Technique Numérique

Les bases

BFH-TI-Biel/Bienne

(Version v3.0)

Dr. Marcel Jacomet et Dr. Theo Kluter

10 septembre 2013

Table des matières

Remerciement

Le cours Technique Numérique de la BFH-TI-Biel/Bienne a été développé par Marcel Jacomet et donné pour la première fois en 1993. Entre temps Michael Höckel, Michael Filiol et Roland Schaefer ont utilisé le cours et les deux premiers ont réalisé la traduction en francais pour laquelle nous aimerons les remercier. La traduction française a été corrigée par Onesime Tamoh Gounoue et Benjamin Kern.

1 Introduction

But : Introduction aux techniques numériques (circuits numériques combinatoires et séquentiels). Les étudiants doivent être capables d'analyser et de concevoir seuls des circuits numériques simples. Le cours servira de base pour d'autres cours tels que microélectronique (VLSI-DESIGN), et traitement numérique du signal (DSP), etc. Prestations attendues : acquérir et maîtriser la matière à partir du cours et de la littérature. Résoudre des exercices de manière indépendante.

1.1 Conception

Description du script :

1. Introduction
2. Algèbre booléenne et portes logiques : constantes et variables booléennes, fonctions de base, fonctions spéciales, analyse de circuits logiques, formes canoniques conjonctives et disjonctives, simplification et optimisation des équations par la méthode de Karnaugh et Quine/McCluskey.
3. Bascules : bascule bistable élémentaires (latch), latch à entrées synchrones et asynchrones, comportement dans le temps, commande par niveau ou par flanc, flip-flop (bascule bistable), monoflop (bascule monostable), multivibrateur (bascule astable), aléas.
4. Séquenceur : bases des automates à états finis (machine de mealy, de moore, de medwedjew), tableaux de transition et de sortie, diagramme d'état, analyse et synthèse des séquenceurs, états parasites, aléas et synchronisation des entrées.
5. Nombres : Systèmes de numération, transformation de nombres décimaux, arithmétique binaire (addition, soustraction, multiplication, division).
6. Codes : binaire codé décimal, code cyclique, détection d'erreurs, correction d'erreurs (Hamming), codes alphanumériques.

1.2 Literature et Links

Vous trouverez tous les documents du cours sur la site Web du MicroLab (laboratoire micro-électronique) : www.microlab.ch. La littérature recommendée est à la fin du document .

1.3 Qu'est-ce que la technique numérique ?

La technique numérique s'occupe de la représentation de grandeurs par des nombres. Expressions :

- *digit* : chiffre, nombre
- *numérique* : avec des nombres ou des chiffres

Les domaines :

- technique de commande numérique
- technique de réglage numérique
- technique de mesure numérique
- technique de transmission numérique

1.4 Analogique versus Digital

Les circuits et systèmes analogiques traitent des signaux (tension, courant ou autres) qui varient de manière continue dans le temps, c'est à dire qui peuvent prendre n'importe quelle valeur entre deux extrêmes. Les circuits digitaux quant à eux traitent des signaux discrets qui, à chaque instant, ne peuvent prendre que deux valeurs distinctes notées 1 ou 0 (HIGH ou LOW, FALSE ou TRUE).

1.5 Les avantages de la technique numérique

- Reproductibilité : les circuits analogiques sont sensibles à des influences internes ou externes comme par exemple la température, le vieillissement des composants, le bruit etc.
- Conception simple : les circuits peuvent facilement être divisés en blocs, réutilisés ou redimensionnés (IC-Design).
- Sauvegarde et traitement de signaux simples et sans perte de précision.

- Electronique de puissance : il suffit d'un simple interrupteur commandé en tant qu'amplificateur (conception simple, pertes faibles).
- Flexibilité et fonctionnalité :
  - Pour combiner, comparer et classifier on utilise des circuits logiques simples.
  - Les problèmes mathématiques sont résolus numériquement par l'utilisation de 4 opérateurs élémentaires (plus particulièrement par addition et multiplication).
  - Certains problèmes de traitement du signal ne peuvent être résolus que de manière numérique.
- Plusieurs problèmes peuvent être résolus à l'aide d'un même montage, configuré ou programmé différemment.
- Des mémoires *longue durée* avec des temps d'accès rapides sont faciles à réaliser.
- La précision d'un système peut être améliorée (et donc adaptée au problème) par simple augmentation du nombre de bits utilisés. Pour représenter les nombres il est facile de transposer un système dans une nouvelle technologie (IC).

2 Algèbre de Boole, portes et circuits logiques

Buts : Après avoir étudié ce chapitre, l'etudiant devra maîtriser les notions suivantes et/ou etre capable de répondre aux questions suivantes :

- Les différences entre l'algèbre de Boole et l'algèbre classique.
- Les fonctions de base et les fonctions spéciales de l'algèbre de Boole.
- Appliquer les règles de calcul de l'algèbre de Boole.
- Analyser des circuits logiques combinatoires quelconques.
- Sur la base des tables de vérité dériver les formes canoniques disjonctive et conjonctive des circuits.
- Simplifier des circuits par des méthodes graphiques et algorithmiques.

2.1 Les constantes et variables de Boole

En technique numérique on utilise des éléments qui ont un comportement binaire. Les signaux de sortie de ces éléments numériques peuvent seulement prendre deux états différents. Le mathématicien George Boole a développé en 1847 sur la base de cette bivalence la théorie de l'algèbre de Boole. Plus tard Shannon a amélioré cette théorie et l'a utilisée pour le calcul de circuits logiques. L'algèbre de Boole diffère de l'arithmétique conventionnelle par ses constantes et variables qui peuvent seulement prendre les deux valeurs 0 et 1.

2.2 Les fonctions de base de l'algèbre de Boole

Tout comme dans l'algèbre usuelle, les relations entre variables et constantes sont définies par des fonctions. Cependant, il n' existe pas, dans l'algèbre de Boole, des fonctions continues. On attribue des paires de valeurs discrètes. Une

telle atribution peut être définie sur une table de vérité. Dans une table de vérité toutes les combinaisons des variables sont indiquées, et pour chaque combinaison il existe une valeur de la fonction. La figure 2.1 montre une table de vérité pour

E_1	E_1	Y
0	0	1
0	1	1
1	0	0
1	1	1

TABLE 2.1 – Table de vérité

les variables d'entrée E_1 et E_2 et une variable de sortie Y. La table de vérité est l'un des instruments utilisés pour définir une fonction. Un autre instrument est l'équation de Boole, qui peut être utilisée comme une équations algébriques pour des calculs. Comme expliqué ci-dessus, les fonctions peuvent être définies grâce à des tables de vérité ou des équations de Boole. Mais quelles sont les fonctions dans l'algèbre de Boole ? On demontre facilement qu'avec 1 variable, on peut definir $2^2 = 4$ fonctions, avec deux variables $2^{(2^2)} = 16$ fonctions. On peut ainsi definr une multitude de fonction en utilisant un nombre reduit de fonction de base. Dans l'arithmétique booléene, on distingue trois fonctions de base :

- La conjonction ou multiplication logique (AND logique), symboles : $\cdot$ ou $\wedge$
- La disjonction ou addition logique (OR logique), symbole : $+$ ou $\vee$
- Le complement ou addition logique (OR logique), symboles : $\neg$ oder $^-$

En principe, il suffit d'une combinaison de complément et de disjonction ou de complément et de conjonction pour décrire chaque fonction de Boole. Ci-dessous, sont décrites les trois fonctions de base sous forme d'équations de Boole, des tables de vérité et de leurs symboles logiques :

2.3 Les fonctions spéciales de l'algèbre de Boole

En utilisant deux variables d'entrées on peut définir dans l'algèbre de Boole 16 fonctions différentes. En plus des fonctions de base déjà mentionnées, il existe

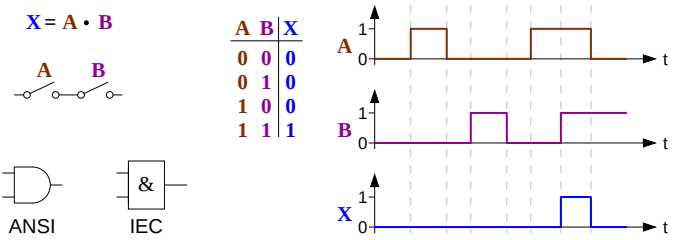


FIGURE 2.1 – Conjonction : Porte logique AND

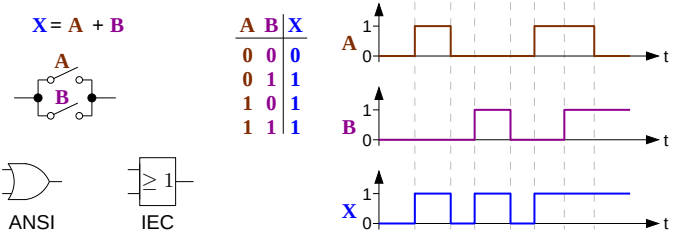


FIGURE 2.2 – La disjonction : Porte logique OR

d'autres fonctions de Boole tels que l'équivalence et l'antivalence.

2.4 Analyse des circuits logiques

Si on connaît le schéma logique ou bien l'équation de Boole d'un circuit, on peut déterminer sa valeur de sortie par rapport aux valeurs d'entrées. Comme exemple analysons le schéma du circuit de la figure 2.6. On doit déterminer pour

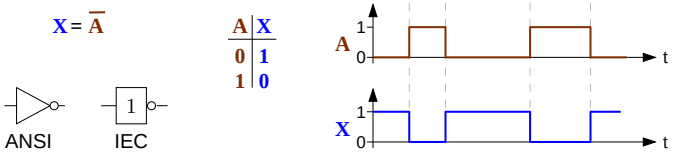


FIGURE 2.3 – Le complément : Porte logique NOT

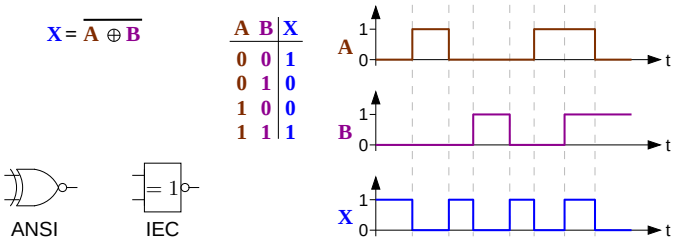


FIGURE 2.4 – L'équivalence : identité pour plusieurs entrée (pour deux entrées : porte exclusif NOR).

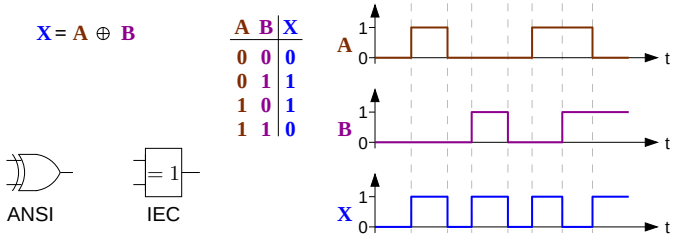


FIGURE 2.5 – L'antivalence : non-identité pour plusieurs entrées (pour deux entrées : porte exclusif OR).

le vecteur d'entrée choisi $(A, B, C, D) = (0, 1, 1, 0)$ la valeur de sortie. On trouve très facilement la fonction du circuit sous forme d'une équation de Boole ou de table de vérité.

2.5 Formes canoniques disjonctives et conjonctives

En algèbre, on peut décrire des fonctions par différentes équations. Les équations peuvent être développées, ou factorisées. On peut aussi faire des transformations similaires avec des équations de Boole. Un mode de représentation des équations de Boole est la forme canonique disjonctive et conjonctive. On parle d'une forme canonique disjonctive lorsqu'on a une fonction disjonctive entre des termes conjonctifs, et de forme canonique conjonctive lorsqu'on a une fonction conjonctive entre des termes disjonctifs. On doit dériver systématiquement les formes canon-

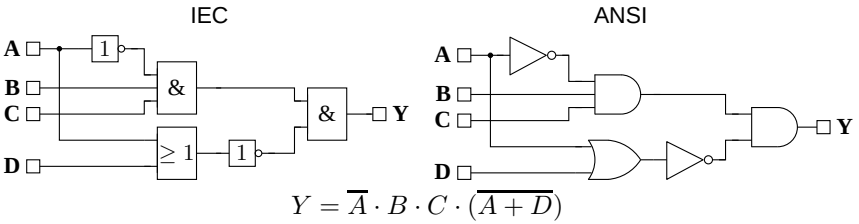


FIGURE 2.6 – Exemple d'un circuit logique.

iques pour deux équations de Boole en utilisant ses tables de vérité : Les formes

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Minterm

$X = ABC \vee \bar{A}\bar{B}C$

A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Maxterm

$Y = (\bar{A} \vee B \vee C)(\bar{A} \vee \bar{B} \vee C)$

FIGURE 2.7 – Formes canoniques disjonctive (gauche) et conjonctive (droite).

canoniques disjonctives et conjonctives qui en résultent sont très utile pour la simplification des équations logiques. Si on laisse les équations de Boole dans ces formes standardisées, il en résulte une logique en deux étages (figure 2.8).

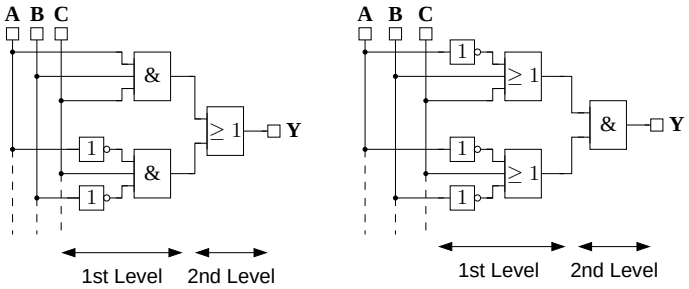


FIGURE 2.8 – Logique en deux étages (PLA).

2.6 Règles de calcul de l'agèbre de Boole.

Les règles les plus importantes sont rassemblées sans commentaire dans le tableau 2.2.
En arithmétique Booléene, on se sert souvent de la loi de De Morgan. Avec celle-ci on peut transformer les portes AND en portes OR et vice-versa.

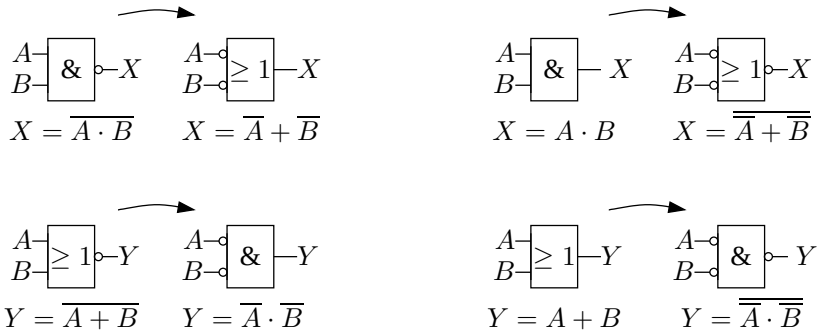


FIGURE 2.9 – La loi DeMorgan.

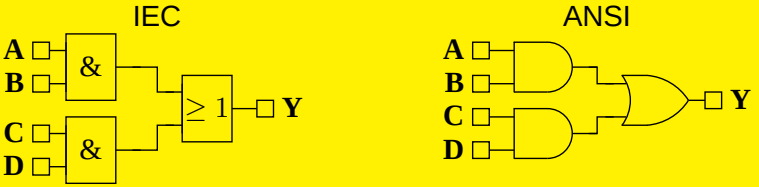
Les lois de DeMorgan sont un cas particulier du théorème de Shannon :

$$\overline{f(e_m, \overline{e_n}, \vee, \wedge, \oplus, \bar{\oplus}, \cdots)} = f(\overline{e_m}, e_n, \wedge, \vee, \bar{\oplus}, \oplus, \cdots)$$

Elémentaires	(1)	$A \cdot A$	=	A
	(2)	$A + A$	=	A
	(3)	$A \oplus A$	=	0
Constantes	(4)	$A \cdot 1$	=	A
	(5)	$A \cdot 0$	=	0
	(6)	$A + 1$	=	1
	(7)	$A + 0$	=	A
	(8)	$A \oplus 1$	=	$\overline{A}$
	(9)	$A \oplus 0$	=	A
Compléments	(10)	$A \cdot \bar{A}$	=	0
	(11)	$A + \bar{A}$	=	1
	(12)	$A \oplus \bar{A}$	=	1
	(13)	$\overline{\bar{A}}$	=	A
DeMorgan	(14)	$\overline{A \cdot B}$	=	$\bar{A} + \bar{B}$
	(15)	$\overline{A + B}$	=	$\bar{A} \cdot \bar{B}$
Commutative	(16)	$A \cdot B$	=	$B \cdot A$
	(17)	$A + B$	=	$B + A$
	(18)	$A \oplus B$	=	$B \oplus A$
Distributive	(19)	$A \cdot (B + C)$	=	$(A \cdot B) + (A \cdot C)$
	(20)	$A + (B \cdot C)$	=	$(A + B) \cdot (A + C)$
	(21)	$A \cdot (B \oplus C)$	=	$(A \cdot B) \oplus (A \cdot C)$
Associative	(22)	$A \cdot (B \cdot C)$	=	$(A \cdot B) \cdot C$
	(23)	$A + (B + C)$	=	$(A + B) + C$
	(24)	$A \oplus (B \oplus C)$	=	$(A \oplus B) \oplus C$
Absorption	(25)	$A \cdot (A + B)$	=	A
	(26)	$A + (A \cdot B)$	=	A

TABLE 2.2 – Règles de calcul de l'agèbre de Boole.

Exercise : Le circuit logique ci-dessous doit être modifié afin, d'utiliser seulement des portes logiques Nand :



2.7 Simplification et optimisation de fonctions

La forme canonique disjonctive et conjonctive est rarement la forme la plus courte et la plus simple d'une fonction de Boole. On peut souvent simplifier les fonctions et éliminer des variables. La réalisation des fonctions à partir d'équations simplifiées est plus facile que la réalisation à partir de leur forme canonique. Pour la simplification systématique on doit se baser sur la forme canonique disjonctive ou conjonctive. Il existe diverses méthodes de simplification. Deux méthodes seront traitées ci-après :

- la méthode graphique de Karnaugh et Veitch
- l'évaluation numérique de Quine et McCluskey

Dans chaqu'une des méthodes on ne peut simplifier que des fonctions ayant une seule variable de sortie. Des méthodes pour optimiser des fonctions ayant plusieurs variables de sorties se basent souvent sur les deux méthodes mentionnées ci-dessus.

2.7.1 Simplification par le diagramme de Karnaugh

La méthode de Karnaugh et Veitch est appropriée pour simplifier une fonction ayant jusqu'à 5 (ou 6 au maximum) variables d'entrées. Chaque ligne de la table de vérité est assignée une cellule du diagramme de Karnaugh (voir figure 2.10). Les mintermes jointifs peuvent être reunis dans groupes de 2^n. la règle en vigueur ici est la loi $AB \vee A\bar{B} = A$. Prenons comme exemple la fonction de Boole avec 2 variables d'entrée de la figure 2.11.

Pour la simplification avec le diagramme de Karnaugh on applique les règles suivantes :

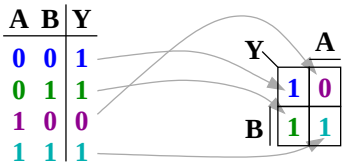
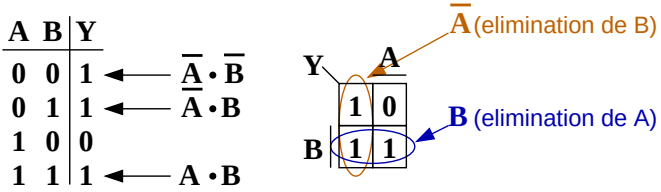


FIGURE 2.10 – Diagramme de Karnaugh.



$$Y = \bar{A}\bar{B} + \bar{A}B + AB = \bar{A} + B$$

FIGURE 2.11 – Simplification d'une fonction avec 2 variables d'entrées.

- On regroupe tous les 1 (ou 0) en prenant soin de faire le moins de regroupement possible.
- On commence par regrouper des cellules (2^n) comprenant le plus grand nombre de 1 (ou 0) possible.

On trouve ci après des exemples de simplification des fonctions de Boole avec plusieurs variables. Les graphiques des figures 2.12, 2.13 et 2.14 nous montrent des exemples de simplifications des fonctions avec 3, 4 ou 5 variables d'entrées. Pour l'exemple d'une fonction avec 5 variables d'entrées le circuit corespondant est donné ci après. (voire figure 2.15). Lors du regroupement, on cherche à mettre le plus grand nombre de 1 adjacents dans le même groupe. Dans certains cas, les 1 logiques sont distribués comme en losange (echiquier). Pour ces cas on ne trouve pas de possibilités de simplification. Cependant, si on n'utilise pas l'équation $AB \vee A\bar{B} = A$, (qui est la base des simplifications du tableau de karnaugh), mais plutôt la relation $A\bar{B} \vee \bar{A}B = A \oplus B$ ces echiquiers peuvent être simplifiés, comme c'est illustré à l'exemple de la figure 2.16.

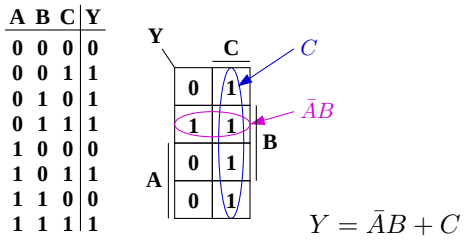


FIGURE 2.12 – Simplification d'une fonction avec 3 variables d'entrées.

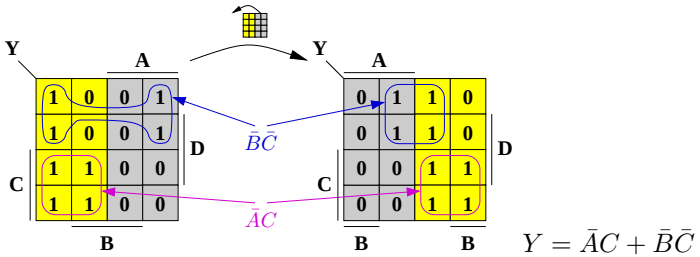


FIGURE 2.13 – Simplification d'une fonction avec 4 variables d'entrées.

2.7.2 Les fonctions incompletement définies

Lorsque dans une table de vérité toutes les combinaisons possibles des valeurs des variables ne sont pas définies (dont't care), on parle d'une table incomplètement définie. Les valeurs indéfinies ("-" ou "X"), ou *don't cares* peuvent prendre n'importe quelle valeur 1 ou 0. Ils augmentent les possibilités d'optimisation (voir figure 2.17).

2.7.3 La simplification par la méthode de Quine et McCluskey

La méthode de Quine et McCluskey est une procédure mathématique. On peut traiter des équations de Boole avec une variable de sortie. On compare les mintermes de la forme canonique disjonctive afin d'éliminer les variables de type $A \vee \bar{A} = 1$. Deux mintermes peuvent être simplifiés, seulement s'ils se distinguent d'une seule variable. Pour trouver ces relations, les mintermes sont composés en groupes de variables non inversées. Pour trouver des simplifications on compare

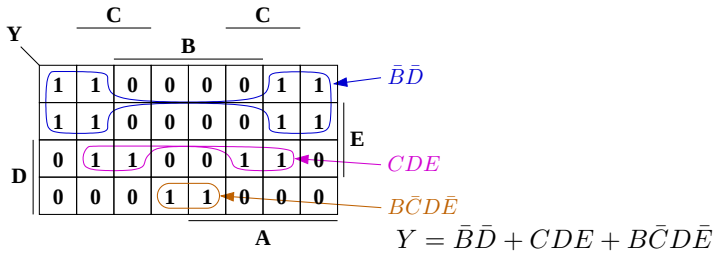


FIGURE 2.14 – Simplification d'une fonction avec 5 variables d'entrées.

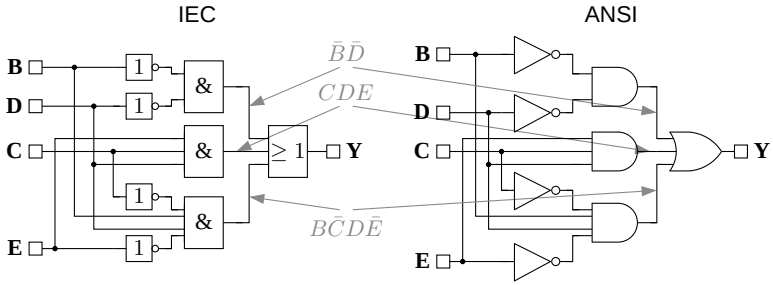


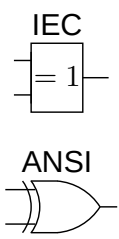
FIGURE 2.15 – Schema de l'exemple d'une fonction avec 5 (4) variables d'entrées.

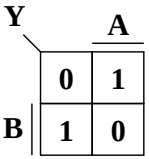
les groupes consécutifs. soit l'exemple de l'équation de Boole suivante :

$$\begin{aligned} Y \quad = \quad & \bar{A}\bar{B}C\bar{D} \vee \bar{A}\bar{B}CD \vee \bar{A}B\bar{C}\bar{D} \vee \bar{A}BC\bar{D} \\ & \vee A\bar{B}CD \vee AB\bar{C}\bar{D} \vee AB\bar{C}D \vee ABCD \end{aligned}$$

Le deux mintermes du "groupe 1" sont à comparer avec ceux du "groupe 2" (voir tableau 2.3). Si on a deux mintermes, qui se distinguent seulement par une variable, les deux mintermes sont à pointer, et le terme simplifié est inscrit dans la colonne 1ère simplification. Cette procédure est à continuer, jusqu'a ce qu'on ne puisse plus faire de simplification. Les mintermes non pointés sont appellés primtermes. La forme minimale de l'équation de Boole est la combinaison disjonctive de ces printermes, par lesquels tous les mintermes sont saisis. Pour trouver la fonction minimale de l'équation de Boole, on doit établir un nouveau tableau (tableau 2.4). Il montre la dépendance des mintermes des primtermes.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

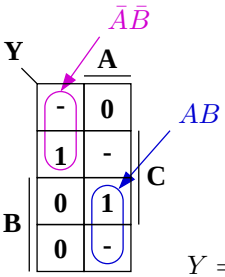




$$Y = A \oplus B$$

FIGURE 2.16 – Le modèle échiquier dans le diagramme de Karnaugh donne des portes EXNOR.

A	B	C	Y
0	0	0	-
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	-
1	1	0	-
1	1	1	1



$$Y = \bar{A}\bar{B} + AB = \overline{A \oplus B}$$

FIGURE 2.17 – Simplification dans un exemple avec des don't cares.

Pour notre exemple on trouve deux solutions de la forme minimale :

$$\begin{aligned} Y_1 = p_1 \vee p_3 \vee p_4 &= ACD \vee \bar{B}C \vee B\bar{C} \\ Y_2 = p_2 \vee p_3 \vee p_4 &= ABD \vee \bar{B}C \vee B\bar{C} \end{aligned}$$

groupe	forme canonique disjonctive	ok	1ère simplification	ok	2ème simplification	ok
1	$\bar{A}\bar{B}C\bar{D}$	ok	$\bar{A}\bar{B}C$	ok	$\bar{B}C$	p_3
	$\bar{A}B\bar{C}\bar{D}$	ok	$\bar{B}C\bar{D}$	ok	$B\bar{C}$	p_4
2	$\bar{A}\bar{B}CD$	ok	$\bar{A}B\bar{C}$	ok		
	$\bar{A}B\bar{C}D$	ok	$B\bar{C}\bar{D}$	ok		
	$A\bar{B}C\bar{D}$	ok	$\bar{B}CD$	ok		
	$AB\bar{C}\bar{D}$	ok	$B\bar{C}D$	ok		
3	$A\bar{B}CD$	ok	$A\bar{B}C$	ok		
	$AB\bar{C}D$	ok	$AB\bar{C}$	ok		
4	$ABCD$	ok	ACD	p_1		
			ABD	p_2		

TABLE 2.3 – Simplification de Quine et McCluskey

Primterme	$p_1 =$	$p_2 =$	$p_3 =$	$p_4 =$
Minterme	ACD	ABD	$\bar{B}C$	$B\bar{C}$
$\bar{A}\bar{B}C\bar{D}$			x	
$\bar{A}B\bar{C}\bar{D}$				x
$\bar{A}\bar{B}CD$			x	
$\bar{A}B\bar{C}D$				x
$A\bar{B}C\bar{D}$			x	
$AB\bar{C}\bar{D}$				x
$A\bar{B}CD$	x		x	
$AB\bar{C}D$		x		x
$ABCD$	x	x		

TABLE 2.4 – Tableau Minterm-Primterm.

3 Elements de Mémoire

Buts : Après l'étude de cet chapitre on devra maîtriser les notions et être capable de répondre de façon précise aux questions suivantes :

- Comportement et différentes applications des bascules asynchrones du type SR.
- Différences entre des entrées synchrones et asynchrones, commande par impulsions ou par transitions.
- Basules synchrones et asynchrones.
- Comportement et caractéristique dans le temps des éléments de mémoire avec le temps de setup, de hold et de retard ainsi que la largeur minimale d'impulsion de commande.
- Différences entre les bascules bistables, monostables et astables.
- Comportement des mémoires SR-, D-, JK-, T-, etc.
- Analyser les mises en application des master-slaves et des elements de mémoire à base d'un seul flip-flop.
- Comment déterminer les temps des monoflops et quels sont les différents types existant ?
- A quoi sont dû les aléas et comment peut-on les éviter et où est-ce qu'on doit les éviter ?

3.1 Introduction

Jusqu'ici nous n'avons parler que des circuits logiques dont les signaux de sortie dependent des états actuels des entrées. À present, nous allons etudier les bascules, dont les signaux de sortie dependent aussi des états précédent des entrées. Les bascules ont une mémoire. Elles peuvent donc mémoriser des données. On distingue en principe trois types : les bascules bistables, monostables et asta-

bles.

3.2 Les bascules élémentaires bistables

Deux inverseurs peuvent être utilis pour construire un circuit logique, qui ne peut changer d'état logique, une fois qu'il l'a revêtut. Cette structure permet de mémoriser 1 bit.

FIGURE 3.1 – Mémoire de 1 bit.

3.2.1 Les bascules asynchrones du type SR

S	R	Q	
0	0	Q	ancienne valeur mémorisée
0	1	0	reset
1	0	1	set
1	1	-	valeur aléatoire

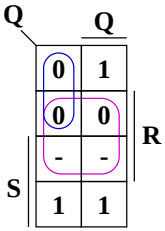


FIGURE 3.2 – Table de vérité d'un bascule asynchrones du type Set-Reset

Si on remplace les deux inverseurs par des portes NAND ou NOR, on obtient un élément de mémoire que l'on peut commander, dénommé le bascule asynchrones du type Set-Reset (Set-Reset Latch). La fonction de cet élément de mémoire peut être obtenue par la table de vérité et le diagramme de Karnaugh. L'équation de Boole simplifiée pour la sortie inversée est :

$$\overline{Q} = R + \bar{S}\,\overline{Q}$$

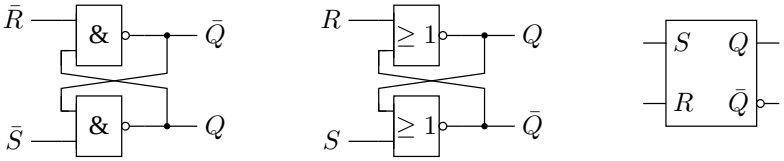


FIGURE 3.3 – Les schéma d'un bascules asynchrones du type SR construit par des portes NAND et NOR.

Ainsi on peut réaliser le bascules asynchrones du type SR avec deux portes NOR. Typiquement, pour un bascule asynchrones, un changement à l'entrée est transféré immédiatement à la sortie (bascules asynchronous transparent). Les entrées de ce bascule sont dites asynchrones. La figure 3.4 nous montre le comportement dans le temps d'un bascule asynchrones du type SR, construit à partir de portes NOR. On y reconnaît la dépendance directe des sorties et des entrées ainsi que l'occurrence d'une valeur de sortie indeterminée, si les deux entrées changent en même temps de 1 à 0. Les Bascules asynchrones du type SR sont les

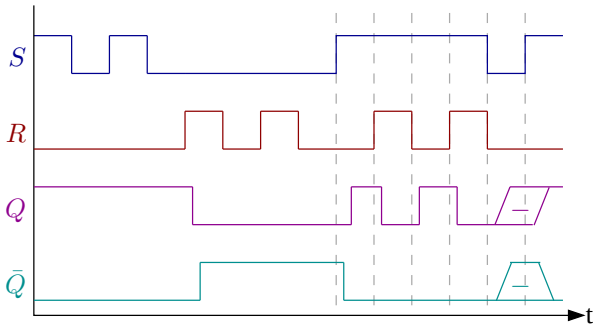


FIGURE 3.4 – Le comportement temporel d'un (NOR-NOR) bascule asynchrone.

éléments de base des éléments de mémoires plus complexes. Exemples d'utilisations : Normalement, les contacts mécaniques rebondissent quand on les fermes. On peut supprimer ces rebondissements avec des bascules asynchrones du type SR. Un autre bascule avec une entrée synchrone c'est le bascule asynchrones du type D. Les données de l'entrée D apparaissent aux sorties si le signal de contrôle $E = 1$. Si E devient égal 0, la dernière valeur sera mémorisée.

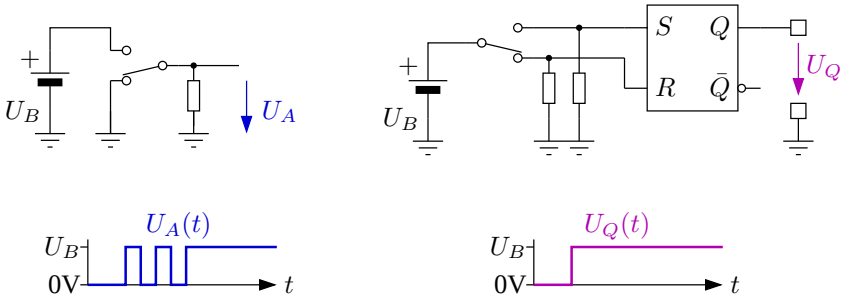


FIGURE 3.5 – Exemple pour un interrupteur sans rebond.

3.2.2 Bascules asynchrones à entrées synchrones

Il est souvent plus avantageux de contrôler l'accès des signaux d'entrées du bascule par des signaux d'activation (Gate, Strobe, Enable, Control Signal). Pour cela on peut disposer un réseau simple en tête d'un bascule asynchrone SR, qui contrôle les entrées set et reset. Si le signal $E = 0$, les deux entrées S et R ne peuvent pas changer l'état du bascule.

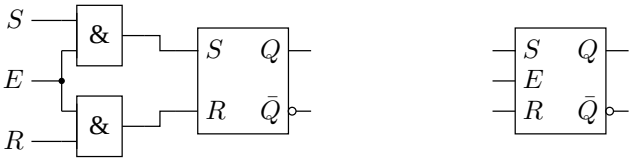


FIGURE 3.6 – Bascule asynchrone avec l'entrée de contrôle.

3.3 Comportement temporel

Pour assurer un fonctionnement correct d'un bascule asynchone il faut observer quelques précautions au niveau temporel :

Setup-time (t_{su}) : La durée minimale pendant laquelle les données d'entrées doivent être disponible avant le moment de mémorisation (t_0).

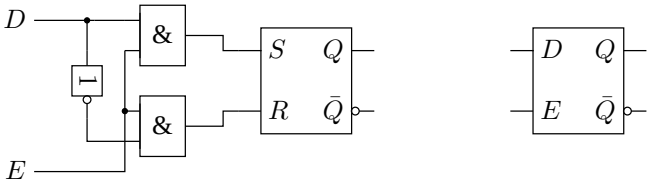


FIGURE 3.7 – Bascule asynchrone du type D (D-Latch).

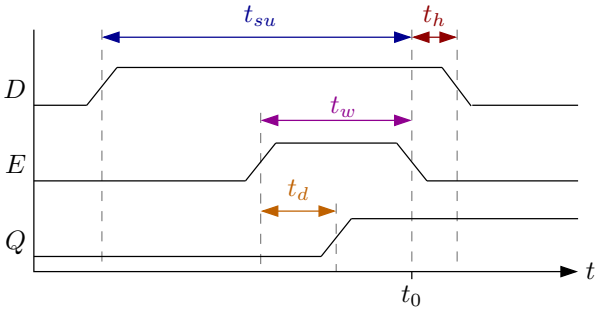


FIGURE 3.8 – Le comportement temporel des bascules asynchones.

Hold-time (t_h) : La durée minimale, pendant laquelle les données d'entrées doivent être disponible après le moment de mémorisation (t_0).

Largeur d'impulsion de contrôle (t_w) : Le temps minimal pendant lequel le signal de commande doit être actif.

Temps de retard (t_d) : Le temps de retard.

Le Setup-Time est nécessaire, pour être sûr, que chaque changement des données d'entrée a été pris en compte par le montage, avant que le signal de contrôle devienne inactif. Le setup-time et la largeur minimale d'impulsion de contrôle assurent, que les valeurs des signaux internes apparaissent à la sortie, avant que les signaux d'entrée ne disparaissent.

3.4 La commande par niveau logique (hachage) ou par transition

Il y existe deux types de commande pour les éléments de mémoire :

La commande par niveau logique : Dans le cas d'une commande par niveau logique, les entrées d'informations influencent le contenu de la mémoire tant que le signal de l'entrée de commande reste actif.

La commande par transition (le déclenchement sur flanc) : Dans le cas d'une commande par transition, les entrées d'information influencent le contenu de la mémoire seulement au flanc actif du signal de commande.

La figure 3.9 nous montre la différence entre ces deux types de commandes dans un diagramme temporel. On notera, que dans le cas d'une commande par niveau logique, le niveau du signal actif et dans le cas du réglage par transition, le flanc peuvent changer le contenu de la mémoire.

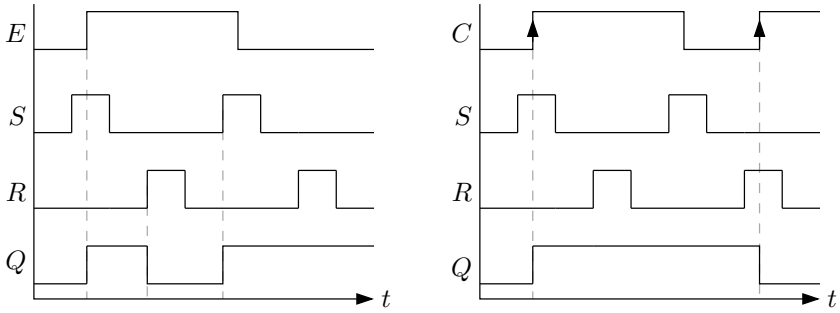


FIGURE 3.9 – Comparaison de commande par niveau logique (à gauche) et par transition (à droite).

3.5 Bascules synchrones (les bascules bistables)

Dans les systèmes synchrones numériques, les éléments de mémoire sont piloté par un seul signal de commande. Les éléments peuvent aussi être couplés en série de sorte que les sorties d'une première cellule soient les entrées d'une

cellule secondaire. Les bascules asynchrones ne peuvent pas être utilisés dans ces cas, puisque qu' étant transparent, transmettent les informations sur la chaîne entière. On utilise donc un un élément nouveau, le bascule synchrone (Flip-Flop). On entend par bascule synchrone un élément de mémoire bistable et commandé, qui ne connaît pas le mode transparent. Ce qui veut dire, qu'un changement de la sortie d'un bascule synchrone ne peut jamais être le résultat direct d'un changement de l'entrée synchrone. On peut donc monter des bascules synchrones en série sans perdre des données.

3.5.1 Bascule synchrone du type SR

La figure 3.10 nous montre, qu'un bascule synchrone maître-esclave se compose en principe de deux simples bascules asynchrones commandés. Si on utilise deux bascules asynchrones SR commandés, on obtient un bascule synchrone maître-esclave. Le comportement temporel d'un bascule synchrone maître-esclave

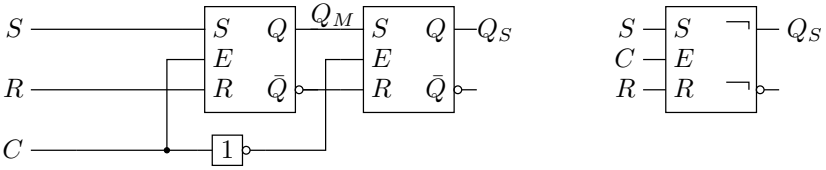


FIGURE 3.10 – Schema et symbole d'un bascule synchrone maître-esclave SR.

est montré dans le diagramme d'impulsions de la figure 3.11. Lorsque le signal C est au niveau haut, les données sont lues dans le premier bascule asynchrone, appellé maître. Les sorties du bascule synchrone gardent leur état precédent. Si le signal C passe à 0, les données seront transmisent au deuxième bascule, dénommé esclave, et donc transférées à la sortie du bascule synchrone. Le bascule asynchrone maître est dans cette phase en état de mémoire et ne laisse pas passer de nouvelles données. Le signal Q_M est la sortie Q du maître, le signal Q_S est la sortie du esclave et donc la sortie du bascule synchrone entier. Comme on peut le voir sur le diagramme ci-dessus, le deuxieme passage à 1 de l'entrée S se fait lorsque le signal C est déja à 1. On constate donc qu'on a un bascule synchrone commandé par niveau logique, puisque ce changement de l'entrée est pris en compte par le bascule maître et transmis à la sortie lorsque l'entrée C est

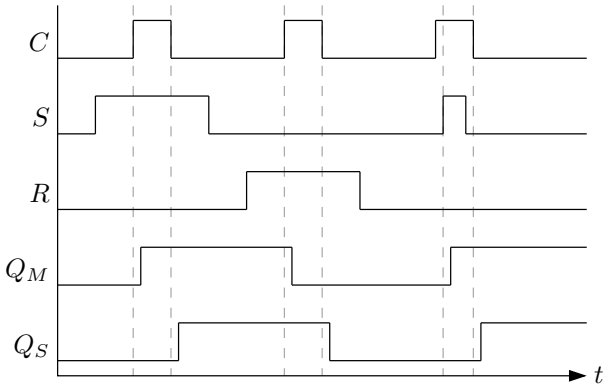


FIGURE 3.11 – Le diagramme d'impulsion du bascule synchrone maître-esclave SR.

au niveau bas. Le signal de sortie ne réagit donc pas à un flanc de l'entrée C mais a son niveau.

3.5.2 Bascule synchrone du type D

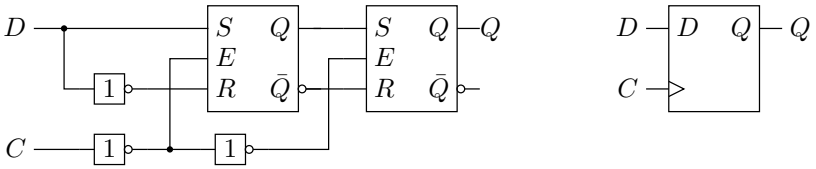


FIGURE 3.12 – Implementation maître-esclave d'un bascule synchrone commandé par transition.

Tout comme il existe divers types des bascules asynchrones, on peut donc réaliser plusieurs variantes des bascules synchrones. Si on utilise le principe du maître-esclave avec un bascule asynchrone D, on obtient une autre variante du bascule synchrone D (figure 3.12). Si l'entrée D change pendant que le signal C est à l'état bas, le bascule maître prend en compte cette modification seulement temporairement. C'est seulement pendant le flanc montant que le signal d'entrée,

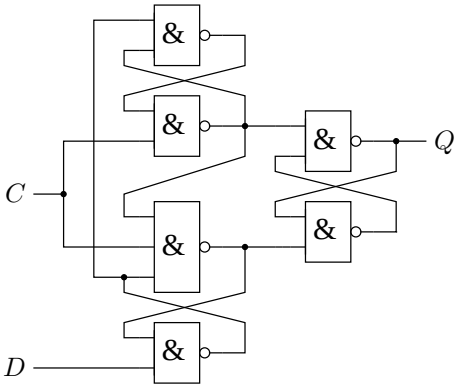


FIGURE 3.13 – Implémentation d'un bascule asynchrone D.

influence le signale de sortie du bascule synchrone. C'est pourquoi on parle d'un bascule synchrone D commandé par transition. Le schéma de la figure 3.13 nous montre une autre variante plus moderne d'un bascule synchrone D commandé par transition. On n'a pas une structure de maître-esclave, mais une simple mémoire avec des portes logiques couplées en amont. Si on analyse le mode de fonctionnement de ce couplage de près, on constate que la première bascule ne fonctionne pas comme mémoire lorsque le signal de cycle C est au niveau bas, puisque les sorties de cette bascule sont indépendantes de l'entrée D logique 1. Cependant, si le signal de commande passe de 0 à 1, la première cellule prend d'abord la valeur correspondante à l'entrée, puis après la deuxième cellule.

3.5.3 Bascule synchrone du type JK

Bien qu'il existe une grande variété des bascules synchrones on utilise souvent le bascule synchrone JK (JK-Flip-Flop). Tout comme le bascule synchrone SR il a deux entrées de données, l'un qui met à 1 le bascule et l'autre qui le remet à 0. La seule véritable différence, avec le bascule synchrone SR c'est le fait, que pour le JK les deux entrées de données peuvent être à 1 en même temps. La fonction des deux entrées J et K est montré dans la table de vérité du tableau 3.1. en utilisant la sortie Q comme variable d'entrée, on trouve une relation pour l'entrée S d'un bascule synchrone SR (figure 3.14). La procédure est similaire

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

TABLE 3.1 – La table de vérité d'un bascule synchrone JK.

pour l'entrée R. Le schéma du bascule synchrone JK est donné à la figure 3.15.

J	K	Q	S
0	0	0	0
0	0	1	-
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	-
1	1	0	1
1	1	1	0

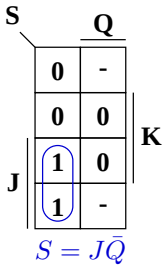


FIGURE 3.14 – Dérivation de la condition pour l'entrée S.

On voit dans la figure, qu'on peut dériver un bascule synchrone JK d'un bascule synchrone SR.

3.6 Les entrées asynchrones des bascules synchrones

Les bascules synchrones déjá traités peuvent avoir en plus des entrées de données synchrones, des entrées asynchrones. On a habituellement besoin de ces entrées asynchrones supplémentaires pour mettre à 1 (Preset) ou mettre à 0 (Clear) un bascule :

- L'entrée Preset asynchrone (P)
- L'entrée Clear asynchrone (R)

L'entrée Preset met à 1 la sortie Q du Flip-Flop, tandis que l'entrée Clear la met à 0. La figure 3.16 montre le symbole d'un bascule synchrone D commandé par transition avec des entrées $\overline{P}$ et $\overline{R}$. L'implémentation, en utilisant le circuit

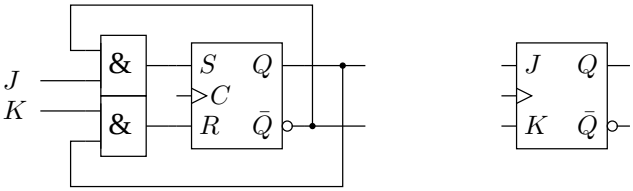


FIGURE 3.15 – Le bascule synchrone JK construit à partir d'un bascule synchrone SR.

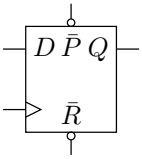


FIGURE 3.16 – Symbole d'un bascule synchrone D avec des entrées asynchrones Preset (mise à 1) et Clear (mise à 0)

SN7474 est montrée en figure 3.17. Les deux signaux sont actifs bas. Cela veut dire qu'ils ne doivent pas prendre la valeur logique 0 en même temps, puisque la valeur de la sortie Q est indéfinie. Lorsqu'on conçoit un circuit utilisant des bascules synchrones, on doit faire attention à ce que les bascules utilisés soient mit dans un état défini pendant le start-up du système . C'est pour cela qu'on n'utilise que des bascules synchrones qui ont des entrées asynchrones Set ou Reset. Pour des raisons de test, il est très utile d'avoir la possiblité de forcer le bascule dans un êtat défini par un signal d'initialisation, en général le RESET.

3.7 Bascules monostables

Un bascule monostable (monoflop) est un montage qui peut revêtir comme un bascule synchrone deux etats. A la différence du bascule synchrone, le bascule monostable a seulement un état stable. L'autre état garde sa valeur seulement pendant un temps défini. Après cet temps il bascule de nouveau a son état stable. Le bascule monostable peut être réaliser avec des bascules asynchrones SR ou des bascules synchrones ainsi que des éléments pour un circuit de retard ou un différentiateur pour ajuster la constante de temps. On utilise des éléments RC

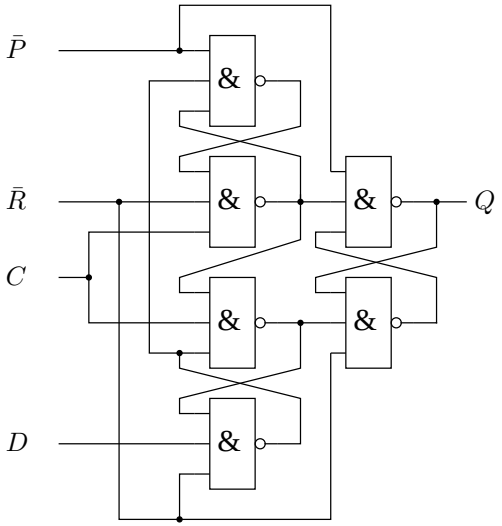


FIGURE 3.17 – L'implémentation d'un SN7474 bascule synchrone D.

externes. La figure 3.18 montre le symbole d'un bascule monostable. S'il est

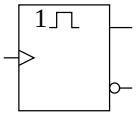


FIGURE 3.18 – Symbole d'une bascule monostable commandé par transition.

déclenché, il produit une impulsion de durée définie. On en distingue deux types :

- Les bascule monostable redéclenchable.
- Les bascule monostable non redéclenchable.

La différence entre les deux types est illustrée sur le diagramme de temps de la figure 3.19. Les impulsions n'ont pas d'influence sur le signal de sortie du bascule monostable non redéchlenchable s'il est déjá dans l'etat non stable. Le circuit SN74123 est un exemple de bascule monostable. Il peut réagir aux flancs montants ou descendants selon les éléments périphériques et peut travailler comme redéclencheable ou non redéclencheable.

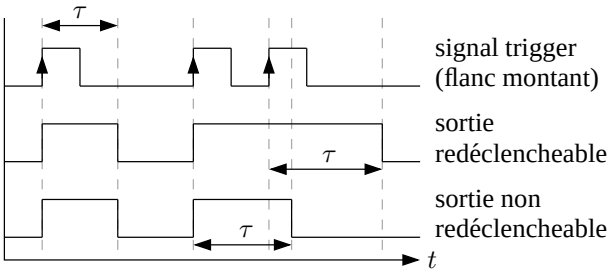


FIGURE 3.19 – Le diagramme temporel d'un bascule monostable avec et sans redéclenchement.

3.8 Le Multivibrateur (la bascule astable)

Le multivibrateur est un montage qui a deux états entre lesquelles il bascule perpetuellement. Pour cette raison on l'appelle aussi bascule astable ou générateur de rectangle. On arrive à ce comportement astable par une rétroaction dépendant d'un temps defini grâce aux éléments d'un circuit de retard ou d'un différentiateur. Un montage très simple de multivibrateur se compose d'un trigger

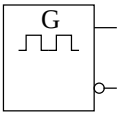


FIGURE 3.20 – Symbole d'un Multivibrateur.

de Schmitt et d'un circuit de retard, comme montré en figure 3.22. Un trigger de Schmitt ressemble à un inverseur, mais contrairement à ce dernier, il possède une tension de seuil basse et haute à l'entrée. On l'utilise souvent pour de supprimer les perturbations. La tension U_c aux bornes du condensateur croît dans la phase de charge de U_{su} vers U_{Qmax}. Dès qu'elle atteints la valeur de limite supérieure, la phase de charge sera remplacée par la phase de décharge. La tension U_c decroît de U_{so} vers U_{Qmin}, jusqu'à ce que U_{su} soit atteintte. Dans la phase 0- t_1 on peut écrire l'équation suivante :

$$U_c = (U_{Qmax} - U_{su})(1 - e^{-\frac{t}{RC}}) + U_{su}$$

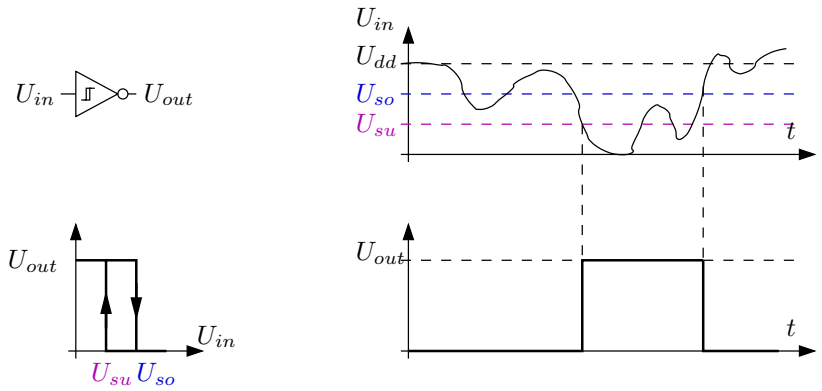


FIGURE 3.21 – Courbes de tension d'un Trigger de Schmitt.

A l'instant $t = t_1$ la tension du condensateur est égale à la tension limite U_{so}. Avec cela on obtient pour la demi-periode T l'équation siuvante :

$$T = t_1 = RC \ln \frac{U_{Qmax} - U_{su}}{U_{Qmax} - U_{so}}$$

Cette équation nous montre de quelle manière on peut ajuster la période T grâce à des éléments RC.

3.9 Les Aléas (hasards)

Jusqu'á présent nous avons toujours supposé, qu'une porte logique est idéalle et le temps de propagation des signaux nul. C'est pourquoi le problème des effets transitoires a été négligé. Si en outre de leur fonction logique on tient compte de leur comportement temporel, on remarque alors des phénomènes bizares qui peuvent anéantir le fonctionnement correct du circuit. La figure 3.23 montre une porte logique réelle. Elle se compose d'une équation de Boole et d'un élément de retard pour modeliser le temps de propagation des signaux. Si on tient compte de ces temps on peut voir des phénomènes transitoires non désiré.

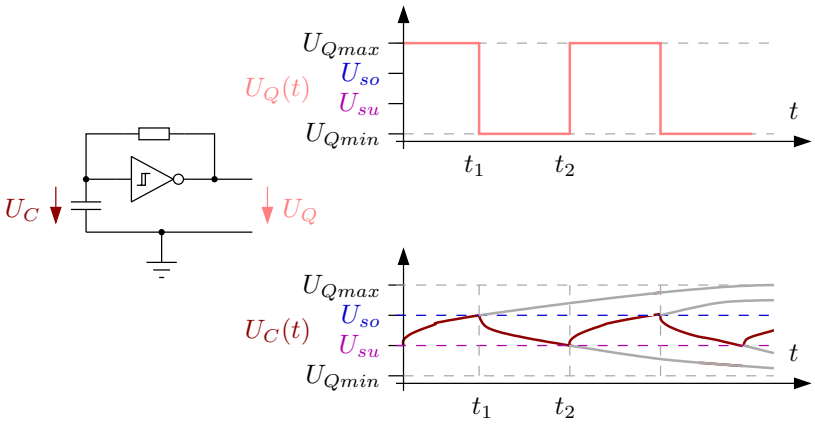


FIGURE 3.22 – Comportement temporel d'un multivibrateur simple.

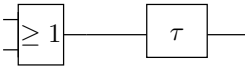


FIGURE 3.23 – Une porte logique réelle avec délai de propagation.

3.9.1 Naissance des Aléas

Si dans un circuit logique combinatoire le signal de sortie dépend non seulement des signaux d'entrée mais aussi des temps de propagation, le circuit aura des aléas. Dans un circuit logique combinatoire les phénomènes transitoires peuvent occasionner des aléas. On distingue deux groupes d'aléas

- Aléas statiques
- Aléas dynamiques

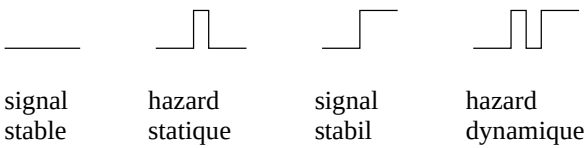


FIGURE 3.24 – Une comparaison des Hasards statiques et dynamiques.

On parle d'aléas statique, si le signal change sa valeur seulement pour un instant. La valeur statique n'est pas concernée. Si on a un aléa dynamique, le signal change plusieurs fois avant de devenir stable. La figure 3.25 montre un exemple

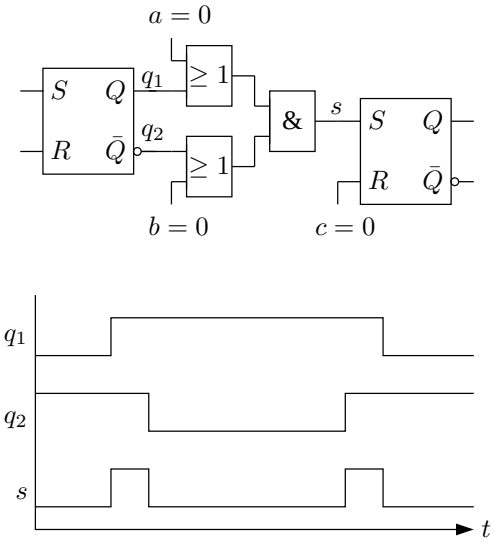


FIGURE 3.25 – Circuit avec un aléa statique au signal s.

de circuit, qui ne fonctionne pas à cause d'un aléa. Le premier latch SR présente à la sortie Q une valeur logique 0. Si elle passe à 1, d'abord le signal q_1 passe à 1 et après le q_2 passe à 0. Puisque pour un instant les deux entrées de la porte AND sont à 1, le signal s présente un aléa statique. Cet aléa met 1 le deuxième latch, ce qui est un fonctionement erroné. Les aléas peuvent apparaître seulement si plus d'un des signaux d'entrés ou signaux internes changent en même temps. On peut voir sur le diagramme de Karnaugh si un aléa peut apparaître dans une situation (figure 3.26). Les signaux $ABCD$ changent de 0000 à 0101. Puisque les signaux B et D ne peuvent pas changer en même temps, l'un doit changer d'abord. Un aléa statique apparaît si le signal B change d'abord de 0 à 1 et après le signal D, comme marqué par la flèche a. Si l'ordre est inverse, le chemin de la flèche b mène seulement sur des 1 logiques, cela veut dire que le signal de sortie reste stable et n'a pas d'aléa.

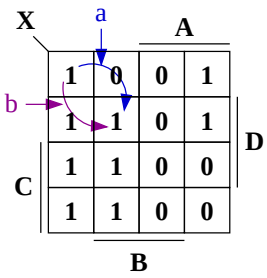


FIGURE 3.26 – Naissance d'un Hasard.

3.9.2 Eviter les hazard

Les aléas ne sont pas toujours facile à supprimer dans les circuits logiques. Mais on peut éviter les situations qui occasionnent des aléas. Pour cela il est important de déterminer les signaux critiques sensibles aux aléas et de prendre des mesures correspondantes. En général on peut constater que les signaux clock, set, et reset des Flip-Flop sont des signaux critiques. Un aléa statique sur une telle ligne provoque toujours un comportement faux du circuit. Si on respecte les règles de construction suivantes, les aléa peuvent être éliminés :

- Pas des liaisons combinatoires pour la génération des signaux clock, set ou reset.
- Synchronisation correcte des signaux qui sont sensibles aux aléas.

On peut constater qu'en général, les circuits asynchrones sont sensibles aux aléas. C'est pourquoi on doit concevoir autant que possible des circuits synchrones. On gagne en sécurité avec un circuit synchrone, mais on doit intégrer en plus le mécanisme de synchronisation et cela cause des temps de retard plus longs.

4 Nombres

4.1 Systèmes de numération

- Les listes de comptage (base 1) ne sont pas utilisables pour un traitement ultérieur (grande dépense technique)
- Les chiffres romains : CCCIC = 399, CD = 400. Contrairement au système décimal, la position des chiffres ne détermine pas leur poids.
- Le système de nombres décimaux (systèmes de position des valeurs ou systèmes polyadiques) :

$$\begin{aligned} 3970.25 &= 3 \cdot 1000 + 9 \cdot 100 + 7 \cdot 10 + 2 \cdot 0.1 + 5 \cdot 0.01 \\ &= 3 \cdot 10^3 + 9 \cdot 10^2 + 7 \cdot 10^1 + 0 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2} \end{aligned}$$

Somme des produits des coeficients compris entre 0 et 9 par des puissance de 10.

Forme generale des systèmes polyadiques à la base B :

$$\begin{aligned} Z &= c_{n-1} \cdot B^{n-1} + c_{n-2} \cdot B^{n-2} + \cdots + c_1 \cdot B^1 + c_0 \cdot B^0 + \cdots \\ &\quad + c_{-1} \cdot B^{-1} + c_{-(m-1)} \cdot B^{-(m-1)} + c_{-m} \cdot B^{-m} \\ Z &= \sum_{i=-m}^{n-1} c_i B^i \end{aligned}$$

Les règles : $0 \leq c_i \leq B - 1$ avec n pour la quantité de chiffres à gauche de la virgule, m pour la quantité de chiffres à droite de la virgule, B pour la base du système de nombres, c_i pour le coefficient et i pour le nombre ordinal. La technique numérique utilise le système binaire (base 2) et les nombres 0 et 1. Les circuits de traitement des nombres binaires sont faciles à réaliser comme on va le voir plus tard.

4.2 Transformation en/de nombres décimaux

Représentation abrégée d'un nombre en base B en un nombre décimal : somme de multiplications ou de divisions avec/par la base B.

$$
\begin{aligned}
&[c_{n-1}c_{n-2}\cdots c_1c_0c_{-1}\cdots c_{-(m-1)}c_{-m}] = \\
&\quad [(((c_{n-1}B + c_{n-2})B + \cdots)B + c_0)] + \\
&\quad [(((((c_{-m}/B + c_{-(m-1)})/B + \cdots)/B + c_{-1})/B)]
\end{aligned}
$$

Transformation de la partie entière de nombres en base quelconque en nombres décimaux :

$$
\begin{aligned}
c_{n-1} &= S_1 \\
S_1B + c_{n-2} &= S_2 \\
S_2B + c_{n-3} &= S_3 \\
&\vdots \\
S_{n-2}B + c_1 &= S_{n-1} \\
S_{n-1}B + c_0 &= S_n
\end{aligned}
$$

Transformation de la partie fractionnaire de nombres en base quelconque en nombres décimaux :

$$
\begin{aligned}
c_{-m} &= s_1 \\
s_1/B + c_{-(m-1)} &= s_2 \\
s_2/B + c_{-(m-2)} &= s_3 \\
&\vdots \\
s_{m-1}/B + c_{-1} &= s_m \\
s_m/B &= s_{m+1}
\end{aligned}
$$

Le nombre décimal se compose de la somme des S_i et des s_i. Exemples de transformation en nombres décimaux :

- Le nombre binaire 111000.011_{bin} doit être transformé en nombre décimal.
  Solution : 56.375_{dec}
- Le nombre octale (base 8) 111000.011_{oct} doit être transformé en nombre décimal.
  Solution : $37376.0175781_{\text{dec}}$

Pour la transformation d'un nombre décimal en un nombre de la base B, on peut

dériver des règles semblables.
A gauche de la virgule : En divisant par B le nombre suivant

$$Z = c_{n-1} \cdot B^{n-1} + c_{n-2} \cdot B^{n-2} + \cdots + c_1 \cdot B^1 + c_0 \cdot B^0$$

on obtient :

$$Z/B = c_{n-1} \cdot B^{n-2} + c_{n-2} \cdot B^{n-3} + \cdots + c_1 \cdot B^0 + \text{Rest } c_0$$

Après n divisions on trouve le coefficient c_{n-1}.
A droite de la virgule : En multipliant par B le nombre suivant,

$$z = c_{-1} \cdot B^{-1} + c_{-2} \cdot B^{-2} + \cdots + c_{-(m-1)} \cdot B^{-(m-1)} + c_{-m} \cdot B^{-m}$$

on obtient :

$$zB = c_{-1} + c_{-2} \cdot B^{-1} + \cdots + c_{-(m-1)} \cdot B^{-(m-2)} + c_{-m} \cdot B^{-(m-1)}$$

Après n multiplications on trouve le coefficient c_{-m}

Transformation de la partie entière de nombres décimaux en nombres en base B quelconque.

$$\begin{array}{rcll} Z/B & = & S_1 & \text{Rest } c_0 \\ S_1/B & = & S_2 & \text{Rest } c_1 \\ S_2/B & = & S_3 & \text{Rest } c_2 \\ & \vdots & & \\ S_{n-2}/B & = & S_{n-1} & \text{Rest } c_{n-2} \\ S_{n-1}/B & = & 0 & \text{Rest } c_{n-1} \end{array}$$

Transformation de la partie fractionnaire de nombres décimaux en nombres en base B quelconque.

$$\begin{array}{rcl} z \cdot B & = & s_1 + c_{-1} \\ s_1 \cdot B & = & s_2 + c_{-2} \\ s_2 \cdot B & = & s_3 + c_{-3} \\ & \vdots & \\ s_{m-1} \cdot B & = & s_m + c_{-(m-1)} \\ s_m \cdot B & = & s_{m+1} + c_{-m} \end{array}$$

Exemple de transformation d'un nombre décimal. Le nombre décimal 109.78125_{dec} est à transformer

a) en un nombre binaire :
Solution : 1101101.11001_{bin}
b) en un nombre octal :
Solution : 155.62_{oct}

4.3 Représentation des nombres à virgule fixe et flottante

En cas de représentation des nombres à virgule fixe, la virgule se trouve à une place déterminée ; par exemple en comptabilité, les francs et les centimes 246.23 sFr. En cas de représentation des nombres à virgule flottante, les nombres sont décrits par une mantisse et un exposant (fourchette de nombres très grandes pour les calculs techniques - scientifiques). Les chiffres excédentaires sont supprimés pour les valeurs basses. L'exemple ci dessous donne une représentation Fortran/Basic avec 8 chiffres de mantisse :

$$\begin{aligned} 43.5 &= 0.435 \cdot 10^2 \\ &= 0.43500000E+2 \\ 156.378 &= \\ 0.00032703 &= \\ 237438965.17 &= \end{aligned}$$

4.4 Représentation des nombres positifs et négatifs

Représentation du signe et de la valeur absolue : En technique numérique le signe $+$ est décrit par la valeur binaire 0, le signe $-$ par la valeur binaire 1. En général le signe se trouve a gauche des bits des valeurs absolues.

Représentation complémentaire (nombres co-négatifs) : Complément à un : s'obtient par l'inversion de chaque bit. Complément à deux : est égal au complément à un plus 1.

Représentation offset-binaire (Excess-N) : Le nombre le plus négatif (-N) est décrit par le nombre binaire le plus bas (00...00).

nombre décimal	signe valeur-absolue	complément à un	complément à deux	offset binaire
+7	0111	0111	0111	1111
+6	0110	0110	0110	1110
+5	0101	0101	0101	1101
+4	0100	0100	0100	1100
+3	0011	0011	0011	1011
+2	0010	0010	0010	1010
+1	0001	0001	0001	1001
+0	0000	0000	0000	1000
-0	1000	1111	0000	1000
-1	1001	1110	1111	0111
-2	1010	1101	1110	0110
-3	1011	1100	1101	0101
-4	1100	1011	1100	0100
-5	1101	1010	1011	0011
-6	1110	1001	1010	0010
-7	1111	1000	1001	0001
-8			1000	0000

TABLE 4.1 – Représentation des nombres positifs et négatifs.

4.5 L'arithmétique binaire

Addition : Comme dans le système decimal ! ? : $1 + 1 = 10$

Soustraction : Indirecte : $D = M - S = M + (B^n - S - B^n) = M + K - B^n$. La différence D s'obtient en additionnant à M le complément à B^n du terme S, ($K = B^n - S$) et en soustrayant de nouveaux B^n au resultat (où n est le nombre maximum des chiffres à gauche de la virgule). On trouve le complément K en réduisant B^n par une valence, puis on fait la soustraction facile et ensuite on additionne cette valeur de nouveau (complémentation). En ce qui concerne la soustraction de B^n on doit distinguer deux cas :

– $M > S$, la différence est positve : On supprime seulement le 1 de la position $n + 1$.

– $M < S$, La difference est négative : (en position $n+1$ il y a un 0). Le terme $M + K - B^n$ doit être remplacé par $-(B^n - M - K) = -(B^n - (M + K))$. Ainsi on calcule le complément une deuxième fois (complément inverse).

Exemples :

– On doit calculer la difference $123 - 11$ en système decimal et en système binaire par l'addition du complément.
Solution : 112_{dec} ; 1110000_{bin}

– On doit calculer la difference $52 - 417$ en système decimal et en système binaire par l'addition du complément.
Solution : -365_{dec} ; -101101101_{bin}

Multiplication : C'est la même procédure qu'avec des nombres décimaux, mais plus simple ($1 \cdot 1 = 1$).

Exemple : Calculer le produit $217 \cdot 35$ en binaire.
Solution : 1110110101011_{bin}

Division : La soustraction continue est remplacée par l'addition du complément. Chaque dénominateur sera arrondi à la puissance en base B immédiatement supérieure. Si l'addition du complément ne produit pas de retenue, le résultat partiel est 0 et on doit calculer avec la même deminuende complété par le chiffre prochain du dividende.

Exemple : On doit calculer le quotient $667/29$ en binaire.
On a $667_{dec} = 1010011011_{bin}$ et $29_{dec} = 11101_{bin}$.
Solution : Tout d'abord on doit exprimer le complément à 2 du dénominateur à la puissance de base B immédiatement supérieure. On obtient : $1010011011/11101 = 010111 = 10111_{bin} = 23_{dec}$

5 Codes

But : Après l'étude de ce chapitre l'étudiant devra maîtriser les notions et/ou être capable de resoudre les problèmes suivants :
- Définition des codes.
- Application du code cyclique.
- Codes à détection d'erreurs (par exemple parity bit).
- Codes correcteurs d'erreurs (par exemple code Hamming).

5.1 Introduction

Un code est une correspondance entre les symboles de deux jeux de caractères. On a développé une variété de codes qui satisfont différentes exigences :
- capacité de stockage réduite
- vitesse de traitement
- éviter ou même corriger des défauts (par exemple erreurs de balayage)
- suppression de la composante DC
- extraction du signal d'horloge
- etc...

Il existe une grande variétée de codes permettant d'atteindre différents buts.

> **Définition :** Un code est une règle précise pour joindre les élements de deux quantités.

5.2 Codes BCD

Les codes BCD (décimal codé binaire) représentent les 10 $(0-9)$ nombres décimaux à l'aide de 4 chiffres binaires. $2^4 = 16$ combinaisons sont possibles. Il

binaire 2^4 2^3 2^2 2^1 2^0	octal 8^1 8^0	décimal 10^1 10^0	hexadécimal 16^1 16^0
00000	00	00	00
00001	01	01	01
00010	02	02	02
00011	03	03	03
00100	04	04	04
00101	05	05	05
00110	06	06	06
00111	07	07	07
01000	10	08	08
01001	11	09	09
01010	12	10	0A
01011	13	11	0B
01100	14	12	0C
01101	15	13	0D
01110	16	14	0E
01111	17	15	0F
10000	20	16	10
10001	21	17	11
⋮	⋮	⋮	⋮
11110	36	30	1E
11111	37	31	1F

TABLE 5.1 – Exemple de codes binaire, octal, décimal et hexadécimal.

y a donc 6 combinaisons de trop. Pour représenter les 10 chiffres décimaux avec 4 caractères, il y a un nombre très élevé de possibilités de codage.

$$2^4!/6! \quad = \quad 2.9\,10^{10}$$

Les critères qui guident le développement d'un code sont :

- La valence : chaque position binaire a une valeur définie (convertisseur numérique-analogique).
- La sécurité d'échantillonnage : codes cycliques.
- La sécurité de transmission : une erreur ne doit pas provoquer un nouveau symbol.

– La capacité de calcul : certains codes permettent d'exécuter des calculs plus vite (par exemple de calculer le complément).

Dans le tableau ci-dessous on trouve des exemples de codes typiques :

– Codes Aiken, Stibitz : négation symétrique (complément à 9).
– Code 4-2-2-1 : utilisé pour des compteurs rapides.
– Code White : les nombres obtenus avec ce code ne sont pas des puissances de 2.
– Codes Gray, Glixon, O'Brien, Excess3 reflecté : code cycliques.

nom	code 8-4-2-1	code Aiken	asym-metric-code 2-4-2-1	code Stibitz Exzess-3-code	code 4-2-2-1	code White	code Glixon	code O'Brien	Exzess 3-Code reflecté
valance	8-4-2-1	2-4-2-1	2-4-2-1		4-2-2-1	5-2-1-1			
0000	0	0	0	-	0	0	0	-	-
0001	1	1	1	-	1	1	1	0	-
0010	2	2	2	-	2	-	3	2	0
0011	3	3	3	0	3	2	2	1	-
0100	4	4	4	1	-	-	7	4	4
0101	5	-	5	2	-	3	6	-	3
0110	6	-	6	3	4	-	4	3	1
0111	7	-	7	4	5	4	5	-	2
1000	8	-	-	5	-	5	9	-	-
1001	9	-	-	6	-	6	-	9	-
1010	-	-	-	7	6	-	-	7	9
1011	-	5	-	8	7	7	-	8	-
1100	-	6	-	9	-	-	8	5	5
1101	-	7	-	-	-	8	-	-	6
1110	-	8	8	-	8	-	-	6	8
1111	-	9	9	-	9	9	-	-	7
0	0000	0000	0000	0011	0000	0000	0000	0001	0010
1	0001	0001	0001	0100	0001	0001	0001	0011	0110
2	0010	0010	0010	0101	0010	0011	0011	0010	0111
3	0011	0011	0011	0110	0011	0101	0010	0110	0101
4	0100	0100	0100	0111	0110	0111	0110	0100	0100
5	0101	1011	0101	1000	0111	1000	0111	1100	1100
6	0110	1100	0110	1001	1010	1001	0101	1110	1101
7	0111	1101	0111	1010	1011	1011	0100	1010	1111
8	1000	1110	1110	1011	1110	1101	1100	1011	1110
9	1001	1111	1111	1100	1111	1111	1000	1001	1010

TABLE 5.2 – Code BCD 4 bits typiques.

5.3 Codes cycliques

L'utilisation des codes cycliques permet d'éviter les erreurs d'échantillonnage. Par exemple pour coder des distances ou des angles, on utilise des règles ou des roues codeuses. Or lors de leur échantillonnage les transitions peuvent provoquer des erreurs (voir figure 5.1). La figure 3.2 montre une roue codeuse (code gray) utilisée pour éviter les erreurs d'échantillonage. Cependant le code Gray n'est

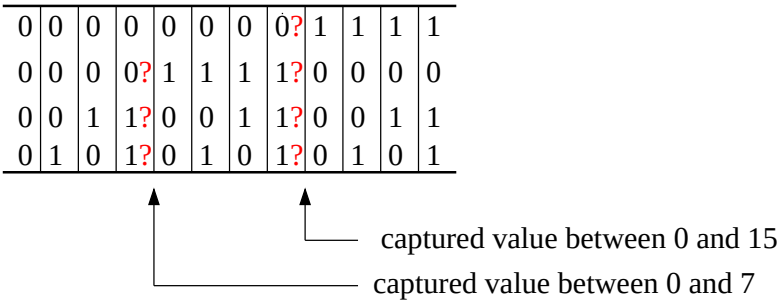


FIGURE 5.1 – On peut recevoir des erreurs de codages si on utilise des codages non-cycliques, parce que plusieurs bits peuvent changer de valeur lors de la même transition.

pas approprié pour du décimal codé en binaire (BCD), car le passage de 9 à 0 nécessite le changement de trois bits.

5.4 Codes à détection d'erreurs

Un autre critère à prendre en compte lors de la création d'un code est la sécurité de transmission. Celle-ci peut être améliorée par l'adjonction de redondances. La redondance dans un code permet de détecter, voire même de corriger, des erreurs. Les codes à détection d'erreurs sont ceux à l'aide desquels il est possible de détecter une erreur simple, c'est à dire le changement d'un bit de 0 à 1 ou vice-versa.

Le contrôle de parité : Le contrôle de parité est la méthode la plus simple pour détecter des erreurs. Il consiste à ajouter un bit 1 (le bit de parité) de façon a ce que la somme de tous les bits 1 soit paire (even) ou impaire (odd). Grâce au contrôle de parité :

- Les erreurs simples sont détectées.
- Les erreurs doubles (2 bits qui changent simultanément) ne seront pas détectées.
- Les triples sont détectées.

Application pour un codage à 4 bit :

Code m parmi n : Le contrôle de parité n'est pas optimum pour les codes BCD. On définit pour cela un nouveau code, le code *m parmi n*. Le code *m parmis*

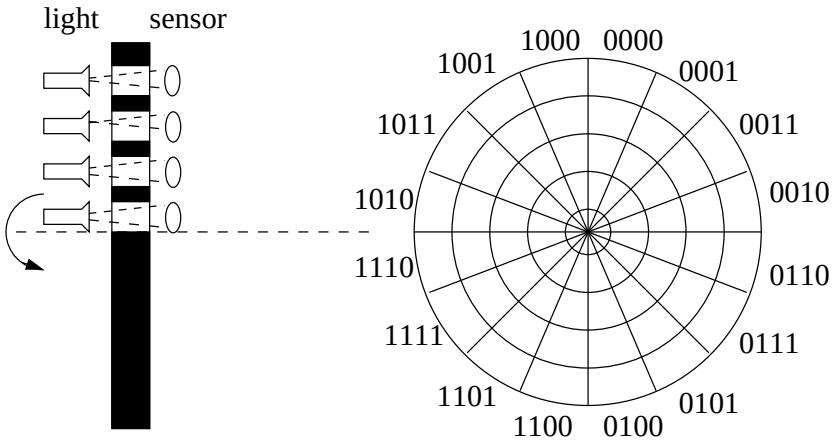


FIGURE 5.2 – Codes cycliques.

n nous donne N nombres différents :

$$N = \left(\begin{array}{c} n \\ m \end{array} \right) = \frac{n!}{m!(n-m)!}$$

Avec ce code, m parmi n bits sont toujours égaux à 1. Le contrôle est très simple, compter le total des 1, il doit toujours être egal à 2 (exception pour le 0). Grâce au code m parmis n :

– Les erreurs simples sont détectées.
– Les erreurs doubles sont détectées à condition qu'elles soient identiques.
– Les erreurs triples sont détectées.

La comparaison des deux codes montre que l'on trouve plus d'erreurs avec le code *m parmi n* qu'avec le code de *parité*.

5.5 Codes correcteurs d'erreurs

Introduction : Pour corriger les erreurs de transmission, il existe plusieurs possibilités :

– La répétition automatique si une erreur est détectée par le parity-check.

nombre décimal	code Gray
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

TABLE 5.3 – Code Gray

– Le traitement en bloc : en plus du bit de parité on ajoute à un bloc de bytes un mot de contrôle. Exemple pour Code-Aiken a) bloc sans erreurs b) avec une erreur c) avec deux erreurs sur la même ligne d) avec deux erreurs sur des lignes et colonnes différentes (il n'est pas possible de corriger deux erreurs à la fois, mais seulement de les détecter). Code Hamming (ou autres codes correcteurs d'erreurs, comme Reed Solomon).

Hamming Code :

– On peut détecter et corriger une erreur dans chaque mot.
– Pour pouvoir corriger un bit unique, il faut un deuxième bits de parité (2 parmis 3). Par Exemple : dans le mot 111 (Bit d'information, 1er bit de parité, 2ème bit de parité) ; Où est l'erreur : 110, 101, ou 011 ?

Ainsi donc pour pouvoir corriger des erreurs il faut augmenter la redondance du code de détection d'erreurs.

Prenons l'exemple de mots de 2 bits qu'il faut compléter par des bits de parité pour permettre la correction d'erreurs. Comme un seul bit de parité ne suffit pas, essayons avec 2 : Comme on peut le voir dans l'exemple précédent, il faut plus

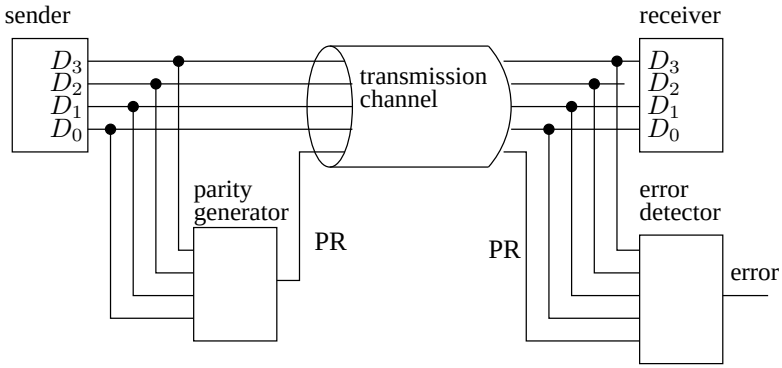


FIGURE 5.3 – Le contrôle de parité pour detecter des erreurs

nombre décimal	2^2	2^1	2^0	PB
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

TABLE 5.4 – Bit de parité pour une parité paire.

que 2 bits pour corriger 2 bits de données. Le code de Hamming montre combien de bits de parité sont nécessaires pour une certaine quantité de données. Dans la figure suivante 3 bits de parité sont nécessaires pour détecter, localiser et corriger les erreurs qui surviennent dans des mots de 4 bits. Le tableau 5.7 donne la signification des bits de correction, c'est à dire la position de l'erreur. Les bits de correction doivent se trouver aux positions 1, 2 et 4 (8, 16 ...) afin que l'évaluation des bits k indique la position des erreurs. Du tableau on peut déduire que :

- k1 complète m1, m2, m4
- k2 complète m1, m3, m4
- k3 complète m2, m3, m4

valeur	2 parmis 5 Code 74210
0	11000
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

TABLE 5.5 – Code 2 parmis 5

m_1	m_0	k_1	k_0
	x		x
x	x	x	
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1
1	1	1	1 ←

TABLE 5.6 – Code avec erreurs. Où est l'erreur ?

A l'aide de 3 bits de parité (8 valeurs) on peut corriger au maximum 4 bits de données. Pour corriger des erreurs simples, il faut remplir les deux conditions suivantes pour le code Hamming :

- Chaque colonne doit avoir une contribution.
- On doit pouvoir distinguer les colonnes

Soit k le nombre de bits de parité et m le nombre de bits de données, la relation suivante les lie :

$$2^k - 1 \geq m + k \quad = \quad n$$

n_7	n_6	n_5	n_4	n_3	n_2	n_1
m_4	m_3	m_2	$\mathbf{k_3}$	m_1	$\mathbf{k_2}$	$\mathbf{k_1}$
x	x	x	x			
x	x			x	x	
x		x		x		x

F_3	F_2	F_1	
v	v	v	pas d'erreur
v	v	f	k_1, n_1 sont faux
v	f	v	k_2, n_2 sont faux
v	f	f	m_1, n_3 sont faux
f	v	v	k_3, n_4 sont faux
f	v	f	m_2, n_5 sont faux
f	f	v	m_3, n_6 sont faux
f	f	f	m_4, n_7 sont faux

TABLE 5.7 – Code Hamming

Cette formule permet de calculer pour toute quantité de données le nombre de bits de parité à ajouter pour effectuer la correction d'erreurs simples. Le code Hamming (4 bits de données et 3 de parité) a été utilisé lors de la transmission des données ci dessous. L'information reçue contient des erreurs. Localisez ces erreurs et trouvez l'information correcte (originale).

a) 0010100
b) 1000100
c) 0001100

5.6 Les codes alphanumériques

A côté du code BCD standard (6-Bit) et EBCDI (Extended-BCD-Interchange-Code, 8-Bit) le code d'ASCII (american standard code of information interchange) s'est imposé pour l'échange de données entre ordinateurs. C'est un code à 7-bits avec un bit de parité. Il contient à côté du jeux de caractères de base des caractères de commandes pour la transmission (EOT-End of Transmission), pour le mise en forme (CR-Carriage Return) et pour la séparation (RS Record - Separator).

			n_7	0	0	0	0	1	1	1	1
			n_6	0	0	1	1	0	0	1	1
			n_5	0	1	0	1	0	1	0	1
n_4	n_3	n_2	n_1								
0	0	0	0	NUL	DLE	0	SP	@	P		p
0	0	0	1	SOH	DC1	1	!	A	Q	a	q
0	0	1	0	STX	DC2	2	”	B	R	b	r
0	0	1	1	ETX	DC3	3	#	C	S	c	s
0	1	0	0	EOT	DC4	4	$	D	T	d	t
0	1	0	1	ENQ	NAK	5	%	E	U	e	u
0	1	1	0	ACK	SYN	6	&	F	V	f	v
0	1	1	1	BEL	ETB	7	’	G	W	g	w
1	0	0	0	BS	CAN	8	(	H	X	h	x
1	0	0	1	HT	EM	9	)	I	Y	i	y
1	0	1	0	LF	SUB	:	‡	J	Z	j	z
1	0	1	1	VT	ESC	;	+	K	[	k	{
1	1	0	0	FF	FS	$<$	,	L	\	l	|
1	1	0	1	CR	GS	$=$	−	M	]	m	}
1	1	1	0	SD	RS	$>$	.	N	ˆ	n	’
1	1	1	1	Sl	US	?	/	O	-	o	DEL

TABLE 5.8 – Code ASCII à 7 bit

6 Machine d'Etat Finis

Les buts : Après l'étude de cet chapitre on doit être capable de résoudre les problèmes suivants ou de répondre aux questions suivantes de façon précise :

- Quelles sont les différences entre les automates de Mealy, de Moore et de Medwedjew et quels sont leurs avantages particuliers.
- Sur la base de la description complète d'un algorithme pouvoir, construire le diagramme d'état correspondant.
- Connaitre les problèmes des états parasitaires et l'initialisation d'automate.
- Sur la base du diagramme d'état pouvoir construire la table de transition et la table de sortie et à l'aide de cela synthétiser la forme minimisée de cet automate.
- Connaitre les problèmes d'aléas dans les séquenceurs et pour la commande des voies externes des données.
- Connaitre les problèmes de synchronisation des signaux asynchrones dans les sequenceurs.

6.1 Structures de base des séquenceurs

Les séquenceurs sont des circuits, qui génèrent une séquence de signaux de sortie à partir d'une séquence des signaux d'entrée. On trouve les machines d'etats finis (anglais : finite state machine, FSM) - aussi appelé séquenceurs, automates ou circuits séquentiels - souvent dans le monde de la technique numérique et surtout dans les circuits VLSI. Les raisons sont évidentes : Les commandes séquentiels peuvent être décrites simplement et de façon compréhensible par des diagrammes d'état. La synthèse des circuits sur la base des diagrammes d'état est complètement automatique et grâce à cela, efficace dans le temps et sans défauts. Ce qui est très agréable pour la conception des circuits numériques ou de VLSI. On distingue deux types d'automates : l'automate de Mealy et l'automate

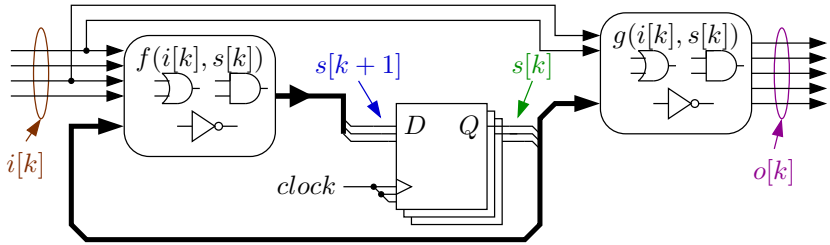


FIGURE 6.1 – Structure de l'automate de Mealy.

de Moore. Tous les deux consistent chaqu'un en deux circuits combinatoires (les foncions f et g) et un registre pour la mémorisation d'état. La figure 6.1 montre la structure d'un automate de Mealy. On voit que les valeurs de sortie o dépendent en même temps des états internes s et des valeurs d'entrée i. L'automate de Mealy satisfait les fonctions suivantes :

$$\begin{aligned} o[k] &= g(i[k], s[k]) \\ s[k+1] &= f(i[k], s[k]) \end{aligned}$$

Dans le cas de l'automate de Moore (voir figure 6.2), il n'y a pas une dépendance directe entre les signaux de sortie et les signaux d'entrée. Cela est exprimé par les équations caractéristiques :

$$\begin{aligned} o[k] &= g(s[k]) \\ s[k+1] &= f(i[k], s[k]) \end{aligned}$$

Une forme particulière de l'automate de Moore c'est l'automate de Medwedjew, pour lequel les variables d'état servent directement comme les variables de sortie (voire figure 6.3). Si on renonce à l'utilisation d'un circuit logique combinatoire, on peut etre sure qu'il n'y aura pas d'aléa sur les variables de sorties. Les équations caractéristiques de l'automate de Medwedjew sont moins compliquées :

$$\begin{aligned} o[k] &= s[k] \\ s[k+1] &= f(i[k], s[k]) \end{aligned}$$

Pour tous les trois automates, le circuit combinatoire f est réalisé par une structure PLA (AND-OR).

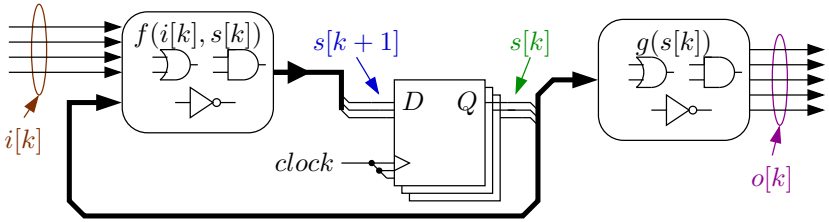


FIGURE 6.2 – Structure de l'automate de Moore.

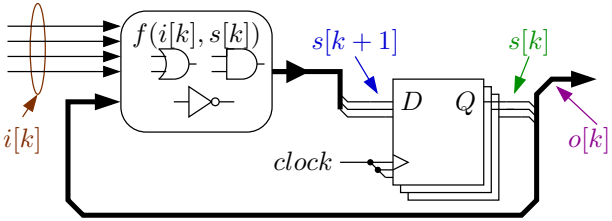


FIGURE 6.3 – L'automate de Medwedjew est un cas particulier de l'automate de Moore.

6.2 Description des séquenceurs

Pour décrire le comportement d'un séquenceur on a en principe deux p ossibilités :

- La table de transition et la table de sortie.
- Le diagramme d'état.

6.2.1 Tables de transition et de sortie

Un séquenceur peut être complètement décrit par la table de transition en combinaison avec la table de sortie. Ce mode de description est proche de celui des tables de vérité connues. La table de transition, qu'on nomme aussi la table d'état, définit la transition d'un état présent vers un état futur. Contrairement aux tables de vérité de la logique combinatoire où les sorties dépendent seulement des entrées, la table de transition doit décrire les valeurs de sortie dépendantes des valeurs d'entrée et des états précédents. Si on regarde les structures des séquenceurs montrés aux figures 6.1, 6.2 et 6.3 et les équations corespondantes,

il est évident, que la fonction $f(i[k], s[k])$ décrit ces conditions de transition. La table de transition représente ainsi la fonction $f(i[k], s[k])$. Un exemple d'une

état présent		entrée	état futur	
nom	$s[k]$	$i[k] = (a, b)$	nom	$s[k+1]$
état-1	00	1 -	état-2	01
état-1	00	0 -	état-4	11
état-2	01	0 -	état-2	01
état-2	01	1 -	état-3	10
état-3	10	- 1	état-1	00
état-3	10	00	état-3	10
état-3	10	10	état-4	11
état-4	11	- -	état-1	00

TABLE 6.1 – Exemple d'une table de transition

table de transition simple est donné à la table 6.1. Si l'automate est à l'*état-1* et si l'entrée a est 1, il saute dans la période élémentaire suivante, *état-2*. Si l'entrée a est 0, il saute a l'*état-4*. Par cette méthode simple, chaque ligne peut être interprété comme une condition de transition. Si on a des tables de transition plus grandes, l'interprétation reste sans équivoque mais la vue d'ensemble diminue très vite. Dans les automates de Moore et de Mealy (voir figure 6.1 et 6.2) les fonctions $g(s[k])$ ou $g(i[k], s[k])$ génèrent les valeurs de sortie propres. Pour une description complète de ces deux types d'automate on a donc besoin d'une autre table, la table de sortie. Ces tables de sortie montrent le chaînage combinatoire des entrées considérées comme des tables de vérité ordinaires.

6.2.2 Diagramme d'état

Le diagramme d'état décrit la fonction d'un séquenceur d'une manière graphique. La fonctionnalité peut ainsi être visualisé très clairement. Si on utilise des dénominations qui décrivent les états et les conditions de transition, on peut travailler sur un niveau d'abstraction plus élevé. Dans le cas des séquenceurs très grands et complexes, on peut garder la bonne disposition dans chaque diagramme par un entrelacement hiérarchique de plusieurs diagrammes d'état : La Figure 6.4 montre le même exemple pour une diagramme d'état que la table de transition de la table 6.1, mais la solution est visualisée par le diagramme d'état. Dans ce dia-

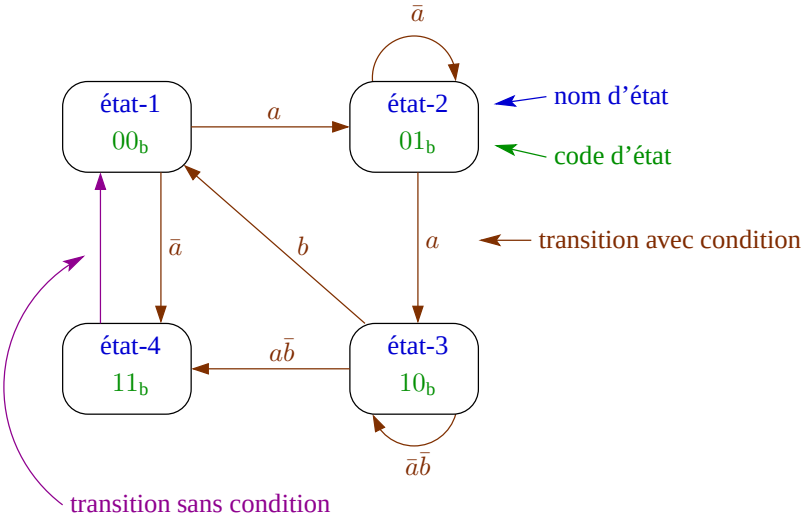


FIGURE 6.4 – Exemple d'un diagramme d'état.

gramme un état est représenté par un nœud (cercle). Les flèches représentent les transitions d'un état à un autre. Les flèches qui ne sont pas étiquettées indiquent un changement d'état sans conditions. Les flèches avec une étiquette signalent une transition avec conditions. Dans notre exemple, nous avons une transition de l'*état-3* à l'*état-4* seulement si les variables d'entrées satisfont la condition $a\bar{b} = 1$.

6.3 Analyse des séquenceurs

Pour analyser un séquenceur, on part du schéma d'où on derive la table d'état ou bien la table de sortie. A l'aide des tables élaborées on peut construire le

diagramme d'état, qui représente une description plus clair du comportement.

$$\begin{aligned} D_2 &= \overline{(Q_1 \oplus e)} \cdot \overline{Q_0} + Q_0 \cdot Q_2 \\ D_1 &= (Q_2 \oplus e) \cdot Q_0 + \overline{Q_0} \cdot Q_1 \\ D_0 &= (Q_1 \oplus Q_2) \oplus e \end{aligned}$$

La table de transition et le diagramme d'état peut être dérivées du montage

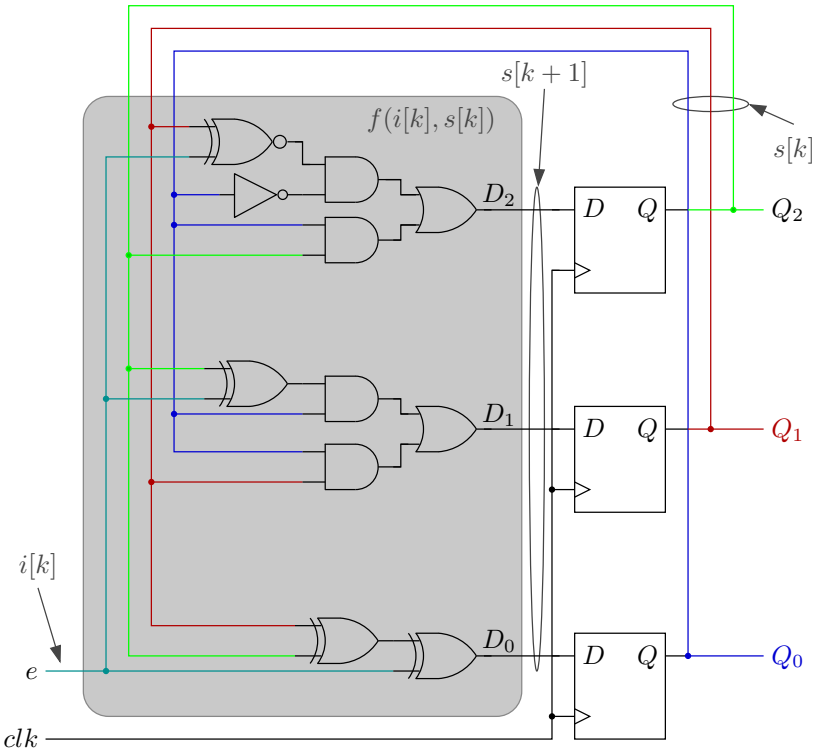


FIGURE 6.5 – Une machine d'états finis réalisez avec des D-Flip-Flops

de la figure 6.5. On peut alors determiner facilement la fonction du montage.

Tout d'abord, on pose les équations algébriques auxquelles les entrées des Flip-Flop doivent correspondre. Sur la base de ces équations on peut écrire la table

e	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	1	1
0	0	1	1	0	0	1
0	1	0	0	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	1	0
0	1	1	1	1	1	0
1	0	0	0	0	0	1
1	0	0	1	0	1	1
1	0	1	0	1	1	0
1	0	1	1	0	1	0
1	1	0	0	0	0	0
1	1	0	1	1	0	0
1	1	1	0	1	1	1
1	1	1	1	1	0	1

TABLE 6.2 – Table de transition de l'automate de la figure 6.5.

de transition, qui définie les valeurs logiques de l'entrée e et des états présents des éléments de mémoire Q^n les états futurs Q^{n+1} (D) de la mémoire (voir table 6.2). Avec les informations de la table de transition, le diagramme d'état peut être dessiné. Dans la table, huit états divers sont codés et cela correspond à huit états (nœuds) dans le diagramme d'état. Si on decrit aussi les conditions par des flèches avec ses désignations, on obtient une description complète d'un séquenceur sous forme graphique (voir figure 6.6). Le diagramme montre très bien, que le séquenceur compte positivement dans le sens horaire et de façon cyclique pour l'entrée $e = 1$, et qu'il compte à rebours pour l'entrée $e = 0$ selon le même cycle. Puisque les états des mémoires sont utilisés dans le montage en figure 6.6 directement pour les entrées, on parle d'un automate de Medwedjew.

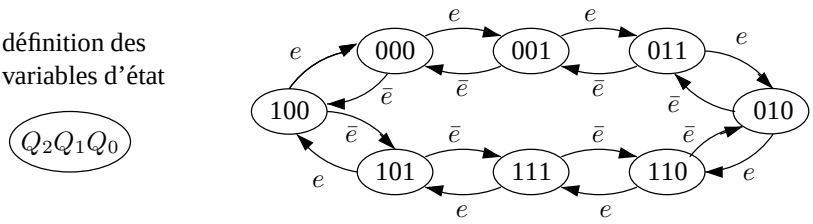


FIGURE 6.6 – Diagramme d'état de l'automate de la figure 6.5.

6.4 La synthèse des séquenceurs

La conception du séquenceur se base sur le diagramme d'état montré à la figure 6.4 (voir aussi figure 6.6). On assigne aux états, *état-1* jusqu'a *état-4*, des valeurs pour leur identification. Pour les automates de Mealy et de Moore, il en résult des circuits combinatoires optimalisés par une assignation habile des valeurs aux variables d'état. La séquence suivie par le diagramme d'état ainsi que les valeurs de sortie sont déterminées par les variables d'entrée a et b, qui doivent être dans cet exemple les mêmes, que les variables de sortie (structure Medwedjew). Pour implementer un automate à partir du diagramme d'état, on doit dériver la table de transition (voir table 6.1). Chaque ligne de la table représente une transition (flèche) d'un état présent à un état futur sur la base d'une condition de transition remplie.

6.4.1 Intégralité et consistance

En construisant le diagramme d'état, on doit respecter les règles suivantes :

- Pour chaqu'un des états, les deux conditions de transition ne peuvent être vrais en même temps. Cela veut dire, que la conjonction des deux conditions de transition quelconques (flèches), qui quittent un état (nœud), doit être 0.
- En outre, on ne peut pas avoir de combinaisons d'entrée à laquelle aucune transition n'est assignée. Cela veut dire, que la disjonction de toutes les conditions de transition (flèches quittant un noeud) doit être toujours 1.

Si une des conditions n'est pas vrai, le diagramme d'état est en contradiction ou n'est pas complet et à cause de cela n'est pas réalisable.

FIGURE 6.7 – Diagramme d'état inconsistent.

FIGURE 6.8 – Diagramme d'état incomplet.

6.4.2 Etats parasitaires

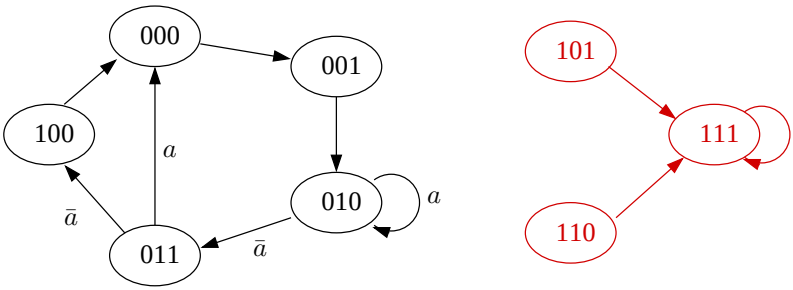


FIGURE 6.9 – Le diagramme d'état avec le diagramme d'état d'ombre des états parasitaires.

On doit être prudent envers les états parasitaires (voir figure 6.9). Ce sont des états qui ne sont pas intégrés dans la commande séquentielle. Ils apparaissent lorsqu'on n'utilise pas tous les 2^n codifications d'états possibles (pour n variables d'état). On doit s'assurer qu'on ne tombe pas dans un état parasitaire et qu'on ne peut pas exécuter des diagrammes d'état d'ombre, qui sont des états parasitaires. Le but dans chaque développement etant de concevoir le montage

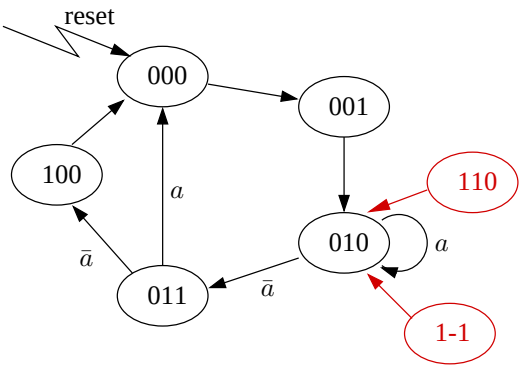


FIGURE 6.10 – Les états parasitaires de la figure 6.9 sont intégrés au diagramme d'état.

le plus robuste possible. Les conséquences des défauts temporaires doivent être limitées aux temps très court. Il existe des règles permettant de gérer les états parasitaires lors de la conception des séquenceurs :

- On doit premièrement associer tous les états parasitaires à des états vrais ou parasitaires (par exemple les états 111 et 101).
- Tous les états parasitaires doivent être menés directement au diagramme d'état (par exemple les états 110 et 010).
- On doit pouvoir sauter à létat initial depuis n'importe quel état grace au reset.

À la figure 6.10 on a associé deux des trois états parasitaires à des états parasitaires. Les deux états parasitaire mène directement au diagramme d'état. On peut sauter de plus par le signal de reset à l'état initial. Des séquenceurs volumineux ont souvent besoin d'un plan AND-OR très grand. Si on répartit un sequenceur volumineux en plusieurs petits séquenceurs, on diminue d'un côté la surface de silicium et de l'autre côté les temps de propagations du circuit combinatoire. Il n'est pas toujours bien de construire un séquenceur en utilisant une structure AND-OR. Des outils modernes de synthèse utilisant plusieurs degrés de liberté (AND, OR, EXOR, portes complexes) peuvent générer des séquenceurs compactes beaucoup plus rapidement qu'avec des structures classiques basé sur des PLA.

6.4.3 Les aléas dans les séquenceurs

Les circuits logiques combinatoires sont des sources potentielles de production d'aléas. Puisque dans un séquenceur il existe des blocs combinatoires, on risque d'avoir des états indésirables. Si on regarde les structures des séquenceurs montrées aux figures 6.1 jusqu'à 6.3, on constate que seulement le bloc combinatoire g peut produire des aléas à la sortie. L'automate de Medwedjew ne produit pas d'aléas. Pour les autres types on peut supprimer les aléas en utilisant un registre, mais il en résulte une temps de retard supplémentaire.

6.4.4 Synchronisation

Si on regarde les structures d'implémentation d'une machine d'états finis (figures 6.1 jusqu'à 6.3) on se rend compte, que le bloc combinatoire g augmente le temps de setup des entrées. Tant que les entrées de l'automate changent de faccon synchrone, il ne peut pas apparaître des violations des temps de setup. Des problèmes surviennent, si les valeurs d'entrées peuvent changer de façon asynchrone. A cause des durées différentes des parcours dans le bloc combinatoire, il peut apparaître pendant un court instant aux entrées des registres des combinaisons des valeurs d'entrée inattendues, qui peuvent mener a un mauvais état (éventuellement un état parasitaire). Une autre possibilité pour résoudre le

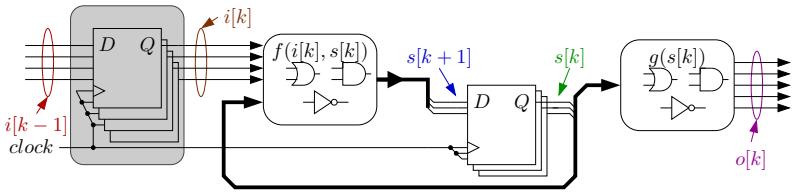


FIGURE 6.11 – Synchronisation de l'automate de Moore.

problème des entrées asynchrones dans les séquenceurs c'est la synchronisation de ces signaux. La figure 6.11 montre comment on peut réaliser un synchronisation avec un registre placé en amont d'un automate de Mealy. On paye une sécurité plus haute avec un temps de retard plus long.

6.4.5 Exemple de conception : Commande des feux de signalisation

Pour une commande simple de feux de signalisation on doit synthétiser un automate de Moore. Le fonctionnement des feux se décrit comme suit.

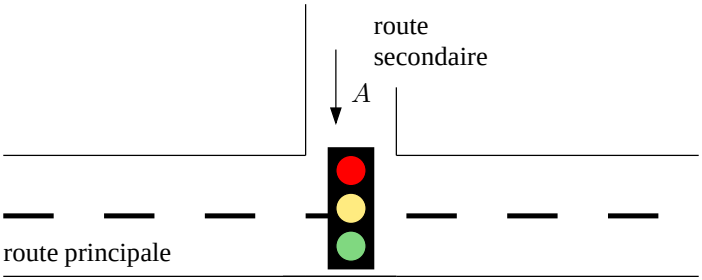


FIGURE 6.12 – Commande des feux de signalisation avec une route principale H et une route secondaire L

Une rue principale très fréquentée a un croisement avec une rue latérale. La rue latérale équipée d'un détecteur, annonce la présence des véhicules ($A = 1$, véhicule attendant). S'il n'y a pas des véhicules sur la rue latérale, le signal de la rue principale doit être vert ($H_G = 1$) et celui de la rue latérale doit être rouge ($L_R = 1$). Si un véhicule attend sur la rue latérale, le signal de la rue principale change à l'orange ($H_O = 1$) et au prochain coup d'horloge, au rouge ($H_R = 1$). En même temps le signal de la rue latérale change au vert ($L_G = 1$). Après deux unités de temps le signal de la rue latérale change à l'orange ($L_O = 1$) et après au rouge, si sur la rue latérale n'apparaissent pas de véhicules. Sinon le signal de la rue latérale reste au total au maximum 3 unités de temps au vert. Le signal de la rue latéral peut au plus tôt après 4 cycles rouge commuter au vert.

Le premier pas pour développer un automate est la constitution du diagramme d'état. Selon les règles de la définition du problème, les états et leurs transitions sont représentés graphiquement. Puisque dans cet exemple il y a seulement huit états differents, il sont codifiés par 3 bit, ce qui représente des valeurs de sortie des registres. Les signaux necessaires pour les deux feux de signalisation avec les trois lampes doivent être des états. Pour la réalisation de ce montage on utilise un automate de Moore. À partir du diagramme d'état, la table de transition peut

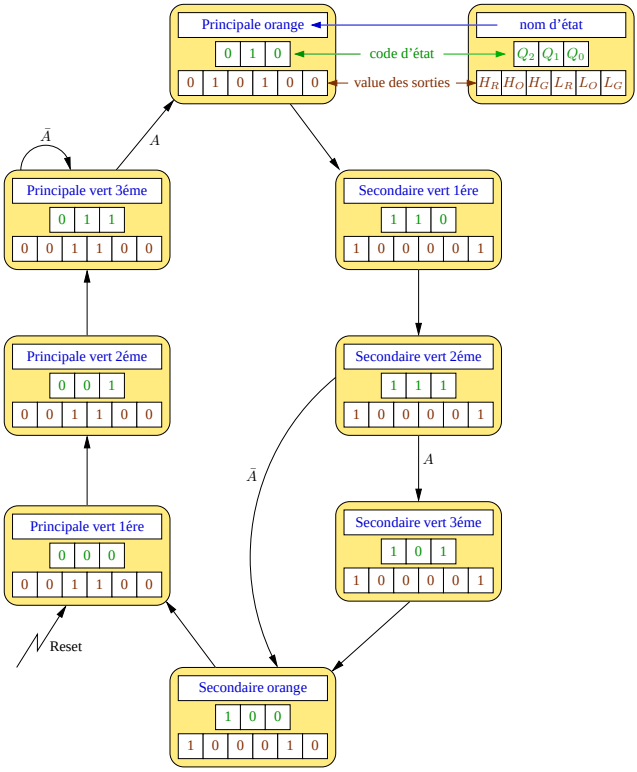


FIGURE 6.13 – Diagramme d'état de la commande des feux de signalisation.

être dérivée simplement en créant pour chaque condition de transition une ligne dans la table. La table 6.3 montre la table de transition correspondante. On doit seulement générer par des bits d'état codifiés, la commande pour les feux tricolores de circulation. Pour cela on crée une table des valeurs de sortie. la table 6.4 montre une telle table. Les trois bits d'état Q_i sont les valeurs d'entrées de la table de vérité, les variables de commande des signaux lumineux sont les valeurs de sortie. Avec la table des valeurs de sortie et le diagramme d'état ou la table de transition, la fonction d'automate de Moore est déterminé sans problème. Après simplifications sur des diagrammes de Karnaugh des deux tables, on peut

A	Q_2	Q_1	Q_0	D_2	D_1	D_0
-	0	0	0	0	0	1
-	0	1	0	1	1	0
-	1	0	0	0	0	0
-	1	0	1	1	0	0
-	1	1	0	1	1	1
-	0	0	1	0	1	1
0	0	1	1	0	1	1
1	0	1	1	0	1	0
0	1	1	1	1	0	0
1	1	1	1	1	0	1

TABLE 6.3 – Table de transitions d'état du problème.

Q_2	Q_1	Q_0	H_G	H_O	H_R	L_G	L_O	L_R
0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	0	1
0	1	1	1	0	0	0	0	1
0	1	0	0	1	0	0	0	1
1	1	0	0	0	1	1	0	0
1	1	1	0	0	1	1	0	0
1	0	1	0	0	1	1	0	0
1	0	0	0	0	1	0	1	0

TABLE 6.4 – Table des valeurs de sortie du problème.

maintenant dessiner un circuit logique pour l'installation des feux lumineux. On obtient par la table les relations :

$$
\begin{aligned}
D_2 &= \overline{Q_0}Q_1 + Q_0Q_2 \\
D_1 &= \overline{Q_0}Q_1 + Q_0\overline{Q_2} \\
D_0 &= \overline{Q_1}\,\overline{Q_2} + \overline{A}Q_0\overline{Q_2} + \overline{Q_0}Q_1Q_2 + AQ_1Q_2
\end{aligned}
$$

Par la table des valeurs de sortie on obtient les équations de Boole suivantes très simples :

$$\begin{array}{lcllcl} H_G & = & Q_0\overline{Q_2} + \overline{Q_1}\ \overline{Q_2} & L_G & = & Q_0Q_2 + Q_1Q_2 \\ H_O & = & \overline{Q_0}Q_1\overline{Q_2} & L_O & = & \overline{Q_0}\ \overline{Q_1}Q_2 \\ H_R & = & Q_2 & L_R & = & \overline{Q_2} \end{array}$$

À partir de ces équations simplifiées, on peut directement dessiner le schéma logique de cette commande sous la forme d'un automate de Moore, comme montré en figure 6.14. les machines d'états finis peuvent être utilisés pour des

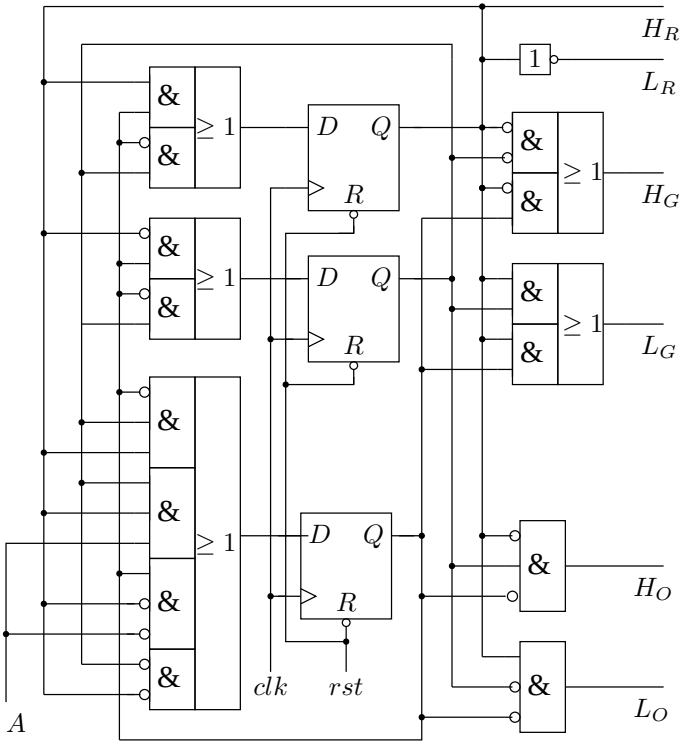


FIGURE 6.14 – Le schema de la commande des feux de signalisation.

problèmes divers. On peut construire des compteurs de modes très différents comme montré à la figure 6.6. Le domaine pricipal d'application des machine d'états finis sont les commandes séquentielles, que l'on trouve dans pratiquement tout montage numérique complexe. Le processus et le fonctionnement d'ensemble est contrôlé par des signaux de commande, qui sont générés par l'automate. Un exemple très simple est la commande des signaux lumineux traîtée, puisque on crée seulement la commande pour les lampes.

Bibliographie

[Beuth, 1990] Beuth, K. (1990). *Digitaltechnik*. Vogel Fachbuch / Elektronik 4, 7. auflage edition.

[Borucki, 1989] Borucki, L. (1989). *Digitaltechnik*. Verlag B. G. Teubner Stuttgart.

[Hayes, 1993] Hayes, J. P. (1993). *Introduction to Digital Logic Design*. Addison-Wesley Publishing Company.

[Mange, 1978] Mange, D. (1978). *Traité d'Électricité, Analyse et synthése des systèmes logiques*. Editions Georgi.

[Vahid, 2011] Vahid, F. (2011). *Digital Design with RTL Design, VHDL, and Verilog*. John Wiley & Sons, Inc., 2nd edition edition.

[Wakerly, 2005] Wakerly, J. F. (2005). *Digital Design, Priciples and Practices*. Prentice Hall, 2nd edition edition.