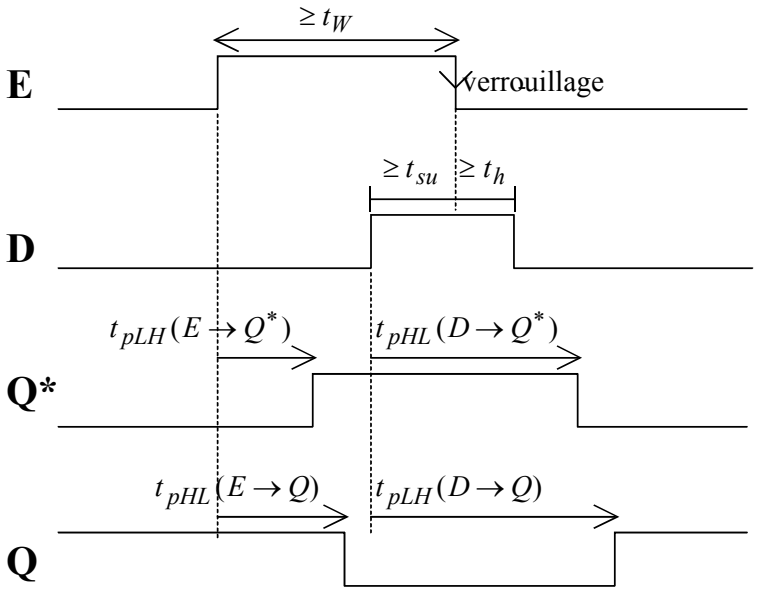


figure 5.14 : principales caractéristiques temporelles d'une D latch

3.3.3.1 Temps de propagation

Lorsque la bascule est en mode transparent ($E = 1$), si D change de valeur, les sorties commutent avec un retard $t_p(D \rightarrow Q)$ pour Q et $t_p(D \rightarrow Q^*)$ pour Q^*. Lorsque la bascule passe du mode mémorisation au mode transparent, si les valeurs de D et Q sont différentes, les sorties commutent avec un retard $t_p(E \rightarrow Q)$ pour Q et $t_p(E \rightarrow Q^*)$ pour Q^*. La valeur de ces temps de propagation est

obtenue en sommant les temps de propagation des opérateurs élémentaires situés sur le chemin entre l'entrée considérée et Q ou Q^*.

N. B. Suivant que les sorties commutent de 1 vers 0 ou de 0 vers 1, il faut considérer les temps de propagation à la montée t_{pLH} ou à la descente t_{pHL}. Tous les cas de transitions *HL* et *LH* sur les sorties ne figurent pas sur la figure 5.14.

3.3.3.2 Contraintes sur les entrées

Pour qu'une opération de mémorisation dans la bascule se déroule correctement, deux types de contraintes sont à respecter sur l'application des entrées :

- D'une part, lorsque la bascule passe en mode transparent, l'entrée *E* doit rester à 1 assez longtemps pour que la valeur de *D* présente à l'entrée puisse se propager dans la boucle de rétroaction. Il faut donc respecter une durée minimale d'impulsion t_W sur *E*.

- Lors du verrouillage de la bascule (passage du mode transparent $E = 1$ au mode mémorisation $E = 0$), la mémorisation de la valeur de *D* ne se déroule correctement que si l'entrée *D* reste stable pendant un certain temps avant et après le verrouillage.

  - Le temps t_{su}, appelé **temps de prépositionnement** ou **setup time**, désigne la durée minimale pendant laquelle l'entrée *D* doit rester stable <u>avant</u> le verrouillage de la bascule.

  - Le temps t_h, appelé **temps de maintien** ou **hold time**, désigne la durée minimale pendant laquelle l'entrée *D* doit rester stable <u>après</u> le verrouillage de la bascule.

3.4 La bascule D à déclenchement sur front ou D flip-flop

3.4.1 Introduction

Avec les bascules à déclenchement sur front, on atteint le stade ultime qui permet de considérer le temps comme une grandeur entière, et de raisonner dans un espace intégralement numérique.

La synchronisation sur niveau, bien qu'elle définisse des instants privilégiés d'activité des bascules, ne permet pas une complète discrétisation du temps. Un exemple pour le démontrer : si la bascule D latch discrétise le temps, il est alors possible de réaliser un dispositif permettant de compter ses instants d'activité lorsque le temps s'écoule. Cette fonction peut, en principe, être réalisée à l'aide d'un dispositif tel que celui de la figure 5.15 (qui traite le cas particulier de données codées sur 4 bits). Il est constitué d'une fonction combinatoire d'incrémentation, associée à un ensemble de bascules chargées d'enregistrer le résultat de l'incrémentation lorsque le signal de synchronisation est actif (*CK* = 1). Le contenu des bascules est ensuite réinjecté en entrée de l'incrémenteur. Le but de ce dispositif est donc d'incrémenter le contenu des bascules à chaque fois que le signal de synchronisation est actif.

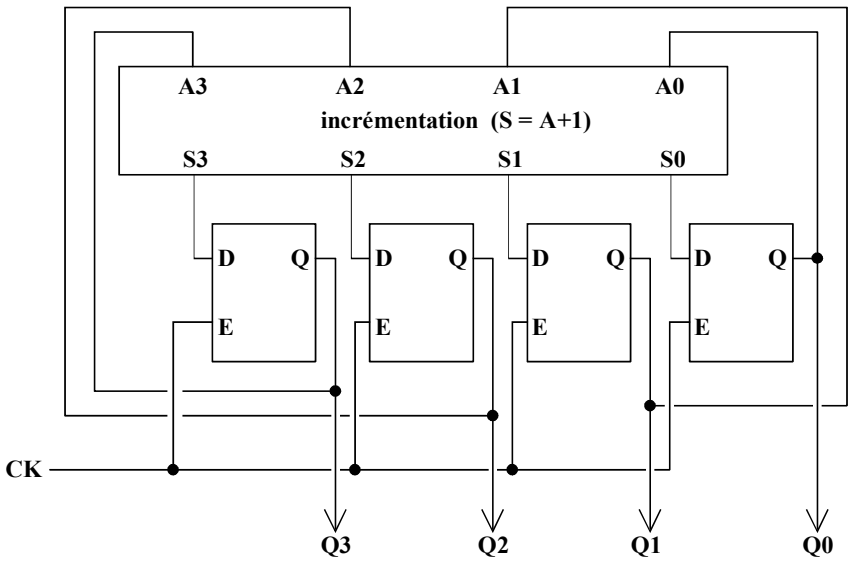


figure 5.15 : tentative de réalisation d'un compteur à l'aide de bascules D latch

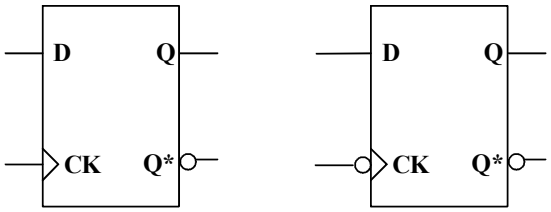
Le comportement attendu du circuit est l'énumération d'une suite binaire naturelle sur les sorties $Q_3Q_2Q_1Q_0$. En pratique le comportement observé n'est pas du tout celui espéré. Lorsque le signal de synchronisation est actif (CK = 1), les bascules sont en mode transparent et les données circulent en boucle fermée dans le dispositif. On obtient alors un comportement global apparemment désordonné, et dépendant à la fois de la durée d'activation du signal de synchronisation et des temps de propagation de chacun des éléments du montage. Ce comportement erratique est lié au fait que plusieurs données successives peuvent passer dans la bascule lorsque CK = 1. Pour obtenir un comportement correct, il faudrait que chaque bascule ne mémorise qu'une seule donnée à chaque fois que le signal de synchronisation est actif. Les **bascules à déclenchement sur front** ou **flip-flops** permettent d'obtenir un tel comportement.

N. B. Les fonctions de comptage à base de flip-flops sont étudiées en détail en section 5.

3.4.2 Principe de fonctionnement de la bascule D à déclenchement sur front

Le mode de fonctionnement de la bascule D flip-flop est lié à l'état de l'entrée d'**horloge** ou de **synchronisation** *CK* (Clock). On distingue les bascules dites **à déclenchement sur front montant** ou « **positive edge triggered** » et les bascules **à déclenchement sur front descendant** ou « **negative edge triggered** » (figure 5.16).

Les bascules à déclenchement sur front se caractérisent par le fait que leurs sorties ne peuvent commuter que lors du passage du front actif (montant ou descendant) de l'horloge. Le tableau 5.4 donne la table de transition d'une bascule D à déclenchement sur front montant.



(a) (b)

figure 5.16 : représentation symbolique de la bascule D flip-flop
(a) à déclenchement sur front montant (b) à déclenchement sur front descendant

CK	D	Q^+	Q^{*+}
0	X	Q	$\overline{Q}$
1	X	Q	$\overline{Q}$
↑	0	0	1
↑	1	1	0

tableau 5.4 : table de transition de la bascule D à déclenchement sur front montant

Si on ne prend en compte que le comportement purement synchrone de la bascule, c'est-à-dire son évolution aux fronts actifs de l'horloge *CK*, ses équations séquentielles s'écrivent $\boxed{Q^+ = D}$ et $\boxed{Q^{*+} = \overline{D}}$. Avec ce formalisme, l'action de l'horloge est implicite.

3.4.3 Structure et analyse du comportement de la bascule D flip-flop

La réalisation la plus courante des bascules à déclenchement sur front fait appel à une structure dite **maître-esclave** qui utilise deux latches (figure 5.17).

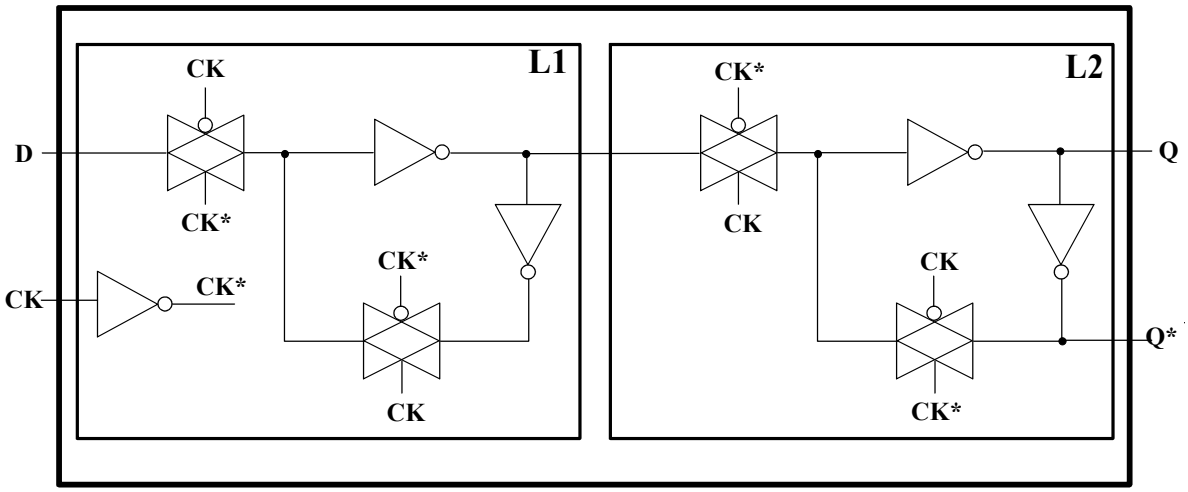


figure 5.17 : structure d'une bascule D à déclenchement sur front montant

Le mode de fonctionnement de chacune des deux latches est lié à l'état de l'horloge *CK* :

- $CK = 0$

La bascule *L*1 est en mode transparent et la bascule *L*2 est verrouillée. Ainsi, la sortie de *L*1 suit l'entrée *D*, à une inversion près, mais la sortie de *L*2 reste bloquée.

- *CK* = 1

  Lorsque *CK* passe à 1, la bascule *L*1 se verrouille, et mémorise la valeur de *D* alors présente à l'entrée. D'autre part, la bascule *L*2 passe en mode transparent et affiche la valeur de *D* mémorisée par *L*1 sur la sortie *Q*.

Pour résumer, la bascule *L*1 est chargée de faire l'acquisition, sur niveau bas de *CK*, de la valeur de *D* à mémoriser. Elle est appelée bascule d'**enregistrement**, d'**acquisition** ou bascule **maître**. La bascule *L*2 a pour rôle d'afficher sur les sorties *Q* et Q^* la valeur acquise par *L*1, lorsque *CK* passe à 1. Elle est appelée bascule d'**affichage** ou bascule **esclave**. Puisque, lorsque *CK* = 1 ou *CK* = 0, l'une des deux latches est bloquée, un changement d'état sur *D* ne peut pas être répercuté sur les sorties. La valeur présente sur l'entrée de donnée *D* n'est recopiée en sortie qu'à l'instant où *CK* passe de 0 à 1. Tout se passe donc comme si la copie de *D* sur la sortie *Q* avait lieu au moment du front montant de *CK*.

Dans le cas où les commandes des interrupteurs sont inversées par rapport au schéma de la figure 5.17, la bascule effectue la copie de *D* sur les fronts descendants de l'horloge.

3.4.4 Analyse temporelle du comportement de la D flip-flop

Les principaux paramètres temporels qui caractérisent le comportement dynamique de la bascule D flip-flop sont représentés sur la figure 5.18.

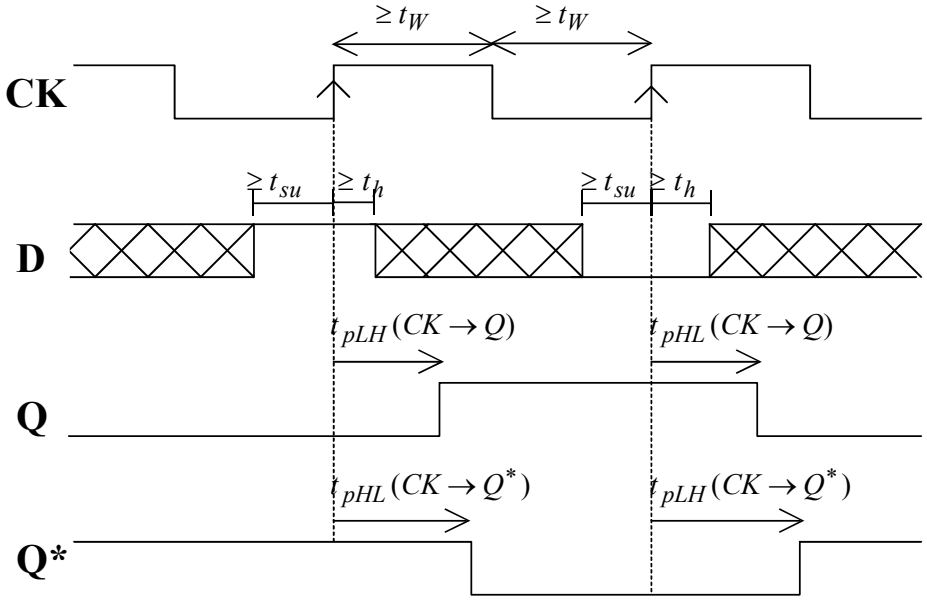


figure 5.18 : caractéristiques temporelles d'une D flip-flop à déclenchement sur front montant

3.4.4.1 Temps de propagation

A la différence de la bascule D latch, tous les temps de propagation de la bascule flip-flop sont référencés par rapport aux fronts actifs de l'horloge (fronts montants dans l'exemple de la figure 5.18).

3.4.4.2 Contraintes sur les entrées

En pratique, pour que la copie de *D* sur *Q* (et Q^*) se déroule correctement, il est nécessaire, comme dans le cas de la D latch, de respecter certaines contraintes temporelles sur les entrées.

- Durée minimale de stabilité de l'entrée *D* au voisinage du front actif de l'horloge :
  - D'une part, pour une acquisition correcte de la valeur de *D* dans *L*1, l'entrée *D* doit être stable pendant un temps minimum avant le front actif de *CK*. Il s'agit du temps de prépositionnement ou setup time t_{su} de la bascule.
  - D'autre part, pour un affichage correct en sortie de la bascule, il est nécessaire que *D* reste stable pendant un temps minimum après le front actif de l'horloge. Il s'agit du temps de maintien ou hold time t_h de la bascule.
- Il faut également respecter une durée minimale d'impulsion t_W sur les deux niveaux de l'horloge pour garantir le bon fonctionnement des deux latches (cf. § 3.3.3.2).

3.5 La bascule JK à déclenchement sur front

La bascule JK est une version synchrone de la bascule RS. L'entrée *J* joue le rôle de l'entrée *S* (mise à un), et *K* joue le rôle de *R* (mise à zéro). Tout comme pour la D flip-flop, l'activité de cette bascule peut être conditionnée par les fronts montants ou descendants du signal d'horloge.

Le tableau 5.5 donne la table de transition d'une bascule JK à déclenchement sur front descendant (figure 5.19).

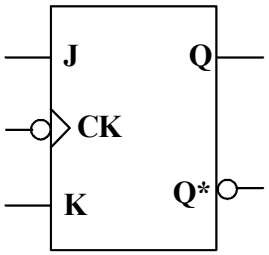


figure 5.19 : représentation symbolique d'une bascule JK à déclenchement sur front descendant

CK	J	K	Q^+	Q^{*+}
0	X	X	Q	$\overline{Q}$
1	X	X	Q	$\overline{Q}$
↓	0	0	Q	$\overline{Q}$
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	$\overline{Q}$	Q

tableau 5.5 : table de transition d'une bascule JK à déclenchement sur front descendant

Le fonctionnement synchrone de la bascule JK, qui est représenté par les 4 dernières lignes de la table de transition, est régi par les équations séquentielles suivantes :

$$Q^+ = J\overline{Q} + \overline{K}Q$$
$$Q^{*+} = \overline{J}\,\overline{Q} + KQ$$

Ce type de bascule est en pratique peu utilisé dans les circuits CMOS. Notamment, les outils de synthèse logique réalisent les fonctions séquentielles à base de bascules D exclusivement. La bascule D est, en effet, d'un encombrement moindre que la bascule JK et sa fonctionnalité est plus simple. La bascule JK est néanmoins présente dans la plupart des bibliothèques d'opérateurs logiques.

3.6 Cohabitation de fonctions asynchrones et synchrones

3.6.1 Initialisation des circuits séquentiels

En pratique, les circuits séquentiels nécessitent souvent, en plus du mode synchrone, un mode de fonctionnement permettant de forcer leur état interne de manière inconditionnelle et indépendante de l'horloge. Ce mode asynchrone est en général appliqué à la mise sous tension du circuit, pour l'amener dans un état initial connu avant le démarrage du mode opératoire normal. Il s'agit de la **phase d'initialisation** du circuit.

Les bascules sont, à cet effet, en général dotées d'entrées de mise à zéro ou à un prioritaire des sorties. Ces entrées sont le plus souvent actives à zéro. Le tableau 5.6 donne la table de transition de la bascule D flip-flop de la figure 5.20. Celle-ci possède une entrée de mise à zéro prioritaire CLR^* (**clear** ou **reset**), et une entrée de mise à un prioritaire PR^* (**preset** ou **set**).

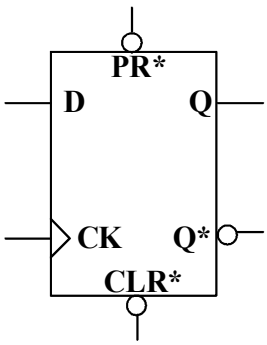


figure 5.20 : représentation symbolique d'une D flip-flop à déclenchement sur front montant avec clear et preset actifs à 0

PR^*	CLR^*	CK	D	Q^+	Q^{*+}	
0	0	X	X			état interdit
0	1	X	X	1	0	mise à 1 asynchrone
1	0	X	X	0	1	mise à 0 asynchrone
1	1	0	X	Q	$\overline{Q}$	
1	1	1	X	Q	$\overline{Q}$	
1	1	↓	0	0	1	
1	1	↓	1	1	0	

tableau 5.6 : table de transition de la bascule de la figure 5.20

L'insertion de modes de mise à zéro ou à un prioritaires et asynchrones entraîne la prise en compte de paramètres temporels supplémentaires :

- Temps de propagation : temps de réaction des sorties Q et Q^* après activation de CLR^* ou PR^*.
- Contraintes sur l'application de CLR^* et PR^* :
  - durée minimale d'impulsion sur les entrées CLR^* et PR^*,
  - …

Entrées statiques et dynamiques d'un circuit séquentiel synchrone

Pour résumer, on distingue deux types d'entrées dans un système séquentiel synchrone :

- Les entrées dont l'action sur le circuit est conditionnée à un événement de synchronisation (niveau d'un signal de validation ou front d'un signal d'horloge) sont qualifiées d'**entrées statiques**. Par exemple, les entrées *D* ou *J* et *K* d'une flip-flop sont des entrées statiques. Une telle entrée ne peut entraîner une action sur le système considéré que si la commande de synchronisation est active (par exemple au moment du front actif l'horloge, pour une flip-flop).
- Les entrées dont une commutation suffit à provoquer un changement d'état du circuit sont qualifiées d'**entrées dynamiques**. C'est le cas par exemple des entrées *R* et *S* d'une bascule du même nom, des entrées d'initialisation, ou bien de l'entrée d'horloge d'un circuit synchrone. Si l'on excepte l'entrée d'horloge, une entrée dynamique a une action asynchrone sur le système, car indépendante de l'horloge.

3.7 Modèle structurel d'un système séquentiel synchrone

Le modèle structurel de la machine de Mealy présenté au § 2.2 est général et permet de matérialiser tout système séquentiel, qu'il soit asynchrone ou synchrone. Dans le second cas, on peut dériver un modèle spécifique en substituant à la boîte introduisant le délai temporel Δt (figure 5.3) une batterie de bascules flip-flops, à raison d'une bascule par variable interne. Le modèle obtenu, qui est générique pour tout système séquentiel synchrone, est représenté sur la figure 5.21. C'est sur la base de ce modèle que l'on pourra par la suite analyser ou synthétiser tout système séquentiel synchrone de complexité raisonnable.

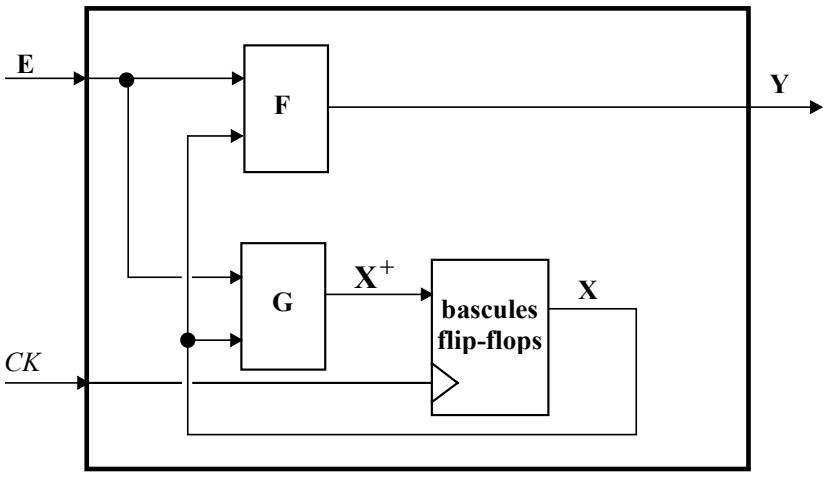


figure 5.21: modèle structurel de Mealy d'un système séquentiel synchrone

N. B.

- L'utilisation de flip-flops permettant de discrétiser la grandeur temps, on note parfois $\mathbf{X_n}$ et $\mathbf{X_{n+1}}$ les états présent et futur de la machine de Mealy, en référence aux instants nT et $(n+1)T$, où T est la période de l'horloge *CK*.
- Pour obtenir une machine de Mealy dont le comportement est globalement synchrone, il est indispendable que les entrées **E** soient synchronisées sur *CK*. Dans le cas contraire, les sorties **Y** n'évoluent pas en synchronisation avec *CK*.

Dans les systèmes numériques, les informations traitées sont en général des mots ou des nombres de *n* bits, et non des bits considérés indépendamment les uns des autres. Pour la réalisation de fonctions séquentielles, on a alors besoin d'opérateurs permettant de traiter les mots binaires : mémorisation d'un mot, extraction d'un bit ou d'un ensemble de bits du mot, pour effectuer un traitement particulier en fonction de la valeur ou de l'emplacement de ces bits, etc. On a également besoin de fonctions permettant d'établir un ordre de succession entre des événements. Deux types d'opérateurs séquentiels couramment utilisés pour réaliser ces traitements sont les **registres** et les **compteurs**.

4. Les registres

4.1 Introduction

Un registre est un ensemble de bascules permettant de stocker une information en attendant son traitement. Suivant l'interconnexion des bascules, les données stockées peuvent être soumises à différents types de manipulations.

4.2 Les registres de mémorisation ou registres tampons

Un registre de mémorisation est un ensemble de bascules synchronisées par la même horloge *H* et permettant de stocker momentanément un mot ou un nombre binaire (figure 5.22).

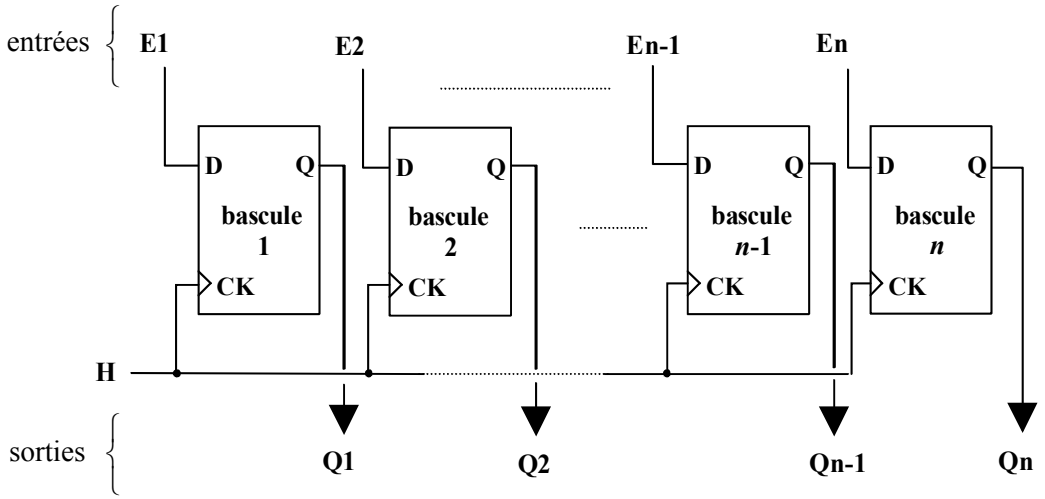


figure 5.22 : registre permettant la mémorisation de mots de *n* bits

Le registre de la figure 5.22 utilise des bascules à déclenchement sur front (flip-flops), mais pourrait également utiliser des bascules à verrouillage (latches).

4.3 Les registres à décalage

Les bascules du registre sont interconnectées de telle façon qu'à chaque période d'horloge l'information contenue dans le registre soit décalée. Le déplacement s'effectue vers la droite ou vers la gauche. La réalisation de la fonction de décalage nécessite l'utilisation de bascules flip-flops. La réalisation de tels registres à l'aide de latches poserait les mêmes problèmes que ceux évoqués en section 3.4.1.

4.3.1 Fonction décalage à droite

La bascule de rang *i* recopie la sortie de la bascule de rang *i* - 1. Son entrée *D* est donc connectée à la sortie *Q* de rang *i* - 1 (figure 5.23).

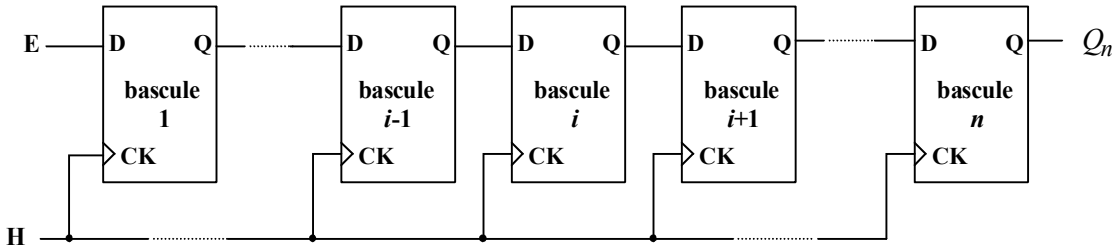


figure 5.23 : réalisation de la fonction décalage à droite

Dans l'exemple de la figure 5.23, la valeur appliquée sur l'entrée *E* est recopiée sur la sortie de la bascule *n* après *n* fronts actifs de l'horloge *H*. Le contenu du registre peut être lu en série sur la sortie Q_n, ou bien en parallèle sur l'ensemble des sorties $Q_1 \cdots Q_n$ des bascules. On peut, à partir de ce registre, réaliser une fonction de décalage circulaire (rotation) à droite en connectant Q_n à *E*.

4.3.2 Fonction décalage à gauche

La bascule de rang *i* recopie la sortie de la bascule de rang *i* + 1. Son entrée *D* est donc connectée à la sortie *Q* de rang *i* + 1 (figure 5.24).

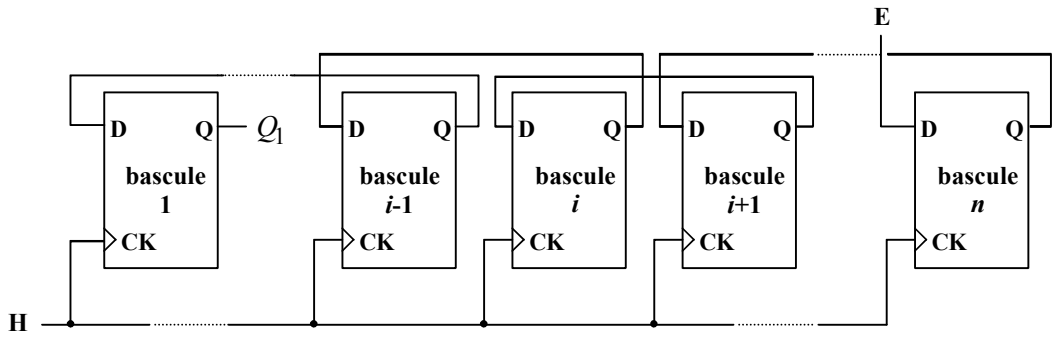


figure 5.24: réalisation de la fonction décalage à gauche

Dans l'exemple de la figure 5.24, la valeur appliquée sur l'entrée *E* est recopiée sur la sortie de la bascule 1 après *n* fronts de l'horloge *H*. Le contenu du registre peut être lu en série sur la sortie Q_1 ou

bien en parallèle sur l'ensemble des sorties Q des bascules. On peut, à partir de ce registre, réaliser une fonction de décalage circulaire (rotation) à gauche en connectant Q_1 à E.

4.3.3 Chargement parallèle

Le registre à décalage peut être doté d'une fonction de chargement parallèle permettant de mémoriser directement un mot binaire dans les bascules. Lorsque l'entrée de chargement *LOAD* est activée, le mot présenté en entrée est mémorisé dans la bascule lors du front actif de l'horloge H. La figure 5.25 montre un exemple de réalisation de commande de chargement parallèle de mots de 4 bits. Lorsque la commande *LOAD* est à 1, le mot binaire (EP_1, EP_2, EP_3, EP_4) est chargé dans le registre. Lorsque *LOAD* est à 0, le registre est configuré en mode décalage.

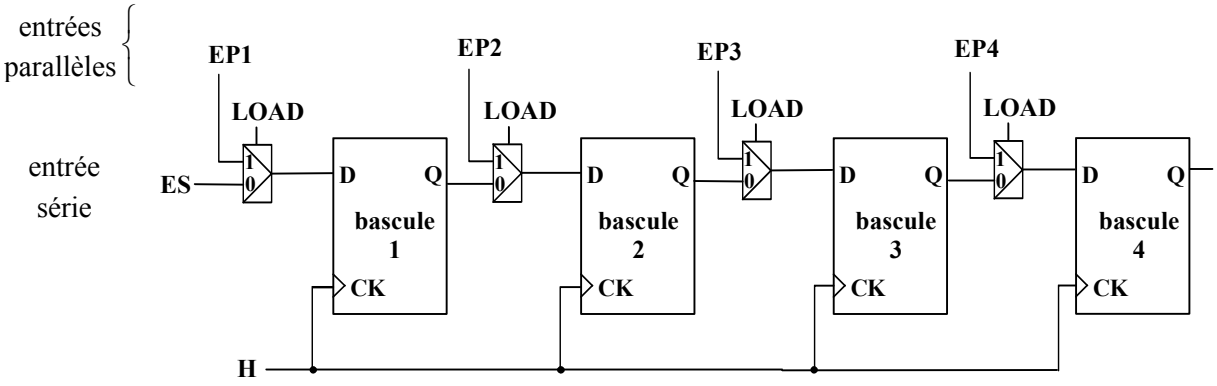


figure 5.25 : registre à décalage à chargement parallèle

4.3.4 Initialisation

Lorsqu'elle est activée, cette commande force le contenu du registre à une valeur prédéterminée. Elle utilise les entrées dynamiques de remise à zéro ou à un des différentes bascules du registre (cf. § 3.6.1). Il s'agit, le plus souvent, d'une remise à zéro générale du registre. La figure 5.26 montre le cas d'une initialisation à 0101 d'un registre à décalage de taille 4. La commande d'initialisation *INIT* est active à zéro.

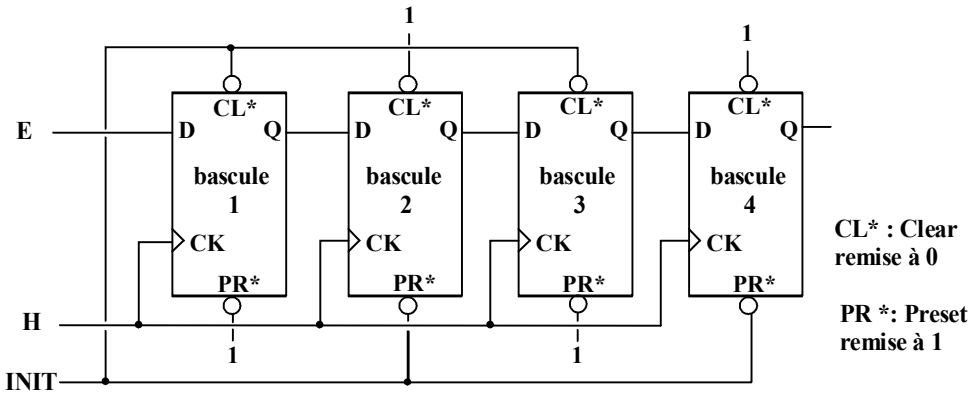


figure 5.26 : exemple de mécanisme d'initialisation d'un registre à décalage

N. B. : La commande d'initialisation asynchrone d'un circuit séquentiel est principalement utilisée à la mise sous tension du circuit. Sauf cas particulier, elle ne doit pas être activée lors du fonctionnement normal du circuit, sous peine de provoquer un fonctionnement erroné du circuit (cf. § 6.2).

4.3.5 Registres universels

Les fabricants de circuits intégrés proposent dans leurs catalogues de circuits standard des registres dits « universels » regroupant les différentes fonctionnalités décrites dans les sections précédentes. Ces circuits permettent, en mode synchrone, de réaliser les fonctions suivantes :

- chargement parallèle ou série,
- lecture parallèle ou série,
- décalage à gauche ou à droite

La vue externe d'un tel registre est donnée par la figure 5.27.

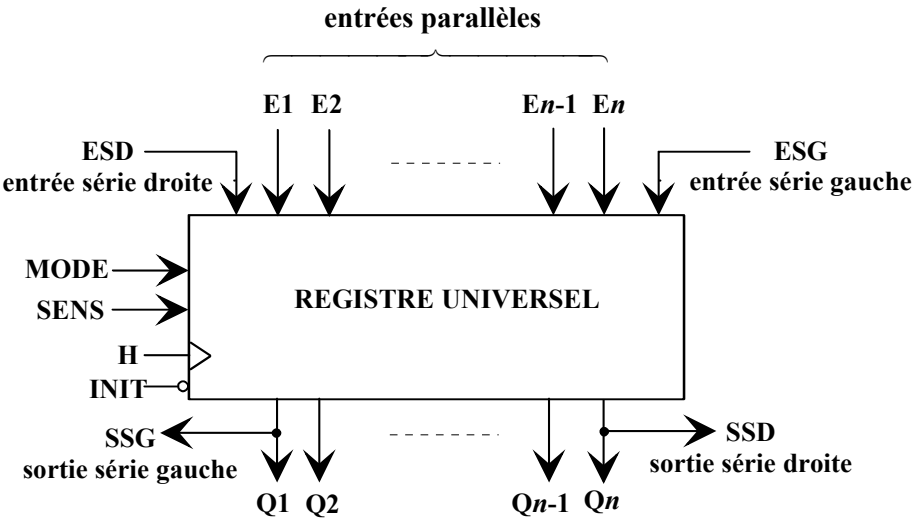


figure 5.27 : vue externe d'un registre universel

Le mode de fonctionnement du registre est déterminé par deux **entrées de contrôle**, *MODE* et *SENS*. La valeur logique de l'entrée *MODE* indique le mode de fonctionnement du registre, décalage ou chargement parallèle. La valeur logique de *SENS* indique le sens du décalage, droite ou gauche. Les **entrées de données** prises en compte sont *ESD* pour un décalage à droite, *ESG* pour un décalage à gauche, $E_1 \cdots E_n$ pour un chargement parallèle.

Le contenu du registre peut être lu en parallèle sur les sorties $Q_1 \cdots Q_n$ des bascules, ou en série sur *SSD* (*SSG*) dans le cas d'un décalage à droite (à gauche).

Toutes les entrées, excepté *INIT,* sont statiques, car leur état n'est pris en compte que sur le front actif de l'horloge *H*. L'entrée d'initialisation est dynamique car son activité est indépendante de celle de l'horloge (asynchrone). Elle est, par conséquent, prioritaire sur les autres entrées.

4.3.5.1 Structure d'une cellule

La figure 5.28 montre un exemple de réalisation d'une cellule du registre décrit ci-dessus, avec les commandes suivantes :

- *MODE* = 0, mode chargement parallèle (ou mode registre tampon),
- *MODE* = 1, mode décalage,
  - *SENS* = 1, décalage à droite,
  - *SENS* = 0, décalage à gauche

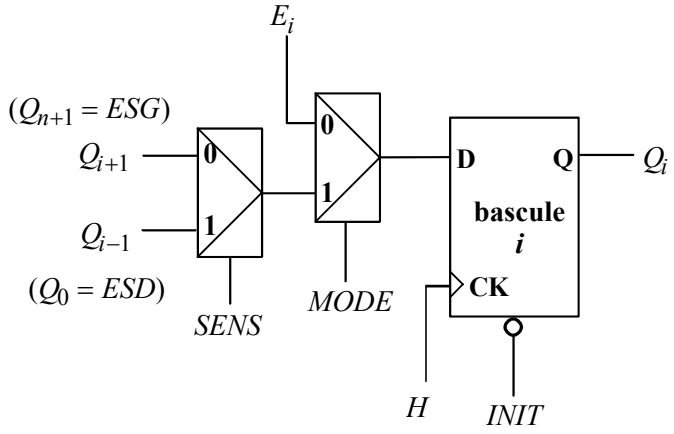


figure 5.28 : détail d'une cellule de registre universel

4.3.5.2 Identification avec le modèle de Mealy

A partir de la figure 5.28, l'identification du registre à décalage universel avec le modèle synchrone de la machine de Mealy (figure 5.21) est immédiate.

- Entrées : $\mathbf{E} = (E_1, \ldots, E_n,\ MODE,\ SENS)$
- Horloge : H
- État présent : $\mathbf{X} = (Q_1, \ldots, Q_n)$
- État futur : $\mathbf{X}^+ = (D_1, \ldots, D_n)$
- Sorties : $\mathbf{Y} = (Q_1, \ldots, Q_n)$

La fonction de sortie F est triviale puisque les sorties du registre sont données par les sorties Q des bascules et la fonction « état suivant » G est donnée par

$$D_i = (Q_{i-1} SENS + Q_{i+1} \overline{SENS})\, MODE + E_i\, \overline{MODE}\ , \text{ pour } 1 \le i \le n\ ,$$
avec $Q_0 = ESD,$ et $Q_{n+1} = ESG$.

4.3.6 Applications des registres à décalage

4.3.6.1 Conversions parallèle-série et série-parallèle d'un train d'information

Si l'information à traiter est constituée de mots binaires de *n* bits, l'utilisation d'un registre de taille *n* permet une conversion parallèle-série de cette information :

1. Premier front actif d'horloge : chargement parallèle d'un mot dans le registre,
2. *n - 1* fronts suivants : fonctionnement du registre en mode décalage pour récupérer le mot en série sur la sortie SSD ou SSG, suivant le sens du décalage, et retour à l'étape 1.

Pour une récupération des données en série à un rythme *f* (fréquence de *H*), les mots doivent être appliqués en parallèle au rythme *f* / *n*.

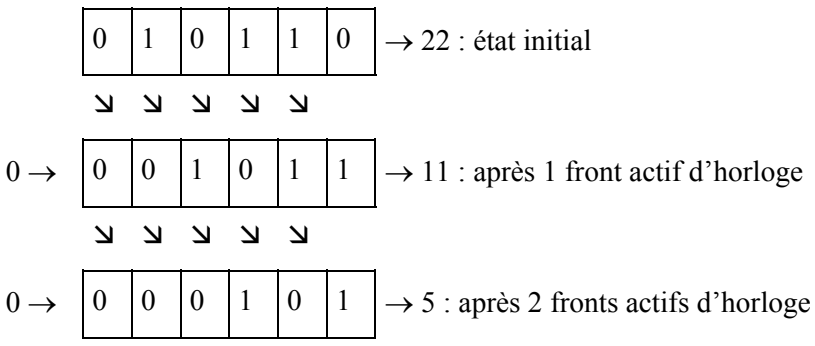
Si, au contraire l'information traitée est insérée en série au rythme *f* par décalage dans le registre (entrée *ESD* ou *ESG* suivant le sens du décalage), la saisie des *n* sorties des bascules en parallèle au rythme *f* / *n* permet une conversion série-parallèle.

4.3.6.2 Ligne à retard numérique

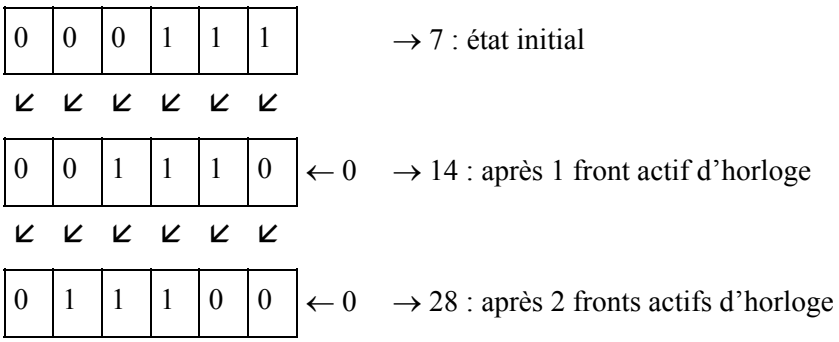
Un registre de taille *N* programmé en mode décalage permet de retarder un train de bits de *N* périodes d'horloge. Le train de bit est appliqué sur l'entrée série du registre et récupéré sur la sortie série. Pour retarder des mots de *n* bits, il faut utiliser *n* registres identiques, un pour chaque bit du mot.

4.3.6.3 Multiplication et division par 2^n

Un décalage à droite de *n* bits, avec un remplissage à gauche avec des zéros (*ESD* = 0), permet d'effectuer une division par 2^n. Par exemple,



Un décalage à gauche de *n* bits, avec un remplissage à droite avec des zéros (*ESG* = 0), permet d'effectuer une multiplication par 2^n. Par exemple,



4.3.6.4 Réalisation de générateurs de séquences pseudo-aléatoires

La structure générale d'un générateur de séquences pseudo-aléatoires est donnée par la figure 5.29.

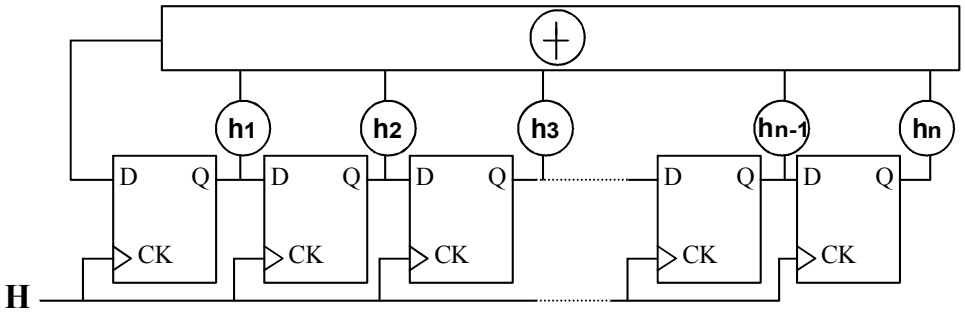


figure 5.29 : forme générale d'un générateur de séquences pseudo-aléatoires.

Les sorties de certaines bascules du registre à décalage sont rebouclées sur l'entrée du registre par l'intermédiaire d'opérateurs OUEX en fonction de la valeur des paramètres h_i. La structure exacte du générateur pseudo-aléatoire est donnée par son polynôme générateur $P(X) = 1 + h_1X + h_2X^2 + \cdots + h_{n-1}X^{n-1} + h_nX^n ; h_i \in \{0,1\}$. Si $h_i = 1$, la sortie de la bascule n° *i* est rebouclée sur l'entrée, si $h_i = 0$, elle ne l'est pas.

La séquence produite par un tel registre est de caractère pseudo-aléatoire (séquence périodique de $2^n - 1$ mots de *n* bits dont la fonction d'autocorrélation discrète est proche de celle d'un bruit blanc) si le polynôme générateur *P*(*X*) est un polynôme primitif dans le corps de Galois GF(2) (i. e. *P*(*X*) irréductible et $k = 2^n - 1$ est le plus petit entier tel que *P*(*X*) divise $1 + X^k$). La liste des polynômes convenant est donnée dans tous les ouvrages spécialisés dans la théorie du codage.

La figure 5.30 donne la structure du générateur de polynôme $P(X) = 1 + X + X^4$.

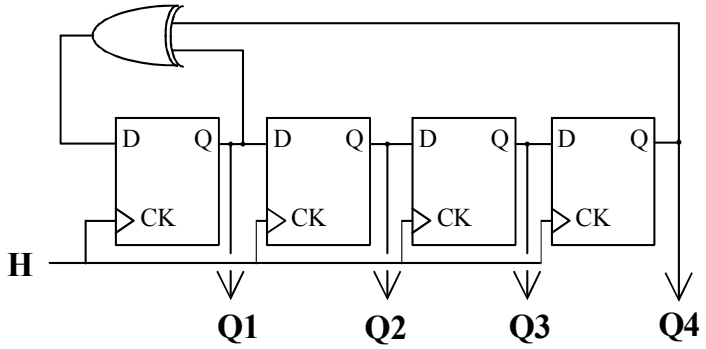
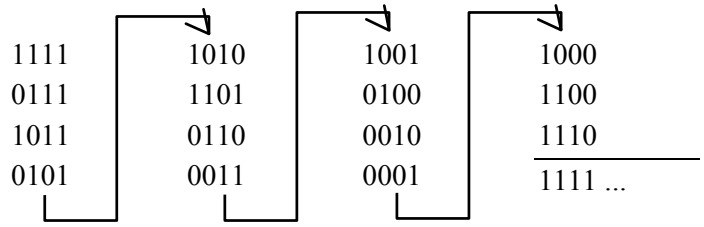


figure 5.30 : exemple de générateur pseudo-aléatoire

La séquence résultante, obtenue par la saisie en parallèle des sorties des bascules à chaque période de *CK*, est constituée de tous les mots binaires de 4 bits, excepté 0000. La séquence produite est la suivante (le registre est initialisé au départ à 1111) :



5. Les compteurs

5.1 Introduction

5.1.1 La fonction de comptage

Un compteur est un circuit séquentiel permettant d'établir une correspondance univoque entre le nombre d'impulsions appliquées sur son entrée d'horloge et l'état de sortie correspondant. Les opérateurs de base d'un compteur sont les bascules à déclenchement sur front. L'état du comptage à chaque instant est donné par la sortie des bascules. Un compteur constitué de n bascules peut délivrer au plus 2^n combinaisons de sortie. Les compteurs les plus courants énumèrent le codage binaire naturel.

5.1.2 Le diviseur par 2

La fonction de comptage la plus élémentaire, comptage sur 1 bit, est obtenue en connectant une bascule D comme indiqué sur la figure 5.31. Le chronogramme présenté part de la condition initiale $Q = 0$, soit $D = 1$. Au premier front actif (montant, dans ce cas) de l'horloge H, la valeur de $D = Q^* = \overline{Q}$ est recopiée sur Q qui s'inverse donc. La sortie Q change ainsi de valeur à chaque front actif de l'horloge. Ce montage est plus couramment appelé diviseur par 2 car les sorties des bascules, Q et Q^* sont des signaux de fréquence $f / 2$, si f est la fréquence de l'horloge H. Ces signaux sont, de plus, de **rapport cyclique** 1/2 car les durées des niveaux haut et bas sont égales.

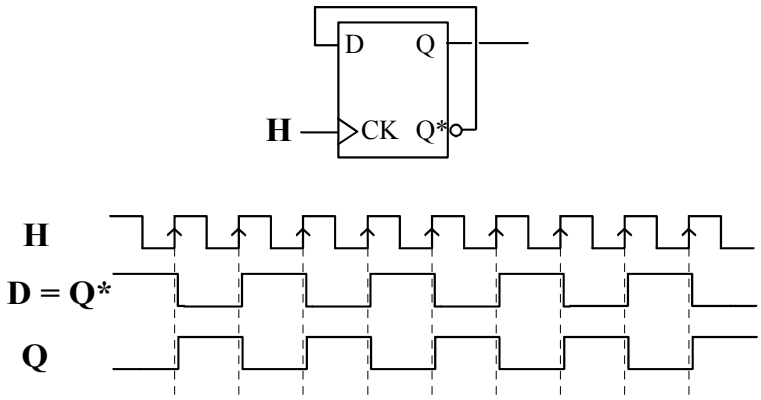


figure 5.31 : bascule montée en diviseur par 2

N.B. Pour un fonctionnement correct du diviseur par 2, il faut s'assurer que le temps de propagation entre l'entrée D et la sortie Q^* est supérieur au temps de maintien ou hold time t_h de la bascule, ce qui est toujours le cas en pratique.

5.1.3 Comptage synchrone / asynchrone

Les compteurs sont classés en deux catégories suivant leur mode de fonctionnement et leur structure. On distingue :

- les compteurs **asynchrones**
- les compteurs **synchrones**

Un système séquentiel est **synchrone** lorsque tous les changements d'états du système sont liés à l'activité du même signal d'horloge (cf. § 2.4). Si cette condition n'est pas vérifiée, le système est dit **asynchrone**. Par exemple, les registres à décalage et le diviseur par 2 sont des circuits synchrones. Les compteurs étudiés dans la section suivante sont asynchrones.

5.2 Les compteurs asynchrones

Un compteur asynchrone possède un signal d'horloge, mais celui-ci ne sert qu'au déclenchement de la première bascule. Le signal d'horloge des bascules suivantes résulte d'une combinaison logique des sorties des autres bascules. La structure des compteurs asynchrones permet de propager en cascade l'ordre de changement d'état des bascules.

5.2.1 Compteurs binaires asynchrones à cycles complets

Un compteur binaire à cycle complet sur *n* bits est constitué de *n* bascules et permet d'énumérer dans l'ordre les 2^n valeurs du code binaire naturel. Les bascules sont montées en diviseur par 2 et cascadées comme suit : la première bascule est cadencée par l'horloge du circuit *H*, et chaque bascule suivante utilise comme horloge la sortie Q^* de la bascule précédente. A titre d'exemple, la figure 5.32 présente un compteur asynchrone à cycle complet sur 3 bits. Les changements de valeur sur Q_2 et Q_3 sont déclenchés sur les fronts descendants respectifs de Q_1 et Q_2. Le chronogramme de la figure 5.32 a été tracé à partir de l'état initial 000. Le compteur énumère sur ses sorties le code binaire naturel sur 3 bits de manière cyclique.

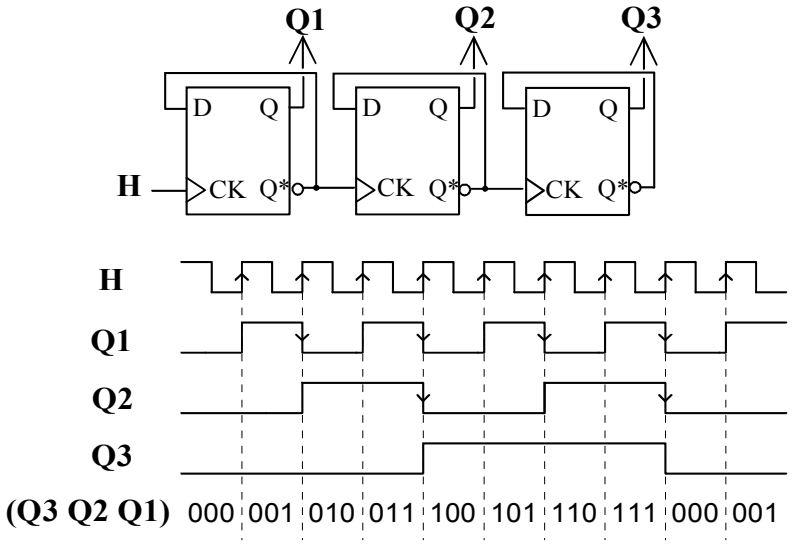


figure 5.32 : compteur asynchrone à cycle complet sur 3 bits

On notera que ce compteur réalise également les fonctions de division de fréquence par 2 (sorties Q_1 et Q_1^*), par 4 (sorties Q_2 et Q_2^*), et par 8 (sorties Q_3 et Q_3^*).

Le chronogramme de la figure 5.32 ne tient pas compte des temps de propagation dans les opérateurs. En pratique, la structure cascadée du compteur induit un cumul des retards entre l'horloge *H* et les sorties. Les temps de propagation sur les 3 sorties sont donnés par :

$$t_p(H \to Q_1) = t_p(CK \to Q)_{bascule1}$$
$$t_p(H \to Q_2) = t_p(H \to Q_1^*) + t_p(Q_1^* \to Q_2) = t_p(CK \to Q^*)_{bascule1} + t_p(CK \to Q)_{bascule2}$$
$$t_p(H \to Q_3) = t_p(H \to Q_1^*) + t_p(Q_1^* \to Q_2^*) + t_p(Q_2^* \to Q_3^*)$$
$$= t_p(CK \to Q^*)_{bascule1} + t_p(CK \to Q^*)_{bascule2} + t_p(CK \to Q)_{bascule3}$$

Les temps de propagation s'ajoutant d'une bascule à la suivante, les compteurs asynchrones de grande taille sont relativement lents. D'autre part, le décalage des sorties les unes par rapport aux autres produit des états transitoires indésirables après chaque front actif d'horloge.

5.2.2 Décompteurs binaires asynchrones à cycles complets

La structure du décompteur binaire est très proche de celle du compteur binaire : les bascules sont également utilisées en diviseur par 2 et cascadées, mais utilisent comme horloge la sortie *Q* des bascules précédentes, et non la sortie Q^*. La figure 5.33 donne la structure d'un décompteur sur 3 bits. Le chronogramme a été tracé à partir de l'état initial 000. Le décompteur énumère à l'envers le code binaire naturel sur 3 bits de manière cyclique.

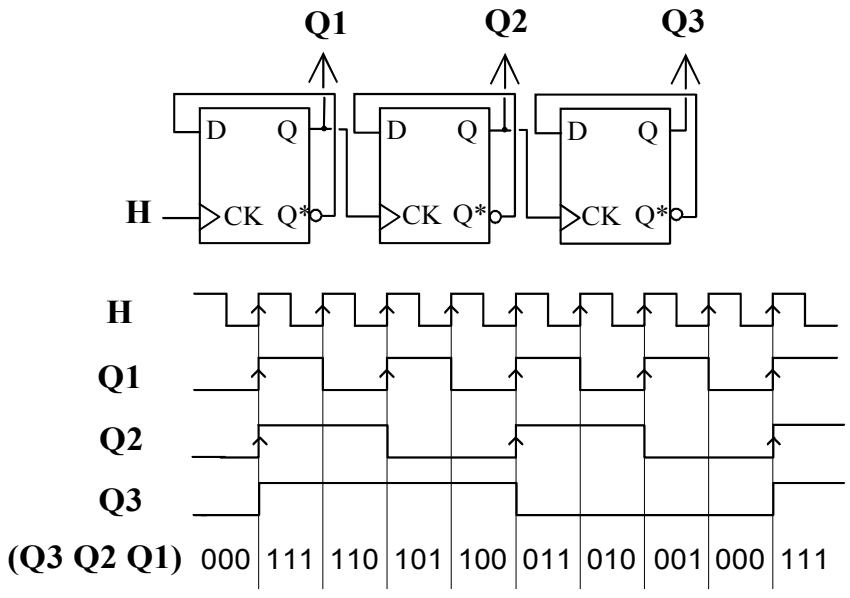


figure 5.33 : décompteur asynchrone à cycle complet sur 3 bits

5.2.3 Compteurs / décompteurs asynchrones à cycles incomplets

Avertissement : la méthode présentée ci-après est parfois utilisée en pratique, mais nous ne la recommandons pas car elle peut entraîner des aléas de fonctionnement, c'est-à-dire des erreurs de fonctionnement difficilement prévisibles (cf. § 6.2).

Les compteurs à cycles complets permettent un comptage binaire de 0 à $2^n - 1$, si n est le nombre de bascules du compteur. Pour un comptage modulo N, où N n'est pas une puissance de 2, la méthode généralement utilisée est la suivante :

1. On réalise le compteur à cycle complet dont le modulo est immédiatement supérieur à N. Ce compteur est constitué de n bascules, où n vérifie $2^{n-1} < N < 2^n$. Par exemple, pour réaliser un compteur décimal (modulo 10), on part d'un compteur modulo 16 à 4 bascules.
2. On tronque son cycle par un rebouclage asynchrone : lorsque le cycle est terminé, le compteur revient à l'état 0...0 par activation des entrées dynamiques de remise à zéro (clear ou reset) des bascules. Il faut pour cela ajouter une fonction combinatoire de détection de l'état N.

La figure 5.34 montre la structure et le fonctionnement **théorique** d'un compteur modulo 5 asynchrone.

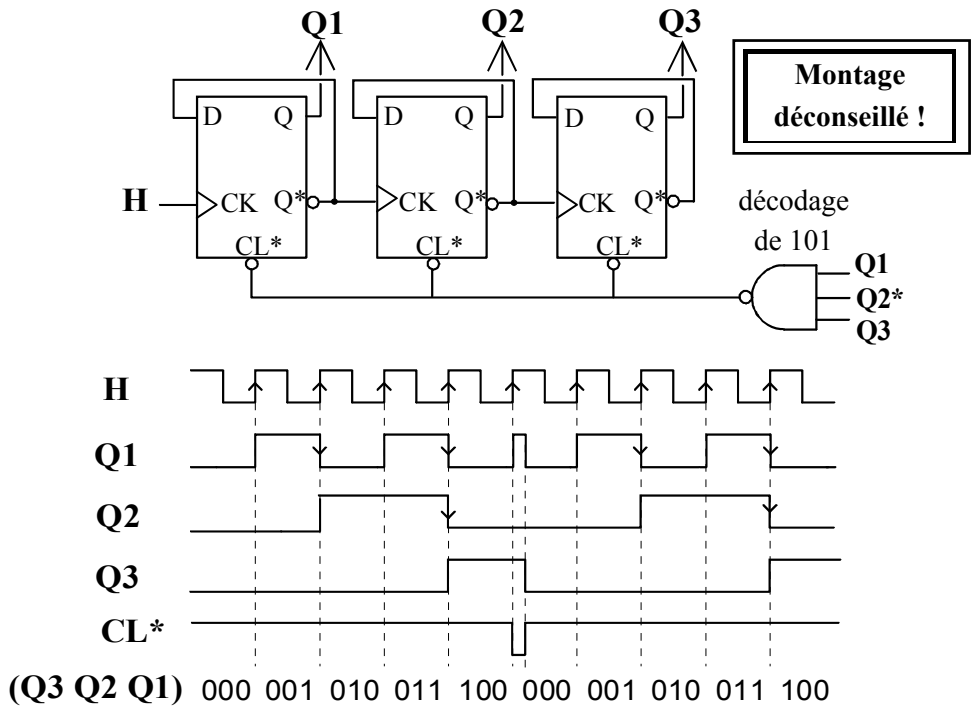


figure 5.34 : structure et comportement **théorique** d'un compteur asynchrone modulo 5

Le comptage débute normalement. Lorsque le compteur passe à l'état 5, cette configuration est détectée (passage à 0 de la sortie de la porte NAND), le reset des bascules est activé, et les sorties Q des bascules sont forcées à 0. Dès que la configuration 101 n'est plus présente en entrée de la porte NAND, l'entrée de remise à zéro des bascules repasse à 1, et le compteur retourne dans un état de comptage normal.

La réalisation de tels compteurs **requiert beaucoup de précautions** :

- D'une part, il faut prendre en compte la présence d'un état transitoire indésirable sur les sorties *Q* lors de la détection entraînant la remise à zéro.
- Cette méthode présente, surtout, des **risques d'aléas**, c'est-à-dire de fonctionnements erronés difficilement prévisibles. En particulier, si les bascules présentent des dispersions importantes sur les temps de réaction à l'activation du reset, il se peut que l'entrée CL^* des bascules repasse à 1 avant que toutes les sorties ne soient forcées à 0. De plus, pour certains montages, le décalage des sorties les unes par rapport aux autres peut entraîner l'apparition indésirable de la configuration de remise à zéro (cf. § 6.2).

5.2.4 Conclusion sur l'utilisation des compteurs asynchrones

L'avantage principal des compteurs asynchrones réside dans leur simplicité de réalisation. En revanche, leur utilisation présente des inconvénients non négligeables :

- vitesse de fonctionnement limitée car les temps de propagation s'ajoutent,
- présence d'états transitoires indésirables après chaque front actif de l'horloge,
- **risque d'aléas de fonctionnement dans le cas de compteurs / décompteurs à cycles incomplets.**

En pratique, les compteurs à cycles complets sont parfois utilisés pour des applications ne nécessitant pas une fréquence d'horloge très élevée, mais les compteurs à cycles incomplets ne présentant pas une sûreté de fonctionnement suffisante, on leur préfère de loin l'utilisation de compteurs synchrones.

5.3 Les compteurs synchrones

La réalisation de compteurs synchrones permet d'éliminer l'ensemble des inconvénients cités ci-dessus. A cette fin, toutes les bascules du compteur doivent être synchronisées par la même horloge.

5.3.1 Méthode de synthèse des compteurs synchrones

La structure des compteurs synchrones est un peu plus complexe que celle des compteurs asynchrones. En revanche, il est possible de réaliser des compteurs couvrant une gamme beaucoup plus large de fonctions que pour les compteurs asynchrones. Une méthode systématique unique permet de construire facilement des compteurs à cycle complet, à cycle incomplet, des décompteurs, ou de mettre en œuvre des énumérations autres que le comptage binaire naturel.

La réalisation d'un compteur à *N* états nécessite, comme dans le cas des compteurs asynchrones, *n* bascules, avec $2^{n-1} < N \leq 2^n$. La construction du compteur passe par l'écriture de sa table de transition, c'est-à-dire la table donnant pour chaque état du compteur l'état suivant correspondant. Cette table permet ensuite de calculer l'équation logique de l'entrée *D* de chaque bascule en fonction des sorties *Q* de l'ensemble des bascules. Nous allons traiter plusieurs exemples pour illustrer ce principe.

5.3.1.1 *Exemple de synthèse de compteur binaire synchrone à cycle complet : compteur modulo* 8

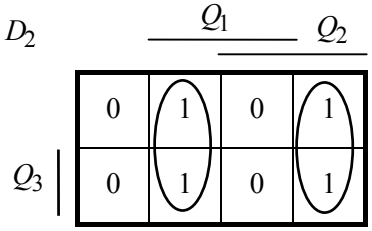
La table de transition réduite du compteur binaire modulo 8 est donnée par le tableau 5.7. On n'y fait apparaître que le fonctionnement purement synchrone du compteur, c'est-à-dire son comportement au moment des fronts actifs d'horloge.

Q_3	Q_2	Q_1	Q_3^+	Q_2^+	Q_1^+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

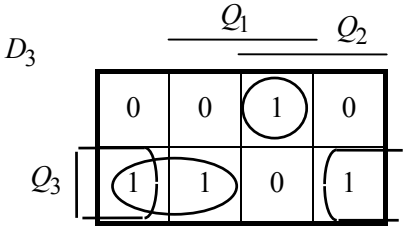
tableau 5.7 : table de transition réduite d'un compteur binaire modulo 8

Le compteur est donc constitué de 3 bascules. L'expression des entrées *D* des bascules en fonction de Q_1, Q_2, et Q_3 est directement déduite de la table de transition précédente, car pour les bascules D, $Q_i^+ = D_i, i \in \{1;2;3\}$ (cf. § 3.4.2). Ces expressions peuvent être simplifiées soit directement, soit en utilisant la méthode de Karnaugh.

- L'obtention de D_1 est immédiate : $D_1 = \overline{Q_1}$
- Pour D_2 et D_3, on trace les diagrammes de Karnaugh :



On en déduit $D_2 = Q_1\overline{Q_2} + \overline{Q_1}Q_2 = Q_1 \oplus Q_2$



On en déduit $D_3 = \overline{Q_2}Q_3 + \overline{Q_1}Q_3 + Q_1Q_2\overline{Q_3}$. Cette expression peut encore s'écrire sous la forme : $D_3 = (\overline{Q_1} + \overline{Q_2})Q_3 + Q_1Q_2\overline{Q_3} = \overline{Q_1Q_2}Q_3 + Q_1Q_2\overline{Q_3} = Q_1Q_2 \oplus Q_3$

La structure du compteur correspondant est donnée en figure 5.35. On a utilisé au maximum les sorties Q^* des bascules pour minimiser le nombre de portes logiques nécessaires à la réalisation de la fonction « état suivant ». Notamment, la fonction ET ($Q_1 . Q_2$) est réalisée à l'aide d'une porte NOR, moins encombrante qu'une porte ET en logique CMOS (cf. chapitre 3).

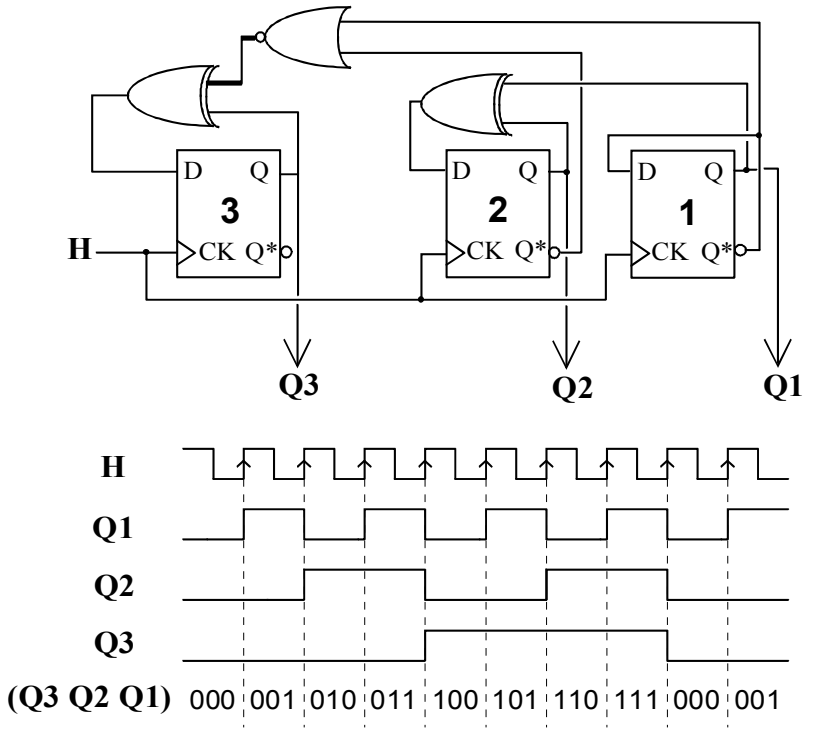


figure 5.35 : structure et chronogramme d'un compteur synchrone modulo 8

Le chronogramme obtenu est globalement le même que celui d'un compteur asynchrone, mais toutes les sorties du compteur sont directement obtenues à partir des fronts actifs d'horloge, si bien que, si les temps de propagation des bascules sont identiques, toutes les sorties du compteur commutent en même temps.

N.B. On peut montrer que pour un compteur binaire à cycle complet quelconque, la structure du compteur est donnée par les relations suivantes :

$$D_1 = \overline{Q_1}$$
$$D_i = (Q_1 \ldots Q_{i-1}) \oplus Q_i, \text{ pour } i > 1$$

5.3.1.2 *Exemple de synthèse de compteur binaire synchrone à cycle incomplet : compteur modulo* 5

Le principe de réalisation d'un tel compteur est la même que celle appliquée précédemment. La table de transition réduite du compteur binaire modulo 5 est la suivante :

Q_3	Q_2	Q_1	Q_3^+	Q_2^+	Q_1^+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0

tableau 5.8 : table de transition réduite d'un compteur binaire modulo 5

Le compteur est donc constitué de 3 bascules, tout comme le compteur modulo 8. La méthode d'obtention des expressions des entrées D des bascules en fonction de Q_1, Q_2, et Q_3 est identique. On pourra utiliser les états indifférents des tables de Karnaugh lors de la simplification.

- Expression de D_1

D_1	Q_1		$\overline{Q_1}$	
	Q_2	$\overline{Q_2}$	$\overline{Q_2}$	Q_2
Q_3	1	0	0	1
$\overline{Q_3}$	0	—	—	—

On obtient $D_1 = \overline{Q_1}\,\overline{Q_3}$, fonction que l'on réalisera préférentiellement sous la forme $D_1 = \overline{Q_1 + Q_3}$ à l'aide d'opérateurs CMOS.

- Expression de D_2

D_2	Q_1		$\overline{Q_1}$	
	Q_2	$\overline{Q_2}$	$\overline{Q_2}$	Q_2
Q_3	0	1	0	1
$\overline{Q_3}$	0	—	—	—

On obtient $D_2 = Q_1\overline{Q_2} + \overline{Q_1}Q_2 = Q_1 \oplus Q_2$.

- Expression de D_3

D_3	Q_1		$\overline{Q_1}$	
	Q_2	$\overline{Q_2}$	$\overline{Q_2}$	Q_2
Q_3	0	0	1	0
$\overline{Q_3}$	0	—	—	—

On obtient $D_3 = Q_1 Q_2$, que l'on réalisera préférentiellement sous la forme $D_3 = \overline{\overline{Q_1} + \overline{Q_2}}$ à l'aide d'opérateurs CMOS, puisque les sorties Q^* sont disponibles.

La structure du compteur correspondant est donnée en figure 5.36.

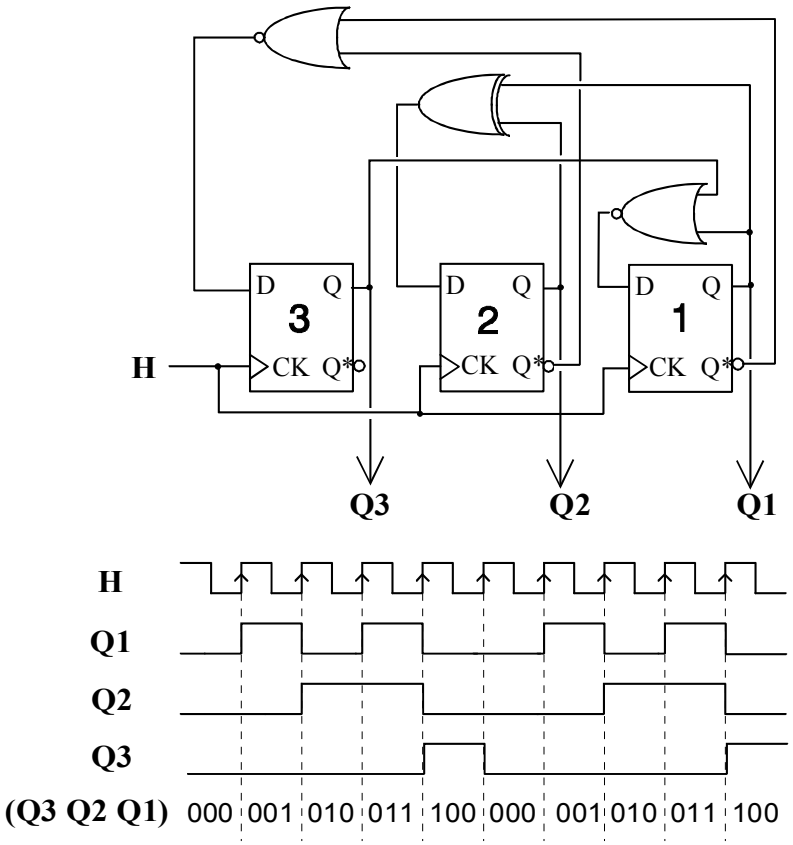


figure 5.36 : structure et chronogramme d'un compteur synchrone modulo 5

5.3.1.3 Exemple de synthèse de décompteur

Nous allons étudier la réalisation d'un décompteur synchrone, dont le cycle décompte de 6 à 1 ($6\rightarrow 5\rightarrow 4\rightarrow 3\rightarrow 2\rightarrow 1\rightarrow 6\rightarrow ...$). Nous nous bornerons à établir les expressions de D_1, D_2, et D_3.

La table de transition d'un tel décompteur est la suivante :

Q_3	Q_2	Q_1	Q_3^+	Q_2^+	Q_1^+
1	1	0	1	0	1
1	0	1	1	0	0
1	0	0	0	1	1
0	1	1	0	1	0
0	1	0	0	0	1
0	0	1	1	1	0

- $D_1 = \overline{Q_1}$
- Expression de D_2

Q_3 \ Q_1 Q_2		1	1	0
	1	0	—	0

$$D_2 = \overline{Q_1}\,\overline{Q_2} + Q_1\overline{Q_3}$$

- Expression de D_3

Q_3 \ Q_1Q_2	00	01	11	10
0	0	1	—	1
1	0	1	—	0

$$D_3 = \overline{Q_2}Q_1 + Q_2Q_3$$

5.3.1.4 Initialisation d'un compteur synchrone

Lors de l'initialisation du compteur par activation des entrées de remise à zéro ou à un dynamiques (à la mise sous tension du circuit notamment), il faut bien s'assurer que celui-ci est forcé dans un état appartenant au cycle de comptage. Dans le cas d'un compteur à cycle complet, il n'y a pas de problème. En revanche, si le cycle est tronqué, l'initialisation dans un état n'appartenant pas au cycle peut lancer le compteur dans un cycle erroné. Par exemple, pour l'initialisation du décompteur du 5.3.1.3, il ne faut surtout pas activer les entrées de mise à zéro des trois bascules, car l'état 000 ne fait pas partie du cycle de décomptage. Il faut utiliser la remise à zéro ou à un suivant la bascule.

5.3.2 Les compteurs programmables

Les fabricants de circuits intégrés proposent dans leurs catalogues de circuits standard des compteurs qui suivent le code binaire pur et qui possèdent en général les fonctionnalités suivantes :

- Chargement, en général synchrone, d'un état dans le compteur,
- Commande de validation/inhibition du comptage,
- Programmation du sens du comptage (comptage ou décomptage),

Ces circuits sont dits « compteurs programmables ». La figure 5.37 montre les principales entrées / sorties d'un tel compteur.

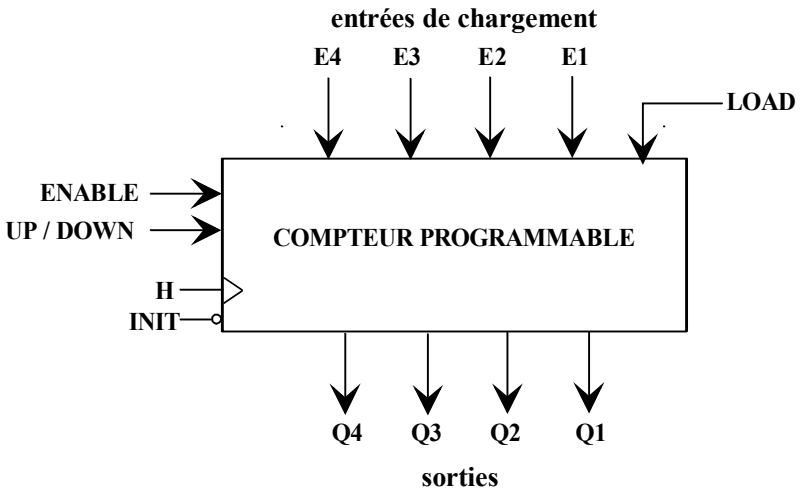


figure 5.37 : vue externe d'un compteur programmable sur 4 bits

La figure 5.38 montre un exemple de réalisation d'une cellule d'un tel compteur, avec les entrées de commande suivantes :

- *LOAD* = 1, mode chargement parallèle,
- *LOAD* = 0, mode comptage
  - *ENABLE* = 0, comptage inhibé,
  - *ENABLE* = 1, comptage validé,
    - $\Rightarrow$ *UP*/*DOWN* = 1, mode comptage,
    - $\Rightarrow$ *UP*/*DOWN* = 0, mode décomptage.

Le chargement synchrone à l'aide des entrées *LOAD* et E_i est similaire au chargement parallèle synchrone d'un registre à décalage (cf. 4.3.3). Parfois, certains compteurs possèdent également une entrée de remise à zéro synchrone, qui permet le chargement synchrone direct de l'état 0...0, sans utiliser les entrées E_i .

L'entrée *ENABLE* valide le comptage : si elle est active, le comptage se déroule normalement, sinon il y a rebouclage de la sortie *Q* de chaque bascule sur son entrée *D*.

Pour un compteur / décompteur, les deux fonctionnalités sont implantées dans le circuit, et l'entrée de programmation *UP*/*DOWN* permet de sélectionner la fonction logique correspondante à appliquer sur l'entrée de chaque bascule.

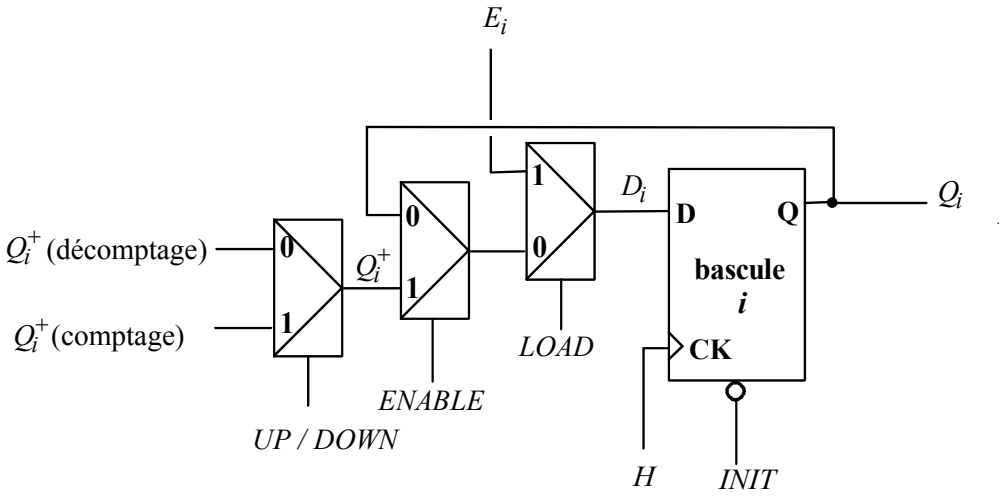


figure 5.38 : détail d'une cellule de compteur programmable

En général, les circuits programmables possèdent également une sortie de détection de l'état final du comptage (tous Q_i à 1).

5.4 Applications des compteurs

Les compteurs sont très couramment utilisés en logique séquentielle. Leurs applications les plus courantes sont les suivantes :

- Comptage d'événements (comptage d'impulsions sur une entrée par exemple).
- Division de fréquence : Dans un compteur binaire à cycle complet, la bascule de rang n produit en sortie un signal de fréquence $f / 2^n$, de rapport cyclique 1/2, où f est la fréquence de l'horloge du compteur. Pour obtenir des divisions de fréquence de rapports différents de 2^n, on peut utiliser un compteur à cycle incomplet et décoder un état particulier. Par exemple, dans le cas du compteur modulo 5, le décodage par exemple de l'état 000 (à l'aide d'une simple porte NOR à 3 entrées) permet de produire un signal passant à 1 pendant une période de l'horloge sur 5. C'est donc un signal de fréquence $f / 5$, mais de rapport cyclique 1/5 et non 1/2.
- Adressage de mémoires (cf. § 8.2) : par exemple pour la réalisation d'un séquenceur (cf. chapitre 6, section 3).

6. Paramètres dynamiques et règles d'assemblage des opérateurs séquentiels

6.1 Chemin critique et fréquence maximale de fonctionnement d'un circuit synchrone

6.1.1 Définition

Dans un circuit séquentiel synchrone, l'état des éléments séquentiels est remis à jour à chaque front actif de l'horloge. Pour étudier les retards liés à la propagation des valeurs logiques dans le circuit, il faut donc s'intéresser à ce qui se passe pendant chaque période du signal d'horloge. Les chemins de propagation considérés ne sont alors pas les chemins allant des entrées du circuit jusqu'à ses sorties, comme pour une fonction combinatoire, mais les chemins de propagation entre deux opérateurs de mémorisation, deux bascules par exemple (figure 5.39). On définit alors le **chemin critique** d'un circuit séquentiel comme étant le chemin de propagation qui limite sa fréquence de fonctionnement.

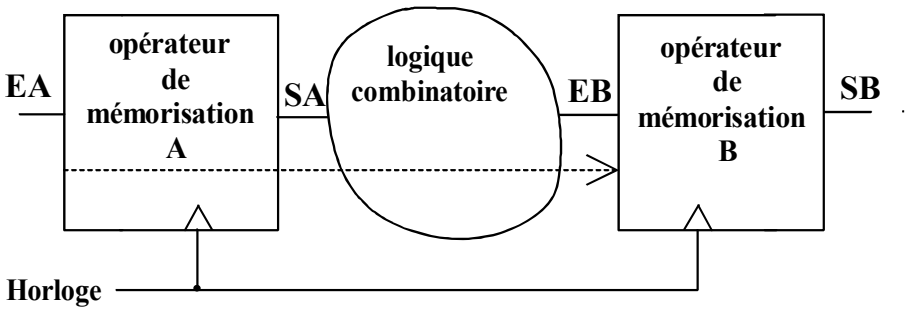


figure 5.39 : chemin de propagation dans un circuit séquentiel synchrone

La figure 5.40 permet d'analyser les différentes contraintes temporelles à prendre en compte sur le chemin de propagation de la figure 5.39. Pendant une période *T* de l'horloge, il faut prendre en compte :

- les temps de propagation de l'élément de mémorisation A (bascule) et de la logique combinatoire,
- le temps de maintien de l'entrée de A, *EA*, après le premier front actif de l'horloge,
- le temps de prépositionnement de l'entrée *EB* du second élément de mémorisation *B*, avant le front actif suivant de l'horloge.

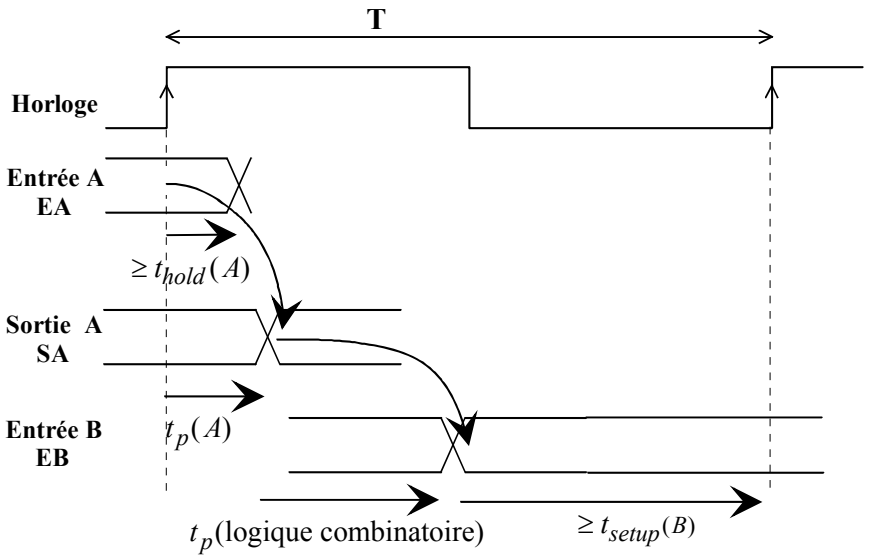


figure 5.40 : contraintes temporelles sur le chemin de propagation d'un circuit séquentiel synchrone

En pratique, les différents temps de propagation d'une bascule sont toujours supérieurs au temps de maintien, quelle que soit la charge de la bascule. La période *T* du signal d'horloge doit donc vérifier pour tous les chemins de propagation la relation suivante :

$$T \geq t_{p\max}(A) + t_{p\max}(\text{logique combinatoire}) + t_{setup}(B)$$

Le chemin critique est le chemin de propagation « le plus long » du circuit, c'est-à-dire celui qui impose la contrainte la plus forte sur *T*.

N.B. Les 2 fonctions de mémorisation de la figure 5.39 ne sont pas, en pratique, nécessairement distinctes. Il peut s'agir de la même bascule (cas du rebouclage d'une sortie sur l'entrée).

6.1.2 Exemples de calcul de la fréquence maximale de fonctionnement d'un circuit séquentiel

6.1.2.1 Registre à décalage

Considérons un registre à décalage simple réalisant la fonction de décalage à droite (cf. § 4.3.1). Tous les chemins de propagation sont identiques, il n'y a pas de logique aléatoire entre les bascules. La fréquence maximale de l'horloge est donnée par $f_{\max} = 1 / T_{\min}$, où $T_{\min} = t_{p\max}(\text{bascule D}) + t_{setup}(\text{bascule D})$.

6.1.2.2 Compteur modulo 8

Considérons le compteur modulo 8 synthétisé au § 5.3.1.1 et représenté en figure 5.35. Six chemins de propagation sont à considérer :

1. Rebouclage sur la bascule 1, passant par Q_1^*,
2. Rebouclage sur la bascule 2, passant par Q_2,
3. Rebouclage sur la bascule 3, passant par Q_3,

4. Bascule 1 → bascule 2, passant par Q_1,

5. Bascule 1 → bascule 3, passant par Q_1^*,

6. Bascule 2 → bascule 3, passant par Q_2^*.

Le chemin 1 ne comporte pas de logique combinatoire, les chemins 2, 3, et 4 traversent chacun 1 porte logique, et les chemins 5 et 6 en traversent 2. D'autre part, si les 3 bascules sont identiques, le temps de propagation du chemin 5 est le plus grand, car la sortance de Q_1 est supérieure à la sortance de Q_2^* (à cause du rebouclage sur D_1). Le chemin critique est donc le chemin 5 et $f_{\max} = 1 / T_{\min}$, où $T_{\min} = t_{p\max}(\text{bascule 1}) + t_{p\max}(\text{NON OU}) + t_{p\max}(\text{OUEX}) + t_{setup}(\text{bascule 3})$.

6.2 Règles d'assemblage séquentiel et aléas de fonctionnement

Nous donnons dans cette section quelques règles générales à respecter lors de la synthèse de systèmes séquentiels synchrones. Elles sont destinées à éviter de manière systématique les aléas de fonctionnement.

6.2.1 Initialisation

Lors de la conception de systèmes/circuits séquentiels, il faut toujours prévoir une commande d'initialisation des circuits. Cette commande est connectée aux entrées dynamiques de remise à zéro ou à un des bascules et permet de les forcer dans un état connu, à la mise sous-tension du circuit notamment.

Pour certaines applications, la présence d'une commande d'initialisation est facultative, comme ce peut être le cas pour un registre à décalage utilisé en ligne à retard numérique. En revanche, pour d'autres applications, elle est obligatoire, en particulier pour des circuits contenant des rebouclages : générateur pseudo-aléatoire, compteurs. Pour de telles fonctions, il est nécessaire de pouvoir à un moment donné forcer l'état des bascules.

6.2.2 Horloge

6.2.2.1 Décalage d'horloge

Si deux fonctions séquentielles réagissant l'une sur l'autre sont commandées par la même horloge, il ne faut pas insérer de retard sur les signaux d'horloges. La figure 5.41 montre un exemple d'aléa de fonctionnement lié à un décalage d'horloge.

Le concepteur de cette fonction a voulu réaliser un registre à décalage fonctionnant sur front descendant de l'horloge H_1 et utilisant deux bascules différentes. L'horloge de la bascule 2, H_2, est obtenue à partir de H_1 par inversion. Le rôle de la bascule 2 consiste à copier la valeur de Q_1 sur Q_2 aux fronts montants de H_2. On peut observer sur le chronogramme de la figure 5.41 que le fonctionnement de la bascule 2 va dépendre du rapport entre les temps de propagation de l'inverseur et de la bascule 1. Suivant que $t_p(\text{inv}) < t_p(\text{bascule 1})$ ou le contraire, la valeur stockée dans la bascule 2 est différente. D'autre part, même si $t_p(\text{inv}) < t_p(\text{bascule 1})$, il y a de fortes chances pour que le

temps de prépositionnement en entrée de la bascule 2 ne soient pas respecté. Ce montage a donc toutes les chances de ne pas fonctionner correctement en raison du décalage d'horloge.

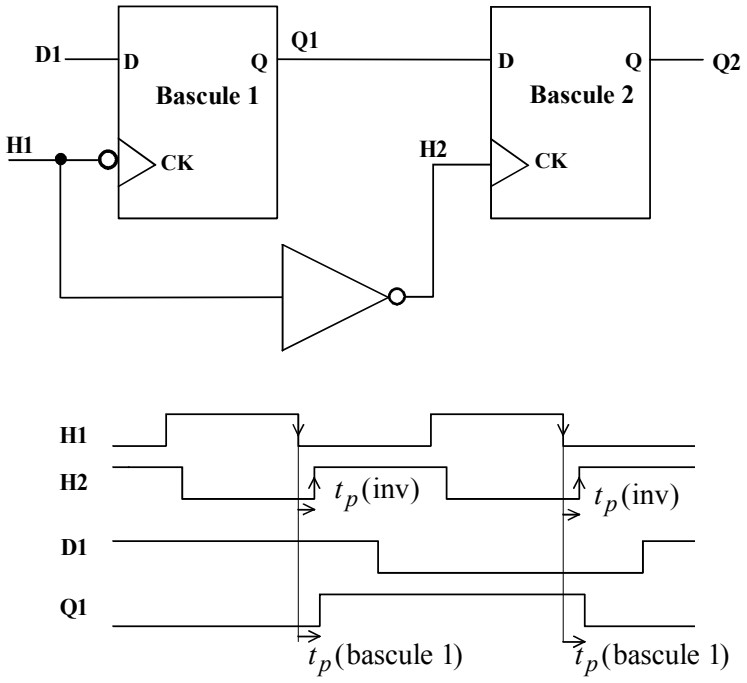


figure 5.41 : aléa de fonctionnement dû à un décalage d'horloge

6.2.2.2 Intégrité du signal d'horloge

Il faut toujours s'assurer que les signaux d'horloge ne présentent pas d'états parasites. En particulier, il faut éviter de construire un signal d'horloge à partir de logique combinatoire. Il n'est, en effet, pas rare d'observer à la sortie d'une fonction combinatoire des états parasites transitoires, liés à la présence de plusieurs chemins de propagation induisant des retards différents. Par conséquent, des fronts parasites peuvent apparaître en sortie d'une telle fonction. Un tel signal appliqué sur l'entrée d'horloge d'une bascule provoquerait un comportement erroné.

6.2.3 Entrées statiques / entrées dynamiques

Les entrées dynamiques des circuits séquentiels autres que l'horloge ne doivent être utilisés que pour son initialisation. Les autres entrées, de commande ou de données, doivent être statiques. La figure 5.42 montre un exemple de comportement erroné lié à l'utilisation d'une commande dynamique asynchrone.

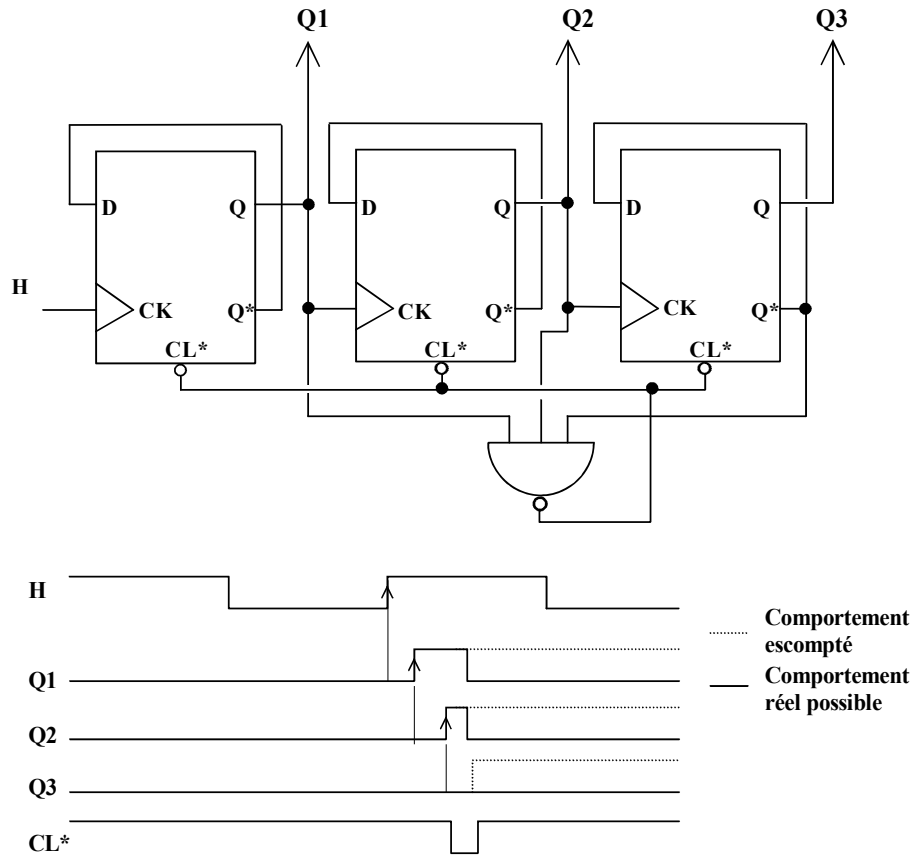


figure 5.42 : aléa de fonctionnement dû à une commande asynchrone

L'intention du concepteur était de réaliser un décompteur asynchrone à cycle incomplet : $7 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 0 \rightarrow 7 \rightarrow ...$ Lorsque l'état 3 ($Q_3 Q_2 Q_1 = 011$) est détecté, le décompteur est remis à zéro. En pratique, le cycle de fonctionnement du circuit n'est pas celui escompté : il a été vu, à la section 5.2.1, que des états parasites apparaissent nécessairement sur les sorties Q_2 et Q_3 en raison du cumul des retards sur les sorties. Or, dans ce cas de figure, l'état 3 apparaît de manière transitoire en sortie du décompteur entre les états 0 et 7. Dans ces conditions, l'entrée de remise à zéro CL^* des bascules est activée, et finalement, le décompteur ne peut pas quitter l'état 0, si CL^* est activée suffisamment longtemps.

En utilisant un décompteur synchrone, le problème du décalage cumulé des sorties disparaît. Mais, même dans ce cas, l'utilisation d'une telle commande asynchrone n'est pas fiable : il suffit que les bascules présentent des temps de réaction à CL^* très différents les uns des autres (si les sorties des bascules n'ont pas la même sortance, par exemple), et la commande asynchrone peut forcer le décompteur dans un état autre que celui prévu. Ce type de commande est donc à proscrire !

Dans le cas où un circuit séquentiel est commandé par des signaux extérieurs, généralement sans lien avec l'horloge du circuit, c'est-à-dire asynchrones, il faut **synchroniser** ces signaux avant leur utilisation. La figure 5.43 montre la synchronisation d'une entrée par une bascule D flip-flop.

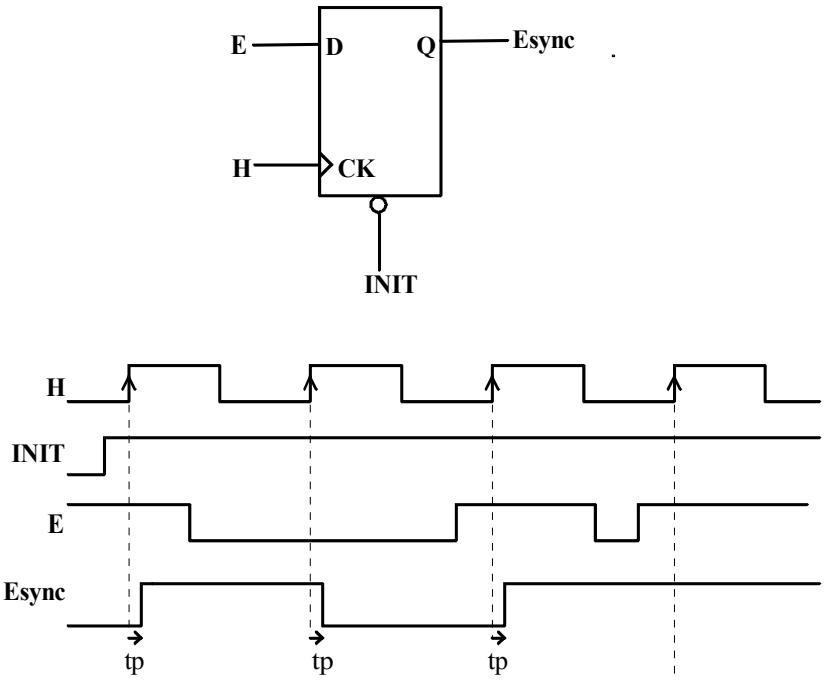


figure 5.43 : synchronisation d'un signal à l'aide d'une bascule D

7. Les mémoires à semi-conducteur

7.1 Introduction

Une mémoire est un dispositif permettant de stocker puis de restituer une information. Un tel dispositif de stockage a été étudié avec le registre tampon (cf. § 4.2). Néanmoins, cette solution à base de registres devient encombrante dès lors que la quantité d'information à stocker devient importante (à partir de quelques centaines de bits). C'est la raison pour laquelle il existe des fonctions spécialement dédiées à la mémorisation de grandes quantités d'information, ce sont les mémoires intégrées à semi-conducteur.

Elles sont apparues dans les années 70. Elles se sont alors rapidement substituées aux mémoires à tore de ferrite et à tambour magnétique dans l'équipement de la mémoire centrale des ordinateurs, en raison de leurs multiples avantages : absence de pièces mécaniques, emploi aisé dû au conditionnement sous forme de circuits intégrés, et grande rapidité d'accès aux informations.

Les grandeurs permettant de caractériser une mémoire sont :

- **Sa capacité** : C'est la quantité d'information qui peut être stockée dans la mémoire. Elle s'exprime en bits, ou en mots de n bits, si l'information est stockée sous la forme de mots binaires. L'**octet** (mot de 8 bits) est couramment utilisé dans les systèmes à microprocesseurs.

- **Son organisation** : Les mémoires sont en général structurées sous la forme d'un tableau de mots binaires. L'organisation de la mémoire désigne les dimensions de ce tableau : « l lignes et c colonnes » (sa capacité est alors égale à $l \times c \times n$ bits, dans le cas du stockage de mots de n bits).

- **Ses modes de fonctionnement** : Les deux opérations possibles sur une mémoire sont l'**écriture** d'informations dans la mémoire et la **lecture** de ces informations. On distingue de ce point de vue deux grandes catégories de mémoires à semi-conducteur :

  - Les **mémoires vives** dont le contenu est accessible aussi bien en lecture qu'en écriture,

  - Les **mémoires mortes**, qui ne sont accessibles qu'en lecture lors de leur utilisation dans un système numérique, et pour lesquelles l'opération d'écriture représente un événement exceptionnel et est plutôt appelée **programmation**.

- **Son type d'accès** : Il n'est évidemment pas réaliste d'envisager une connexion externe pour chaque mot mémorisé. Il faut alors recourir à une technique de multiplexage pour minimiser la connectique. Le principe du multiplexage donne lieu à deux types d'implémentations :

  - Multiplexage spatial : A chaque cellule mémoire est associée une **adresse**. Toutes les cellules sont alors connectées à un dispositif de décodage (le **décodeur d'adresses**) qui permet de sélectionner les mots en fonction de l'adresse appliquée sur son entrée. La mémoire est alors dite à **accès aléatoire** ou **direct**.

  - Multiplexage temporel : Le principe consiste à relier les cellules mémoires de façon à former une chaîne, comme dans un registre à décalage. L'accès aux informations n'est

alors possible qu'aux extrémités de la mémoire. On parle alors de mémoire à **accès séquentiel**.

Un paramètre important permettant de caractériser les performances dynamiques d'une mémoire est le **temps d'accès**, c'est-à-dire le temps écoulé entre une demande de lecture et la présence de l'information en sortie de la mémoire. Le temps d'accès est fortement lié au type d'accès. En effet, dans le cas d'un accès direct, le temps d'accès est le même pour toutes les cellules mémoires. En revanche, l'accès séquentiel n'offre pas la même souplesse d'utilisation : pour accéder à une information, il faut la faire transiter jusqu'à une extrémité de la mémoire. Le temps d'accès est donc lié à la position du mot dans la chaîne.

De ce fait, on constate que les mémoires à accès aléatoire sont employées dans une grande majorité des applications. Les mémoires à accès séquentiel sont cependant utilisées pour des applications bien spécifiques : mémoires de files d'attente FIFO (First In First Out), mémoires de piles LIFO (Last In First Out). Certaines mémoires, comme les mémoires vidéo sur les cartes graphiques, allient les deux types d'accès : écriture des pixels par adressage (accès aléatoire en écriture), et envoi des informations vers le dispositif de visualisation (lecture) sous forme de trames, dans l'ordre croissant des adresses (accès séquentiel en lecture).

Pour un type d'accès donné, le temps d'accès est également lié à la capacité et à l'organisation de la mémoire.

N.B. Pour certains types de mémoires (DRAM notamment), la notion de temps d'accès n'est pas suffisante pour caractériser les performances en vitesse de la mémoire. On définit alors la notion de **temps de cycle** (cf. section 7.2.2.3, page 54).

7.2 Les mémoires à accès aléatoire

7.2.1 Structure

Dans une mémoire à accès aléatoire, chaque mot est accessible directement par une adresse, et en un temps constant quelle que soit l'adresse sélectionnée. La structure générale d'une telle mémoire est représentée sur la figure 5.44.

L'adresse de lecture ou d'écriture A est codée sur un bus de p bits. Le décodeur d'adresses permet de sélectionner un mot de la mémoire parmi au plus 2^p en fonction de la valeur de A. En général, le décodage est en fait réalisé à l'aide de deux décodeurs tirant parti de l'organisation de la mémoire : le premier sélectionne une ligne de la mémoire (décodeur ligne), et le second sélectionne une colonne (décodeur colonne). Ceci permet de réduire la complexité de la circuiterie de décodage.

Le bus de données *DO* (Data Out) permet de lire le mot sélectionné par l'adresse A en mode lecture, et le bus de données *DI* (Data In) porte le mot à écrire à l'adresse A en mode écriture. Dans le cas d'une mémoire morte, seule l'opération de lecture est possible, et le bus *DI* est absent. Dans le cas d'une mémoire vive, les entrées *DI* et les sorties *DO* peuvent être regroupées en n accès bidirectionnels, utilisant des opérateurs 3 états. Certaines mémoires vives, dites à **double port**, possèdent deux jeux de bus (adresses + données) distincts permettant deux opérations simultanées (2 lectures, 2 écritures, ou 1 lecture et 1 écriture).

La logique de contrôle reçoit les commandes de sélection de la mémoire, d'écriture/lecture, et gère l'écriture et la lecture des mots de la mémoire. Toute mémoire à accès aléatoire possède au moins les entrées de commande suivantes:

- Une entrée *R* / *W* de sélection de la lecture ou de l'écriture pour les mémoires vives,
- Une entrée *CS* (Chip Select) de sélection du circuit permettant d'inhiber (opérateurs d'entrée et de sortie en haute impédance) ou de valider une opération de lecture ou d'écriture (lecture pour une mémoire morte). Cette commande est utilisée lorsque plusieurs boîtiers mémoires sont placés en parallèle.

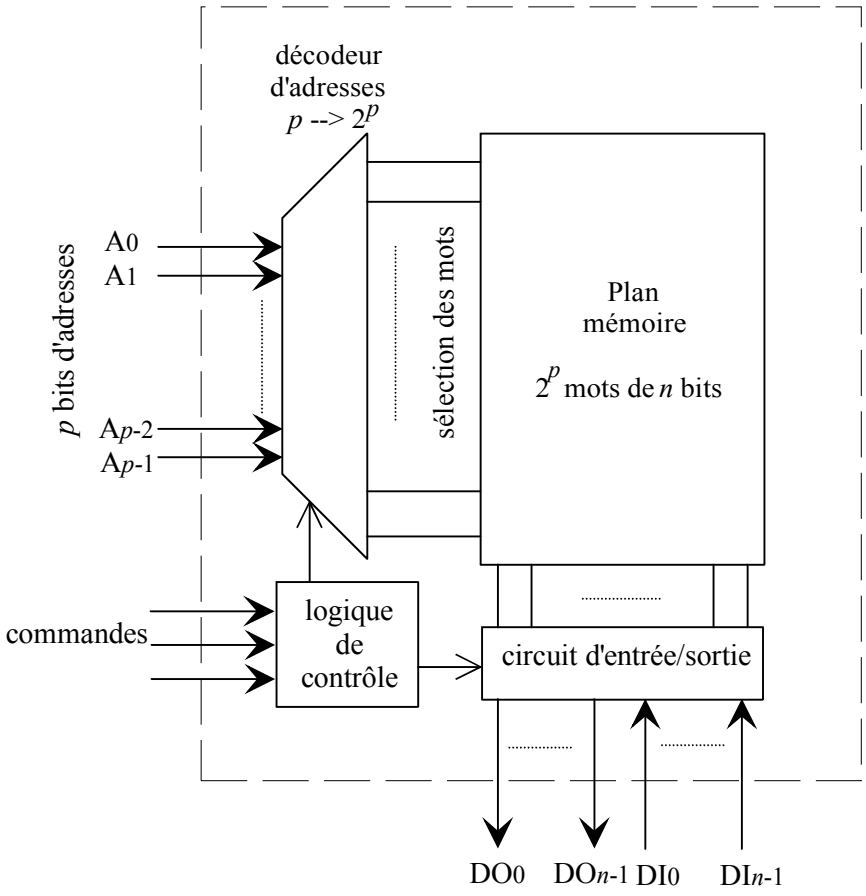


figure 5.44 : structure d'une mémoire à accès aléatoire

7.2.2 Les mémoires vives ou RAM

Le terme **RAM**, qui signifie « mémoire à accès aléatoire » (Random Access Memory) désigne en fait, et à tort, les mémoires vives à accès aléatoires. Ces mémoires sont utilisées dans les mémoires centrales des ordinateurs, les automates programmables.

Suivant la structure de la cellule mémoire, il existe plusieurs types de mémoires RAM. On distingue principalement :

- Les mémoires **statiques**, dont la cellule de base est constituée d'un bistable (cf. § 3.1), et qui conservent l'information tant que le circuit reste sous tension,
- Les mémoires **dynamiques**, dont la cellule de base est constituée par un condensateur.

Les RAM statiques et dynamiques sont des mémoires dites **volatiles** car leur contenu disparaît lorsque l'alimentation est coupée. Il existe cependant des RAM non volatiles (NVRAM) munies d'une source d'énergie, en général constituée de piles au lithium incorporées au boîtier du circuit.

7.2.2.1 Les RAM statiques

La RAM statique ou **SRAM** (Static Random Access Memory) permet de stocker de l'information de façon permanente quand l'alimentation est maintenue. L'élément de base est une bascule bistable avec des accès pour la lecture et l'écriture. La figure 5.45 montre la structure d'une telle cellule.

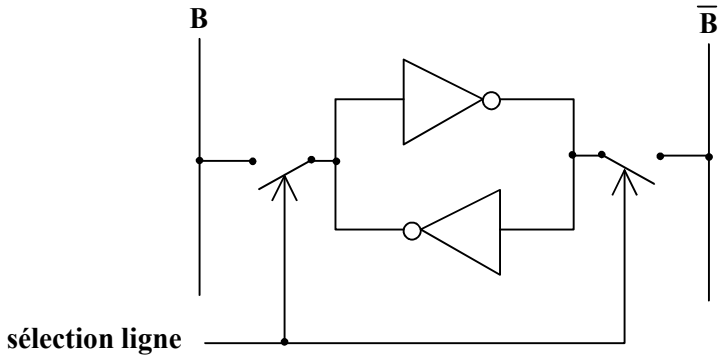


figure 5.45 : structure générale d'une cellule SRAM

L'écriture et la lecture sont autorisées lorsque la ligne sur laquelle se situe la cellule mémoire est sélectionnée. La cellule pratique CMOS est donnée par la figure 5.46.

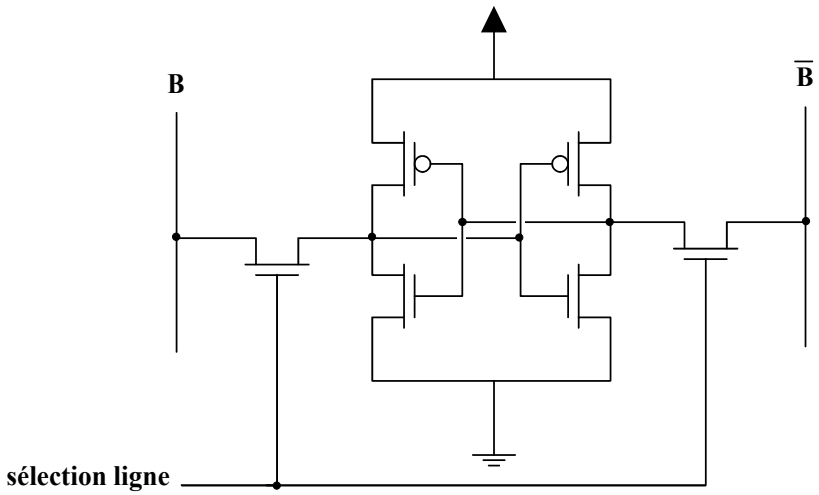


figure 5.46 : cellule SRAM en logique CMOS

Pour l'écriture d'une valeur logique dans une cellule mémoire, deux connexions sont utilisées en pratique, une pour chaque accès du point mémoire : l'application d'un 0 logique sur *B* permet d'écrire

un 0 dans la cellule et l'application d'un 0 sur $\overline{B}$ permet d'écrire un 1. Pour la lecture, les connexions verticales B et $\overline{B}$ sont connectées à l'entrée d'un amplificateur différentiel, utilisé en comparateur, qui délivre un 1 logique si $B = 1$ et $\overline{B} = 0$, et un 0 logique dans le cas contraire. La structure d'une SRAM de 32 mots de 1 bit est donnée en figure 5.47. Le décodage des adresses (5 bits) se décompose en un adressage ligne sur 3 bits (L_0, L_1, L_2), et un adressage colonne sur 2 bits (C_0, C_1). Pour chaque adresse, une ligne et une colonne sont activées, et les interrupteurs MOS correspondants sont fermés. Par exemple sur la figure 5.47, lors de l'application de l'adresse $L_2L_1L_0C_1C_0 = 00111$, la deuxième ligne et la quatrième colonne sont activées. L'écriture d'un 0 ou d'un 1 dans la cellule située à l'intersection est obtenue par application d'un 1 logique sur la grille du transistor adéquat.

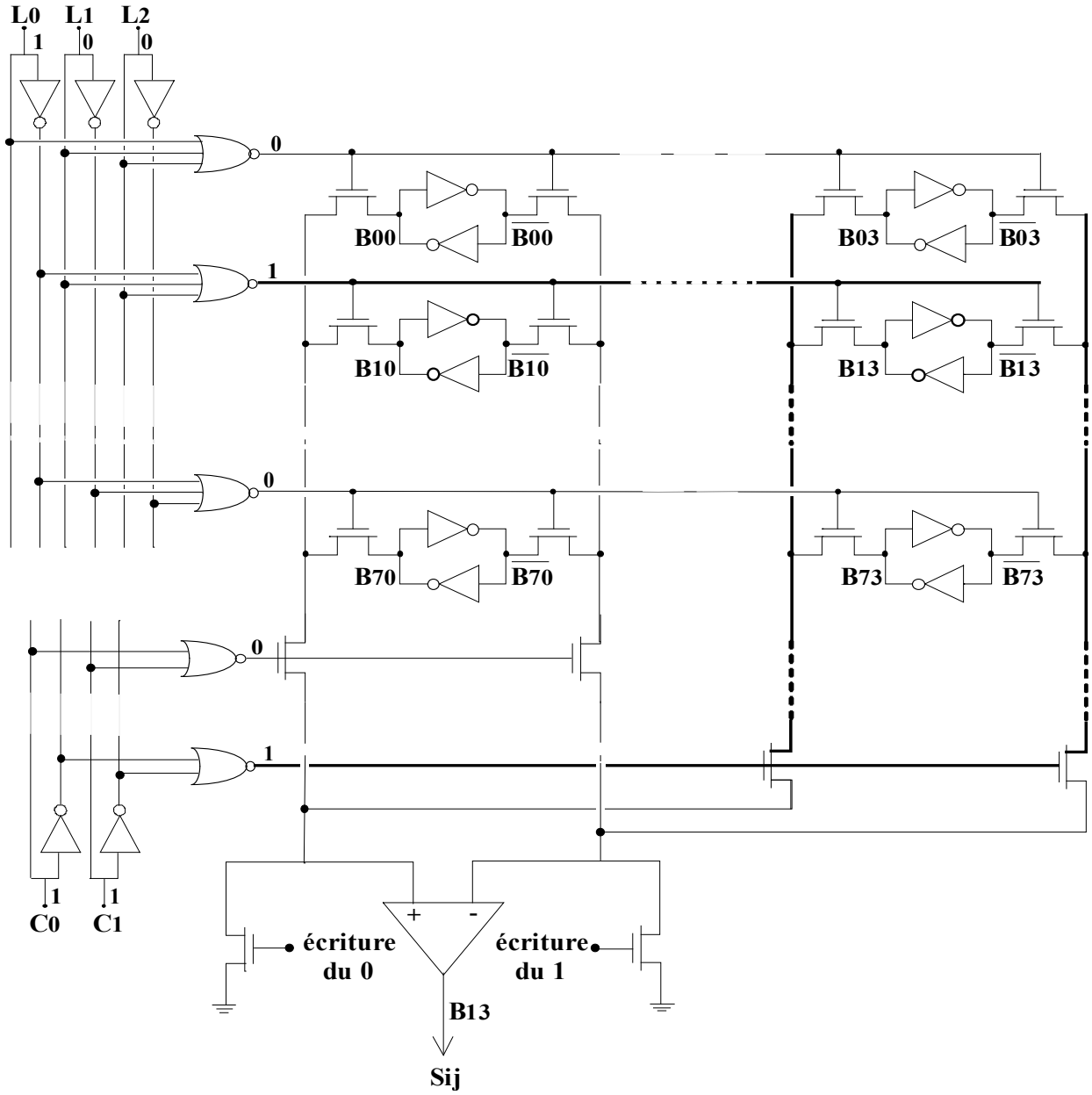


figure 5.47 : structure d'une SRAM de 32 bits

7.2.2.2 Les RAM dynamiques

Les mémoires statiques nécessitent un nombre élevé de transistors : 6 transistors par cellule de base. Le concept des mémoires dynamiques ou **DRAM** (Dynamic Random Access Memory) permet

d'augmenter notablement la densité d'intégration des mémoires vives. L'élément mémorisant n'est pas une bascule mais un condensateur de capacité C_s (figure 5.48) réalisé à l'aide d'un dépôt de polysilicium cristallin. L'écriture consiste à imposer un potentiel aux bornes de la capacité C_s lorsque le transistor de la cellule est passant.

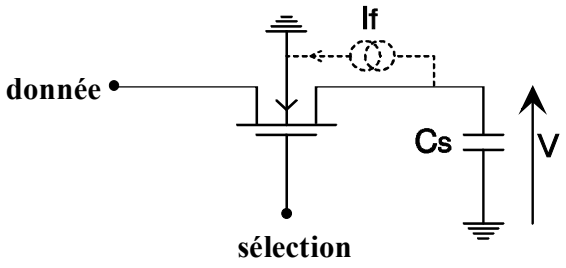


figure 5.48 : cellule d'une DRAM

L'inconvénient majeur de ce type de mémoire vient de la faible durée de rétention de la charge au niveau de la capacité C_s, en raison de l'existence d'un courant de fuite I_f entre la source du transistor et son substrat (courant inverse de jonction).

Les ordres de grandeur de C_s et I_f sont les suivants :

$C_s \approx 0{,}01$ pF

$I_f \approx 10^{-10}$ A

d'où $dV / dt = -I_f / C_s \approx -0{,}1$ V / ms.

Avec ces valeurs, l'information est donc disponible pendant une durée de l'ordre de la dizaine de millisecondes. En raison de cette courte durée de rétention, il faut intégrer dans la circuiterie de la mémoire un dispositif de **rafraîchissement** de la mémoire. Le contenu des cellules est régénéré à une fréquence de l'ordre de quelques centaines de Hz.

D'autre part, la lecture du contenu d'une cellule requiert des organes de contrôle supplémentaires :

- Les niveaux récupérés en lecture sont très faibles à cause de la capacité parasite de la ligne de lecture, beaucoup plus élevée que C_s (rapport 50, typiquement). Par conséquent, la lecture nécessite la présence d'amplificateurs en sortie.
- L'opération de lecture détruit la valeur mémorisée, il faut donc prévoir un dispositif de réécriture des données lues.

C'est grâce à ce type de mémoire de très grande capacité que peuvent progresser les technologies MOS, en particulier en ce qui concerne les réductions de géométrie des transistors et l'augmentation du nombre de couches métalliques. En effet, la structure très répétitive des mémoires dynamiques permet l'optimisation d'une technologie sur les critères de vitesse et de densité d'intégration.

7.2.2.3 Critères de choix SRAM / DRAM

Les RAM statiques sont beaucoup plus encombrantes que les RAM dynamiques. Ces dernières permettent ainsi une intégration beaucoup plus dense, et donc une très grande capacité de mémorisation à un moindre coût par bit.

En contre-partie, les organes de contrôle d'une DRAM sont beaucoup plus complexes que ceux d'une SRAM : dispositifs de rafraîchissement, amplificateurs de lecture, etc... Leur utilisation est

beaucoup plus délicate que celle des SRAM, les contraintes dynamiques étant nombreuses. Notamment, étant donné leurs grandes capacités, les constructeurs ont été souvent amenés à multiplexer l'accès aux adresses, pour réduire le nombre de broches des circuits. L'utilisateur doit alors présenter séquentiellement sur les mêmes fils l'adresse ligne puis l'adresse colonne. On définit alors pour les mémoires dynamiques le **temps de cycle** de la mémoire, qui représente la durée minimale à respecter entre deux accès consécutifs. D'autre part, l'accès à la mémoire est bloqué pendant le rafraîchissement. En conséquence, dans le cas d'application à fortes contraintes temporelles, on utilise de préférence des SRAM (ex : mémoire cache).

Pour des applications imposant des hautes performances en vitesse, il existe des RAM dites **synchrones** [Pri96], incluant une interface synchrone chargée de gérer les opérations d'écriture et lecture. En entrée et sortie de la mémoire, des latches commandées par l'horloge du système permettent de stocker temporairement les adresses, les signaux de contrôle et les données. L'interface synchrone est gérée de telle sorte qu'une opération d'écriture ou de lecture peut être exécutée par cycle d'horloge. Cependant, en raison du temps d'accès de la mémoire, les données ne sont disponibles en sortie qu'après un nombre fixé de cycles d'horloge (latence).

Le tableau suivant donne un aperçu du standard du marché des RAM en 2008 (RAM les plus couramment utilisées par les équimentiers) :

	capacité	temps d'accès ou temps de cycle typique
SRAM asynchrone CMOS 1,8-3,3V	256 kbit à 36 Mbit	$t_{acc} \approx 10$ ns
SRAM synchrone CMOS 1,8-3,3 V	2 Mbit à 72 Mbit	$t_{cy} \approx 0.5$ à 4 ns
DRAM synchrone CMOS 1,8-3,3V	64 Mbit à 8 Gbit	$t_{cy} \approx 2$ à 10 ns

7.2.3 Les mémoires mortes ou ROM

Les mémoires mortes ou ROM (Read-Only Memories) sont utilisées pour mémoriser des informations ne devant pas être modifiées, par exemple pour les mémoires-programmes des séquenceurs microprogrammés ou des processeurs. Le contenu d'une mémoire morte est non volatil, et les informations inscrites sont disponibles en permanence.

Le procédé d'inscription peut être inaccessible à l'utilisateur, c'est le cas des mémoires ROM proprement dites dont l'inscription est effectuée par une opération technologique à la fabrication, ou accessible d'une façon irréversible (ROM programmables) ou réversible (ROM reprogrammable).

7.2.3.1 Les mémoires ROM et ROM programmables (PROM)

La cellule mémoire est constituée d'un transistor. La figure 5.49 donne la structure d'une ROM NMOS dont le contenu est représenté par la matrice $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$. Le dispositif de décodage des adresses n'est pas représenté sur cette figure.

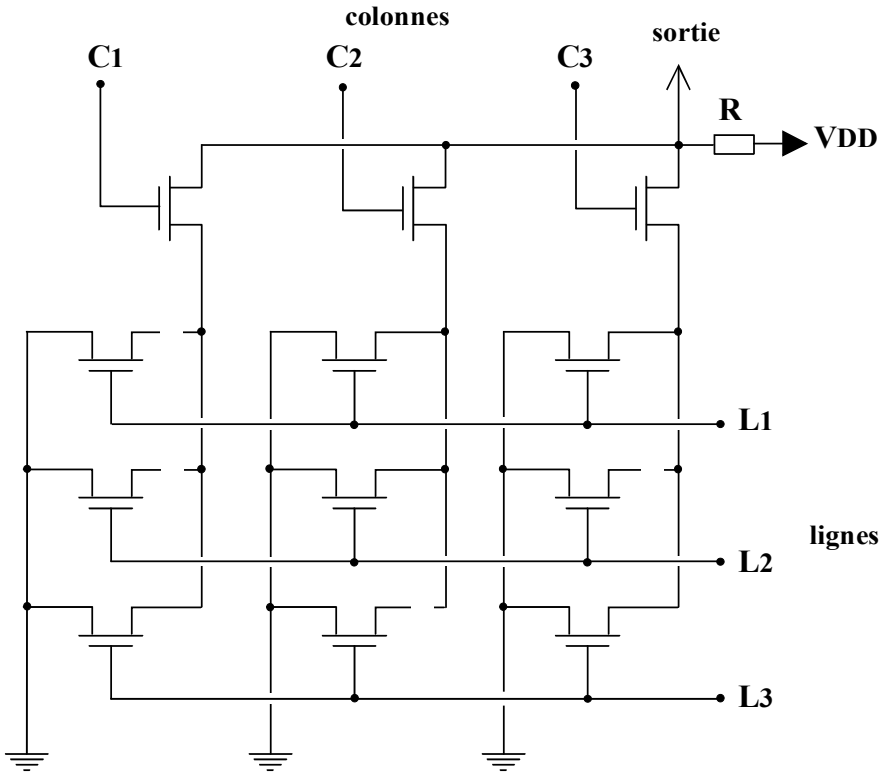


figure 5.49 : structure d'une ROM NMOS de 3 × 3 bits

On observe la présence d'un transistor à l'intersection de chaque ligne et chaque colonne. Suivant la valeur inscrite dans la cellule considérée, 0 ou 1, le transistor est connecté (inscription d'un 0) ou non (inscription d'un 1) à sa colonne de sélection. Suivant la technique de réalisation de la mémoire, les transistors sont « déconnectés » suivant des procédés différents : dans le cas d'une ROM (non programmable), le constructeur dote les transistors à « déconnecter » d'une couche d'oxydation supplémentaire qui augmente considérablement leur tension de seuil et les bloque de manière permanente. Dans le cas des ROM programmables, chaque transistor est relié à la colonne de sélection

par un fusible (figure 5.50), qui peut être détruit en faisant passer un courant fort dans le transistor correspondant.

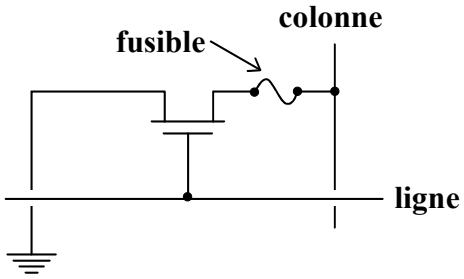


figure 5.50 : structure d'une cellule de ROM programmable

Lorsque la colonne *i* et la ligne *j* sont sélectionnées, si le transistor sélectionné est valide, il est passant et impose un 0 logique sur la sortie, sinon, il est bloqué et la sortie est ramenée à 1.

Les ROM et les PROM réalisées en logique CMOS mettent rarement en œuvre un réseau dual de transistors PMOS. En pratique, l'étage P est limité à un MOS correctement polarisé, ce qui ramène au cas de la matrice NMOS étudiée précédemment.

N. B. La fabrication de masques spécifiques pour réaliser une ROM est une opération lourde et coûteuse, qui n'est justifiée que pour la fabrication de grandes séries (au moins 100 000 pièces).

7.2.3.2 Les mémoires reprogrammables REPROM

Le caractère irréversible de la destruction des fusibles dans les PROM reste un obstacle à la mise au point de prototypes. C'est la raison pour laquelle les ROM reprogrammables ou REPROM connaissent un grand succès. Elles sont principalement de trois types :

- Les **EPROM** (Erasable PROM), dont la programmation électrique peut être effacée par une exposition du circuit à un rayonnement ultraviolet,
- Les **EEPROM** (Electrically Erasable PROM), qui sont programmables et effaçables électriquement,
- Les mémoires **flash** sont les mémoires programmables les plus récentes (début des années 90). Elles sont également programmables et effaçables électriquement, mais beaucoup plus denses que les EEPROM.

L'organisation des EPROM est identique à celle des PROM, les transistors MOS et les fusibles étant remplacés par des transistors à grille flottante ou FAMOS (Floating gate Avalanche injection MOS), qui peuvent être rendus passants par l'application d'une impulsion de tension, puis être de nouveau rendus isolants lorsqu'il sont exposés un certain temps à des U.V. (entre 5 et 30 mn).

Une limitation importante des EPROM est leur effacement par U.V. qui n'est pas réalisable *in situ* (il faut ôter le circuit de la carte). D'autre part, une opération d'effacement entraîne la déprogrammation de la totalité de la mémoire. Les EEPROM sont des mémoires dont l'opération d'effacement est réalisée électriquement et peut être sélective et ne porter que sur un mot. Ces mémoires font appel à des technologies très spécifiques, dérivées de la technologie MOS (notamment transistors MNOS, Metal Nitride Oxide Semiconductor [PBOZ97]). Le principal inconvénient de ces composants est leur faible densité d'intégration par rapport aux EPROM. Les EEPROM sont de plus en plus remplacées par les mémoires flash.

Les mémoires flash connaissent un développement rapide depuis ces dernières années. Ce sont des dispositifs qui combinent la densité d'intégration des EPROM et les facilités d'effacement et de reprogrammation des EEPROM. Les points mémoires sont constitués de transistors dits « à grille flottante » [PBOZ97]. Les temps d'effacement et de programmation sont respectivement de l'ordre de quelques millisecondes et de quelques centaines de microsecondes. En 2008, le standard du marché des mémoires flash utilise typiquement des mémoires de 2 à 16 GOctets alimentées sous 2,5 Và 3,3V, avec des temps d'accès de25 ns.

7.3 Les mémoires à accès séquentiel

Les mémoires intégrées à accès séquentiel sont essentiellement des mémoires vives. On trouve néanmoins sur le marché des PROM dites « à lecture série » destinées au téléchargement de programmes (pour la programmation des circuits programmables de type FPGA par exemple). D'un point de vue interne, les mémoires à accès séquentiel utilisent le plus souvent les mêmes cellules mémoires que les mémoires à accès aléatoire (ROM, SRAM ou DRAM) et diffèrent principalement par leur mode d'adressage.

On distingue deux catégories de mémoires vives à accès séquentiel : les files d'attente ou FIFO (First In First Out), et les piles ou LIFO (Last In First Out). Dans une FIFO, les informations sont accessibles dans l'ordre où elles ont été écrites. Dans une LIFO, elles sont accessibles dans l'ordre inverse de leur écriture.

La figure 5.51 représente l'architecture d'une FIFO. Il s'agit en fait d'une RAM double-port dont un bus d'adresses est dédié à la lecture et l'autre à l'écriture, et dont l'adressage est géré par des compteurs. A chaque écriture ou lecture, le pointeur correspondant est incrémenté par comptage. Ce dispositif sert le plus souvent d'interface entre deux flots de données de fréquences différentes. La capacité de la FIFO, ainsi que les mécanismes prévenant les dépassements doivent être dimensionnés en fonction de l'application à traiter.

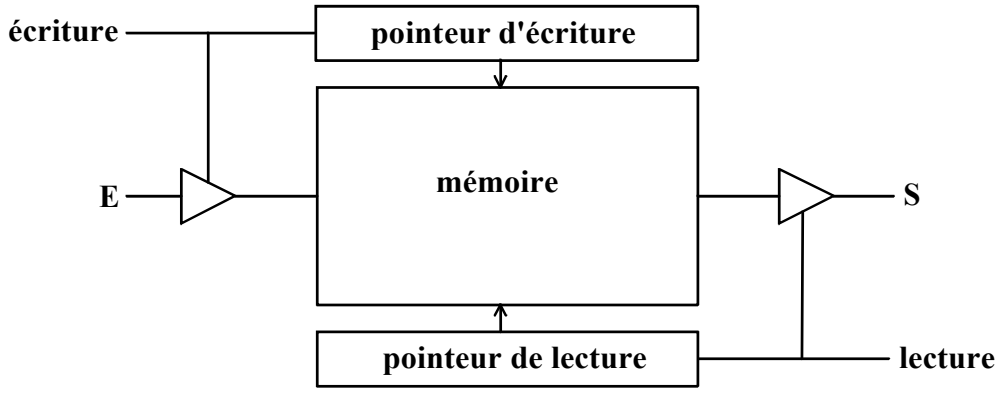


figure 5.51 : architecture d'une FIFO

Les LIFO ne comportent qu'un pointeur associé à un compteur / décompteur qui s'incrémente à chaque écriture et se décrémente à chaque lecture.

Pour certaines applications telles que le stockage momentané d'images (Vidéo RAM), on utilise des dispositifs dits « à transfert de charge » ou CCD (Charge Coupled Devices) qui sont à la frontière du numérique et de l'analogique car il s'agit d'éléments réalisés en technologie MOS capables de

décaler des quantités de charges variables en synchronisme avec deux ou trois signaux d'horloge décalés dans le temps.

8. Bibliographie

[CLDS86] M. Cand, J.-L. Lardy, E. Demoulin, et P. Senn, *Conception des circuits intégrés MOS, éléments de base, perspectives*, Collection Technique et Scientifique des Télécommunications, Eyrolles, Paris, 1986.

[GR87] M. Gindre et D. Roux, *Electronique numérique : logique séquentielle, cours et exercices*, McGraw-Hill, Paris, 1987.

[LSI89] LSI Logic, *Logic Design Manual fir ASICs*, LSI Logic, 1989.

[NB90] P. Naish et P. Bishop, *Conception des ASICs*, Masson, 1990.

[OM93] G. Ouvradou et G. Martineau, *Logique combinatoire et séquentielle*, polycopié Télécom Bretagne, 1993.

[Dan96] J. D. Daniels, *Digital Design from Zero to One*, Wiley & Sons Inc., 1996.

[Pri96] B. Prince, *High Performance Memories*, Wiley & Sons Inc., 1996.

[PBOZ97] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, « Flash memory cells - An overview », Proc. of the IEEE, vol. 85, n°8, august 1997.

Chapitre 6 : Fonctions et systèmes séquentiels complexes

1. Introduction

1.1 Définitions

Les systèmes séquentiels qui ont été décrits jusqu'ici peuvent être qualifiés de simples car leur structure est directement superposable au modèle général des systèmes séquentiels, l'automate à états finis (AEF) présenté au chapitre 5, § 2.1.

Néanmoins, lorsque l'on franchit un certain seuil de complexité dans la spécification comportementale d'un système séquentiel, ce modèle ne peut plus être directement retenu pour structurer sa réalisation. Il se prête mal à l'abstraction et à la hiérarchisation car les notions de **données** (objets subissant un traitement) et de **contrôle** (sujet acteur du traitement) y sont totalement intriquées.

Pour aborder les problèmes de conception de manière progressive et hiérarchique, il est nécessaire de partitionner le système en sous-ensembles dévolus soit au traitement des données, soit au pilotage ou séquencement de ces traitements (cf. figure 6.1). Les sous-ensembles du premier type constituent la **partie opérative** du système, les seconds, la **partie contrôle** (encore appelée **commande**, ou **supervision**). Ces deux types de sous-ensembles relèvent de techniques de réalisation qui leur sont propres.

La réalisation des sous-ensembles de la partie opérative est très dépendante de la fonctionnalité qu'ils abritent. On peut ainsi différencier :

- les unités de traitement (ex : opérateur arithmétique),
- les unités de mémorisation (ex : file d'attente),
- les unités d'interface (entrées/sorties du système),

Le comportement général d'une unité de contrôle consiste à positionner des signaux destinés au contrôle de la partie opérative en fonction d'un état interne et de l'état de signaux d'entrée provenant de la partie opérative ainsi que de l'extérieur du système. Le comportement d'une unité de contrôle répond donc strictement à la définition d'un système séquentiel simple tel qu'on l'a défini dans le chapitre 5. L'objet de ce chapitre est de présenter les possibilités de réalisation de telles unités. On exclut toutefois les solutions à base de microcontrôleurs car elles relèvent plutôt d'un cours sur les processeurs.

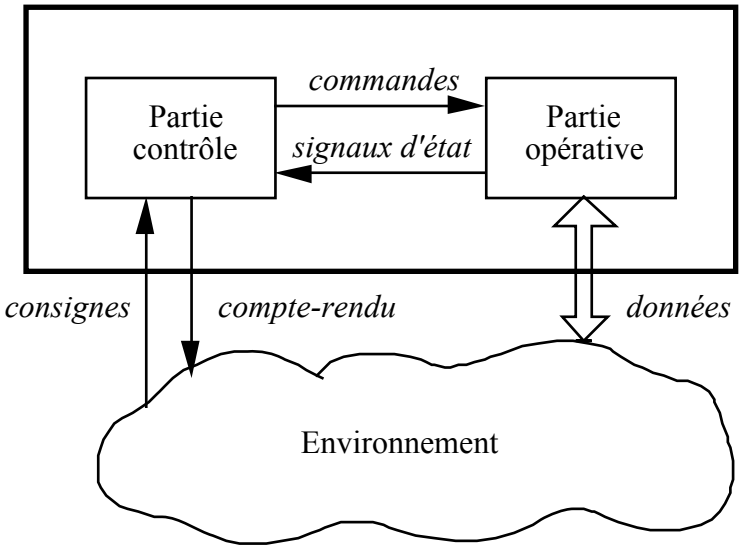


figure 6.1 : structure d'un système séquentiel complexe

1.2 Solutions architecturales étudiées pour la réalisation d'une unité de contrôle

On envisage deux approches distinctes dans ce chapitre :

- La première est basée sur la mise en œuvre directe de l'automate à états finis, appelée **machine à états finis**. On étudiera deux variantes de cette machine, l'une baptisée **machine de Mealy**, déjà présentée dans le chapitre 5, l'autre nommée **machine de Moore**, qui est une restriction de la précédente.
- La seconde approche fait appel au concept de **séquenceur**, utilisé dans les unités de contrôle des premières générations de processeurs. La complexité de l'unité de contrôle y est réduite au prix de restrictions comportementales. Nous étudierons deux variantes de mise en œuvre, le **séquenceur câblé** et le **séquenceur microprogrammé**.

2. Les machines à états finis

2.1 Systèmes synchrones versus asynchrones

On a vu, à travers les fonctions séquentielles étudiées dans le chapitre précédent (registre à décalage, compteur), que le recours à une base de temps commandant des bascules de type flip-flop (i.e. systèmes synchrones) permet de maîtriser plus aisément le comportement et l'exploitation du système. C'est, en général[1], la démarche adoptée pour réaliser des systèmes complexes. C'est pourquoi seules les unités de contrôles synchronisées sont envisagées par la suite. Notons toutefois que l'action de l'horloge pose un problème lorsque l'on modélise le comportement d'un système

[1] Il existe toutefois des systèmes complexes n'utilisant pas d'horloge. De tels systèmes sont dits autosynchronisés (self timed) ou insensibles aux délais (delay insensitive). Leur conception relève de techniques bien particulières qui n'ont pas lieu d'être abordées dans ce bloc.

(graphe d'états) car cette action est en général implicite. Il faut donc jamais perdre de vue que l'on doit raisonner dans un espace temporel discret.

2.2 Machine de Mealy versus machine de Moore

La structure de la machine de Mealy synchrone est rappelée en figure 6.2.

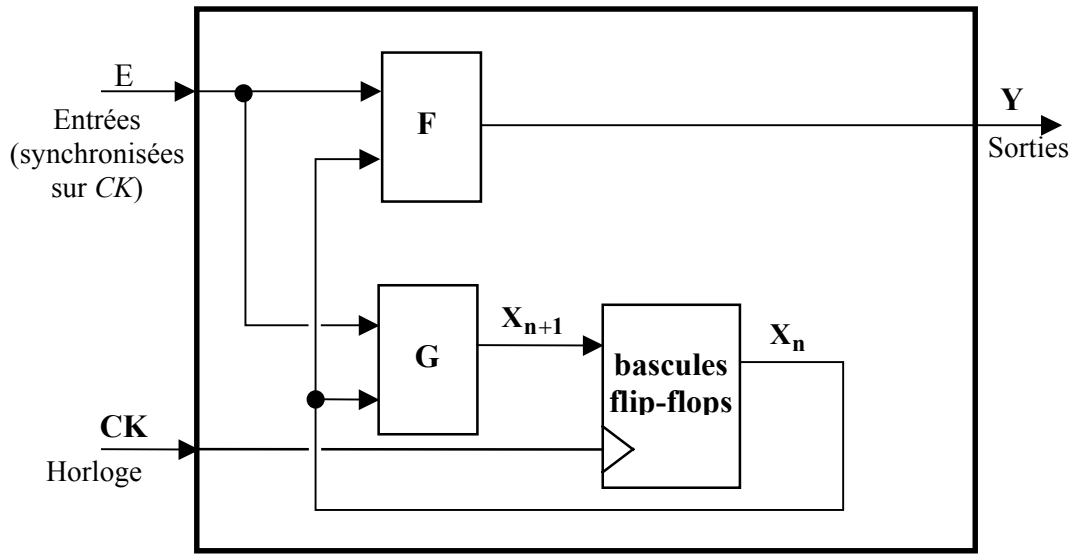


figure 6.2 : machine de Mealy synchrone

A titre de comparaison, la machine de Moore est présentée figure 6.3.

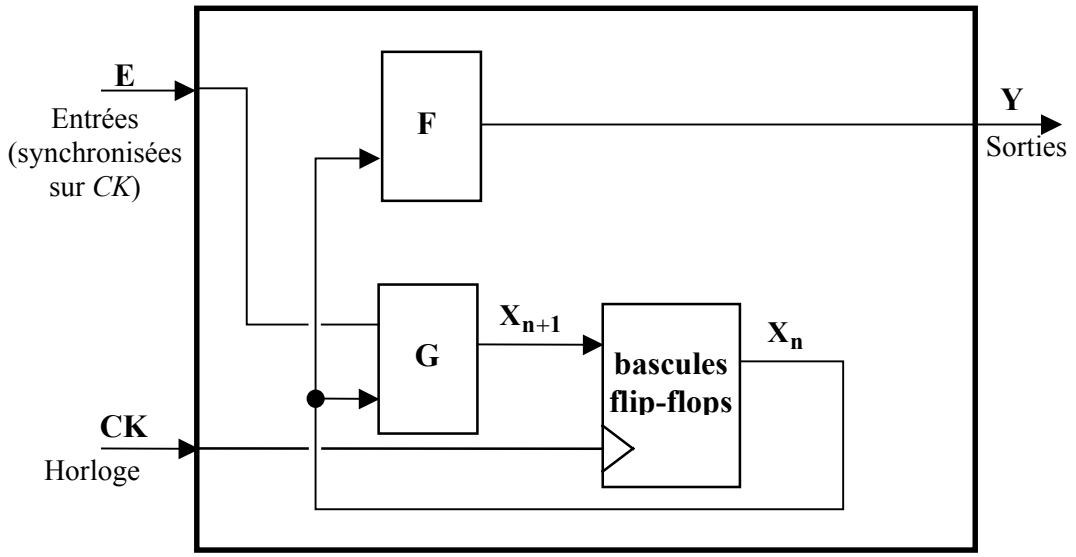


figure 6.3 : machine de Moore synchrone

Il s'agit d'une restriction de la machine de Mealy. On constate que la fonction de sortie F ne reçoit plus le vecteur d'entrée **E**. En conséquence les valeurs de sortie ne sont plus directement liées aux entrées, elles ne dépendent plus que de l'état interne de la machine.

Au plan comportemental, cette restriction implique qu'il n'est plus possible comme dans le cas de l'automate de Mealy de conditionner directement les valeurs de sortie à celles des entrées de l'automate. Comme l'illustre la figure 6.4 pour un automate à une entrée *E* et une sortie *S*, un état E1 exhibant une sortie conditionnelle dans la spécification d'un automate de Mealy (figure 6.4 (a) et (b)) ne peut être traduit à l'aide d'un automate de Moore qu'au prix de l'insertion d'un état supplémentaire E'1 (figure 6.4 (c) et (d)). La machine de Moore est donc potentiellement moins rapide que la machine de Mealy.

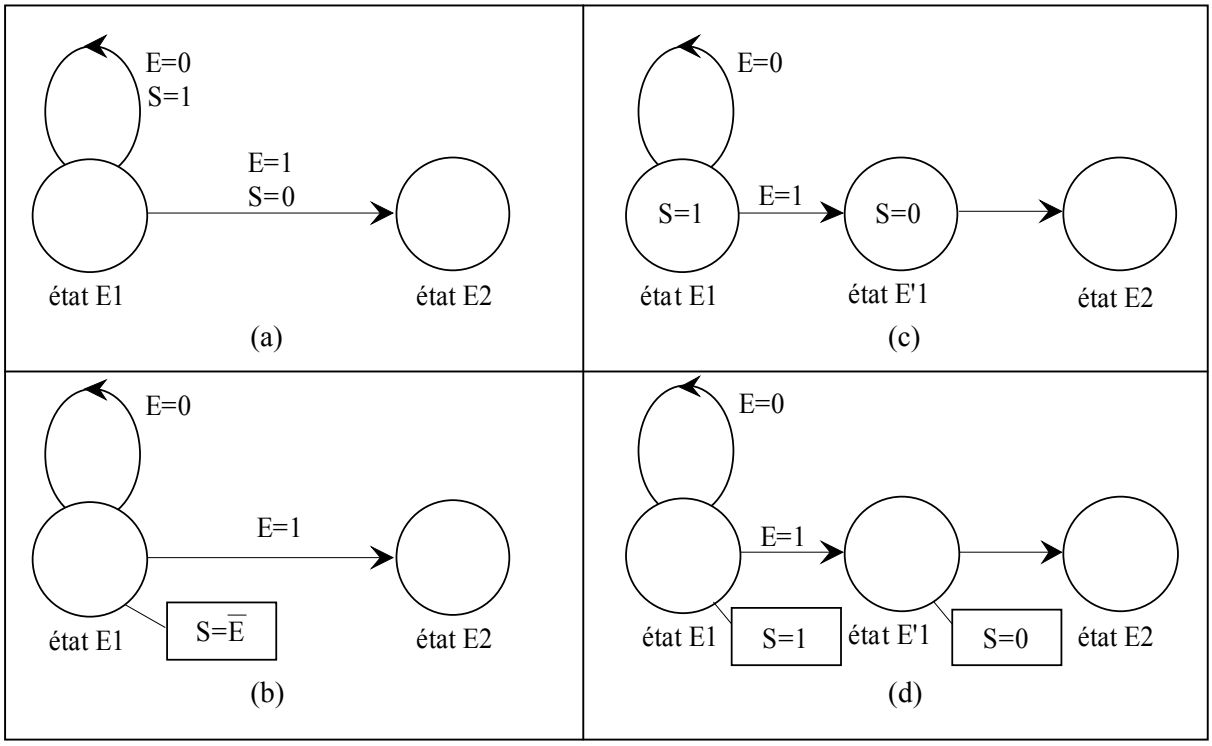


figure 6.4 : Mealy (a) et (b) versus Moore (c) et (d)

Notation : Sur la figure 6.4, les schémas (a) et (b) donnent deux représentations graphiques possibles d'une même portion du graphe d'états d'une machine de Mealy. Les schémas (c) et (d) montrent deux représentations graphiques possibles d'une même portion du graphe d'états d'une machine de Moore.

2.3 Mise en œuvre des automates

La réalisation des machines de Mealy et de Moore relève d'une démarche qui est présentée dans la section 5.1. Cette démarche conduit à spécifier les deux blocs combinatoires F et G du modèle sous forme d'équations booléennes. La suite consiste pour l'essentiel à opérer un choix de technologie (qui peut être imposé par le cahier des charges) pour implémenter les blocs combinatoires et les bascules.

Nous citons ici une possibilité de réalisation d'une machine de Moore ou de Mealy à l'aide de deux composants seulement. Elle consiste à utiliser une mémoire PROM et un registre tampon R à déclenchement sur front, tel que le montre la figure 6.5.

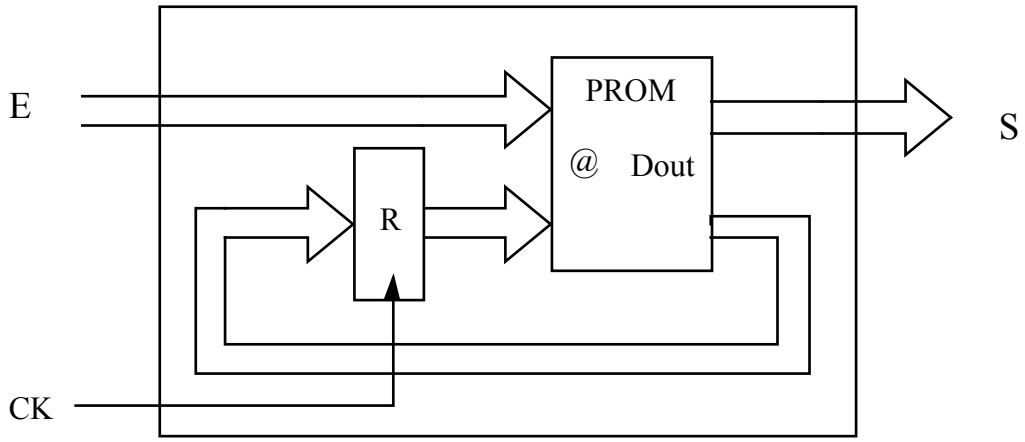


figure 6.5 : réalisation d'une machine à états finis par PROM et registre

Sur cette figure, on constate que le vecteur d'entrée et le vecteur d'état sont connectés aux entrées d'adresse (notées @) de la PROM. Le codage des fonctions F et G est effectué sous forme de table de vérité au sein de la PROM. Chaque mot mémoire correspond à une combinaison des variables internes et d'entrée, son contenu est défini comme les valeurs correspondantes qui sont prises par les fonctions F et G. Cette solution a l'avantage d'être compacte et générale (structure unique pour Mealy et Moore). En contrepartie, elle ne se prête à aucune optimisation dans la réduction du coût de réalisation.

N. B. : Dans la sémantique de la machine à états synchrone, les commutations des entrées sont supposées être synchrones avec les commutations des variables d'état. Si tel n'est pas le cas, le comportement de la machine est erratique. Dans une telle situation, il est indispensable de resynchroniser les entrées avant de les exploiter dans la machine (cf. chapitre 5, § 6.2.3). L'implantation PROM-registre permet d'intégrer très simplement cette modification comme le montre la figure 6.6. Toutefois, cette disposition a une influence dont il faut tenir compte dans le comportement de l'automate, puisqu'il y a un retard dans la réaction de celui-ci vis-à-vis de ses entrées d'au plus une période d'horloge.

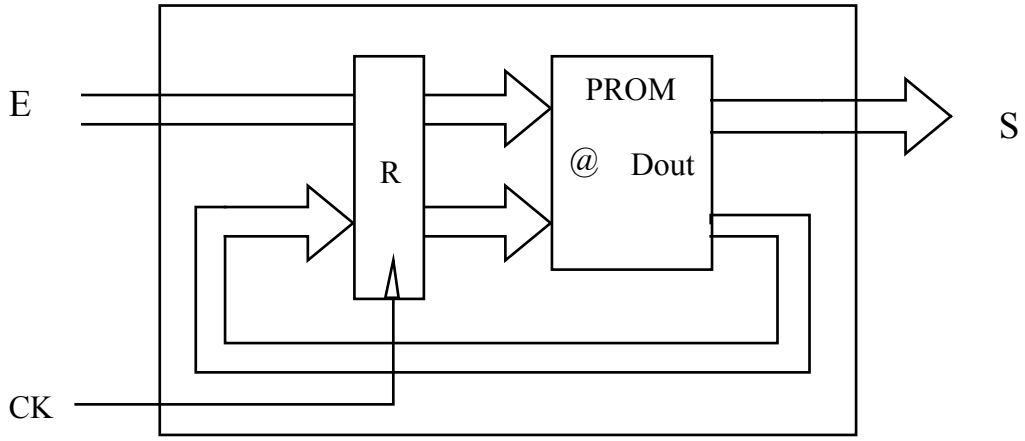


figure 6.6 : réalisation d'une machine à états finis par PROM et registre avec traitement des entrées asynchrones

2.4 Complexité des machines à états finis

Lorsque l'on réalise une application, que ce soit sous forme logicielle ou matérielle, il faut s'interroger sur sa complexité. En effet, ce critère permet de prédire la « non-faisabilité » éventuelle de l'application. Le concepteur s'intéresse en général à l'estimation de l'évolution de la complexité[2] du système en fonction de la taille du problème que ce dernier est censé résoudre.

Dans le cas d'une réalisation logicielle, la complexité s'exprime en nombre d'opérations élémentaires. Dans une réalisation matérielle, on s'intéresse au nombre d'opérateurs élémentaires, c'est-à-dire au nombre de portes logiques nécessaires à la réalisation.

Dans le cas d'une machine séquentielle, la complexité associée à la réalisation des blocs combinatoires F et G dépend de deux facteurs :

- le nombre *n* d'entrées de la machine,
- le nombre *P* d'états du graphe spécifiant son comportement.

On peut exprimer la complexité d'une fonction booléenne comme le nombre de mintermes associés à ses entrées. La complexité d'une fonction booléenne de *n* entrées croît donc en $O(2^n)$. D'autre part, le nombre de variables internes *p* nécessaires au codage de *P* états est compris entre $p = \log_2 P$ (codage dit dense) et $p = P$ (codage dit « one hot » : une bascule par état), ce qui correspond en terme de complexité à $\Omega(\log_2 P)$ et $O(P)$. Les fonctions F et G reçoivent en entrée les variables internes et les entrées externes. La complexité matérielle d'un automate à *n* entrées et *P* états est donc majorée par $O(2^{n+P})$. Il s'agit d'une loi exponentielle qui engendre ce que l'on appelle « l'explosion combinatoire » lorsque *n* et *P* croissent. Pour limiter ce phénomène, de nombreuses méthodes d'optimisation combinatoire ont été imaginées pour s'approcher de la borne

[2] Comme il s'agit d'une estimation, on exprime généralement la complexité à une constante multiplicative près. Parfois même, on ne s'intéressera qu'à son comportement asymptotique (i.e. quand la taille du problème tend vers l'infini). Les fonctions définissant respectivement les bornes inférieure et supérieure de la complexité sont notées $\Omega()$ et $O()$.

inférieure de la complexité. On peut ainsi facilement atteindre $P2^n$ en opérant un codage dense des états de l'automate.

Toutefois, lorsque l'on restreint le problème au cas de la réalisation d'unités de contrôle, une autre voie d'optimisation s'offre : identifier un comportement stable de l'unité et tenter d'en tirer parti pour réduire la complexité. Une autre voie, qui peut être complémentaire, consiste à jouer sur un compromis entre complexité matérielle et temporelle (fréquence maximale de fonctionnement). Ce sont ces deux idées qui sont exploitées conjointement dans le concept de **séquenceur** qui est présenté dans la suite.

3. Les séquenceurs

Ce concept est issu de travaux consacrés au développement des ordinateurs qui datent des années 50. Le point de départ en a été la constatation que le caractère dominant dans le comportement d'une unité de contrôle est la **séquence**. D'où l'idée du séquenceur en tant que machine à produire des séquences. Une telle machine se construit assez naturellement à partir d'un compteur binaire. Mais un comportement réel montre des irrégularités qui consistent à commuter (i.e. faire un saut) d'une séquence à une autre. Un compteur programmable, grâce à sa fonction de chargement, peut répondre à ce besoin. Enfin, une unité de contrôle ne se limite pas à un comportement aveugle, elle doit pouvoir réagir à des événements externes, il faut donc pouvoir conditionner l'exécution de ces sauts à des signaux externes. C'est ici qu'intervient le compromis entre complexité matérielle et temporelle. Contrairement à une machine à états finis, on impose dans un séquenceur des restrictions sur l'ensemble des entrées influant sur l'exécution d'un saut. La position extrême qui consiste à limiter, pour un saut donné, ce sous-ensemble à une seule entrée, est dans les faits souvent retenue. D'un point de vue matériel, cette fonctionnalité de sélection « 1 parmi n » est naturellement prise en charge par un multiplexeur adressé par la position courante du séquenceur.

Ce principe ainsi mis en œuvre est appelé **séquenceur câblé**. On verra par la suite, une évolution visant à intégrer un niveau logiciel dans la machine, que l'on nomme **séquenceur microprogrammé**.

3.1 Le séquenceur câblé

La structure générale d'un séquenceur câblé est décrite par la figure 6.7. Par rapport à la description précédente, on découvre deux éléments nouveaux :

- Un **décodeur d'actions** dont le rôle est d'activer une sortie en fonction de la position courante du compteur.
- Un **transcodeur** dont le rôle est de calculer l'adresse des sauts, c'est-à-dire la valeur qu'il faudra charger dans le compteur si la condition de saut est satisfaite.

Il est important de comprendre qu'il n'existe qu'une alternative dans le cas d'un saut : soit il se réalise, soit le compteur progresse d'un pas, c'est-à-dire, en séquence. C'est encore une restriction comportementale par rapport à une machine à états finis, qui permet de réduire la complexité matérielle. On note enfin que certaines entrées du multiplexeur de conditions peuvent être connectées au 0 logique (i.e. condition toujours fausse) ou au 1 logique (i.e. condition toujours vraie). La condition toujours fausse impose au compteur le fonctionnement en séquence, c'est-à-dire son mode

de base. La condition toujours vraie est une facilité offerte au concepteur pour organiser plus aisément l'algorithme qui sera implémenté par le séquenceur.

Une fois le câblage et la fonction de transcodage établis, le comportement du séquenceur est figé, d'où l'expression de séquenceur « câblé ».

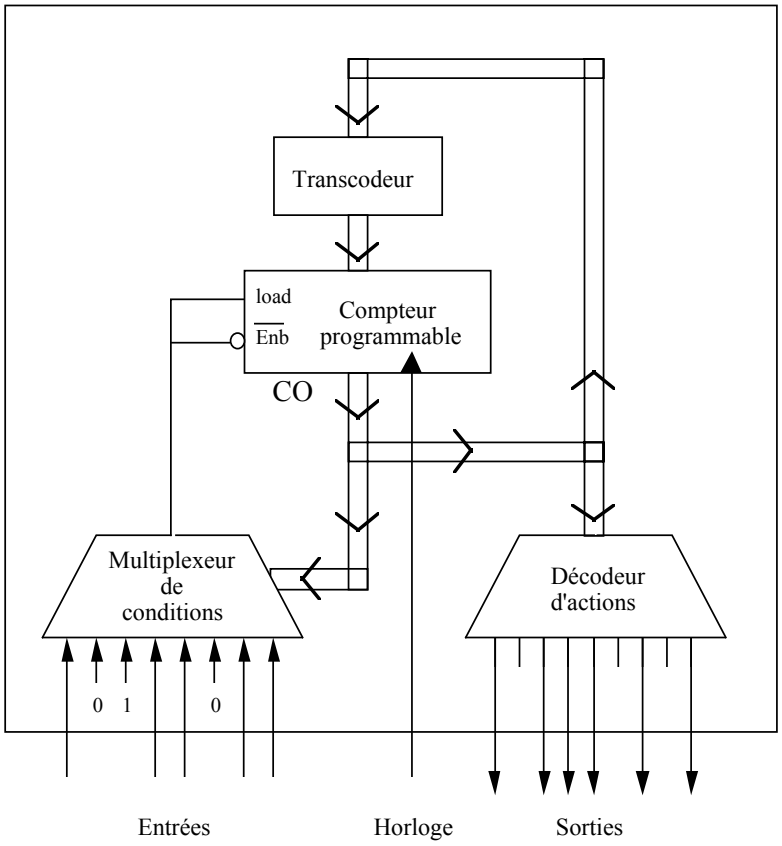


figure 6.7 : structure générale d'un séquenceur câblé

Une notion essentielle qui apparaît ici est celle d'algorithme. En effet, pour spécifier le comportement d'un séquenceur, on n'utilise plus le graphe d'états comme pour les AEF, mais l'algorithme. L'algorithme représente par rapport au graphe d'états la même restriction comportementale que le séquenceur par rapport à la machine à états finis. Dans la conception d'une unité de contrôle par séquenceur, on sera donc parfois amené à transformer un graphe d'états en algorithme, celui-ci devant répondre aux restrictions de séquencement du séquenceur. Ce problème est abordé dans la section 5.2.2.

Ce rapprochement avec le concept de programmation impérative amène à baptiser le compteur programmable du séquenceur **compteur ordinal** (CO), car son rôle est similaire à celui du compteur ordinal qui existe dans l'unité centrale des ordinateurs conventionnels.

3.2 L'approche microprogrammée

En 1951, Wilkes a proposé une version de séquenceur plus proche encore du concept de programmation : le **séquenceur microprogrammé**. Le principe consiste à contraindre le comportement du séquenceur, non plus exclusivement par le câblage, mais aussi par une **couche logicielle**. Le but recherché était double. Tout d'abord permettre de synthétiser des comportements

plus complexes en offrant au concepteur (programmeur) une vue plus abstraite du support d'exécution. Ensuite, gagner en flexibilité. Ainsi, à partir d'un même support matériel d'exécution, on peut engendrer des comportements variables.

Dans cette optique, une mémoire a été insérée au niveau des sorties du compteur ordinal (cf. figure 6.8). En conséquence, celui-ci n'agit plus directement sur le reste de la structure. Il sélectionne un mot dans la mémoire, et c'est le contenu de ce mot qui agit sur la structure. Les mots stockés en mémoire constituent les instructions du séquenceur, ou plus exactement ses **micro-instructions** afin de faire le distinguo avec les instructions machine d'un ordinateur.

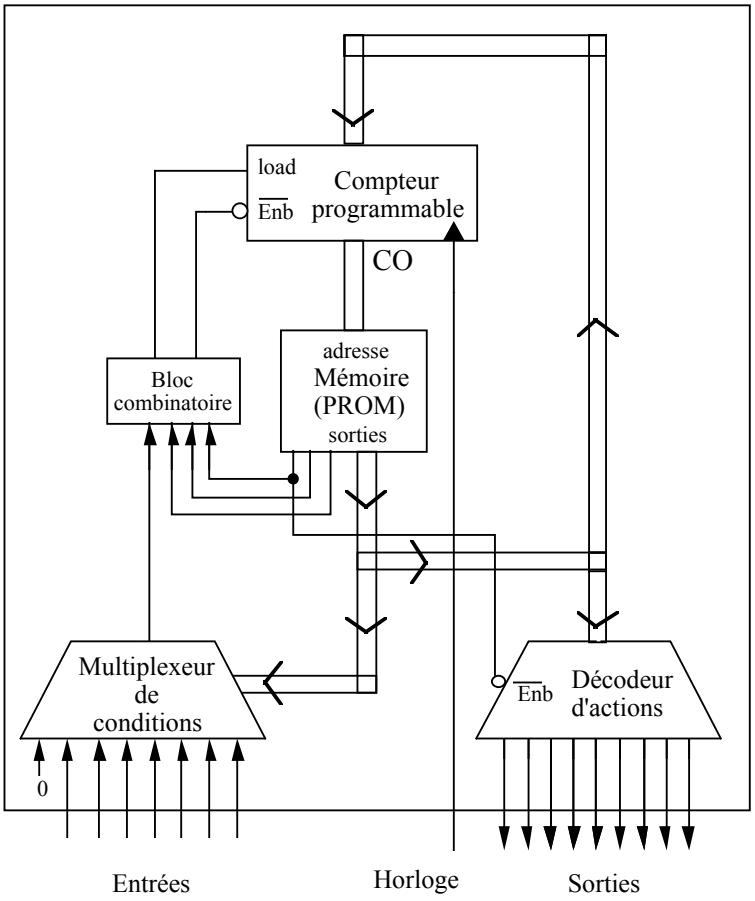


figure 6.8 : schéma de principe d'un séquenceur micro-programmé.

Ainsi qu'on l'a déjà évoqué, la microprogrammation favorise la synthèse de comportements complexes. On peut notamment enrichir les modes de séquencement en ajoutant, par exemple, l'attente conditionnelle, le choix de la polarité de la condition, voire même l'appel de sous-microprogrammes en adjoignant une **mémoire pile** pour sauvegarder le contenu du compteur ordinal. Cette situation offre encore une multitude de possibilités pour jouer sur le compromis complexité matérielle/temporelle.

Par principe, le comportement d'un séquenceur microprogrammé ne peut être appréhendé sans lui associer la structure de la micro-instruction élaborée par le concepteur. La figure 6.9 présente, à titre d'exemple, une structure typique de micro-instruction pour le séquenceur de la figure 6.8.

figure 6.9 : un exemple de structure de micro-instruction

Une interprétation possible du contenu de la micro-instruction est proposé dans la suite de cette section : le bit *C*1 constitue le **code opératoire** ou **code d'ordre** de la micro-instruction et les bits restants (*D*1, …, *D*9) forment le **champ opérande** de celle-ci.

On distingue ici 2 types de micro-instructions :

- les micro-instructions de type action : (*C*1= 0) activation de sorties, l'enchaînement est alors séquentiel.

  Dans ce cas les bits *D*1 à *D*9 codent le numéro de la sortie à activer (en fait 3 bits parmi les 9 suffisent pour sélectionner une sortie sur 8 du décodeur). Ce champ peut lui-même être segmenté s'il est nécessaire d'activer plusieurs sorties simultanément. On peut même afficher directement l'état des sorties dans ce champ.

- les micro-instructions de type séquencement conditionnel : (*C*1 = 1).

  Les bits *D*1 à *D*9 sont segmentés en 4 champs :

  - *D*1 sélectionne le saut ou l'attente conditionnels (1/0),
  - *D*2 définit la polarité de la condition (directe/inverse),
  - *D*3 … *D*5 sélectionnent l'entrée du multiplexeur de conditions (3 bits sont nécessaires pour sélectionner 1 entrée parmi 8),
  - *D*6 … *D*9 codent l'adresse du saut (sans effet en cas d'arrêt conditionnel).

Ces différents champs associés au câblage à 0 de l'entrée d'indice zéro du multiplexeur de conditions enrichit singulièrement les possibilités de séquencement. On dispose en effet des modes suivants :

- enchaînement inconditionnel (attente condition et condition fausse),
- saut inconditionnel (saut sur condition et condition vraie),
- saut sur condition vraie,
- saut sur condition fausse,
- attente condition vraie,
- attente condition fausse.

Un bloc combinatoire est chargé d'élaborer les commandes du compteur ordinal (cf. figure 6.8). Il reçoit en entrée les bits *C*1, *D*1, *D*2 de la micro-instruction ainsi que le signal issu du multiplexeur de conditions.

4. Bilan comparatif

Si l'on prend comme critère la complexité, on peut établir un bilan assez simple dans la comparaison des approches « machine à états » et séquenceur (figure 6.10). On peut même y introduire aisément le microcontrôleur qui offre un support tout intégré à une approche fortement dominée par le logiciel.

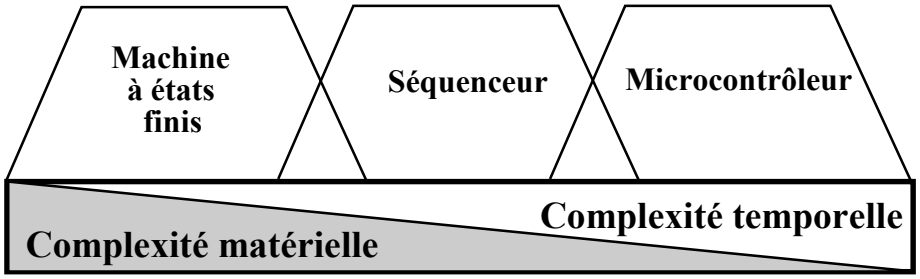


figure 6.10 : comparaison de la complexité des différentes réalisations de systèmes séquentiels complexes étudiées

D'un point de vue pratique, il s'ensuit qu'une machine à états est plus rapide que les solutions concurrentes, qui, elles, font des économies en matériel au prix d'une séquentialisation du comportement.

Toutefois, cette figure ne mentionne pas des critères également importants tels que :

- les limites en complexité comportementale induites par les différentes approches,
- le degré d'universalité des approches (ou flexibilité),
- l'existence de solutions technologiques spécifiques avantageuses.

La figure 6.11 tente d'évaluer ces deux premiers critères pour les différentes approches envisagées.

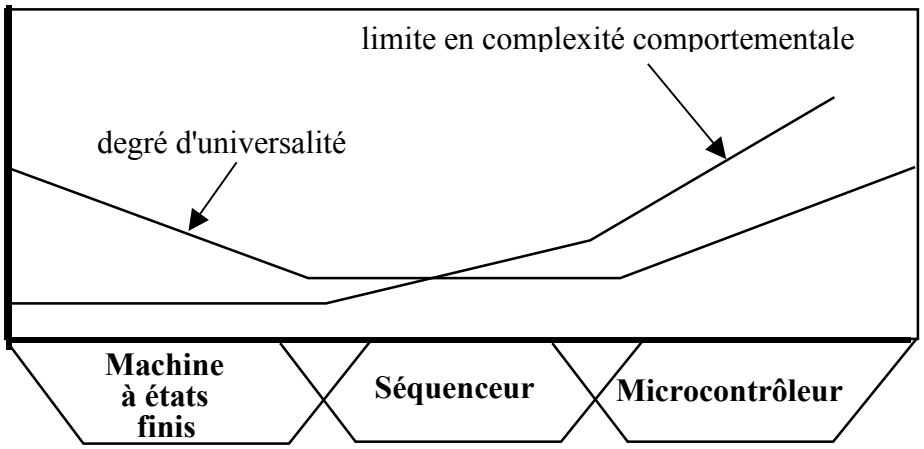


figure 6.11 : comparaison des approches câblées et programmées en termes de flexibilité et de complexité comportementale

L'abstraction du matériel grâce à une couche logicielle permet de repousser les limites de la complexité comportementale synthétisable. Il est peut-être moins évident de comprendre l'évolution du degré d'universalité qui montre un creux pour l'approche séquenceur. En fait, le modèle conceptuel sur lequel repose la machine à états, (i.e. l'automate à états finis), est à la fois très simple et très puissant. Un graphe est beaucoup plus général qu'un algorithme, ce dernier étant obtenu au prix de contraintes comportementales fortes. Il s'ensuit que la structure du séquenceur est plus restrictive sur le comportement que celui-ci peut engendrer. Autrement dit, la structure du séquenceur est plus fortement dépendante du comportement à synthétiser que celle de la machine à états. On pourrait s'attendre à ce que cette dépendance entre comportement et structure soit encore plus marquée dans le cas du microcontrôleur. La réalité est inverse car le comportement de celui-ci est synthétisé par l'intermédiaire d'un langage. Ce n'est plus la structure du support matériel qui fixe le degré d'universalité, mais le langage auquel on recourt pour fixer son comportement. La seule exigence étant que le support d'exécution offre les mécanismes de base nécessaires à l'implémentation du langage.

5. Méthodes de conception d'une unité de contrôle

5.1 Démarche associée à la conception d'une machine à états finis

Dans quelles circonstances opter pour une machine à états ?

- Quand la vitesse de réaction exigée ne peut être obtenue par un microcontrôleur ou un séquenceur.

  Si la complexité matérielle n'est pas raisonnable (dépend de la technologie de réalisation), il faut procéder à une décomposition de l'automate en un ensemble d'automates plus simples qui collaborent pour engendrer le comportement global attendu.

- Quand la complexité de l'automate est faible et que l'on recherche une implantation compacte, une solution plus coûteuse à base de séquenceur ou de microcontrôleur ne se justifie pas.

Si l'on n'est pas dans l'un ou l'autre des cas précédents, on s'orientera plutôt vers une réalisation à base de microcontrôleur. Toutefois, s'il n'existe pas sur le marché de modèle suffisamment performant pour satisfaire aux contraintes de temps de l'application, ou encore si une réalisation ASIC est imposée, on optera pour un séquenceur.

5.1.1 Synthèse d'une machine de Mealy : la méthode d'Huffman

La méthode dite d'Huffman guide le concepteur à partir du cahier des charges d'un automate jusqu'à l'établissement des équations des fonctions F et G de la machine de Mealy.

5.1.1.1 Méthode manuelle

1. Établir les spécifications comportementales issues du cahier des charges de l'automate sous la forme d'un graphe d'états et optimiser éventuellement le graphe,
2. Coder les états du graphe,
3. Établir la table de transition de l'automate,
4. Choisir le support matériel pour l'implantation de l'automate,
5. Faire la synthèse des fonctions F et G en tenant compte de la structure du support matériel choisi,
6. Établir le schéma logique (et les listings de programmation dans le cas où des composants programmables sont utilisés),
7. Réaliser et tester l'automate.

Chacune de ces étapes peut donner lieu à un travail d'optimisation éventuel (réduction du nombre d'états, codage astucieux des états, minimisation des fonctions F et G). Ces opérations sont souvent délicates ou fastidieuses car elles sont en général à base d'heuristiques (i.e. de recettes de cuisine) [CV86].

Un exemple de synthèse est traité en section 5.3.3 afin d'illustrer cette méthode.

5.1.1.2 Méthode utilisant des outils de synthèse logique automatique et de simulation

La disponibilité d'un outil de CAO (Conception Assistée par Ordinateur) est devenue aujourd'hui indispensable dans un cadre professionnel. Elle permet d'automatiser une partie du processus de conception et surtout de valider plus facilement chacune de ses étapes.

1. Traduire les spécifications comportementales de l'automate dans un langage accepté par le synthétiseur (VHDL, Verilog, ...),
2. Ecrire un fichier de test et faire une simulation fonctionnelle de la spécification comportementale ($\rightarrow$ corrections éventuelles),
3. Choisir la cible d'implantation, régler les paramètres d'optimisation (surface, vitesse, consommation) et exécuter la synthèse logique,
4. Faire une simulation fonctionnelle et temporelle du schéma logique obtenu ; corriger les paramètres si nécessaire, jusqu'à l'obtention de résultats satisfaisants,
5. Réaliser le composant et le tester.

5.2 Démarche associée à la conception des séquenceurs

5.2.1 Partie commune aux séquenceurs câblés et microprogrammés

1. La première étape de la conception consiste à établir une spécification algorithmique du comportement de l'unité de contrôle à réaliser. Si le comportement de l'automate a déjà fait l'objet d'une spécification sous forme de graphe d'états. Il s'agit alors de traduire ce graphe sous forme d'algorithme en respectant les contraintes décrites en section 5.2.2.
2. La seconde étape consiste à choisir entre une structure câblée et une structure microprogrammée. L'abstraction qu'offre le concept de programmation, ainsi que la plus grande flexibilité des structures microprogrammées plaident naturellement en faveur de cette seconde solution. Toutefois, si l'unité de contrôle à réaliser présente un comportement relativement simple ou si les contraintes en temps sont trop sévères pour une réalisation microprogrammée, on s'orientera plutôt vers une solution câblée.

5.2.2 De l'automate à états finis vers la machine de Von Neumann

Nous supposons dans cette section que la spécification comportementale initiale se présente sous la forme d'un graphe d'états.

- **Restriction du nombre d'entrées impliquées dans les transitions de l'automate**

Une seule condition logique peut intervenir à un moment donné dans l'évolution d'un séquenceur. Si le graphe initial laisse apparaître des transitions conditionnées à plus d'une entrée, il est donc nécessaire d'y remédier. Il existe deux possibilités :

- La première consiste à restructurer le graphe en introduisant des états supplémentaires qui servent à séquentialiser le test (voir figure 6.12 et figure 6.13). L'inconvénient est un accroissement du temps de réaction de l'automate.

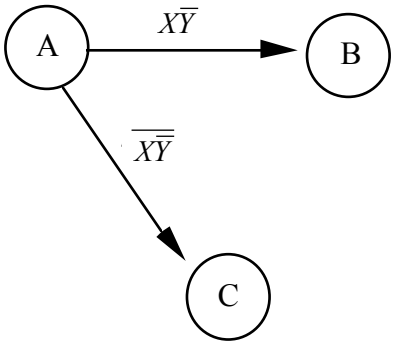


figure 6.12 : extrait d'un graphe d'états

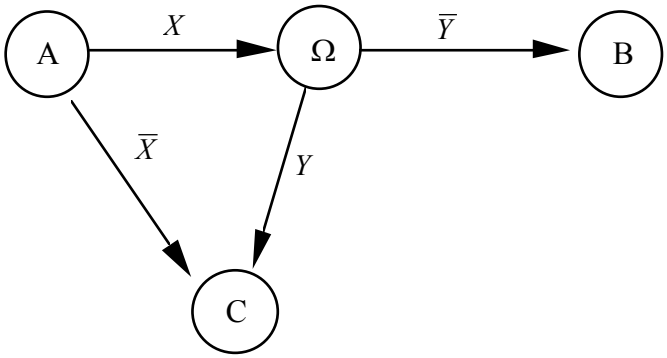


figure 6.13 : addition d'un état Ω dans le graphe de la figure 6.12, permettant de limiter le test à une seule entrée

- L'autre possibilité requiert une modification de l'architecture du futur séquenceur. Elle consiste à calculer en amont du multiplexeur (cf. structure du séquenceur de la figure 6.7) la condition logique résultant de la composition des entrées en cause. Cette approche a pour avantage de n'introduire qu'une faible pénalité sur la vitesse du séquencement (chemin critique).

  La figure 6.14 donne une illustration de ce principe pour l'exemple de la figure 6.12. Dans cet exemple, on suppose que les états de l'automate sont codés sur 3 bits Q_2, Q_1, Q_0, et que le code correspondant à l'état A est 010.

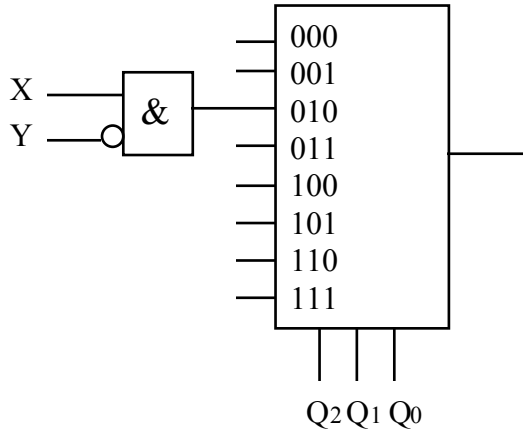


figure 6.14 : composition d'entrées en amont du multiplexeur de conditions

Toutefois, cette seconde méthode n'est pas applicable lorsque les états présentent plus de deux transitions sortantes. Dans ce cas, on ne peut se soustraire à une restructuration du graphe.

- **Restriction dans le comportement des sorties de l'automate**

Elle consiste à interdire aux sorties d'être conditionnées à des entrées de l'automate. En effet, dans l'architecture du séquenceur, il n'y a pas en principe de connexion logique directe entre les entrées et les sorties. On est confronté ici au même problème que lors de la transformation d'une machine de Mealy en une machine de Moore. La démarche est, par conséquent, la même.

En conclusion, lorsque ces opérations ont été réalisées (si besoin est), la fonction chargée d'élaborer les sorties du séquenceur peut s'exprimer par un simple décodage des sorties du compteur ordinal.

- **Conséquences de la séquentialisation d'un graphe**

Les opérations décrites précédemment ne doivent pas altérer le comportement de l'automate par rapport aux spécifications du cahier des charges. Il faut donc vérifier que l'automate séquentialisé (c'est-à-dire le séquenceur résultant) n'introduit pas de dysfonctionnements. Les éventuels dysfonctionnements peuvent provenir principalement de l'introduction artificielle de priorités dans la prise en compte des entrées ainsi que dans l'activation des sorties, ou encore, de l'allongement (en nombre de périodes d'horloge) du temps de réaction de l'automate.

Ces problèmes, s'ils surviennent, peuvent éventuellement être réglés en multipliant la fréquence de l'horloge qui cadence l'automate ou bien encore en réintroduisant du parallélisme dans l'architecture du séquenceur. C'est cette dernière possibilité qui rend l'approche des séquenceurs délicate car ce concept est ouvert à de nombreux compromis entre complexité matérielle et temporelle.

5.2.3 Séquenceur câblé

Une fois la spécification algorithmique établie, on procède aux étapes suivantes :

3. Déstructuration de l'algorithme afin de restreindre les structures de contrôle au seul saut conditionnel (mise en place d'étiquettes). Chaque ligne de l'algorithme est mise sous la forme :

```
    [étiquette]   action, séquencement ;
```
où **`séquencement = IF condition_booléenne GOTO étiquette`**

4. Assemblage du programme : numérotation des lignes du programme (numéros ⇔ adresses) et remplacement des références aux étiquettes par les adresses.
5. Établissement des équations des entrées du multiplexeur de conditions et des sorties du décodeur d'actions.
6. Choix de l'amplitude de comptage du compteur ordinal et synthèse des équations des entrées de chargement de celui-ci (adresses des étiquettes référencées dans les sauts conditionnels).
7. Réalisation du schéma logique du séquenceur.

5.2.4 Séquenceur microprogrammé

Une fois la spécification algorithmique établie, on procède aux étapes suivantes :

3. Définition d'un code d'ordre (i.e. jeu d'instructions).
4. Élaboration d'un langage mnémonique, puis réécriture de l'algorithme en langage mnémonique.
5. Définition du jeu de micro-instructions :
   - Affectation des entrées/sorties du séquenceur
   - Choix de la taille de l'adresse des instructions
   - Établissement de la structure de la micro-instruction.
6. Assemblage (à la main ou avec un outil de compilation configurable) du code mnémonique pour produire le code exécutable.
7. Établissement du schéma logique du séquenceur.
8. Écriture du microprogramme de test et simulation.
9. Réalisation du séquenceur, programmation de la mémoire d'instructions et tests.

5.3 Illustration avec le contrôleur d'alternat pour liaisons synchrones

Ce système numérique a été présenté à titre d'exemple lors du cours introductif du bloc (cours 1). Il est utilisé par la suite pour illustrer les méthodes présentées dans les sections 5.1 et 5.2.

5.3.1 Spécification de l'application

La finalité de ce système (voir figure 6.15) est d'offrir une communication « full duplex », c'est-à-dire simultanément bidirectionnelle entre deux stations qui sont, en réalité, reliées par un canal unique. Le principe adopté consiste à effectuer un multiplexage temporel des communications, le canal de

transmission devant avoir un débit suffisant pour donner ainsi l'illusion aux stations clientes que la communication s'effectue simultanément dans les deux sens.

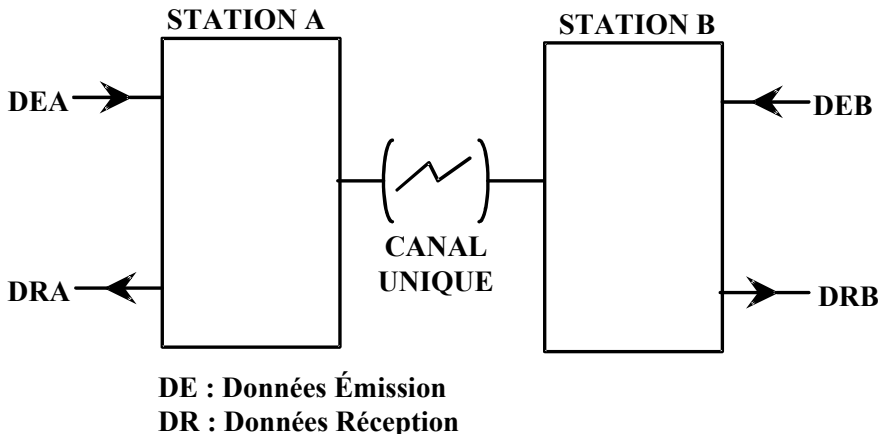


figure 6.15 : principe du contrôleur d'alternat pour liaisons synchrones.

Le transport des informations sur le canal ne peut se faire directement sous forme brute. En effet, certaines données technologiques (récupération d'horloge, détection d'erreurs de transmission, …) et la synchronisation nécessaire au dialogue entre les stations requièrent la mise en place de **protocoles de communication**. Ceci implique l'ajout d'informations supplémentaires aux données brutes à transporter. On parle d'**encapsulation des données**. L'ensemble des données véhiculées par le canal est structuré sous la forme de trames de communication. Dans le cas de l'application considérée, la structure de la trame de communication est présentée en figure 6.16.

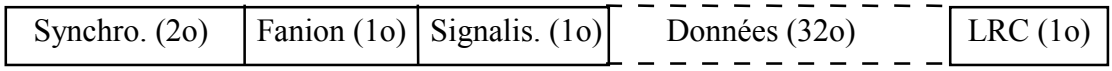


figure 6.16 : structure de la trame de communication

Le rôle des différents champs est le suivant :

- **Synchronisation** (2 octets 0101...01) :

  Les stations qui communiquent à travers le système d'alternat ont chacune leur horloge propre, qui sont calées aussi précisément que possible sur la même fréquence. Toutefois, il n'existe pas de référence de phase commune, ce qui pose problème côté réception pour échantillonner les bits. Le champ de synchronisation contient une alternance de 1 et de 0 qui permet à une boucle à verrouillage de phase pilotant le récepteur de se caler sur la phase du train de bits reçu.

- **Fanion** (1 octet 01111100) :

  Le fanion est un motif binaire attendu par le récepteur pour lui indiquer la fin du champ de synchronisation et afin de se repérer dans la trame.

- **Signalisation** (1 octet) :

  Ce champ est utilisé pour gérer le protocole de communication. Il permet notamment à une station de signaler son intention d'émettre une trame de données, d'acquitter la

demande de son interlocuteur, de signaler une erreur de transmission sur la dernière trame reçue, …

- **Données** (32 octets) :

  Il s'agit de l'information destinée à la station réceptrice.

- **LRC** (1 octet) :

  Il s'agit d'un code de détection d'erreurs de transmission (Longitudinal Redundancy Check) dont chaque bit reflète la parité de l'ensemble des bits de même rang du champ d'information. Ce code est calculé par l'émetteur et inséré dans le champ qui lui est destiné. Le même calcul est opéré à la réception. S'il n'y a pas égalité du LRC transmis et du LRC recalculé, c'est qu'il y a eu erreur de transmission.

La première étape de conception du système, la spécification, procède en identifiant les différentes fonctions que doit réaliser le système pour répondre à sa finalité. Ces fonctions sont les suivantes :

- gestion du protocole de communication (construction et analyse des trames),
- synchronisation des stations et gestion des débits,
- stockage temporaire des données,
- formatage des données à l'émission (multiplexage 1, 2 ou 4 voies d'entrées, conversion de flots continus de données en paquets),
- formatage des données en réception (conversion des paquets en flots continus de données, démultiplexage vers les voies de sortie).

5.3.2 Découpage fonctionnel du contrôleur d'alternat

La phase de conception du système consiste tout d'abord à identifier les interfaces entre ces fonctions en procédant éventuellement à une décomposition préalable en blocs fonctionnels. On obtient ainsi une **vue structurelle du système**, représentée figure 6.17, qui doit être accompagnée des documents de spécification comportementale propres à chaque bloc.

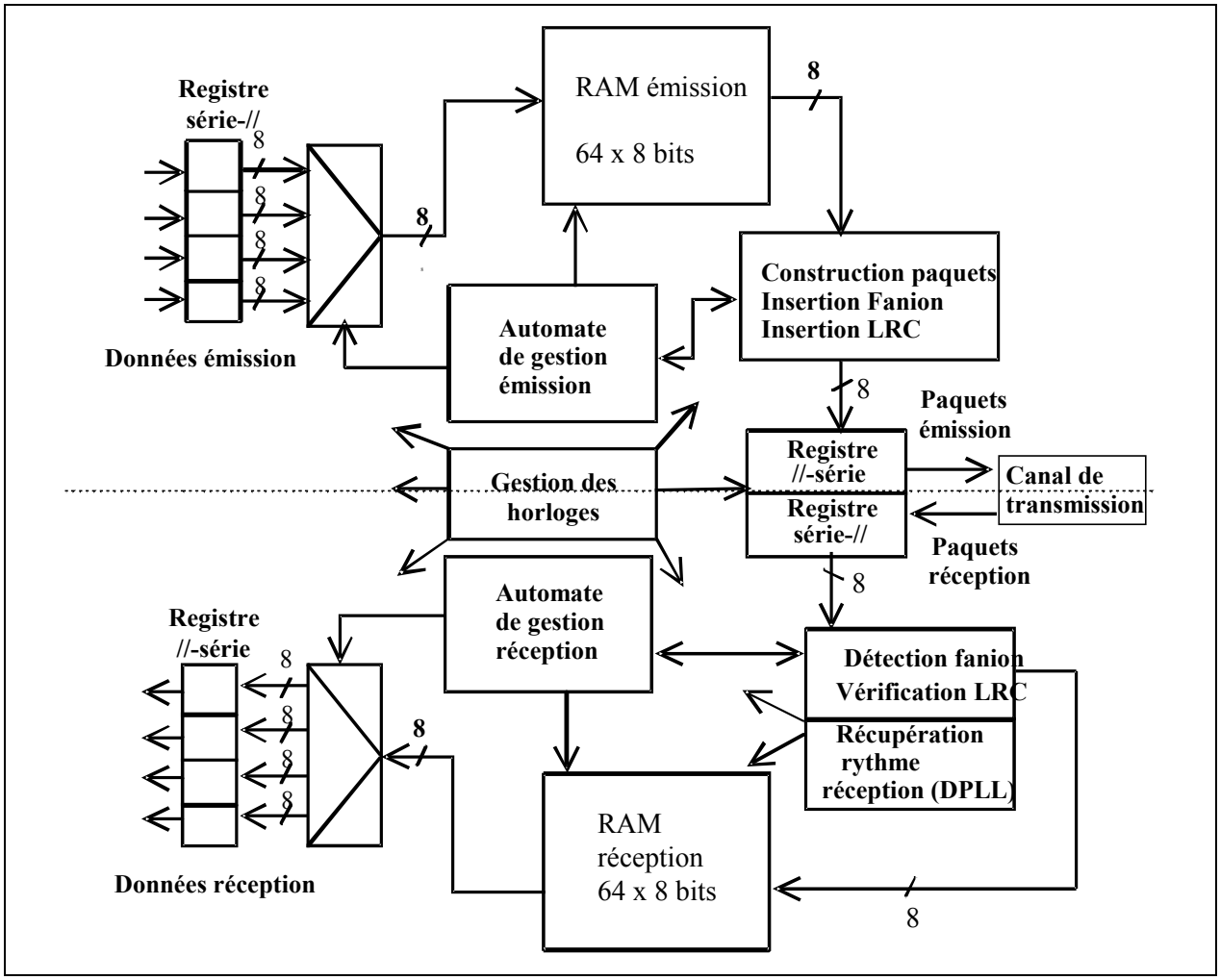


figure 6.17 : vue structurelle détaillée du système d'alternat

En suivant le cheminement des données dans la voie d'émission, on y trouve tout d'abord un étage de parallélisation des flux de données qui convertit les trains de bits en octets. Cette opération permet d'effectuer ensuite le multiplexage de ces voies sans contention. En effet, grâce à ce formatage en octets, on conserve le débit d'information présent à l'entrée tout en travaillant avec une fréquence d'horloge 8 fois plus faible. Les octets de données ainsi multiplexés sont mis dans une mémoire (RAM émission) en attente de leur insertion dans une trame de communication (bloc « construction paquets, insertion fanion et LRC ») et de leur conversion sous forme d'un train binaire qui est injecté sur le canal de transmission.

En suivant le cheminement de la voie de réception, on retrouve les fonctionnalités réciproques de la voie d'émission.

Finalement, cette analyse nous a conduits à laisser de côté trois blocs : d'une part, les deux automates assurant respectivement la gestion de l'émission et de la réception et, d'autre part, un bloc appelé « gestion des horloges », dont le rôle est de fournir les signaux de cadencement des autres blocs du système.

On s'intéresse, dans la suite, à la conception des unités de contrôle du système à travers les deux principaux modèles de réalisation présentés en sections 5.1 et 5.2 : les machines à états finis synchrones (de Mealy ou de Moore) et les séquenceurs, câblés ou microprogrammés.

5.3.3 Illustration de la méthode d'Huffman : réalisation de l'automate de la fonction de détection de fanion

En réception, le bloc de détection de fanion peut être réalisé à l'aide d'un registre à décalage, d'un décodeur et d'un automate (cf. figure 6.18). Nous allons réaliser l'automate correspondant.

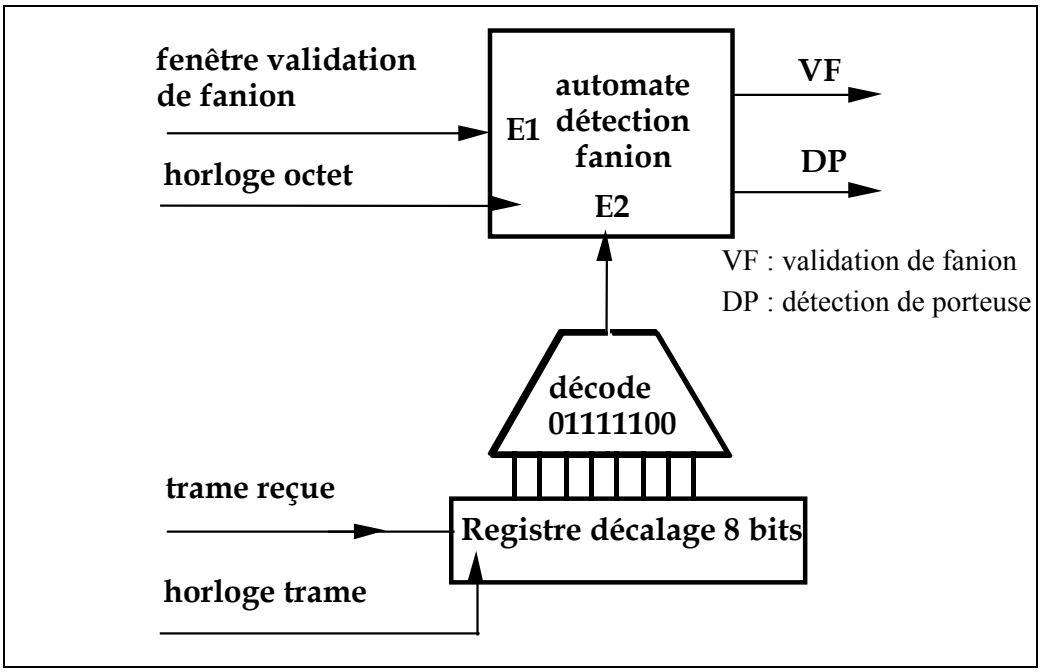


figure 6.18 : structure du bloc de détection de fanion

5.3.3.1 Spécification comportementale de l'automate

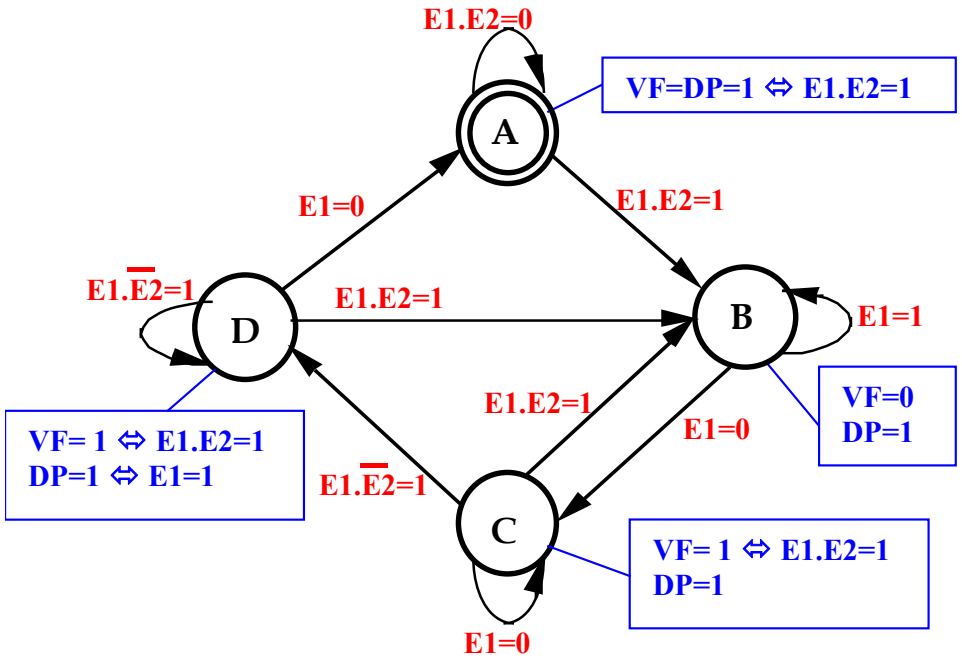


figure 6.19 : graphe d'états de l'automate du bloc de détection de fanion

5.3.3.2 Codage des états de l'automate

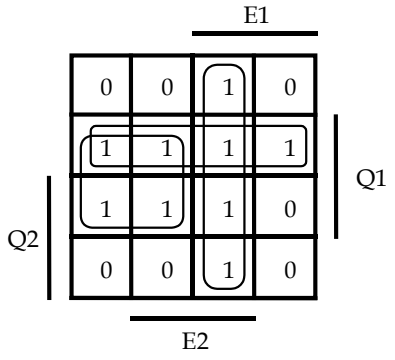
Etat	Q2	Q1
A	0	0
B	0	1
C	1	1
D	1	0

5.3.3.3 Établissement de la table de transition de l'automate

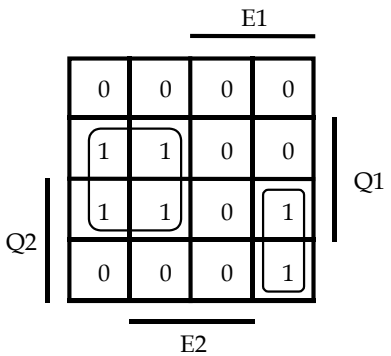
Etat présent		Entrées		Etat futur		Sorties	
Q2	Q1	E1	E2	$Q2^+$	$Q1^+$	VF	DP
0	0	0	X	0	0	0	0
0	0	1	1	0	1	1	1
0	0	1	0	0	0	0	0
0	1	1	X	0	1	0	1
0	1	0	X	1	1	0	1
1	1	0	X	1	1	0	1
1	1	1	1	0	1	1	1
1	1	1	0	1	0	0	1
1	0	0	X	0	0	0	0
1	0	1	1	0	1	1	1
1	0	1	0	1	0	0	1

5.3.3.4 Réalisation de la machine à l'aide de composants élémentaires (portes et bascules D)

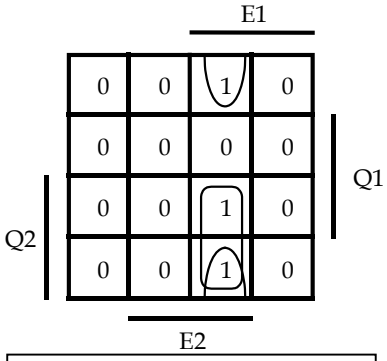
- Synthèse des fonctions G et F



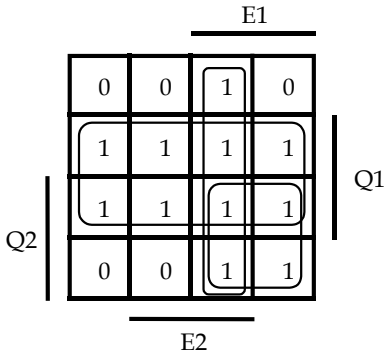
$$Q_1^+ = E_1E_2 + \overline{Q_2}Q_1 + \overline{E_1}Q_1$$



$$Q_2^+ = \overline{E_1}Q_1 + E_1\overline{E_2}Q_2$$



$$VF = E_1 E_2 \overline{Q_1} + E_1 E_2 Q_2$$



$$DP = Q_1 + E_1 E_2 + E_1 Q_2$$

- Schéma logique de l'automate

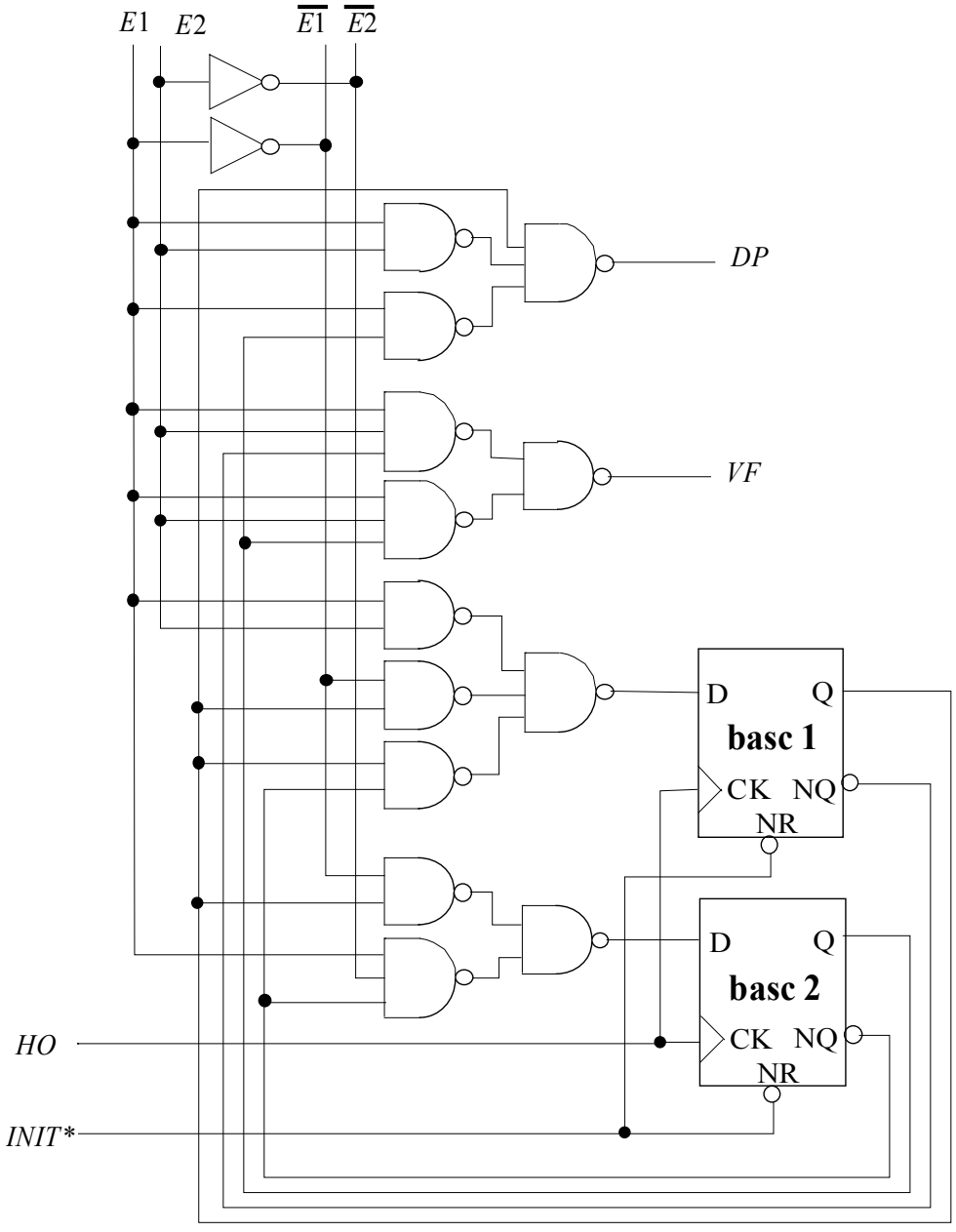
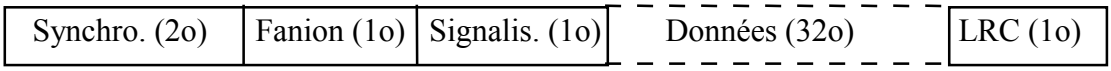


figure 6.20 : schéma logique de l'automate du bloc de détection de fanion

5.3.4 Illustration de la méthode de synthèse d'un séquenceur : réalisation de l'automate d'émission

5.3.4.1 Cahier des charges de l'automate d'émission

Il est chargé du contrôle de la fabrication de la trame et de son émission. Rappel de la structure de la trame de communication :



Définition des entrées/sorties de l'automate :

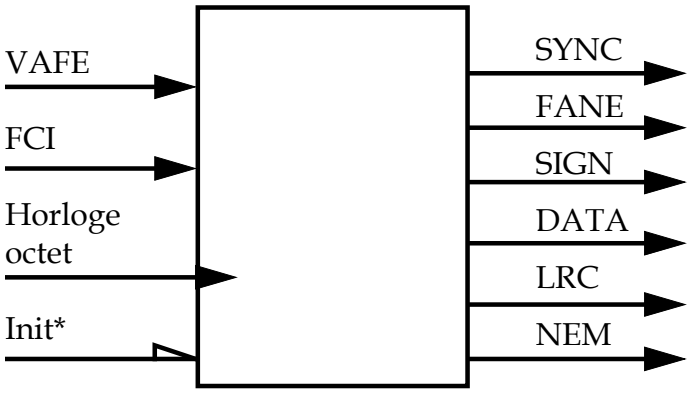


figure 6.21 : entrées/sorties de l'automate d'émission

Les entrées VAFE et FCI correspondent respectivement à « VAlidation du Flux d'Emission » et « Fin du Champ d'Information ». Les sorties correspondent aux différents champs de la trame d'émission. La sortie NEM indique que l'émission de la trame n'a pas encore commencé ou est terminée « No EMission ».

5.3.4.2 Spécification comportementale de l'automate

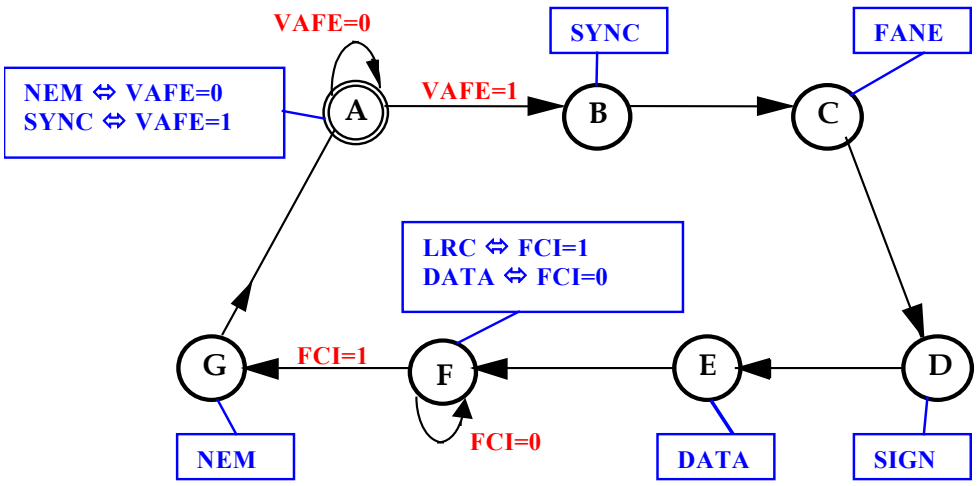


figure 6.22 : graphe d'états de l'automate d'émission

Étant donné le nombre important de sorties, on n'a spécifié sur les arcs du graphe que les sorties activées, c'est-à-dire les sorties valant 1.

5.3.4.3 Reformalisation du graphe en algorithme

```
 SYNC,FANE,SIGN,DATA,LRC,NEM Bool Out)
{
  while TRUE
  {
    while(not(VAFE)) NEM;
    SYNC;
    SYNC;
    FANE;
    SIGN;
    DATA;
    while(not(FCI)) DATA;
    LRC;
    NEM;
  }
}
```

5.3.4.4 Cas du séquenceur câblé

- **Déstructuration de l'algorithme**

L'objet de cette opération est de traduire les structures de contrôle en sauts (ou branchements) conditionnels. Pour ce faire, on insère une étiquette sur chaque ligne qui est destination d'un saut. Les lignes sont alors mises sous la forme : **`[étiquette]  action, séquencement;`**

```
/*seq_émission(VAFE,FCI Bool In;
 SYNC,FANE,SIGN,DATA,LRC,NEM Bool Out)*/
debut   NEM,IF NOT(VAFE) GOTO debut;
        SYNC,IF FALSE GOTO ANYWHERE;
        SYNC,IF FALSE GOTO ANYWHERE;
        FANE,IF FALSE GOTO ANYWHERE;
        SIGN,IF FALSE GOTO ANYWHERE;
encore  DATA,IF NOT(FCI) GOTO encore;
        LRC,IF TRUE GOTO debut;
```

- **Assemblage du programme**

Cette opération consiste à assigner une adresse à chaque ligne de l'algorithme en prévision de son implantation dans le séquenceur.

```
/*seq_émission(VAFE,FCI Bool In;
 SYNC,FANE,SIGN,DATA,LRC,NEM Bool Out)*/
00      NEM,IF NOT(VAFE) GOTO 00;
01      SYNC,IF FALSE GOTO xx;
02      SYNC,IF FALSE GOTO xx;
03      FANE,IF FALSE GOTO xx;
04      SIGN,IF FALSE GOTO xx;
05      DATA,IF FALSE GOTO xx;
06      DATA,IF NOT(FCI) GOTO 06;
07      LRC,IF TRUE GOTO 00;
```

On peut alors déterminer le câblage correspondant du multiplexeur de conditions et du décodeur d'actions :

Câblage du mux de conditions:	**Câblage du décodeur d'actions:**	
C0 = $\overline{\text{VAFE}}$	NEM = Y0	SIGN = Y4
C1..5 = 0	SYNC = Y1+Y2	DATA = Y5+Y6
C6 = $\overline{\text{FCI}}$	FANE = Y3	LRC = Y7
C7 = 1		

- **Synthèse des entrées de chargement du compteur ordinal**

CO	CO^+
0	0
1	X
2	X
3	X
4	X
5	X
6	6
7	0

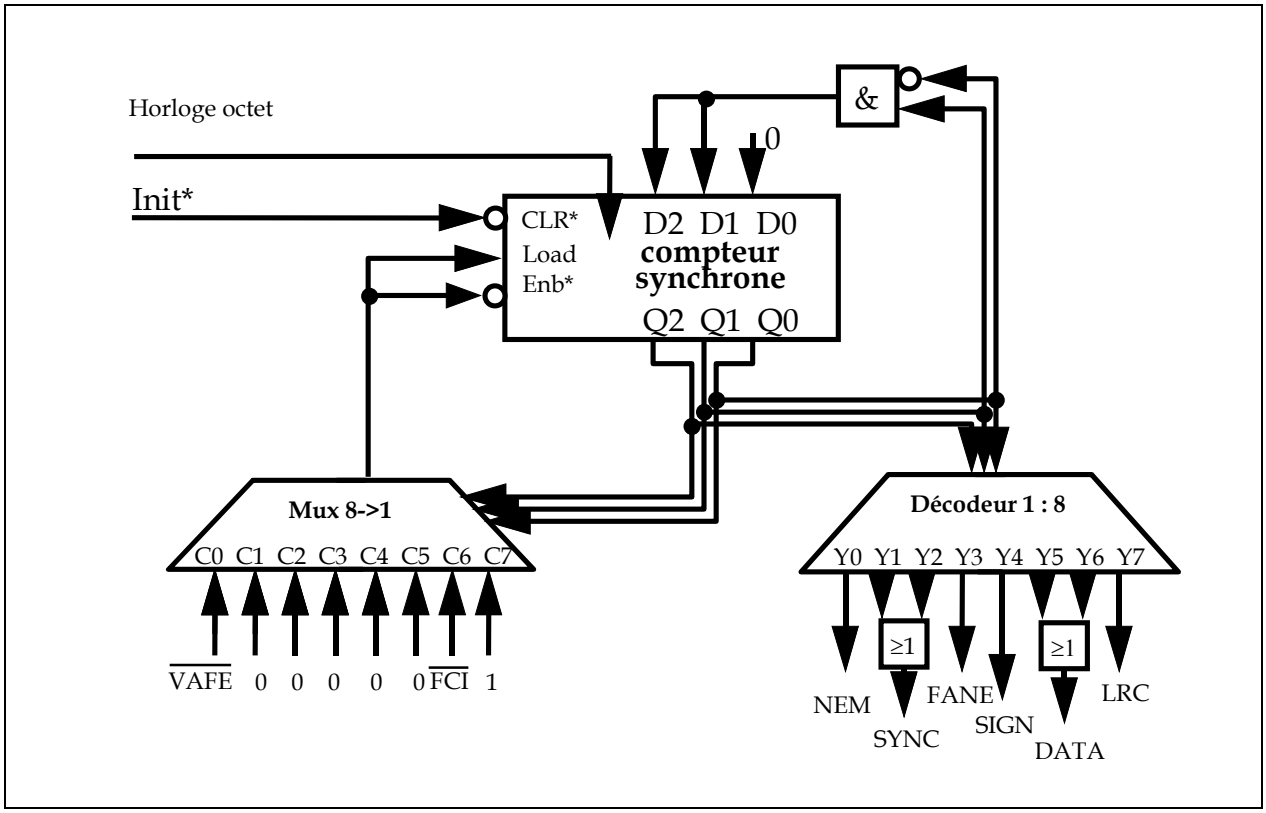
Le programme comptant 8 lignes, on code l'adresse sur 3 bits Q_2, Q_1, et Q_0, ce qui conduit à la table de transition ci-dessus. On la dispose sous forme de tableau de Karnaugh en vue de la minimisation des fonctions de calcul des entrées de chargement du compteur ordinal.

$Q2^+ Q1^+ Q0^+$	Q0			
	000	XXX	XXX	XXX
Q2	XXX	XXX	000	110
				Q1

D'où

$$Q_0^+ = 0$$
$$Q_2^+ = Q_1^+ = Q_1 \overline{Q_0}$$

- **Schéma logique du séquenceur câblé**



5.3.4.5 Cas du séquenceur microprogrammé

- **Établissement d'un code d'ordre**

Il s'agit de créer le jeu d'instructions du séquenceur. On distinguera d'un part les **instructions de séquencement** et, d'autre part, les **instructions d'actions**. Dans le cas étudié, la structure de l'algorithme est très simple. On limite les modes de séquencement au **saut conditionnel**. Pour plus de souplesse dans l'écriture du programme, on prévoit toutefois le saut sur condition vraie et le saut sur condition fausse. Pour les instructions d'actions, comme il n'y a pas de simultanéité d'actions dans l'algorithme, on se limite à un champ unique pour coder le numéro d'action.

Si l'on réexamine l'algorithme à implémenter, on constate que des actions interviennent à chaque pas d'horloge. On peut difficilement engendrer un tel comportement si l'on alterne des instructions de séquencement et des instructions d'action. On pourrait éventuellement s'affranchir de ce problème en synchronisant le séquenceur sur une horloge de fréquence multiple de celle qui anime la partie opérative. On préfère ici, par souci de simplicité, prendre une autre voie. Elle consiste à fusionner les instructions de séquencement et les instructions d'action. Ainsi, chaque pas de programme peut exécuter une action tout en contrôlant la progression du compteur ordinal.

Pour traduire l'algorithme en un programme dédié au séquenceur que nous sommes en train de concevoir, nous définissons un **langage mnémonique**. A titre d'exemple :

```
instruction   ::=  [étiquette]   action, IF cdt. GOTO étiquette;
                 | [étiquette]   action, IF NOT(cdt.) GOTO étiquette;
action        ::=  NOP|NEM|SYNC|FANE|SIGN|DATA|LRC
cdt.          ::=  TRUE|VAFE|FCI
```

N. B.

- On constate l'existence d'une action nulle (NOP) qui permet au programmeur d'insérer éventuellement des lignes de code qui ne génèrent pas d'action.
- On constate également la présence de la condition TRUE (toujours vraie). Cet artifice permet d'engendrer, à peu de frais, le saut inconditionnel (IF TRUE GOTO) et surtout l'enchaînement séquentiel de base (IF NOT(TRUE) …).

- **Reformalisation de l'algorithme en langage mnémonique**

On rappelle ci-dessous l'algorithme spécifiant le comportement du séquenceur.

```
/*seq_émission(VAFE,FCI Bool
SYNC,FANE,SIGN,DATA,LRC,NEM Bool Out)*/
{
  while TRUE
  {
    while(not(VAFE)) NEM;
    SYNC;
    SYNC;
    FANE;
    SIGN;
    DATA;
    while(not(FCI)) DATA;
    LRC;
    NEM;
  }
}
```

Sa traduction dans le langage mnémonique retenu est la suivante :

```
/*seq_émission(VAFE,FCI Bool In;
 SYNC,FANE,SIGN,DATA,LRC,NEM Bool Out)*/
debut   NEM,IF NOT(VAFE) GOTO debut;
        SYNC,IF NOT(TRUE) GOTO ANYWHERE;
        SYNC,IF NOT(TRUE) GOTO ANYWHERE;
        FANE,IF NOT(TRUE) GOTO ANYWHERE;
        SIGN,IF NOT(TRUE) GOTO ANYWHERE;
encore  DATA,IF NOT(FCI) GOTO encore;
        LRC,IF TRUE GOTO debut;
```

- **Affectation des E/S du séquenceur, définition de la taille de l'adresse des instructions et de la structure de l'instruction**

Il faut à présent matérialiser le séquenceur. Ceci consiste tout d'abord à interconnecter le multiplexeur de conditions et le décodeur d'actions avec l'environnement opérationnel.
Il y a 3 entrées TRUE, VAFE et FCI à connecter au multiplexeur. On les alloue dans l'ordre suivant : (TRUE $\rightarrow$ E0, VAFE $\rightarrow$ E1 et FCI $\rightarrow$ E2). Il y a 7 sorties (NOP, NEM, SYNC, FANE, SIGN, DATA et LRC) à prélever sur le décodeur. On les câble dans cet ordre sur les sorties Yi (i = 0 … 6) du décodeur.

Il faut ensuite définir les caractéristiques de la mémoire de programme et du compteur ordinal. Le programme compte 8 lignes. Il faut donc que la mémoire ait une capacité d'au moins 8 mots et le compteur ordinal de 3 bits.

Il reste enfin à coder (« en dur ») l'instruction :

- Pour le champ de séquencement, 1 bit est nécessaire à la polarité de la condition, 2 bits pour la sélection de la condition (1 parmi 3) et 3 bits pour le codage de l'adresse de saut. Soit respectivement P, <C1, C0>, <D2 … D0>.
- Pour coder le numéro d'action, 3 bits sont nécessaires (1 parmi 7) soit <A2 … A0>.

On obtient donc la structure suivante :

P	C1	C0	D2	D1	D0	A2	A1	A0

On en déduit que la taille du mot mémoire doit être supérieure ou égale à 9 bits.

- **Établissement du schéma logique du séquenceur**

A ce stade, les caractéristiques des composants et leur câblage sont complètement définis, on peut donc tracer le logigramme du séquenceur.

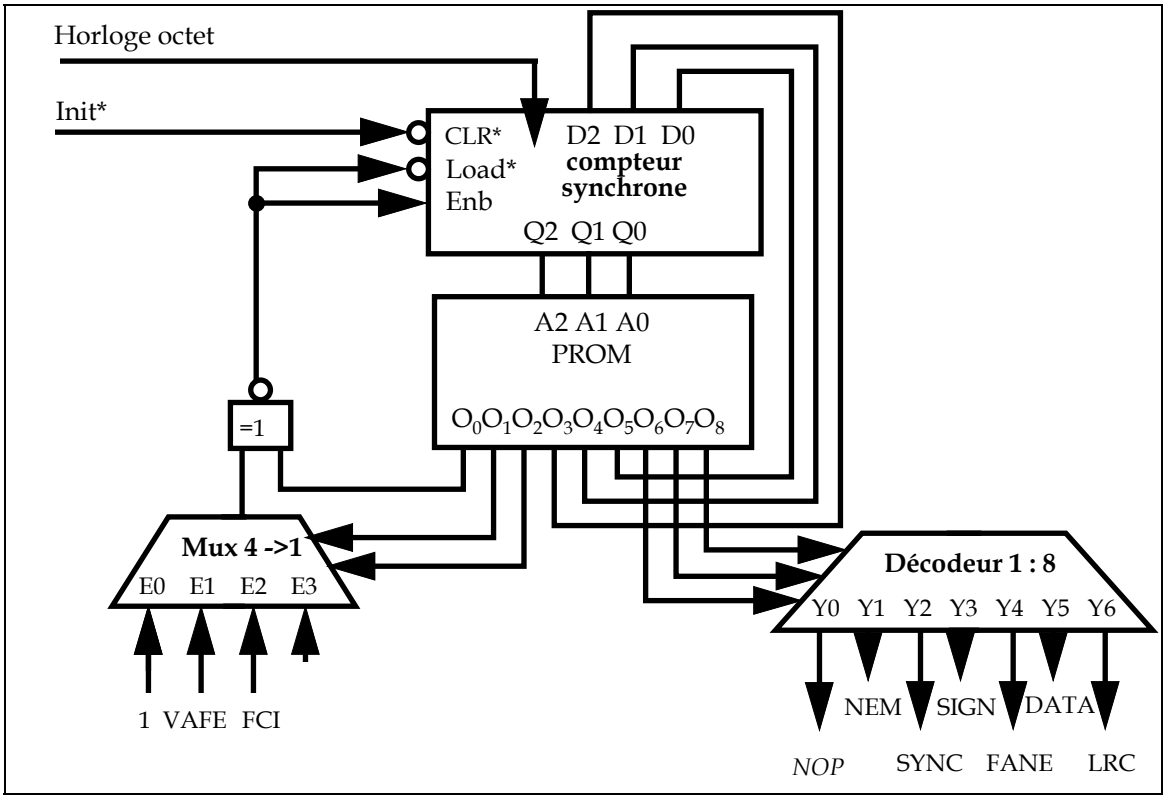


figure 6.23 : schéma logique du séquenceur microprogrammé

Il reste à élaborer le programme en code binaire qui doit être chargé dans la mémoire PROM.

- **Assemblage du code mnémonique pour produire le code exécutable**

Cette opération s'effectue en deux passes. Tout d'abord, en affectant une adresse d'implantation à chaque instruction du programme. On substitue donc aux étiquettes des adresses.

```
/*seq_émission(VAFE,FCI Bool In;
 SYNC,FANE,SIGN,DATA,LRC,NEM Bool Out)*/
00  NEM,IF NOT(VAFE) GOTO 00;
01  SYNC,IF NOT(TRUE) GOTO xx;
02  SYNC,IF NOT(TRUE) GOTO xx;
03  FANE,IF NOT(TRUE) GOTO xx;
04  SIGN,IF NOT(TRUE) GOTO xx;
05  DATA,IF NOT(TRUE) GOTO xx;
06  DATA,IF NOT(FCI) GOTO 06;
07  LRC,IF TRUE GOTO 00;
```

Il faut ensuite générer le code binaire correspondant au programme à partir de la structure qui a été retenue pour l'instruction.

(CO)	P	C_1C_0	$D_2D_1D_0$	$A_2A_1A_0$
000	1	01	000	001
001	1	00	XXX	010
010	1	00	XXX	010
011	1	00	XXX	011
100	1	00	XXX	100
101	1	00	XXX	101
110	1	10	110	101
111	0	00	000	110

Ce tableau doit être saisi sous la forme d'un fichier informatique. Ce fichier est ensuite utilisé par un dispositif de programmation de mémoire afin d'être implanté dans le composant PROM qui a été choisi pour réaliser le séquenceur.

6. Bibliographie

[CV86] M. Courvoisier et R. Valette, *Commande des procédés discontinus, logique séquentielle*, Collection Dunod-Université, Dunod, 1986.

[BH82a] J.M. Bernard et J. Hugon, *De la logique câblée aux microprocesseurs, tome 3 : méthodes de conception des systèmes*, Collection technique et scientifique des télécommunications, Eyrolles, 1982.

[BH82b] J.M. Bernard et J. Hugon, *De la logique câblée aux microprocesseurs, tome 4 : application des méthodes de synthèse*, Collection technique et scientifique des télécommunications, Eyrolles, 1982.

[BH90] J.M. Bernard et J. Hugon, *Pratique des circuits logiques*, Collection technique et scientifique des télécommunications, Eyrolles, 1990.

[Gr86] D. Green, *Modern logic design*, Electronic Systems Engineering Series, Addison-Wesley, 1986.