Cours ELE2300 (3.313)

SYSTÈMES LOGIQUES I

Quatrième édition

Jean-François Harvey
Mohamad Sawan

Département de génie électrique
et de génie informatique

ÉCOLE POLYTECHNIQUE DE MONTRÉAL
Septembre 1999

PRÉFACE

L’apprentissage des concepts relatifs aux Systèmes Logiques fait l'objet de recherches pédagogiques diversifiées. De nombreux ouvrages paraissent chaque année apportant des variantes plus ou moins accentuées sur les nombreux aspects de la logique.

Au moment d'aborder ce domaine, on a déjà développé des connaissances sur les systèmes linéaires dont les variables d'entrées et sorties sont continues et s'expriment analytiquement. Les relations entrées/sorties sont définies par des équations différentielles ou par leurs transformées.

En logique les variables sont discontinues et ces fonctions de transfert ne s'expriment plus en termes de temps ou de fréquences, mais sous forme de tables ou de relations booléennes.

L'étude des systèmes logiques implique donc une représentation différente des variables et des relations entrées/sorties. Par conséquent, ces méthodes d'analyse et de synthèse de ces systèmes impliquent des postulats, des conventions, des hypothèses, un formalisme et des algorithmes, qui n’ont rien à voir avec le domaine linéaire.

Ce document est une contribution au processus d’apprentissage relatif à ces méthodes. Il couvre de façon modulaire les principaux sujets qui doivent être traités dans un cours introductif portant. sur les Systèmes Logiques, dans un ordre qui rencontre nos vues sur l'aspect pédagogique que nous avons développé au cours des années.

Table des matières

CHAPITRE 1 : INTRODUCTION 1

1.1 OBJECTIFS 1

1.1.1 Objectifs généraux 1

1.1.2 Objectifs spécifiques 2

1.2 PLAN DU COURS 3

1.3 CONCEPTS FONDAMENTAUX 4

CHAPITRE 2 : LA LOGIQUE COMBINATOIRE 7

2.1 ÉLÉMENTS DE BASE 7

2.2 ÉCRITURE ET LECTURE DE SCHÉMAS 9

2.3 ALGÈBRE DE BOOLE 11

2.3.1 Propriétés de l'algèbre booléenne 12

2.4 SYNTHÈSE ET SIMPLIFICATION 16

2.4.1 Karnaugh 17

2.4.2 Quine/McCluskey 26

2.4.3 Table de Karnaugh à variable inscrite 33

2.5 ÉLÉMENTS DE BASE II 35

2.6 MODULES ET FAMILLES ÉLECTRONIQUES 42

2.6.1 Modules et éléments logiques 42

2.6.2 Familles logiques 43

2.6.3 Spécifications électriques 45

CHAPITRE 3 : LA LOGIQUE MIXTE 49

3.1 LOGIQUE POSITIVE 49

3.2 LOGIQUE NÉGATIVE 49

3.3 LOGIQUE MIXTE 49

3.4 IMPACT DES CONVENTIONS 50

Table des matières (suite)

3.5 SYMBOLES ET NOTATION.....51

3.5.1 La barre oblique.....51

3.5.2 Le OOPS.....52

3.5.3 Les équivalences de portes.....53

3.5.4 Avantages de la logique mixte.....54

3.5.4 Analyse de schéma en logique mixte.....56

3.5.5 Synthèse de schéma en logique mixte. 57

CHAPITRE 4 : LES RÉSEAUX ITÉRATIFS.....61

4.1 INTRODUCTION 61

4.2 FORME GÉNÉRALE DES RÉSEAUX ITÉRATIFS 62

4.3 MÉTHODOLOGIE DE DESIGN.....63

CHAPITRE 5 : CODES ANALYTIQUES ET REPRÉSENTATIFS.....71

5.1 DÉFINITIONS 71

5.2 CODES FRÉQUENTS.....71

5.2.1 Binaire naturel 71

5.2.2 Binaire naturel avec signe 72

5.2.3 Complément à 1.....72

5.2.4 Complément à 2.....73

5.2.5 Code hexadécimal 75

5.2.6 Code octal.....75

5.2.7 Code BCD (Binary Coded Decimal).....76

5.2.8 Code de Gray 76

5.3 APPLICATION DES CODES À LA DÉTECTION ET CORRECTION D'ERREURS 77

5.3.1 Définitions.....77

5.3.2 Redondance.....81

5.3.3 Parité orthogonale.....83

5.3.4 Code de Hamming 85

Table des matières (suite)

CHAPITRE 6 : LA LOGIQUE SÉQUENTIELLE 90

6.1 DÉFINITIONS 90

6.2 SYSTÈMES ASYNCHRONES 92

6.2.1 Analyse 92

6.2.2 Synthèse 95

6.3 ÉLÉMENTS DE MÉMOIRE 104

6.4 SYSTÈMES SYNCHRONES 109

6.4.1 Synthèse 109

6.4.2 Analyse 111

CHAPITRE 7 : UNITÉ DE COMMANDE INDUSTRIELLE (ICU MC14500) 114

7.1 DESCRIPTION 114

7.2 CONSTITUANTS 115

7.3 LES INSTRUCTIONS 117

7.4 LIMITATIONS 119

7.5 ÉCRITURE DE PROGRAMMES 119

CHAPITRE 8 : INITIATION À LA LOGIQUE PROGRAMMABLE 122

8.1 INTÉRÊTS DE LA LOGIQUE PROGRAMMABLE 122

8.2 FAMILLES DE LOGIQUE PROGRAMMABLE 123

Liste des figures

Figure 2.1: Équation et représentation schématique d'une fonction logique 10
Figure 2.2: Schéma du circuit de l'exemple 2.1 11
Figure 2.3: Schéma de l'exemple 2.3 15
Figure 2.4: Schéma simplifié de l'exemple 2.3 16
Figure 2.5: Tables de Karnaugh à 2, 3, 4 et 5 variables 20
Figure 2.6: Formation d'impliquants dans les tables de Karnaugh 21
Figure 2.7: Remplissage de la table de Karnaugh 22
Figure 2.8: Formation des impliquants 22
Figure 2.9: Formation optimale des impliquants 22
Figure 2.10: Effets d'un impliquant redondant 24
Figure 2.11: Symétrie de la table de Karnaugh 25
Figure 2.12: Symétrie de la table de Karnaugh 25
Figure 2.13: Symétrie de la table de Karnaugh 26
Figure 2.14: Table de Karnaugh à variable inscrite 34
Figure 2.15: Représentation générale d'un multiplexeur 36
Figure 2.16: Caractéristiques du 74153 36
Figure 2.17: Représentation générale d'un démultiplexeur 37
Figure 2.18: Caractéristiques d'un démultiplexeur 37
Figure 2.19: Représentation générale d'un encodeur 38
Figure 2.20: Caractéristiques du 74148 38
Figure 2.21: Représentation générale d'un décodeur 39
Figure 2.22: Caractéristiques d'un décodeur 1 parmi 4 39
Figure 2.23: Représentation générale des mémoires ROM et RAM 40
Figure 2.24: Différence de comportement entre une bistable et une bascule 41
Figure 2.25: Comportement d'un compteur 4 bits 41
Figure 2.26: Réduction du nombre de modules logiques 43
Figure 2.27: Élément logique est ses spécifications électriques 45
Figure 2.28: Spécifications électriques du 7400 47
Figure 2.29: Branchement de charge sur des modules logiques 48

Liste des figures (suite)

Figure 3.1: Symboles de la logique mixte 51
Figure 3.2: Utilisation du symbole de changement de convention 51
Figure 3.3: Utilisation du OOPS 53
Figure 3.4: Schéma de génération de l'inverse de A 53
Figure 3.5: Schéma de l'exemple 3.2 57
Figure 3.6: Première étape de la synthèse en logique mixte 58
Figure 3.7: Troisième étape de la synthèse en logique mixte 58
Figure 3.8: Schéma final de l'exemple 3.3 59
Figure 3.9: Troisième étape de la synthèse en logique mixte 59
Figure 3.10: Schéma final de l'exemple 3.4 60

Figure 4.1: Forme générale d'un réseau itératif spatial 62
Figure 4.2: Schéma bloc de l'exemple 4.1 66
Figure 4.3: Table de Karnaugh de l'exemple 4.1 67
Figure 4.4: Schéma d'une cellule du comparateur 67
Figure 4.5: Schéma bloc de la solution 2 68
Figure 4.6: Schéma de la solution 2 70

Figure 5.1: Schéma de conversion du code binaire vers le code de Gray 77
Figure 5.2: Système de transmission binaire 78
Figure 5.3: Canaux bruyants 78

Figure 6.1: Réseau séquentiel simple 90
Figure 6.2: Aperçu général des machines de Mealy et de Moore 92
Figure 6.3: Machine asynchrone 93
Figure 6.4: Diagramme de fusionnement de l'exemple 6.4 99
Figure 6.5: Diagramme des transitions 100
Figure 6.6: Schéma du circuit de l'exemple 6.4 103
Figure 6.7: Formes directe et optimisée d'une bascule S/R 105

Liste des figures (suite)

Figure 6.8: Schéma d'une bascule SR avec horloge et initialisation 106
Figure 6.9: Schéma d'une bistable 107
Figure 6.10: Schéma d'une bascule JK 108
Figure 6.11: Diagramme d'états de l'exemple 6.5 110
Figure 6.12: Schéma de l'exemple 6.6 112
Figure 6.13: Séquence des états de l'exemple 6.6 113

Figure 7.1: Schéma bloc d'un système basé sur le 14500 115
Figure 7.2: Vue interne du 14500 117
Figure 7.3: Entrées/sorties du système utilisé pour les exemples 119

Figure 8.1: Architectures de composants programmables 124
Figure 8.2: Solutions de l'exemple 8.1 125

Liste des tableaux

Tableau 2.1: Code binaire naturel 18
Tableau 2.2: Code de Gray à 1, 2 et 3 bits 19
Tableau 2.3: Principales caractéristiques des familles logiques 46

Tableau 3.1: Convention des logiques positive et négative 50
Tableau 3.2: Impact des conventions sur un 7408 50
Tableau 3.3: Effets du changement de convention 52
Tableau 3.4: Effets du changement de convention 52
Tableau 3.5: Effets du OOPS 53
Tableau 3.6 : Équivalence des portes en logique positive et négative 54
Tableau 3.7: Équivalence des portes de base 55
Tableau 3.8: Clarté de la logique mixte 55
Tableau 3.9: Optimisation de circuits avec la logique mixte 56

Tableau 5.1: Code de gray à 4 bits 76

Tableau 6.1: Tableau d'excitation complet 101
Tableau 6.2: Synthèse des variables d'états 102
Tableau 6.3: Excitation et sortie d'une bascule S/R 105
Tableau 6.4: Table de transitions des différentes bascules 108

CHAPITRE 1

Introduction

1.1 Objectifs

1.1.1 Objectifs généraux

L'étude d'une matière est une occasion pour un étudiant d'enrichir sa culture, d'élaborer sa formation et de pousser son entraînement afin de développer des attributs, des habiletés et des comportements qui seront appréciés au cours de sa pratique éventuelle de la profession d'ingénieur.

Le diplôme d'ingénieur ouvre une fenêtre sur un vaste choix de carrières. Il implique généralement ce dernier dans diverses responsabilités de gestion, de développement, de recherche ou autres.

Dans l'ensemble de ses fonctions, l'ingénieur est un agent de progrès qui influence de façon significative l'évolution et la qualité de vie de son entourage. Ses actions et ses décisions risquent d'avoir un impact sur tout un éventail de domaines, dont:

- l'innovation et le progrès technique;
- l'évolution compétitive des méthodes industrielles;
- la santé économique des entreprises;
- la gestion efficace des ressources;
- la protection des personnes et de l'environnement.

Cette liste, qui ne prétend pas être exhaustive, suggère que l'appréciation d'un ingénieur repose aussi sur un éventail de compétences, d'habiletés et de qualités dont l'acquisition fait l'objet de l'entraînement qu'il s'impose au cours de ses années de formation. Il n'est pas facile de faire une revue complète de ces attributs et attitudes, mais dans les grandes lignes, on peut citer :

- des connaissances acquises dans le domaine des sciences appliquées;
- une créativité résultant d'un esprit alerte;

- une méthode de travail ordonnée;
- une capacité de planification et un sens de l'organisation;
- une volonté d'assumer des responsabilités;
- un comportement intègre envers son institution et la société.

On peut toujours ajouter des items à cette liste, mais si un individu excelle dans toutes ces rubriques, il y a fort à parier qu'il aura un choix de propositions intéressantes devant lui en temps opportun.

Votre curriculum, dans son ensemble, doit fournir des terrains de développement pour combler les besoins d'entraînement relatifs à la commande détaillée ci-dessus.

Systèmes Logiques I, qui fait partie de cet ensemble, est un cours de nature introductive. Il apporte des connaissances nouvelles, exige un comportement méthodique et propose des projets qui en font une contribution enrichissante dans la poursuite des objectifs énumérés.

D'un aspect pratique, les systèmes logiques sont à la base d'un grand nombre d'applications dans une variété de domaines. On ne peut en faire le tour complet, mais on peut citer quelques exemples :

- l'instrumentation électronique;
- le matériel informatique;
- les systèmes de communications numériques;
- la gestion de l'énergie électrique;
- la commande des machines-outils;
- l'instrumentation biomédicale.

1.1.2 Objectifs spécifiques

Plus près de nous, sur le plan académique, le cours Systèmes Logiques I se veut une introduction générale au domaine numérique.

Un objectif immédiat du cours vise à développer l'habileté à :

- percevoir le sens des définitions et concepts de base de la logique;
- exploiter des méthodes de design et d'analyse appliquées :

  - aux systèmes combinatoires (sans mémoire),
  - aux systèmes itératifs,
  - aux systèmes séquentiels synchrones ou asynchrones(avec mémoire),
- étudier et exploiter un processeur simple;
- concevoir des systèmes fonctionnels (ALU, fréquencemètre...);
- sélectionner des composants pour réaliser un système;
- interpréter les spécifications des fabricants;
- prévoir et observer l'évolution temporelle de signaux;
- simuler ces systèmes avec un outil industriel (*ViewLogic*).

1.2 Plan du cours

Le cours :

- Nombre de crédits : 3 crédits (3-1, 5-4, 5)
- Cours corequis 3.310 – Introduction aux circuits électriques ou,

  ELE2301 (3.315) – Circuits électroniques ou,

  ELE2602 (3.352) – Travaux pratiques de circuits électriques.
- Applications et approfondissements éventuels :
  - ELE4301 (3.413) - Systèmes logiques II
  - ELE4302 (3.553) - Introduction aux microprocesseurs
  - ELE4306 (3.565) - Électronique des communications
- Site Internet : http://www.gegi.polymtl.ca/cours/3313/

MÉTHODES D'ÉVALUATION

- Quiz 35%
- Laboratoires l à 5 inclusivement 15%
- Final 50%

100%

Cours :

- Trois heures par semaine;
- Communication des informations théoriques et pratiques;
- Illustration à l'aide de multiples exemples;
- Préparation pour les laboratoires;
- Présence et participation fortement suggérées.

Laboratoires :

- Trois heures par quinzaine;
- 5 TP au cours du trimestre;
- 10 équipes de deux étudiants max. par section de lab.;
- 1 préparation par équipe;
- 1 rapport à remettre par équipe à partir du TP3;
- points accordés au laboratoire pour la préparation et le fonctionnement;
- local du laboratoire : A-210;
- pondération : 15% de la note finale.

Le quiz, d'une durée de 50 minutes, a habituellement lieu la semaine 8 (voir calendrier du cours remis en classe) en début de soirée (jour et heure à confirmer) et portera sur la logique combinatoire ainsi que sur les laboratoires 1 et 2. L'examen final portera sur toute la matière vue en classe et au laboratoire.

1.3 Concepts fondamentaux

- La logique n'est pas une étude et une modélisation de système de la nature comme la physique, la chimie ou la biologie.
- La logique est une technique élaborée par l'humanité pour se doter d'outils dédiés à diverses fonctions.
- La logique implique un certain nombre de concepts fondamentaux dans le but de synthétiser systématiquement des systèmes.

Dans ce qui suit, les concepts fondamentaux des systèmes logiques sont présentés.

Variable continue (infinité de valeurs)

Quantité, représentée par un symbole, qui peut prendre une infinité de valeurs.

$y = ax^3 + bx^2 + c$ où

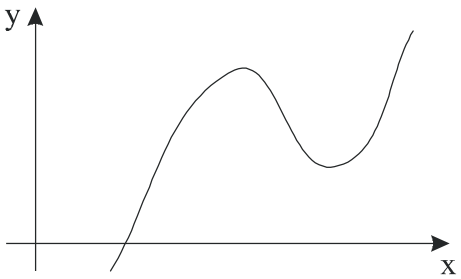
y est la fonction,

= est le symbole d'égalité,

x est la variable,

+ est un opérateur,

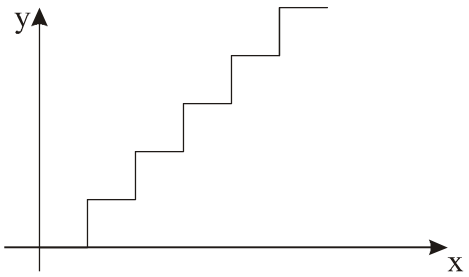
a, b et c sont des constantes



Variable discrète

Variable susceptible de prendre un nombre limité de valeurs prédéfinies et discontinues.

$y = \text{int}(x)$ (partie entière de x)



Variable binaire

Variable discrète qui ne peut prendre que deux et seulement deux valeurs.

Variable logique

Variable binaire qui peut prendre deux états associés au caractère vrai ou faux d'un événement.

État Logique

Valeur attribuée à une variable logique. L'état d'une variable peut être vrai ou faux. On représente l'état vrai par "1" et l'état faux par "0". Une variable dans son état vrai est dite "active".

Opérateurs Logiques

Les opérateurs logiques de base sont ET, OU et NON.

Fonction Logique

Ensemble de variables logiques reliées par des opérateurs logiques. Une fonction logique ne peut prendre que deux valeurs: 0 ou 1.

Signal Logique

Quantité physique qui représente une variable logique dans l'un ou l'autre de ses deux états possibles.

Système Logique

Ensemble de composants qui effectuent des fonctions sur des signaux logiques dans le but de stocker, communiquer ou de transformer de l'information.

CHAPITRE 2

La logique combinatoire

2.1 Éléments de base

Variable d'entrée

Les variables d'entrée sont celles sur lesquelles on peut agir directement. Ce sont des variables logiques indépendantes.

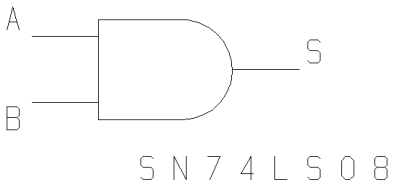
Variable de sortie

Variable contenant l'état de la fonction après l'évaluation des opérateurs logiques sur les variables d'entrée.

ET

S = A•B = AB S est vraie si A est vraie et B est vraie.

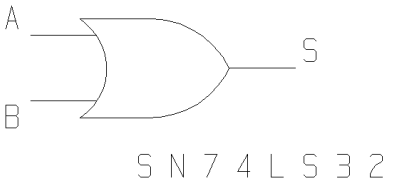
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1



OU

S = A + B S est vraie si A est vraie ou B est vraie, ou les deux.

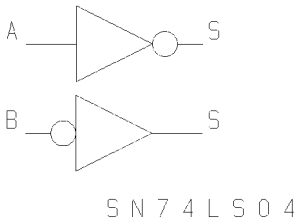
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1



NON

$S = \overline{A}$ S est vraie si A est fausse

A	S
0	1
1	0

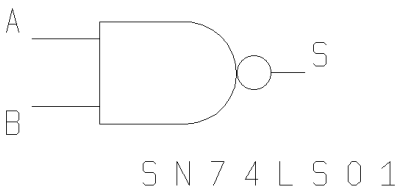


NON-ET

$S = \overline{A \bullet B} = \overline{AB}$ S est vraie si (A et B) est fausse.

S est vraie si A est fausse ou B est fausse, ou les deux.

A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

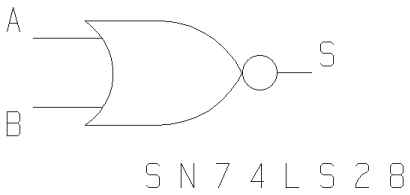


NON-OU

$S = \overline{A + B}$ S est vraie si (A ou B) est fausse.

S est vraie si A est fausse et B est fausse.

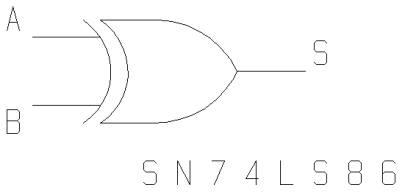
A	B	S
0	0	1
0	1	0
1	0	0
1	1	0



XOR (OU EXCLUSIF)

$S = A \oplus B$ S est vraie si A est vraie ou B est vraie, mais pas les deux.

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

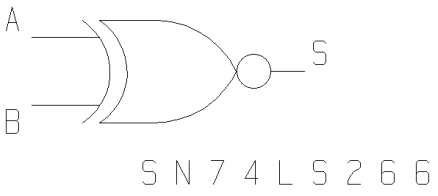


Note: L'expression $S = A \oplus B$ est non-analytique. L'expression analytique du ou exclusif est $S = \overline{A}B + A\overline{B}$.

XNOR (NON-OU EXCLUSIF)

$S = A \otimes B = \overline{A \oplus B}$ S est vraie si A est fausse ou B est fausse, mais pas les deux.

A	B	S
0	0	1
0	1	0
1	0	0
1	1	1



Note: L'expression $S = A \otimes B$ est non-analytique. L'expression analytique du non ou exclusif est $S = AB + \overline{A}\,\overline{B}$.

Généralisation

À l'exception des portes XOR et XNOR, ces notions peuvent être généralisées pour des portes à plusieurs entrées.

2.2 Écriture et lecture de schémas

Les étapes de transformation schématique d'une équation, ainsi que celles d'écriture d'une équation à partir du schéma, sont très directes puisque les deux représentations doivent strictement conformes. Dans le cas de la création du schéma à partir de l'équation, chaque opération est remplacée par son symbole en respectant la hiérarchie des opérations.

Pour trouver l'équation de la fonction réalisée à partir du schéma, il convient de noter l'équation à chaque nœud du schéma en fonction des entrées et de l'opération logique.

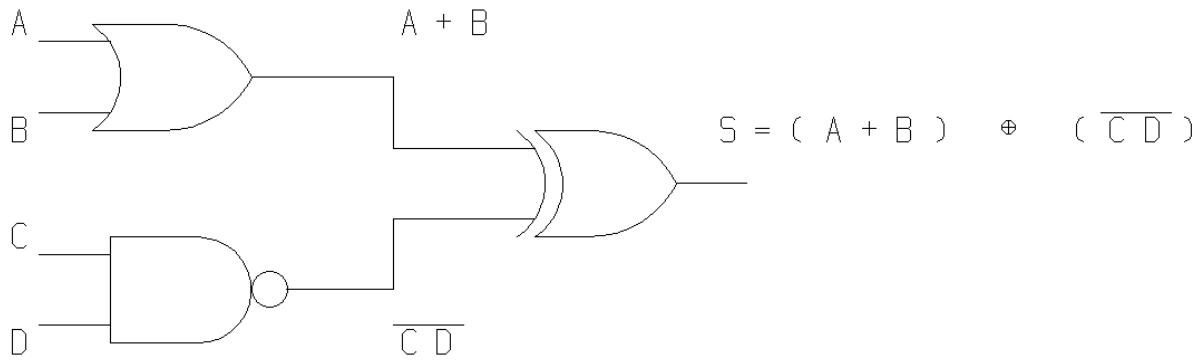


Figure 2.1: Équation et représentation schématique d'une fonction logique.

Exemple 2.1

On veut concevoir un système qui commande la lampe L à partir de trois commutateurs (CA, CB et CC). Les commutateurs sont activés par un levier à deux positions. La lampe doit être éteinte si tous les commutateurs sont en position basse. Aussitôt qu'un commutateur est changé, l'état de la lampe s'inversera.

Variables d'entrée

On peut définir:

- A = Le levier du commutateur CA est haut (actif).
- B = Le levier du commutateur CB est haut (actif).
- C = Le levier du commutateur CC est haut (actif).

Variable de sortie

- L = La lampe est allumée.

Une analyse rapide permet de déterminer que la lampe sera allumée (active) si un des trois commutateurs est actif ou si les trois sont actifs.

Donc: $L = ABC + A\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}\overline{B}C$

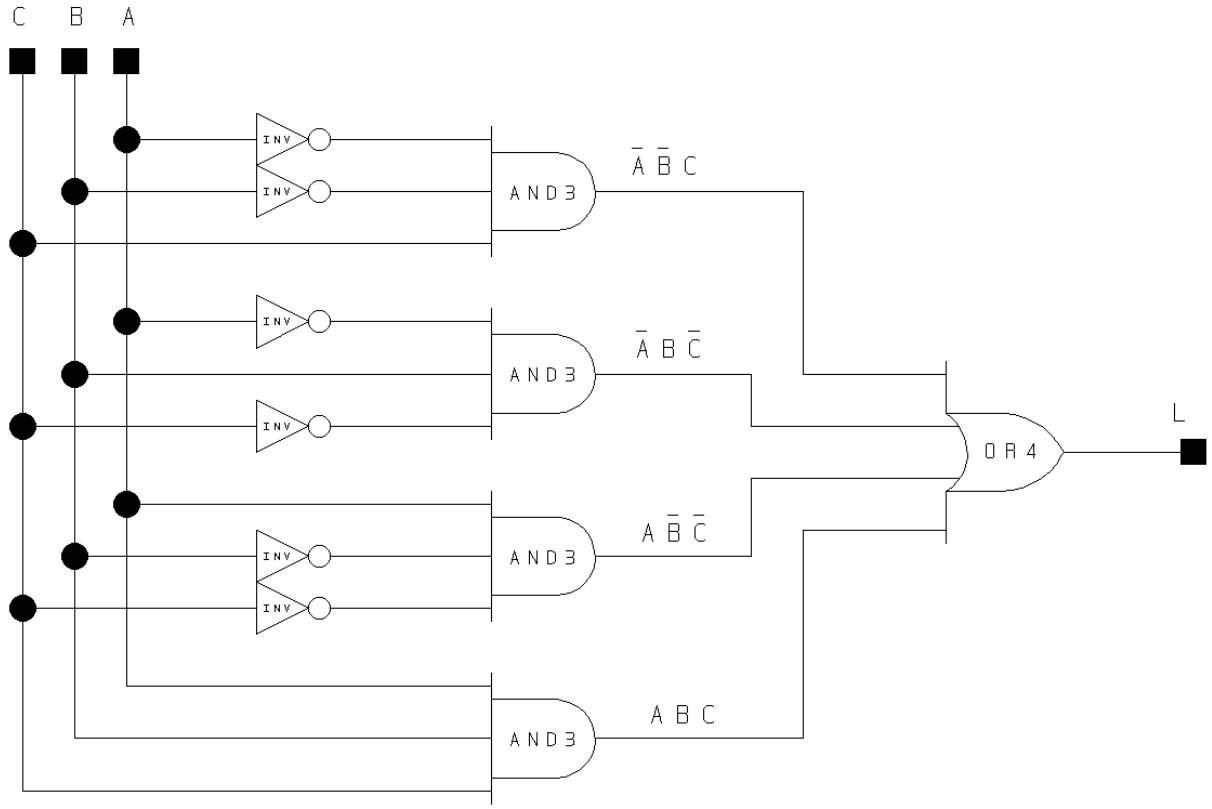


Figure 2.2: Schéma du circuit de l'exemple 2.1.

Notez que ce schéma n'est pas la représentation la plus simple de la fonction réalisée. Des méthodes de simplifications avancées anisi que des notations plus compactes sont le sujet des prochaines sections.

2.3 Algèbre de Boole

Avoir la représentation la plus simple possible d'une fonction est avantageux. Premièrement, les équations les plus simples sont plus faciles à manipuler et le risque de faire des erreurs est réduit. Deuxièmement, une équation simplifiée implique un nombre réduit de composants électroniques, ce qui a l'avantage d'augmenter la fiabilité et de réduire les coûts de fabrication.

La première méthode de simplification consiste à utiliser intuitivement les propriétés définies par l'algèbre booléenne en effectuant des mises en facteurs ou autres manipulations pouvant conduire à la simplification de la fonction. Pratiquement, l'utilisation de cette méthode est limitée par la complexité des équations à manipuler lorsque le nombre de variables est grand.

2.3.1 Propriétés de l'algèbre booléenne

$A + 0 = A$ $\qquad$ $A \bullet 0 = 0$ $\qquad$ $A = \overline{\overline{A}}$

$A + 1 = 1$ $\qquad$ $A \bullet 1 = A$

$A + A = A$ $\qquad$ $A \bullet A = A$

$A + \overline{A} = 1$ $\qquad$ $A \cdot \overline{A} = 0$

Hiérarchie des opérations

Dans une expression sans parenthèses, on effectue d'abord les opérations "ET" et, par la suite, les "OU".

Induction parfaite

Dans le domaine linéaire, il n'est pas possible de prouver une équation en la vérifiant pour toutes les valeurs des variables. En logique, puisque les variables sont limitées à deux états, on peut prouver une relation en la vérifiant pour toutes les combinaisons de valeurs pour les variables d'entrée.

Équivalence

Deux fonctions sont équivalentes si on peut leur faire correspondre la même table de vérité.

Si $\quad F = \overline{A \cdot B}$

et $\quad G = \overline{A} + \overline{B}$

alors $\quad F = G$

et on dit que F est équivalente à G.

Complémentarité

Deux fonctions sont dites complémentaires si l'une est l'inverse de l'autre pour toutes les combinaisons d'entrées possibles.

Si $F = \overline{A} \cdot \overline{B}$

et $G = A + B$

alors $F = \overline{G}$

et on dit que F et G sont complémentaires.

Dualité

Deux expressions se correspondent par dualité si l'on obtient l'une en changeant dans l'autre, les "ET" par des "OU", les "OU" par des "ET", les "1" par des "0" et les "0" par des "1".

Si on sait que $\overline{A \cdot B} = \overline{A} + \overline{B}$

Alors, on saura que $\overline{A + B} = \overline{A} \cdot \overline{B}$

par dualité.

Associativité

$A + B + C = (A + B) + C = A + (B + C)$

$A \bullet B \bullet C = (A \bullet B) \bullet C = A \bullet (B \bullet C)$

Commutativité

$A + B = B + A$

$A \bullet B = B \bullet A$

Distributivité

$A \bullet (B + C) = A \bullet B + A \bullet C = AB + AC$

Théorème de DeMorgan

Première Forme: $\overline{A+B+C+...} = \overline{A} \bullet \overline{B} \bullet \overline{C} \bullet ...$

Deuxième Forme: $\overline{A \bullet B \bullet C \bullet ...} = \overline{A} + \overline{B} + \overline{C} + ...$

Forme canonique

Une expression est sous sa forme canonique si tous les symboles qui représentent les variables apparaissent dans tous les termes qui la constitue. Lorsqu'une équation est écrite à partir de sa table de vérité, elle est dans sa forme canonique.

Si une fonction est une somme de produits, on a une somme canonique.

Exemple: $F = \overline{A}BC + ABC + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C}$

Si une fonction est un produit de somme, on a un produit canonique.

Exemple: $G = (\overline{A} + B + C) \cdot (A + B + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C})$

Exemple 2.2

Trouvez la somme et le produit canonique de la fonction suivante:

A	B	C	L
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$L = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$$

$$M = \overline{L} = \overline{A}\overline{B}\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C}$$

$$L = \overline{M} = \overline{\overline{A}\overline{B}\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C}}$$

$$L = (\overline{\overline{A}\overline{B}\overline{C}})(\overline{\overline{A}BC})(\overline{A\overline{B}C})(\overline{AB\overline{C}})$$

$$L = (A + B + C)(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + C)$$

Exemple 2.3

On veut simplifier le circuit suivant:

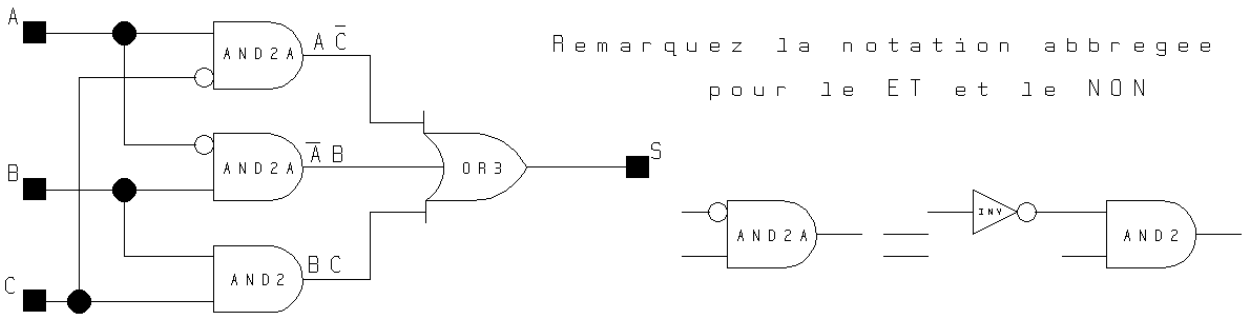


Figure 2.3: Schéma de l'exemple 2.3.

La transcription directe du schéma en équation donne:

$$S = A\overline{C} + \overline{A}B + BC$$

En reformulant sous la forme canonique on obtient:

$$S = A\overline{C}(B + \overline{B}) + \overline{A}B(C + \overline{C}) + BC(A + \overline{A})$$

$$S = AB\overline{C} + A\overline{B}\overline{C} + \overline{A}BC + \overline{A}B\overline{C} + ABC + \overline{A}BC$$

Le terme $\overline{A}BC$ apparaît deux fois, on peut en éliminer un:

$$S = AB\overline{C} + A\overline{B}\overline{C} + \overline{A}BC + \overline{A}B\overline{C} + ABC$$

On regroupe pour mettre AB et $\overline{A}B$ en facteur:

$$S = AB\overline{C} + ABC + \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\overline{C}$$

$$S = AB(\overline{C} + C) + \overline{A}B(C + \overline{C}) + A\overline{B}\overline{C}$$

$$S = AB + \overline{A}B + A\overline{B}\overline{C}$$

Mise en facteur de B:

$$S = B(A + \overline{A}) + A\overline{B}\overline{C}$$

$$S = B + A\overline{B}\overline{C}$$

Création d'un nouveau terme pour mettre $A\overline{C}$ en évidence:

$$S = B\,(1 + A\overline{C}) + A\overline{B}\overline{C}$$

$$S = B + AB\overline{C} + A\overline{B}\overline{C}$$

$$S = B + A\overline{C}\,(B + \overline{B})$$

$$\boxed{S = B + A\overline{C}}$$

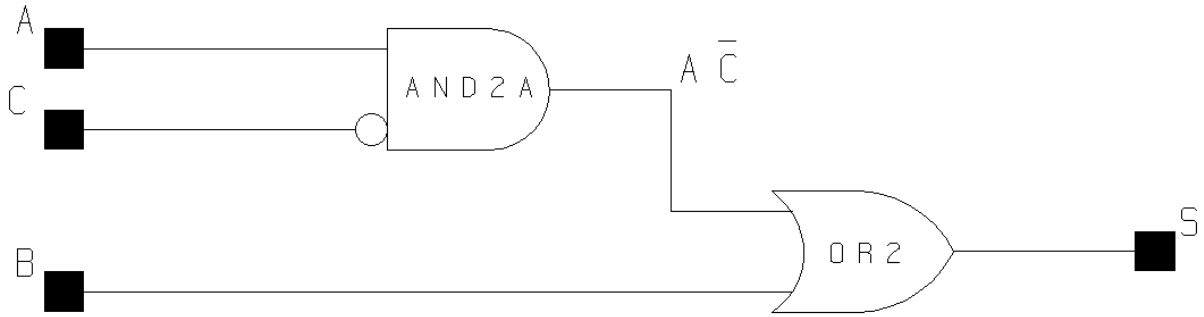


Figure 2.4: Schéma simplifié de l'exemple 2.3.

Exemple 2.4

Reprise de l'exemple 2.1:

$$L = ABC + A\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}\overline{B}C$$

$$L = A(BC + \overline{B}\overline{C}) + \overline{A}(B\overline{C} + \overline{B}C)$$

$$L = A(\overline{B \oplus C}) + \overline{A}(B \oplus C)$$

$$L = A \oplus B \oplus C$$

2.4 Synthèse et simplification

La synthèse des systèmes combinatoires permet de représenter une fonction logique sous une forme telle qu'on puisse la réaliser dans un système avec un nombre minimal de composants, pour la simplicité et la performance.

2.4.1 Karnaugh

La méthode de Karnaugh permet de visualiser une fonction et d'en tirer intuitivement une fonction simplifiée. L'élément de base de cette méthode est la table de Karnaugh qui représente, sous forme de tableau, toutes les combinaisons d'états possibles pour un nombre de variables donné. Chaque dimension du tableau pouvant représenter visuellement deux variables, on peut facilement simplifier des fonctions à quatre variables à l'aide d'un tableau à deux dimensions. Bien que plus difficile à visualiser, la simplification de fonctions à cinq ou six variables est possible à l'aide de tableau à trois dimensions.

Codes

Avant de poursuivre les explications de la table de Karnaugh, il faut introduire quelques concepts sur lesquels Karnaugh s'est basé pour développer ses tables. Le premier concept est l'utilisation d'une représentation particulière de l'information, il s'agit des codes. Les codes et systèmes représentatifs seront vus en détail aux chapitre 5. Lorsque plusieurs variables logiques sont exprimées ensemble, ce groupe peut être appelé nombre binaire. Par analogie aux nombres décimaux où chaque chiffre est un multiplicateur d'une puissance de dix, chaque chiffre qui le compose est le multiplicateur d'une puissance de deux. Le nombre 123 peut donc être exprimé de la façon suivante:

Nombre décimal	Nombre binaire
$123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$	$123 = 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
$123 = 100 + 20 + 3$	$123 = 64 + 32 + 8 + 4 + 2 + 1$

Cette notation, appelée code binaire naturel, permet de représenter 2^n nombres en utilisant *n* bits (variables binaires). Avec quatre bits, il est donc possible de représenter 16 nombres allants de 0 à 15. Le tableau suivant (tableau 2.1) donne les combinaisons possibles pour un mot binaire naturel de 4 bits, A_3, A_2, A_1 et A_0.

Nombre décimal	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Nombre décimal = $A_3 \bullet 2^3 + A_2 \bullet 2^2 + A_1 \bullet 2^1 + A_0 \bullet 2^0$

Tableau 2.1: Code binaire naturel.

Le code binaire naturel est utilisé pour énumérer toutes les combinaisons possibles d'entrées lors de la création de la table de vérité d'une fonction.

Le code de Gray, quant à lui, a été élaboré à partir des deux caractéristiques suivantes:

- La transition d'un mot au mot suivant implique qu'un, et seulement un, bit change d'état.
- Le code est cyclique.

Ces deux caractéristiques ne sont pas présentes dans le code binaire naturel. Par exemple, en passant du nombre 7 au nombre 8, tous les bits doivent changer d'état. De plus, le code binaire naturel n'est pas cyclique puisque le nombre suivant 15 est 16 et qu'il n'est pas possible de représenter ce nombre avec le code binaire naturel à 4 bits. Dans un compteur binaire naturel, le

nombre suivant 15 sera 0 avec une indication de débordement. La table suivante montre les codes de Gray à 1, 2 et 3 bits.

Gray à 1 bit	Gray à 2 bits	Gray à 3 bits
0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100

Tableau 2.2: Code de Gray à 1, 2 et 3 bits.

Théorème d'adjacence

Le deuxième concept nécessaire à la compréhension des tables de Karnaugh est l'adjacence logique. Deux mots binaires sont dits adjacents s'ils ne diffèrent que par la complémentarité d'une, et seulement une, variable. Si deux mots sont adjacents sont sommés, ils peuvent être fusionnés et la variable qui diffère est éliminée. Par exemple, les mots ABC et $AB\overline{C}$ sont adjacents puisqu'ils ne diffèrent que par la complémentarité de la variable C. Le théorème stipule donc que $ABC + AB\overline{C} = AB$. La preuve est simple:

$$ABC + AB\overline{C} = AB$$

$$AB(C + \overline{C}) = AB$$

$$AB = AB$$

La première caractéristique du code de Gray spécifie que deux mots consécutifs ne diffèrent que par l'état d'un bit, ils sont donc adjacents.

Tables de Karnaugh

La table de Karnaugh a été construite de façon à faire ressortir l'adjacence logique de façon visuelle. La figure suivante représente des tables de Karnaugh à deux, trois, quatre et cinq variables.

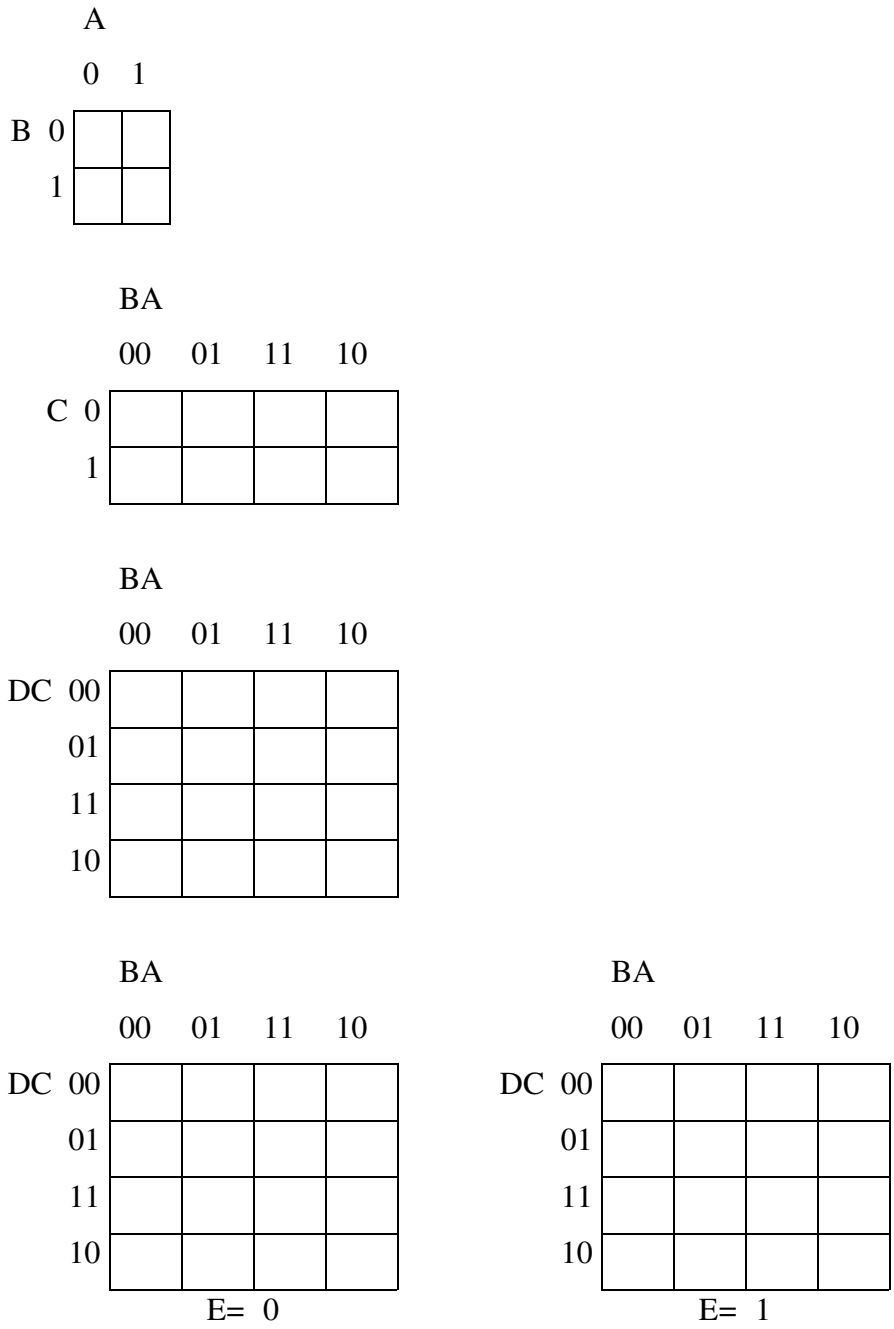


Figure 2.5: Tables de Karnaugh à 2, 3, 4 et 5 variables.

La méthode de Karnaugh consiste à indiquer dans la table les cases correspondantes aux états de variable d'entrées produisant une sortie vraie. Cela peut être déterminé à partir de l'équation de la fonction ou sa table de vérité. Il faut toutefois porter une attention particulière lors du transfert d'information de la table de vérité puisque celle-ci utilise un code binaire naturel, qui est sensiblement différent du code de Gray de la table de Karnaugh. Lorsque toute la fonction

est représentée dans la table, on procède à des regroupements de "1" qui se situent les uns à côté des autres. Puisque la table de Karnaugh utilise un code de Gray, ces groupements identifient des termes adjacents. La figure suivante identifie certains groupements typiques. Il est important de noter que les groupements sont toujours des rectangles (les carrés sont aussi des rectangles) contenant un nombre de "1" qui est une puissance de deux.

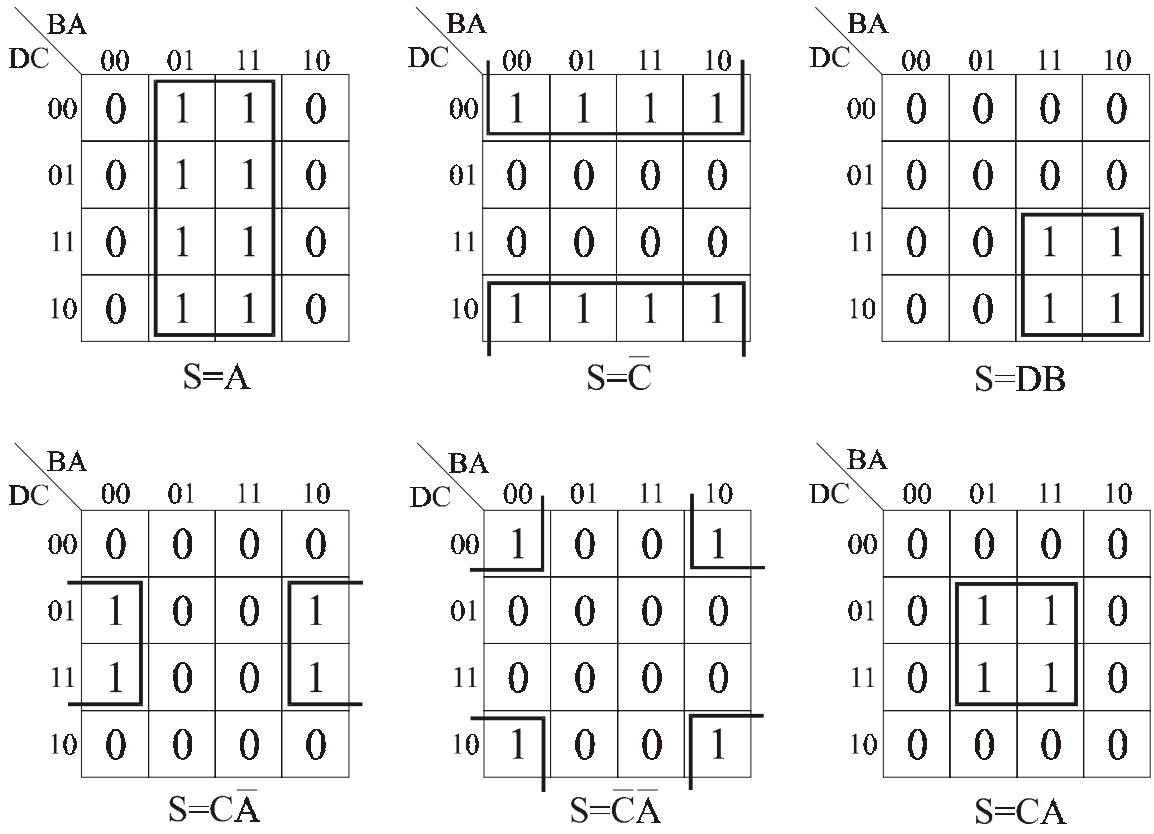


Figure 2.6: Formation d'impliquants dans les tables de Karnaugh.

Remarques:

- Lorsqu'un impliquant couvre 2^n fenêtres, 2^n termes sont fusionnés en un seul et *n* variables sont éliminées.
- Tous les impliquants sont dits "premiers".
- Si une fenêtre n'est couverte que par un seul impliquant, il est essentiel.
- Si un impliquant couvre des fenêtres déjà impliquées, il est redondant, mais il peut être utile pour éviter des erreurs momentanées (*glitchs*).
- La table est cyclique dans les deux sens.
- Si un système possède plusieurs sorties, il faut une table par sortie.

Exemple 2.5

On veut simplifier $S = A\overline{C} + \overline{A}B + BC$

Il faut tout d'abord remplir la table de Karnaugh à l'aide de l'équation de la fonction. Cette étape est très importante puisqu'il faut remplir toutes les cases qui correspondent à une combinaison d'entrées produisant une sortie vraie.

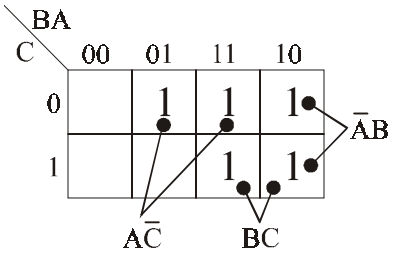


Figure 2.7: Remplissage de la table de Karnaugh

Une fois que la table est remplie, il faut procéder aux groupements. Il est très important d'utiliser tous les "1" de la table sans exceptions.

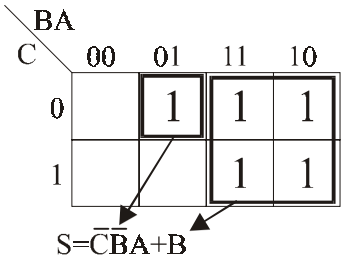


Figure 2.8: Formation des impliquants.

La forme obtenue n'est cependant pas la plus compacte. Pour obtenir la forme la plus compacte possible, il faut créer les groupements les plus grands possibles. Notez qu'il est possible d'utiliser les "1" aussi souvent que désiré. Les groupements précédents peuvent alors être exprimés ainsi. La forme présentée à la figure 2.9 est la plus compacte possible.

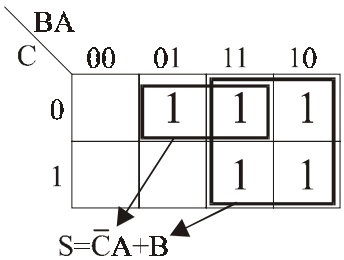


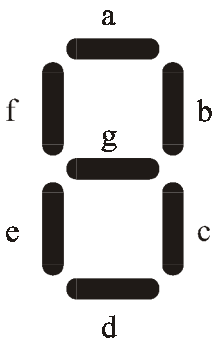
Figure 2.9: Formation optimale des impliquants.

Conditions facultatives

Sous certaines conditions, il se peut qu'un système soit conçu avec des combinaisons d'entrées qui ne se présentent jamais ou qui sont sans intérêts. On désigne les états de sorties correspondantes par des conditions facultatives (*don't care*) et on les considère comme des "1" ou des "0", selon ce qui est avantageux. On représente ces conditions facultatives dans les tables par le symbole "X" ou "∅"

Exemple 2.6

Les afficheurs à sept segments sont bien connus. Ils servent à afficher un chiffre allant de 0 à 9. Ce nombre est représenté sur 4 bits qui permettent de représenter les nombres de 0 à 15. Les combinaisons d'entrées de 10 à 15 sont des conditions facultatives puisqu'elles ne devraient jamais être présentes.



$$a = \sum (0,2,3,5,6,7,8,9) + \phi(10,11,12,13,14,15)$$

$$b = \sum (0,1,2,3,4,7,8,9) + \phi(10,11,12,13,14,15)$$

.

.

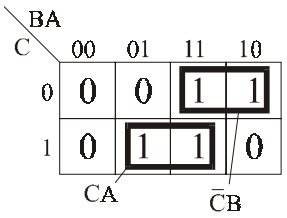
.

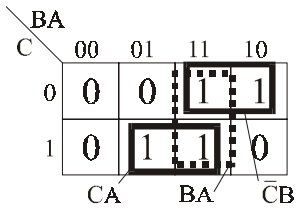
Impliquants redondants

Un impliquant est dit redondant si toutes les fenêtres qu'il couvre dans une table de Karnaugh sont déjà couvert par un autre impliquant. Ce terme peut être enlevé de l'équation sans changer la table de vérité. Sous certaines conditions, ce terme peut stabiliser le circuit en enlevant des erreurs momentanées (*glitchs*) de fonctionnement.

Exemple 2.7

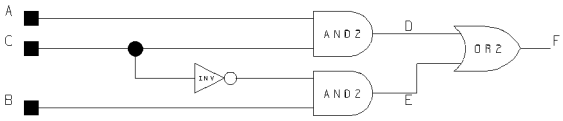
$$F = \overline{C}B\overline{A} + \overline{C}BA + CBA + C\overline{B}A$$

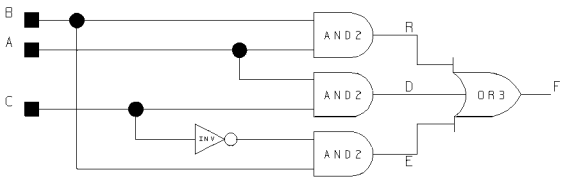


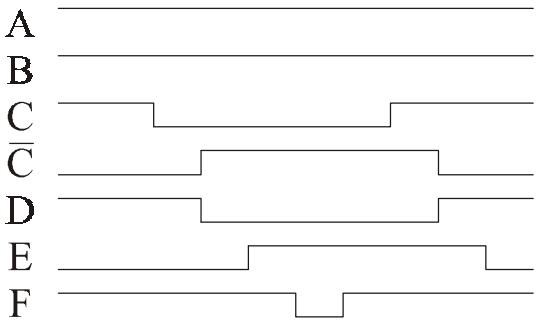


$$F = \overline{C}B + CA$$

$$F = \overline{C}B + CA + BA$$







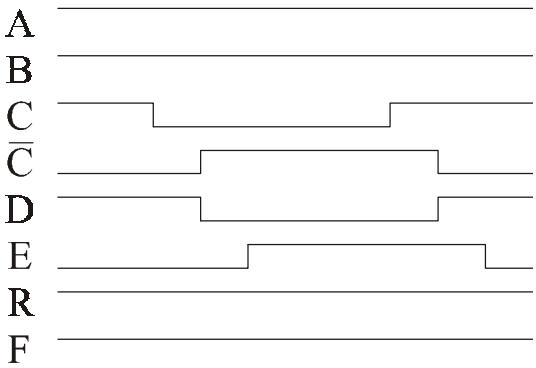


Figure 2.10: Effets d'un impliquant redondant.

Symétrie et pliage

Certaines configurations courantes des tables de Karnaugh permettent de simplifier les équations. La réduction d'équation en utilisant ces configurations, ou des techniques de pliage, est toutefois très intuitive et peut introduire des erreurs lorsqu'elles ne sont pas appliquées avec attention. Il est préférable de maîtriser les tables de Karnaugh avant d'utiliser ces techniques et configurations.

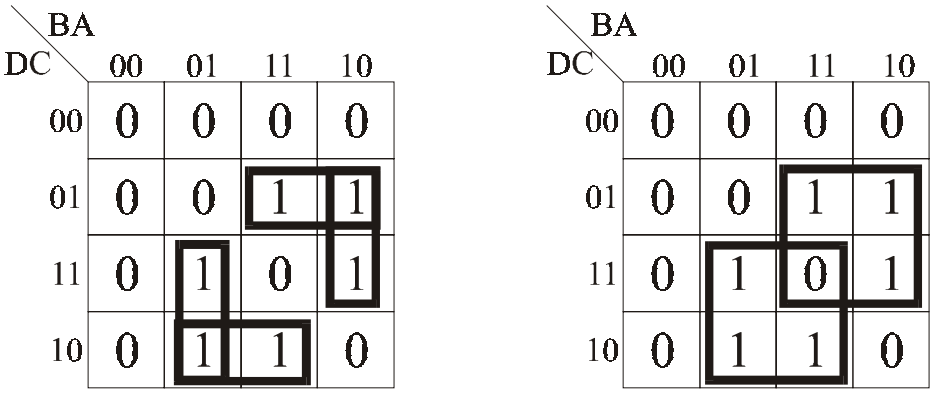


Figure 2.11: Symétrie de la table de Karnaugh

$F = D\overline{B}A + D\overline{C}A + \overline{D}CB + CB\overline{A}$ $F = DA \oplus BC$

$F = DA(\overline{B} + \overline{C}) + CB(\overline{D} + \overline{A})$

$F = DA\overline{BC} + CB\overline{DA}$

$F = DA \oplus BC$

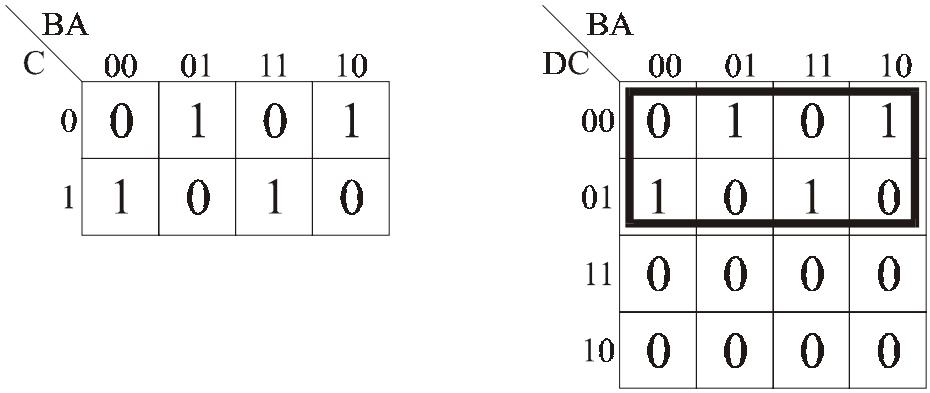


Figure 2.12: Symétrie de la table de Karnaugh.

$F = A \oplus B \oplus C$ $F = \overline{D}(A \oplus B \oplus C)$

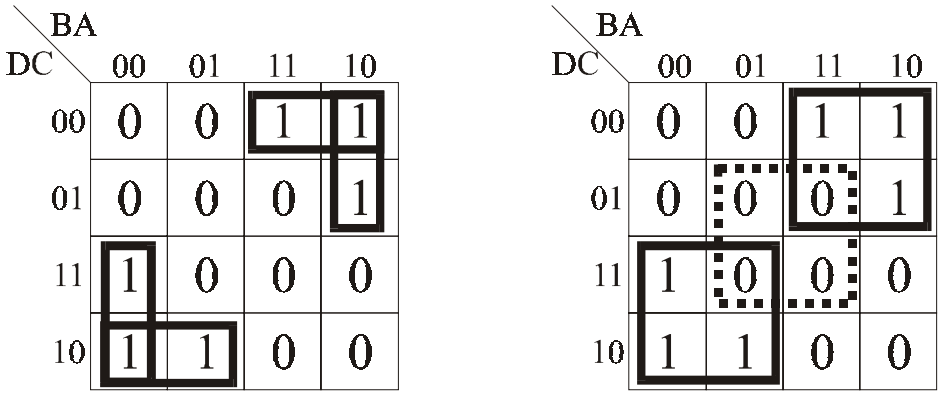


Figure 2.13: Symétrie de la table de Karnaugh.

$F = D\overline{B}\overline{A} + D\overline{C}\overline{B} + \overline{D}\overline{C}B + \overline{D}B\overline{A}$

$F = \overline{A}(D\overline{B} + \overline{D}B) + \overline{C}(D\overline{B} + \overline{D}B)$

$F = (\overline{A} + \overline{C})(B \oplus D)$

$F = \overline{A\overline{C}}(B \oplus D)$

$F = \overline{A\overline{C}}(\overline{B}D + B\overline{D})$

$F = \overline{A\overline{C}}(B \oplus D)$

2.4.2 Quine/McCluskey

Avec les méthodes précédentes, les simplifications ont été obtenues de façon purement intuitive; rien ne nous assure que la fonction obtenue est réellement la plus simple que l'on peut obtenir. De plus, en l'absence d'algorithme bien défini, ces méthodes ne peuvent être implantées efficacement dans un logiciel.

La méthode Quine/McCluskey, constituée d'une procédure bien définie. garantit une simplification maximale de la fonction obtenue, sous forme de somme de produits. I1 n'existe aucune autre fonction équivalente contenant moins de termes. La méthode de Quine/McCluskey utilise un algorithme pour faire ressortir l'adjacence entre les termes.

La procédure à suivre est la suivante:

1. Mettre la fonction sous forme canonique.
2. Transformer les termes en nombres binaires.
3. Grouper ces nombres selon leur poids (nombre de "l").
4. Placer les nombres par ordre croissant à l'intérieur de chaque groupe.
5. Comparer chaque terme d'un groupe avec chaque terme du groupe suivant: deux termes n'ayant qu'un bit qui ne correspond pas génèrent un nouveau terme où le bit de différence est remplacé par un "X"; les nouveaux termes engendrés forment une liste de nouveaux nombres binaires groupés par poids.
6. Refaire l'étape 5 à partir de la nouvelle liste obtenue jusqu'à ce qu'aucune autre nouvelle liste ne soit générée.
7. Identifier les impliquants, c'est à dire les éléments qui n'ont pas été utilisés pour générer un élément de la nouvelle liste.
8. Identifier les impliquants essentiels, c'est à dire ceux dont la représentation est unique pour certaines solutions.
9. Vérifier si l'ensemble des impliquants essentiels représente toutes les solutions. Si oui, la solution minimale est trouvée. Si non, on doit ajouter un ou plusieurs autres impliquants afin de représenter toutes les solutions. Il n'existe aucune façon précise pour choisir les autres impliquants.

Exemple 2.8

On veut simplifier la fonction suivante:

$$S = \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}\,\overline{C}D + AB\overline{C}D$$

Étape 1: La fonction est déjà sous forme canonique.

Étape 2: Transformation en nombres binaires.

S=0010+0100+0101+0110+0111+1001+1101

Étapes 3 et 4: Classification.

Poids 1	0010
	0100
Poids 2	0101
	0110
	1001
Poids 3	0111
	1101

Étapes 5 et 6: Comparaisons.

1	0010	a	0X10 (1-4)	01XX (b-f et c-d)
2	0100	b	010X (2-3)	
	------	c	01X0 (2-4)	
3	0101		-------	
4	0110	d	01X1 (3-6)	
5	1001	e	X101 (3-7)	
	------	f	011X (4-6)	
6	0111	g	1X01 (5-7)	
7	1101			

Étape 7: Identification des impliquants

Les termes qui n'ont jamais engendrés de nouveaux termes sont marqués d'un "*"

0010	0X10 *	01XX *
0100	010X	
------	01X0	
0101	-------	
0110	01X1	
1001	X101 *	
------	011X	
0111	1X01 *	
1101		

Étape 8 et 9: Identification des impliquants essentiels

	0010	0100	0101	0110	0111	1001	1101
0X10	[✓]			(✓)			
X101			✓				✓
1X01						[✓]	(✓)
01XX		[✓]	(✓)	(✓)	[✓]		

✓ indique que l'impliquant couvre le terme.

[✓] indique que l'impliquant est essentiel à ce terme.

(✓) indique un terme couvert par un impliquant essentiel.

Les impliquants essentiels sont: 0X10, 1X01 et 01XX. Puisqu'ils sont suffisants pour représenter toutes les solutions, la fonction simplifiée est:

$$S = \overline{A}C\overline{D} + A\overline{C}D + \overline{A}B$$

Exemple 2.9

On veut simplifier la fonction suivante:

$$S = \overline{A}BD + ABCD + A\overline{B}CD + ABC\overline{D} + A\overline{B}C\overline{D}$$

Étape 1: Mettre la fonction sous forme canonique.

$$S = \overline{A}BD(\overline{C} + C) + ABCD + A\overline{B}CD + ABC\overline{D} + A\overline{B}C\overline{D}$$

$$S = \overline{A}B\overline{C}D + \overline{A}BCD + ABCD + A\overline{B}CD + ABC\overline{D} + A\overline{B}C\overline{D}$$

Étape 2: Transformation en nombres binaires.

S=0101+0111+1111+1011+1110+1010

Étapes 3 et 4: Classification.

Poids 2	0101
	1010
Poids 3	0111
	1011
	1110
Poids 4	1111

Étapes 5, 6 et 7: Comparaisons et identification des impliquants.

Il est aussi possible d'indiquer un "✓" lorsque les termes sont utilisés. Les impliquants seront alors les termes sans "✓".

0101 ✓	01X1	1X1X
1010 ✓	101X ✓	
------	1X10 ✓	
0111 ✓	------	
1011 ✓	X111	
1110 ✓	1X11 ✓	
------	111X ✓	
1111 ✓		

Étape 8 et 9: Identification des impliquants essentiels

	0101	1010	0111	1011	1110	1111
01X1	[✓]		(✓)			
X111			✓			✓
1X1X		[✓]		[✓]	[✓]	(✓)

Les impliquants essentiels sont: 01X1 et 1X1X. Puisqu'ils sont suffisants pour représenter toutes les solutions, la fonction simplifiée est:

$$S = \overline{A}BD + AC$$

Exemple 2.10

On veut simplifier la fonction suivante. Notez les conditions facultatives.

$$F = \sum (2,4,7,12,10,15) + \phi \sum (6,9,11,14)$$

Étape 1: La fonction est sous forme canonique.

Étape 2: Transformation en nombres binaires.

F=0010+0100+0111+1100+1010+1111+∅(0110+1001+1011+1110)

Étapes 3 et 4: Classification.

À partir de cette étape, les conditions facultatives sont traitées comme les autres termes.

Poids 1	0010
	0100
Poids 2	0110
	1001
	1010
	1100
Poids 3	0111
	1011
	1110
Poids 4	1111

Étapes 5, 6 et 7: Comparaisons et identification des impliquants.

0010 ✓	0X10 ✓	XX10
0100 ✓	X010 ✓	X1X0
------	01X0 ✓	--------
0110 ✓	X100 ✓	X11X
1001 ✓	-------	1X1X
1010 ✓	011X ✓	
1100 ✓	X110 ✓	
------	101X ✓	
0111 ✓	1X10 ✓	
1011 ✓	10X1	
1110 ✓	11X0 ✓	
------	-------	
1111 ✓	X111 ✓	
	1X11 ✓	
	111X ✓	

Étape 8 et 9: Identification des impliquants essentiels

Notez que seulement les termes essentiels sont inscrits. L'utilité des conditions facultatives est terminée puisqu'elle se limite à permettre une plus grande simplification.

	0010	0100	0111	1100	1010	1111
10X1						
XX10	[✓]				(✓)	
X1X0		[✓]		[✓]		
X11X			[✓]			(✓)
1X1X					✓	✓

Les impliquants essentiels sont: XX10, X1X0 et X11X. Puisqu'ils sont suffisants pour représenter toutes les solutions, la fonction simplifiée est:

$$S = B\overline{A} + C\overline{A} + CB$$

Le choix d'une méthode de simplification dépend de la fonction à simplifier. Généralement, on utilisera:

- La méthode algébrique pour des fonctions à deux variables ou des fonctions de plus de deux variables mais comportant peu de termes sous forme canonique.
- La méthode de Karnaugh pour des fonctions de trois, quatre, cinq ou six variables.
- La méthode de Quine/McCluskey pour des fonctions à cinq variables ou.

2.4.3 Table de Karnaugh à variable inscrite

La table de Karnaugh à variable inscrite (KVI) est née de la difficulté de réduire les fonctions de plus de quatre variables en utilisant la table de Karnaugh normale. Cette méthode est cependant plus complexe et beaucoup d'expérience avec la table de Karnaugh est un atout majeur. Il est préférable d'avoir complété la série d'exercices sur Karnaugh avant de débuter KVI.

L'idée derrière KVI est d'ajouter un niveau d'abstraction sur la table de vérité de la fonction à réduire. Le résultat n'est pas seulement donné en terme de "0" et de "1" mais en fonction d'une, ou plusieurs, variable d'entrée. La table de vérité suivante est transformée afin d'inscrire la variable C.

A	B	C	F	
0	0	0	0	0
0	0	1	0	
0	1	0	1	$\overline{C}$
0	1	1	0	
1	0	0	0	C
1	0	1	1	
1	1	0	1	1
1	1	1	1	

A	B	F
0	0	0
0	1	$\overline{C}$
1	0	C
1	1	1

La KVI est ensuite construite de la même façon que la table de Karnaugh. La seule différence est que les fenêtres peuvent maintenant contenir des variables en plus des "0" et "1". Les groupements se font maintenant entre termes identiques. Notez qu'il est possible de grouper un "C" avec un "1" car le "1" contient un "C" ($1 = \overline{C} + C$). Tout comme la Karnaugh ordinaire où il était très important d'utiliser tous les "1", dans la KVI, il faut absolument utiliser toutes les combinaisons de la variable inscrite. Si un "C" a été groupé avec un "1", il faut utiliser le "$\overline{C}$" qui fait partie du "1", soit comme "$\overline{C}$", soit en utilisant le "1".

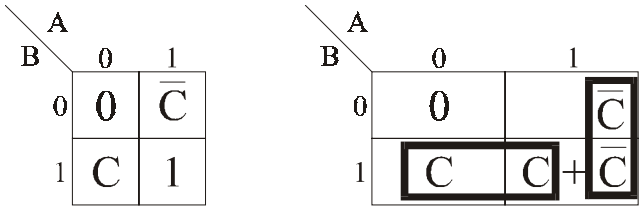


Figure 2.14: Table de Karnaugh à variable inscrite.

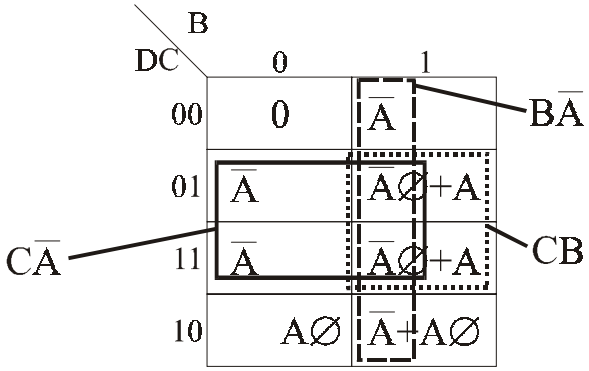
$$F = CB + \overline{C}A$$

Exemple 2.11

Simplifier la fonction suivante:

$$F(D,C,B,A) = \sum(2,4,7,12,10,15) + \phi\sum(6,9,11,14)$$

D	C	B	A	F	
0	0	0	0	0	0
0	0	0	1	0	
0	0	1	0	1	$\overline{A}$
0	0	1	1	0	
0	1	0	0	1	$\overline{A}$
0	1	0	1	0	
0	1	1	0	∅	$A+\overline{A}\phi$
0	1	1	1	1	
1	0	0	0	0	$A\phi$
1	0	0	1	∅	
1	0	1	0	1	$\overline{A}+A\phi$
1	0	1	1	∅	
1	1	0	0	1	$\overline{A}$
1	1	0	1	0	
1	1	1	0	∅	$A+\overline{A}\phi$
1	1	1	1	1	



$$F = C\overline{A} + B\overline{A} + CB$$

2.5 Éléments de base II

Les composants utilisés jusqu'à maintenant (ET, OU, NON-ET, XOR, ...) faisaient partie de la catégorie SSI (*Small Scale Integration*). Principalement à cause de leur simplicité, ce sont les premiers circuits intégrés à avoir été réalisés avec succès au tout début de l'ère électronique moderne. Les progrès techniques réalisés en conception de circuits intégrés ont permis de concevoir des circuits un peu plus complexes permettant de réaliser des fonctions plus générales. Ces circuits d'intégration moyenne (MSI – *Medium Scale Integration*) sont présentés dans cette section.

Le multiplexeur (MUX)

Le multiplexeur est un système combinatoire ayant pour fonction de sélectionner une parmi 2^n entrées et de la transmettre à la sortie. La sélection est faite à l'aide de *n* lignes d'adresse. La notation usuelle du MUX est: MUX 2^n à 1. Par exemple, un MUX 8 à 1 aura 3 lignes d'adresse. La figure 2.15 présente la forme générale d'un MUX.

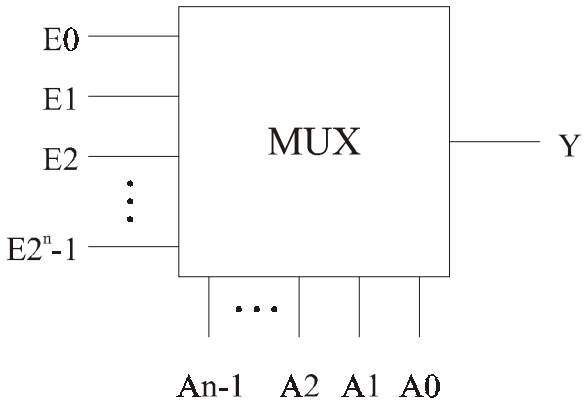
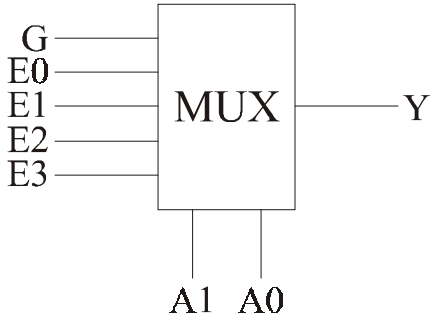


Figure 2.15: Représentation générale d'un multiplexeur.

La figure 2.16 montre le composant 74153 qui contient deux MUX 4 à 1. Le signal supplémentaire *G* (*Strobe*) est un signal d'activation du composant. Si *G* est inactif, la sortie du MUX sera obligatoirement inactive.



G	A1	A0	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	E0
1	0	1	E1
1	1	0	E2
1	1	1	E3

Figure 2.16: Caractéristiques du 74153.

Le démultiplexeur (DEMUX)

Le démultiplexeur est un système combinatoire ayant pour fonction de transmettre une entrée vers une des 2^n sorties. La sélection est faite à l'aide de *n* lignes d'adresse et les sorties sont mutuellement exclusives. La notation usuelle du DEMUX est: DEMUX 1 à 2^n. Par exemple, un DEMUX 1 à 8 aura 3 lignes d'adresse. La figure 2.17 présente la forme générale d'un DEMUX.

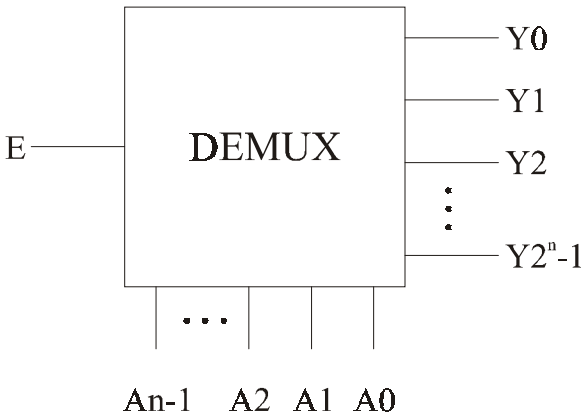
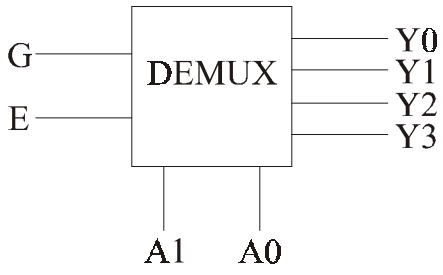


Figure 2.17: Représentation générale d'un démultiplexeur.

La figure 2.18 montre un DEMUX 1 à 4. Le signal supplémentaire *G* (*Strobe*) est un signal d'activation du composant. Si *G* est inactif, toutes les sorties du DEMUX seront obligatoirement inactives.



G	A1	A0	Y
0	0	0	Yn inactifs
0	0	1	Yn inactifs
0	1	0	Yn inactifs
0	1	1	Yn inactifs
1	0	0	Y0=E
1	0	1	Y1=E
1	1	0	Y2=E
1	1	1	Y3=E

Figure 2.18: Caractéristiques d'un démultiplexeur.

L'encodeur

L'encodeur est un système combinatoire ayant pour fonction de retourner l'index d'activation d'une parmi 2^n entrées. L'index d'activation est donné sur *n* lignes d'adresse. Lorsque plusieurs entrée sont activées, l'encodeur accorde la priorité à l'entrée dont l'index est supérieur. La notation usuelle de l'encodeur est: encodeur 2^n à n. Par exemple, un encodeur 8 à 3 aura 8 entrées et 3 lignes d'adresse en sortie. La figure 2.19 présente la forme générale d'un encodeur.

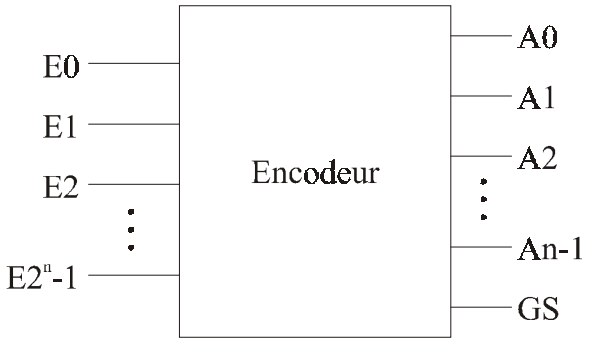
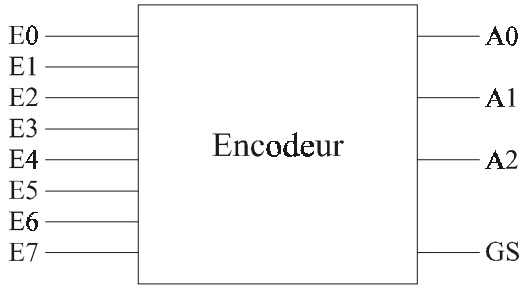


Figure 2.19: Représentation générale d'un encodeur.

La figure 2.20 montre une simplification du composant 74148 qui contient un encodeur 8 à 3. Le signal supplémentaire *GS* (*Got Something*) est un signal qui indique qu'une des entrées est active dans le but de faire la différence entre l'entrée 0 active et lorsqu'aucune entrée n'est activée.



E7	E6	E5	E4	E3	E2	E1	E0	GS	A2	A1	A0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	X	1	0	0	1
0	0	0	0	0	1	X	X	1	0	1	0
0	0	0	0	1	X	X	X	1	0	1	1
0	0	0	1	X	X	X	X	1	1	0	0
0	0	1	X	X	X	X	X	1	1	0	1
0	1	X	X	X	X	X	X	1	1	1	0
1	X	X	X	X	X	X	X	1	1	1	1

Figure 2.20: Caractéristiques du 74148.

Le décodeur

Le décodeur est un système combinatoire ayant pour fonction d'activer une des 2^n sorties. La sélection est faite à l'aide de *n* lignes d'adresse et les sorties sont mutuellement exclusives. La notation usuelle du décodeur est: décodeur 1 parmi 2^n. Le décodeur se comporte exactement comme un DEMUX avec son entrée toujours à 1. Par exemple, un décodeur 1 parmi 8 aura 3 lignes d'adresse. La figure 2.21 présente la forme générale d'un décodeur.

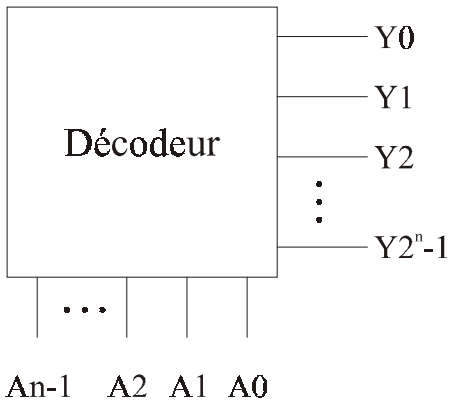
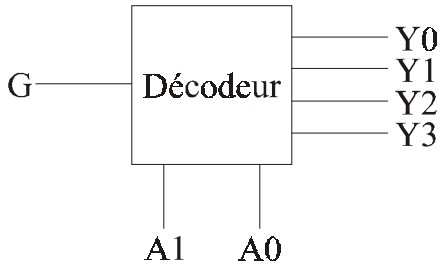


Figure 2.21: Représentation générale d'un décodeur.

La figure 2.22 montre un décodeur 1 parmi 4. Le signal supplémentaire *G* (*Strobe*) est un signal d'activation du composant. Si *G* est inactif, toutes les sorties du DEMUX seront obligatoirement inactives.



G	A1	A0	Y
0	0	0	Yn inactifs
0	0	1	Yn inactifs
0	1	0	Yn inactifs
0	1	1	Yn inactifs
1	0	0	Y0=1
1	0	1	Y1=1
1	1	0	Y2=1
1	1	1	Y3=1

Figure 2.22: Caractéristiques d'un décodeur 1 parmi 4.

Les mémoires

Il existe une très grande quantité de catégories de mémoires. Les plus connues sont les ROM (*Read Only Memory*) et les RAM (*Random Access Memory*). Une mémoire est un réseau combinatoire ayant pour fonction d'emmagasiner de l'information. La mémoire est une matrice de cellules capable d'emmagasiner une information unitaire (bit) qui a une adresse spécifique unique. Les cellules ont souvent une largeur *m* supérieure à 1. L'accès aux cellules se fait à l'aide de *n* bits d'adresse pour une mémoire d'une capacité de $m \bullet 2^n$ bits. La différence entre les deux catégories principales est que les ROM ne peuvent être que lues tandis que les RAM peuvent être utilisées en écriture tout aussi bien qu'en lecture. La figure 2.23 montre le schéma général d'une ROM et d'une RAM. Le signal *OE* (*Output Enable*) permet l'activer les sorties. Si *OE* n'est pas actif, les sorties sont électriquement inactives, ce qui signifie qu'il n'y a ni un "0" ni un "1" de présent. Le signal *WE* (*Write Enable*) permet d'écrire des données dans une RAM. À ce moment,

le signal *OE* doit être inactif pour permettre aux données d'arriver à la mémoire sans conflits. Même si les deux types de mémoires ne diffèrent que par le signal *WE*, le fonctionnement interne est différent.

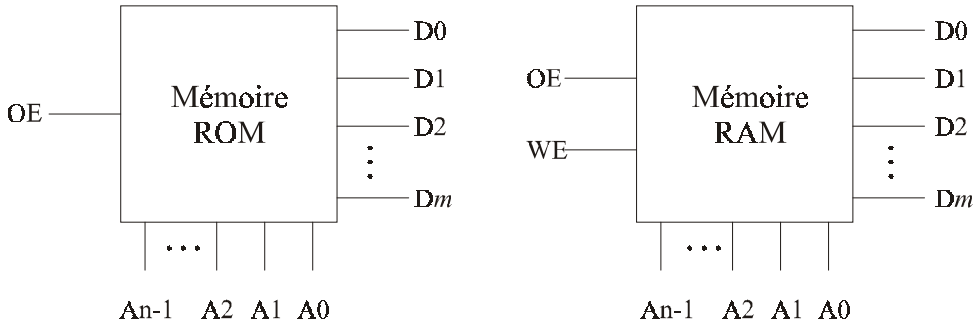


Figure 2.23: Représentation générale des mémoires ROM et RAM.

Les registres

Contrairement aux composants MSI vus jusqu'ici, les registres sont ne sont pas des systèmes combinatoires, ce sont des composants permettant d'emmagasiner de l'information. Pour une même combinaison d'entrées, ce type de composant peut produire des sorties différentes, selon l'information emmagasinée. Il existe deux types de registres: les bascules (*flip-flop*) est les bistables (*latchs*) qui diffèrent par leur signal d'activation. Les bistables ont un signal d'activation qui est basé sur un niveau logique tandis que le signal d'activation des bascules est sensible à une transition de niveau. Lorsque le signal d'activation d'une bistable est actif, tout ce qui est à l'entrée est transféré directement à la sortie. Lorsque le signal d'activation devient inactif, la sortie conserve sa valeur jusqu'à ce que le signal redevienne actif. Avec une bascule, le comportement est légèrement différent. L'information à l'entrée est copiée à la sortie sur une transition spécifique du signal d'activation, habituellement la transition du "0" vers le "1". La figure 2.24 présente la forme générale d'une bistable et d'une bascule ainsi qu'un diagramme de temps pour chacun.

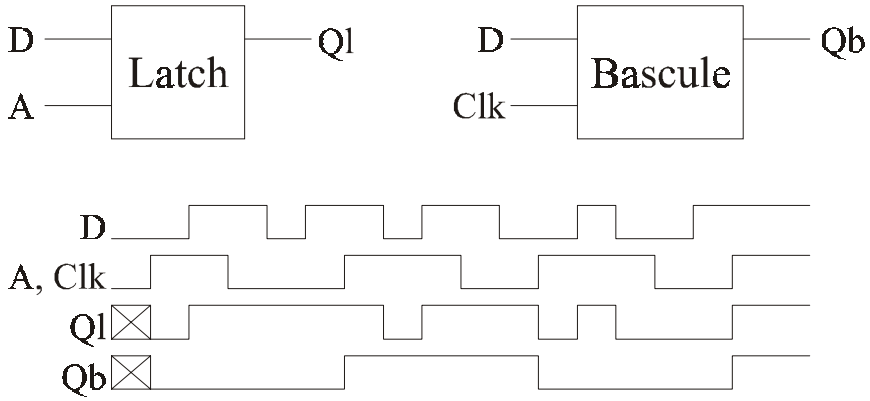


Figure 2.24: Différence de comportement entre une bistable et une bascule.

Les compteurs

Comme dans le cas des registres, les compteurs sont ne sont pas des systèmes combinatoires, ce sont des composants permettant d'emmagasiner et de traiter de l'information. Pour une même combinaison d'entrées, ce type de composant peut produire des sorties différentes, selon l'information emmagasinée. Le compteur dispose de deux signaux importants: le signal de compte est le résultat qui est représenté sur *n* bits. Sur une transition spécifique du signal de compte, habituellement la transition du "0" vers le "1", un incrément est de 1 est ajouté au résultat. Même si ma majorité des compteurs suit un ordre qui est binaire naturel, il est possible de trouver des compteurs avec d'autres séquences. Lorsque le compteur est rendu à la limite représentable de sa séquence, certains compteurs ont un signal *TCO* (*Terminal Count Output*) ou *RCO* (*Ripple Carry Output*) qui signifie qu'un débordement aura lieu au prochain incrément. La figure 2.25 présente la forme générale d'un compteur 4 bits ainsi qu'un diagramme de temps.

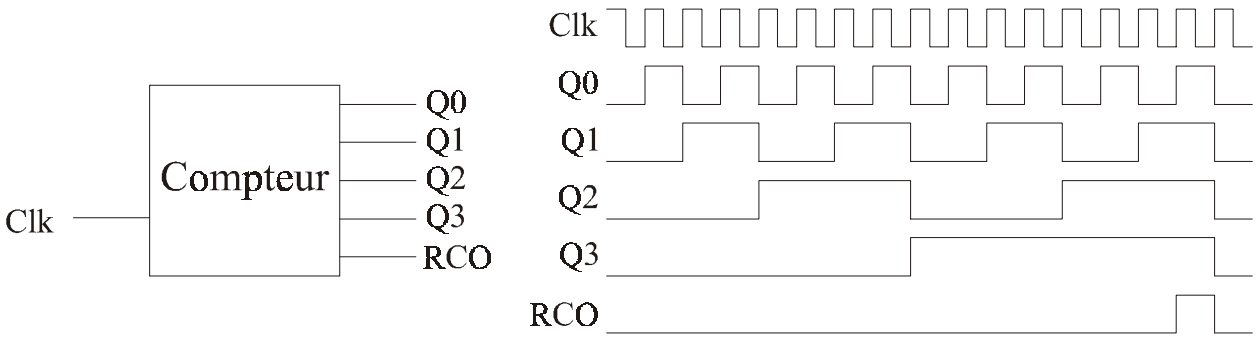


Figure 2.25: Comportement d'un compteur 4 bits.

2.6 Modules et familles électroniques

Cette section explique les différences entre les éléments logiques et les modules électroniques qui exécutent les fonctions. Un survol des familles logiques existantes ainsi que certaines de leurs spécifications électriques complétera ce chapitre.

2.6.1 Modules et éléments logiques

Jusqu'à maintenant, les fonctions logiques ont toujours été réalisées sans se préoccuper des modules logiques. Les modules logiques sont les puces (*chips*) dans lesquelles les circuits électroniques sont réalisés. Le *TTL databook* est une excellente source d'informations sur tous les aspects des modules électroniques. On y trouve des informations sur l'utilisations des broches, sur les délais de réaction des modules, sur les fonctions réalisées est d'autres informations sur les caractéristiques électriques. Un module comporte souvent plusieurs éléments logiques. Un 7408 par exemple contient 4 portes "ET". De plus, les modules logiques nécessitent une alimentation (Vcc) et une masse (GND). Lors de la réalisation d'un circuit à partir d'un schéma, il est parfois préférable d'apporter quelques modifications dans le but de réduire le nombre de modules électroniques. Le chapitre 3, portant sur la logique mixte, fournit des méthodes avancées pour réduire le nombre de modules. Cependant, les règles d'associativité et de distributivité doivent aussi être appliquées afin de réduire le nombre de composants. L'exemple suivant montre comment ces règles peuvent être utiles.

Exemple 2.12

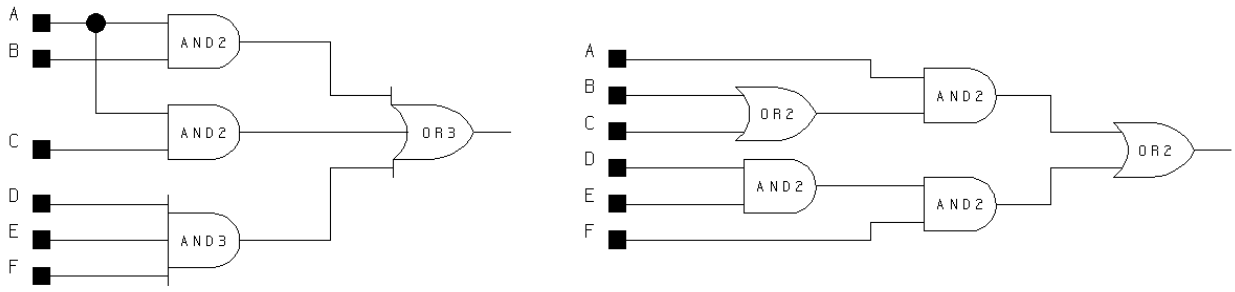
Réaliser la fonction suivante en minimisant les modules logiques.

Z=AB+AC+DEF

La fonction peut être réalisée directement comme dans le premier cas de la figure 2.26. Certaines transformations peuvent cependant être appliquées pour réduire le nombre et la diversité des modules logiques nécessaires à la réalisation de la fonction.

La mise en facteur du A permet de se débarrasser du "OU" à trois entrées qui, de toute façon, n'existe pas en logique TTL (les familles logiques sont expliquées dans la prochaine section). En utilisant les règles d'associativité, le terme DEF devient (DE)F, ce qui élimine le

"ET" à trois entrées. L'équation devient alors Z=A(B+C)+(DE)F. Il est important de noter que même si le nombre d'éléments logiques augmente, le nombre de modules logiques sera moins important.



Cas 1: Z=AB+AC+DEF Cas 2: Z=A(B+C)+(DE)F

Figure 2.26: Réduction du nombre de modules logiques.

ET 2 entrées: ½ 74LS08
ET 3 entrées: ½ 74LS11
OU 3 entrées: ½ 74HC4075
Total: 3 modules logiques

ET 2 entrées: ¾ 74LS08
OU 2 entrées: ½ 74LS32
Total: 2 modules logiques

2.6.2 Familles logiques

Cette section donne un aperçu des caractéristiques et des différences des trois principales familles logiques: TTL (*Transistor Transistor Logic*) standard, LS (Low-power Schottky) et HC (High-speed CMOS). Une famille logique est une ensemble de modules électroniques accomplissant les fonctions logiques de base. La fonction logique ne dépend pas de la famille logique ou du grade du module, de sorte qu'un 54LS08 et un 74HC08 accomplissent exactement la même fonction logique. Les modules sont parfois donnés avec la notation '08, qui met en évidence que la famille et le grade ne sont pas importants à ce point.La notation usuelle des modules logiques est toujours de la forme suivante:

GGG*FFF*MMM où:

GGG est le grade	*FFF* est la famille logique	MMM est le module ou fonction logique
• 74 pour les grades commerciaux. • 54 pour les grades militaires.	• vide: TTL standards. • LS: Low-power Schottky. • HC: High-speed CMOS.	Ce suffixe indique la fonction réalisée. La liste complète des fonctions se trouve dans le *TTL Databook*.

Famille TTL standard:

La famille TTL standard est première famille logique complète. Ce n'est pas la première technologie logique puisqu'auparavant des technologies comme DL (*Diode Logic*), RTL (*Resistor Transistor Logic*) et DTL (*Diode Transistor Logic*) étaient utilisées. Cette technologie est basée sur des transistors bipolaires et, même si l'état logique est déterminé par la tension, ces modules nécessitent des courants, tant au niveau statique qu'aux changements d'états.

Famille LS:

La famille LS utilise l'architecture TTL et est basée sur la famille TTL standard. Les différences portent sur deux points importants: la vitesse et la consommation énergétique. En augmentant les résistances internes, la consommation énergétique est réduite puisque les courants sont moins importants. Cependant, la vitesse est aussi réduite. En ajoutant une diode entre la base et le collecteur des transistors, un délai important est éliminé dans les changements d'états logiques et le composant devient plus rapide. L'effet combiné de ces deux modifications donne un module logique qui est plus rapide et consomme moins que ceux de la famille TTL standard, c'est donc un excellent compromis.

Famille HC:

La famille HC différente des deux familles précédentes. Elle n'est pas basée sur l'architecture TTL puisqu'elle utilise des transistors MOSFET (*Metal Oxyde Semiconductor Field Effect Transistor*). Ces transistors n'utilisent des courants que lors des transitions d'état logique. La consommation énergétique statique est donc pratiquement nulle. Au niveau de la vitesse, les modules HC sont typiquement plus rapides que les modules TTL à la même alimentation. Si la tension d'alimentation des modules HC augmente, la vitesse et la consommation augmenteront elles aussi. Parallèlement, si ta tension d'alimentation est réduite, la vitesse et la consommation seront réduites. De nos jours, la forte majorité des circuits logiques toutes catégories confondues sont fabriqués en CMOS (*Complementary* MOS) qui est très similaire à la technologie HC.

2.6.3 Spécifications électriques

Parmi les spécifications les plus courantes, on retrouve les alimentations, les courants d'états et les niveaux logiques en tensions. La figure 2.27 symbolise un élément logique et les valeurs importantes.

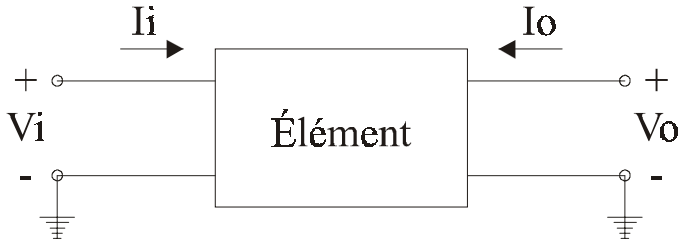


Figure 2.27: Élément logique est ses spécifications électriques.

Certaines caractéristiques sont contraintes et d'autres sont des propriétés. Par exemple, Vi et Ii sont des contraintes puisque, pour fonctionner correctement, ces valeurs doivent être respectées. Io et Vo représentent ce que l'élément a la capacité de fournir. Toutes ces caractéristiques sont fournies avec détail dans le *TTL Databook*. L'exemple suivant est tiré du 7400.

Familles TTL				
Description	Notation	Propagation (ns/porte)	Consommation (mW/porte)	Fanout
Standard	74*xxx*	10	10	10
Low-Power	74LSxxx	33	1	20
High Speed	74H*xxx*	6	22	10
Schottky	74S*xxx*	3	19	10
Low-Power Schottky	74LS*xxx*	9.5	2	20
Advanced Schottky	74AS*xxx*	1.7	8	40
Advanced LP Schottky	74ALS*xxx*	4	1.2	20
Fast	74F*xxx*	3	4	33
Familles CMOS				
Série A	40*xxx*A	25	10	50
Série B	40*xxx*B	60	10	50
Série UB	40*xxx*UB	30	7	50
C	74C*xxx*	50	5	50
HCT	74HCT*xxx*	8	10	50
FACT	74AC*xxx*	5	2	50
HC	74HC*xxx*	8	1	50
AHCT	74AHCT*xxx*	7	.12	50
HCTLS	74AHCTLS*xxx*	10	.12	50
Autres Familles				
RTL		12	16	5
DTL		30	12	5
MDTL		30	10	5
ECL 10K	101*xxx*, 105*xxx*	2	25	83
ECL 10K	102*xxx*, 106*xxx*	1.5	25	54
ECL 10K	108*xxx*	1-2.5	2.3	63
ECL 10KH	10 KH*xxx*	1	25	83
ECL 100K	1001*xxx*	.75	40	

Tableau 2.3: Principales caractéristiques des familles logiques.

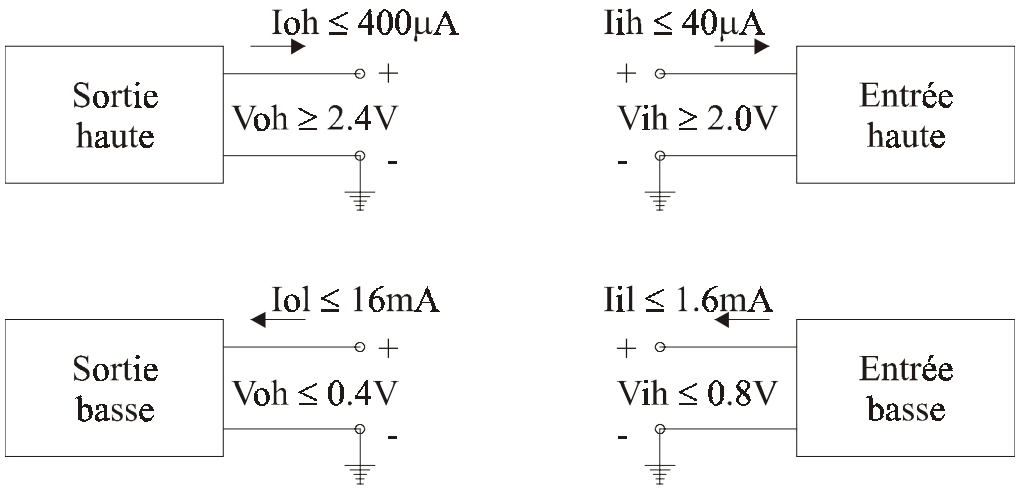


Figure 2.28: Spécifications électriques du 7400.

Puisque ces valeurs doivent être interprétées correctement, voici leurs explications:

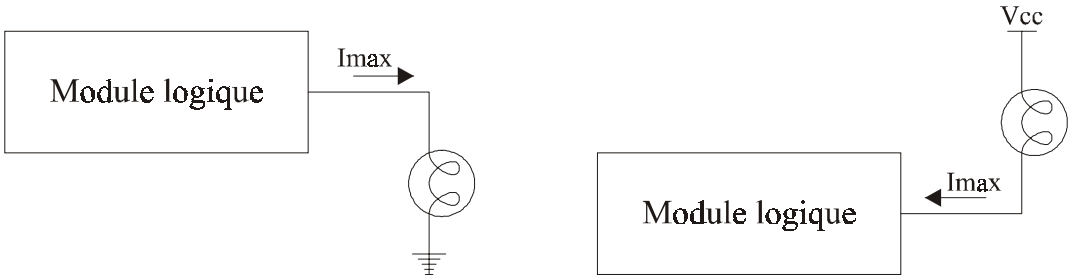
- Sortie haute: l'élément peut produire un courant maximal de 400µA. Si ce courant limite est respecté, la tension de sortie sera toujours supérieure à 2.4V.
- Sortie basse: l'élément peut recevoir un courant maximal de 16mA. Si ce courant limite est respecté, la tension de sortie sera toujours inférieure à 0.4V.
- Entrée haute: l'élément peut demander un courant maximal de 40µA. La tension d'entrée doit toujours être supérieure à 2.0V.
- Entrée basse: l'élément peut demander un courant maximal de 1.6mA. La tension d'entrée doit toujours être inférieure à 0.8V.

Pour une sortie haute branchée sur des entrées, on voit que toutes les règles sont respectées tant que le nombre d'entrées ne dépasse pas 10. À ce moment, il y a 10 entrées pouvant nécessiter un maximum de 40µA chacune. La sortie peut donner un maximum de 400µA tout en conservant sa sortie supérieure à 2.4V pour les entrées, qui elles, nécessitent plus de 2.0V. Le nombre d'entrées que le 7400 est capable d'alimenter, nommé "*fanout*", est donc de 10.

Une autre caractéristique importante à noter est que le courant Iol est beaucoup plus grand que Ioh ce qui signifie que les modules logiques ont la capacité de recevoir beaucoup plus de courant qu'ils ne peuvent en donner. Cela implique que si le composant doit alimenter un composant électronique de moyenne puissance, il sera beaucoup plus facile de le faire en

utilisant le module logique comme interrupteur de masse. L'exemple suivant montre comment on peut utiliser un module logique pour commander une ampoule de basse puissance.

Exemple 2.13



Montage incorrect Montage correct

Figure 2.29: Branchement de charge sur des modules logiques.

De la façon incorrecte, le courant disponible pour l'ampoule n'est que de 400µA. En procédant de la bonne façon, l'ampoule dispose de 16mA pour allumer, elle sera donc environ 40 fois plus brillante.

CHAPITRE 3

La logique mixte

Il existe plusieurs façons de représenter l'information. Jusqu'à maintenant, la logique utilisée a toujours été positive. Cependant, ce n'est pas la seule façon de représenter l'information. Ce chapitre montre les limitations inhérentes à la logique positive et la logique négative. Il introduit la logique mixte, qui réussi à éliminer les limitations des deux logiques en fournissant des outils permettant de les utiliser simultanément sur le même schéma.

3.1 Logique positive

Dans la convention de la logique positive, le niveau logique "1", actif, est associé à une tension haute. Le niveau logique "0", inactif, quant à lui, est associé à une tension basse. Cette convention est utilisée depuis le début du présent document car elle est relativement simple et intuitive.

3.2 Logique négative

Dans la convention de la logique négative, le niveau logique "1", actif, est associé à une tension basse. Le niveau logique "0", inactif, quant à lui, est associé à une tension haute. La nécessité d'utiliser des signaux en logique négative est apparue avec l'utilisation de signaux d'activation de composants (*Strobe*). Pour des raisons de simplicité et de rapidité, il est avantageux que ces signaux soient actifs lorsque la tension est basse. Une autre utilité vient des DEL (Diode Électro Luminescente) et ampoules de faible puissance. Dans le chapitre précédent, il a été vu qu'il est préférable d'activer ce genre de composants avec une sortie qui correspond à la masse. Puisque le signal qui alimente l'ampoule est actif lorsqu'il y a 0V, il est en logique négative.

3.3 Logique mixte

La logique mixte est une méthode permettant d'utiliser de façon conviviale les deux logiques, positive et négative, sur le même schéma. Elle permet de séparer l'état logique et la

convention utilisée. Cette méthode respecte toutes les règles de la logique vues jusqu'à maintenant et introduit de nouvelles règles permettant de simplifier les schémas de façon plus efficace. La logique mixte n'introduit pas de nouveaux éléments logiques mais intègre une notation légèrement différente. Le tableau suivant résume les conventions des deux logiques.

État	Logique positive	Logique négative
Actif (1)	H (5V)	L (0V)
Inactif (0)	L (0V)	H (5V)

Tableau 3.1: Convention des logiques positive et négative.

3.4 Impact des conventions

Les conventions ont un impact très important sur les fonctions réalisées. Physiquement, la fonction réalisée est toujours identique, peu importe la convention. Remarquez que la table de vérité des modules du *TTL databook* est toujours donnée en terme de H et L, qui sont toujours associés aux niveaux de tensions 5V et 0V respectivement. Le 7408 accomplit toujours le même travail avec les tensions, peu importe que les signaux soient en logique positive ou négative. Ce n'est que l'interprétation qui change. Prenons par exemple la table de vérité du 7408 et appliquons les conventions des deux logiques pour étudier l'impact des conventions.

Table de vérité			Logique positive			Logique négative		
A	B	Y	A	B	Y	A*	B*	Y*
L	L	L	0	0	0	1	1	1
L	H	L	0	1	0	1	0	1
H	L	L	1	0	0	0	1	1
H	H	H	1	1	1	0	0	0

Tableau 3.2: Impact des conventions sur un 7408.

Il est très important de remarquer que la fonction logique accomplie par le 7408 est différente si le concepteur décide de l'utiliser avec des signaux ayant une logique positive ou négative. Dans le premier cas, la fonction logique réalisée est un "ET" et dans le deuxième cas, c'est un "OU" logique. Le 7408 peut donc être appelé "ET positif" ou "OU négatif". D'ailleurs, il

porte le nom "*Quadruple 2-input positive-and gates*" (Quadruple porte "ET positif" à 2 entrées) dans le *TTL databook*.

3.5 Symboles et notation

La logique mixte n'utilise que quatre symboles. Ces symboles sont très similaires à ceux utilisés jusqu'ici. Chaque symbole a cependant un certain nombre de variations pour tenir compte de la convention à la sortie et à chacune des entrées. Les quatre symboles de base sont:

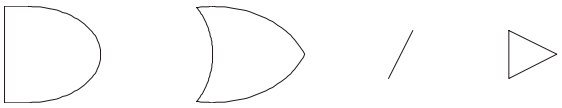


Figure 3.1: Symboles de la logique mixte.

Les deux premiers sont les symboles du "ET" et du "OU" respectivement. Le troisième symbole (/ ou *slash*) est utilisé pour signifier un changement de convention, et le dernier symbole, l'inverseur sans la petite boule (◦), est un tampon (*buffer*). À ces symboles s'ajoute une notation qui indique la convention du signal. La notation est la petite boule qui vient s'insérer entre le symbole et le fil du signal en logique négative. Un fil représentant une variable en logique négative aura toujours une petite boule à tous ses nœuds. Une notation spécifique s'applique aussi aux variables pour identifier si elles sont en logique positive ou négative. Les variables en logique positive demeurent inchangées tandis que celles en logique négative doivent être suivies d'un "*". Certains concepteurs utilisent aussi la notation A(H) et A(L) pour une variable A en logique positive et négative respectivement.

3.5.1 La barre oblique

Le seul nouveau symbole est la barre oblique ("/") qui signifie un changement de convention. Ce symbole se place sur un fil pour séparer deux parties d'un circuit. La convention n'est qu'une interprétation des tensions physique, alors la barre oblique ne correspond à aucun composant physique puisque les tensions demeurent inchangées. Ce n'est que leur interprétation en logique positive ou négative qui engendre une différence. L'exemple suivant illustre divers cas possibles de l'utilisation de la barre oblique.

Figure 3.2: Utilisation du symbole de changement de convention.

Comme la barre oblique indique un changement de convention, il est tout à fait normal et essentiel que chaque coté de la barre soit dans une logique différente. Dans le premier cas, si la variable A est présente sur le fil, de l'autre coté du changement, l'inverse de A sera disponible en logique négative.

Variable	Logique positive	Tension	Logique négative	Résultat
A	Actif (1)	5V	Inactif (0)	$\overline{A}^*$
	Inactif (0)	0V	Actif (1)	

Tableau 3.3: Effets du changement de convention.

Dans le deuxième cas, si A est disponible en logique négative, de l'autre coté du changement, l'inverse de A sera disponible en logique positive.

Variable	Logique négative	Tension	Logique positive	Résultat
A^*	Actif (1)	0V	Inactif (0)	$\overline{A}$
	Inactif (0)	5V	Actif (1)	

Tableau 3.4: Effets du changement de convention

Ces deux cas illustrent toutes les possibilités de l'utilisation du "/ " puisque ces exemples peuvent être lus dans les deux sens.

L'utilisation de la petite boule du coté en logique négative de la barre oblique n'est pas obligatoire. Il est recommandé de l'utiliser au début pour se familiariser à son utilisation. Les concepteurs qui sont à l'aise avec la logique mixte ont tendance à utiliser la barre oblique sans utiliser la boule.

3.5.2 Le OOPS

En logique mixte, le symbole connu jusqu'à maintenant comme un inverseur change de signification. L'inverseur ne change pas l'état de la variable mais seulement la convention utilisée. Pour ce faire, la tension du signal est inversée. L'état du signal (actif ou inactif) restera toujours identique. Il n'y a que le "/" qui peut changer un signal d'état. Puisque l'inverseur n'inverse pas l'état, il est préférable d'utiliser un autre terme pour le désigner. À partir de ce point, le terme *OUPS* (ou OOPS) sera utilisé (certains concepteurs utilisent aussi le terme *inverseur de tension*). L'exemple suivant couvre les cas de l'utilisation du OUPS.

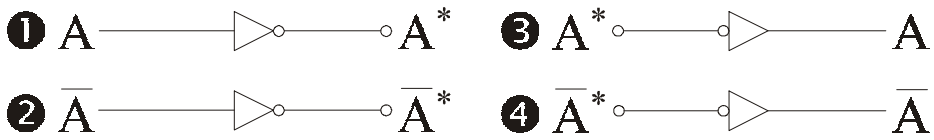


Figure 3.3: Utilisation du OOPS.

	État	Tension	OOPS	Tension	État	
A	Actif	5V	↓	0V	Actif	A^*
$\overline{A}$	Inactif	0V	↑	5V	Inactif	$\overline{A}^*$

Tableau 3.5: Effets du OOPS.

L'explication des cas 3 et 4 peut être trouvée en lisant le tableau de droite à gauche.

Exemple 3.1

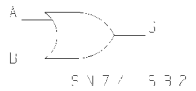
Comment générer $\overline{A}$ à partir de A?

Le seul symbole qui permet une inversion d'état est le *slash*. Cependant, il inverse également la convention. Heureusement, le OUPS inverse la convention sans changer l'état. Les deux possibilités suivantes permettent de générer $\overline{A}$ à partir de A.

Figure 3.4: Schéma de génération de l'inverse de A.

3.5.3 Les équivalences de portes

Les tables de vérités de la figure 3.6 montrent qu'une porte ET en logique positive et une porte OU en logique négative accomplissent la même fonction logique. Une porte OU en logique positive accomplira la même fonction qu'une porte ET en logique négative. La règle de base pour les portes équivalentes d'un ET est de changer le ET pour un OU, d'enlever les petites boules où il y en a et d'ajouter des boules où il n'y en avait pas. De façon similaire, la porte équivalente d'un OU est de changer le OU pour un ET, d'enlever les petites boules où il y en a et d'ajouter des boules où il n'y en avait pas. Dans le cas du OOPS, le symbole reste le même mais l'opération d'inversion des petites boules est aussi appliquée.



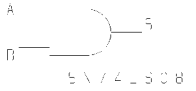
H	H	H
H	L	H
L	H	H
L	L	L

Logique Positive

1	1	1
1	0	1
0	1	1
0	0	0

Logique négative

0	0	0
0	1	0
1	0	0
1	1	1



H	H	H
H	L	L
L	H	L
L	L	L

Logique positive

1	1	1
1	0	0
0	1	0
0	0	0

Logique négative

0	0	0
0	1	1
1	0	1
1	1	1

Tableau 3.6 : Équivalence des portes en logique positive et négative.

Le tableau 3.7 résume les portes équivalentes.

3.5.4 Avantages de la logique mixte

Il existe au moins trois avantages à l'utilisation de la logique mixte par rapport à l'utilisation de l'une ou l'autre des deux logiques. Le premier avantage est la clarté. La fonction logique accomplie par un circuit respectant la logique mixte est beaucoup plus facile à trouver à partir du schéma.

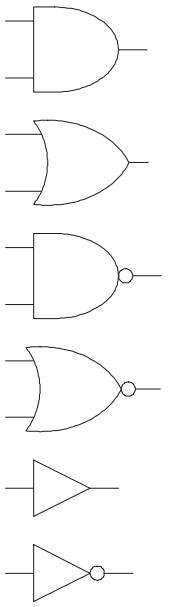
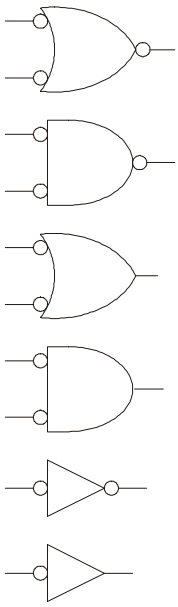
Porte existante	Porte équivalente
	

Tableau 3.7: Équivalence des portes de base.

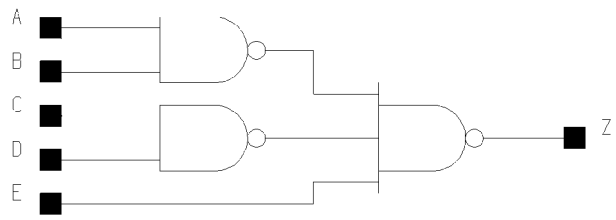
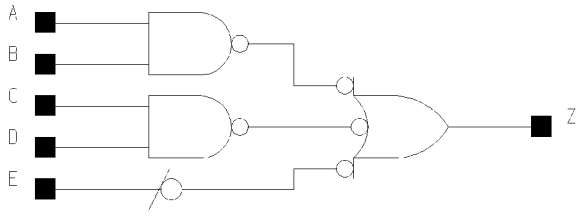
Logique Positive	Logique Mixte
	
$Z = \overline{\overline{AB}\,\overline{CD}\,E}$	$Z = AB + CD + \overline{E}$

Tableau 3.8: Clarté de la logique mixte.

Les deux circuits du tableau 3.8 exécutent la même fonction logique et sont réalisés avec les mêmes composants physiques. Le schéma utilisant la notation de la logique mixte est cependant beaucoup plus facile à convertir en équation.

Le deuxième avantage est la facilité avec laquelle les circuits peuvent être optimisés pour réduire le nombre de composants. En utilisant les représentations équivalentes pour les

composants physiques, il est souvent possible d'en réduire le nombre. Cette facilité s'applique aussi aux restrictions d'utilisation de composants auxquelles doivent se plier les concepteurs.

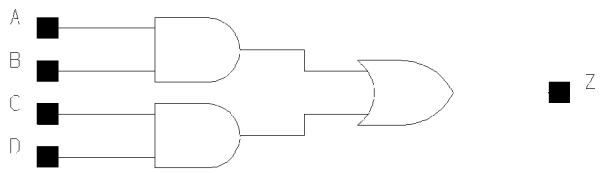
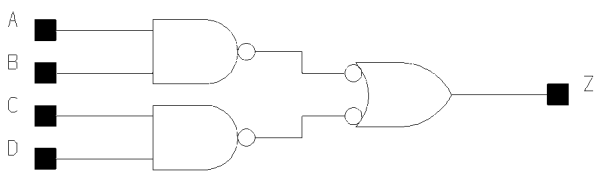
Logique Positive	Logique Mixte
	
2 composants: 1 ET et 1 OU	1 composant: 1 NON-ET

Tableau 3.9: Optimisation de circuits avec la logique mixte.

Le troisième avantage de l'utilisation de la logique mixte vient des recommandations de l'industrie. La majeure partie des compagnies œuvrant dans le domaine de la conception de circuits numériques utilisent la logique mixte à grande échelle. Il est alors très avantageux en tant que concepteur d'apprendre à utiliser la logique mixte le plus tôt possible.

L'utilisation des règles de la logique mixte demande un peu de pratique et d'expérience avant de pouvoir en apprécier pleinement des avantages. Si, à première vue, son utilisation ne semble qu'apporter une complexité supplémentaire et inutile, une fois ses concepts bien acquis, il devient évident que la logique mixte est un outil très puissant et essentiel aux designs de grande envergure.

3.5.4 Analyse de schéma en logique mixte.

Cette section décrit les étapes nécessaires pour trouver les équations de sortie d'un schéma réalisé en logique mixte. Il n'y a en fait que deux étapes à suivre:

1. Écrire l'équation en fonction des symboles ET et OU présents sur le schéma. Ne pas se soucier de la présence de barre oblique ("/"), de boules ou de la logique des variables.
2. Ajouter une barre de complémentarité sur les termes des fils ayant une barre oblique.

Exemple 3.2:

Trouver l'équation logique réalisée par le circuit suivant:

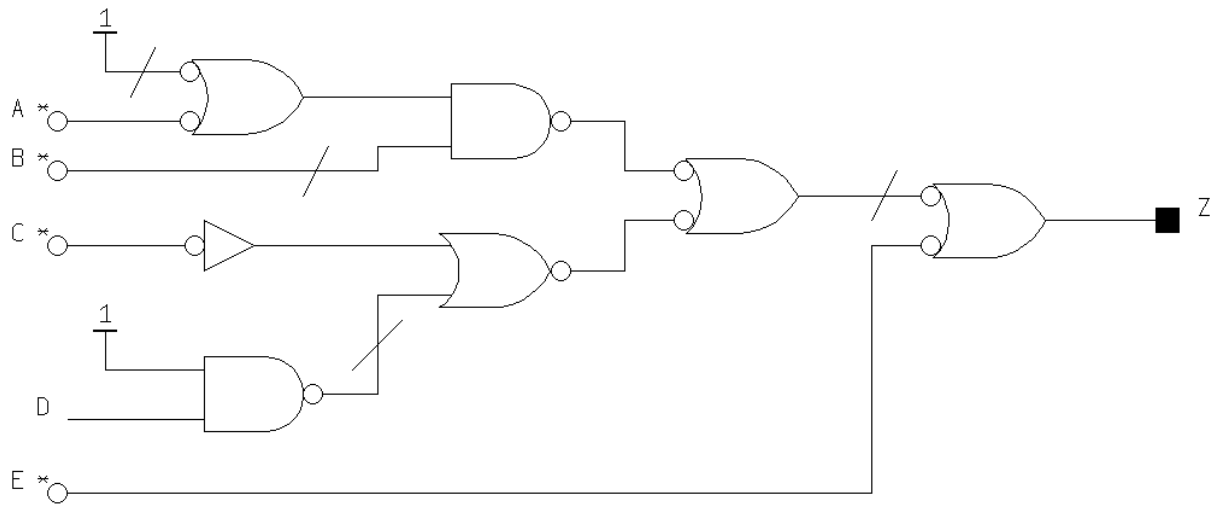


Figure 3.5: Schéma de l'exemple 3.2.

Étape 1:

$$Z = (A + 1)B + (C + (D \cdot 1)) + E$$

Étape 2:

$$Z = \overline{(A + \bar{1})\bar{B} + (C + \overline{(D \cdot 1)})} + E$$

$$Z = \overline{A\bar{B} + C + \bar{\bar{D}}} + E$$

3.5.5 Synthèse de schéma en logique mixte.

La synthèse de schéma en logique mixte suit des étapes simples et directes, comme pour l'analyse. Il y a très peu d'étapes où des décisions doivent être prises. Les étapes sont:

1. Dessiner le schéma en utilisant des ET et des OU, tel qu'ils sont présents dans l'équation et sans se soucier de la logique des variables d'entrée ou de sortie.
2. Transformer le circuit afin de n'utiliser que les portes disponibles.
3. Ajouter les barres obliques pour complémenter les variables.

4. Ajouter les cercles et les OOPS pour respecter la logique mixte. Cette opération s'effectue en deux parties. Tout d'abord, les boules sont ajoutées aux entrées/sorties en fonction de ce qui est disponible et demandé. Ensuite, les boules et les OOPS sont ajoutées en commençant par la sortie et en remontant jusqu'aux entrées.

Exemple 3.3:

Dessiner le schéma de l'équation suivante en respectant la logique mixte:

$Z = A\overline{B} + \overline{C + D}$ A, B*, C et D sont disponibles. Z* est désiré.

Étape 1:

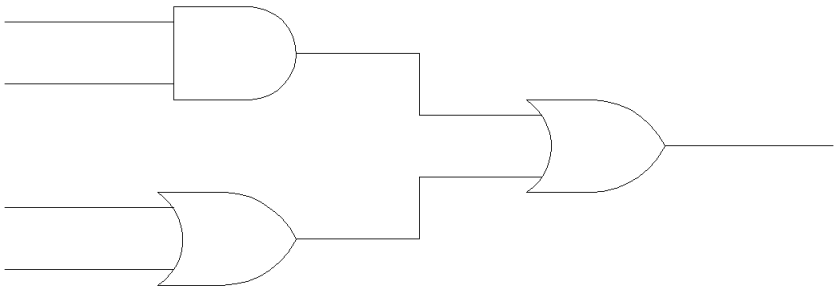


Figure 3.6: Première étape de la synthèse en logique mixte.

Étape 2:

Il n'y a aucune restriction quant aux composants disponibles.

Étape 3:

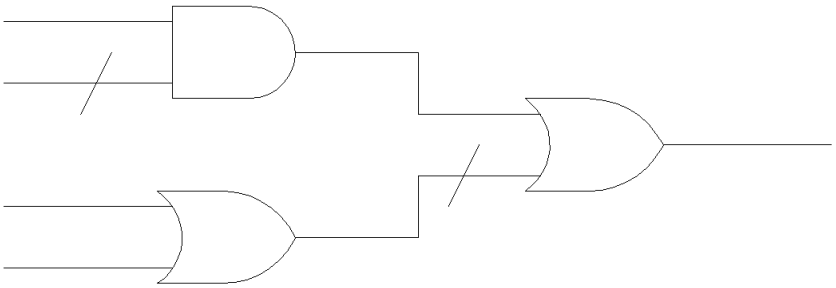


Figure 3.7: Troisième étape de la synthèse en logique mixte.

Étape 4:

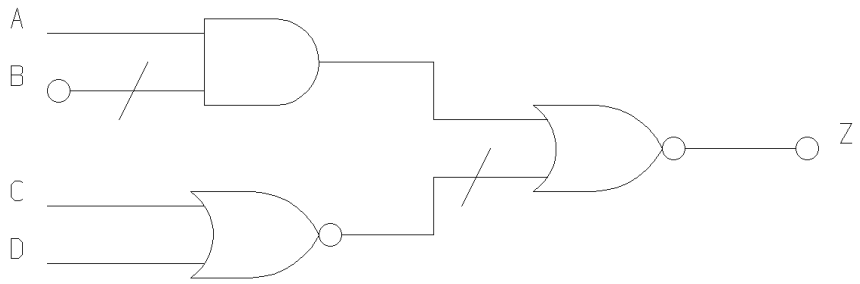


Figure 3.8: Schéma final de l'exemple 3.3.

Exemple 3.4:

Reprendre l'exemple précédent mais en n'utilisant que des 7400 (NON-ET Positif).

Étape 1:

Cette étape est identique à l'étape 1 de l'exemple précédent.

Étape 2 et 3:

Les portes sont remplacées par la même fonction dans sa forme disponible avec les portes équivalentes. Les ET doivent rester des ET, il en va de même pour les OU. Les barres obliques sont ensuite ajoutées.

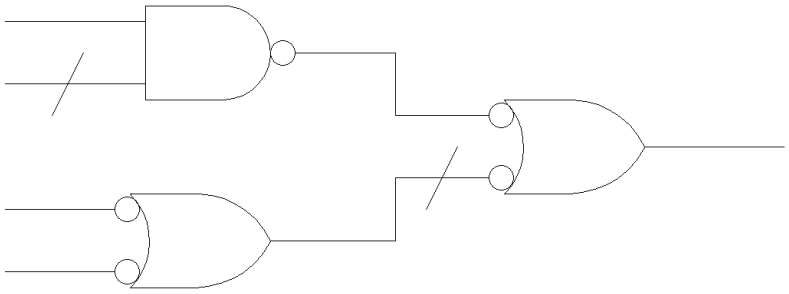


Figure 3.9: Troisième étape de la synthèse en logique mixte.

Étape 4:

Dans cette étape, il faut ajouter des OOPS car il y a conflit de conventions sur les fils.

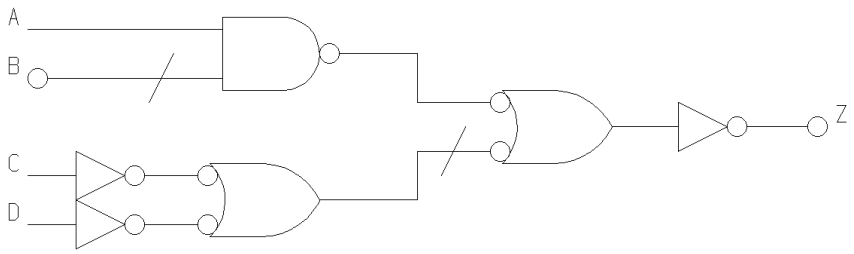


Figure 3.10: Schéma final de l'exemple 3.4

Cette étape n'est cependant pas complète sans spécifier comment les OOPS ont été réalisés. Il est impossible d'utiliser des 7404 car ils ne sont pas disponibles. Une façon possible en utilisant un 7400 est de brancher les deux entrées ensemble. Il est aussi possible de le réaliser en branchant une des deux entrées à 5 Volts.

CHAPITRE 4

Les réseaux itératifs

4.1 Introduction

Il est évident que la conception d'un système logique se complique rapidement au fur et à mesure que le nombre de variables croît. Même des algorithmes programmables, comme la méthode de Quine et McCluskey, deviennent lourds d'utilisation lorsque, par exemple, on a 10 variables d'entrée entraînant une possibilité de 1024 termes.

Dans le cas particulier où une fonction logique se répète sur des sous-ensembles des variables d'entrée, il est possible de synthétiser un sous-réseau effectuant cette fonction logique et de structurer le réseau pour qu'elle soit répétée sur tous les sous-ensembles constituant ces variables d'entrée. Ces réseaux sont dits itératifs.

Définition:

Les réseaux itératifs constituent une classe de réseaux qui effectuent des fonctions logiques de façon répétitive sur un grand nombre de variables.

Itération temporelle:

Lorsque les fonctions répétitives sont réparties sur des intervalles de temps successifs, on a une itération temporelle.

Exemple: Un comparateur binaire sériel.

Itération spatiale:

Lorsque les fonctions répétitives sont effectuées parallèlement par plusieurs sous-réseaux identiques, on a une itération spatiale.

Exemple: Un comparateur binaire parallèle.

Dans les réseaux à itération temporelle, les variables d'entrée sont mémorisées dans des registres et acheminées vers un seul sous-réseau qui confectionne les variables de sortie et les accumule dans une autre mémoire au les dirige vers une destination prédéfinie. Le sous-réseau est de nature combinatoire, mais l'ensemble de la configuration est à caractère séquentiel.

Dans les réseaux à itérations spatiales, les variables d'entrée sont transmises en parallèle à une cascade de sous-réseaux identiques interconnectés entre eux, et la fonction logique répétitive est effectuée par chaque sous-réseau sur chacun des sous-ensembles des variables. Ce sont donc des réseaux combinatoires requérant autant de sous-réseaux qu'il existe de sous-ensembles au niveau des variables d'entrée. Ces réseaux sont plus rapides que les précédents et les techniques d'intégrations les rendent peu coûteux. C'est donc sur ce type de réseau que la méthodologie de conception est présentée.

4.2 Forme générale des réseaux itératifs

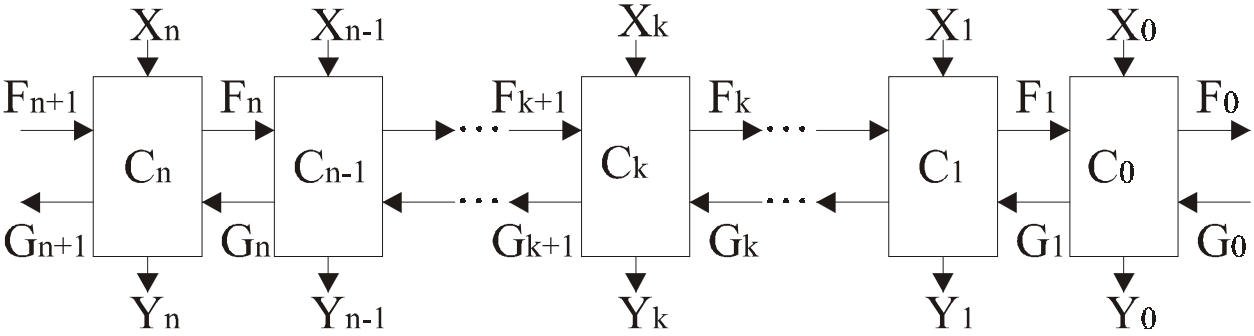


Figure 4.1: Forme générale d'un réseau itératif spatial

Un réseau itératif spatial prend la forme générale montrée à la figure 4.1. Un sous-réseau typique Ck reçoit un sous-ensemble de l'entrée X désignée par Xk constitué d'une ou plusieurs variables. Les sous-réseaux adjacents s'échangent des informations entre eux par l'intermédiaire de variables internes désignées par Fk, Gk, Fk+1,» Gk+1,... Ces variables peuvent être multiples ou même inexistantes. Chaque sous-réseau produit de façon générale une sortie Yk selon le choix du concepteur ou les spécifications de la fonction logique à réaliser. Cette sortie peut constituer la sortie principale du réseau, être transformée ou inexistante.

Par exemple, si le réseau effectue une sommation binaire, on retrouve la somme en Y, la fonction G assure la transmission de la retenue de droite à gauche et la fonction F est inexistante.

Un réseau qui réalise la comparaison entre deux mots binaires de gauche à droite ne requiert qu'un maximum de trois sorties selon le résultat spécifié. La sortie Y et la fonction G ne sont pas nécessaires selon l'algorithme développé.

La forme générale du réseau représente par conséquent le cas général où toutes les possibilités sont illustrées, mais, en pratique les spécifications de la fonction et les décisions du concepteur font de chaque design un cas particulier.

4.3 Méthodologie de design

ÉTAPE 1: Schéma bloc

On déduit de ce qui précède que le concepteur doit d'abord dessiner un schéma bloc du réseau itératif selon les spécifications de la fonction logique à réaliser. Cette étape est importante parce que, dans la plupart des cas, il faut prendre des décisions, poser des hypothèses ou définir des conventions. Cet aspect des réseaux itératifs fournit à leur étude des éléments de formation intéressants relatifs au développement du jugement dans les choix et de la créativité.

Pour concrétiser cette pensée au moyen d'un exemple, supposons qu'on doive réaliser un comparateur de deux nombres A et B fournissant une sortie S = 1 si A ≥ B et S = 0 si A < B, avec

A = an an-1 ... ak ... a2 a1 a0

B = bn bn-1 ... bk ... b2 b1 b0

Il est facile de décider que chaque sous réseau reçoit à son entrée une variable ak et bk. Mais doit-on propager la comparaison à partir du chiffre le plus significatif vers le moins significatif, ou vice-versa? Comment la sortie S est-elle confectionnée? Utilise-t-on les sorties Yi? Il est difficile à ce point de déterminer ce qui donnera le réseau minimal. L'expérience et l'intuition sont des facteurs à développer et à exploiter.

Cette étape devrait définir les points suivants:

- Un sous-réseau typique (Ck).
- Les sous-réseaux aux extrémités (Cn et C0).

- Les variables d'entrée/sortie sont conformes aux spécifications.
- Un réseau auxiliaire, si requis.

ÉTAPE 2: Algorithme

Après avoir défini le schéma bloc du réseau, il faut définir les conséquences des décisions qui sont prises. Il faut définir ce qu'un sous-réseau Ck effectue sur les entrées en fonction de ce qu'il reçoit des sous-réseaux voisins, ce qu'il doit transmettre vers ces sous-réseaux et vers la sortie si nécessaire. C'est à cette étape qu'on définit les conventions adoptées relativement aux variables internes du réseau. Il est très important d'expliciter clairement la spécification des relations entrées/sorties d'un sous-réseau type avec les hypothèses qui s'y rattachent.

Cette étape devrait définir les points suivants de l'algorithme:

- Le fonctionnement de l'ensemble.
- Les variables d'entrée/sortie de Ck..
- Les relations de transfert de Ck.
- Les hypothèses sur les variables.

ÉTAPE 3: Synthèse d'un sous-réseau type

La synthèse d'un sous-réseau type s'effectue selon les méthodes relatives aux réseaux combinatoires. On considère le sous-réseau Ck avec ses entrées Xk, Fk+1, et Gk et ses sorties Yk, Fk et Gk+1. Une table de vérité est établie pour représenter les diverses fonctions logiques. Celles-ci sont réduites par algèbre, tables ou algorithme et le sous-réseau est dessiné dans sa forme schématique.

Cette étape doit contenir les points suivants:

- La table de vérité contenant toutes les sorties.
- Les tables de Karnaugh.
- Les expressions logiques de chacune des sorties.
- Le diagramme schématique du sous-réseau Ck.

ÉTAPE 4: Particularités aux extrémités (conditions aux frontières)

Dans la plupart des cas, il faut porter une considération particulière aux sous-réseau: Cn et C0 . Il s'agit souvent de définir le niveau logique d'une variable pour amorcer l'opération du réseau ou pour rendre le résultat conforme aux spécifications.

Cette étape n'est pas complète si les points suivants ne sont pas abordés:

- Conditions particulières à C0.
- Conditions particulières à Cn.

Les exemples qui suivent illustrent ce point et les étapes qui précèdent.

Exemple 4.1

Un comparateur de deux nombres binaires A et B doit produire une sortie S = 1 si A >= B et S = 0 sinon.

Les nombres A et B sont constitués d'un grand nombre de chiffres binaires.

A = an an-1 ... ak ... a2 a1 a0

B = bn bn-1 ... bk ... b2 b1 b0

Pour démontrer la non-unicité des solutions, le design de ce réseau est effectué de diverses façons. Ceci met également en évidence l'importance des deux premières étapes de la méthodologie.

Solution 1

Même si à priori, par intuition, on devrait effectuer la comparaison du chiffre le plus significatif vers le moins significatif, la comparaison sera amorcée entre a0 et b0 et procédera vers la gauche.

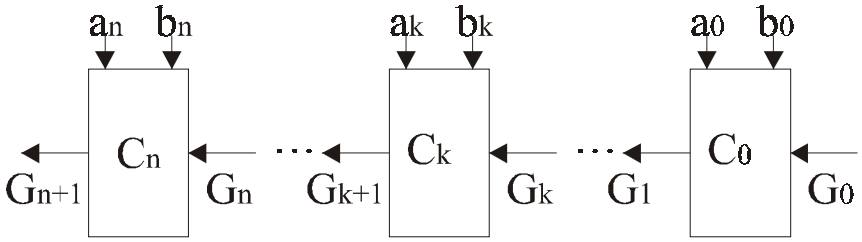


Figure 4.2: Schéma bloc de l'exemple 4.1.

Algorithme

La comparaison s'effectuant de C0 vers Cn , il est évident que la décision finale ne sera pas prise avant de parvenir au sous-réseau Cn.

Un sous-réseau type Ck doit recevoir l'information à savoir si les chiffres de rangs inférieurs signalent que A >= B ou que A < B. Une seule variable est nécessaire pour identifier l'un au l'autre de ces états.

Après avoir combiné les entrées ak, bk et Gk, le sous-réseau Ck doit propager une variable Gk+1 vers le sous-réseau Ck+1. En procédant de cette façon jusqu'au sous-réseau Cn , on comprend que Gn+1, constitue la sortie S du réseau.

Les variables Fi et Yi n'ont pas d'utilité dans cette application et la seule hypothèse à expliciter est la correspondance entre le niveau logique des Gi, et sa signification. Ici il semble évident qu'on évitera un OOPS à la sortie si Gi = 1 correspond *à* A >= B.

Cet exposé est un peu plus détaillé que nécessaire. Lors d'un design, il suffit d'expliciter comment l'ensemble du réseau effectue la fonction, ce qu'un sous-réseau type doit recevoir et transmettre et finalement les conventions rattachées aux variables.

Synthèse du sous-réseau Ck

Ce qui précède permet d'écrire la table de vérité suivante:

Gk	ak	bk	Gk+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

La table de Karnaugh:

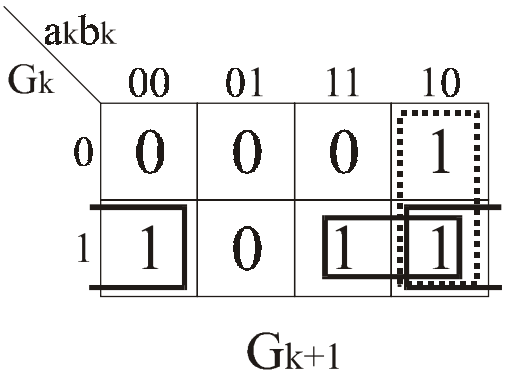


Figure 4.3: Table de Karnaugh de l'exemple 4.1.

D'où $G_{k+1} = a_k\overline{b_k} + G_k(a_k + \overline{b_k})$

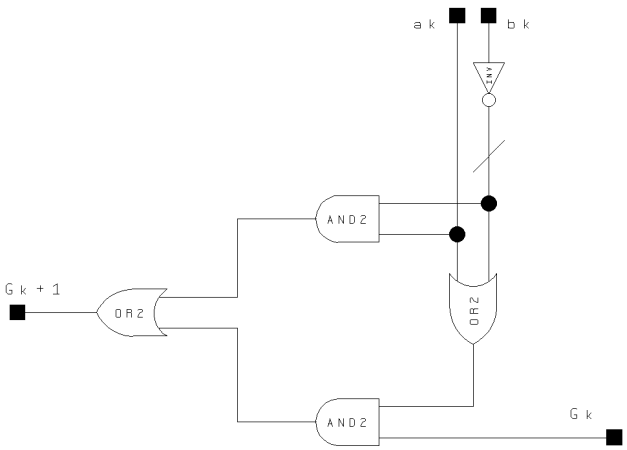


Figure 4.4: Schéma d'une cellule du comparateur.

Particularités aux extrémités

Sous-réseau C0 :

La particularité du sous-réseau C0 provient du fait que son entrée G0 n'est pas déterminée. Il faut par conséquent, forcer G0 à "0" ou à "1". On peut utiliser la table de vérité décrite à l'étape précédente ou raisonner sur le schéma. Selon l'algorithme, si a0 = b0 , on veut G1 = 1 puisqu'on a égalité. La table de vérité indique qu'on doit forcer G0 = 1.

Sous-réseau Cn :

Le sous-réseau Cn a ceci de particulier: sa sortie Gn+1 constitue la sortie S du réseau itératif global.

Solution 2

Il est plausible de concevoir un comparateur itératif effectuant la fonction spécifiée ci-dessus à partir des bits an et bn et procédant vers la droite. Même si la fonction globale reste la même. cette décision mène à une conception différente. En effet. si on a égalité dans les bits de rangs supérieurs, il faut propager cette information vers les sous-réseaux de rangs inférieurs puisque la décision doit y être prise. Si on a inégalité entre deux bits ai et bi, les sous-réseaux de rangs inférieurs sont inhibés et on utilise une combinaison des Xi, pour confectionner la sortie S.

Schéma bloc:

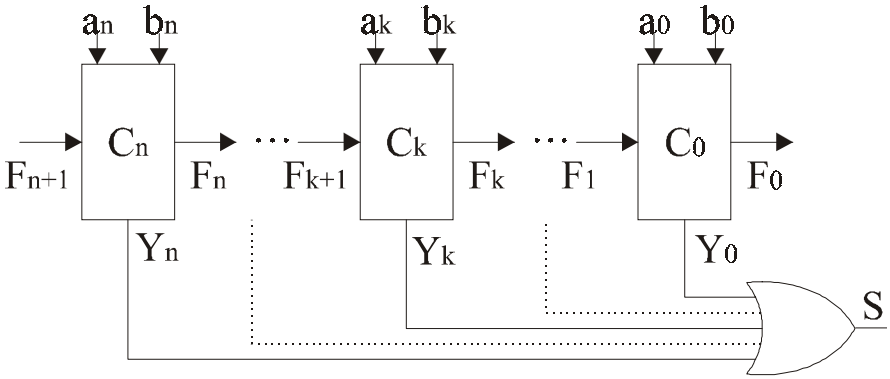


Figure 4.5: Schéma bloc de la solution 2.

Algorithme

Un sous-réseau Ck doit recevoir les entrées:

- Fk+1 pour indiquer si on a égalité dans les sous-réseaux de rangs supérieurs

  Hypothèse: FK+1 indique égalité
- ak et bk des nombres à comparer.

Le sous-réseau doit transmettre:

- une sortie Fk pour propager le signal d'égalité
- une sortie Yk puisque la sortie S = SYi,

  Si Fk+1 = 0 : Yk = 0

  Si Fk+1 = 1 : Yk = 1 si ak > bk

  Yk = 0 si ak < bk

  Yk = 0 si ak = bk

Cette dernière condition ne semble pas conforme aux spécifications, mais elle est nécessaire dans la configuration choisie. En effet, si on a égalité jusqu'au sous-réseau de rang k, on pourrait trouver dans les sous-réseaux de rangs inférieurs que A > B ou A < B ou A = B.

Synthèse d'un sous-réseau

Fk+1	ak	bk	Fk	Yk
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

$$F_k = F_{k+1}\overline{(a_k \oplus b_k)}$$

$$Y_k = F_{k+1}a_k\overline{b_k}$$

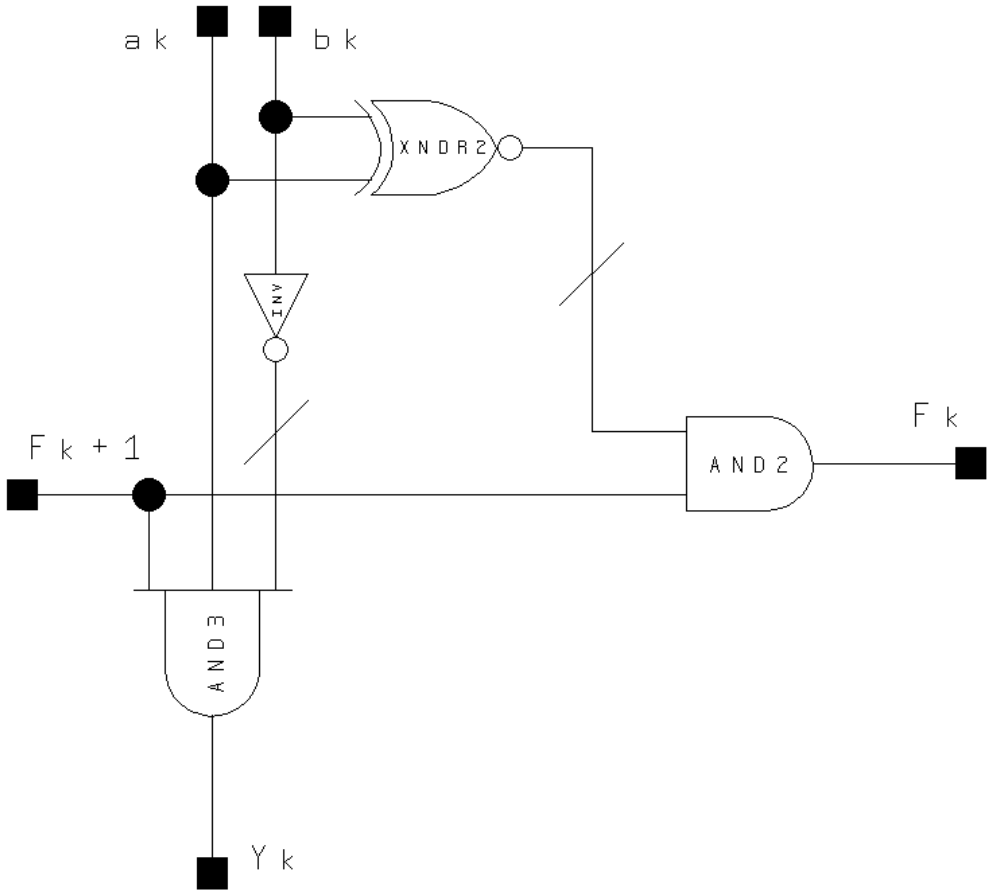


Figure 4.6: Schéma de la solution 2.

Particularités aux extrémités

Sous-réseau C0

Si on a égalité A et B, on veut S = 1 et on a F0 = 1. Donc on connecte F0 à l'entrée de la fonction "OU".

Sous-réseau Cn

Fn+1 = 1 puisqu'on a pas d'inégalité dans les rangs supérieurs à n qui sont inexistants.

Remarque

On obtient une solution différente, plus ou moins lourde selon les décisions du concepteur dans les premières étapes de la synthèse d'un réseau itératif.

CHAPITRE 5

Codes analytiques et représentatifs

5.1 Définitions

Mot

Un mot est une séquence de symboles représentant un message.

Exemple: livre, 164885, ψνφαυρ.

Mot binaire

Un mot binaire est une séquence de symboles dans laquelle on ne retrouve que deux symboles, généralement 0 et 1, qu'on désigne par chiffres binaires, ou bits.

Exemple: 0110 1001, 1000 1111 1001.

Code

Un code est un ensemble de mots auxquels on confère une signification pour représenter une catégorie de messages: des personnes, des objets, des nombres, des angles, des coordonnées et autres concepts.

Code binaire

Un code est binaire s'il ne contient que des mots binaires.

5.2 Codes fréquents

Cette section décrit, dans un ordre quelconque, les codes les plus fréquemment rencontrés.

5.2.1 Binaire naturel

Le code binaire naturel est une interprétation de plusieurs variables logiques qui sont exprimées ensemble. Les nombres binaires naturels sont toujours représentés du bit le plus

significatif au moins significatif, de la gauche vers la droite. Par analogie aux nombres décimaux où chaque chiffre est un multiplicateur d'une puissance de dix, chaque chiffre qui le compose est le multiplicateur d'une puissance de deux. Pour un nombre de *n* bits, le bit le plus significatif aura un poids de 2^{n-1}, le bit le deuxième plus significatif vaudra 2^{n-2}, ainsi de suite jusqu'à 2^0, le bit le moins significatif. La plage des valeurs possibles à représenter va de 0 à 2^n-1.

Exemple 5.1

Le nombre 123 peut être exprimé de la façon suivante:

Nombre décimal	Nombre binaire
$123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$	$123 = 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
$123 = 100 + 20 + 3$	$123 = 64 + 32 + 8 + 4 + 2 + 1$

5.2.2 Binaire naturel avec signe

Avec les algorithmes de traitement de données, le besoin de représenter des nombres négatifs apparaît très souvent. Une des façons les plus simples est de dédier un bit, habituellement le plus significatif, pour représenter le signe. Pour un mot de *n* bits, le bit le plus significatif représente le signe, le bit le deuxième plus significatif vaudra 2^{n-2}, ainsi de suite jusqu'à 2^0, le bit le moins significatif. La plage des valeurs possibles à représenter va de -2^{n-1}-1 à 2^{n-1}-1 et il existe deux représentations possibles pour la valeur 0.

Exemple 5.2

26=	0	0011010
	Bit de signe	Représentation de 26

-26=	1	0011010
	Bit de signe	Représentation de 26

5.2.3 Complément à 1

Dans un code à complément à 1, on obtient le code du négatif d'un nombre en effectuant le complément du mot de code de ce nombre, c'est-à-dire en inversant tous les bits. Avec un mot de *n* bits, il est possible de représenter les valeurs de -2^{n-1}-1 à 2^{n-1}-1 et il existe deux représentations possibles pour la valeur 0.

Exemple 5.3

26= 0001 1010

-26= 1110 0101

5.2.4 Complément à 2

Dans un code à complément à 2, on obtient le code du négatif d'un nombre en effectuant le complément du mot de code de ce nombre et en lui ajoutant 1 sans garder de retenue. Avec un mot de *n* bits, il est possible de représenter les valeurs de -2^{n-1} à $2^{n-1}-1$ et il n'existe qu'une représentations possibles pour la valeur 0.

Exemple 5.4

26=	0001 1010
inversion	1110 0101
+1	1
	1110 0110 = -26

Il est aussi possible d'interpréter un nombre de *n* bits de la façon suivante:

bit	n-1	n-2	…	1	0
poids	-2^{n-1}	2^{n-2}	…	$2^1=2$	$2^0=1$

Le code à complément à 2 est très important puisqu'il permet d'effectuer des opérations arithmétiques avec signes sans avoir à se préoccuper de celui-ci.

Exemple 5.5

Faire l'addition de 127 et -26 ainsi que de -127 et 26

127	0111 1111		0111 1111
-26	- 0001 1010	←→	+ 1110 0110
101			1 0110 0101 = 101 (décimal)

La retenue est rejetée.

26	0001 1010		0001 1010	
-127	- 0111 1111	←→	+ 1000 0001	
-101			1001 1011	Ce résultat est négatif puisque le bit le plus significatif est à "1". Il faut faire le complément à 2.
			0110 0100	Inversion
			+ 1	
			0110 0101	= -101 (décimal)

Dans ce genre d'opérations, une erreur de débordement (*Overflow*) est détectée en effectuant un XOR entre les retenues transmises par les deux bits les plus significatifs.

Exemple 5.6:

Ces opérations sont effectuées sur des nombres de 8 bits en complément à 2.

	Retenues	0	0	0	1	1	0	1	0
26		0	0	0	1	1	0	1	0
+ 27		+ 0	0	0	1	1	0	1	1
53		0	0	1	1	0	1	0	1

$0 \oplus 0 = 0$ → pas de débordement

	Retenues	0	1	1	1	1	1	0
126		0	1	1	1	1	1	1	0
+ 127		+ 0	1	1	1	1	1	1	1
253		1	1	1	1	1	1	0	1

$0 \oplus 1 = 1$ → débordement

	Retenues	1	1	0	0	0	0	0	0
- 60		1	1	0	0	0	1	0	0
- 64		+ 1	1	0	0	0	0	0	0
- 124		1	1	0	0	0	0	1	0	0

$1 \oplus 1 = 0$ → pas de débordement

	Retenues	1	0	0	0	0	0	0	0	
- 100				1	0	0	1	1	1	0	0
- 30			+	1	1	1	0	0	0	1	0
-130		1		0	1	1	1	1	1	1	0

$1 \oplus 0 = 1 \rightarrow$ débordement

5.2.5 Code hexadécimal

Avec l'apparition des processeurs 8, 16 et surtout 32 bits, la notation binaire est vite devenue pénible. La notation hexadécimale n'est qu'une façon d'écrire les nombres binaires avec une représentation plus compacte. L'interprétation du nombre binaire peut être binaire naturelle, binaire avec signe, complément à 1 ou à 2. En notation hexadécimale, les nombres binaires sont séparés en groupes de quatre bits. Chaque groupe est ensuite remplacé par un des 16 symboles de la notation hexadécimale. Les nombres de 0 à 9 sont replacés par les chiffres 0 à 9 et les nombres de 10 à 15 sont remplacés par les lettres A à F.

Exemple 5.7:

Énoncer les nombres suivants en notation hexadécimale:

234 en binaire naturel sur 8 bits

$$234 = \underset{E}{1110}\ \underset{A}{1010} \qquad (234)_{10} = (1110\ 1010)_2 = (EA)_{16}$$

-44 en complément à 2 sur 8 bits

$$-44 = -0010\ \ 1100 \rightarrow \underset{D}{1101}\ \underset{4}{0100} \qquad (-44)_{10} = (1101\ 0100)_2 = (D4)_{16}$$

5.2.6 Code octal

Durant l'évolution des ordinateurs modernes, certaines compagnies ont développé des ordinateurs basés sur des bus à 12 bits. Il était alors pratique courante d'utiliser une représentation octale. La représentation octale groupe les bits et les remplace par un symbole. Cette fois, les bits sont par groupes de 3 et les symboles vont de 0 à 7.

5.2.7 Code BCD (*Binary Coded Decimal*)

Le code BCD a surtout été utilisé aux débuts des machines à calculer. Il est parfois encore utilisé dans des systèmes qui doivent souvent afficher de l'information numérique à l'usager ou pour des systèmes très simples qui manipulent de l'argent. Le code BCD encode le nombre à représenter de façon très directe. Chaque chiffre du nombre est encodé sur 4 bits. Les possibilités binaires de 10 à 15 ne sont pas utilisées.

Exemple 5.8

Représenter 863 avec un code BCD

8	6	3
0100	0110	0011

5.2.8 Code de Gray

Le code de Gray a été établi pour éviter les problèmes de transition lors du changement d'un mot au mot suivant. Dans l'ordre séquentiel du code, il n'y a qu'un bit qui change entre deux mots consécutifs qui sont, par conséquent, adjacents. De plus, ce code est cyclique.

Exemple 5.9

Code de Gray à 4 bits

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

Tableau 5.1: Code de gray à 4 bits.

Il est possible d'obtenir le code de Gray à partir du code binaire naturel. Il suffit de faire un XOR entre le mot binaire nature et le même mot décalé vers la droite. La figure 5.1 montre le schéma d'un circuit de conversion.

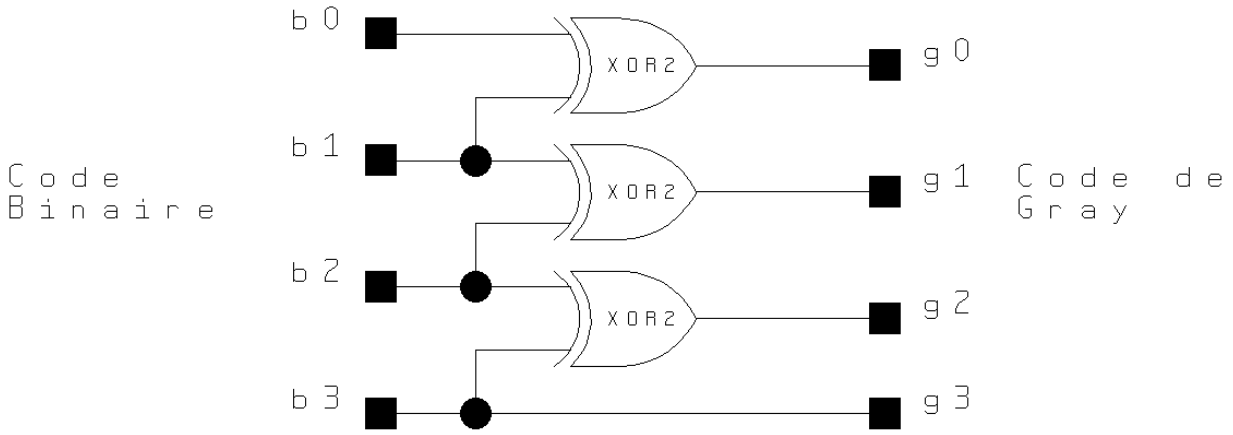


Figure 5.1: Schéma de conversion du code binaire vers le code de Gray.

5.3 Application des codes à la détection et correction d'erreurs

Cette section décrit différentes méthodes d'encodage pour transmettre de façon plus fiable de l'information binaire sur un canal de transmission. Différents concepts de fiabilité de transmission sont énoncés et une attention particulière est apportée aux deux grandes méthodes pour augmenter la fiabilité d'une transmission: la redondance et la parité.

5.3.1 Définitions

Système de transmission binaire

La figure 5.2 représente les principaux éléments d'un système de communication par transmission de codes binaires. La principale source de problèmes dans un tel système est évidemment le bruit qui perturbe aléatoirement le signal transmis.

Si on transmet un 1, il est probable qu'un 1 soit reçu, il est aussi possible qu'un 0 soit reçu. Ces probabilités constituent une caractéristique du canal de transmission telle qu'illustrée à la figure 5.3.

Dans cette représentation, Pij signifie: probabilité de recevoir un j étant donné qu'un i a été transmis. En général, on utilise le modèle du canal binaire symétrique (figure 5.3 où e <= 0.5, ce modèle étant adéquat pour représenter un canal de transmission binaire réel.

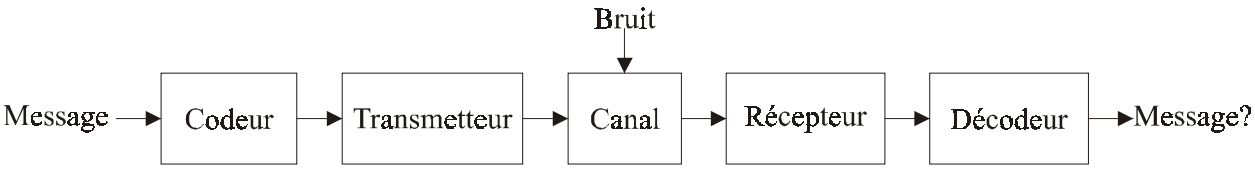


Figure 5.2: Système de transmission binaire.

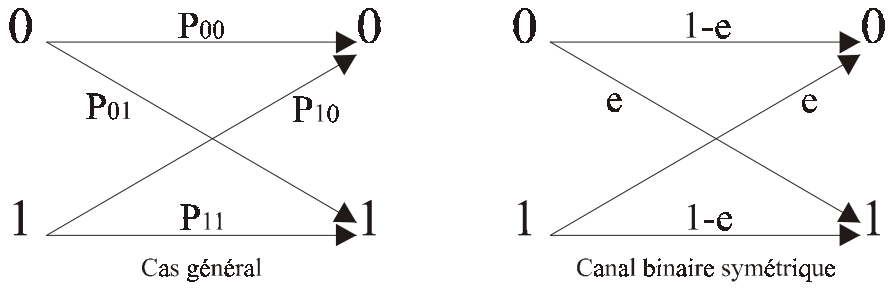


Figure 5.3: Canaux bruyants.

Efficacité d'un code

À chaque message différent pouvant être transmis, on doit évidemment associer un mot de code unique. Le choix du code à assigner est très important si le canal est loin d'être idéal.

Exemple 5.10

Soit A et B, deux messages pouvant être transmis; on les code de la façon suivante:

Message	Code
A	0
B	1

Si le canal binaire symétrique est idéal (e = 0) alors la transmission sera parfaite. Par contre, si e = 0.3, la probabilité d'erreur (recevoir A si B est transmis ou recevoir B si A est transmis) est de 0.3.

Un autre code peut être utilisé pour augmenter la fiabilité:

Message	Code
A	000
B	111

À la réception, il y a 8 codes possibles:

000	100
001	101
010	110
011	111

Établissons la règle du maximum de vraisemblance pour déterminer quel message a été transmis: si dans le mot de code reçu les 0 sont majoritaires, alors on supposera qu'un A (000) à été transmis; si les 1 sont majoritaires, on supposera qu'un B (111) a été transmis.

Mot de code reçu	Décision quant au message transmis
000	A
001	
010	
100	
111	B
110	
101	
011	

On peut démontrer que la probabilité d'erreur dans la décision quant au message transmis est de 0.216, comparativement à 0.3 avec le code précédent. Soulignons toutefois que cette diminution de la probabilité d'erreur coûte une transmission de trois fois plus de bit par message.

Distance de Hamming

La distance de Hamming est le nombre de bits qui changent entre deux mots d'un code.

Exemple 5.11

$d_H(0010,1000) = 2$

$d_H(000,111) = 3$

Distance minimale d'un code

La distance minimale d'un code est le nombre minimal de bits qui changent entre deux mots valides d'un code. La distance de Hamming du code binaire naturel est de 1 car il existe au moins deux mots ayant une distance de Hamming de 1.

$d_H(000,100) = 1$

Le code où A=(000) et B=(111) à une distance minimale de 3.

Relation fondamentale

La théorie de l'information stipule que:

$M - 1 = C + D$

où: M = Distance minimale du code

D = Nombre maximal d'erreurs détectables

C = Nombre maximal d'erreurs corrigibles

Remarque

On ne peut pas corriger plus d'erreurs qu'on en détecte donc:

$C \leq D$

Par conséquent, pour qu'un code soit capable de détecter et de corriger une erreur de transmission:

$C = D = 1$

$M - 1 = C + D$

$M = 3$

5.3.2 Redondance

Une des façons les plus simples d'augmenter la fiabilité d'une transmission est d'utiliser la redondance. Si le message est transmis plusieurs fois, il est relativement simple de détecter, et même corriger, des erreurs.

Exemple 5.12

Cet exemple illustre les effets de la redondance avec un code à deux messages, A et B.

a) Avec 1 bit

A → 0
B → 1 M = 1

Transmis	Reçu	
A → 0	1 → B	Interprété comme B

L'erreur n'est donc ni détectée, ni corrigée.

Généralisation

Lorsque toutes les combinaisons de bits qui forment un code sont utilisées pour les messages, une erreur n'est ni détectable, ni corrigible.

C = D = 0

b) Avec 2 bits

A → 00
B → 11 M = 2 C + D = 1

Transmis	Reçu	
A → 00	01 → ?	Erreur détectée, non corrigible.
	11 → B	2 erreurs non détectées

D'où D = 1 et C = 0.

Généralisation

On peut toujours conférer à un code une distance minimum M = 2 en ajoutant à chaque mot un bit de parité.

c) Avec 3 bits

A → 000
B → 111 M = 3 C + D = 2

Transmis	Reçu	
A → 000	001 → A	A avec une erreur (valide)
	011 → B	B avec une erreur (non valide)
	111 → B	B sans erreurs (non valide)

D'où D = 1 et C = 1.

d) Avec 4 bits

A → 0000
B → 1111 M = 4 C + D = 3

Transmis	Reçu	
A → 0000	0001 → A	A avec une erreur (valide)
	0011 → ?	2 erreurs (valide)
	0111 → B	B avec une erreur (non valide)
	1111 → B	B sans erreurs (non valide)

D'où D = 2 et C = 1.

Pour corriger des erreurs, on doit ajouter beaucoup de redondance qui diminue beaucoup d'efficacité du canal de transmission.

5.3.3 Parité orthogonale

Avant de procéder, il est nécessaire de définir deux types de parité qui seront utilisés pour la parité orthogonale.

Parité transversale

Soit le code suivant

Message	Code
A	00
B	01
C	10
D	11

Il est impossible de détecter ou de corriger une erreur car M = 1 ➔ C = D = 0.

En ajoutant un bit de parité à chaque code de façon à ce qu'il y ait toujours un nombre pair de 1 dans chaque mot du code on obtient:

Message	Code
A	000
B	011
C	101
D	110

S'il y a une erreur de transmission, elle pourra être détectée car les mots reçus ne devraient jamais avoir un nombre impair de 1. M = 2 ➔ D = 1, C = 0.

Il est aussi possible d'utiliser une parité impaire:

Message	Code
A	001
B	010
C	100
D	111

Parité longitudinale

En reprenant le code précédent:

Message	Code
A	00
B	01
C	10
D	11

Si le système doit transmettre le message A-B-A-C, le bloc de mots à transmettre est:

00
01
00
10

Si une erreur de transmission survient dans ce bloc, il est impossible de la détecter. En ajoutant un cinquième mot à transmettre de façon à avoir un nombre pair de 1 dans chaque colonne du bloc, on transmet:

00
01
00
10
11

Cela permet de détecter une erreur de transmission dans le bloc au complet.

Parité orthogonale

En combinant les parités transversales et longitudinales, la capacité d'un système augmente de façon significative. Cette combinaison se nomme parité orthogonale. En reprenant l'exemple précédent où la séquence A-B-A-C est transmise, on obtient:

00	0
01	1
00	0
10	1
11	0

S'il y a une seule erreur de transmission. Il est possible non seulement de la détecter mais aussi de la corriger car elle sera située à l'intersection d'une ligne et d'une colonne de parités fausses.

	00	0	
	01	1	
Bit en erreur ←	10	0	→ Parité fausse
	10	1	
Parité fausse ←	11	0	

S'il y a deux erreurs, elles seront détectées mais elles ne seront pas corrigibles. Cela implique que pour la parité orthogonale C = 1 et D = 2, d'où M=4. La distance minimale de la parité orthogonale peut être observée en ne changeant qu'un bit du message. Ce changement entraînera un changement dans la parité longitudinale, dans la parité transversale et à l'intersection des deux parités. Il y a donc une différence de 4 bits entre deux mots valides.

5.3.4 Code de Hamming

Un code de Hamming permet la détection et la correction d'une seule erreur de transmission. Les mots du code contiennent *m* bits pour le message et *k* bits pour la parité. Les *k* bits doivent être suffisants pour donner l'adresse (la position) de l'erreur dans le code reçu. L'erreur peut se situer, si elle existe, n'importe où dans le message reçu, soit à un bit de message m_i, soit à un bit de vérification k_i.

m et *k* doivent respecter la relation suivante: $m + k + 1 \leq 2^k$

Les premières valeurs de *m* et *k* qui satisfont cette relation sont:

k	m_{max}	
1	0	Aucun message.
2	1	Plus de redondance que de m.
3	4	Environ autant de message que de parité, redondance de 2.
4	11	m est plus grand que k.
5	26	m est beaucoup plus grand que k.
6	57	m est beaucoup beaucoup plus grand que k.

Structure des mots de Hamming

Les bits k sont répartis dans un mot de Hamming selon les puissances de 2.

Exemple 5.13

Formation d'un mot de Hamming avec m = 4 et k = 3 pour encoder le message 1011.

k_2 = Parité (5, 6, 7)

k_1 = Parité (3, 6, 7)

k_0 = Parité (3, 5, 7)

7	6	5	4	3	2	1
m_4	m_3	m_2	k_2	m_1	k_1	k_0
1	0	1		1		
			0		0	1
1	0	1	0	1	0	1

Décodage des mots

Codage de la position de l'erreur pour k=3, m=4:

E_2	E_1	E_0	Position	
0	0	0	0	← Pas d'erreur
0	0	1	1	
0	1	0	2	
0	1	1	3	E_2 = Parité(4, 5, 6, 7)
1	0	0	4	E_1 = Parité(2, 3, 6, 7)
1	0	1	5	E_0 = Parité(1, 3, 5, 7)
1	1	0	6	
1	1	1	7	

Exemple 5.14

Le mot de Hamming 1010101 qui contient le message 1011 a été transmis et les mots suivants ont été reçus:

a)

7	6	5	4	3	2	1
1	0	1	0	1	0	1

E_2 = Parité(4, 5, 6, 7) = 0

E_1 = Parité(2, 3, 6, 7) = 0

E_0 = Parité(1, 3, 5, 7) = 0 ➔ Position 000 = 0 → pas d'erreurs.

b)

7	6	5	4	3	2	1
1	**1**	1	0	1	0	1

E_2 = Parité(4, 5, 6, 7) = 1

E_1 = Parité(2, 3, 6, 7) = 1

E_0 = Parité(1, 3, 5, 7) = 0 ➔ Position 110 = 6 → Le mot corrigé est 1010101.

c)

7	6	5	4	3	2	1
1	0	1	0	1	**1**	1

E_2 = Parité(4, 5, 6, 7) = 0

E_1 = Parité(2, 3, 6, 7) = 1

E_0 = Parité(1, 3, 5, 7) = 0 ➔ Position 010 = 2 → Le mot corrigé est 1010101.

d)

7	6	5	4	3	2	1
1	[1]	1	0	1	[1]	1

E_2 = Parité(4, 5, 6, 7) = 1

E_1 = Parité(2, 3, 6, 7) = 0

E_0 = Parité(1, 3, 5, 7) = 0 ➔ Position 100 = 4→ Le mot corrigé est 1111111, ce qui est invalide. En présence de deux erreurs le système est incapable de détecter correctement les deux bits qui ont changés au cours de la transmission. La limite de code de Hamming est D = 1 et, lorsqu'une est erreur est présente, elle est corrigible donc C = 1.

M = C + D + 1 = 3

Encodage de parité

Dans le code de Hamming à 7 bits, un bit de parité doit être généré à partir de trois bits.

A	B	C	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Cette fonction est une chaîne de XOR où P = A ⊕ B ⊕ C.

Décodage de parité

Dans le code de Hamming à 7 bits, un bit de parité doit être généré à partir de 4 bits lors de la vérification des mots reçus.

A	B	D	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Pour une parité à partir de 4 bits, la fonction est $P = A \oplus B \oplus C \oplus D$. Pour des codes de Hamming à plus de 7 bits, il faut ajouter des étages à la chaîne de XOR. Une parité à partir de 5 bits sera la fonction $P = A \oplus B \oplus C \oplus D \oplus E$, et ainsi de suite.

CHAPITRE 6

La logique séquentielle

Dans tout ce qui a été vu jusqu'à maintenant, les sorties des systèmes pouvaient toujours être exprimées en fonction des entrées. Dans les systèmes séquentiels, les sorties dépendent aussi de l'état du système. L'état d'un système évolue en fonction des séquences reçues à son entrée, de son état présent et de son état initial. Pour une même combinaison d'entrées, il est donc possible d'avoir plusieurs états du système et plusieurs combinaisons de sorties. La distinction entre les états s'effectue au moyen de variables internes, dites variables d'état.

6.1 Définitions

Notion de stabilité

Les machines séquentielles sont des circuits où l'état actuel des variables dépend de la séquence de leurs états antérieurs. Les séquences sont caractérisées par une phase d'instabilité avant d'atteindre l'état stable final de l'étape.

Exemple 6.1

Le circuit suivant est dans son état initial A = Y = 0:

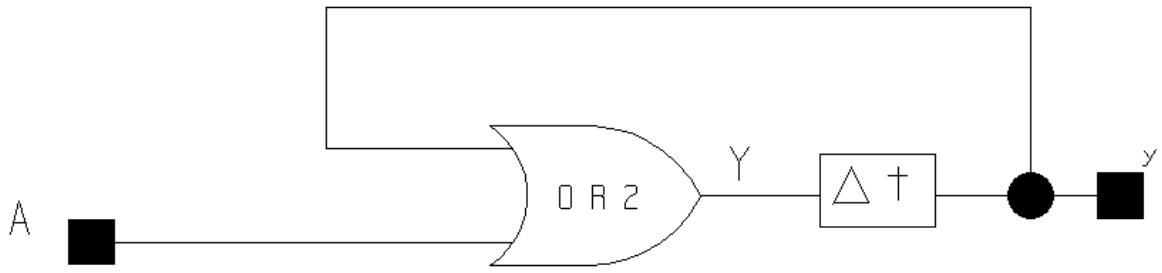


Figure 6.1: Réseau séquentiel simple.

L'état initial est stable. Une transition de A vers 1 entraîne un état instable pendant le temps Δt où Y = 1 et y = 0. Au moment où y devient 1, le système est stable. Une transition de A vers 0 n'affecte pas le circuit qui reste dans un état stable où y = 1.

Entrée	Excitation	État	
A = 0	Y = 0	y = 0	→ état stable
A = 1	Y = 1	y = 0	→ état stable
A = 1	Y = 1	y = 1	→ état stable
A = 0	Y = 1	y = 1	→ état stable

Lorsque l'excitation et la variable d'état sont à des niveaux logiques identiques, on a la stabilité. Cette notion se généralise à plusieurs états.

État

L'état d'une machine séquentielle est une combinaison d'états logiques de variables internes en contre-réaction ayant une valeur particulière. Cette valeur peut, ou non, avoir une signification particulière à haut niveau. Par exemple, il est possible de définir au moins trois états pour un four micro-onde à contrôle numérique:

1. Programmation;
2. Activation;
3. Attente.

Il est cependant probable que le système utilise, à l'interne, beaucoup plus de trois états.

Forme générale

Les machines séquentielles peuvent être séparées en deux grandes catégories: les machines de Mealy et les machines de Moore. Les deux machines fonctionnent sensiblement de la même façon au niveau des états et de leurs transitions. La différence se situe au niveau des sorties où les sorties des machines de Mealy dépendent à la fois des entrées et des états tandis que les sorties des machines de Moore ne dépendent que des états. La figure 6.2 montre la forme générale des circuits séquentiels.

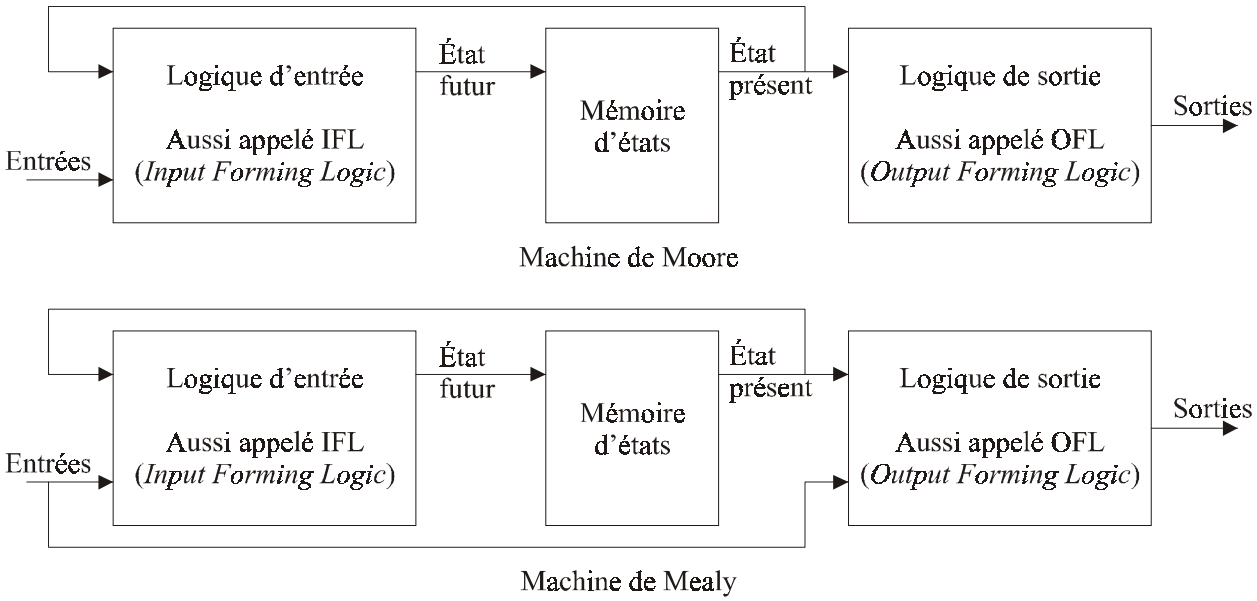


Figure 6.2: Aperçu général des machines de Mealy et de Moore.

6.2 Systèmes asynchrones

Les machines asynchrones ont la particularité de changer d'état lors d'une variation sur les entrées. Si les entrées demeurent stables, le système demeurera dans le même état. La mémoire d'état tel que représenté à la figure 6.2 est faite par un circuit de délai. Le système est stable lorsque la valeur avant et après le délai sont identiques (l'état et l'excitation sont égaux).

6.2.1 Analyse

La méthode de Huffman d'analyse des machines séquentielles consiste à élaborer une table de Karnaugh à partir des équations du circuit à analyser. Les états stables y sont identifiés par la correspondance entre les états d'excitation et leurs effets. Voici une liste complète des étapes pour l'analyse des circuits séquentiels asynchrones:

1. Par inspection du diagramme d'un système existant, on écrit la fonction d'excitation de chaque variable d'état.
2. On dispose chaque fonction dans une table de Karnaugh: les colonnes sont désignées par des combinaisons de variables d'entrée, et les lignes, par des combinaisons de variables d'état.
3. Une table d'excitation est confectionnée en fusionnant les tables ainsi obtenues, en groupant dans chaque fenêtre les excitations de toutes les variables d'état.

4. Les états stables sont identifiés (excitation = état) et encerclés pour les mettre en évidence.

5. La table d'excitation permet de déterminer le comportement détaillé du système et de statuer sur son caractère prévisible ou aléatoire.

6. II est habituellement utile d'associer des noms ou des nombres décimaux aux états stables et instables.

7. On peut illustrer le comportement du système par une table de transitions ou un diagramme de transitions d'états.

Exemple 6.2

Analyser le circuit suivant:

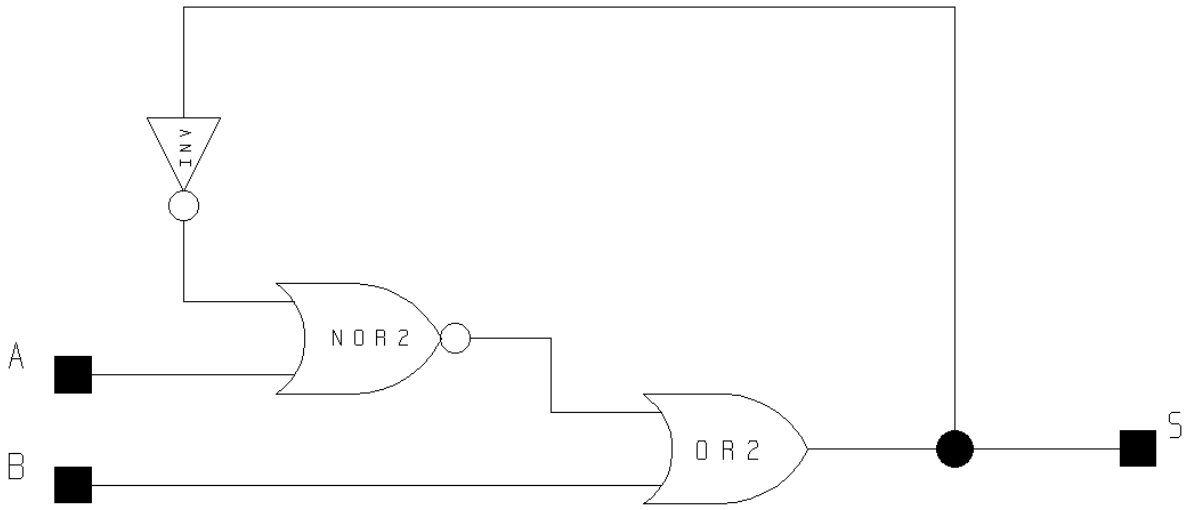


Figure 6.3: Machine asynchrone.

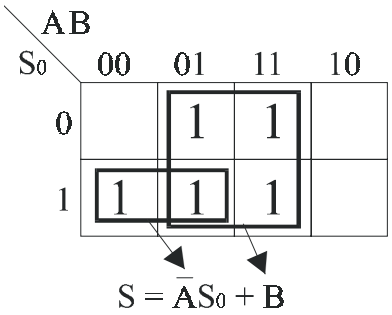
Soit S_0 représentant l'état initial de la sortie S, l'équation du circuit est:

$$S = \overline{(\overline{S_0} + A)} + B$$

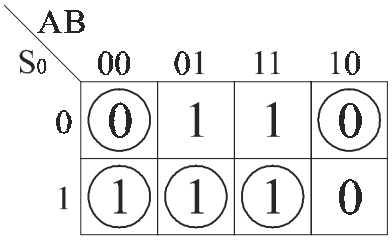
En utilisant la loi de Morgan, l'équation peut être écrite sous cette forme:

$$S = \overline{A}S_0 + B$$

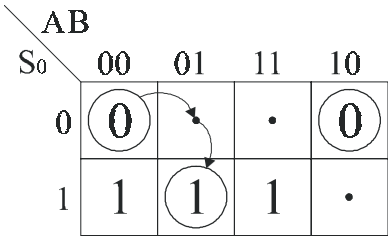
En représentant cette équation dans une table de Karnaugh, on obtient:



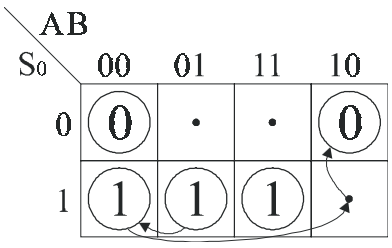
Il faut ensuite identifier les états stables où la fonction S est égale à S_0. Les états stables sont identifiés en les encerclant.



Cette table permet aisément de voir quels sont les états stables du circuit et nous renseigne sur la façon d'effectuer les changements. Par exemple, avec S0=0, A = B = 0, si on veut S = 1, on doit changer l'état de B. Le circuit se stabilisera à S0 = 1, A = 0 et B = 1.

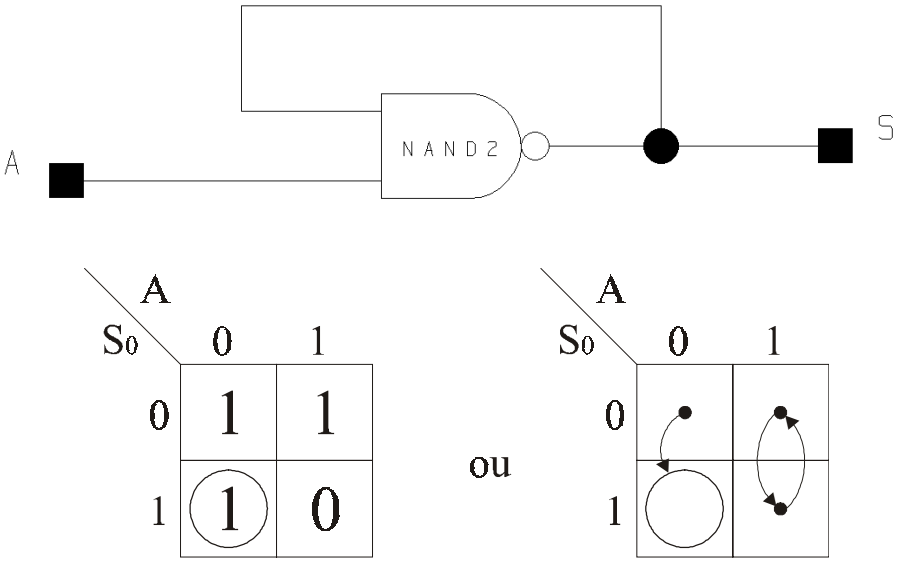


En remettant B = 0, le circuit demeure dans un état stable avec S = 1 car l'état S0 = 1, A = B = 0 est aussi un état stable. Pour replacer la sortie à 0, il faut poser A = 1 pour que le circuit se stabilise à S0 = 0, A = 1 et B = 0.



Exemple 6.3

Analyser la machine suivante:



La table nous montre que pour A = 1, il n'y a pas d'état stable; le système oscillera. Si A devient 0, la sortie stabilisera à 1.

6.2.2 Synthèse

Un réseau séquentiel asynchrone comprend un ensemble d'éléments de mémoire qui ne tombe pas sous la commande d'une horloge unique. La méthode d'Huffman a été formulée dans le but de concevoir un réseau séquentiel de ce type qui rencontre toutes les contraintes définies dans ses spécifications en nécessitant un minimum de composants.

Si la synthèse est le processus inverse de 1'analyse décrite à la section 6.2.1, il demeure une nécessité d'ajouter des étapes en raison du nombre d'inconnus, dont le nombre d'éléments de mémoire requis pour supporter la fonction spécifiée.

La méthode est présentée par étape et illustrée par un exemple.

Étapes

1. À partir des spécifications, on écrit une table de séquences primaire pour décrire de façon primitive, l'évolution du système pour toutes les séquences des variables d'entrée.
   - On définit dans cette table les états du système en fonction des séquences des variables d'entrées.
   - Les combinaisons de variables d'entrées désignent les colonnes de la table. On juxtapose une colonne pour chaque sortie.
   - À chaque ligne de cette table, on ne définit qu'un seul état stable auquel correspond l'état des sorties.
   - On inscrit dans la table toutes les séquences qui sont possibles compte tenu des contraintes imposées par les spécifications du système.
   - On écrit la table sans séquences redondantes en exploitant les états stables déjà définis dans les lignes antérieures.
2. On réduit le nombre de lignes de cette table primaire au moyen d'un diagramme de fusionnement pour obtenir une table fusionnée. Le nombre de variables d'état sera ainsi minimisé.
   - On peut fusionner deux lignes de la table primaire pourvu que les noms soient équivalents dans chacune des colonnes.
   - Les états facultatifs peuvent être fusionnés avec n'importe quel état.
   - La ligne résultante conserve, dans l'ordre, les états stables, les états instables et les états facultatifs.
   - On dessine un diagramme de fusionnement pour optimiser le système de façon méthodique.
   - Un nœud correspond à une ligne de la table primaire, un arc indique un fusionnement possible entre deux lignes. Un ensemble de points complètement interconnectés correspond à une seule ligne de la table fusionnée.
3. On effectue une étude des transitions du système et on associe une combinaison des variables d'état à chaque ligne de la table fusionnée.

4. En appliquant le concept de stabilité, la table précédente est convertie en binaire et elle devient une table d'excitation. À ce stade, on produit l'expression logique des sorties du système.
5. La table d'excitation est décomposée en tables de Karnaugh et on écrit l'expression logique de l'excitation correspondant à chaque variable d'état.
6. On dessine le diagramme selon la convention et la technique choisie.

Chaque étape implique des opérations et des règles à suivre qui sont décrites dans la réalisation du système ci-après.

Exemple 6.4

On doit concevoir une machine ayant 2 entrées X1 et X0, et une sortie Z. Les relations entrées/sorties sont spécifiées comme suit:

- Les entrées X1 et X0 ne peuvent changer d'état simultanément.
- La sortie prend l'état Z = 1 si X1 = X0 = 1 si la séquence des entrées est la suivante:
  X1X0 = 00, 10, 11
- La sortie conserve l'état Z = 0 pour tout autre séquence des entrées X1X0.

Étape 1: Tableau des séquences primaires

Le tableau de primaire constitue une première description des relations entrées/sorties. Dans ce tableau, on doit:

- décrire toutes les séquences possibles des entrées;
- chaque ligne du tableau doit contenir un seul état stable;
- par conséquent, il existe autant de lignes au tableau que d'états stables.

Le tableau est constitué de colonnes correspondant aux états des entrées et aux sorties. Sur chaque ligne, on définit un état stable et des états instables auxquels on accède par des changements des variables d'entrée. Pour chaque état stable, on définit l'état des sorties.

Les états stables sont indiqués par des chiffres encerclés et les états instables, par des chiffres simplement. Les états impossibles sont indiqués par des "-". Voici le tableau primaire relatif à 1'exemple.

	X_1X_0				Z
	00	01	11	10	
1	①	4	-	2	0
2	1	-	3	②	0
3	-	4	③	6	1
4	1	④	5	-	0
5	-	4	⑤	6	0
6	1	-	5	⑥	0

Remarque:

Il est important de définir un nombre minimum de lignes. On y parvient en utilisant un état stable existant lors de l'entrée dans une colonne, pourvu que la sortie soit conforme aux contraintes des spécifications.

Étape 2: Diagramme de fusionnement

On confectionne le diagramme de fusionnement pour réduire au minimum le nombre de lignes définies dans le tableau primaire. Pour ce faire, il faut respecter les règles suivantes:

- on fusionne deux lignes si les nombres sont identiques dans les colonnes correspondantes;
- les états impossibles "-" permettent la fusion avec n'importe quel nombre;
- lors d'une fusion, on conserve, dans l'ordre, les états stables, les états instables et les états impossibles.

Pour visualiser les simplifications apportées par cette opération, on représente d'abord chaque ligne du tableau primaire par un point identifié par le numéro de la ligne. Un trait est ajouté entre les points correspondant aux lignes qu'il est possible de fusionner. Lorsqu'un groupe de points sont tous interconnectés entre eux, il est possible de fusionner toutes les lignes correspondantes en une seule. La figure 6.4 montre le diagramme de fusionnement de cet exemple.

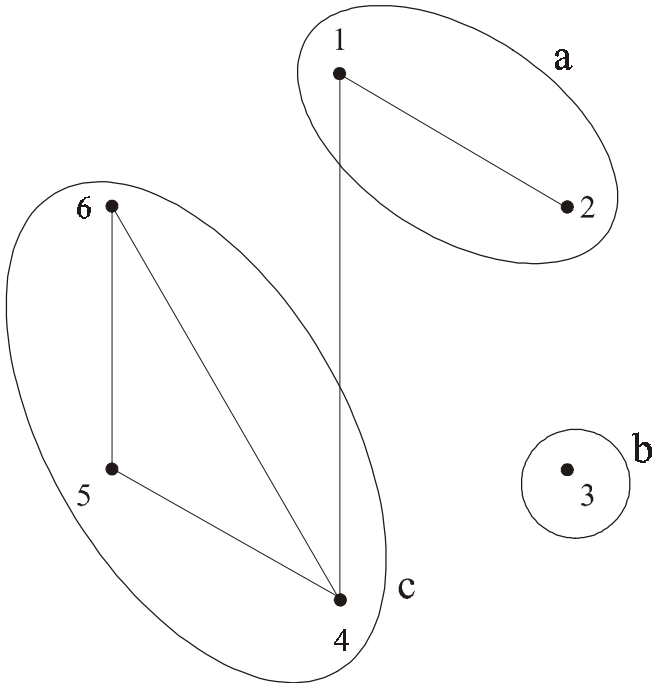


Figure 6.4: Diagramme de fusionnement de l'exemple 6.4.

Ce diagramme de fusionnement résulte du tableau primaire relatif au problème déjà amorcé. On obtient un tableau fusionné à trois lignes (a, b, c) où (a) est la fusion des lignes 1 et 2 et (c) est la fusion des lignes 4, 5 et 6. La ligne (b) est identique à la ligne 3 puisqu'elle n'a pas été fusionnée.

	X_1X_0			
	00	01	11	10
a	①	4	3	②
b	-	4	③	6
c	1	④	⑤	⑥

Remarques:

- Puisque plusieurs lignes ont été fusionnées, il est normal d'avoir plusieurs états stables par ligne.
- On ne se préoccupe pas des sorties à ce stade de la synthèse.

Étape 3: Représentation des états secondaires.

Chaque ligne du tableau fusionné doit être représenté par un ensemble de variables d'états. N variables d'états peuvent représenter 2^N lignes de tableau fusionné. De plus, une étude

des transitions est requise pour déterminer les aléas ou courses. Le choix des états représentant les lignes doit être effectué en minimisant le nombre de changements simultané des variables d'état.

Le tableau fusionné précédent a été réduit à trois lignes, 2 variables d'états seront suffisantes puisqu'elles permettent de représenter 4 lignes. Il reste à associer les lignes a, b et c à une combinaison de variables d'états Y_1 et Y_0. Pour ce faire, voici un diagramme des transitions possible dessiné à partir du tableau fusionné.

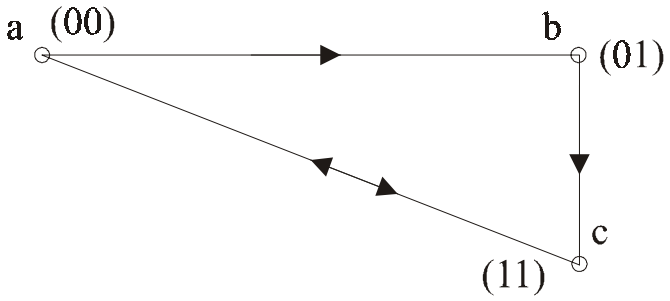


Figure 6.5: Diagramme des transitions.

Le tableau indique que le système peut passer de la ligne (a) à la ligne (b), de (b) à (c), de et de (a) à (c). En associant $Y_1Y_0 = 00$ à (a) et $Y_1Y_0 = 01$ à (b), cette transition implique le changement d'une seule variable, ne créant pas d'aléa. Si la ligne (c) correspond à $Y_1Y_0 = 11$, le transfert de (b) à (c) ne représente pas d'aléa lui non plus.

Il y a des aléas lorsque le système passe de (a) à (c) et de (c) à (a) car Y_1Y_0 tente de passer de (00) à (11) ou l'inverse. En réalité, les variables d'états passeront par 10 ou 01 puisqu'il est impossible que les deux variables changent d'état simultanément. Il faut donc tenir compte de ces états intermédiaires et le tableau avec représentation des états secondaires devient:

	X_1X_0			
Y_1Y_0	00	01	11	10
00	①	4	3	②
01	1	4	③	6
11	1	④	⑤	⑥
10	1	4	X	X

Étape 4: Définition des excitations.

À partir du moment où on a défini les états secondaires, on peut revenir à la représentation binaire des excitations et les décrire analytiquement. C'est également un bon endroit pour retrouver la sortie définie pour les 6 états stables. Le tableau précédent devient:

y_1y_0 \ X_1X_0	00	01	11	10
00	(00)	11	01	(00)
01	00	11	(01)	11
00	00	(11)	(11)	(11)
01	00	11	XX	XX

Y_1Y_0

X_1X_0	00	01	11	10
	0	X	X	0
	X	X	1	X
	X	0	0	0
	X	X	X	X

Z

Tableau 6.1: Tableau d'excitation complet.

La table représentant la sortie est remplie de conditions facultative à tous les états instables, ce qui permet. d'écrire pour la sortie:

$$Z = \overline{Y_1}Y_0$$

Remarque:

La présence de conditions facultatives de la sortie correspondant aux états instables du système est un choix du concepteur. Elle mène à une expression plus simple de la sortie Z. L'inconvénient qu'il faut attendre en retour est la présence d'impulsions courtes sur la ligne de sortie si le système passe par ces états instables. Il incombe au concepteur de juger de la pertinence de cette décision.

On peut décrire l'excitation des éléments du système avec le tableau précédent. Il faut le subdiviser en autant de tables qu'il existe d'éléments dans le système. Le système comporte deux éléments de mémoire: Y_1 et Y_0.

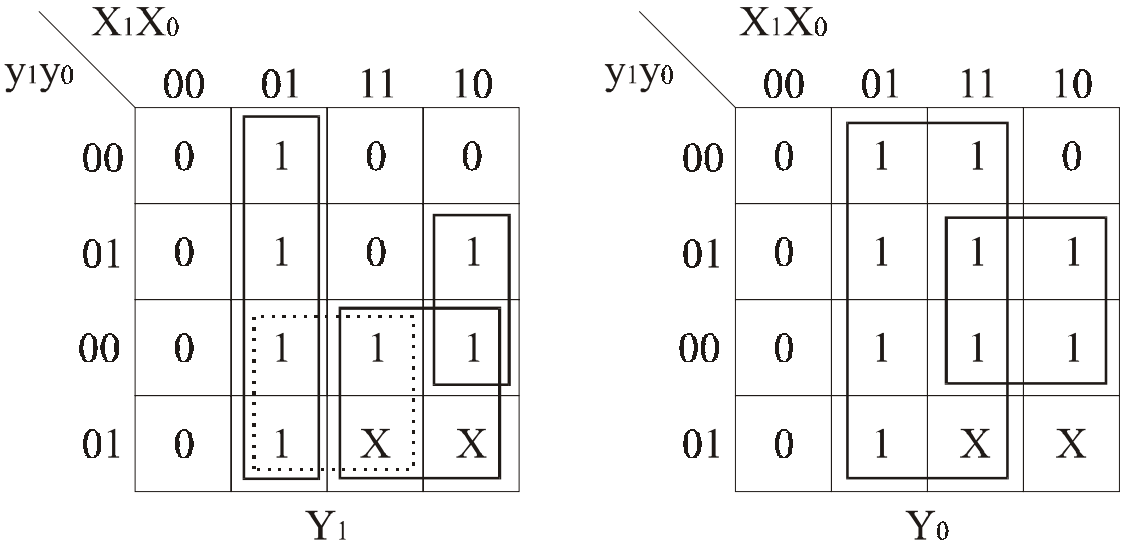


Tableau 6.2: Synthèse des variables d'états.

Ces deux tables permettent d'écrire:

$$Y_1 = \overline{X_1}X_0 + y_1X_1 + X_1\overline{X_0}y_0 + (X_0y_1)$$
$$Y_0 = X_0 + X_1y_0$$

Remarque:

Le terme entre parenthèses correspond à un impliquant redondant, montré en pointillé. S'il n'est pas essentiel, d'un point de vue pratique, il contribue à rendre le système prévisible. Lorsqu'on veut passer d'un état stable à un autre, on est certain d'éviter un comportement imprévisible si les changements l'états s'effectuent à l'intérieur d'un impliquant.

Étape 5: Dessin du schéma.

À partir des expressions qui décrivent les relations logiques entre les éléments du système, il est possible de dessiner le schéma de ce dernier.

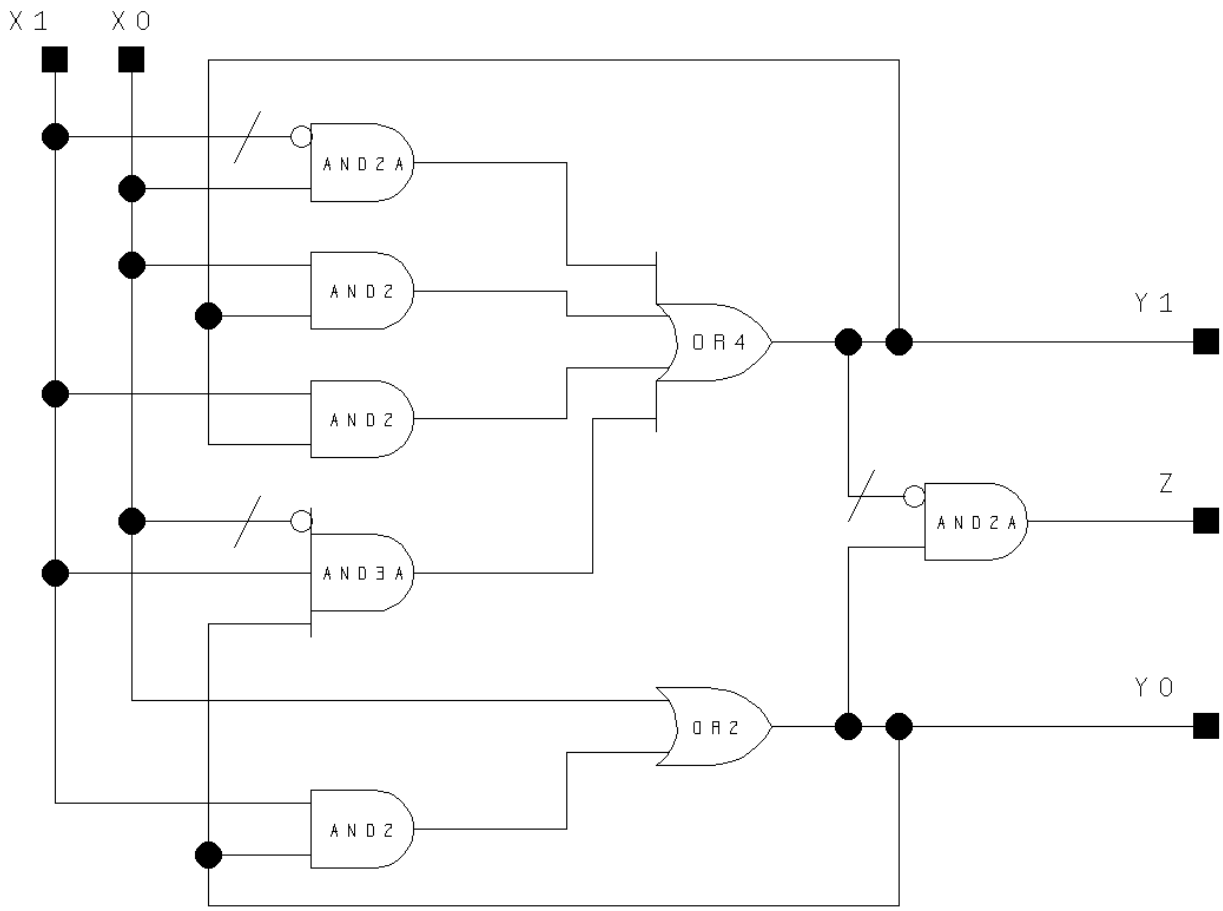


Figure 6.6: Schéma du circuit de l'exemple 6.4.

Remarques:

- Il faut toujours régler les courses et les erreurs momentanées pour rendre un système séquentiel utilisable.
- On peut les régler de diverse façon selon les possibilités:
  - En définissant un état intermédiaire instable.
  - Par une définition appropriée des états secondaires et des excitations non définies.
  - En retournant au diagramme de fusionnement.
  - En dernier recours, ajouter un état secondaire.
- Les impliquants redondants rendent un réseau plus prévisible dans certains cas.
- On peut permuter des lignes du tableau de séquences primaires fusionné lorsque celui-ci utilise encore des symboles.

6.3 Éléments de mémoire

Certains éléments de mémoire (les bascules et les bistables) ont été vus brièvement dans la section 2.5. Cette section explique avec plus de détails les éléments de mémoire les plus couramment utilisés pour les machines à états.

Bascule S/R (Set/Reset)

La bascule S/R est un élément de mémoire dont la sortie peut être établie à un état actif ou inactif en forçant des conditions temporaires sur les entrées S et R. L'entrée S sert à mettre la sortie à 1 et l'entrée R met la sortie à 0. Si aucune des entrées n'est active, la sortie conserve la valeur courante. De plus, il est illogique que les deux entrées soient actives en même temps. Ce cas est à éviter en tout temps lors que ce composant est utilisé comme bascule.

S	R	Q
0	0	Q_0
0	1	0
1	0	1
1	1	X

Synthèse:

	00	01	11	10	Q
1	①	2	-	3	0
2	1	②	-	-	0
3	4	2	-	③	1
4	④	2	-	3	1

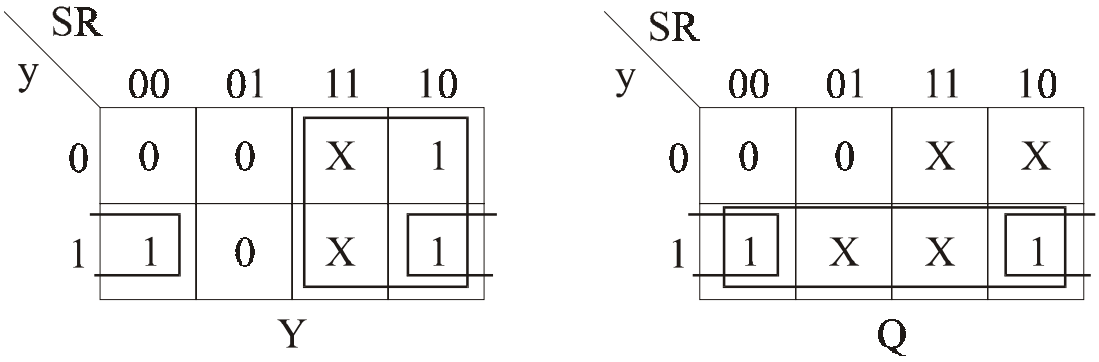


Tableau 6.3: Excitation et sortie d'une bascule S/R.

$$Y = S + y\overline{R} \qquad Q = y \text{ ou}$$

$$Q = y\overline{R}$$

La figure 6.7 présente le schéma du circuit dans sa forme directe et sa forme optimisée. La configuration de circuit de la forme optimisée, ainsi que l'équivalent où les NON-OU sont remplacés par des NON-ET, sont la fondation des circuits synchrones. Dans la version avec des NON-ET, il est important de savoir que les entrées R et S sont en logique négative.

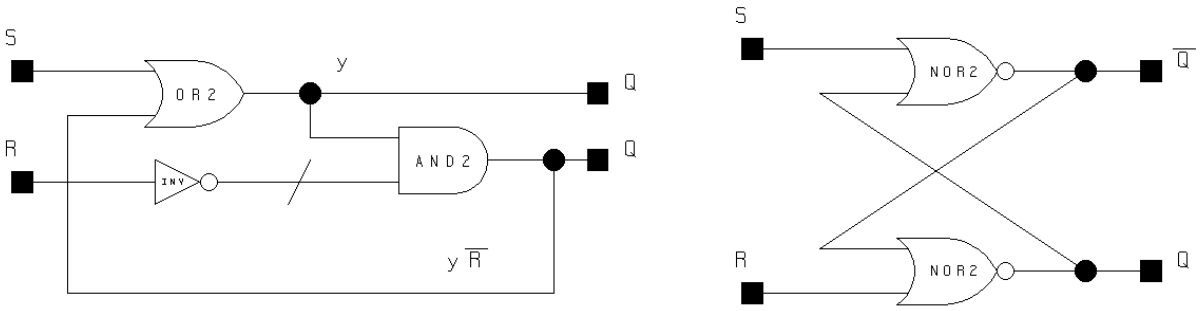


Figure 6.7: Formes directe et optimisée d'une bascule S/R.

Bascule S/R avec horloge.

Dans cette configuration, l'horloge est comme un signal d'activation. La bascule ne pourra changer de valeur que si l'horloge est à un niveau actif. Les signaux *PRESET* et *CLR* servent à mettre la sortie Q à 1 et à 0 respectivement, et ce, sans tenir compte de l'horloge. La table de vérité est la même que pour la bascule S/R.

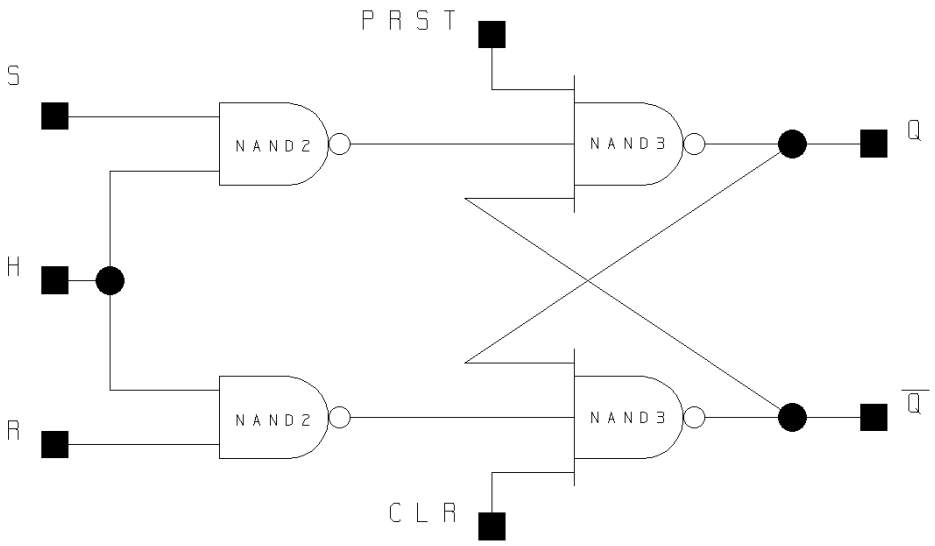


Figure 6.8: Schéma d'une bascule SR avec horloge et initialisation.

Bistable (*latch*)

La bistable est une mémoire où l'entrée est copiée à la sortie tant que l'horloge est active. Lorsque l'horloge devient inactive, la sortie conserve son état. L'horloge est active sur un niveau haut.

D	H	Q
0	0	Q_0
0	1	0
1	0	Q_0
1	1	1

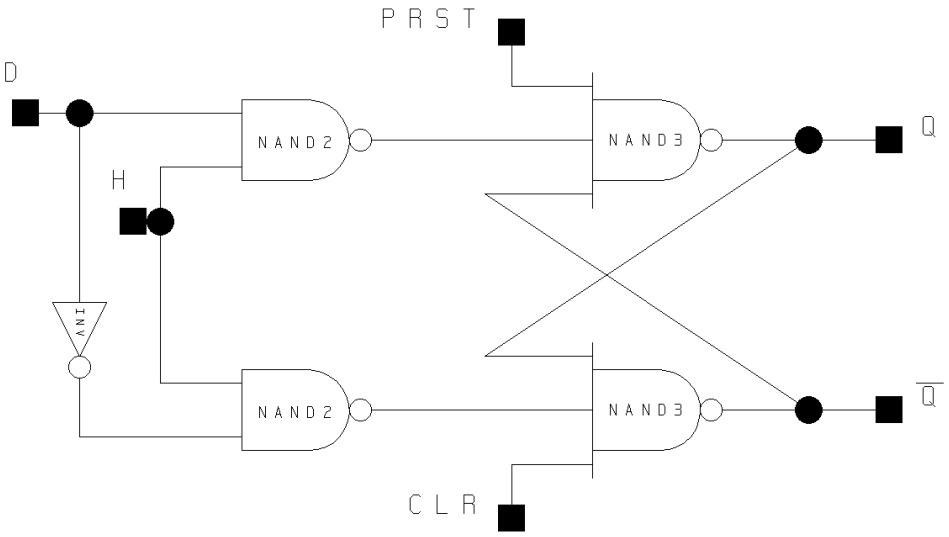


Figure 6.9: Schéma d'une bistable

Bascule D

La bascule D est la plus intuitive puisque son excitation est égale à son résultat. L'horloge est active sur une transition et l'entrée D est copiée à la sortie à ce moment.

D	Q
0	0
1	1

Bascule T

La bascule T (*toggle*) est un élément de mémoire n'ayant qu'une entrée T. Lors d'une transition de l'horloge, la sortie sera inversée si T est actif et conservera son état sinon.

T	Q
0	Q_0
1	$\overline{Q_0}$

Bascule JK

La bascule JK est la plus complexe mais aussi la plus versatile. Son comportement est commandé par deux signaux qui permettent de recréer le comportement de toutes les bascules vues jusqu'ici. Lors d'un design, l'utilisation des bascules JK demande un peu plus de travail pour le concepteur mais résulte habituellement en un nombre minimal de composants. Contrairement

aux autres éléments de mémoire jusqu'ici, l'horloge est active sur une transition de 0 vers 1. Certains composants utilisent une transition de 1 vers 0 mais aucun composant n'est actif sur les deux transitions.

J	K	Q
0	0	Q_0
0	1	0
1	0	1
1	1	$\overline{Q_0}$

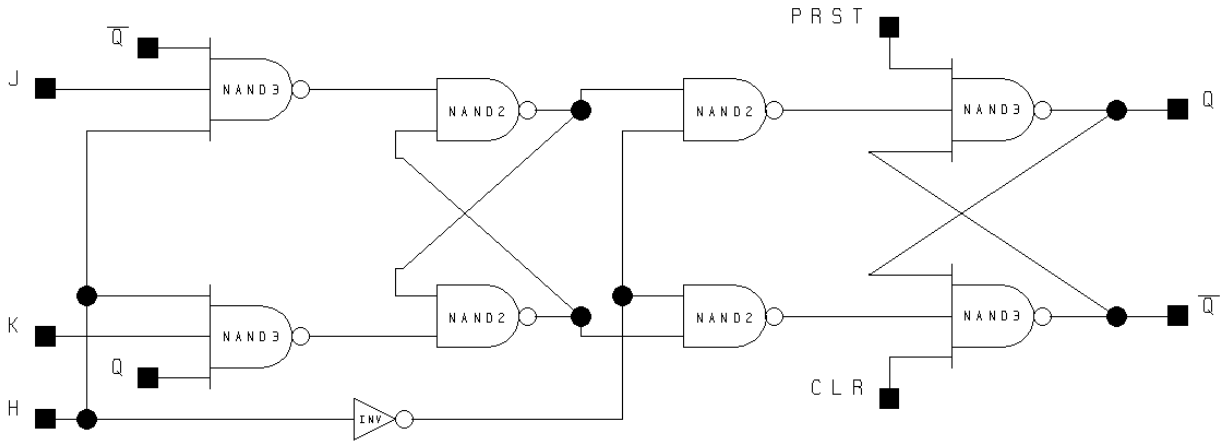


Figure 6.10: Schéma d'une bascule JK.

Transitions d'états

Lors de la conception de machines synchrones, il faut définir l'excitation nécessaire aux bascules pour provoquer le changement d'état désiré. L'information des tables de vérité des bascules D, T et JK peut aussi être écrite sous la forme d'une table de transitions.

État présent	État futur	Bascule D	Bascule T	Bascules JK	
Q	Q_0^+	D	T	J	K
0	0	0	0	0	X
0	1	1	1	1	X
1	0	0	1	X	1
1	1	1	0	X	0

Tableau 6.4: Table de transitions des différentes bascules.

6.4 Systèmes synchrones

Contrairement aux systèmes asynchrones qui peuvent changer d'état sur une transition d'une entrée quelconque, les systèmes synchrones ne changent d'état que sur l'activation de l'horloge. Ces systèmes utilisent habituellement des bascules D, JK et T. Les systèmes synchrones sont habituellement plus stables et prévisibles que les machines asynchrones et sont aussi plus faciles à réparer lorsque des fautes sont présentes. Comme pour les machines asynchrones, les machines synchrones sont divisées en deux catégories: les machines de Mealy et les machines de Moore. La figure 6.2 présente la forme générale de ces deux types de machines.

6.4.1 Synthèse

Les étapes de synthèse sont présentées et sont suivies d'un exemple.

1. À partir des spécifications, on dessine un diagramme de transitions d'états pour illustrer l'évolution du système pour toutes les séquences énumérées.
2. Ce diagramme permet de déterminer le nombre d'états du système et donc le nombre de variables d'états (relation 2^n).
3. On écrit la table des états suivants et des excitations. Cette table contient la liste des états présents (avec les entrées) en ordre binaire naturel croissant, la liste des états suivants et la liste des excitations.
4. Une table de Karnaugh est écrite pour chaque excitation en fonction des variables d'état (et d'entrée le cas échéant).
5. Les états facultatifs sont vérifiés pour éviter le verrouillage (deadlock).
6. On écrit l'expression logique de l'excitation correspondant à chaque mémoire d'état.
7. Le système est dessiné.

Exemple 6.5

Réaliser un compteur modulo 3.

Étape 1

Un tel circuit requiert deux bascules

Étape 2: Définir la séquence.

La séquence est définie de façon arbitraire. La séquence binaire naturelle est utilisée dans cet exemple.

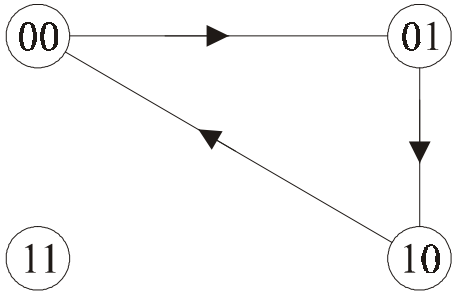


Figure 6.11: Diagramme d'états de l'exemple 6.5.

Dans ce diagramme d'états, l'état 11 ne fait pas partie de la séquence. Si le circuit ne possède pas de RAZ (remise à zéro), il est possible que le premier état soit 11, il faut donc prévoir un parcours entre cet état invalide et un état valide. Pour l'instant, l'état 11 est ignoré. Il faut cependant ajouter une étape supplémentaire de vérification lors de la simplification des équations d'excitation pour vérifier qu'il n'y a pas de verrouillage du circuit dans l'état 11.

Étape 3: Matrice d'excitation

Étape 4: Minimisation

Étape 5: Réalisation

État présent		État futur		Bascules D		Bascules T		Bascules JK			
Q_1	Q_0	Q_1^+	Q_0^+	D_1	D_0	T_1	T_0	J_1	K_1	J_0	K_0
0	0	0	1	0	1	0	1	0	X	1	X
0	1	1	0	1	0	1	1	1	X	X	1
1	0	0	0	0	0	1	0	X	1	0	X
1	1	X	X	X	X	X	X	X	X	X	X

Les équations suivantes peuvent être obtenues:

$$D_1 = Q_0$$
$$D_0 = \overline{Q}_1 \overline{Q}_0$$

$$T_1 = Q_1 + Q_0$$
$$T_0 = \overline{Q}_1$$

$$J_1 = Q_0$$
$$K_1 = 1$$
$$J_0 = Q_1$$
$$K_0 = 1$$

La réalisation avec des bascules JK ne demandera pas d'éléments logiques puisque les bascules ont toujours les sorties Q et son inverse de disponible. La réalisation avec des bascules D demandera un ET et la réalisation avec des bascules T demandera un OU.

6.4.2 Analyse

Les étapes d'analyse sont similaires à celles de la synthèse. Elles sont cependant exécutées dans l'ordre inverse. Les cinq étapes sont:

1. Par inspection du diagramme schématique d'un système existant, on écrit la fonction d'excitation de chaque bascule mémorisant une variable d'état.
2. On confectionne une table des états suivants et des excitations dans laquelle on trouve la liste binaire naturelle des états présents (et des entrées), des excitations et des états suivants.
3. Le diagramme des transitions d'états est dessiné pour illustrer le comportement détaillé du système et de statuer sur son caractère fonctionnel ou problématique.
4. Les états sont identifiés par des cercles contenant un nom, un code des variables d'état et le symbole des sorties actives à cet état encerclés pour les mettre en évidence.
5. Les états sont reliés par des arcs illustrant les transitions. Les valeurs des entrées produisant une transition sont écrites sur l'arc reliant l'état présent et l'état suivant.

Exemple 6.6:

Analyser le circuit suivant:

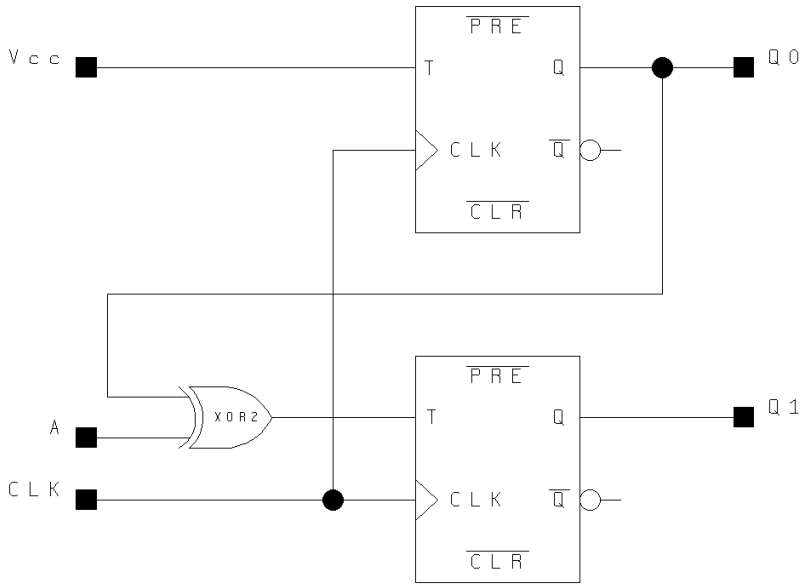


Figure 6.12: Schéma de l'exemple 6.6.

$T_0 = 1$

$T_1 = A \oplus Q_0$

État présent			Excitation		État futur	
A	Q_1	Q_0	T_1	T_0	Q_1^+	Q_0^+
0	0	0	0	1	0	1
0	0	1	1	1	1	0
0	1	0	0	1	1	1
0	1	1	1	1	0	0
1	0	0	1	1	1	1
1	0	1	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	1	1	0

Séquence de sortie:

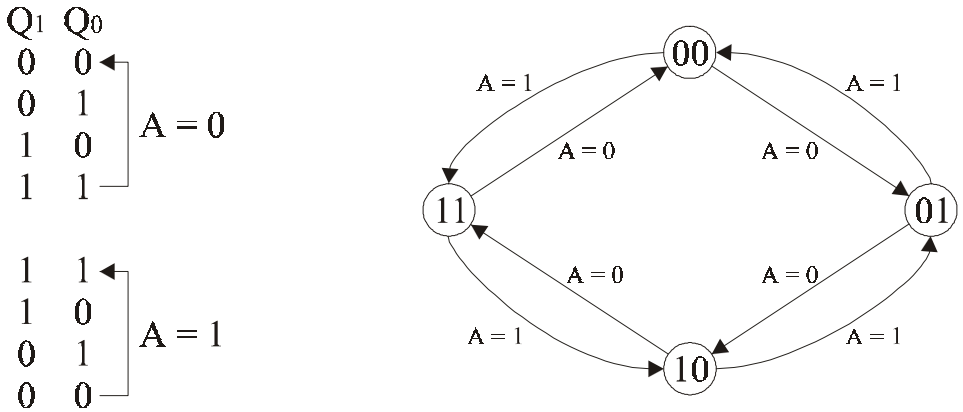


Figure 6.13: Séquence des états de l'exemple 6.6.

CHAPITRE 7

Unité de commande industrielle (ICU MC14500)

Ce chapitre se veut une introduction au MC14500 de Motorola. Ce module logique est un processeur à un bit qui dispose de 16 instructions. Ce module a été conçu pour remplacer de grands réseaux combinatoires par un composant programmé simple et fiable. Les informations complémentaires au sujet de ce composant peuvent être trouvées en annexe.

7.1 Description

Le 14500 est un module logique de 16 broches conçu pour remplacer des circuits contenant un grand nombre de modules logiques standards. Un système basé sur le 14500 utilise une logique programmée, c'est à dire que la fonction à exécuter se trouve dans une mémoire et le processeur exécute les instructions de façon séquentielle. L'ICU est capable d'exécuter 16 instructions différentes, encodées en mots d'instructions de 4 bits. Les instructions sont des fonctions logiques s'exécutant entre une donnée externe et un registre interne. Le résultat est mémorisé dans le même registre interne effaçant la donnée précédente. Il est à noter que la ligne de donnée est bidirectionnelle, c'est à dire qu'elle sert tout aussi bien à lire les données qu'à écrire les résultats. Cela implique l'utilisation de composants à haute impédance pour communiquer avec le 14500. Un des avantages principaux d'un système basé sur ce contrôleur est de pouvoir modifier complètement le comportement d'un circuit en changeant simplement le contenu de la mémoire qui contient le programme. Aucune modification physique du circuit n'est nécessaire.

Un système basé sur le 14500 doit contenir un nombre minimal de composants. Ces composants sont:

- L'ICU, module logique qui exécute les instructions.
- Une mémoire qui contient le programme destiné à l'ICU. Cette mémoire est habituellement de type ROM mais une RAM peut aussi être utilisée.
- Le compteur de programme qui se charge de mémoriser l'adresse de l'instruction courante et qui fournira l'adresse de l'instruction suivante lorsque l'instruction courante sera terminée.

- Entrées/sorties, modules logiques permettant de *multiplexer* les entrées et *démultiplexer* les sorties.

La figure 7.1 présente le schéma bloc d'un circuit minimal basé sur le 14500.

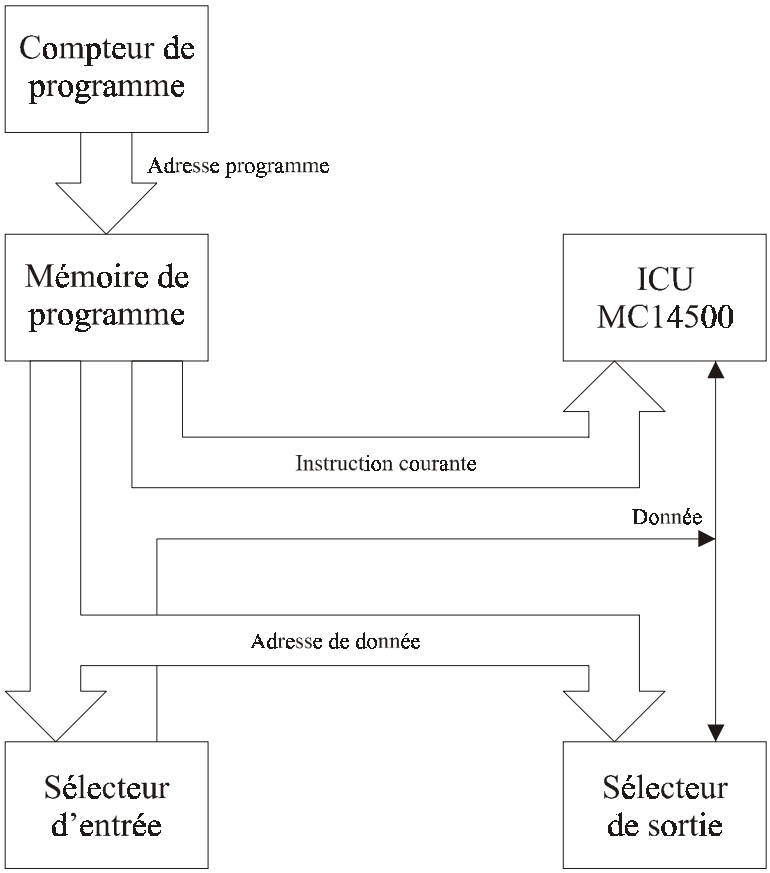


Figure 7.1: Schéma bloc d'un système basé sur le 14500.

7.2 Constituants

Le 14500 peut être plus facile à comprendre en regardant à l'intérieur (figure 7.2). Les divers composants les plus importants sont expliqués ici:

- L'accumulateur RR: Mémorise le résultat des opérations effectuées par le processeur. La broche 15 du processeur permet d'observer le contenu de RR.
- L'unité logique: Effectue les opérations logiques sous la commande de la logique de contrôle qui décode l'instruction contenue dans le registre d'instruction.
- Le registre d'instruction: Mémorise ce que le processeur reçoit comme instruction sur les broches 4 à 7 sur la descente de l'horloge.

- L'Horloge: Oscillateur interne dont la sortie apparaît à la broche 14 et dont la fréquence est réglée en connectant une résistance entre les broches 13 et 14, ou en forçant une horloge sur la broche 13.
- La ligne de donnée: Communication bidirectionnelle qui permet d'échanger des données selon l'état des registres IEN et OEN.
- Sortie Write: Une impulsion apparaît à la broche 2 et signifie un transfert de RR vers un port de sortie lorsque permis pas OEN.
- Les *flags*: Sorties sur les broches 9 à 12 qui devienne actives sur la descente de l'horloge lors du passage d'une instruction JMP, RTN, NOP0 et NOPF.
- Les registre IEN et OEN: Registres internes activés par programmation qui autorisent les entrées/sorties.
- L'entrée RESET: Remet à zéro les registres internes (RR, IEN et OEN) et maintient l'horloge haute.

Le cœur du système est la boucle formée par l'unité logique et le registre de résultat. Lorsqu'une instruction est reçue, le registre d'instruction la mémorise. La logique de contrôle décode l'instruction et active les signaux internes pour effectuer correctement l'instruction. Si l'instruction courante est une opération logique, c'est la logique de contrôle qui se charge de configurer l'unité logique pour que la bonne opération soit effectuée. À la fin du cycle d'instruction, le résultat est mémorisé dans RR.

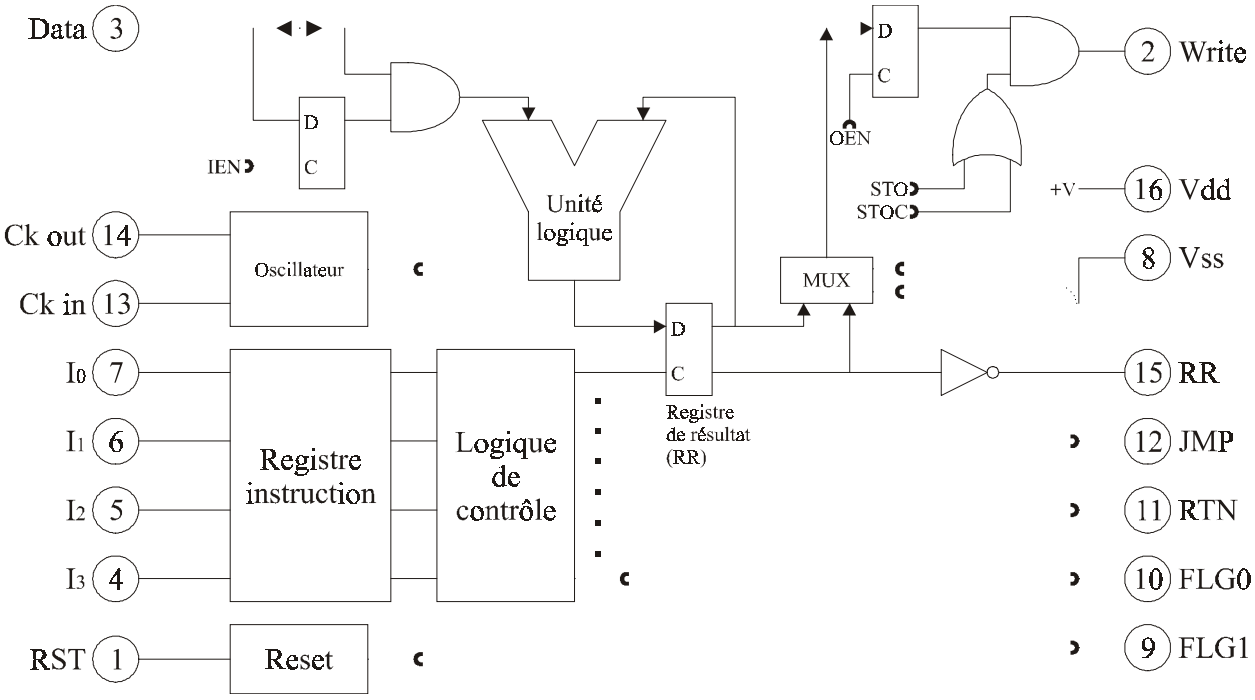


Figure 7.2: Vue interne du 14500.

7.3 Les instructions

Le 14500 dispose de 16 instructions encodées sur 4 bits. Les instructions ainsi qu'une brève explication pour chacune sont données selon la catégorie.

Instructions qui ne font rien.

Code d'instruction	Mnémonique	Description
0	NOP0	RR = RR. Une impulsion est envoyée sur FLG0.
F (15)	NOPF	RR = RR. Une impulsion est envoyée sur FLGF.

Instructions entrées/sorties.

Code d'instruction	Mnémonique	Description
1	LD	Chargement (*LOAD*). RR = Data.
2	LDC	Chargement (*LOAD*). RR = $\overline{\text{Data}}$.
8	STO	Sauvegarde. Data = RR et impulsion sur Write.
9	STOC	Sauvegarde. Data = $\overline{\text{RR}}$ et impulsion sur Write.

Fonctions logiques.

Code d'instruction	Mnémonique	Description
3	AND	ET logique. RR = RR · Data
4	ANDC	ET logique. RR = RR · $\overline{\text{Data}}$
5	OR	OU logique. RR = RR + Data
6	ORC	OU logique. RR = RR + $\overline{\text{Data}}$
7	XNOR	XNOR. Si RR = Data, RR devient 1.

Instructions des registre IEN et OEN.

Code d'instruction	Mnémonique	Description
A (10)	IEN	Activation de l'entrée. Registre IEN = Data.
B (11)	OEN	Activation de la sortie. Registre OEN = Data.

Instructions de sous-routines.

Code d'instruction	Mnémonique	Description
C (12)	JMP	Saut de programme. Impulsion sur JMP.
D (13)	RTN	Retour de sous-routine. Impulsion sur RTN.
E (14)	SKZ	Saut de la prochaine instruction si RR = 0.

Remarques:

- Lorsque IEN est à 0, toutes les entrées sont considérées comme des 0. Si l'instruction LDC est effectuée quand IEN est à 0, ce sera un 1 qui sera lu, comme si l'entrée était effectivement 0.
- Lorsque OEN est à 0, l'impulsion sur Write associée aux écritures sera inhibée, prévenant ainsi l'écriture.
- Les instructions de sous-routines requièrent de la logique externe. Pour effectuer un saut, l'impulsion à la sortie JMP doit faire sauter le compteur de programme vers une adresse où se trouve la sous routine. Il existe des compteurs auxquels il est possible de forcer une sortie au moyen d'une entrée LOAD.
- L'instruction RTN génère une impulsion pour ramener le compteur de programme au point d'où l'instruction JMP avait été exécutée. Ceci doit s'effectuer au moyen de la

logique externe. L'instruction SKZ existe pour éviter le branchement à la sous-routine selon les conditions déterminées par le système.

7.4 Limitations

Même si un système basé sur le 14500 est beaucoup plus flexible qu'un grand réseau de logique combinatoire, il y a des limitations à l'utilisation de l'ICU.

- Le processeur ne peut être utilisé par lui-même. Il doit être associé à divers modules logiques tels que le compteur, la mémoire et les ports d'entrées/sorties.
- L'ICU n'est pas destiné aux calculs. Il remplace de grands réseaux de logique combinatoire.
- La rapidité d'exécution est fortement réduite puisque les opérations logiques sont effectuées en séquence.

7.5 Écriture de programmes

Pour transformer un schéma en programme, il faut transformer le traitement en opérations séquentielles. Les exemples suivants utilisent le un circuit basé sur le schéma bloc de la figure 7.1. La figure 7.3 présente les ports d'entrées/sorties en détail. Notez que la sortie TMP et l'entrée TMP sont branchées ensemble pour permettre de mémoriser un résultat temporaire.

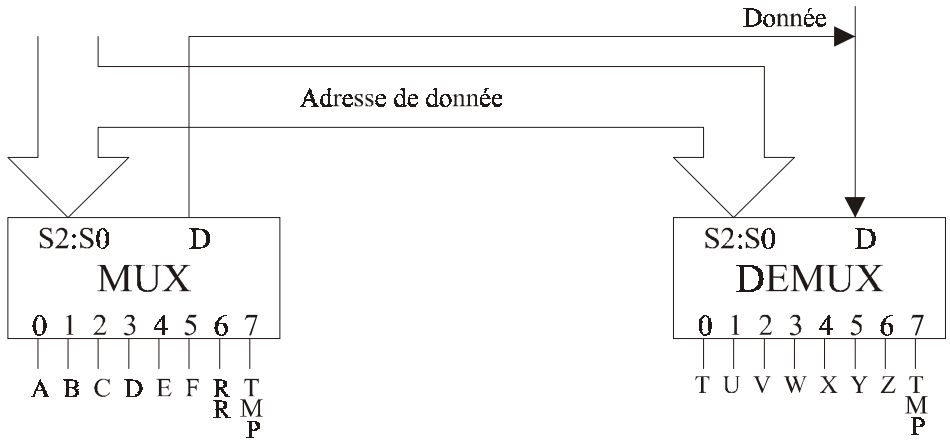
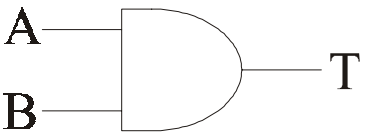


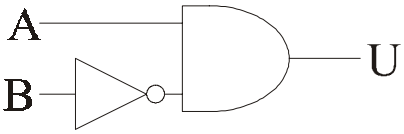
Figure 7.3: Entrées/sorties du système utilisé pour les exemples.

Exemple 7.1

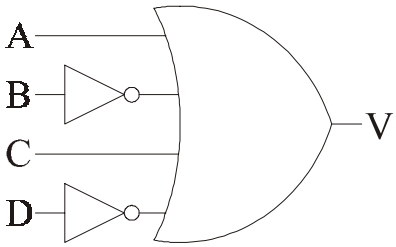
Écrire le programme réalisant les fonctions suivantes:



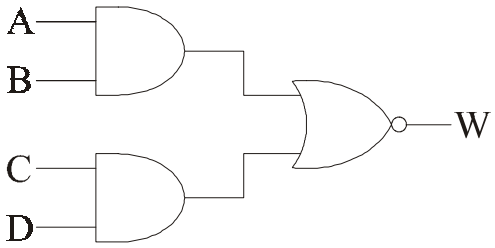

```
LD A
AND B
STO T
```




```
LD A
ANDC B
STO U
```




```
LD A
ANDC B
AND C
ANDC D
STO V
```




```
LD A
AND B
STO TMP
LD C
AND D
OR TMP
STOC W
```

Remarques:

- Lors d'une réinitialisation, le contenu des registres RR, IEN et OEN devient zéro. Cela signifie qu'aucune entrée ni sortie peut se produire. Il faut initialiser ces registres avant de pouvoir utiliser des données externes en les mettant à 1. Pour ce faire, l'instruction ORC RR assure la présence d'un 1 dans RR. Il faut ensuite mettre ce 1 dans les registres IEN et OEN en effectuant les instructions IEN RR et OEN RR.
- Dans leur forme actuelle, ces programmes ne s'exécuteront qu'une seule fois. Pour que les programmes s'exécutent en boucle continue, il faut trouver une façon simple de dire au compteur de programme de repartir au début. Cela peut se faire en branchant la broche FLGF à la broche RESET et à la réinitialisation du compteur. La

dernière étape consiste à ajouter l'instruction NOPF à la fin de chacun de ces programmes. Lorsque cette instruction sera exécutée, elle initialise l'ICU et le compteur qui repart au début du programme.

- Pour effectuer une inversion de RR, il faut exécuter l'instruction LDC RR.

CHAPITRE 8

Initiation à la logique programmable

Pour optimiser des circuits logiques, les concepteurs utilisent souvent la programmation. Il existe deux types de programmation: la logique programmée (logicielle) et les composants programmables (matérielle). Les processeurs et contrôleurs comme le 14500 utilisent une logique programmée car le composant exécute une série d'instructions en séquence. Le comportement physique du composant est toujours identique. Par exemple, la broche 3 du 14500 sert et servira toujours à transmettre les données et les broches 4 à 7 serviront toujours à spécifier l'instruction. De plus, tous les 14500 de même famille et technologie sont identiques et peuvent être interchangés.

Les modules électroniques de logiques programmables ont des comportements physiques qui changent en fonction de la programmation qu'ils ont reçu. La programmation se fait avec un appareil spécifique qui est capable de configurer l'intérieur du composant. Les modules de logique programmable peuvent être configurés pour exécuter directement la fonction logique désirée. Lors d'une programmation, le concepteur décide quelles broches serviront d'entrées et de sorties, si le module le permet, mais aussi quelle fonction logique est réalisée à chacune des sorties. Ce chapitre se veut une introduction à ces composants.

8.1 Intérêts de la logique programmable

Si un circuit est réalisé en logique programmable, il utilisera moins de composants puisque ceux-ci réalisent directement les fonctions logiques désirées. Un circuit ayant moins de composants sera habituellement moins coûteux à concevoir, réaliser et distribuer. La réduction du nombre de composants électronique tend aussi à augmenter la fiabilité des circuits et à réduire la consommation énergétique.

Exemple

Un FPLA à 28 broches peut contenir une structure ET-OU ayant les caractéristiques suivantes:

- 48 portes ET à 16 entrées;
- 8 portes OU à 48 entrées;
- 8 fonctions logiques de 16 variables avec un maximum de 48 termes par fonction.

Le désavantage majeur est qu'il faut se procurer le matériel nécessaire pour programmer les modules programmables. Puisque chaque compagnie fabrique des composants programmables différents, le changement de composant, de compagnie ou de technologie peut être coûteux. Ces désavantages affectent plus particulièrement le développement de nouveaux produits est ces coûts deviennent insignifiants lors de la production à grande échelle.

8.2 Familles de logique programmable

La liste des différentes familles de composants programmables qui suit n'est pas exhaustive. Elle se veut une introduction aux différentes architectures de ces composants.

PLE (*Programmable Logic Element*): Plan ET fixe, plan OU programmable.

PAL (*Programmable Array Logic*): Plan ET programmable, plan OU fixe.

PLA (*Programmable Logic Array*): Plan ET programmable, plan OU programmable.

FPLA (*Field PLA*): PLA reprogrammable.

PEEL (*Programmable Electrically Erasable Logic*)

- EEPROM
- EEPLA
- EEPAL (GAL: *Generic Array Logic*)

Les différents plans sont illustrés à la figure 8.1.

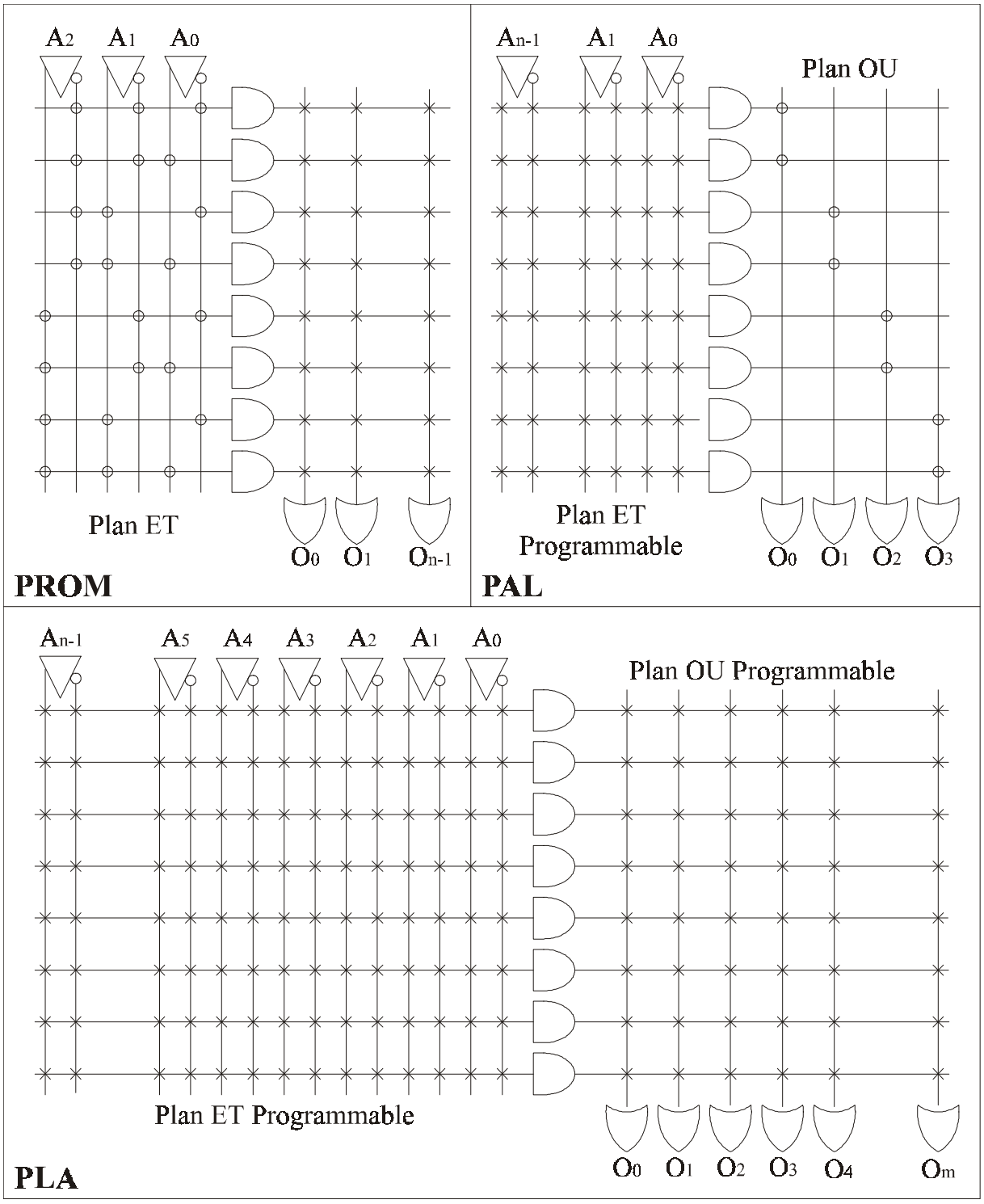


Figure 8.1: Architectures de composants programmables.

Exemple 8.1:

Réaliser la fonction $F = AB + \overline{A}C + BC$ avec un PROM, un PAL et un PLA tels que ceux montrés à la figure 8.1.

En prenant comme hypothèse A = A_0, B = A_1, C = A_2 et F = O_0. Pour le PROM, la fonction doit être sous forme canonique donc:

$$F = AB\overline{C} + ABC + \overline{A}BC + \overline{A}\overline{B}C$$

Pour le PAL, la fonction ne peut être une sommation que de deux produits. En simplifiant, on obtient $F = AB + \overline{A}C$

Pour le PLA, la fonction peut être réalisée directement. La figure 8.2 montre les résultats.

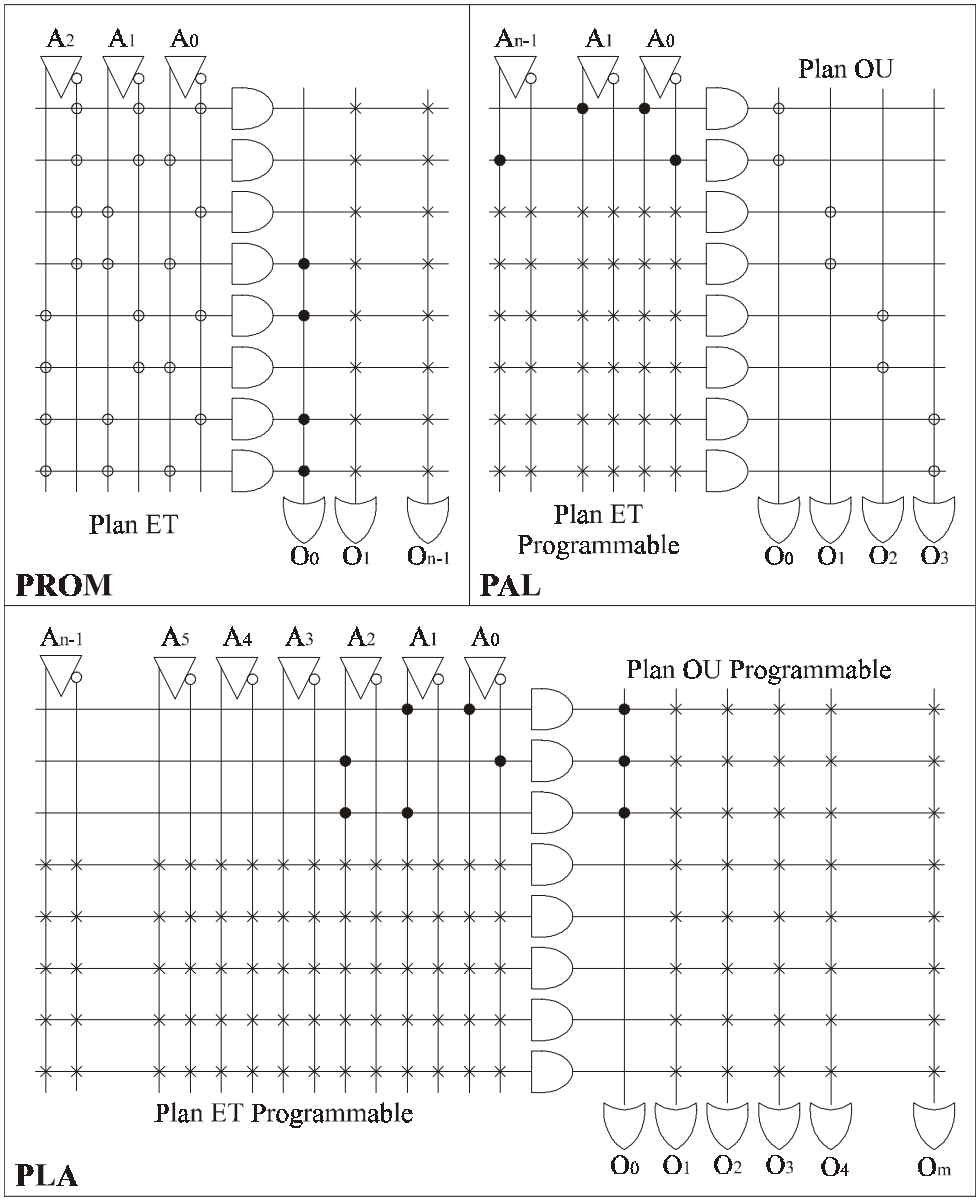


Figure 8.2: Solutions de l'exemple 8.1.

ANNEXE 1

Exercices

Et

Solutions

NOTE: Ces exercices sont complémentaires à ceux que vous pouvez trouver sur le site WEB du cours de 3.313 qui sont en bonne partie solutionnés. Il est essentiel à votre formation de les télécharger et de les compléter.

Chapitre 2: Logique combinatoire

Algèbre de Boole

1- Réduire les fonctions suivantes:

$$F1 = (A\overline{B} + C)(A + \overline{B})C$$

$$F2 = C + AB + AD(B + \overline{C}) + CD$$

$$F3 = A\overline{B} + A\overline{C} + \overline{B}\,\overline{C}$$

$$F4 = A + ABC + \overline{A}C$$

$$F5 = (\overline{X} + \overline{Y})(\overline{X} + \overline{Z})$$

2- Démontrer:

a) $A + AB = A$

b) $A + \overline{A}B = A + B$

c) $AB + A\overline{B} = A$

d) $AC + \overline{A}BC = AC + BC$

e) $AB + AC + \overline{B}C = AB + \overline{B}C$

f) $A\overline{C} + \overline{A}B + BC = A\overline{C} + B$

3- Simplifier:

$$F = \overline{\overline{(\overline{A} + B)} + \overline{(\overline{A} + B)}} + \overline{(\overline{A}\overline{B})(A\overline{B})}$$

SOLUTIONS:

1- $F1 = (A\overline{B} + AC + \overline{B}C)C$

$F1 = A\overline{B}\,C + AC + \overline{B}C$

$F1 = AC(1 + \overline{B}) + \overline{B}C$

$\underline{F1 = AC + \overline{B}C}$

$F2 = C + AB + AD(B + \overline{C}) + CD$

$F2 = C + AB + ABD + A\overline{C}D + CD$

$F2 = C(1 + D) + AB(1 + D) + A\overline{C}D$

$F2 = C(1 + AD) + AB + A\overline{C}D$

$F2 = C + AB + AD(C + \overline{C})$

$\underline{F2 = C + AB + AD}$

$\underline{F3 = A\overline{B} + A\overline{C} + \overline{B}\,\overline{C}}$

$F4 = A(1 + BC) + \overline{A}C$

$F4 = A(1 + C) + \overline{A}C$

$F4 = A + C(A + \overline{A})$

$\underline{F4 = A + C}$

$F5 = (\overline{X} + \overline{Y})(\overline{X} + \overline{Z})$

$F5 = \overline{X} + \overline{X}\,\overline{Z} + \overline{X}\,\overline{Y} + \overline{Y}\,\overline{Z}$

$F5 = \overline{X}(1 + \overline{Z} + \overline{Y}) + \overline{Y}\,\overline{Z}$

$\underline{F5 = \overline{X} + \overline{Y}\,\overline{Z}}$

2- a) $A(1+B) = A$

$A = A$

b) $A(1+B) + \overline{A}B = A + B$

$A + B(A + \overline{A}) = A + B$

$A + B = A + B$

c) $A(B + \overline{B}) = A$

$A = A$

d) $AC(1+B) + \overline{A}BC = AC + BC$

$AC + BC(A + \overline{A}) = AC + BC$

$AC + BC = AC + BC$

e) $AB + AC(B + \overline{B}) + \overline{B}C = AB + \overline{B}C$

$AB(1+C) + \overline{B}C(1+A) = AB + \overline{B}C$

$AB + \overline{B}C = AB + \overline{B}C$

f) $A\overline{C}(1+B) + \overline{A}B + BC = A\overline{C} + B$

$A\overline{C} + B(A\overline{C} + \overline{A} + C) = A\overline{C} + B$

$A\overline{C} + B\left[A\overline{C} + \overline{A}(C + \overline{C}) + C(A + \overline{A})\right] = A\overline{C} + B$

$A\overline{C} + B\left[\overline{A}C + \overline{A}\,\overline{C} + AC + A\overline{C}\right] = A\overline{C} + B$

$A\overline{C} + B\left[\overline{A}(C + \overline{C}) + A(C + \overline{C})\right] = A\overline{C} + B$

$A\overline{C} + B\left[\overline{A} + A\right] = A\overline{C} + B$

$A\overline{C} + B = A\overline{C} + B$

3- $F = \overline{\overline{A}\overline{B}} + \overline{\overline{A}\overline{B}} + \overline{\overline{A}B} + \overline{A\overline{B}}$

$F = \overline{A\overline{B}} + \overline{A}B + A\overline{B}$

$F = \overline{A} + B + \overline{A}B + A\overline{B}$

$F = \overline{A}(B + \overline{B}) + B(A + \overline{A}) + \overline{A}B + A\overline{B}$

$F = \overline{A}B + A\overline{B} + AB + \overline{A}B + \overline{A}B + A\overline{B}$

$F = A(B + \overline{B}) + \overline{A}(B + \overline{B})$

$F = 1$

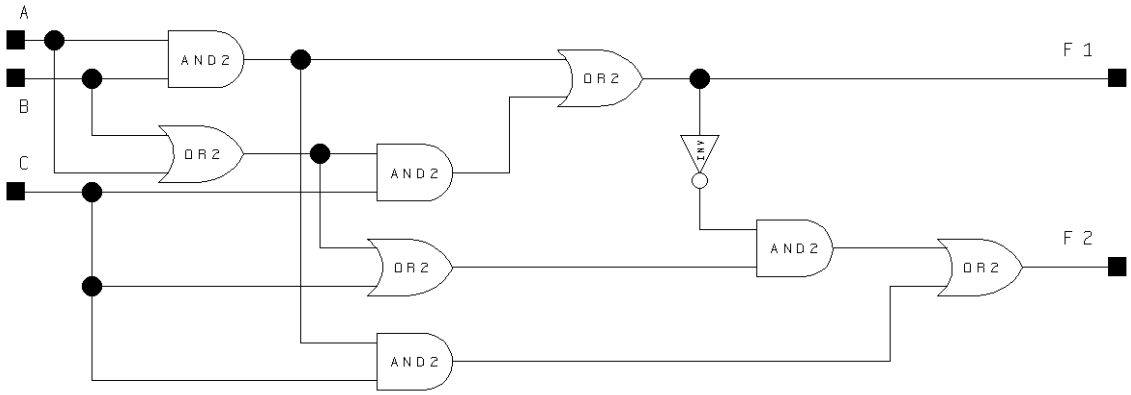
Karnaugh

1- Soit la fonction $F = AB\overline{C}D + AC\overline{D} + \overline{A}\overline{B}C + \overline{A}BD + \overline{A}B\overline{C} + A\overline{B}\overline{D}$

a) Écrire l'équation sous forme canonique.

b) Simplifier avec une table de Karnaugh.

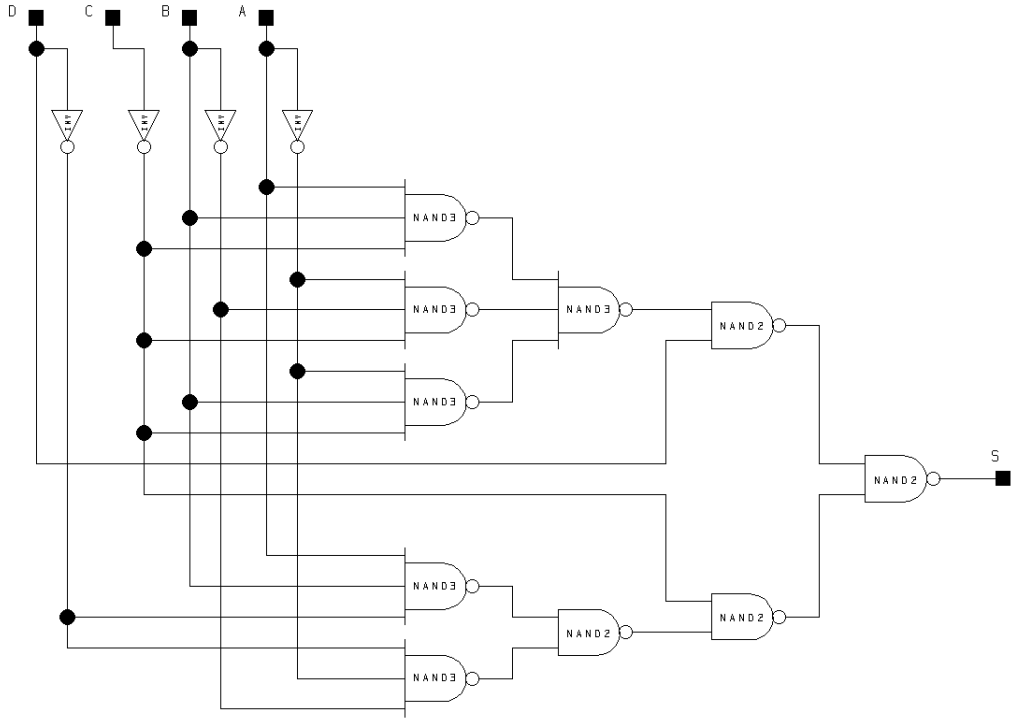
2- Soit le circuit suivant:



a) Écrire les équations des sorties.

b) Réduire les fonctions et dessiner le nouveau schéma.

3- Soit le schéma suivant:



a) Écrire la fonction logique sous forme de somme de produits.

b) Simplifier la fonction avec une table de Karnaugh sachant que ABCD ne prennent jamais la valeur 1001.

c) Dessiner le schéma de la fonction S.

4- Faire la synthèse d'un circuit combinatoire à 4 entrées et 3 sorties. Les 4 entrées forment un chiffre décimal codé binaire avec les poids D=8, C=4, B=2 et A=1. Les sorties sont:

E=1 si le code DCBA n'est pas un nombre BCD valide (nombre > 9).

F=1 si le nombre est un multiple de 3.

G=1 si le nombre est plus grand que 5.

Les sorties F et G sont sans importance si E=1.

a) Dessiner la ou les tables de Karnaugh.

b) Écrire les équations simplifiées.

SOLUTIONS

1- a) $F = AB\overline{C}D + ABC\overline{D} + A\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD +$
$\overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + A\overline{B}C\overline{D} + A\overline{B}\overline{C}D$

b)

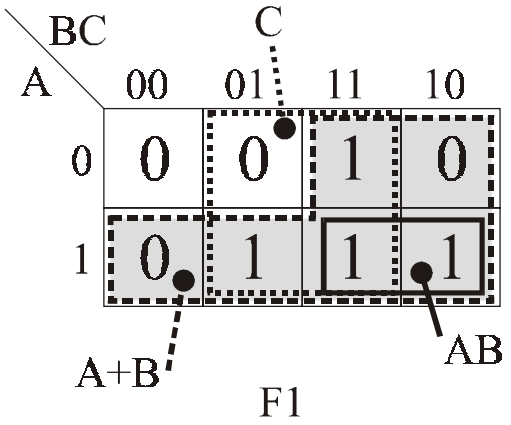
AB \ CD	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	1	0	1
10	1	0	0	1

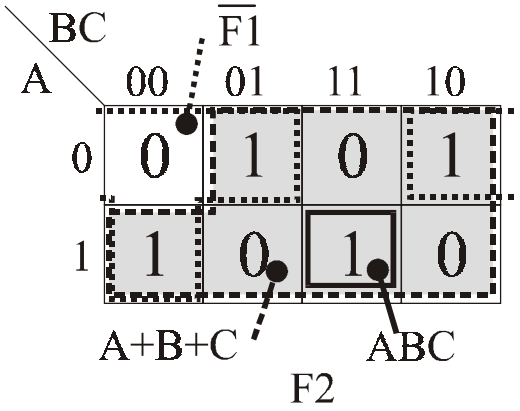
$F = \overline{A}\overline{B} + A\overline{B}CD + AB\overline{C}D + AC\overline{D}$

2- a) $F1 = AB + (A + B)C$

$F2 = \overline{F1}(A + B + C) + ABC$

b)





$F1 = AB + AC + BC$

$F2 = A\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + ABC = A \oplus B \oplus C$

3- a) $S = \overline{\overline{AB\overline{C}}\ \overline{ABC}\ \overline{AB\overline{C}}\ D\ \overline{ABD}\ \overline{ABD}\ \overline{C}}$

$S = \overline{\overline{AB\overline{C}}\ \overline{ABC}\ \overline{AB\overline{C}}}\ D + \overline{\overline{ABD}\ \overline{ABD}}\ \overline{C}$

$S = (AB\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C})\ D + (ABD + \overline{A}\overline{B}D)\ \overline{C}$

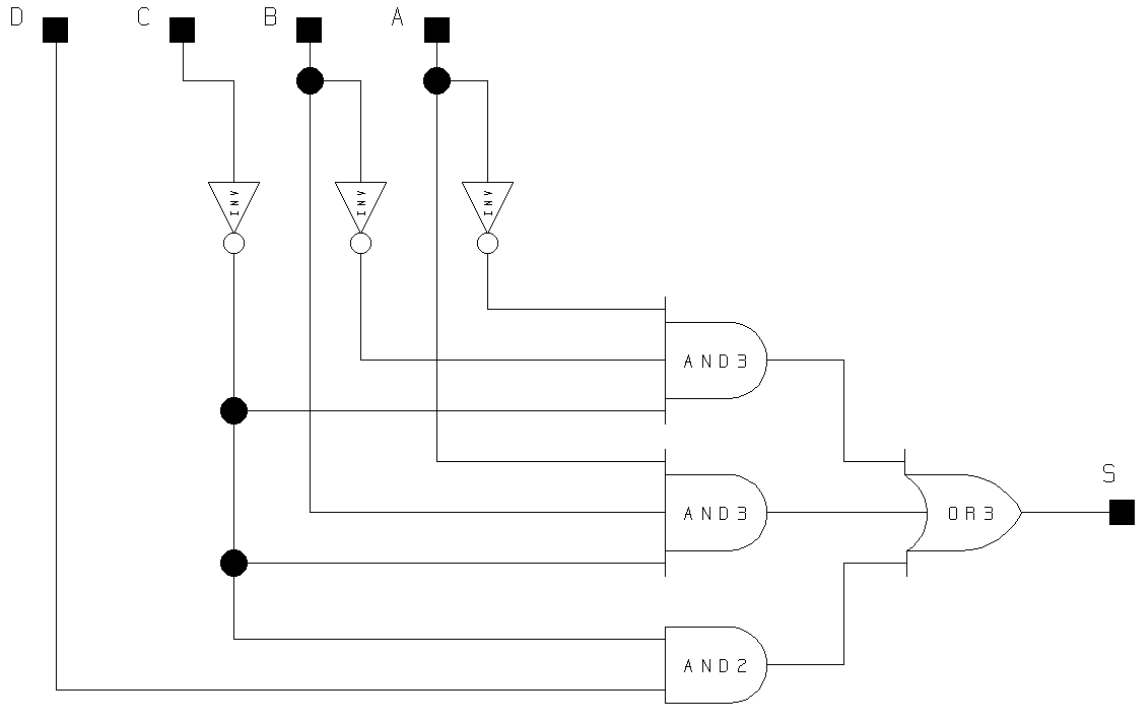
$S = AB\overline{C}D + \overline{A}\overline{B}CD + \overline{A}B\overline{C}D + AB\overline{C}D + \overline{A}C\overline{B}D$

b)

AB \ CD	00	01	11	10
00	1	1	0	0
01	0	1	0	0
11	1	1	0	0
10	0	x	0	0

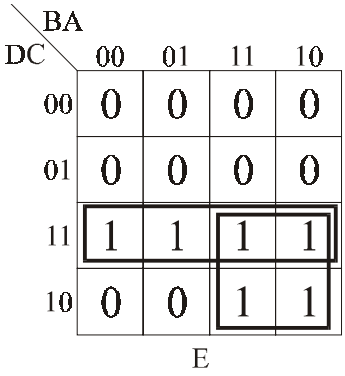
$$S = \overline{A}\,\overline{B}\overline{C} + AB\overline{C} + \overline{C}D$$

c)

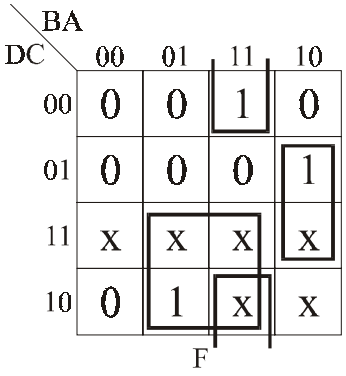


4- a)

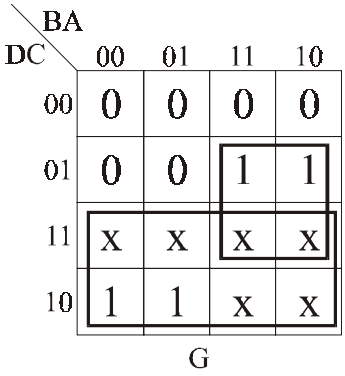
D	C	B	A	E	F	G
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	1	1
0	1	1	1	0	0	1
1	0	0	0	0	0	1
1	0	0	1	0	1	1
1	0	1	0	1	X	X
1	0	1	1	1	X	X
1	1	0	0	1	X	X
1	1	0	1	1	X	X
1	1	1	0	1	X	X
1	1	1	1	1	X	X



E



F



G

b)

$$E = DC + DB$$

$$F = DA + CB\overline{A} + \overline{C}BA$$

$$G = D + BC$$

Quine/McCluskey

1- Simplifier l'équation suivante à l'aide de l'algorithme de Quine/McCluskey

$$F = AD + \overline{B}A\overline{D} + \overline{A}\overline{D}(B \oplus C)$$

2- Soit la fonction $F = \overline{A}\overline{B}\overline{C} + \overline{A}BD + ABC\overline{D} + A\overline{B}C$

Sachant que les combinaisons $\overline{A}\overline{B}CD, ABCD$ et $A\overline{B}\overline{C}$ ne seront jamais présenter aux entrées, faire la synthèse pas Quine/McCluskey.

3- Réaliser un circuit permettant de multiplier deux nombre binaires AB et CD de 2 bits. Le résultat de la multiplication $P_2P_1P_0$ est un nombre de 3 bits. Le nombre AB ne peut jamais dépasser 2.

a) Faire la synthèse par Karnaugh de P_2, P_1 et P_0

b) Refaire la synthèse de P_0 par Quine/McCluskey.

SOLUTIONS:

1-

0100*	X010*	10XX*
0010	10X0	------
1000	100X	1XX1*
------	------	
1010	101X	
1001	1X01	
------	10X1	
1101	------	
1011	11X1	
------	1X11	
1111		

	0100	0010	1000	1010	1001	1101	1011	1111
0100	[✓]							
X010		[✓]		(✓)				
10XX			[✓]	(✓)	(✓)		(✓)	
1XX1					(✓)	(✓)	(✓)	[✓]

Les impliquants sont tous essentiels.

$$F = AD + A\overline{B} + \overline{B}C\overline{D} + \overline{A}BC\overline{D}$$

2-

$$F = \overline{A}\overline{B}C(D + \overline{D}) + \overline{A}B(C + \overline{C})D + ABC\overline{D} + A\overline{B}C(D + \overline{D}) + \phi\,(\overline{A}\overline{B}CD + ABCD + A\overline{B}\overline{C}(D + \overline{D})$$

$F=0000+0001+0111+0101+1110+1011+1010+\varnothing(0011+1111+1001+1000)$

0000	000X	0XX0*
------	X000	------
0001	------	0XX1*
1000	0X01	X0X1*
------	00X1	10XX*
0101	X001	------
1010	10X0	XX11*
0011	100X	1X1X*
1001	------	
------	01X1	
0111	1X10	
1110	101X	
1011	0X11	
------	X011	
1111	10X1	

	X111	
	111X	
	1X11	

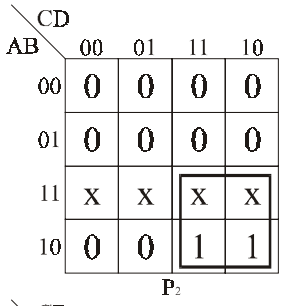
	0000	0001	0111	0101	1110	1011	1010
X00X	[✓]	(✓)					
0XX1		(✓)	(✓)	[✓]			
X0X1		✓				✓	
10XX						✓	✓
XX11			✓			✓	
1X1X					[✓]	(✓)	(✓)

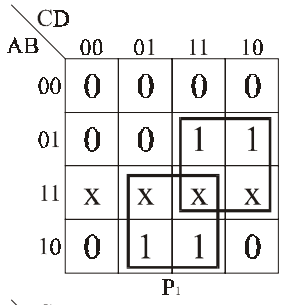
Les impliquants essentiels sont X00X, 0XX1 et 1X1X.

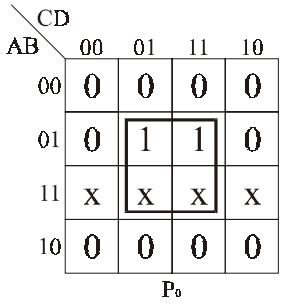
$F = \overline{A}D + \overline{B}\overline{C} + AC$

3-

A	B	C	D	P2	P1	P0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	X	X	X
1	1	0	1	X	X	X
1	1	1	0	X	X	X
1	1	1	1	X	X	X







$P_2 = AC$

$P_1 = AD + BC$

$P_0 = BD$

Par Quine/McCluskey:

P_0=0101+0111+∅(1100+1101+1110+1111)

0101	01X1	X1X1*
1100	X101	11XX*
------	110X	
0111	11X0	
1101	------	
1110	X111	
------	11X1	
1111	111X	

	0101	0111
X1X1	[✓]	[✓]
11XX		

L'impliquant essentiel est X1X1.

P_0 = BD.

Éléments de base

1- Réaliser une MUX 4 à 1 avec un nombre minimal de MUX 2 à 1.

La sélection se fait avec les bits B et A, du plus au moins significatif.

2- Réaliser un MUX 6 à 1 avec un nombre minimal de MUX 4 à 1.

La sélection se fait avec les bits C, B et A, du plus au moins significatif.

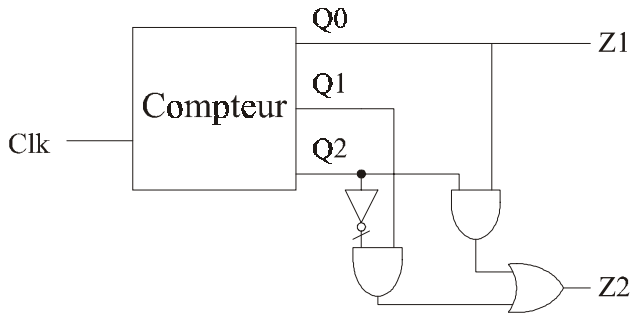
3- Réaliser un DEMUX 1 à 10 avec un nombre minimal de DEMUX 1 à 4.

La sélection se fait avec les bits D, C, B et A, du plus au moins significatif.

La sélection doit suivre un code binaire naturel.

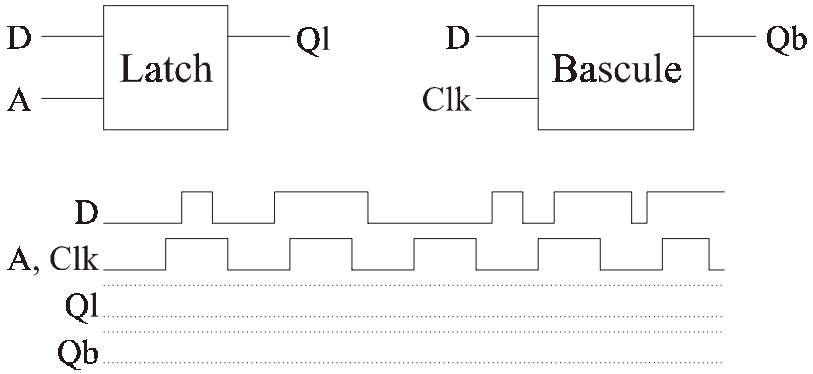
4- Compléter le diagramme de temps pour Ql et Qb.

5- Soit le circuit suivant:



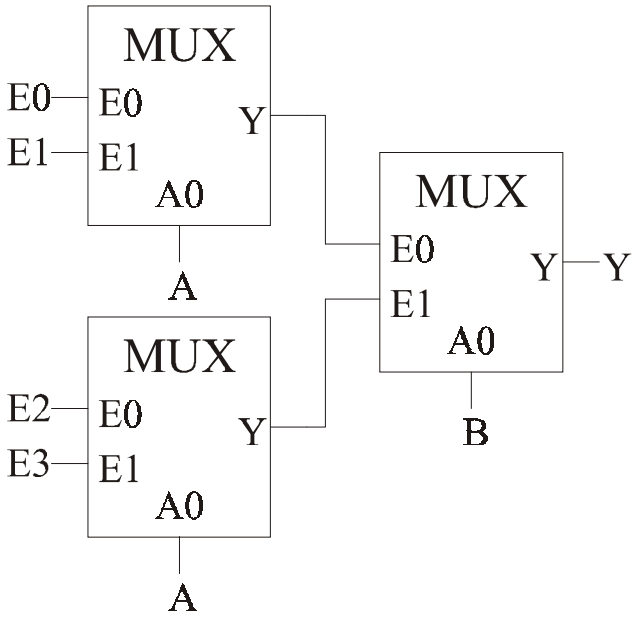
Trouver f(Z1) et f(Z2) sachant que f(clk) = 1MHz.

(La fréquence est le nombre de cycles dans un intervalle divisé par sa durée)

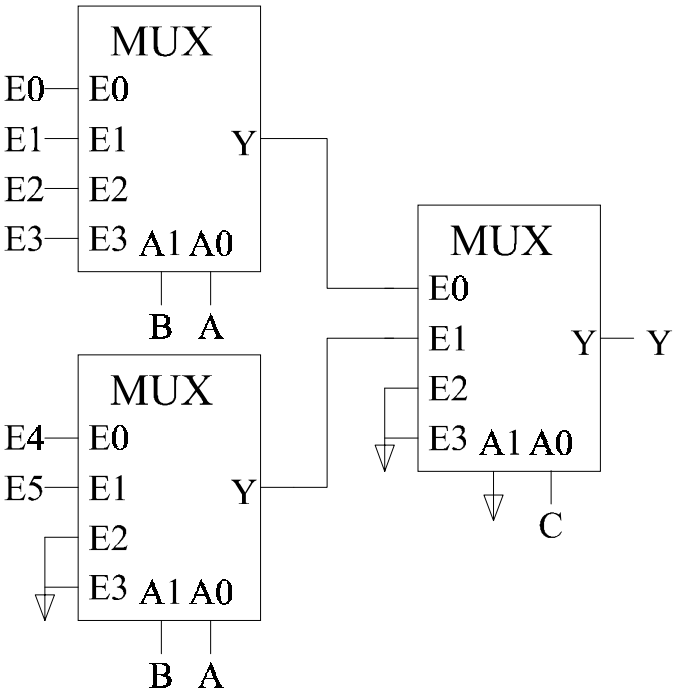


SOLUTIONS:

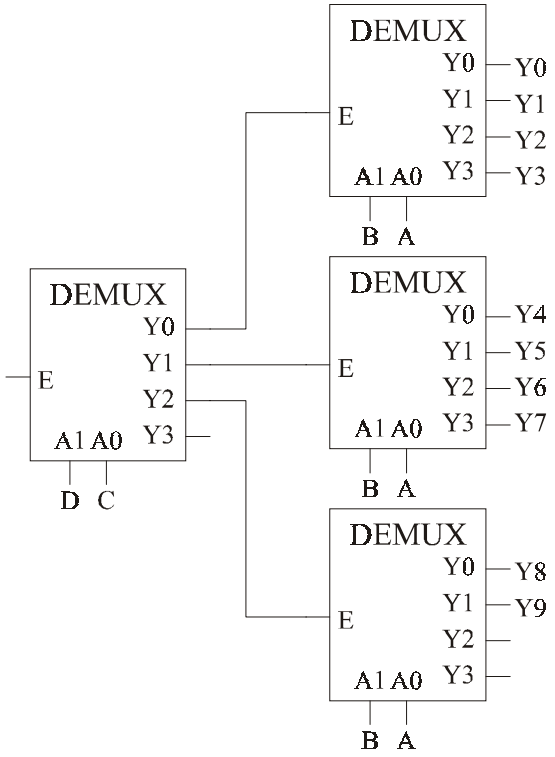
1-



2-

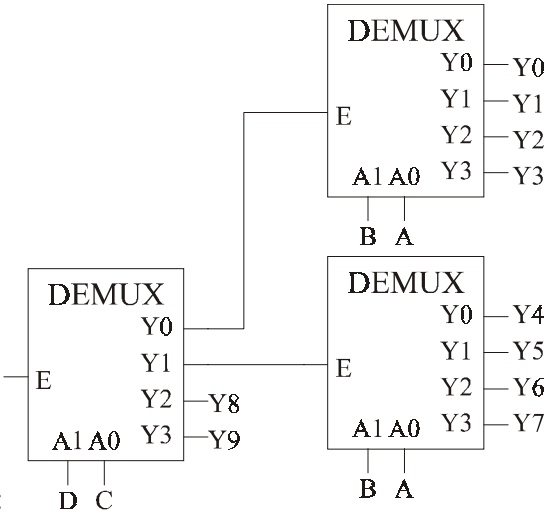


3-

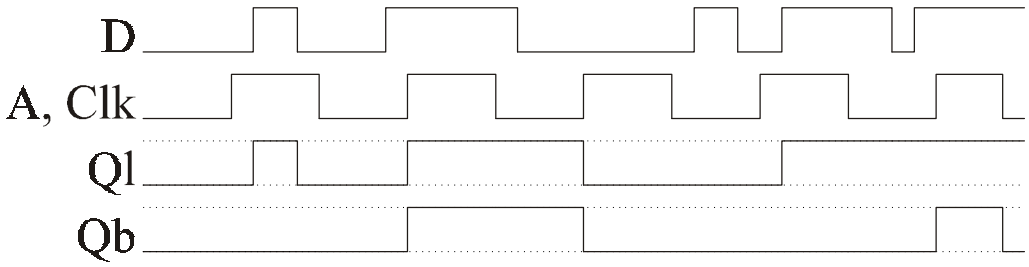


Cette solution est la seule qui respecte une code de sélection binaire naturel. La solution suivante réalise aussi une multiplexage des données mais ne respecte pas le code binaire naturel car l'adresse 1001 active la sortie 8. Pour activer la sortie 9, il faut un code de la forme 11XX, ce qui est invalide.

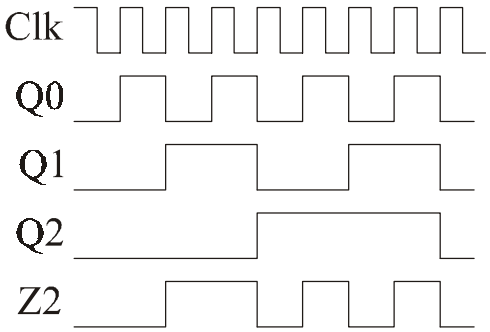
Solution invalide:



4-



5-



$$f(Z1) = f(Q0) = \frac{1}{2}1MHz = 500kHz$$

$$f(Z2) = \frac{3}{8}1MHz = 375kHz$$

CHAPITRE 3 : La Logique Mixte

Synthèse et analyse de la logique mixte

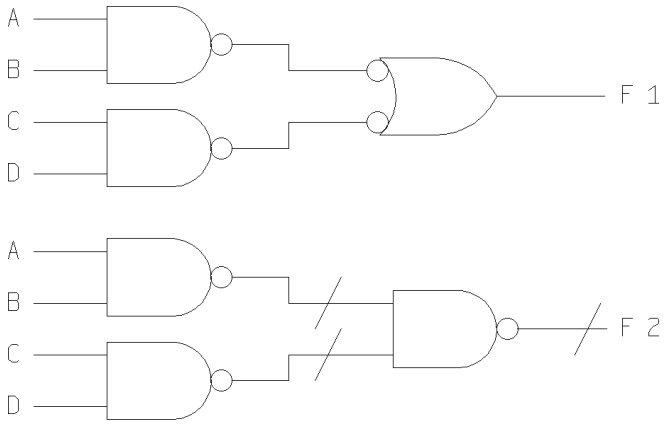
1- Réaliser les fonctions suivantes en respectant les conventions de la logique mixte:

a) $F = AB + A\overline{C} + \overline{B}D$ avec A, B*, C, D* et F.

b) $F = ABC + \overline{A\overline{B}C + \overline{A}B\overline{C}}$ avec A, B, C* et F*

c) $F = \overline{\overline{AB} + \overline{CD}}$ avec A*, B*, C, D et F*.

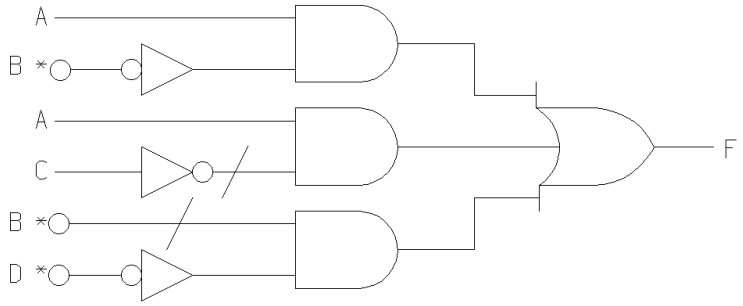
2- Est-ce les fonctions F1 et F2 produisent la même fonction logique?



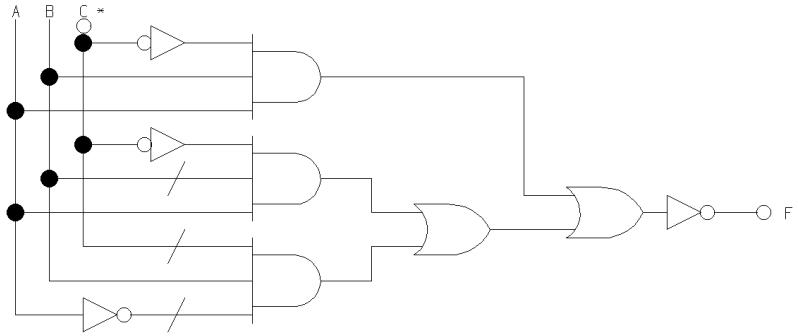
3- Réaliser la fonction $F = ABC + \overline{B}\overline{C}$ avec des 7400 seulement avec A, B*, C et F.

SOLUTIONS

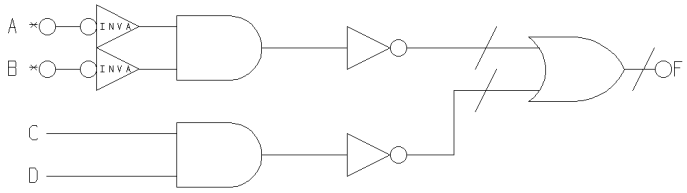
1- a)



b)

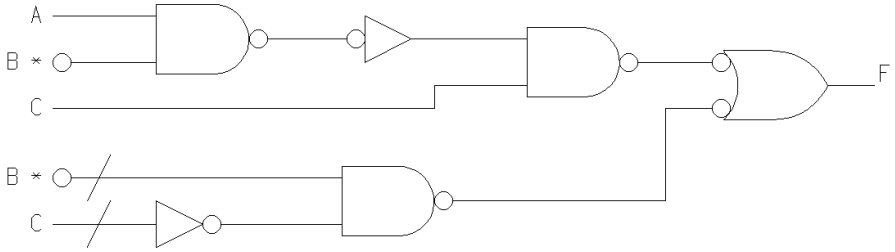


c)



2- Oui

3-



L'inverseur est réalisé en branchant les deux entrées du NON ET ensembles.

Chapitre 4 : Les réseaux itératifs

Synthèse de réseaux itératifs

1- Concevoir un réseau itératif qui détecte une entrée de la forme "…010101…". La sortie S égale 1 si tous les bits de l'entrée sont alternés et 0 sinon.

2- Concevoir un réseau itératif qui reçoit une entrée sous forme de séries de 1 et de séries de 0. Le réseau doit générer un signal d'erreur (E=1) si une de ces séries contient un nombre pair de 1 ou de 0.

Exemple: 00011111000100000111 → E = 0

00011101100011100111 → E = 1

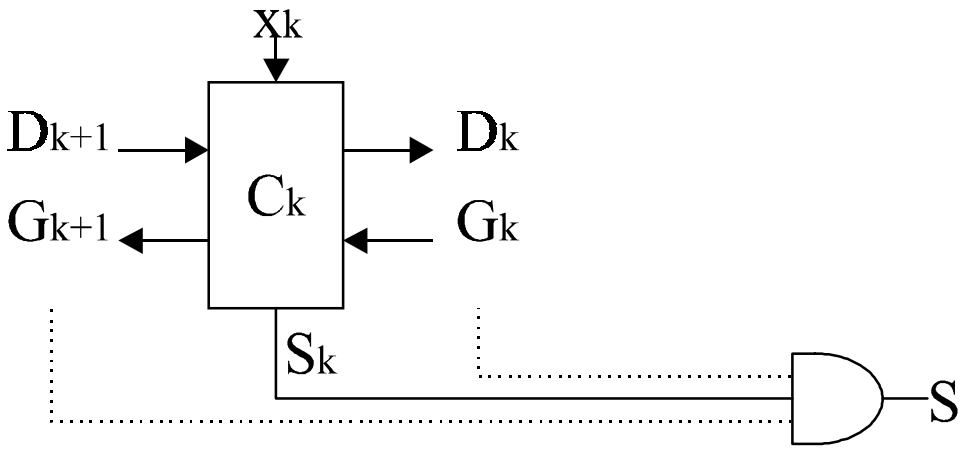
3- Concevoir un réseau itératif qui calcule le complément à deux d'un nombre. Le complément à deux est trouvé en inversant tous les bits et en ajoutant 1. Une fois travaillé, l'algorithme peut être simplifié sous cette forme: en partant du bit le moins significatif, copier tous les zéro et le premier 1. Tous les autres bits sont inversés.

Exemple:

Nombre	1010	1000
Complément à deux	0101	1000

SOLUTIONS:

1-



Sk=1 si Xk+1 et Xk-1 sont l'inverse de Xk.

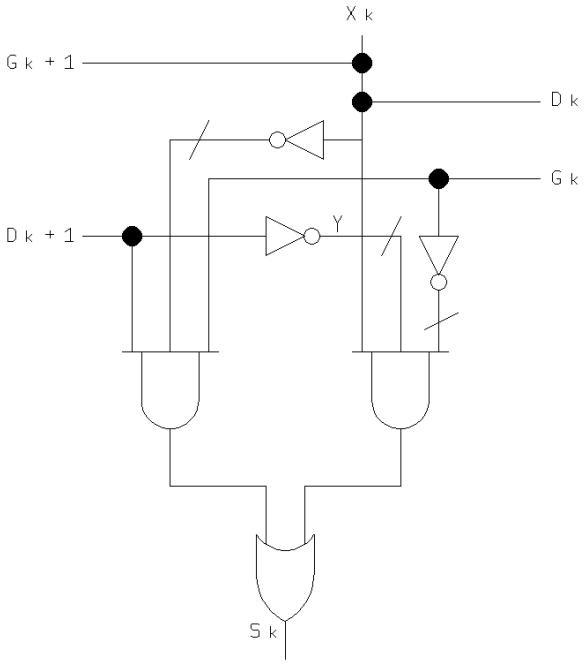
Sk=0 sinon.

Dk=Xk pour cellule Ck-1.

Gk+1=Xk pour cellule Ck+1.

Dk+1	Xk	Gk	Sk
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$$S_k = X_k \overline{D}_{k+1} \overline{G}_k + \overline{X}_k D_{k+1} G_k$$



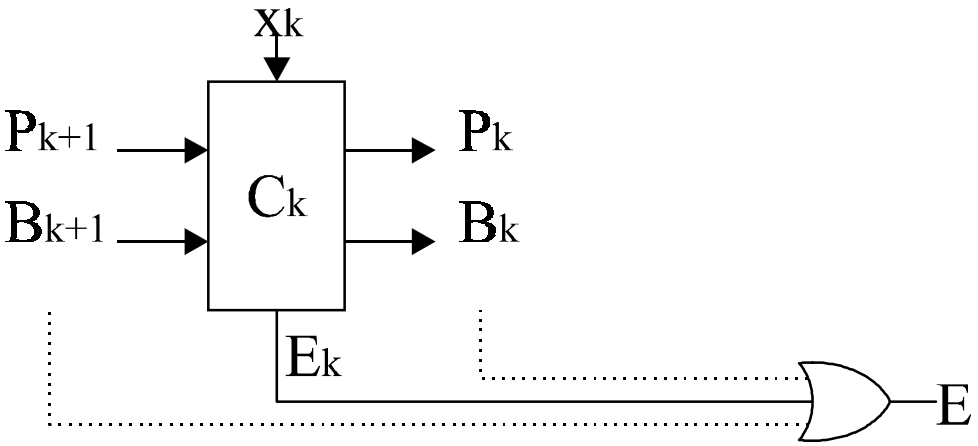
Conditions aux frontières:

C0 doit se comporter comme si la cellule à sa droite était l'inverse de X0. Le raisonnement similaire s'applique aussi à Cn-1 d'où

$$G_0 = \overline{X}_0$$

$$G_n = \overline{X}_{n-1}$$

2-



Bk+1 = Bit précédent

Pk+1 = Parité des bits précédents

Pk+1	Bk+1	Xk	Pk	Bk	Ek
0	0	0	1	0	0
0	0	1	X	X	1
0	1	0	X	X	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	0	1	0

$$P_k = P_{k+1} + (B_k \oplus X_k)$$

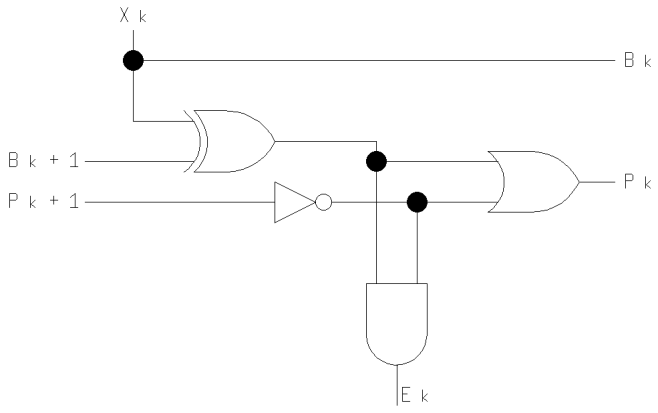
$$B_k = X_k$$

$$E_k = \overline{P}_{k+1}(B_{k+1} \oplus X_k)$$

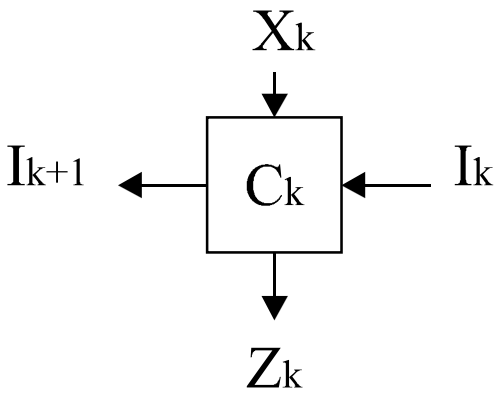
Conditions extrémités:

Pour initier correctement la vérification, on peut poser l'hypothèse que le premier bit est précédé d'un nombre impair de bit inverse. Cela nous assure un démarrage valide à tout coup. $P_n = 1, B_n = \overline{X}_{n-1}$.

Schéma:



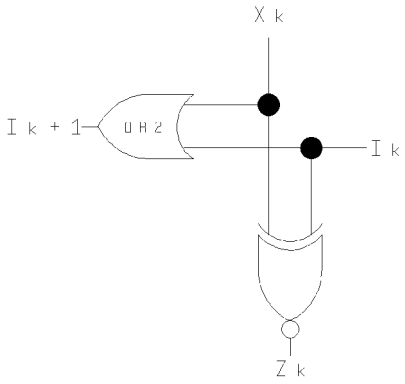
3-



Ik = 1 → Inverser le bit courant.

Ik	Xk	Zk	Ik+1
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

$$I_{k+1} = I_k + X_k$$
$$Z_k = I_k \otimes X_k$$



Chapitre 5 : Codes analytiques et représentatifs

Codes

1- Exprimer les nombres suivants en utilisant le code indiqué avec un nombre minimal de bits. Si la représentation est impossible, indiquez-le.

	Binaire naturel	Binaire avec signe	Complément à 1	Complément à 2
13				
13241				
0				
-64				
-869				

2- Effectuer les opérations suivantes en complément à 2 représenté sur 8 bits. Indiquer s'il y a débordement.

a) 113 - 26

b) 100 + 33

c) 45 - 63

d) -38 + -91

Solutions

1-

	Binaire naturel	Binaire avec signe	Complément à 1	Complément à 2
13	1101	0 1101	0 1101	0 1101
13241	11 0011 1011 1001	0 1011 1001	0 1011 1001	0 1011 1001
0	0	00 ou 10	00 ou 11	0
-64	X	1100 0000	1011 1111	100 0000
-869	X	111 0110 0101	100 1001 1010	10 1001 1011

2-

a)

	Retenues	1	1	1	0	0	0	0	0	0
113			0	1	1	1	0	0	0	1
- 26		+	1	1	1	0	0	1	1	0
87	1		0	1	0	1	0	1	1	1

b)

	Retenues	0	1	1	0	0	0	0	0	0
100			0	1	1	0	0	1	0	0
+ 33		+	0	0	1	0	0	0	0	1
133	Débordement		1	0	0	0	0	1	0	1

c)

	Retenues	0	0	0	0	0	0	0	1	
45			0	0	1	0	1	1	0	1
- 63		+	1	1	0	0	0	0	0	1
- 18			1	1	1	0	1	1	1	0

d)

	Retenues	1	0	0	0	0	0	0	0	
- 38			1	1	0	1	1	0	1	0
- 91		+	1	0	1	0	0	1	0	1
- 129	Débordement	1	0	1	1	1	1	1	1	1

Détection et correction d'erreurs

1- L'encodage par parité orthogonale permet d'augmenter la fiabilité d'un canal de communication en ajoutant des bits de parité. Compléter le tableau suivant en indiquant le ration données/total sous forme fractionnaire (12/31) et sous forme décimale (0.39). Les valeurs de m et n sont des dimensions de la matrice de données.

	1	2	3	4	5	6	7	8
1								
2	X							
3	X	X						
4	X	X	X					
5	X	X	X	X				
6	X	X	X	X	X			
7	X	X	X	X	X	X		
8	X	X	X	X	X	X	X	

2- Choisir la longueur des mots d'un code de Hamming pour transférer des données à un rythme de 8 Mbps ($8x2^{20}$ bits par seconde) sur un canal de transmission qui peut permettre jusqu'à 10 Mbps. Choisir les mots les plus petit possible.

Solutions

1-

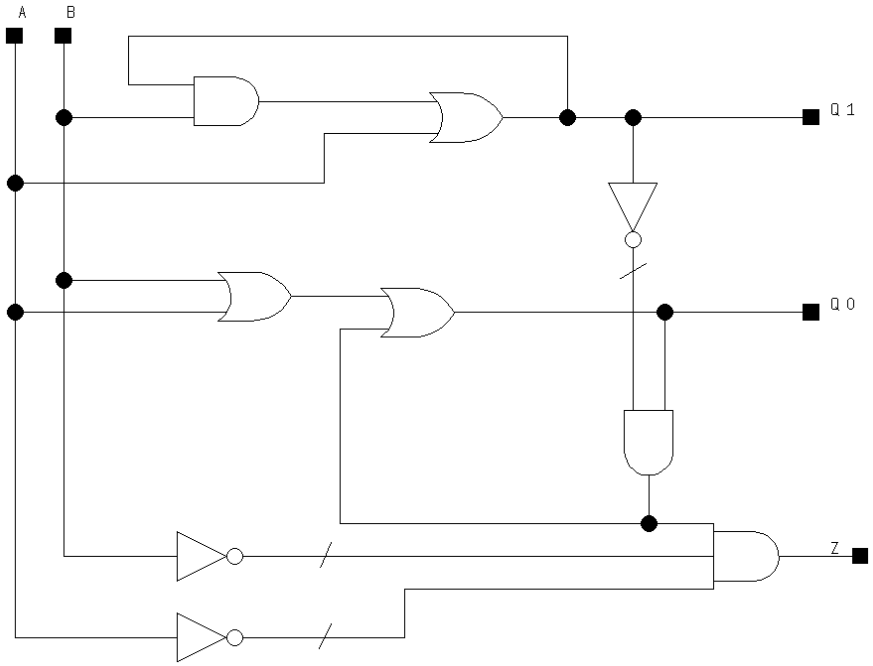
	1	2	3	4	5	6	7	8
1	¼ .25	2/6 .33	3/8 .38	4/10 .40	5/12 .42	6/14 .43	7/16 .44	8/18 .44
2	X	4/6 .44	6/12 .50	8/15 .53	10/18 .56	12/21 .57	14/24 .58	16/27 .59
3	X	X	9/16 .56	12/20 .60	15/24 .63	18/28 .64	21/32 .66	24/36 .67
4	X	X	X	16/25 .64	20/30 .67	24/35 .69	28/40 .70	32/45 .71
5	X	X	X	X	25/36 .69	30/42 .71	35/48 .73	40/54 .74
6	X	X	X	X	X	36/49 .73	42/56 .75	48/63 .76
7	X	X	X	X	X	X	49/64 .77	56/72 .78
8	X	X	X	X	X	X	X	64/81 .79

2- En choisissant des mots de Hamming avec 5 bits de parité, on peut inclure 26 bits de message par mots de 31 bits. Le ration de donnée/total est alors de 26/31 soit 0.839. Ceci implique que pour transmettre 8 Mbps de données, le canal doit permettre 9.54 Mbps. Un mot plus grand réduirait la fiabilité car le code de Hamming ne permet pas plus d'une erreur par mot.

Chapitre 6 : La logique séquentielle

Analyse de circuits asynchrones

1- Analyser le circuit suivant



Solutions

1- $Q_1 = A + Q_1 B$

$Q_0 = A + B + \overline{Q_1} Q_0$

$Z = \overline{Q_1} Q_0 \overline{A}\,\overline{B}$

$Q_1^+ Q_0^+$ \ AB	00	01	11	10
00	(00)	01	11	11
01	(01)	(01)	11	11
11	00	(11)	(11)	(11)
10	00	11	11	11

La sortie Z est active si le système est dans l'état 01 et que les entrées sont 00.

Le système est un détecteur de séquence 00, 01, 00.

Synthèse de circuits asynchrones

1- Concevoir un système séquentiel asynchrone qui détecte la séquence 00, 10, 00, 01 pour deux entrées A et B. Les entrées ne peuvent changer d'état simultanément. Une sortie Z est active lorsque AB vaut 01 et que la séquence a été respectée.

2- Concevoir une bascule D qui est active sur un front descendant.

Solutions

1-

	AB 00	01	11	10	Z
1	(1)	6	-	2	0
2	3	-	5	(2)	0
3	(3)	4	-	2	0
4	1	(4)	5	-	1
5	-	6	(5)	7	0
6	1	(6)	5	-	0
7	1	-	5	(7)	0

La ligne 1 devient a. Les lignes 2 et 3 sont fusionnées pour devenir b. Les lignes 5, 6 et 7 deviennent la ligne d. La ligne 4 devient la ligne c où la sortie est active.

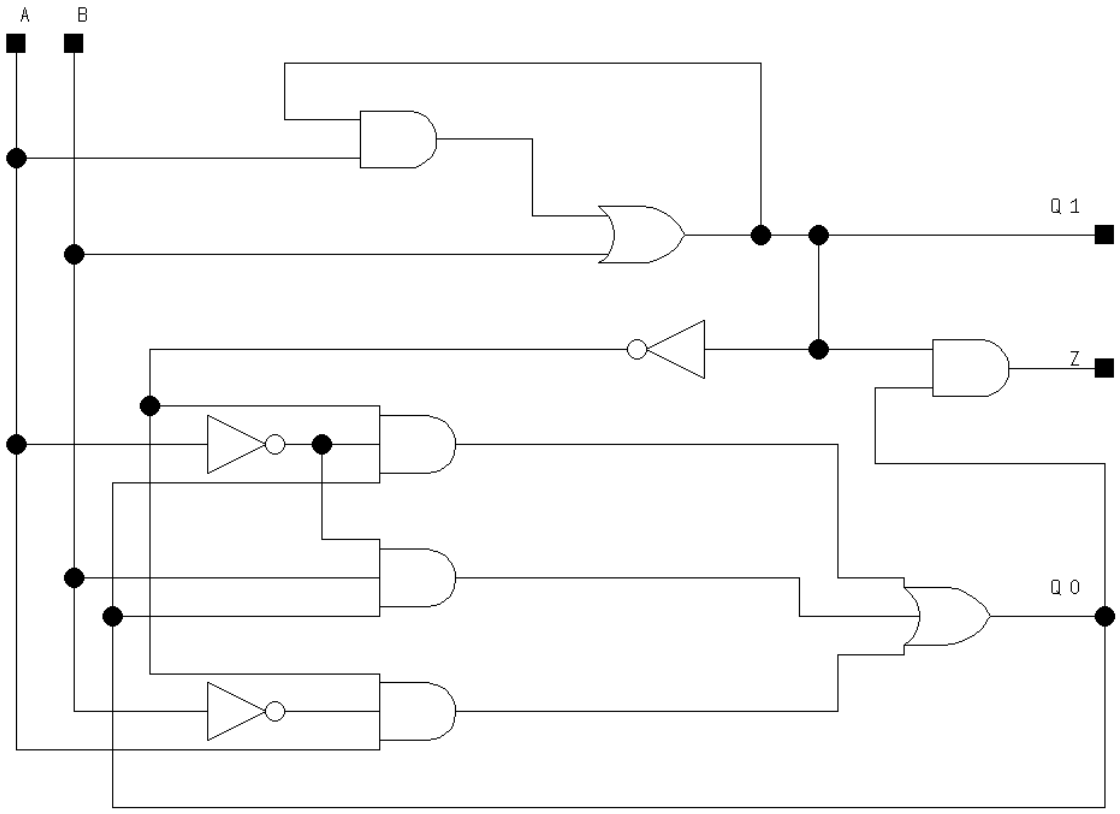
	AB 00	01	11	10	z
a	(1)	6	-	2	0
b	(3)	4	5	(2)	0
c	1	(4)	5	-	1
d	1	(6)	(5)	(7)	0

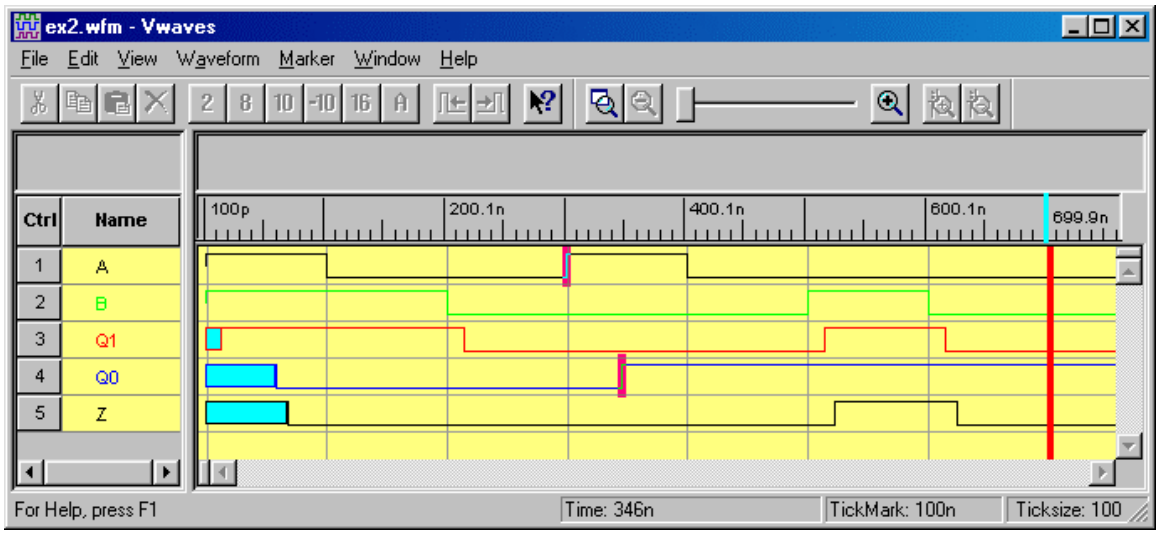
$Q_1^+Q_0^+$	AB 00	01	11	10	z
00	(00)	10	XX	01	0
01	(01)	11	10	(01)	0
11	00	(11)	10	XX	1
10	00	(10)	(10)	(10)	0

$$Q_1^+ = B + Q_1 A$$

$$Q_0^+ = \overline{Q_1} Q_0 \overline{A} + \overline{A} B Q_0 + A \overline{B} \overline{Q_1}$$

$$Z = Q_1 Q_0$$


A
B
Q 1
Z
Q 0


ex2.wfm - Vwaves
File Edit View Waveform Marker Window Help
Ctrl
Name
100p
200.1n
400.1n
600.1n
699.9n
1 A
2 B
3 Q1
4 Q0
5 Z
For Help, press F1
Time: 346n
TickMark: 100n
Ticksize: 100

2- Pour ne pas prendre de chances, tout les cas possible sont considérés. Il y a 4 combinaisons possibles pour D et H. Pour chacune d'elles, il u a deux possibilités pour la sortie Q.

	DH 00	01	11	10	Q
1	(1)	3	-	7	0
2	(2)	4	-	8	1
3	1	(3)	5	-	0
4	1	(4)	6	-	1
5	-	3	(5)	8	0
6	-	4	(6)	8	1
7	1	-	5	(7)	0
8	2	-	6	(8)	1

Les lignes sont fusionnées en tenant compte de Q pour que la sortie soit fonction de l'état seulement (Moore). a = 1 + 3 + 7, b = 5, c = 2 + 6 + 8, d = 4

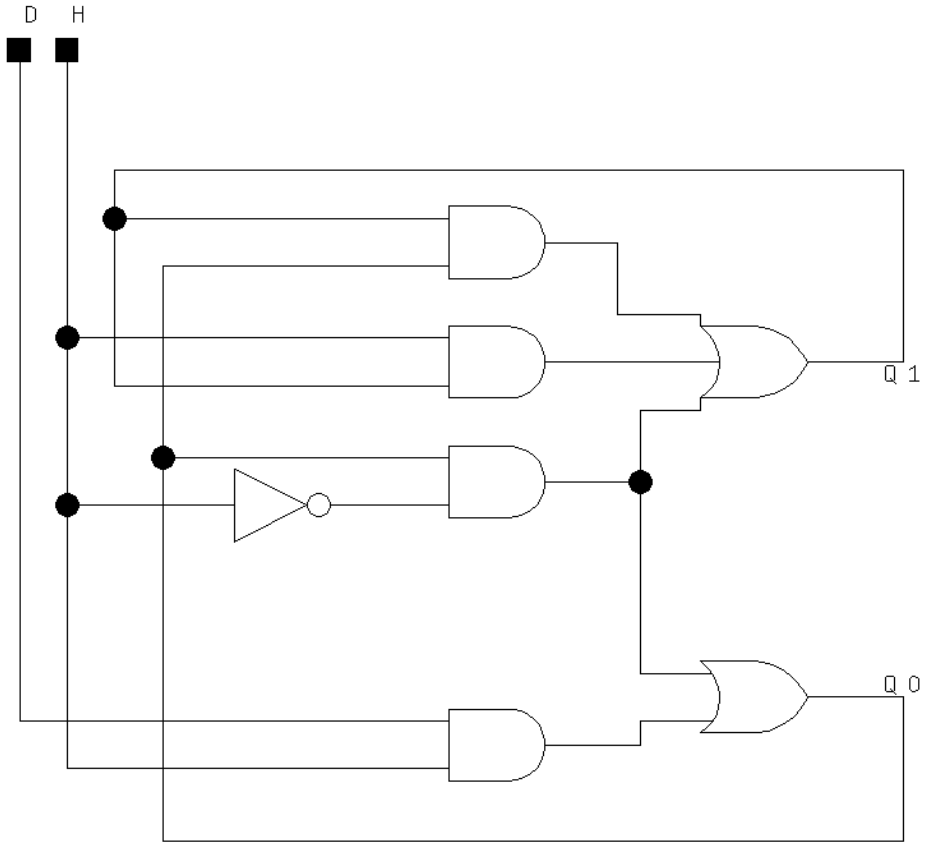
	DH 00	01	11	10	Q
a	(1)	(3)	5	(7)	0
b	-	3	(5)	8	0
c	(2)	4	(6)	(8)	1
d	1	(4)	6	-	1

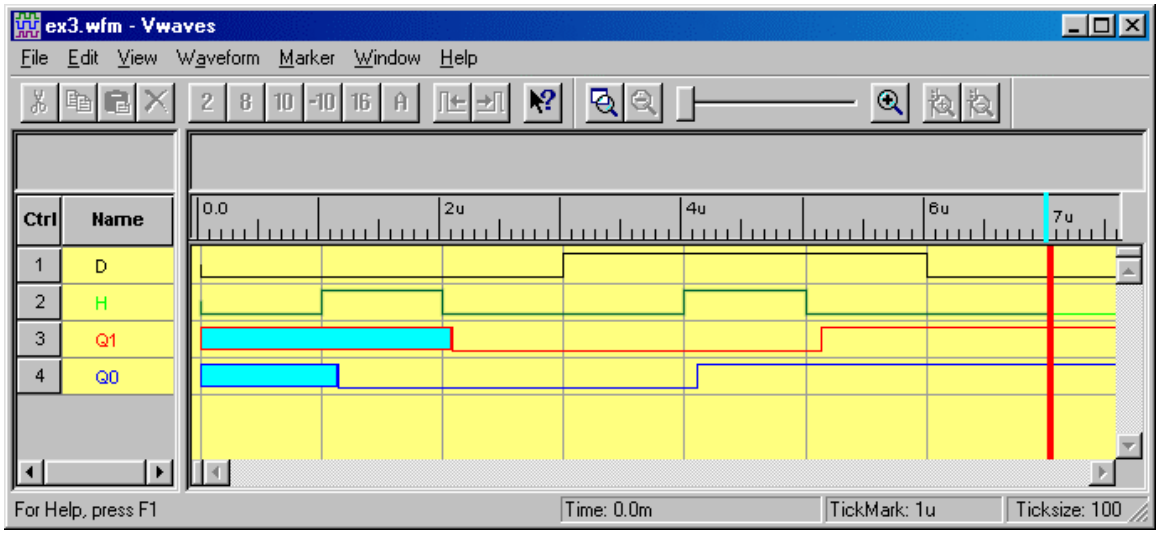
$Q_1^+Q_0^+$	DH 00	01	11	10	Z
00	(00)	(00)	01	(00)	0
01	XX	00	(01)	11	0
11	11	10	11	(11)	1
10	00	(10)	(11)	XX	0

$$Q_1^+ = Q_1Q_0 + Q_1H + Q_0\overline{H}$$

$$Q_0^+ = DH + Q_0\overline{H}$$

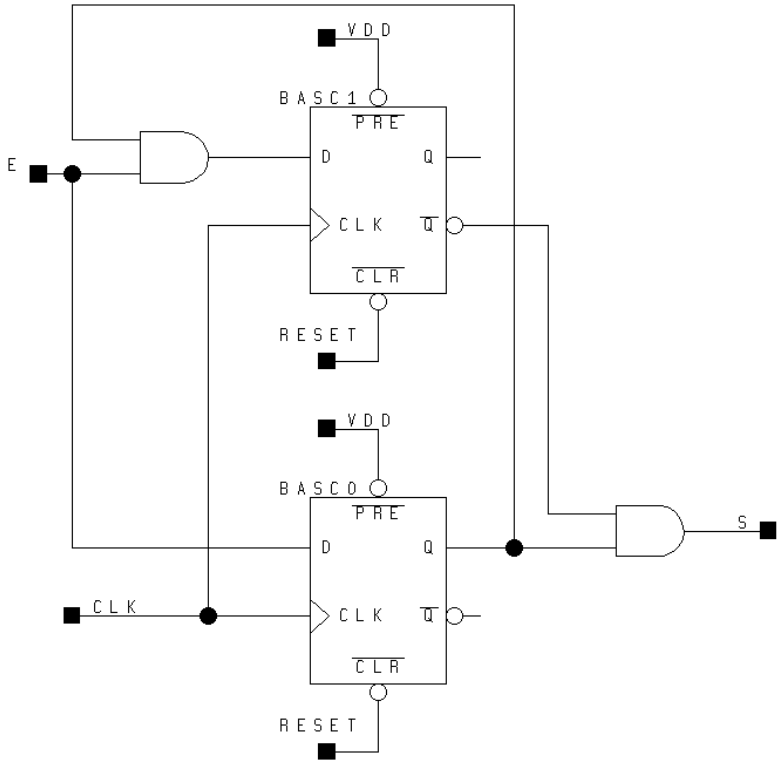
$$Z = Q_1$$





Analyse de circuits synchrone

1- Analyser le circuit suivant



Solutions

1- Les équations sont :

$D_1 = Q_0E$

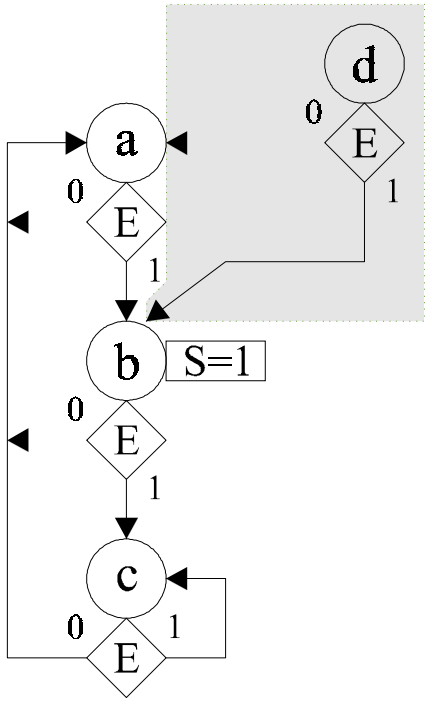
$D_0 = E$

$S = \overline{Q_1}Q_0$

La table d'excitation et de transition d'états est :

Q_1	Q_0	E	$D_1=Q_1^+$	$D_0=Q_0^+$	S
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	1	1	0

Le diagramme d'état est donc :



Ce circuit est un détecteur de fronts montants.

Synthèse de circuits synchrones

1- Réaliser un circuit qui détecte la séquence 00, 01, 00 sur les entrées AB. Les entrées changent sur le front descendant de l'horloge et la machine change d'état sur le front montant. Produire une sortie Z=1 lorsque la séquence est détectée. Réaliser ce circuit avec des bascules D.

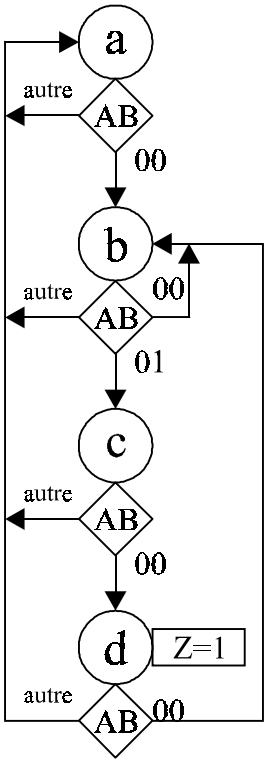
2- Réaliser un circuit qui détecte la séquence 0,0,1,0,1,1 sur une entrée E. L'entrée change sur le front descendant de l'horloge et la machine change d'état sur le front montant. Produire une sortie S=1 lorsque la séquence est détectée. Réaliser ce circuit avec des bascules D.

3- Réaliser le diagramme d'état d'une machine qui compte le nombre de 1 dans une séquence de trois bits d'une entrée I. L'état de I change sur les fronts descendants et l'évaluation du bit I est faite sur les fronts montants.

4- Réaliser le diagramme d'état d'une machine contrôlant une machine distributrice. Le montant à percevoir est 50¢ et la machine ne rend pas la monnaie. Les pièces acceptées sont les pièces de 5¢, 10¢ et 25¢. Les entrées C, D, V correspondent respectivement à l'insertion d'une pièce de 5¢, 10¢ et 25¢. Noter que deux pièces ne peuvent être insérée simultanément.

Solutions

1-



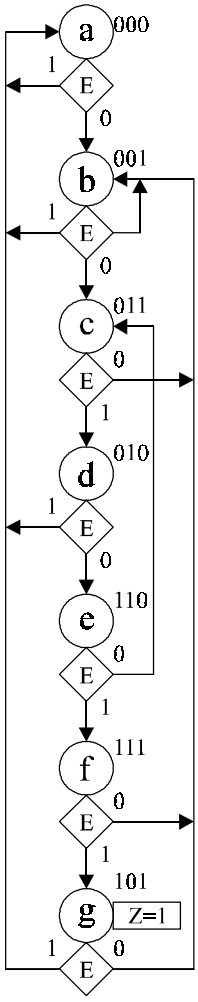
Q_1	Q_0	A	B	$D_1=Q_1^+$	$D_0=Q_0^+$	Z
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	0
0	1	0	1	1	1	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	1	1
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

$$D_1 = \overline{Q_1}Q_0\,\overline{A}B + Q_1Q_0\,\overline{A}\,\overline{B}$$

$$D_0 = \overline{Q_0}\,\overline{A}\,\overline{B} + \overline{Q_1}Q_0\,\overline{A}$$

$$Z = Q_1\,\overline{Q_0}$$

2-



Q_2	Q_1	Q_0	E	$D_2=Q_2^+$	$D_1=Q_1^+$	$D_0=Q_0^+$	Z
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	0	1	0	0	1	1	0
0	0	1	1	0	0	0	0
0	1	0	0	1	1	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	1	0
0	1	1	1	0	1	0	0
1	0	0	0	x	x	x	x
1	0	0	1	x	x	x	x
1	0	1	0	0	0	1	1
1	0	1	1	0	0	0	1
1	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0
1	1	1	0	0	0	1	0
1	1	1	1	1	0	1	0

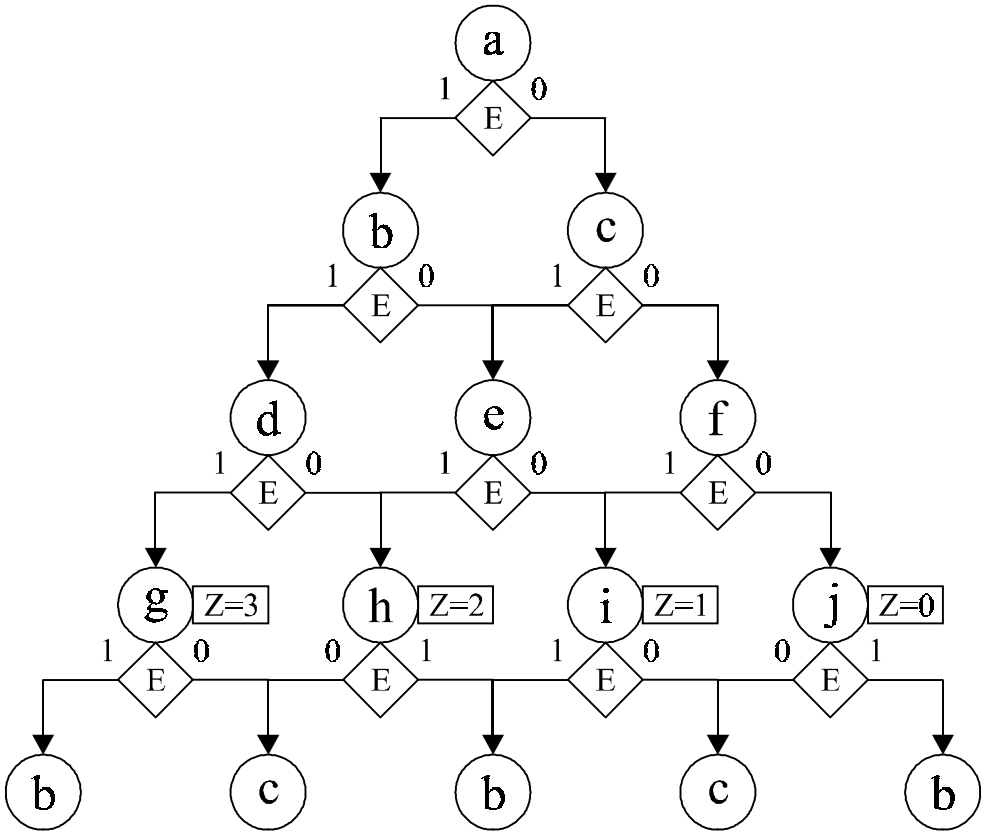
$$D_2 = \overline{Q_2}Q_1\,\overline{Q_0}\,\overline{E} + Q_2Q_1E$$

$$D_1 = Q_2\,\overline{Q_0} + Q_1\,\overline{Q_0}\,\overline{E} + \overline{Q_2}Q_1Q_0E + \overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}\,\overline{E}$$

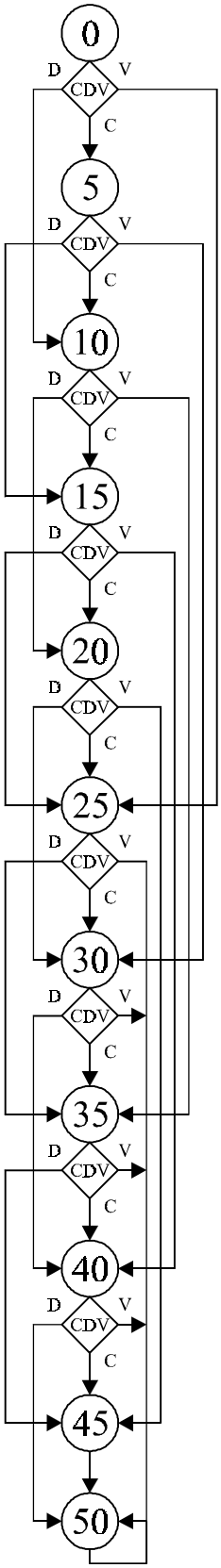
$$D_0 = Q_2Q_1 + Q_0\,\overline{E} + \overline{Q_2}\,\overline{Q_1}\,\overline{E} +$$

$$Z = Q_2\,\overline{Q_1}$$

3-


a
1
0
E
b
c
1
0
E
1
0
E
d
e
f
1
0
E
1
0
E
1
0
E
g
Z=3
h
Z=2
i
Z=1
j
Z=0
1
0
E
0
1
E
1
0
E
0
1
E
b
c
b
c
b

4-


0
D
V
CDV
C
5
D
V
CDV
C
10
D
V
CDV
C
15
D
V
CDV
C
20
D
V
CDV
C
25
D
V
CDV
C
30
D
V
CDV
C
35
D
V
CDV
C
40
D
V
CDV
C
45
50

ANNEXE 2

Fiches techniques

Du

Contrôleur Industriel

MC14500

CHAPTER 1 INTRODUCTION

The Motorola MC14500B is a single chip, one-bit static CMOS processor optimized for decision-oriented tasks. The processor is housed in a 16-pin package and features 16-four-bit instructions. The instructions perform logical operations on data appearing on a one-bit bidirectional data line and data in a one-bit accumulating Result Register within the ICU. All operations are performed at the bit level.

The ICU is timed by a single phase clock signal, generated by an internal oscillator that uses one external resistor. Alternatively, the clock signal may be controlled by an external oscillator. In either case, the clock signal is available for synchronization with other systems. Each of the ICU's instructions execute in a single clock period. The clock frequency may be varied over a wide range. At a clock frequency of 1 MHz, some 8300, plus, instructions, may be executed in a 60 Hz power line half cycle.

In a system, the ICU may be used in conjunction with the complete line of over 100 standard B-series CMOS logic devices. This allows tailoring a system to an application, and allows a judicious mix of customized hardware and software to be achieved.

As an initial example, Figure 1.1 shows a block diagram of a minimal ICU system with four component blocks in addition to the ICU. The blocks are:

- The ICU, or central controller of the system.
- The memory, either permanent Read Only Memory (ROM) or temporary Random Access Memory (RAM). Here, the steps of the program are stored, both individual instructions and addresses of inputs and outputs.

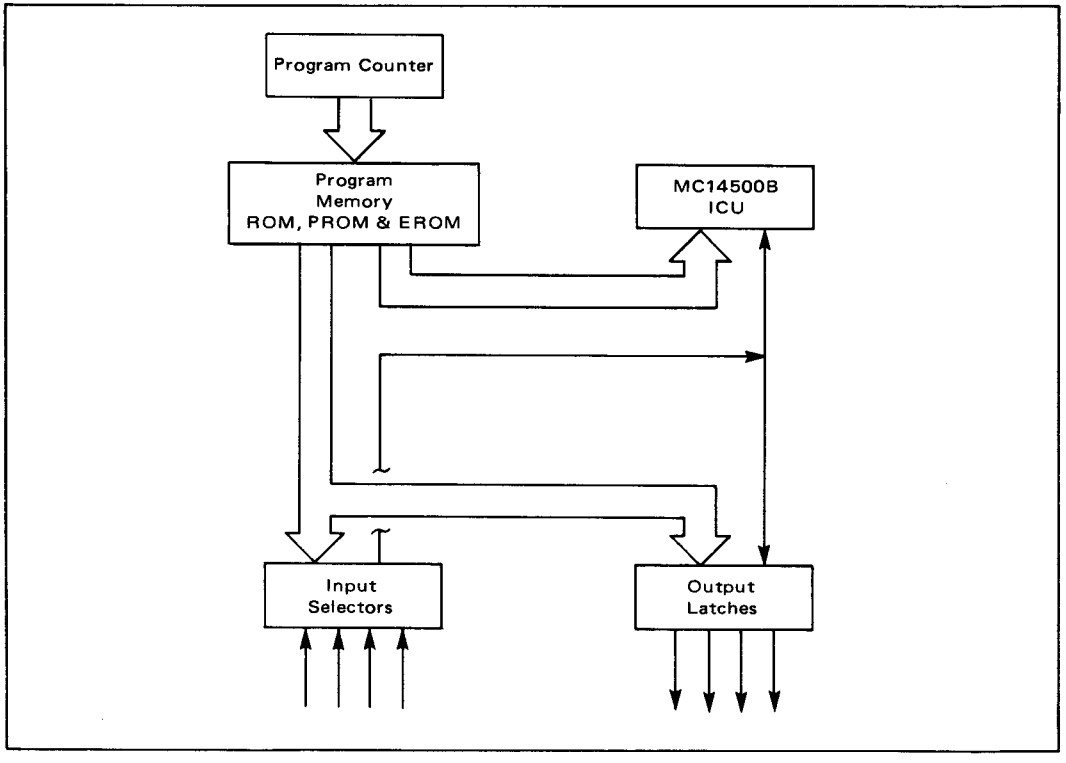


Figure 1.1 Basic ICU System

- The program counter, used to step the machine through the sequence of instructions.
- Inputs and outputs, each individually selected by the machine, from information contained in the memory.

Note that this system can be expanded almost without bound, in terms of inputs and outputs, so long as the memory is sufficiently wide to address the I/O structure.

There are functions for which one bit machines are poorly suited. These functions are complex calculations or parallel word data processing. When there are many calculations, a one-bit machine is at a disadvantage. When a job is dominated by calculations or data logging, a multi-bit processor is appropriate. When the task is decision and command oriented, a one-bit machine is an excellent choice. The tasks that are mixed between decisions and calculations will be decided upon by economics, the designer's familiarity with alternatives, and how comfortable the designer is with the alternatives. Under some circumstances, a combination of an MC6800 MPU and an MC14500B ICU may be the best solution.

A functional diagram of the MC14500B is shown in Figure 1.2. Central to the ICU is the Result Register, (RR), a one-bit accumulator that stores the results of Boolean manipulations. These results are generated in the Logic Unit, (LU), which has as its inputs, signals from external data and the RR. Instructions are presented to the chip on the 4 instruction lines, (I0, I1, I2, I3), and are latched into the Instruction Register, (IR), on the negative-going edge of X1.

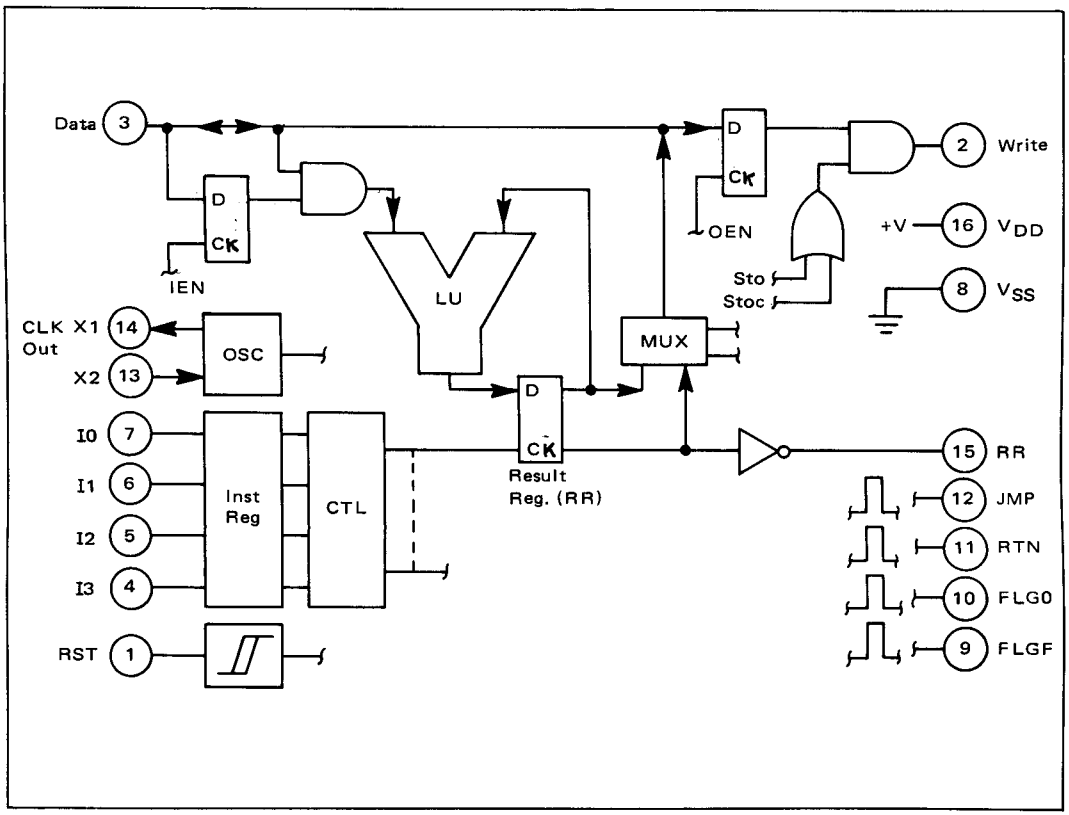


Figure 1.2 MC14500B Block Diagram

The instructions, listed in Figure 1.3, are decoded in the Control Logic (CTL), sending the appropriate logic commands to the LU. Further decoding is also performed in the CTL to send a number of output flags (JMP, RTN, FLG0, FLGF) to pins 9 through 12. These are used as external control signals and remain active for a full clock period after the negative-going edge of X1.

Instruction Code		Mnemonic	Action
#0	0000	NOPO	No change in registers. RR→RR, FLG0← ⎍
#1	0001	LD	Load Result Reg. Data → RR
#2	0010	LDC	Load Complement $\overline{\text{Data}}$ → RR
#3	0011	AND	Logical AND. RR • D → RR
#4	0100	ANDC	Logical AND Compl. RR ∘ $\overline{D}$ → RR
#5	0101	OR	Logical OR. RR + D → RR
#6	0110	ORC	Logical OR Compl. RR + $\overline{D}$ → RR
#7	0111	XNOR	Exclusive NOR. If RR = D, RR ← 1
#8	1000	STO	Store. RR → Data Pin, Write ← 1
#9	1001	STOC	Store Compl. $\overline{RR}$ → Data Pin, Write ← 1
#A	1010	IEN	Input Enable. D → IEN Reg.
#B	1011	OEN	Output Enable. D → OEN Reg.
#C	1100	JMP	Jump. JMP Flag ← ⎍
#D	1101	RTN	Return. RTN Flag ←⎍, Skip next inst.
#E	1110	SKZ	Skip next instruction if RR = 0
#F	1111	NOPF	No change in Registers RR → RR, FLGF ←⎍

Figure 1.3 MC14500B Instruction Set

The timing signals are generated from an on-chip oscillator, (OSC), with the operating frequency set via an external resistor connected between pins 13 and 14. Figure 1.4 shows the relationship between frequency and resistor values. The resultant square wave output appearing at pin 14 is used both within the ICU and as a general system clock. Alternatively, the system may be externally clocked at pin 13.

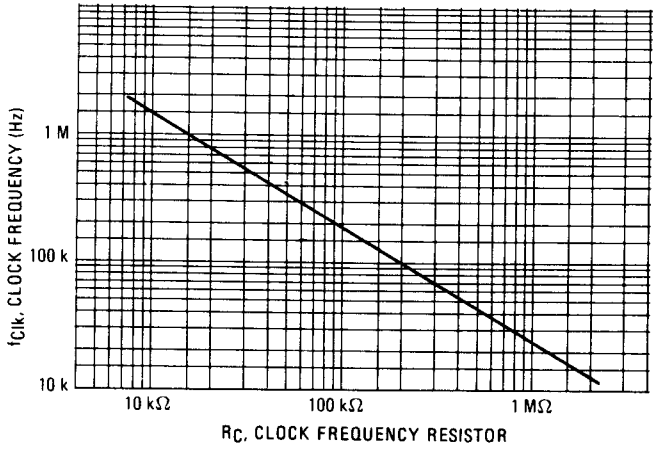


Figure 1.4 Typical Clock Frequency Versus Resistor (R_C)

Two internal latches, Input Enable Register, (IEN), and Output Enable Register, (OEN), control data transfers to and from the ICU. The IEN acts to enable the data path to the LU when in the high state. The OEN, in the high state, enables the Write signal. It should be noted that both of these registers are set via the Data pin.

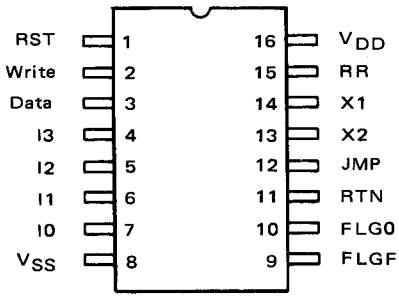


Figure 1.5 Pin Assignment

A Master Reset pin (RST), active high, is provided to clear all registers and hold the FLAG signals within the ICU at zero. The oscillator pin (X1) is held in the high state when RST is high. When RST goes low, the oscillator starts after a delay. In addition, the state of RR is available at the buffered RR pin 15.

The ICU chip is housed in a 16-pin dual-in-line package, available in either plastic or ceramic. The various temperature ranges and package types are as follows:

MC14500BAL: Ceramic package; MIL temperature range

MC14500BCL: Ceramic package, Commercial temperature range

MC14500BCP: Plastic package, Commercial temperature range

Pin assignments are shown in Figure 1.5.

The maximum ratings and the electrical characteristics of the ICU are shown in Figure 1.6. These characteristics conform to JEDEC B-Series specifications governing CMOS B-Series devices which have a recommended supply voltage operating range from 5 to 15 Vdc. In electrically noisy industrial environments, supply voltages of 15 Vdc are recommended to make best use of the excellent noise immunity characteristics of CMOS logic. In addition to being able to work in conjunction with over 100 B-series devices, the ICU also works with non-B-series CMOS parts. Refer to Motorola Semiconductor Data Library, CMOS Volume 5/Series B, for further information regarding the many devices that are compatible with the ICU.

The switching characteristics and explanatory waveforms are shown in Figures 1.7 and 1.8, respectively. All times are related to the pin 14 clock signal, X1. At this printing, only specifications for typical times, at V_{DD} = 10 Vdc, were available. Refer to the MC14500B data sheet for up-to-date specifications.

ELECTRICAL CHARACTERISTICS

Characteristic		Symbol	V_{DD} Vdc	25° Typ	Unit
Output Voltage V_{in} = V_{DD} or 0	"0" Level	V_{OL}	10	0	Vdc
V_{in} = 0 or V_{DD}	"1" Level	V_{OH}	10	10	Vdc
Input Voltage* RST, D, X_2 (V_O = 9.0 or 1.0 Vdc)	"0" Level	V_{IL}	10	4.50	Vdc
(V_O = 1.0 or 9.0 Vdc)	"1" Level	V_{IH}	10	5.50	Vdc
Input Voltage I0, I1, I2, I3 (V_O = 9.0 or 1.0 Vdc)	"0" Level	V_{IL}	10	2.2	Vdc
(V_O = 1.0 or 9.0 Vdc)	"1" Level	V_{IH}	10	3.1	Vdc
Output Drive Current D, Write (V_{OH} = 9.5 Vdc)	Source	I_{OH}	10	-6.0	mAdc
(V_{OL} = 0.5 Vdc)	Sink	I_{OL}	10	6.0	mAdc
Output Drive Current (AL Device) Outputs		I_{OH}			mAdc
(V_{OH} = 9.5 Vdc)	Source	I_{OH}	10	-2.25	mAdc
(V_{OL} = 0.5 Vdc)	Sink	I_{OL}	10	2.25	mAdc
Output Drive Current (CL/CP Device) Other Outputs		I_{OH}			mAdc
(V_{OH} = 9.5 Vdc)	Source		10	-2.25	
(V_{OL} = 0.5 Vdc)	Sink	I_{OL}	10	2.25	mAdc
Input Current (RST)		I_{in}	15	150	μAdc
Input Current (AL Device)		I_{in}	15	±0.00001	μAdc
Input Current (CL/CP Device)		I_{in}	15	±0.00001	μAdc
Input Capacitance (DATA)		C_{in}	—	15	pF
Input Capacitance (All Other Inputs) (V_{in} = 0)		C_{in}	—	5.0	pF
Quiescent Current (AL Device) (Per Package)		I_{DD}	10	0.010	μAdc
Quiescent Current (CL/CP Device) (Per Package)		I_{DD}	10	0.010	μAdc

*T_{low} = -55°C for AL Device, -40°C for CL/CP Device
T_{high} = +125°C for AL Device, +85°C for CL/CP Device.

Noise Margin for both "1" and "0"
level = 2.0 Vdc min @ V_{DD} = 10 Vdc

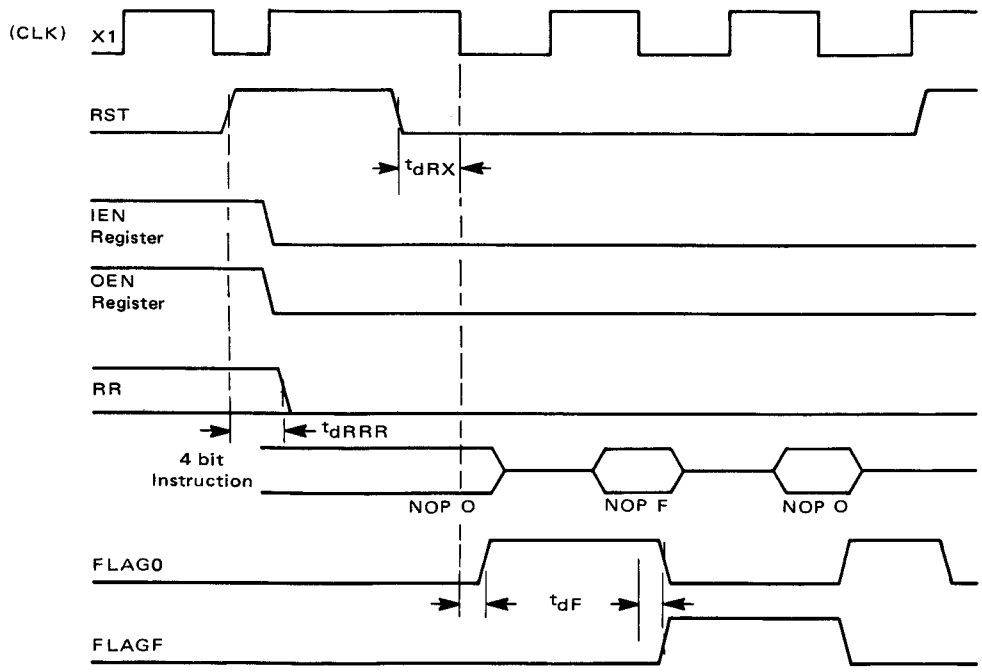
Figure 1.6

SWITCHING CHARACTERISTICS

Characteristic	Symbol	V_{DD} Vdc	All Types Typ	Unit
Propagation Delay Time				
X1 to RR	t_{dR}	10	110	ns
X1 to FLAGF, FLAG0, RTN, JMP	t_{dF}	10	100	ns
X1 to WRITE	t_{dW}	10	125	ns
X1 to DATA	t_{dD}	10	120	ns
RST to RR	t_{dRRR}	10	110	ns
RST to X1	t_{dRX}	10	120	ns
RST to FLAGF, FLAG0, RTN, JMP	t_{dRF}	10	90	ns
RST to WRITE, DATA	t_{dRW}	10	110	ns
Minimum Clock Pulse Width, X1	PW_C	10	200	ns
Minimum Reset Pulse Width, RST	PW_R	10	50	ns
Setup Time				
Instruction	t_{IS}	10	125	ns
DATA	t_{DS}	10	50	ns
Hold Time				
Instruction	t_{IH}	10	0	ns
DATA	t_{DH}	10	30	ns

($T_A = 25^oC$; $t_r = t_f = 20$ ns for X and I inputs; $C_L = 50$ pF for JMP, X, RR, FLAG0, FLAGF; $C_L = 130$ pF + 1 TTL load for DATA and WRITE.)

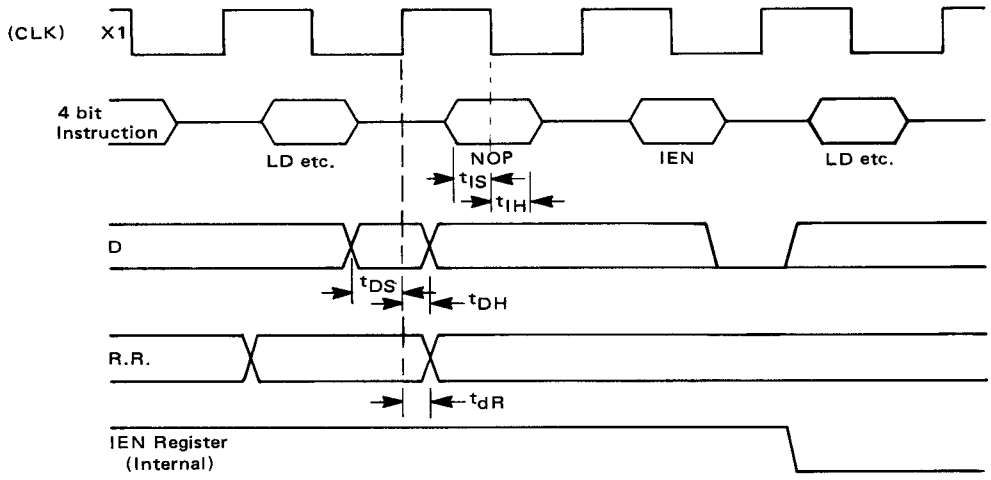
Figure 1.7



Instructions NOPO, NOPF

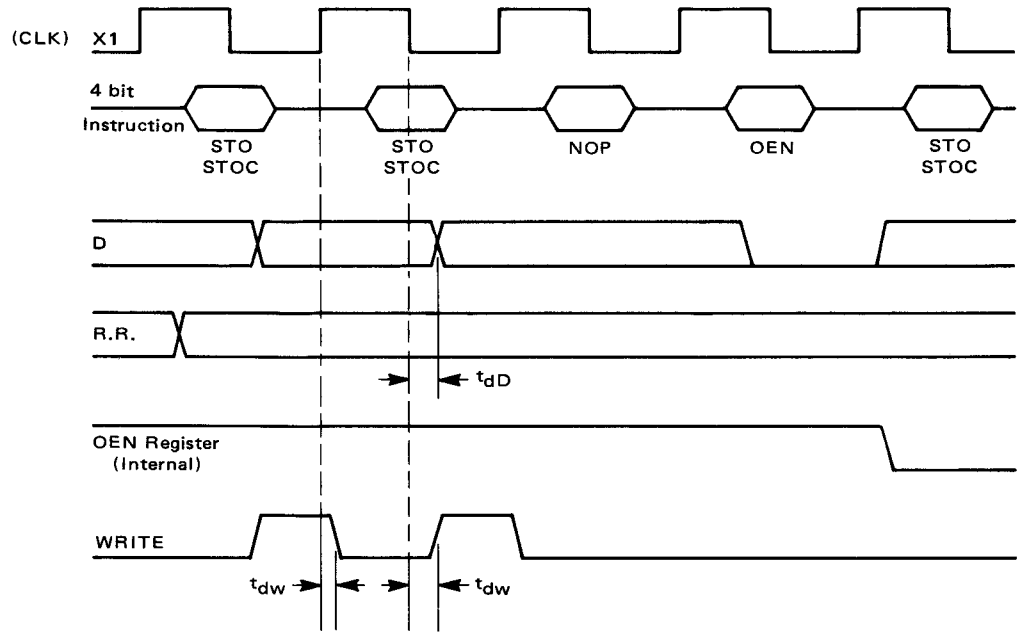
RR, IEN, OEN remain unaffected

Figure 1.8 Timing Waveforms



Instructions LD, LDC, AND, ANDC, OR, ORC, XNOR & IEN

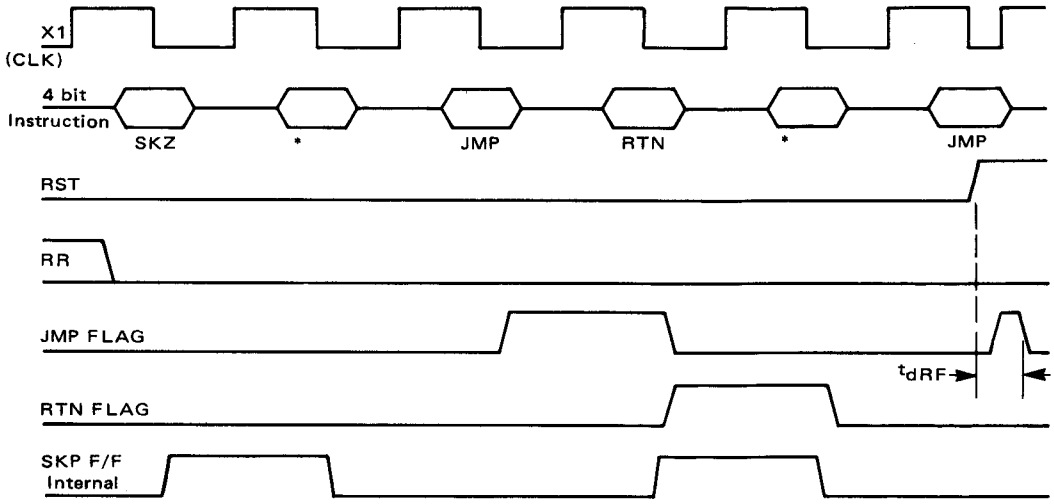
Valid when RST = L



Instructions STO, STOC, OEN

Valid when RST = L

Figure 1.8 Timing Waveforms (Continued)



* Instructions ignored.

Instructions SKZ, JMP, RTN

RR, IEN, OEN remain unaffected.

Figure 1.8 Timing Waveforms (Continued)

CHAPTER 2 BASIC CONCEPTS

The block diagram in Figure 2.1 shows an example of a small ICU-based PLC system. The components, in addition to the ICU, are composed of standard CMOS parts, except for the memory.

The ICU system operates on the principle of a stored program processor. A set of commands, called instructions, reside in the memory of the ICU system. Each command instructs the ICU system to perform one of 16 operations.

The system "fetches" a command, and the necessary information to execute the command, from memory, then "executes" the command. After executing a command, the next sequential command is "fetched" from memory, and the process is repeated ad infinitum.

LD & STO Commands

A typical command might be, LOAD (abbreviated LD). This command instructs the ICU system to read the logic level (logic 1 or logic 0) of an input and store this information in the Result Register within the ICU. To use the LD command, the user programs the memory with the LD instruction and the address of the input to be sampled. The operation of the

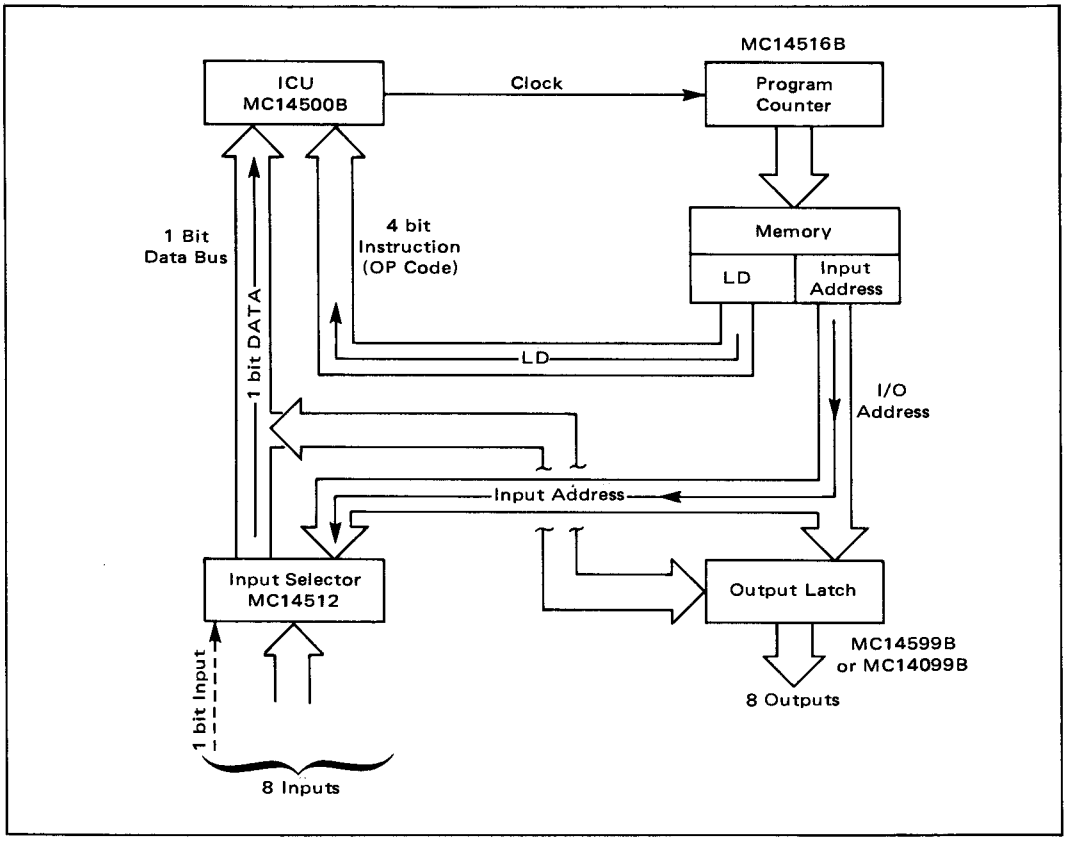


Figure 2.1 Typical Small System Organization & Data Flow

system is as follows: The system memory supplies the ICU with the LD instruction (the instruction is fetched), and supplies the input selector with the address of the input to be sampled. The logic level of the selected input is then transferred over the ICU's one bit data bus to the 1 bit Result Register. (See Figure 2.1).

Another typical command is STORE (abbreviated STO). This command instructs the ICU system to transfer the data contained in its 1 bit Result Register to an output latch. To use the STO command, the user programs the memory with the STO instruction and the address of the output latch which is to receive the data. The operation of the system is as follows. The system memory supplies the ICU with the STO instruction and supplies the output devices with the address of the selected output latch. The data in the ICU's Result Register is then routed to this latch over the ICU's one bit data bus. (See Figure 2.2).

Thus, data can be brought into the system, and also sent out of the system.

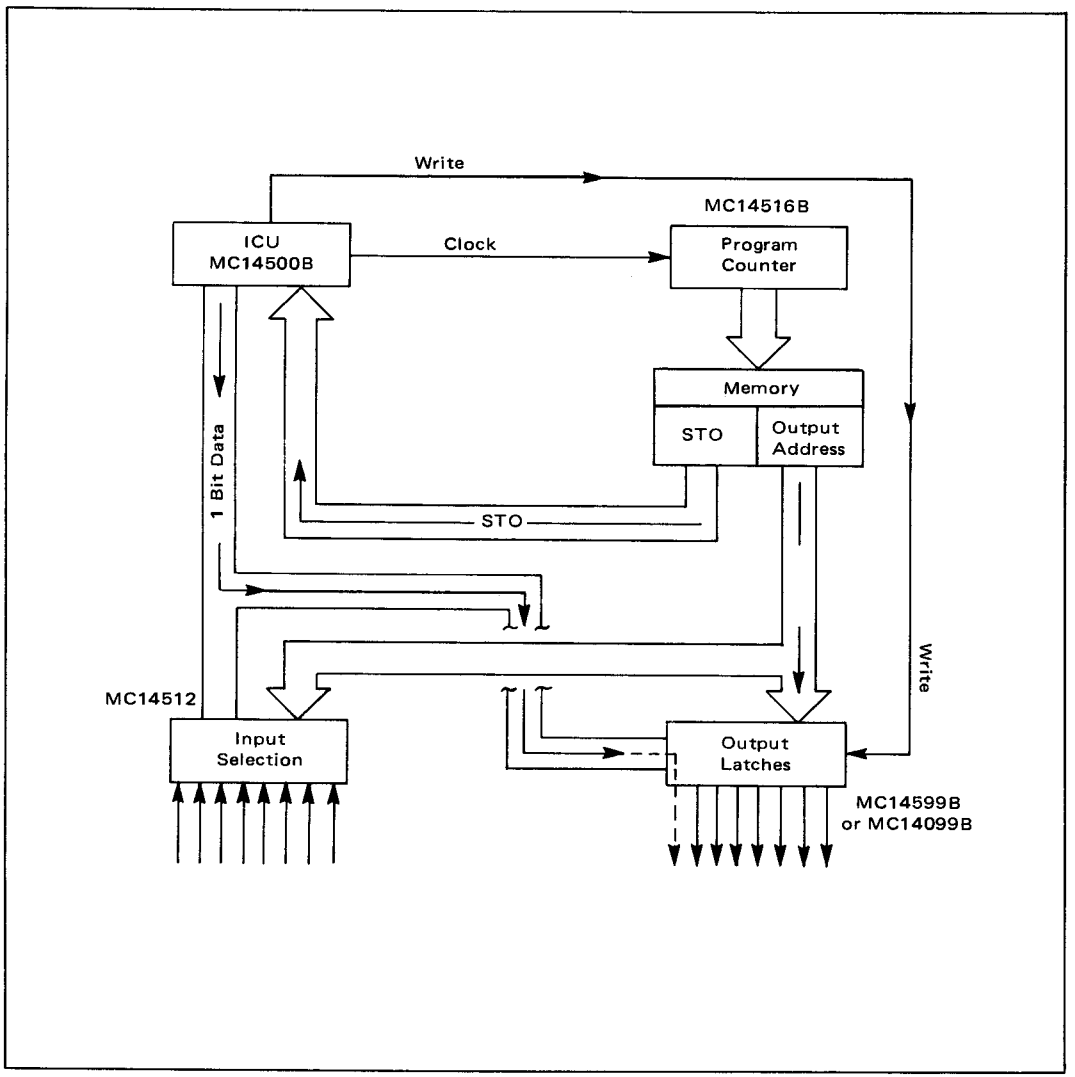


Figure 2.2 System Operation of STO Instruction

SYSTEM COMPONENTS

Memory

The system memory (see Figure 2.3) contains the program which instructs the system to perform its assigned tasks. This program consists of instructions to the ICU in the form of 4 bit operation codes (op-codes) and addresses. The addresses (in binary number form) route the data to and from the ICU's 1-bit bidirectional data bus to the input and output devices.

ICU

The ICU is the central control unit in the system. It controls the flow of data between its internal registers and its 1-bit bidirectional data line, performs logical operations between data in its Result Register and data on its 1-bit bidirectional data line, and sends control signals to the other system components to coordinate the operation of the system.

Program Counter

The program counter (PC) supplies the ICU system memory with the address of the command to be executed. The PC counts up sequentially in binary to its highest value and "wraps around" to zero and counts up again. This causes the sequence of commands in memory to be repeated creating what is known as a looping program.

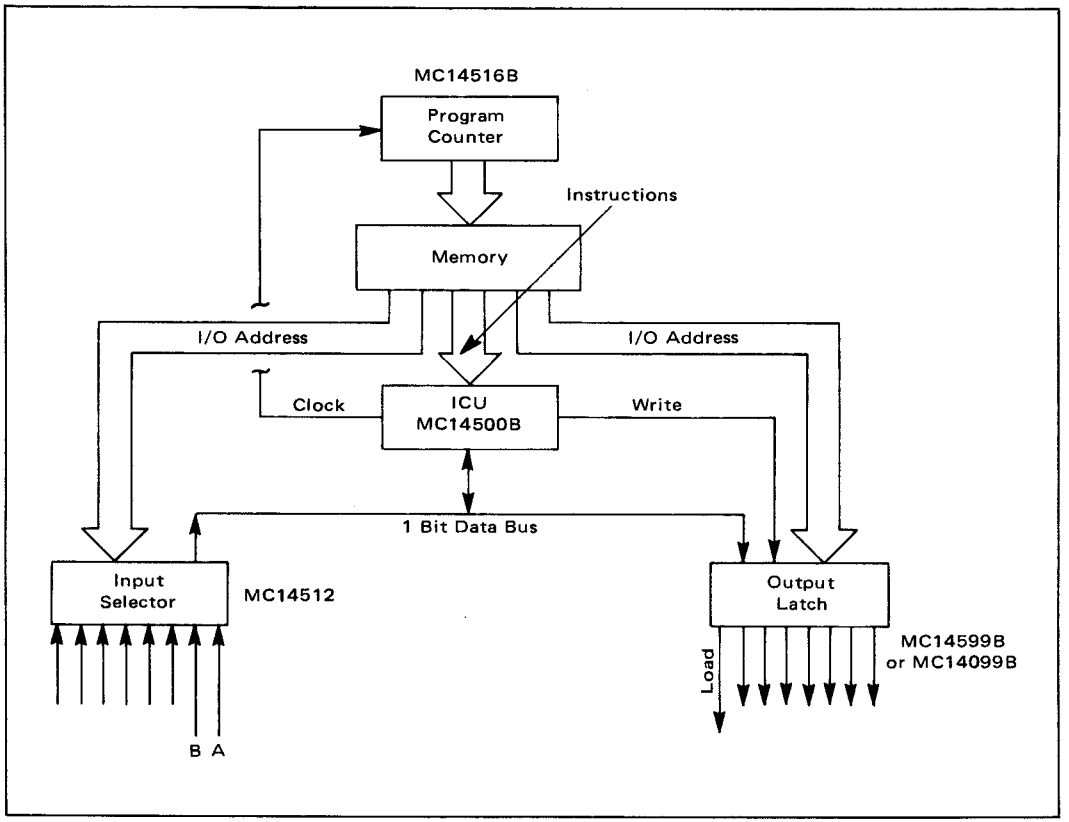


Figure 2.3 Basic System

Input Selectors

The input selectors are used to decide which of the inputs will be used in a particular operation. The ICU system memory supplies the input selectors with the address of the input, then the selector demultiplexes this data onto the ICU's 1-bit bidirectional data line for use by the ICU. Thus, one input is selected from the many inputs.

Output Selectors

The output latches are very similar to the input selectors except the data flow is reversed. When the ICU receives a command to store its Result Register data, it transfers this data to its 1 bit bidirectional data line and signals the output latch with the WRITE control line. The output device then routes this data to the latch specified by the address coming from memory.

The AND Instruction

Before continuing with an example, one more instruction is required — the AND instruction. The operation of the AND instruction is as follows. The system memory supplies the ICU with the AND instruction op-code and supplies the input selectors with the address of an input. The addressed input data is then demultiplexed onto the ICU's bidirectional data line. The information on this line is then logically "ANDed" with the data which is residing in the Result Register. The result of this operation becomes the new content of the Result Register. Notice that the final content of the Result Register will be a logic 1 if and only if the previous content of the Result Register was a logic 1 and the input data was a logic 1. The truth table is:

Input	"AND"	Initial Result Register Contents	=	New Result Register Contents
0		0		0
0		1		0
1		0		0
1		1		1

Example

The basic system of Figure 2.3 is well suited to solving problems presented in the form of relay ladders or solid state logic. Figure 2.4 shows the problem LOAD = A·B in both these forms.

Thus, when A and B are closed (or a logical 1), LOAD is energized (a logical 1).

The ICU solves this problem not once and once only, but once per program loop. Thus, if there are 1000 instructions in the program and the clock frequency is 500 KHz, the inputs will be sampled 500 times per second (every 2 mS) and the output will be energized or de-energized within 2 mS of an input changing. This is known as a looping control structure.

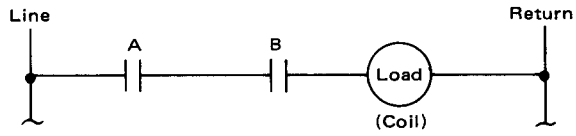


Figure 2.4A Relay Ladder Rung

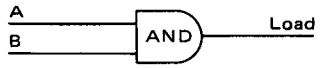
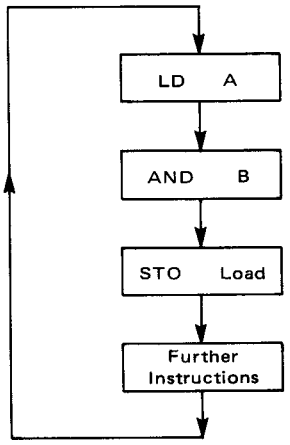


Figure 2.4B Solid State Equivalent of Figure 2.4A

Figure 2.4 Load = A • B

Figure 2.5 shows the ICU program required to solve this problem.

Of course, the sequence could just as readily have been: LD B; AND A; STO LOAD.



Loads the state of input A onto the ICU's Result Register (RR)

Logically ANDs the state of input B with the data in the ICU Result Register (which now contains A). The result of this operation becomes the new contents of the Result Register.

Transfers the data in the RR to the output designated LOAD, thus activating or deactivating the load device

Performs remainder of program and loops back

Figure 2.5 Example LOAD = A • B Program

CHAPTER 3 BASIC PROGRAMMING AND INSTRUCTION SET

Accumulating Result Register

The reader will note that the AND instruction introduced the concept of an accumulating Result Register. In the execution of this instruction, the ICU logically performed an AND function on the data on its bidirectional data line with the data in its internal Result Register. The result of this operation became the new content of the Result Register. The point to be made here is that the Result Register always receives the result of any of the ICU's logical instructions. The Result Register therefore accumulates the logical result of each ICU logical instruction. This is analogous to an adding machine which always displays the subtotal after each operation.

Complement Instruction

It is sometimes desirable to activate an output when one input is in the logic 0 state and another input is in the logic 1 state. This situation occurs in relay controlled systems where "normally closed" relays are used, and occurs in solid state logic systems where inverters are present. Figure 3.1 shows an example of this situation.

The ICU instruction set is prepared for this event. Several logical "complement" instructions invert the logic level of the data on the ICU's bidirectional data line before operation on this data.

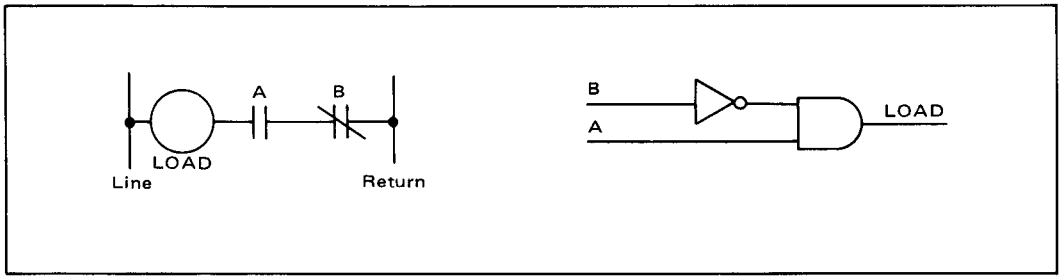
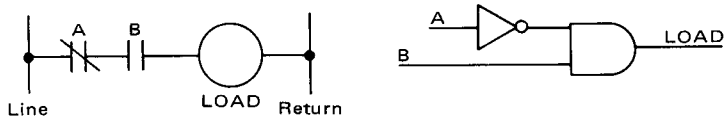
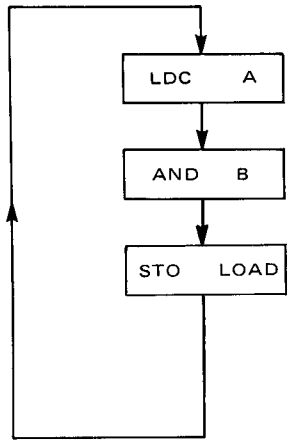


Figure 3.1 Examples of Complemented Signals

The LDC Instruction

An example of one of these instructions is the load complement instruction, abbreviated (LDC). The operation of this instruction is as follows. The ICU system memory supplies the ICU with the LDC instruction and the input selectors with the address of the input to be used in the operation. The input selector then demultiplexes the data of the selected input to the ICU's bidirectional data line. The ICU complements this data and stores the result in its one bit Result Register. The Result Register will receive a logic 1 if the selected input was in the logic 0 state. Figure 3.2 shows an ICU program which solves the problem shown in Figure 3.1, using the LDC command. The reader should be convinced of the operation of this program before reading further.





Loads the logical complement of the A input into the Result Register.

Logically AND's the B input with the content of the Result Register (which contains the complement of the A input). The result of this operation becomes the new content of the Result Register.

Transfers the Result Register data to the output latch designated LOAD.

Note that the STO instruction will only transfer a logic 1 signal to the output latch if the A signal is logic 0 and the B signal is logic 1.

Figure 3.2 Using the LDC Command

The ANDC Instruction

Another example of a logical complement instruction is the "and complement" instruction abbreviated (ANDC). The operation of the ANDC instruction is as follows. The ICU system memory supplies the ICU with the ANDC instruction and the input selectors with the address of a selected input. The input selector then demultiplexes this data onto the ICU's one bit bidirectional data line. The ICU complements this data and logically AND's this data with the data in the Result Register. The result of this operation becomes the new content of the Result Register. The Result Register will receive a logic 1 if the input selected was at logic zero and the Result Register previously contained a logic 1.

With the addition of this instruction the ICU is able to attack some more complicated "chain" calculations. Figure 3.3 shows one such example. Figure 3.4 shows an ICU program which solves the problem depicted in Figure 3.3.

In reviewing the operation of these instructions, the reader should be convinced that the load device will only receive a logic 1 signal if A = 1, B = 0, C = 1, and D = 0.

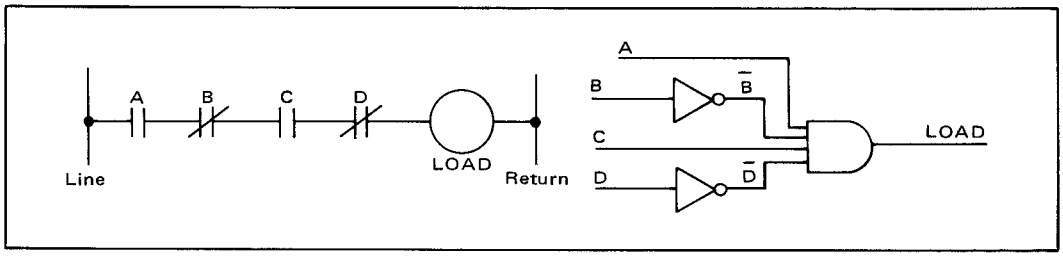


Figure 3.3 Example of a Chain Calculation

Statement	Operator	Operand	Comments
#1	LD	A	Result Register ← A
#2	ANDC	B	Result Register ← A • $\bar{B}$
#3	AND	C	Result Register ← A • $\bar{B}$ • C
#4	ANDC	D	Result Register ← A • $\bar{B}$ • C • $\bar{D}$
#5	STO	LOAD	Result Register = A • $\bar{B}$ • C • $\bar{D}$ → LOAD

Figure 3.4 Program to Solve the Chain Calculation of Figure 3.3

OR and ORC

In many cases, it is also desirable to activate an output when either input is in the logic 1 state. In this event, the "or" instruction, (OR), should be used. The operation of the OR instruction is as follows. The ICU system memory supplies the OR instruction to the ICU and the address of the input to be used in the operation to the input selectors. The input selector then demultiplexes the addressed data onto the ICU's bidirectional data line. The ICU then logically OR's this data with the content of the ICU's Result Register and returns the result of the operation to the Result Register.

The ICU also has an "or complement" instruction, abbreviated ORC, in the event complement logic is needed. The operation of this instruction is exactly like the OR instruction *except* the incoming data is complemented before the OR operation is performed. Figure 3.5 shows some examples where the OR and ORC instructions may be used.

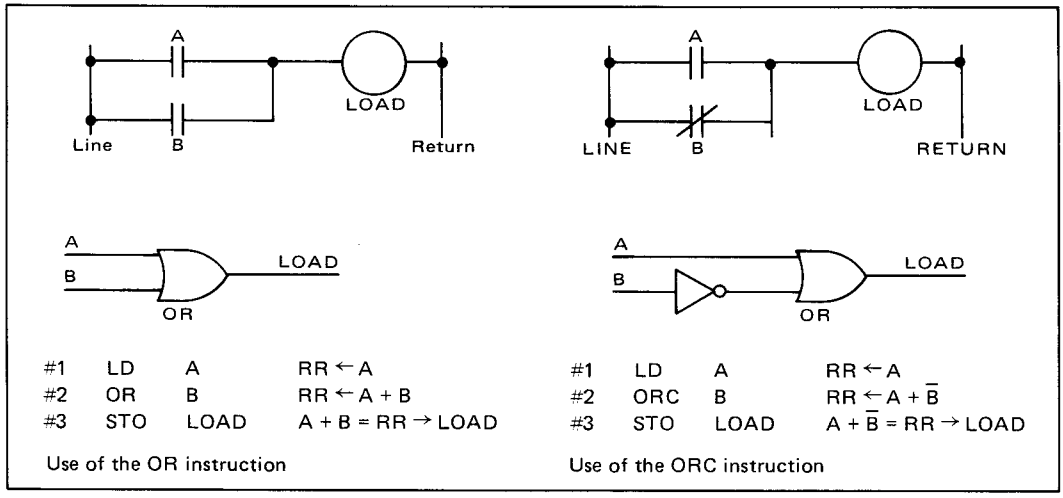


Figure 3.5 Use of the OR and ORC Instructions

In the example of using the OR instruction, the load device will receive a logic 1 signal if the A or B or both inputs are in the logic 1 state. In the example of the ORC instruction, the load device receives a logic 1 signal if the A input is in the logic 1 state or the B input is in the logic 0 state.

Use of Temporary Locations

Many of the logic structures found in the controls industry are branches of several series relays, in parallel with another branch of series relays. Figure 3.6 shows an example of this structure.

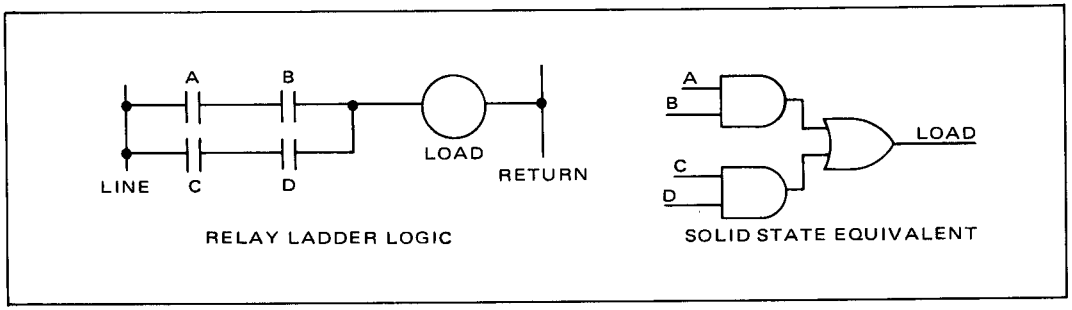
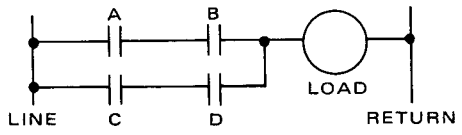


Figure 3.6 Series-Parallel Combinations

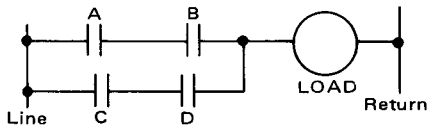
When dealing with this type of problem, it is not always possible to directly "chain" a series of LD, LDC, AND, ANDC, OR, and ORC instructions together to correctly evaluate the logic function required. In some cases, it may be necessary to temporarily store the intermediate results before processing the remainder of the problem. In these cases, the programmer must evaluate the series branches using LD, AND, and ANDC instructions as necessary to evaluate the expression and then store the result in a temporary location. The second series branch must then be evaluated and ORed with the data saved in the temporary location. The result of this operation should then be used to activate or deactivate the load device. Figure 3.7 shows a common error in programming this type of problem and Figure 3.8 describes and the correct approach to the problem. Figure 3.8 shows the correct method for solving this problem by using a temporary storage location.



#1	LD	A	RR ← A	
#2	AND	B	RR ← A • B	**ERROR
#3	OR	C	RR ← A • B + C	
#4	AND	D	RR ← (A • B + C) D	
#5	STO	LOAD	RR → LOAD	

**Note that the final expression incorrectly resulted in the D term being distributed across all other terms. For example, if A, B and C are logic 1 and the D input is logic 0, the load device would receive a logic 0; this is incorrect because the load device should be activated when the A and B inputs are logic 1.

Figure 3.7 Example of Incorrect Programming



#1	LD	A	RR ← A
#2	AND	B	RR ← A • B
#3	STO	TEMP	RR = A • B → TEMP
#4	LD	C	RR ← C
#5	AND	D	RR ← C • D
#6	OR	TEMP	RR ← C • D + (TEMP = A • B) = A • B + C • D
#7	STO	LOAD	RR = A • B + C • D → LOAD

In this program, the logical result of ANDing A and B is stored temporarily, then the logical AND of C and D is ORed with the data previously stored in the temporary location. The correct logical signal is then transferred to the load device. This example demonstrates the need for temporary storage locations before proceeding.

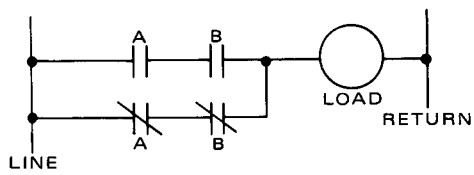
Figure 3.8 Correct Method of Solving the Problem

The XNOR Instruction

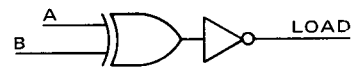
The "exclusive nor" instruction, abbreviated (XNOR), is the final logical instruction in the ICU's repertoire of logical instructions. The XNOR instruction can be thought of as a "match" instruction. That is, whenever the input data is identical to the data in the Result Register, the new content of the Result Register will be a logic 1. Figure 3.9 shows the truth table for the XNOR function and Figure 3.10 an example using the XNOR function. Note the reduction in code that may result from the use of this instruction.

Input Data	Old Result Register Data	New Result Register Data
0	0	1
0	1	0
1	0	0
1	1	1

Figure 3.9 XNOR Truth Table



EQUALS



#1	LD	A
#2	AND	B
#3	STO	TEMP
#4	LDC	A
#5	ANDC	B
#6	OR	TEMP
#7	STO	LOAD

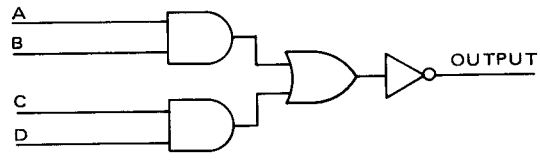
EQUALS

LD	A
XNOR	B
STO	LOAD

Figure 3.10 Example of use of XNOR Instruction

The STOC Instruction

When transferring a signal to activate a load device, it is very useful to be able to store the logical complement of an expression. The ICU therefore has a "store complement" instruction, abbreviated (STOC). The STOC instruction is exactly like the store (STO) instruction, except the logical complement of the Result Register is transferred to the output latch. It should be pointed out that the Result Register retains its original value (i.e. the STOC does not change the Result Register value, it merely transfers the complement of the Result Register to the bidirectional data line for routing to the output latches). This instruction is quite useful when dealing with negative logic or so called "low active" devices. Figure 3.11 shows an example usage of the STOC instruction. Figure 3.12 shows a problem in both the relay ladder and logic formats. Figure 3.13 shows the problem reduced to code.



#1	LD	A	RR ← A
#2	AND	B	RR ← A • B
#3	STO	TEMP	A • B → TEMP
#4	LD	C	RR ← C
#5	AND	D	RR ← C • D
#6	OR	TEMP	$\overline{RR \leftarrow C \cdot D + A \cdot B}$
#7	STOC	OUTPUT	A • B + C • D → OUTPUT

Figure 3.11 Example of the STOC Instruction

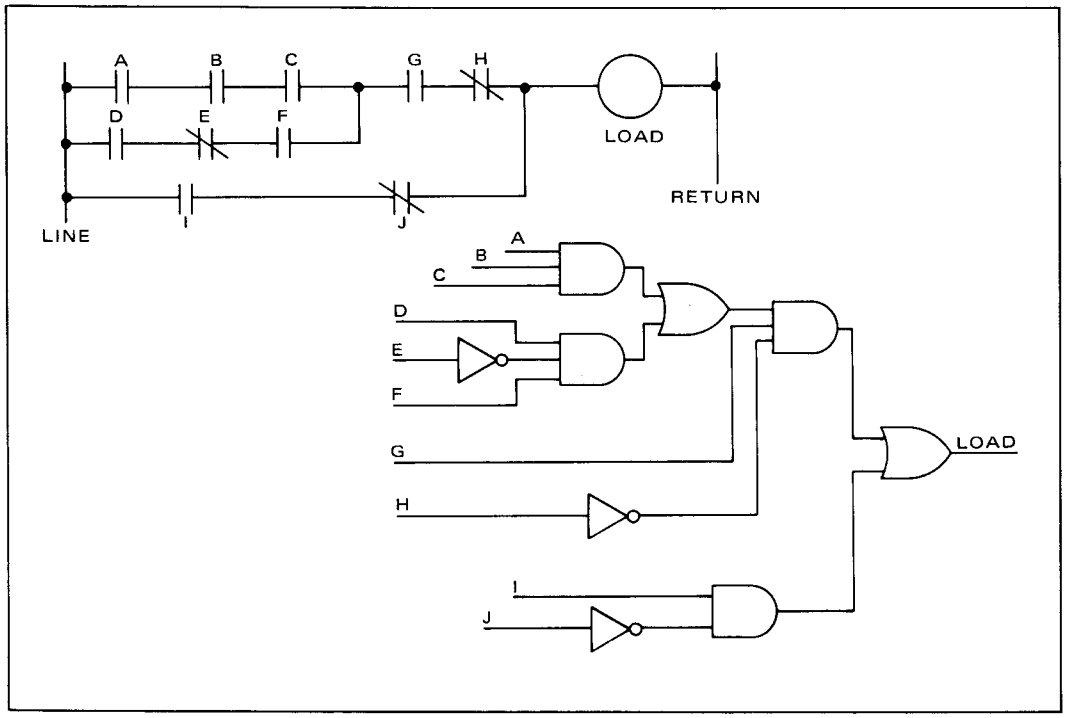


Figure 3.12 Complex Problem

#1	LD	A	RR ← A
#2	AND	B	RR ← A · B
#3	AND	C	RR ← A · B · C
#4	STO	TEMP	A · B · C → TEMP
#5	LD	D	RR ← D
#6	ANDC	E	RR ← D · $\overline{E}$
#7	AND	F	RR ← D · E · F
#8	OR	TEMP	RR ← A · B · C + D · E · F
#9	AND	G	RR ← (A · B · C + D · E · F) · G
#10	ANDC	H	RR ← (A · B · C + D · E · F) · G · $\overline{H}$
#11	STO	TEMP	(A · B · C + D · $\overline{E}$ · F) · G · $\overline{H}$ ← TEMP
#12	LD	I	RR ← I
#13	ANDC	J	RR ← I · $\overline{J}$
#14	OR	TEMP	RR ← (A · B · C + D · $\overline{E}$ · F) · G · $\overline{H}$ + I · $\overline{J}$
#15	STO	LOAD	(A · B · C + D · $\overline{E}$ · F) · G · $\overline{H}$ + I · $\overline{J}$ → LOAD

Figure 3.13 Complex Example Problem Code

The Enabling Instructions, IEN and OEN

In addition to the ICU's logic instructions, the ICU provides two instructions for controlling the program flow in a looping control structure. The reader will remember that, in a looping control structure, each instruction is fetched from memory in sequential order. In some instances, it may be desirable to effectively "jump" over a certain section of the ICU program or to inhibit input data from effecting the system's output.

IEN

The first of these instructions is the "input enabling" instruction, abbreviated (IEN). The operation of the input enabling instruction is as follows. The ICU system memory supplies the ICU with the IEN instruction and the input selectors with the address of the selected input to be used. The input selector demultiplexes the data of the addressed input onto the ICU's bidirectional data line. The ICU then latches the input data into its "input enabling" register. If the input enabling register is loaded with a logic 0, all future input data will be interpreted as logic 0 until the IEN register is loaded with a logic 1 by another IEN instruction. This instruction can be used in a manner similar to the way "master contacts" are used in relay ladder logic. Figure 3.14 shows an example usage of the IEN instruction.

Note, (statement #5), that if the IEN register was loaded with a logic 0 the Result Register can only receive logic 0 data because only LD and AND instructions are used to decide if the load device will be activated.

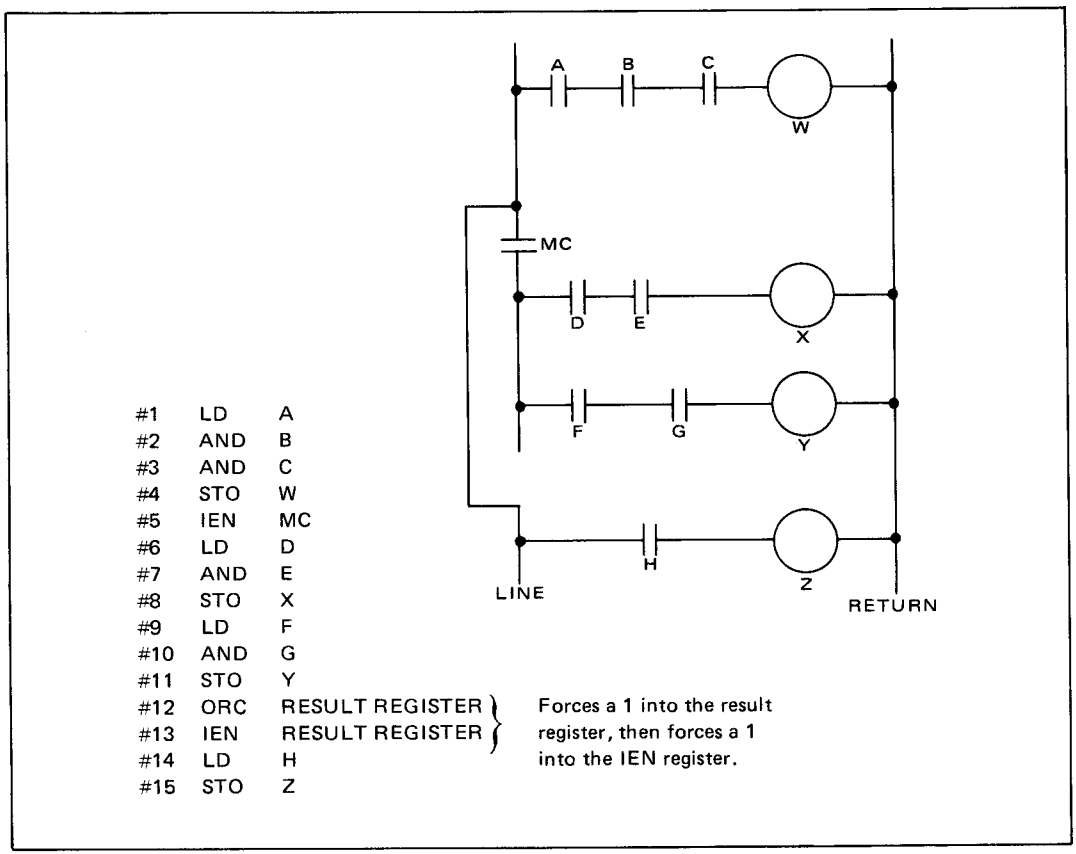


Figure 3.14 Example of Using the IEN Instruction

Caution

Care must be taken using the IEN instruction properly; remember that when the IEN register contains a logic 0 all input data for the ICU will be interpreted as logic 0. This can be tricky. For example, assume the IEN register contains a logic 0. If either an LDC or an ORC instruction is executed, the Result Register will receive a logic 1 regardless of the actual state of the inputs. Additional care must be taken to reload the IEN register with a logic 1 after executing the block of code to be controlled by the IEN register. In the example of Figure 3.14, this is done in statements 12 and 13. Statement 12 forces the Result Register to logic 1 and statement 13 loads the IEN register from the Result Register. Notice that the Result Register data is "pinned out" on the MC14500B and is here assumed to be connected to one of the inputs of the system. In most systems, this connection will be made.

OEN

The second ICU instruction for controlling the operation of programs in a looping control structure is the "output enabling instruction," abbreviated OEN. The operation of the OEN instruction is as follows. The ICU system memory supplies the ICU with the OEN instruction and the input selectors with the address of a selected input. The input selectors then demultiplexes the data of the addressed input onto the ICU's bidirectional data line. The ICU then latches this data into its output enabling register (OEN). If a logic 0 is loaded into the OEN register, the WRITE control signal from the ICU is inhibited. Therefore, the output selectors cannot be instructed to receive new data and remain unchanged by STO and

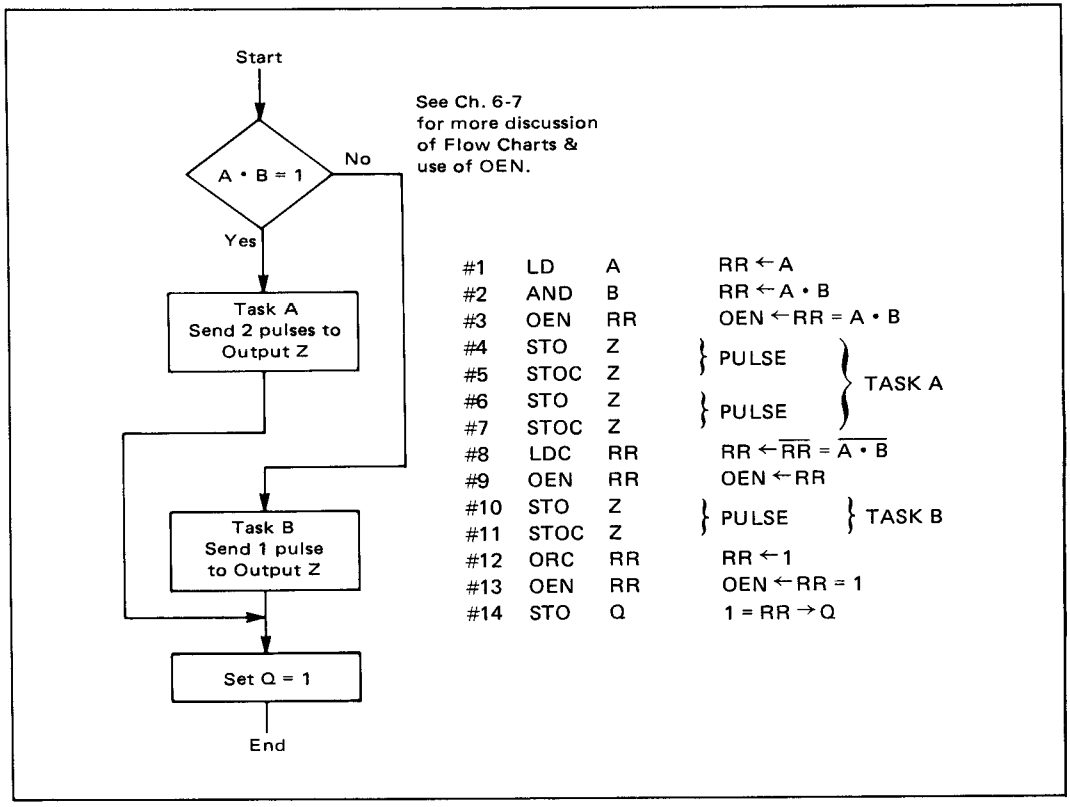


Figure 3.15 Example Use of the OEN Instruction

STOC instructions. The key point is that once the OEN register is loaded with a logic 0, the system outputs remain in their present state until the OEN register is loaded with a logic 1 by another OEN instruction. Then and only then can the system outputs be changed by STO and STOC instructions. Using the OEN instruction, the programmer can effectively "jump" over a block of code by conditionally setting the OEN register to logic 0, causing subsequent instructions to have no effect on the system outputs. The programmer can then set the OEN register back to logic 1 so that future ICU code will operate in normal fashion. Figure 3.15 shows an example use of the OEN instruction. In the example, the program again assumes that the Result Register (RR) is available as a system input. Chapters 7 through 10 describe the OEN structures in greater detail.

In the example of Figure 3.15, if A and B are both true statements 4 through 7 send two pulses to output Z and statements 10 and 11 will not influence output Z effecting a "pseudo" branch around these instructions. If the tested condition fails, statements 4 through 7 will have no effect on the output and statements 10 and 11 will send one pulse to output Z. Statements 12 and 13 return the OEN register to logic 1 so that the Q output will receive a logic 1, and future code will operate in normal fashion. Much more will be said about the use and advantages of an OEN instruction in Chapters 7, 8, and 9.

Thus far, we have studied the LD, LDC, AND, ANDC, OR, ORC, XNOR, STO, STOC, IEN, and OEN instructions. Of the remaining five instructions, two are no operation (NOP) instructions and the other three are for optional use in larger systems which do not have a looping control structure. These will be discussed later.

CHAPTER 4 HARDWARE SYSTEMS

The purpose of this chapter is to begin to acquaint systems designers with the components which are used in a basic ICU looping control system. The system illustrated was not specifically intended to be used in a practical design, however, it illustrates how the components, which comprise the building blocks of an ICU system, might be used. From this point, the system designer can delete, augment or otherwise modify the system illustrated to his own particular needs.

Figure 4.1 is a schematic diagram of a small ICU based system. The system has a looping control structure (i.e. the program counter is never altered by any operation of the ICU.)

System Features

The scheme depicted on Figure 4.1 is a PLC-like system, designed to operate on the principle of a looping control structure. It has 8 inputs, 8 outputs and 8 additional outputs which can be "read" back by the ICU. These outputs can be used for temporary storage. The system memory is capable of holding two separate ICU programs; each individual program can be 256 ICU statements long.

Program Counter

The program counter is composed of two MC14516B binary up-counters chained together to create 8 bits of memory address. This gives the system the capability of addressing 256 separate memory words. The counters are configured to count up on the rising edge of the ICU clock (CLK) signal and reset to zero when the ICU is reset. Notice that the program counter count sequence cannot be altered by any operation of the ICU. This confirms that the system is configured to have a looping control structure.

Memory

The memory for this system is composed of one MCM7641 512-word by 8 bit PROM memory. Because the program counter is only 8 bits wide, only 256 words, (half of the memory), can be used at any one time. However, by wiring the most significant bit of the memory's address high or low, the system designer can select between two separate programs with only a jumper option. This might be a desirable feature if extremely fast system changes are required. Optionally, the designer could chain another counter chip or a single, divide-by-two of flip-flop to the program counter and use the additional memory space for more programming statements. If less than 256 program statements are needed and fast turn around is not needed, a smaller memory may be more economical.

Figure 4.2 shows the format of each memory word. The most significant 4 bits contain the instruction operation code which is routed to the ICU. The 4 least significant bits are routed to the system's input selectors and output latches to address the system's inputs, outputs, and "readable" outputs.

Memory Options

There are, of course, many ways to configure the memory of an ICU system.

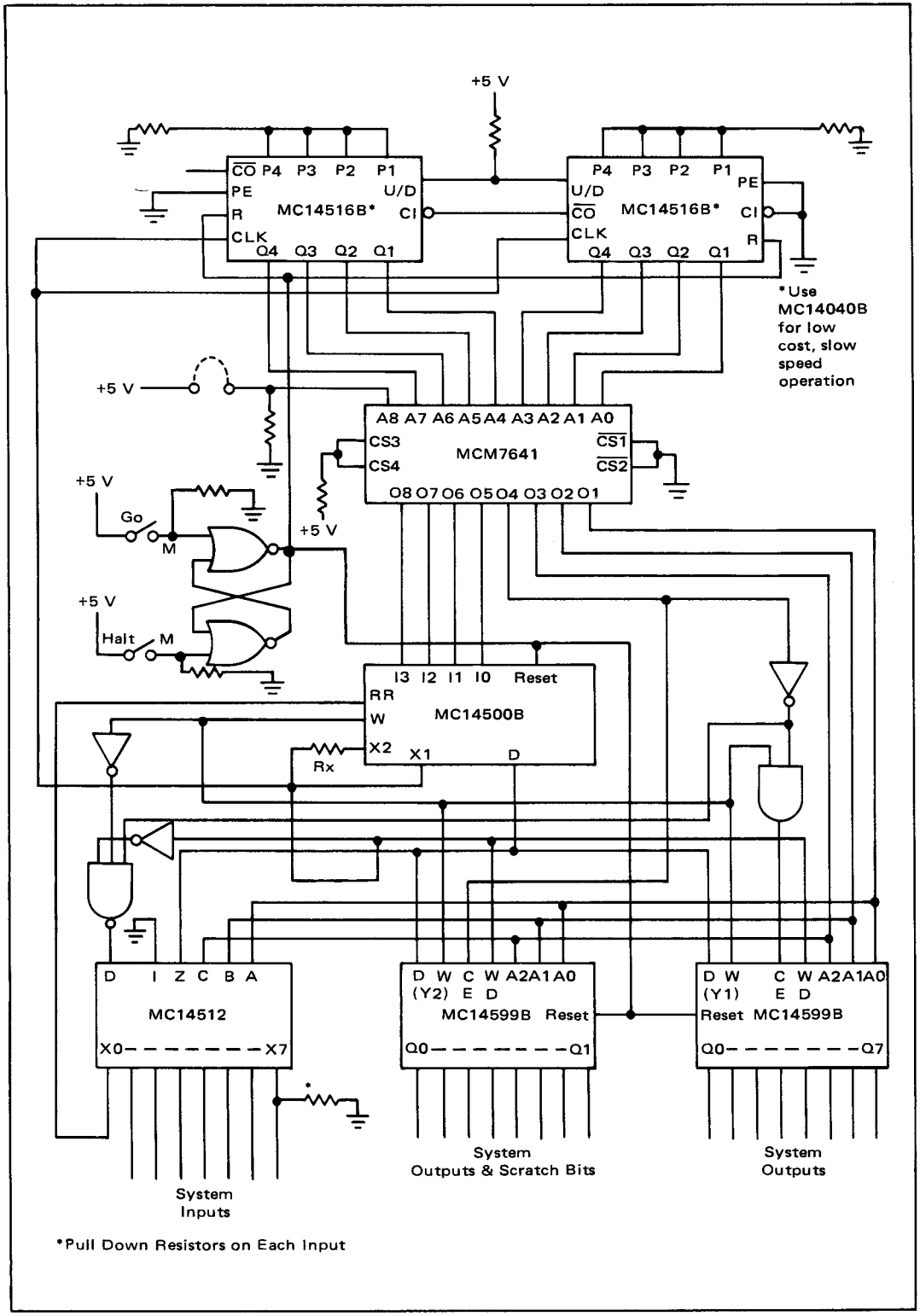


Figure 4.1 A Minimal ICU System

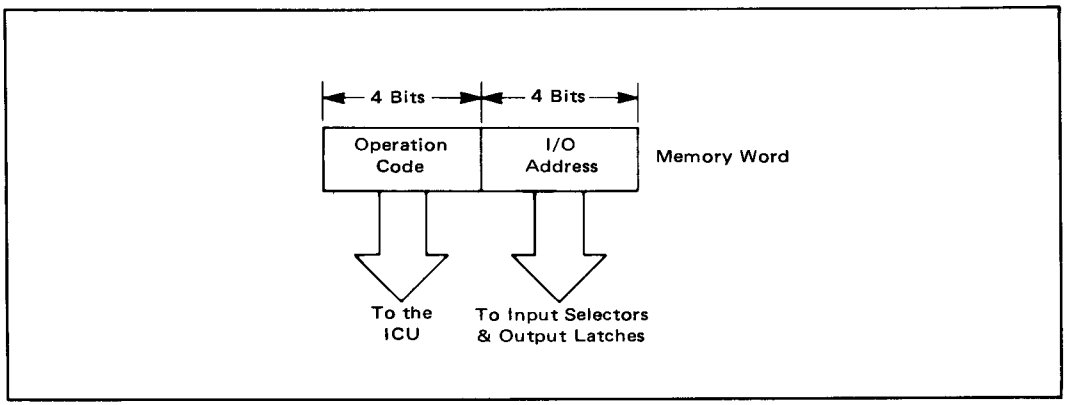


Figure 4.2 Parallel Memory Word Format

Expansion

Figure 4.3 shows a simple approach to expanding the I/O address capability of an ICU system. In this approach, the system memory is broken into two separate sections which share common address lines and bring their data out in parallel. The first of these memories is an N by 4 bit "instruction memory," used to hold only ICU instructions. The MCM7643 1K by 4 bit PROM is capable of holding 1024 ICU instructions and would be a good choice for problems requiring moderate length programs. The second memory is an N word by M bit "address memory" used to hold the address of the operand for each ICU instruction. The MCM7641 512 by 8 bit PROM is a good choice for this application. Two MCM7641, 512 by 8 bit memories and one MCM7643 1K by 4 bit memory would comprise an ICU system memory capable of holding 1024 complete ICU program statements and be capable of addressing 256 inputs and 256 outputs.

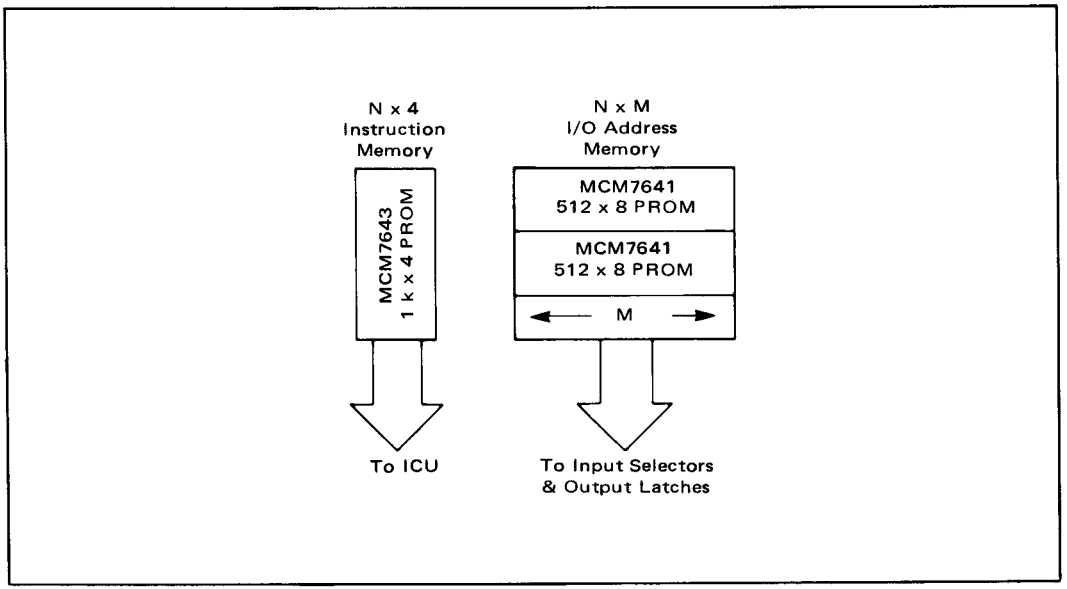


Figure 4.3 An Approach to I/O Address Expansion

Using 4-Bit Wide Memories

It is also possible to "interlace" the instruction operation codes with the I/O addresses in the same memory. In this type of structure, the CLK signal will become the least significant address bit. When the clock signal is high, the memory supplies the ICU with an instruction which will be latched into the ICU on the falling edge of the CLK signal. The memory is then free to supply the I/O sections of the ICU system with an address when the clock signal is low. Figure 4.4 shows this. Thus, a 4 bit wide memory may contain the instructions and addresses for 16 inputs and 16 outputs. This method is used in the demonstration system. Note that as the clock-high and clock-low signals are still used, there is no time penalty involved.

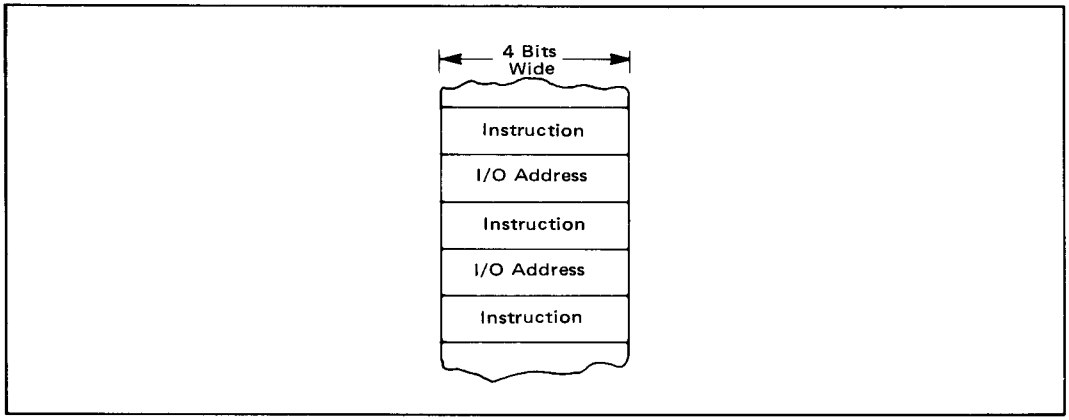


Figure 4.4 Interlaced Memory

Hybrid Expansion

It is also possible to interleave with 8 bit wide memory and thus create a 12 bit wide (4096) I/O structure. See Figure 4.5.

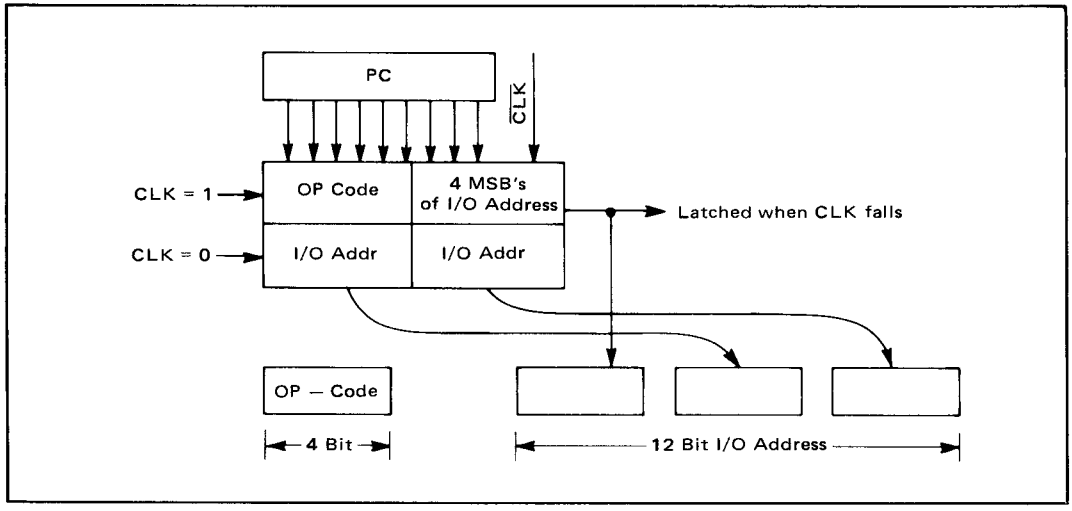


Figure 4.5 Interlaced 8 Bit Memory

Input/Output Structure

The system shown in Figure 4.1 will be considered in more detail here. Figure 4.6 shows the complete I/O map.

Input Selectors

The input selectors used are MC14512 8-channel data selectors. In the example system of Figure 4.1, there is only 1 MC14512 supplying the system with 8 inputs. These inputs occupy addresses 0 through 7 (see Figure 4.6). The input selectors demultiplex the addressed input onto the ICU's bidirectional data during the CLK low phase of each ICU machine cycle for all instructions except the STO and STOC instructions. The number of inputs can be expanded easily by adding additional address lines, the proper address decode, and timing.

Output Latches

The output latches are composed of MC14599B 8 bit (bidirectional data port) latches. In the example system of Figure 4.1, the MC14599B labeled Y1 is used strictly as an output latch supplying the system with 8 outputs. These outputs occupy addresses 0 through 7. (See Figure 4.6.) The MC14599 labeled Y2 is configured as a "readable" output latch. In this configuration the part can be thought of as an 8 bit RAM with the outputs of each location pinned out. Because this chip has the read/write feature implemented, it occupies space in both the input and output sections of the I/O address map. The assigned addresses are 8 through 15.

The output selectors receive the data coming from the ICU over the ICU's bidirectional data line. The information is transmitted during the clock low phase of a machine cycle when the ICU executes an STO or STOC instruction, provided the OEN register contains a logic 1. The ICU signals the output latches that a STO or STOC instruction is being executed. The addressed output latches then receive the data and retain its value until the latch is once again addressed and changed. Again, the number and configuration of the output latches can be expanded easily by adding additional address lines, the proper address decode and timing.

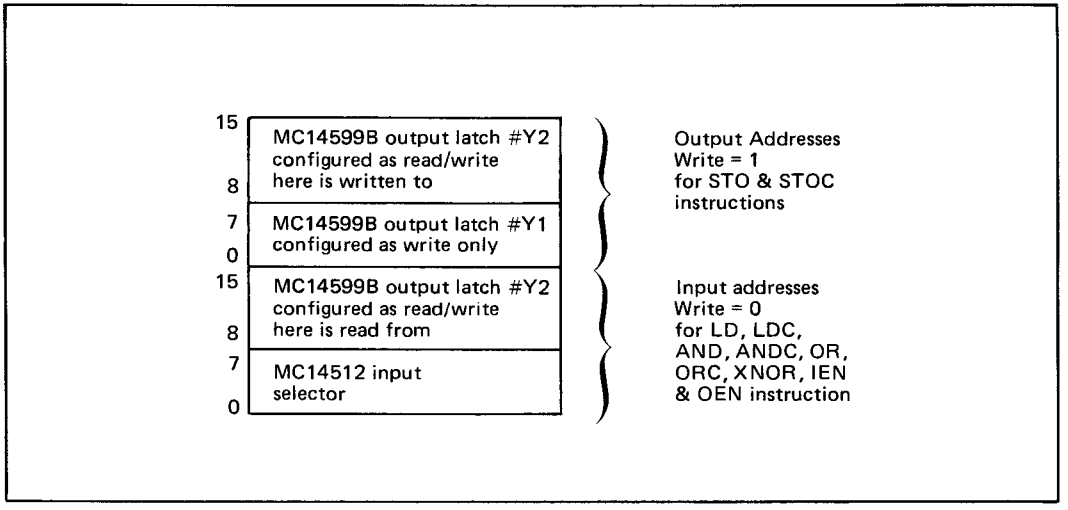


Figure 4.6 I/O Map

I/O Options

In the system shown in Figure 4.1, it may be more desirable to have more system inputs and less temporary storage bits. In this event, the designer can reconfigure the Y2 MC14599B to be a "write-only" output latch. This action would free 8 locations on the I/O address map for 8 more inputs; another MC14512 could be used. The system would then have 16 inputs and 16 outputs. The designer could create temporary storage bits by tying outputs back to inputs. The memory options description showed how memory, and therefore, I/O, may be expanded.

Adding RAM

If the system requires a large number of inputs, outputs and temporary storage bits, it may be more economical to put an N by 1 bit RAM on the data bus rather than using the output latches and input selectors to effect temporary storage bits. See Figure 4.7.

ICU

The MC14500B is the central control element within the system. It coordinates the actions of all the system's components. The system of Figure 4.1 was designed to use the looping control structure of the ICU. In this type of structure, the Result Register is usually tied to one of the system's inputs and in this example, the Result Register is returned to input X_0. The ICU's RESET line is connected to a latch, which is set or reset by two momentary contact switches, giving the system a HALT/RUN feature. Note that when the ICU is halted, the output latches are cleared to zero.

Because the ICU is to be used in a looping control structure, the pulses created by the JMP and RTN instructions are not required. Also, the pulses created by the NOP instructions are not used.

Notice that the ICU has NOP instructions of all 1's or all 0's. This was done because the unprogrammed states of PROM's are all 0's or all 1's. Therefore, in a looping control structure, the ICU can be allowed to sequence through these unprogrammed locations without affecting the logical operation of the system.

Chapter 5 contains an example of an "interlaced" memory system and Chapter 12 contains an example of a hybrid (parallel/interlaced) memory system with a scratchpad RAM.

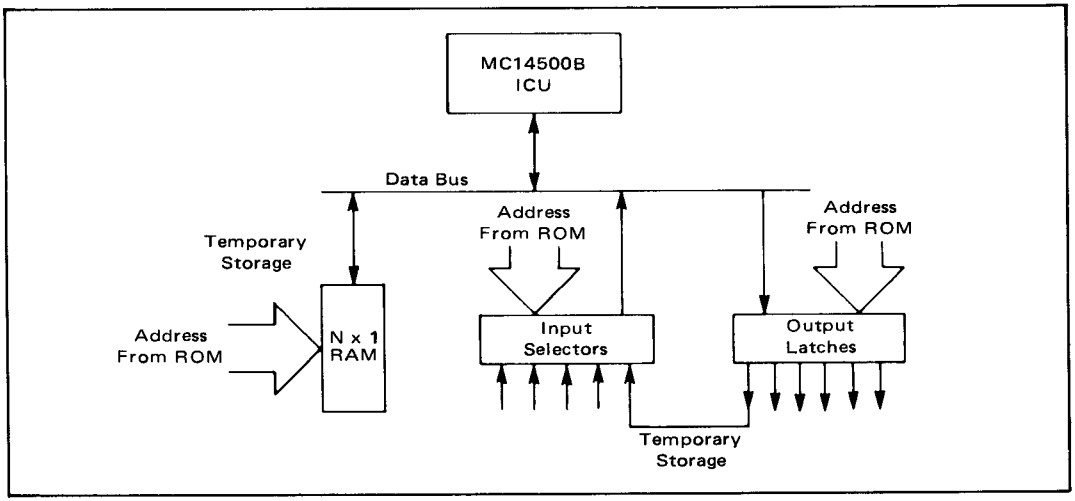


Figure 4.7 Adding RAM to a System