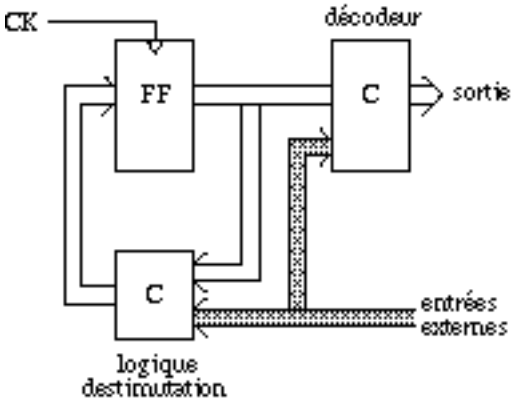
CHAPITRE 7
MACHINE D'ÉTAT SYNCHRONE

Chap. 6 – Circuit sans entrée seulement l'horloge = Machine de MOORE
Chap. 7 – Circuit avec entrées asynchrones
Ex : Load, enable, clear, direction, entrées
– Circuit avec horloge commune et avec un nombre fini d'états "Finite State Machine".

7.1 Machine de Mealy

Circuit dont la sortie dépend de son état interne présent + influence externe.

- On ajoute d'autres entrées

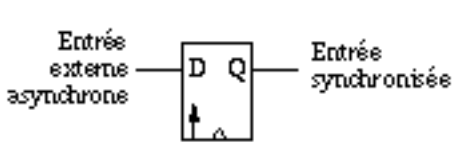


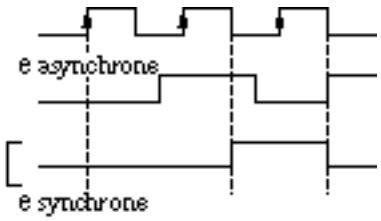
C = logique combinatoire

N.B : On peut aussi avoir une Machine de MEALY sans décodeur de sortie.

- Comme les entrées peuvent changer à tout moment et comme les entrées affectent la logique combinatoire de sortie ⇒ sorties peuvent changer à tout moment.

- Problème des entrées asynchrones à cause du Set-up time.
Certains signaux peuvent causer des perturbations ou être manqués ⇒ On ajoute un circuit de synchronisation avec Tset-up } très faible
Thold } FF rapide





Un des critères de design du circuit de synchronisation : éviter les états métastables, par exemple en ayant 2 étages au circuit de synchronisation.

- Peut-être bon d'employer des FF "D" discrets plutôt que des FF contenus dans les PALs meilleurs spécifications.

Étapes d'analyse (circuit séquentiel synchrone, i.e. avec entrées)

1. Identifier les éléments de la machine de Mealy (décodeur d'état suivant, éléments de mémoire – FF JK, T, D -, décodeur de sortie).
2. Écrire les équations booléennes pour chaque élément de mémoire (FF JK, T, D).
3. Établir la table de stimulation État-Présent ⇒ État Suivant et les sorties.
4. Dessiner le diagramme d'état.

Étapes de design (circuit séquentiel synchrone, i.e. avec entrées)

1. Obtenir la spécification, par exemple sous la forme d'un diagramme d'état.
2. Codage des états, réduction de ceux-ci.
3. Établir la table de stimulation État-Présent ⇒ État Suivant incluant les entrées externes
4. Établir les tables de Karnaugh pour les FF D, JK, T.
5. Synthétiser le décodeur de sortie (tables de Karnaugh).
6. Faire le schéma complet.

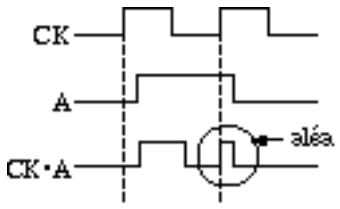
Règles de synchronisme

1. Éléments de mémorisation D $^{et}\!/_{ou}$ J-K et registre $^{et}\!/_{ou}$
   Compteur : horloge commune à tous les éléments

2. Entrées externes doivent être amorties puis synchronisées avec l'horloge.

3. Ne pas créer de signal qui serait fonction de l'horloge

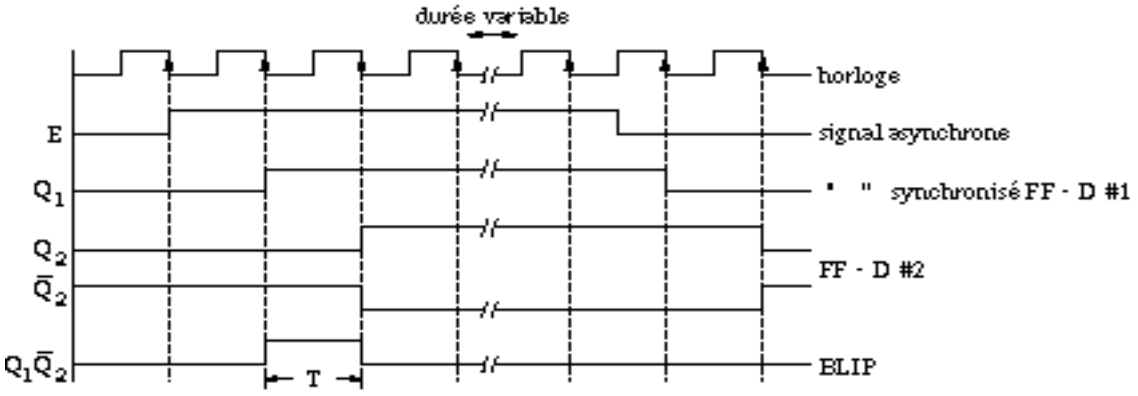
   Ex : CK · A ? retard des signaux synchrone
   A = signal synchrone

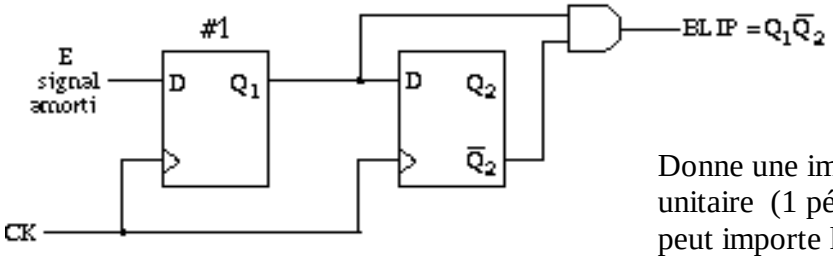


4. Ne pas employer les entrées asynchrones des éléments synchrones (ex : PR ou CLR) sauf pour initialiser.

Réalisation d'un signal de commande impulsionnel appelé "BLIP"

Utile pour synchroniser les signaux d'entrées



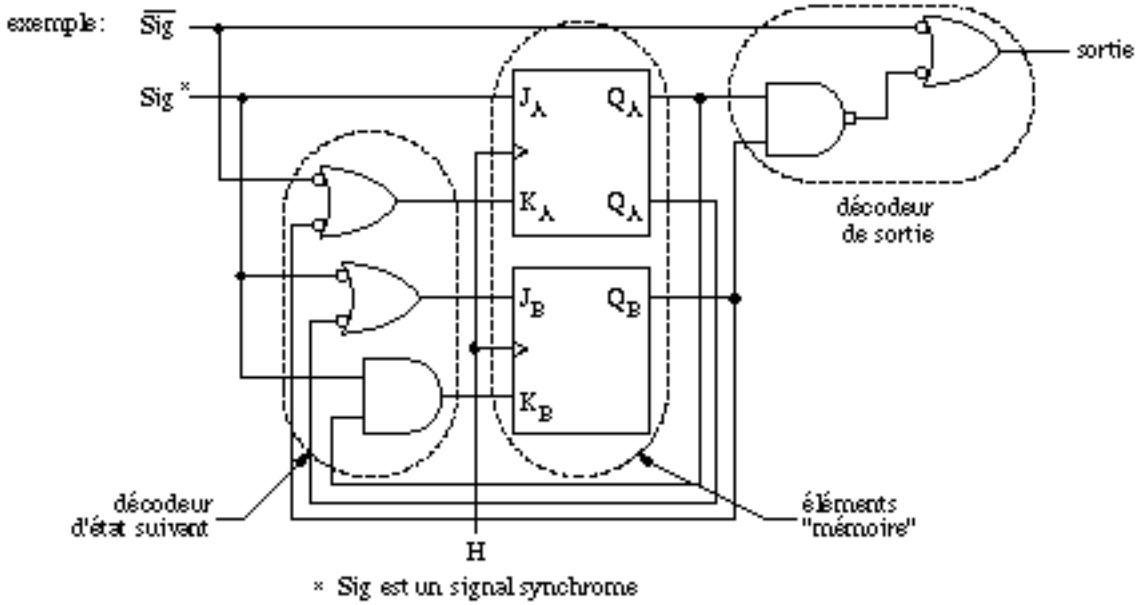


Donne une impulsion d'une durée unitaire (1 période d'horloge), peut importe la durée du signal.

E aura été au préalable amorti.

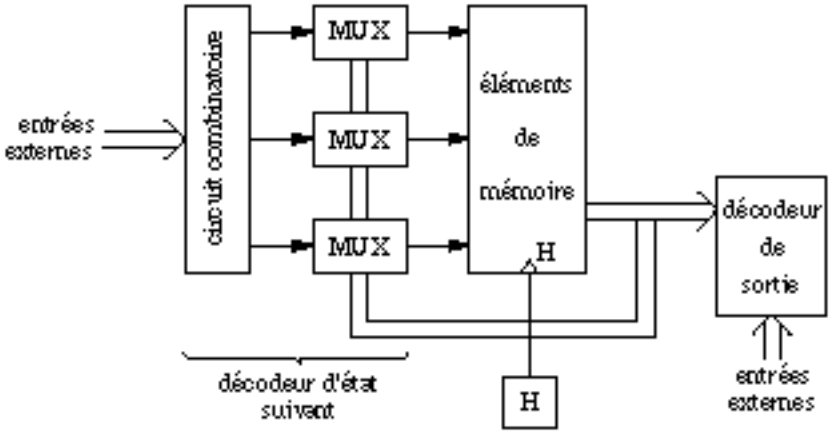
Exercice (sera fait en classe)

Déterminer le diagramme d'état du séquenceur suivant.



Synthèse avec MSI et LSI

On peut simplifier le problème par l'emploi de circuits plus complexes : MUX, PLA, PAL

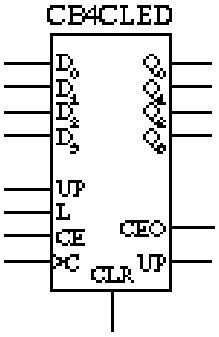


Ici les MUX sont directement adressés

- Sortie MUX = commande des FF
- Adresses MUX = bit de codage de l'état
- Entrées MUX = condition de branchement

Types de compteur dans Xilinx :

74160	BCD	LOAD	CLR asynchrone	EN	RCO
74161	Binaire				
74162	BCD		synchrone Reset	EN	RCO
74163	Binaire				
75168	BCD	load bidirectionnel	----	EN	$\overline{\text{RCO}}$

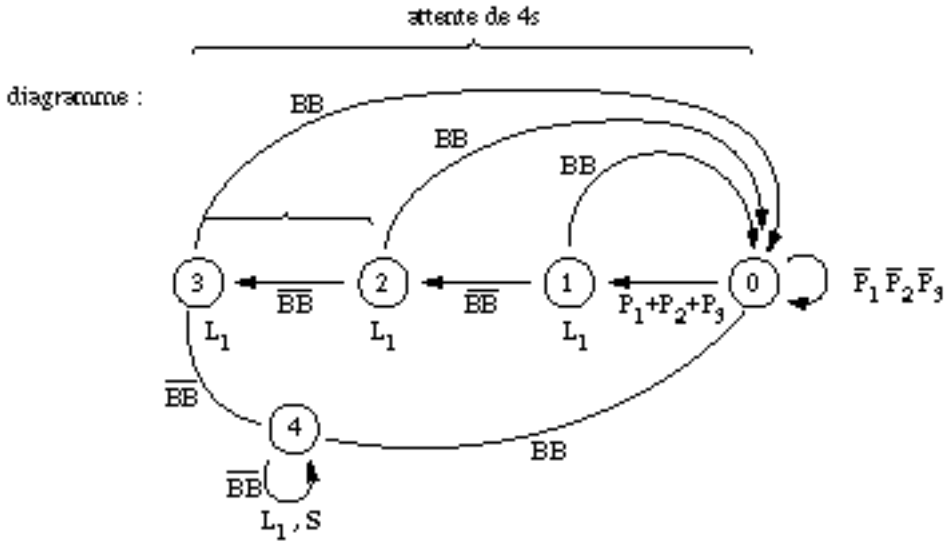


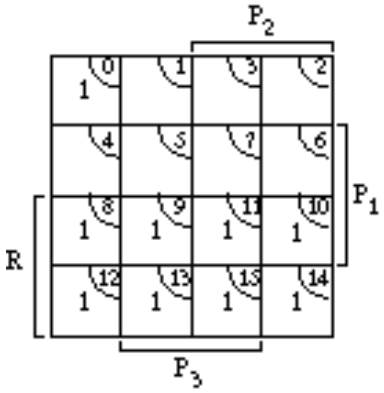
CB4CLED

- Clear – asynchrone, ne pas employer.
- U/D (Up/Down) – si CE = 1, +1 (incrémente) si à un, -1 (décrémente) si à 0.
- Load – charge valeur, synchrone, prioritaire sur CE.
- CE – Count Enable (comptage activé si à 1), ne compte pas si 0.
- CEO – Count Enable Output (indique les passages à 0000 ou à 1111 selon la valeur de U/D.

Exemple : Faire le design d'un système d'alarme simple.

- 3 entrées (portes, fenêtres) = P_1, P_2, P_3
- 2 sorties = sirène S, LED d'avertissement < 1 si alarme activée.
- délai de 4s entre la faute et le déclenchement LI allumée.
- remise à 0 si reset ou plus de fautes
- employer CB4CLED





$$BP = R + \overline{R}\,\overline{P}_1\overline{P}_2\overline{P}_3$$
$$= R + \overline{P}_1\overline{P}_2\overline{P}_3$$

État 0

P1	P_2	P_3	$P_1 + P_2 + P_3$	$\overline{P}_1 + \overline{P}_2 + \overline{P}_3$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Alarme
Attente

Autres États = 1,2,3,4 Retour

	R	P1	P2	P3	AA	BB
0)	0	0	0	0	0	1
1)	0	0	0	1	1	0
2)	0	0	1	0	1	0
3)	0	0	1	1	1	0
4)	0	1	0	0	1	0
5)	0	1	0	1	1	0
6)	0	1	1	0	1	0
7)	0	1	1	1	1	0
8)	1	0	0	0	0	1
9)	1	0	0	1	0	1
10)	1	0	1	0	0	1
11)	1	0	1	1	0	1
12)	1	1	0	0	0	1
13)	1	1	0	1	0	1
14)	1	1	1	0	0	1
15)	1	1	1	1	0	1

Alarme terminée, conditions plus remplies

Fin Car Reset

$BB = R + \overline{P}_1\overline{P}_2\overline{P}_3$

$AA = \overline{BB}$

Alarme se poursuit car conditions toujours présentes

Fin de l'alarme

Actions synchrones disponibles

On a ici 5 états $\Rightarrow Q_2Q_1Q_0$

L – Load
CE – compte
[C – clear pas synchrone]
U – up/down

Table des actions :

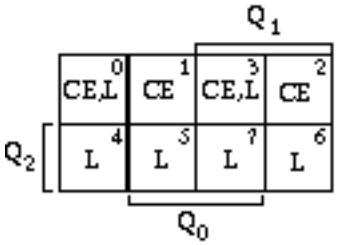
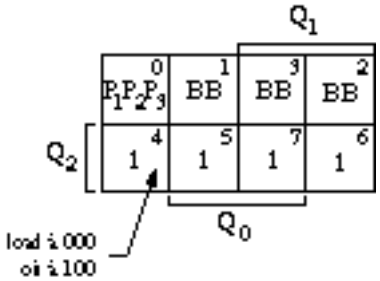
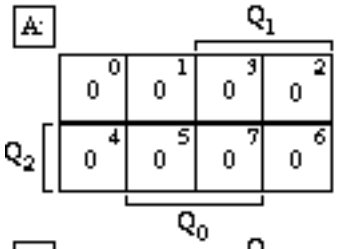


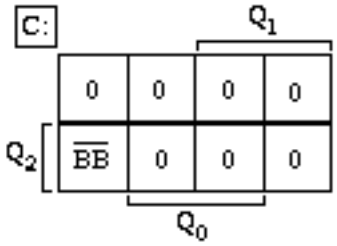
Table Load

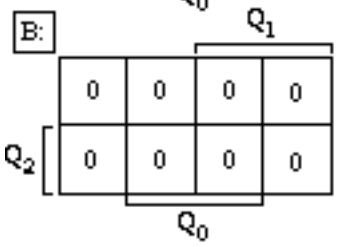


- États 5,6,7 retour à 0 avec load actif.
- Plus simple U toujours à 1.
- Clear toujours à f, plus simple.
- CE toujours à 1 car load prioritaire.

Table des datas A,B,C

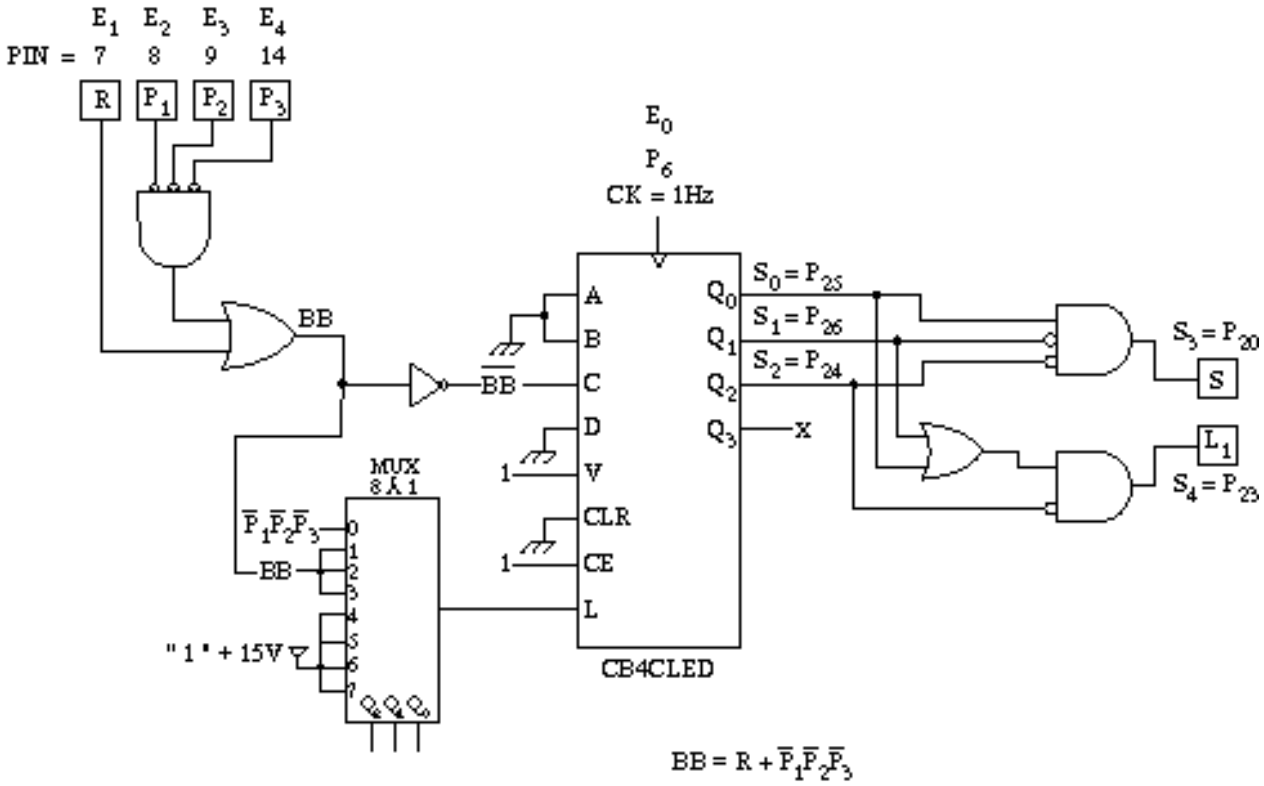




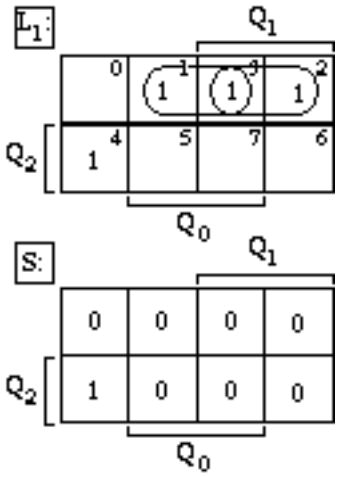


	C	B	A	
À 4 jump à :	0	0	0	← BB
	1	0	0	← $\overline{BB}$

Le schéma final est alors :



Synthèse des sorties

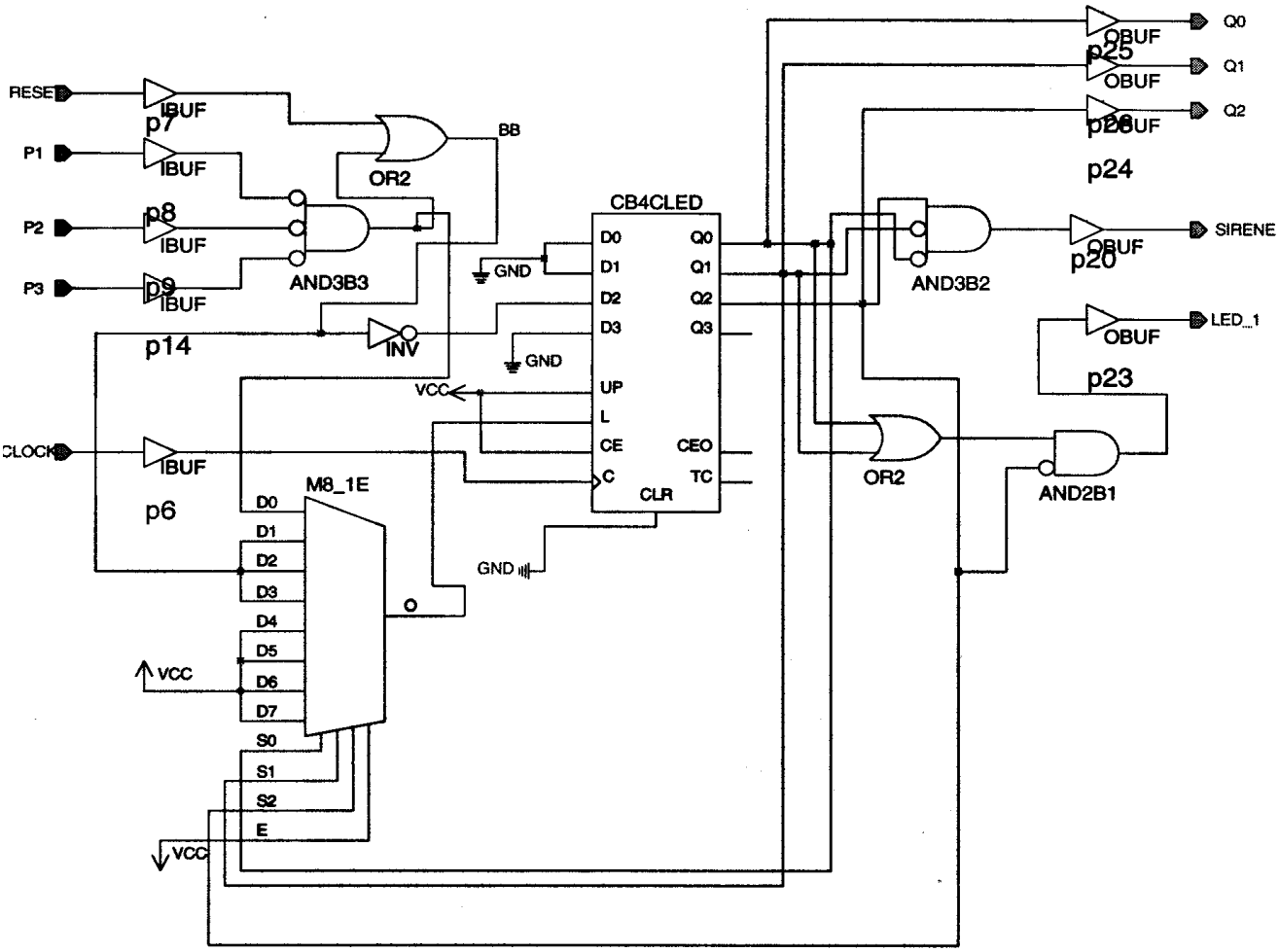


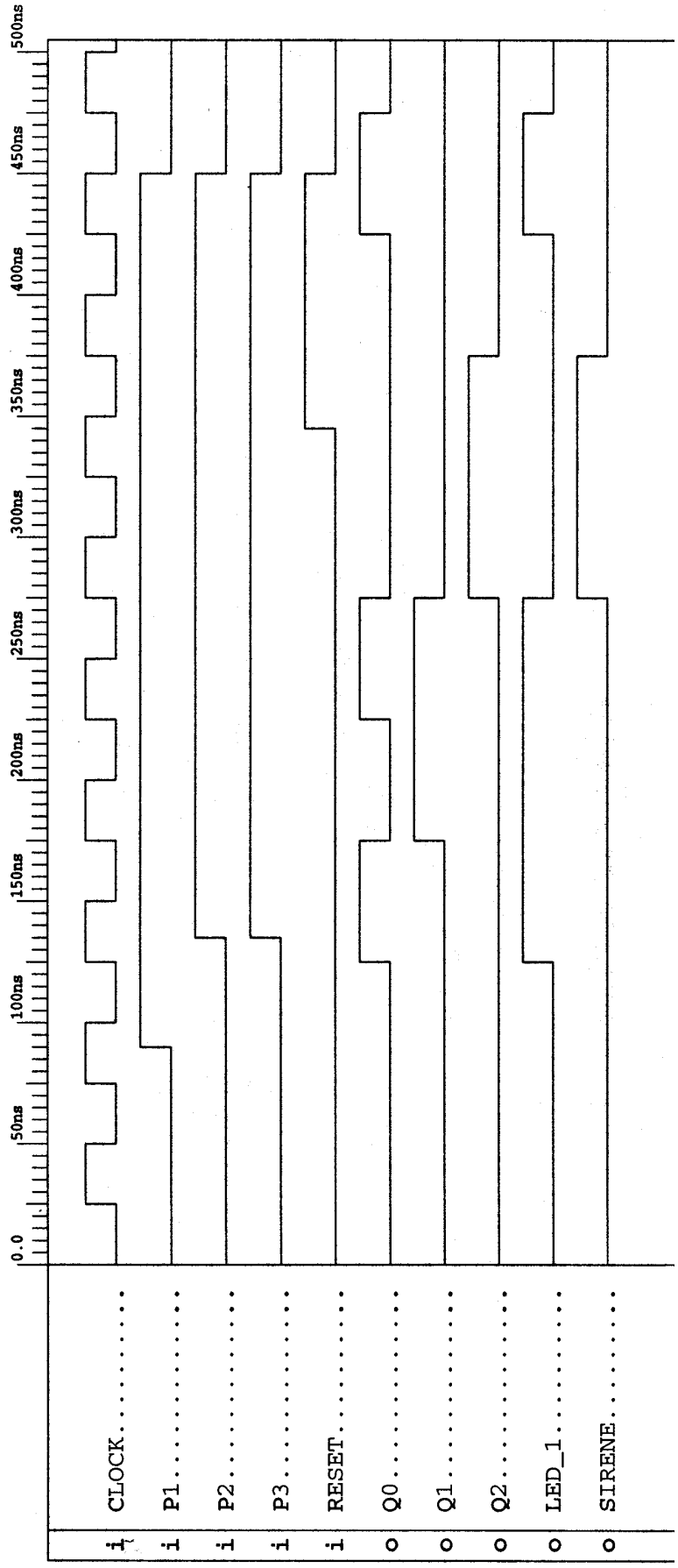
$$L_1 = Q_1\overline{Q}_2 + \overline{Q}_2 Q_0 = \overline{Q}_2\ (Q_1 + Q_0)$$

* Devrait être mis à 1 car autrement LED1 s'éteint lorsque sirène à 1 (état 4).

$$S = Q_2\overline{Q}_1\overline{Q}_0$$

<u>Schémas et simulation dans Xilinx</u> :




0.0
50ns
100ns
150ns
200ns
250ns
300ns
350ns
400ns
450ns
500ns
i CLOCK
i P1
i P2
i P3
i RESET
o Q0
o Q1
o Q2
o LED_1
o SIRENE

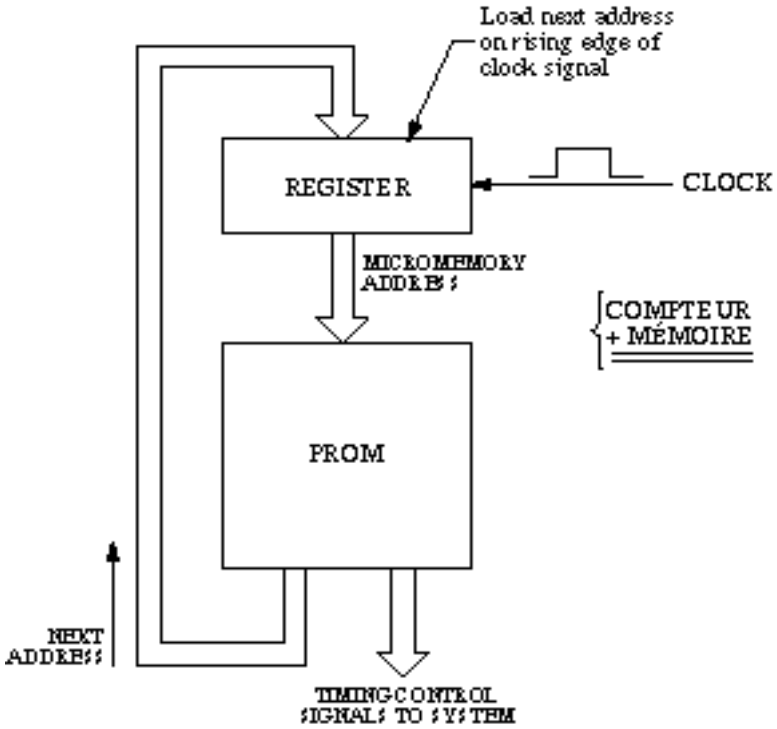
Comme on le voit ici, cette approche de design devient vite compliquée. De façon à avoir une approche plus simple, ie
- modifiable aisément (programmable),
- plus universelle d'un design à l'autre,

On doit penser à une autre approche.

Note : La même approche de design est possible avec registre à décalage plutôt. (Si le codage des états est "décalé".

Microprogrammation

- Jusqu'à présent on a travaillé en logique câblée, le moindre changement exige un renouvellement du design, par contre le fonctionnement est très rapide car il n'y a aucun décodage.
- La microprogrammation est une autre approche qui permet d'implanter un séquenceur universel programmable. Au moyen d'un jeu de *macro-instructions* on peut programmer un ensemble de périphériques. Le jeu de *macro-instructions* est lui-même définissable par microprogrammation à l'aide de *micro-instructions*. L'idée de base est celle du compteur + mémoire avec des embellissements pour : interruptions, pile (stack), sauts, sous-routines, boucles, etc.
- Exemple de *macro-instructions* : ADD A,B *macro-instruction* qui fait l'addition du contenu des deux registres A et B : A < - A+B
- Exemple de *micro-instruction* : L'ensemble des opérations nécessaires pour implanter ADD A,B.



Simplest control unit implementation

Il nous faut d’abord un "séquenceur universel"

On reprend l’idée du ROM + compteur du chapitre 6.

Ceci sera vu dans le cours GIF-10279 – Microprogrammation.

Voici déjà un schéma de ce système :