MARC TREMBLAY

CAPTEUR INTÉGRÉ À TRAITEMENT PARALLÈLE AU PLAN FOCAL POUR LA VISION ARTIFICIELLE

Thèse
présentée
à l'École des gradués
de l'Université Laval
pour l'obtention
du grade de Philosophiae Doctor (Ph.D.)

Département de génie électrique
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

AVRIL 1992

RÉSUMÉ

Ce travail concerne la conception et la réalisation d'une caméra spécialisée pour la vision numérique à des fins de reconnaissance de formes et d'automatisation. Cette caméra est réalisée à l'aide de la technologie d'intégration à très grande échelle selon la technologie CMOS et comprend trois principales parties: un capteur bidimensionnel à topologie hexagonale et à extraction parallèle des signaux analogiques, un module de calcul analogique pour le filtrage spatial de l'image et un module de commande micro-codé qui réalise l'interface avec un ordinateur hôte en plus d'effectuer certaines fonctions de traitement d'images.

Ce système de vision permet entre autre d'extraire simultanément les contours des objets à plusieurs niveaux de résolution et de créer une structure de données hiérarchisée décrivant la scène analysée. Un grand nombre de sous-systèmes spécialisés peuvent être greffés au système de base pour en augmenter la performance.

RÉSUMÉ

L'extraction de l'information visuelle d'une scène nécessite le développement de systèmes dédiés efficaces conçus pour opérer en harmonie avec la vision robotique et avec les tâches d'automatisation. Un des besoins spécifiques est lié à la description hiérarchisée d'une scène, procédure qui ne peut pas être efficacement exécutée par une machine numérique conventionnelle. Certaines solutions matérielles peuvent exploiter le calcul en parallèle de façon à effectuer l'acquisition intelligente et performante de l'information visuelle. Un domaine particulièrement pertinent au développement de solutions utilisant la technologie d'intégration à très grande échelle (ITGE ou VLSI) consiste en l'élaboration de nouveaux capteurs photo-sensibles qui offrent un avantage marqué par rapport à l'utilisation conventionnelle des caméras CCD couplées à un module d'échantillonnage d'images. Cette thèse décrit un nouveau type de processeur rétinien à traitement parallèle analogique pour la vision 2D passive.

La plupart des travaux effectués dans le domaine des rétines artificielles sont généralement orientés sur le traitement d'images massivement parallèle au plan focal sans trop se soucier du problème de l'extraction des résultats et de la structure des données générées. Le système de vision avec capteur à Accès Multi-port de photo-Récepteurs (MAR) regroupe la fonction de la prise d'images avec traitement intégré au plan focal ainsi qu'une certaine partie du traitement d'images inhérent aux différentes procédures de description hiérarchique d'une scène. Le design est orienté de façon à générer des séquences de données provenant de l'image à analyser qui sont propices à la création d'une structure hiérarchisée d'information décrivant la scène. L'élément principal du système MAR est un capteur VLSI à topologie hexagonale réalisé à l'aide de la technologie CMOS qui permet la lecture simultanée d'un groupe de pixels à l'intérieur d'une région d'intérêt. La puissance globale du capteur est enrichie par l'ajout d'un module de calcul analogique qui permet d'effectuer, en parallèle, le filtrage de l'image en temps réel selon plusieurs résolutions spatiales. Le système comprend également une unité de commande VLSI

microprogrammée qui agit en boucle fermée pour effectuer la détection du mouvement ou le suivi d'arêtes et pour définir différents modes de balayage de l'image.

Les technologies VLSI CMOS actuelles avec grilles de 1 µm sur des pastilles de silicium de 1.5 cm permettent la réalisation de capteurs à architecture multi-port ayant une résolution spatiale de l'ordre de 500 x 500 pixels et un fréquence d'opération d'environ 5 MHz. Le premier prototype de cette caméra intégrée réalisée au cours de ce projet de recherche à permis de faire la preuve fonctionnelle du concept et d'effectuer la prise d'images de 128 x 128 pixels et de générer les cartes d'arêtes selon 7 différentes résolutions spatiales.

Le système MAR tel que défini dans ce document constitue une configuration de base très intéressante pour effectuer l'acquisition et le pré-traitement d'images en fonction de procédures de reconnaissance de formes en plus d'offrir les propriétés d'un système performant et compact propre aux applications VLSI. On peut espérer que l'émergence de ce nouveau genre de caméra intelligente munie d'une capacité de traitement d'images intégré au plan focal devrait contribuer prochainement au développement de machines de vision moins coûteuses et plus performantes.

Denis Poussart

Marc Tremblay

AVANT-PROPOS

Je tiens à remercier mon directeur de thèse le Dr. Denis Poussart pour sa précieuse collaboration tout au long du développement de ce travail de recherche. Il m'a toujours bien critiqué et a orienté mon travail dans la bonne direction lorsque ce fût nécessaire. Je ne pourrais oublier de mentionner la précieuse collaboration du Dr. Denis Laurendeau en tant que conseiller technique de grande valeur en vision numérique et en tant qu'ami sincère lors des périodes plus difficiles de mon travail de recherche.

Je remercie également les techniciens Jacques Lévesque, Yvon Chalifour et Gaétan Bernier pour la qualité de leurs travail lors de la fabrication des composantes et le montage du prototype de la caméra MAR de même que Marc Blanchet pour le support des nombreux logiciels d'édition et de simulation requis pour la réalisation VLSI. La rencontre technique proposée par M. Michel Têtu avec son collègue M. Robert Mc Entire de la compagnie GE Electro-optics a été très fructueuse pour la mise au point de photo-capteurs efficaces. Je tiens donc à les remercier sincèrement.

Le travail entrepris pour réaliser ce projet a été grandement accéléré par la collaboration de plusieurs étudiants par l'entremise de divers projets de recherche. Je veux principalement remercier Florent Parent pour avoir entrepris un projet de doctorat portant sur le développement d'un logiciel exploitant les capacités de la caméra MAR et pour sa collaboration à la mise au point de certaines parties du premier prototype. Je tiens à mentionner également la contribution de Martin D'Anjou pour son projet de maîtrise portant sur l'intégration des composantes analogiques du circuit et celle de Jean-François Tremblay et Pierre-Martin Tardif pour leur projet de la fonction de dilatation morphologique invariante. Les projets d'été effectués par Yanik Tremblay, Stéphane Morin et Stéphane Dallaire se sont tous avérés d'une grande qualité et m'ont permis d'accélérer ma recherche. Les projets de fin d'étude entrepris par André Hamel, Luc Chouinard et Alain Kirouack ont également été d'une utilité substantielle. Merci à vous tous.

Mes remerciements s'adressent également au programme de fonds FCAR et au programme IRIS pour leurs subventions de recherche ainsi qu'à toute l'équipe de la société canadienne de micro-électronique pour leur support technique et pour la fabrication des nombreux prototypes de circuits intégrés.

Je tiens à souligner l'intérêt marqué suscité par mes parents Ronald et Aline tout au long de ce travail. C'est eux qui ont su m'inculquer le goût du savoir et la notion si importante du devoir accompli.

La personne a qui je dois le plus est ma femme Roxane. Au cours de ces trois années, elle a donné naissance à deux magnifiques enfants et elle a, par la force des choses, passé la plus grande partie de son temps avec eux à la maison au détriment de sa carrière professionnelle. Merci pour ta patience et ta compréhension. C'est d'ailleurs à toi, Roxane, que je dédie cette thèse, ainsi qu'à nos trois enfants.

TABLE DES MATIÈRES

RÉSUMÉ ii

RÉSUMÉ iii

AVANT-PROPOS v

TABLE DES MATIÈRES vii

LISTE DES TABLEAUX xvi

LISTE DES FIGURES xviii

INTRODUCTION 1

0.1 Vision artificielle 1

0.2 Vers un système de vision compact et flexible 2

CHAPITRE 1

TRAITEMENT D'IMAGES INTÉGRÉ 5

1.1 Vision numérique et traitement d'images 5

1.1.1 Vision 3D 6

1.1.2 Vision 2D passive 6

1.2 Limitation physique de la technologie microélectronique 7

1.3 Différentes approches de traitement d'images 8

1.3.1 Traitement massivement parallèle 9

1.3.2 Traitement en pipe-line 9
1.3.3 les réseaux neuroniques 10
1.3.4 Traitement intégré au plan focal 11
1.4 Opérateurs intégrés au plan focal 11
1.4.1 Filtrage spatial 11
1.4.2 Topologie hexagonale ou cartésienne 12
1.4.3 Traitement analogique 13
1.4.4 Extraction des résultats 14

CHAPITRE 2

LE SYSTÈME MAR 16
2.1 Définition conceptuelle 16
2.1.1 Jeu d'instructions spécialisé 18
2.1.2 Structure de données hiérarchisée et description d'état 19
2.2 Architecture générale du système MAR 20
2.3 Architecture du système hôte 21

CHAPITRE 3

LE CAPTEUR MAR 24
3.1 Photo-capteur spécialisé 24
3.1.1 Compromis résolution complexité 25
3.1.2 Opto-électronique et technologie CMOS 25
3.1.3 Photo-capteur à accès multi-port non destructif 32
3.1.4 Linéarité de la réponse du pixel en fonction du signal lumineux 36
3.2 Modes de balayage 40
3.2.1 Registres à décalage 41
3.2.2 Vérification de débordement et testabilité du capteur 44
3.2.3 Code de déplacement et définition de la parité en topologie hexagonale 45
3.2.4 Initialisation du capteur 49

3.3 Adressage de la région d'intérêt 53
3.3.1 Masque de convolution en topologie hexagonale 53
3.3.2 Masque de convolution à plusieurs pixels d'intérêt 55
3.3.3 Opérateurs à symétrie de révolution appliqués au capteur MAR. 57
3.3.4 Opérateur spatial généralisé 60
3.3.5 Multiplexer analogique des sorties 61
3.3.6 Remise à zéro du processus d'intégration d'illuminance 64
3.4 Réalisation VLSI 64
3.4.1 Compilateur de structures 65
3.4.2 Fréquence maximum de balayage 69
3.4.3 Détails géométriques d'un pixel 70
3.4.4 Circuits de génération et de propagation des horloges. 76
3.4.5 Historique de la réalisation VLSI 77
3.4.6 Description des plots de contact 78

CHAPITRE 4

LE CONTRÔLEUR DE CAMÉRA 80
4.1 Description fonctionnelle 80
4.2 Microcode et jeu d'instructions 82
4.2.1 Unité de logique programmable (PLA) 82
4.2.2 Registre d'instructions 84
4.2.3 Instructions de contrôle 85
4.2.3.1 L'instruction: *RESET_POSITION* 85
4.2.3.2 L'instruction: *SET_CAPTEUR* 86
4.2.3.3 L'instruction de remise à zéro du contrôleur 86
4.2.4 Modes de balayage local du pixel d'intérêt 86
4.2.5 Direction de déplacement généralisé 88
4.2.6 Instructions de déplacement du pixel d'intérêt 89
4.2.7 Instructions de déplacement simple 89
4.2.7.1 Déplacement rapide: *FAST_MOVE* et *FAST_MOVE_N* 89
4.2.7.2 Déplacement linéaire: *MOVE*, *MOVE_N* et *MOVE_ZIGZAG_N* ... 90

4.2.7.3 Rebondissement aux frontières d'une région de l'image 90
4.2.8 Instructions de déplacement complexes 92
4.2.8.1 Déplacement triangulaire: *MOVE_TRI* et *MOVE_TRI_N* 92
4.2.8.2 Déplacement hexagonal: *MOVE_HEX* et *MOVE_HEX_N* 93
4.2.8.3 L'instruction de balayage de région: *SCAN_AREA* 94
4.2.9 L'instruction intelligente de poursuite d'arêtes: *FIND* 94
4.3 Description détaillée du contrôleur MAR 100
4.3.1 Arbitration des instructions 100
4.3.2 Contrôle de boucles externes au PLA 104
4.3.3 Générateur de demandes d'interruptions et auto-arrêt d'exécution 104
4.3.4 Registres de position et pointeur de mémoire MAR 109
4.3.5 Définition de la fenêtre d'étude 111
4.3.6 Module d'exécution conditionnelle 112
4.3.7 Unité arithmétique de direction 112
4.3.8 Module d'évaluation de l'orientation d'arête 116
4.3.9 Détection des passages par zéro 118
4.3.10 Routage et sélection des données 127
4.3.11 Segments rectilignes d'arêtes et déplacements généralisés 129
4.3.12 Détection de retour au point de départ 135
4.3.13 Archivage du balayage et des points de discontinuité d'arête 135
4.3.14 Description des registres de configuration 140
4.4 Circuits périphériques 143
4.4.1 Mémoire MAR de description d'état 143
4.4.2 Générateur d'histogrammes spécialisé 145
4.4.3 Conditionnement des signaux d'arêtes 145
4.5 Réalisation VLSI 145
4.5.1 Générateur de PLA 145
4.5.2 Entrée schématique du circuit 149
4.5.3 Placement et routage automatique 150
4.5.4 Testabilité 150
4.5.5 Historique de la réalisation VLSI 151
4.5.6 Description des plots de contact 151

CHAPITRE 5

TRAITEMENT ANALOGIQUE ET SIMULATIONS 152

5.1 Opérateur analogique de convolution 153

5.1.1 Circuit à miroirs de courant 155

5.1.2 Contrôle numérique des coefficients 156

5.1.3 Calibration des filtres 158

5.2 Acquisition numérique des données 158

5.3 Simulation de l'opérateur de convolution 159

5.4 Analyse de Fourier du masque de convolution 160

5.5 Simulation numérique du contrôleur MAR 163

5.6 Émulateur MAR sur station graphique SUN 163

CHAPITRE 6

MONTAGE EXPÉRIMENTAL, RÉSULTATS ET APPLICATIONS 165

6.1 Prototype de la caméra MAR 165

6.1.1 Banc d'essai pour photo-capteurs spécialisés 165

6.1.2 Composantes mécaniques de la caméra 167

6.1.3 Mise au foyer de l'image sur le capteur 167

6.2 Résultats expérimentaux 171

6.3 Algorithmes de commande et applications spécialisées 177

6.3.1 Détection d'arêtes multi-résolution et segmentation hiérarchisée 177

6.3.2 Interpolation sub-pixel 178

6.3.3 Opérateurs morphologiques 179

6.3.4 Stéréo-vision passive 180

6.3.5 Recouvrement de surfaces douces 182

6.3.6 Poursuite dynamique 187

6.3.7 Reconnaissance de patrons et de textures 187

CONCLUSION 188

RÉFÉRENCES 191

ANNEXE A

Conditionnement du Signal Analogique 195

A.1 Compensation du courant de fuite et de la composante continue 195

ANNEXE B

Microcode du Système MAR 199

B.1 Format du microcode et généralité sur la logique programmable 199

B.2 Microcode du contrôleur MAR (version intégrale) 203

B.3 Microcode du contrôleur MAR (version étendue) 207

ANNEXE C

Langage Assembleur pour le Système MAR 217

C.1 INTRODUCTION 217

C.2 Le Langage Assembleur 217

C.3 Conception du compilateur 218

C.3.1 Compilation 219

C.3.2 Redirection de l'entrée et de la sortie 219

C.3.3 Utilisation 219

C.4 DESCRIPTION DES CHAMPS 220

C.4.1 Champ direction: [DIR#d] 222

C.4.2 Champ condition: [condition] 223

C.4.3 Champ de sélection d'entrée: [INPUT#i] 224

C.4.4 champ S: [HELIX] 225

C.5 INSTRUCTIONS 226

C.5.1 FAST_MOVE 226

C.5.2 FAST_MOVE_N 227

C.5.3 FIND 228

C.5.4 MOVE 230

C.5.5 MOVE_HEX 231
C.5.6 MOVE_HEX_N 233
C.5.7 MOVE_N 236
C.5.8 MOVE_TRI 238
C.5.9 MOVE_TRI_N 240
C.5.10 MOVE_ZIGZAG_N 243
C.5.11 RESET_POSITION 245
C.5.12 SCAN_AREA 246
C.5.13 SET_CAPTEUR 249

ANNEXE D

Test Fonctionnel du Contrôleur MAR 250
D.1 Introduction 250
D.2 Description du circuit: Le contrôleur de caméra MAR 251
D.3 Procédure de test: du contrôleur MAR 260
D.3.1 Vérification des horloges phi1 et phi2 261
D.3.2 Vérification du module d'arbitration des instructions 262
D.3.3 Vérification des compteurs C, E et N 263
D.3.4 Vérification des compteurs x et y 263
D.3.5 Vérification du microcode 264
D.3.6 Vérification du module VISITE 264
D.3.7 Vérification du module ÉROSION 266
D.3.8 Évaluation de la fréquence maximale d'utilisation 267
D.4 "Bugs" du LVMRC 268
D.5 État du circuit LVMRC 269

ANNEXE E

Dessin des Masques du Capteur et du Contrôleur MAR 270

ANNEXE F

Plans schématiques du contrôleur réalisé à l'aide de composantes discrètes 280

ANNEXE G

Programme en C du recouvrement de surface à partir de morceaux de triangles 294

G.1 Programme principal "shape_main.c" 294

G.2 Fonctions requises par le programme principal: "shape_conv.c" 298

ANNEXE H

Test Fonctionnel du Circuit à Miroir de Courant 310

H.1 Vérification du LVMIR: Description du circuit 310

H.2 Procédure de test 315

H.2.1 Vérification du fonctionnement statique: 316

H.2.2 Vérification du fonctionnement dynamique: 316

H.2.3 Évaluation de la réponse en fréquence: 316

H.2.4 Remarques sur les deux courbes: 316

H.2.5 Évaluation du temps de montée et du temps de décroissance: 317

ANNEXE I

Simulation Logicielle de l'Opérateur de Convolution 318

I.1 Fonction de conversion d'une image cartésienne en topologie hexagonale 318

I.2 Fonction qui effectue le filtrage d'une image en topologie hexagonale 319

I.3 Fonction qui effectue le suivi d'arête en topologie hexagonale 320

I.4 Fonction qui déplace le pixel d'intérêt en topologie hexagonale 325

I.5 Fonction qui convertit l'une image résultante en nombre entier 327

I.6 Fonction de conversion de la topologie hexagonale à une image cartésienne .. 327

I.7 Programme principal de simulation de l'opérateur de convolution 328

ANNEXE J

Equations logiques du décodeur pour le pointeur de mémoire MAR **332**

ANNEXE K

Plans mécaniques des composantes de la caméra MAR **333**

ANNEXE L

Plans Électriques du Circuit de Conditionnement pour le Capteur **338**

LISTE DES TABLEAUX

Tableau 3.1	Description sommaire des cellules de base du capteur MAR	**67**
Tableau 4.1	Définition des variables externes du module de condition.....	**112**
Tableau 4.2	Règles arithmétiques modulo 6 utilisée à la Figure 4.22.	**114**
Tableau 4.3	Table de vérité du module de modification de la déviation	**116**
Tableau 4.4	Calcul l'orientation d'arêtes pour les exemples de la Figure 4.29	**118**
Tableau 4.5	Configurations possibles pour les bus MD et IO.....	**127**
Tableau 4.6	Liste des variables d'état qui sont archivées	**136**
Tableau 4.7	Contenu du registre de description d'état (Etat_1<15:0>)	**138**
Tableau 4.8	Contenu du registre de description d'état (Etat_0<15:0>)	**139**
Tableau 4.9	Description du registre de contrôle #2.	**140**
Tableau 4.10	Description du registre de contrôle #1.	**141**
Tableau 4.11	Description du registre de contrôle #0.	**142**
Tableau 4.12	Description des cellules de base du PLA de la Figure 4.43.	**148**
Tableau 4.13	Liste des dimensions et des propriétés du circuit du contrôleur MAR	**149**
Tableau B.1	Description des entrées du microcode du contrôleur MAR	**201**
Tableau B.2	Description des sorties du microcode du contrôleur MAR	**202**
Tableau D.1	Plage d'adressage des Registres du LVMRC.....	**252**
Tableau D.2	Signaux de sorties configurables sur le bus "MD<15:0>".....	**256**
Tableau D.3	Variables d'états pouvant être programmées pour le bus MD<15:0>	**256**
Tableau D.4	Signaux de sorties configurables sur "IO<4:0>".....	**258**
Tableau D.5	Variables d'états pouvant être programmées pour le bus IO<4:0> ...	**258**
Tableau D.6	Signaux de sorties configurables sur "IO<7:5>"	**259**
Tableau D.7	Variables d'états pouvant être programmées pour le bus IO<7:5> ...	**260**
Tableau D.8	Évaluation des valeurs de "S", "H-" et "H+"	**267**
Tableau D.9	"Bugs" du LVMRC	**268**
Tableau F.1	Description des plans du contrôleur MAR en composantes discrètes.	**281**
Tableau H.1	Adressage du LVMIR	**313**

Tableau H.2 Plage d'adressage des registres de LVMIR **314**
Tableau H.3 Analyse transitoire de circuit à miroir de courant **317**

LISTE DES FIGURES

Figure 2.1 Diagramme bloc du système MAR 17
Figure 2.2 Architecture typique d'un système exploitant la caméra MAR 22
Figure 3.1 Création d'une paire électron-trou dans du silicium intrinsèque..... 26
Figure 3.2 Charges mobiles près d'une jonction PN non polarisée pour le silicium... 27
Figure 3.3 Variation de la largeur de la région de charges d'espace 28
Figure 3.4 Variation spatiale de potentiel pour une jonction PN+ 29
Figure 3.5 Photo diode PIN en technologie CMOS..... 30
Figure 3.6 Photo-transistor NPN réalisé en technologie CMOS à puits P..... 31
Figure 3.7 Architecture multi-port du capteur MAR 33
Figure 3.8 Différentes configurations de sources de courant commandées..... 34
Figure 3.9 Diagramme temporel représentant la procédure de la prise d'image 35
Figure 3.10 Configuration électronique d'un pixel adressé par le bus Y 37
Figure 3.11 Familles de courbes de Ids en fonction de Vds et Vgs..... 40
Figure 3.12 Modification de la réponse du pixel avec un transistor M4 plus petit..... 41
Figure 3.13 Diagramme bloc du capteur MAR 42
Figure 3.14 Architecture générale d'un module de registre à décalage..... 43
Figure 3.15 Module de registre à décalage avec horloges conditionnées..... 44
Figure 3.16 Convention du code de direction..... 45
Figure 3.17 Définition de la parité d'une ligne en topologie hexagonale..... 46
Figure 3.18 Topologie hexagonale exacte versus une approximation 7/8..... 47
Figure 3.19 Décodeur de direction du capteur MAR..... 48
Figure 3.20 Exemple d'un capteur de 4 X 4 pixels..... 51
Figure 3.21 Séquence d'initialisation du capteur MAR..... 52
Figure 3.22 Masque de convolution en topolologie hexagonale 54
Figure 3.23 Représentation des signaux analogiques à la sortie du capteur MAR..... 55
Figure 3.24 Masque de convolution à trois pixels d'intérêt..... 56
Figure 3.25 Zone d'influence des pixels du masque de convolution..... 58
Figure 3.26 Filtres directionnels connus appliqués à la topologie hexagonale..... 60
Figure 3.27 Architecture interne du multiplexer analogique de sortie 62
Figure 3.28 Pixel MAR complet tel que fabriqué pour le capteur 256 X 256..... 65
Figure 3.29 Module qui utilise le bus Y pour la remise à zéro des pixels 66
Figure 3.30 Plan de description pour la génération du capteur MAR 67
Figure 3.31 Réponse analogique des sorties du capteur MAR 71
Figure 3.32 Détails géométriques de la réalisation VLSI du capteur MAR 72
Figure 3.33 Exemple d'une matrice de 3 X 3 pixels MAR 74
Figure 3.34 Géométrie du photo-détecteur en forme d'arbre 75
Figure 3.35 Circuit de génération des l'horloges à deux phases sans recouvrement... 76
Figure 3.36 Description des plots de contact du capteur MAR. 79

Figure 4.1 Diagramme bloc du contrôleur de caméra MAR. 81
Figure 4.2 Plan d'ensemble de l'utilisation d'un PLA comme machine à états 83
Figure 4.3 Registre d'instruction du contrôleur MAR 84
Figure 4.4 Position possibles de l'origine du référentiel image du contrôleur MAR .. 86
Figure 4.5 Modes de déplacement applicables aux instructions 87
Figure 4.6 Exemple de déplacement généralisé 88
Figure 4.7 Exécution de l'instruction "MOVE_N" avec option de rebondissement ... 91
Figure 4.8 Exécution des instructions "MOVE_TRI" et "MOVE_TRI_N" 92
Figure 4.9 Exécution des instructions "MOVE_HEX" (a) et "MOVE_HEX_N" 93
Figure 4.10 Instruction de balayage de régions 95
Figure 4.11 Exécution de l'instruction de la poursuite d'une arête 96
Figure 4.12 Exemple de poursuite d'arête en mode aveugle 97
Figure 4.13 Comparaison entre la poursuite d'arête par pas de 30 et de 60 degrés 99
Figure 4.14 Module d'arbitration des instructions 102
Figure 4.15 Détail logique du module d'arbitration des instructions 109
Figure 4.16 Fonctionnement des compteurs d'événements utilisés par le PLA 105
Figure 4.17 Génération du signal d'auto-arrêt "BREAK" 107
Figure 4.18 Compteur de cycles de stabilisation et d'attente virtuelle 108
Figure 4.19 Signal qui permet d'avorter l'exécution d'une instruction 109
Figure 4.20 Pointeur de mémoire MAR et générateur de position 110
Figure 4.21 Détection du débordement de la fenêtre d'étude 111
Figure 4.22 Unité Arithmétique de direction 113
Figure 4.23 Module d'addition modulo 6 utilisé à la Figure 4.22 115
Figure 4.24 Code d'orientation d'arêtes 117
Figure 4.25 Module de calcul de l'orientation d'arête 119
Figure 4.26 Détection de l'occurrence d'une orientation d'arête spécifique 119
Figure 4.27 Détection des passages par zéro du signal R(x,y) 121
Figure 4.28 Module intégré de génération des bits "S" et "H" 122
Figure 4.29 Opérateur de dilatation appliqué au signal "H" 123
Figure 4.30 Exemple 2D de l'opération de détection des passages par zéro 124
Figure 4.31 Problème associé à la trop forte dilatation du signal "H" 125
Figure 4.32 Grille des positions possibles des détections d'arêtes 126
Figure 4.33 Configuration du transfert de données pour les bus MD et IO 128
Figure 4.34 Multiplexer spécialisé pour les signaux de requête d'interruption 130
Figure 4.35 Procédure de détection d'un segment rectiligne d'arête 131
Figure 4.36 Circuit de détection des segments rectilignes d'arêtes 133
Figure 4.37 Composition des signaux de modification de la déviation 134
Figure 4.38 Circuit de détection du retour au point de départ 136
Figure 4.39 Registre à décalage utilisé pour l'archivage des variables d'état 137
Figure 4.40 Evaluation de l'achalandage des pixels 139
Figure 4.41 Circuits périphériques requis pour la mémoire d'état 144
Figure 4.42 Histogrammes pyramidaux 146
Figure 4.43 Organisation des blocs pour la génération automatique du PLA 147
Figure 5.1 Opérateur analogique de convolution 153
Figure 5.2 Ajout d'une composante continue au filtre pour uniformiser le signe 154
Figure 5.3 Circuit à miroir de courant pour le calcul du produit de convolution 156

Figure 5.4 Contrôle numérique des coefficients pour une convolution analogique 157
Figure 5.5 Calibration des opérateurs laplacien de gaussienne 159
Figure 5.6 Résultats de simulation pour deux images 161
Figure 5.7 Analyse de Fourier de l'opérateur de convolution du capteur MAR 162
Figure 5.8 Vue d'ensemble de la fenêtre graphique de l'émulateur logiciel 164
Figure 6.1 Banc d'essai pour photo-capteur spécialisé 166
Figure 6.2 Schéma d'assemblage de la caméra MAR 168
Figure 6.3 Montage requis pour la calibration de l'orientation du capteur 169
Figure 6.4 Patrons des passages par zéro observés lors d'un désalignement 170
Figure 6.5 Résultats obtenus grâce au banc d'essai pour photo-capteurs spécialisés.. 172
Figure 6.6 Images d'arêtes résultantes pour la scène de "SUN" 173
Figure 6.7 Extraction d'arêtes à multiples résolutions sur l'image du symbole d'IRIS 174
Figure 6.8 Résultats pour une scène d'objets polyédriques 175
Figure 6.9 Interpolation sub-pixel de la position d'un passage par zéro 179
Figure 6.10 Modèle de caméra stéréo 181
Figure 6.11 Organisation géométrique d'un morceau de triangle équilatéral 184
Figure 6.12 Recouvrement de l'image à partir de surfaces triangulaires 186
Figure A.1 Compensateur de courant de fuite et de la composante continue 196
Figure A.2 Conversion courant tension 197
Figure A.3 Forme d'onde temporelle du signal VGoff 198
Figure C.1 Exemple de balayage d'un coin d'hexagone en mode zigzag 235
Figure C.2 Balayage d'un coin d'un triangle équilatéral en mode zigzag 242
Figure D.1 Description des plots de contact du LVMRC 251
Figure D.2 Registres de configuration "MD_conf" et "IO_conf" 254
Figure D.3 Registres de configuration "int" 255
Figure D.4 Vérification des signaux "phi1" et "phi2" 261
Figure D.5 Arbitration des instructions 262
Figure D.6 Montage utilisé pour vérifier le module VISITE 265
Figure D.7 Signification des signaux "S_out" et "H_out" 266
Figure E.1 Dessin des masques du capteur MAR de 128 x 128 pixels 271
Figure E.2 Représentation "template" utilisée par le compilateur de structure 272
Figure E.3 Détail du coin inférieur gauche du capteur MAR 273
Figure E.4 Détail du coin supérieur gauche du capteur MAR 273
Figure E.5 Détail du coin supérieur droit du capteur MAR 274
Figure E.6 Détail du coin inférieur droit du capteur MAR 274
Figure E.7 Détail d'un pixel du capteur MAR 275
Figure E.8 Partie de la matrice de pixels de 3 x 3 éléments photo-sensibles. 276
Figure E.9 Circuit complet du contrôleur MAR 277
Figure E.10 Dessin des masques d'un exemple d'un PLA de 2 entrées, 6 sorties 278
Figure E.11 Vue grossière de la description schématique du contrôleur MAR. 279
Figure F.1 PLA, unité arithmétique de direction et registre d'instruction. 282
Figure F.2 Module d'orientation d'arête 283
Figure F.3 Compteurs d'événements N (16 bits), E et C (8bits). 384
Figure F.4 Décodage d'adresse, Registre de configuration de base 285
Figure F.5 Pointeur de mémoire MAR et définition de la fenêtre d'étude. 286
Figure F.6 Générateur d'horloges et Arbitration des instructions 287

Figure F.7 Sélecteur de données et registres de description d'état. **288**
Figure F.8 Sélecteur de données et détection de retour au point de départ. **289**
Figure F.9 Module de dilatation et sélecteur des entrées analogiques. **290**
Figure F.10 Mémoire MAR de description d'état (128 K x 16 bits). **291**
Figure F.11 Configuration des requêtes d'interruptions. **292**
Figure F.12 Connecteur du système hôte et du capteur MAR. **293**
Figure H.1 Description des plots de contacts du LVMIR. **311**
Figure H.2 Circuit miroir de courant. **312**
Figure H.3 Montage utilisé pour vérifier le LVMIR **315**
Figure K.1 Pièce principale du boîtier de la caméra MAR **334**
Figure K.2 Plan du couvert arrière de la caméra **335**
Figure K.3 Plan du cylindre d'assemblage pour le boîtier de la caméra. **336**
Figure K.4 Plan de la forme des circuits imprimés de la caméra **337**
Figure L.1 Plan de connexion pour le capteur MAR **339**
Figure L.2 Plans des 16 premiers convertisseurs courant/tension **340**
Figure L.3 Plans des 10 autres convertisseurs courant/tension **341**
Figure L.4 Définition du connecteur qui relie l'ensemble des circuit imprimés **342**

INTRODUCTION

0.1 Vision artificielle

Parmi les technologies de pointe dans le domaine de l'électronique, la micro-informatique et l'automatisation, on retrouve la vision artificielle qui a pour principal rôle, à son plus bas niveau de définition, l'identification et la description de l'information visuelle d'une scène. Le but premier de ce travail de recherche consiste à développer un système de vision compact, flexible et compatible avec le type de traitements requis par les différentes tâches de reconnaissance de formes et d'automatisation.

La vision artificielle est une science qui intègre plusieurs technologies différentes pour l'acquisition de l'information visuelle, l'exécution d'une multitude de traitements et l'extraction de l'information pertinente à une application spécifique. La vision nécessite un très grand support en termes de calcul numérique et de capacité mémoire principalement à cause de l'aspect bidimensionnel du contenu d'une image. Dans le cas d'une application automatisée, l'information extraite de la scène est utilisée pour asservir la séquence des opérations et effectuer le contrôle de la qualité.

On utilise habituellement une caméra vidéo pour transformer l'image en signal analogique, un ou plusieurs circuits dédiés pour échantillonner l'image, l'emmagasiner et effectuer différents traitements. Le tout est généralement greffé à un ordinateur hôte qui permet d'initialiser le mode d'opération et de commander l'ensemble des circuits dédiés, d'ajuster l'éclairage et d'intégrer l'information provenant d'autres types de capteurs.

La vision artificielle est divisée en deux grandes classes: la vision 2D et la vision 3D. La vision bidimensionnelle tente d'extraire l'information sur la scène à partir de

niveaux de gris ou de couleurs sans avoir une information directe de la distance relative des différents objets par rapport à l'observateur (profondeur). Les images de niveau de gris sont typiquement générées par une caméra de télévision couplée à un échantillonneur d'images. Pour la vision 3D, on se sert par contre de capteurs spéciaux afin d'extraire précisément la profondeur de chaque point de la scène. Ces capteurs 3D utilisent en général un principe de triangulation active bien qu'une caméra 2D conventionnelle puisse être modifiée pour évaluer les données 3D comme c'est le cas pour la caméra BIRIS [11].

Le principal problème rencontré en vision artificielle est la faible vitesse de traitement des machines qui ne peut suffire au débit de calcul requis par le traitement d'images. La presque totalité des systèmes performants de vision (ou machines de vision) utilise une architecture parallèle répétitive pour atteindre un niveau de résolution temporelle acceptable. Le développement de machines de traitement massivement parallèle et de logiciels d'exploitation efficaces permettront bientôt de solutionner certains problèmes reliés au monde de la vision artificielle. La micro-électronique constitue un outil d'envergure permettant la réalisation de ces machines spécialisées dans le traitement d'images. Le niveau de complexité et la haute technologie utilisée rendent cependant ces systèmes encore trop coûteux pour favoriser une intégration massive et généralisée de la vision dans les milieux industriels.

0.2 Vers un système de vision compact et flexible

Plusieurs activités de recherche sont présentement orientées vers le développement de capteurs spécialisés intégrant une partie du traitement d'images directement sur la matrice bidimensionnelle de photo-capteurs. La technologie d'intégration à très grande échelle (ITGE ou VLSI) permet de créer ces photo-capteurs spécialisés en utilisant la technologie CMOS et d'atteindre ainsi un niveau de résolution (nombre de pixels) intéressant grâce à sa faible consommation d'énergie et à son niveau élevé d'intégration (dimension physique des circuits).

La vision biologique est un modèle intéressant de pré-traitement au plan focal et doit forcément inspirer notre réflexion lors de la réalisation de capteurs à traitement intégré. L'oeil utilise une structure tridimensionnelle de neurones spécialisés au niveau de la rétine où l'interaction directe entre les photo-récepteurs voisins diminue la quantité d'information à transmettre au cortex visuel via le nerf optique. Certaines recherches en neuro-biologie et en psychologie ont suggéré que le système de perception visuel humain pré-attentif procède à l'extraction des contours des objets selon plusieurs niveaux d'abstraction pour mener à l'apprentissage et à la reconnaissance de l'information visuelle. Ces résultats nous poussent

à imaginer un capteur spécialisé qui effectue un pré-traitement sous forme de filtrage spatial pour la détection des arêtes selon plusieurs niveaux de résolution. On peut également prévoir un sous-système numérique associé qui combine ces images d'arêtes et exécute une opération plus complexe comme le suivi d'arêtes.

On développe dans cet ouvrage le concept d'une caméra programmable qui offre un vaste répertoire d'instructions à la manière d'un micro-processeur et qui permet d'effectuer à même la caméra une grande variété de pré-traitements. Puisque les traitements les plus laborieux se résument généralement à des opérateurs locaux et répétitifs, un grand nombre de ceux-ci pourront être réalisés directement à l'étape de l'acquisition d'images.

On débute la thèse par une discussion générale sur le traitement d'images intégré. On énonce les principales propriétés qu'un tel système de vision devrait offrir pour combler les besoins des processus de segmentation et de reconnaissance ultérieurs. On y décrit également le genre de structure de données qui doit être générée pour être compatible avec les algorithmes de plus haut niveau. On présente au chapitre 2 la description conceptuelle du système de vision à Accès Multi-port de photo-Récepteurs (MAR). On y présente également l'architecture générale de ce système basé sur un capteur spécialisé à topologie hexagonale et la façon dont on devrait réaliser l'interface avec un ordinateur hôte. Le chapitre 3 comprend la description détaillée du capteur MAR. On y présente l'architecture interne requise pour réaliser l'extraction multiple et simultanée des données analogiques d'une région de pixels de l'image. On discute également dans cette partie de la thèse de considérations opto-électroniques de même que de certains détails de la réalisation VLSI de ce circuit intégré très particulier.

Le balayage du capteur MAR est orchestré par un contrôleur micro-programmé dont la description fonctionnelle fait l'objet du chapitre 4. On décrit dans cette partie du travail le jeu d'instructions spécialisé de ce contrôleur qui en fait un processeur d'images intégré, de même que son mode d'interaction avec un ordinateur hôte qui se fait sous forme de communication numérique asynchrone et par voie d'interruptions multiples. On présente son mode d'opération en plus de décrire les diverses fonctions de traitement d'images qui y sont incluses. Le chapitre 5 est une discussion sur le module de traitement analogique qui opère en périphérie du capteur MAR et qui effectue en parallèle le filtrage multi-résolution de l'image projetée au plan focal de la caméra.

Le dernier chapitre traite de la simulation de certains des modules du système MAR de même que des procédures de test de ses composantes principales. On complète la thèse avec la présentation des résultats expérimentaux et par un aperçu des performances

globales du système. Une section est réservée à la description sommaire de quelques algorithmes de commandes de même que des application spécialisées en vue d'exploiter efficacement le système de vision MAR. On énonce finalement les perspectives de modification en vue d'améliorer le concept de caméra à traitement d'images intégré au plan focal.

CHAPITRE 1

TRAITEMENT D'IMAGES INTÉGRÉ

Ce chapitre est à la fois une discussion sur les différentes approches de traitement de l'information visuelle et une description plus approfondie sur le traitement d'image intégré au plan focal. On y découvre les qualités indispensables des technologies d'intégration à très grande échelle (TTGE) dont le niveau de performance et d'intégration s'apparente à la puissance de calcul requise par le traitement d'images.

1.1 Vision numérique et traitement d'images

La vision numérique est une science relativement récente qui utilise de façon générale l'information fournie par un système d'acquisitions d'images pour extraire des caractéristiques spécifiques en vue de l'automatisation d'un processus quelconque. L'image brute constitue une grande quantité d'information nullement compatible avec les besoins des procédures de reconnaissance et de décision automatisées. Sur ces données brutes, on doit appliquer une multitude de procédures pour en extraire les primitives qui serviront à décrire la scène et, par la même occasion, réduire la taille des données et organiser l'information de façon structurée. La vision numérique est séparée en deux classes distinctes: la vision 3D (ou tri-dimensionnelle) et la vision 2D. Les systèmes

d'acquisitions et les procédures de traitement d'images sont généralement différentes pour chaque classe bien que les objectifs poursuivis sont communs. Les conclusions obtenues par les deux systèmes sont de nature complémentaire et devraient idéalement êtres fusionnés pour enrichir la description de l'environnement.

1.1.1 Vision 3D

La vision 3D consiste en l'extraction de l'information sur la profondeur de chaque point de la scène. Cette information donne une description détaillée de la distance entre les objets de la scène et l'observateur en donnant peu de détails sur leur couleur ou leur texture [4]. Les systèmes d'acquisitions d'images 3D sont généralement basés sur l'utilisation de capteurs spécialisées à balayage laser bien que certains systèmes utilisent une caméra vidéo standard [11]. Ces capteurs spécialisés éclairent la scène avec une source laser pour ensuite extraire la profondeur à partir de la position de la lumière réfléchie et captée par l'organe photo-sensible par triangulation [38] [49]. Les systèmes de vision 3D sont donc généralement des systèmes actifs et doivent être utilisés dans un environnement contrôlé (souvent isolé) afin de garantir la protection des personnes présentes dans le champs de balayage et de réflexion du faisceau laser.

1.1.2 Vision 2D passive

La vision 2D passive utilise l'image produite par la réflexion de la lumière sur de la surface des objets d'une scène pour en extraire les caractéristique visuelles. Celle-ci s'apparente particulièrement au système de vision biologique humain. Pour faciliter la prise d'images et son interprétation, on utilise généralement un éclairage contrôlé et connu en position, couleur et intensité. Plusieurs artifices permettent d'ajouter de la richesse à une simple image 2D ou, par exemple, plusieurs sources lumineuses placées à des positions différentes permettent de discriminer les ombrages en plus d'estimer l'orientation de la normale aux surfaces. L'analyse de plusieurs images prises à des instants différents sert à la détection du mouvement. L'ajout d'un ou plusieurs points de vue permet d'extraire certains points 3D par stéréoscopie. La vision passive a l'avantage de pouvoir s'intégrer sans danger aux environnements fréquentés par les êtres humains. La vision 2D ne permet pas cependant d'extraire simplement les caractéristiques tridimensionnelles de la scène. On doit généralement prévoir une séquence de traitements afin d'extraire l'information désirée. Typiquement, un pré-traitement de l'image d'illuminance comme la détection multi-résolution d'arêtes fait partie intégrante de cette séquence de traitements et devrait idéalement être incluse au système d'acquisitions.

1.2 Limitation physique de la technologie microélectronique

La capacité maximale de traitement des ordinateurs conventionnels est largement dépassée lorsqu'on fait référence au traitement d'images. Par exemple, une simple convolution 9 x 9 sur une image de 256 x 256 peut prendre plusieurs secondes même sur une machine performante comme un SPARC station 2 de SUN et ne constitue qu'une petite partie du traitement requis pour une application spécifique d'analyse d'images. En considérant que l'opération de filtrage spatial requiert plusieurs masques différents, on atteint rapidement une limitation physique associée à la très grande quantité d'information à traiter lors de l'analyse du contenu visuel d'une scène.

L'avènement des processeurs à jeu d'instructions réduit (RISC) offre un niveau de performance impressionnant pour des ordinateurs séquentiels. Cependant, ils ne font que diminuer, par un facteur limité, le temps d'exécution des programmes. L'avènement des ordinateurs parallèles, qui pénètrent présentement le marché avec un indice qualité-prix-performance très satisfaisant pave la voie à l'intégration du sens de la vue aux processus automatisés. Cette affirmation est renforcée par le fait qu'un grand nombre des algorithmes en vision sont exécutables efficacement sur des architectures parallèles. Néanmoins, la puissance de ces nouvelles ressources informatiques a avantage à être exploitée pour la résolution des problèmes de vision de moyen et de haut niveau et il est essentiel de pousser le développement de périphériques intelligents d'acquisitions d'images.

On utilise la micro-électronique pour contrer une partie du problème de la vitesse de traitement en proposant différentes architectures spécialisées qui opèrent à haute cadence et en parallèle. Il est évident que l'amélioration des performance globales de ces circuits spécialisés par rapport aux ordinateurs séquentiels est atténuée par la fonction souvent unique de ces architectures dédiées. Les architectures parallèles, en pipe-line, et intégrées au plan focal sont des exemples de solutions VLSI qui matérialisent le concept de machine de vision avec un niveau de performance acceptable pour des applications en temps réel dans les domaines de la robotique, de la navigation autonome et de l'automatisation de procédés.

La technologie CMOS est un prérequis essentiel au développement d'une machine de vision performante. L'intégration de centaines de milliers de transistors sur une même pastille de silicium permet de concevoir des calculateurs spécialisés qui augmente considérablement la vitesse de traitement d'un algorithme comparativement à son exécution sur un ordinateur séquentiel. La technologie CMOS consomme peu d'énergie, caractéristique essentielle lorsqu'on prévoit utiliser un grand nombre de circuits intégrés

dans un même système. Il existe cependant un lien étroit entre la performance d'un système VLSI et le niveau de spécialisation de son application. En fait, plus un système est performant, plus son utilisation est dédiée à un nombre restreint d'applications.

L'avènement prochain de technologies conjointe bipolaire et CMOS (BICMOS) intégrant sur une même pastille de silicium des transistors à effet de champ (CMOS) et des transistors bipolaires (BJT) donnera accès au développement d'éléments de calcul analogique aux propriétés très intéressantes. Avec cette technologie, il est possible d'intégrer sur une même pastille de silicium des organes de calcul analogique à haute performance de même que des circuits numériques volumineux qui consomment peu d'énergie.

Dans le cas du système MAR, la technologie CMOS permet la réalisation de circuits intégrés de grande dimension. La taille des pastilles de silicium peut atteindre 2 cm de côté avec un taux satisfaisant de fabrication de circuits sans défaut. On peut donc prévoir l'intégration d'éléments photo-sensibles de petite taille nécessaires à la réalisation d'un capteur spécialisé d'environ 500 x 500 pixels de résolution spatiale. La technologie CMOS peut également servir à réaliser certains modules de calcul analogique bien que, dans ce cas, la bande des fréquences d'application soit relativement limitée.

La vitesse maximale de commutation d'un transistor MOS représente une barrière physique au débit de données que peut supporter un module de calcul numérique d'un circuit intégré. Cette fréquence maximale est de l'ordre de 10 MHz à 100 MHz pour les circuits numériques, tout dépendant de la complexité des organes de calcule internes de ces circuits. Les technologies BiCMOS et bipolaire atteignent des fréquences d'opération de l'ordre de 1 GHz mais cela implique une plus grande consommation d'énergie et donc, une limitation sur la dimension réalisable des pastilles de silicium. De plus les technologies BiCMOS ne sont pas encore parfaitement maîtrisées et ne sont pas facilement accessibles. Chaque opération arithmétique s'étend sur un grand nombre de cycles d'horloge lorsqu'on considère le traitement d'images. On doit donc développer des architectures qui augmentent de façon apparente cette fréquence maximale de traitement pour arriver à effectuer l'analyse à l'intérieur d'un intervalle de temps raisonnable.

1.3 Différentes approches de traitement d'images

Pour accomplir un traitement raisonnable sur des images dans un temps assez court, on doit envisager des architectures spécialisées. Plusieurs de ces architectures ont déjà été proposées et d'autres font l'objet de recherches actives. Parmi celles-ci, on trouve les

architectures massivement parallèles [8] [46] [6], le traitement en pipe-line [17] [1], les réseaux neuroniques [32] [7] [22] [23] [31] [33] [50] [27] et les nouvelles approches de traitement intégré au plan focal [16] [37] [44] [39] [42] [34] [13] [24]. De façon générale, on cherche à exploiter le fait que la plupart des algorithmes de traitement d'images sont répétitifs et qu'ils accèdent à des données locales [6] [41].

1.3.1 Traitement massivement parallèle

Le traitement massivement parallèle est l'une des architectures les plus intuitives augmentant de façon significative la vitesse de traitement. On imagine qu'un processeur élémentaire est reproduit un grand nombre de fois et que cet ensemble d'éléments de calcul est disposé selon un arrangement matriciel. L'architecture parallèle est basée sur une exploitation des communications locales et est compatible avec plusieurs algorithmes de traitement d'images où chacun des processeurs fait accès à une partie de l'image voire même à un pixel. Dans le cas de traitement parallèle, on doit utiliser des processeurs très simples ayant un jeu d'instructions extrêmement réduit afin de permettre l'intégration d'un maximum d'organes de calcul par circuit intégré discret.

Un des problèmes majeur consiste à présenter les données simultanément à cette disposition complexe de processeurs et d'en extraire les résultats de façon cohérente. Il est de plus impossible d'intégrer l'ensemble des processeurs élémentaires à l'intérieur d'un même circuit intégré, ce qui implique une augmentation de la complexité au niveau du système. L'intérêt majeur de cette architecture est l'utilisation une seule instruction (SIMD) pour exécuter simultanément un grand nombre d'opérations sur une large partie de l'image. Ce type d'architecture est relativement coûteux dû à la nature discrète des unités élémentaires de traitement qui doivent être assemblées sur des cartes distinctes et intégrées au système complet [46]. Les nombreux liens de communication entre les différents modules définissent un type d'assemblage complexe et peu conventionnel. On doit également considérer l'effort additionnel de programmation requis pour exploiter efficacement un tel système car même si un algorithme est exécutable sur une machine parallèle, les modifications à apporter à cet algorithme peuvent représenter un travail considérable.

1.3.2 Traitement en pipe-line

Une autre approche de traitement consiste à agencer une série de modules de calcul en pipe-line. Puisque la plupart des opérateurs sont locaux et que les images vidéo conventionnelles présentent les données une ligne à la fois, on peut envisager de mémoriser 3,5 ou 7 lignes consécutives à l'aide de registres à décalage pour y effectuer un premier

traitement spatial comme une convolution ou une opération morphologique [1]. Chaque résultat est transféré à un module subséquent pour y réaliser la suite du traitement.

Cette approche a l'avantage d'exploiter la structure de balayage du signal vidéo standard et de fournir le résultat dans le même format avec un décalage temporel proportionnel au niveau de calcul effectué. La quantité de mémoire requise est relativement faible et se limite au nombre de lignes de la fenêtre d'étude. On obtient en général un taux de traitement constant peu importe le degré de complexité du traitement effectué mais, on observe un délai entre l'entrée et la sortie qui reflète ce niveau de calcul. Plusieurs systèmes de vision exploitent ce mode d'opération [17] et augmentent de façon considérable la vitesse de traitement sans entraîner un coût trop exorbitant. La principale limitation du traitement en pipe-line réside dans le format même des données qui oblige un traitement ligne par ligne. On peut donc difficilement envisager l'exécution d'une procédure de suivi d'arêtes à l'aide de ce type d'architecture

1.3.3 les réseaux neuroniques

Une toute nouvelle approche au traitement d'images est née avec l'avènement des réseaux neuroniques. Cette technique exploite la propriété de localité des opérateurs propre au traitement d'images [32] [7] et constitue un domaine de recherche très prometteur et particulièrement populaire depuis quelques années. A la manière des neurones biologiques, on cherche à établir des canaux de communication entre chaque élément de calcul afin d'effectuer simultanément un grand nombre d'opérations. Les interactions entre chaque paire de neurones sont souvent de type excitation-inhibition et évoquent ainsi la réalisation possible de réseaux neuroniques analogiques. Le niveau de complexité est atteint par l'arrangement spécial des canaux de communication entre des processeurs simples plutôt que par le développement de processeurs complexes [50].

L'avènement de calculateurs neuroniques intégrés permettrait de développer des machines de vision dont le traitement s'apparente avec les arrangement en couches du traitement cérébral humain et qui pourraient concurrencer très sérieusement avec les machines basés sur la redondance de calcul numérique. La plupart des architectures qui utilisent une approche de réseaux neuroniques ne peuvent être intégrées à cause de leur grande complexité et d'un niveau d'intégration encore inadéquat. Bien que certains calculateurs neuroniques sont réalisés à l'aide de technologies VLSI contemporaines [31] [41], les techniques d'intégration de circuits VLSI multi-couches (3D) permettront éventuellement de réaliser certains circuits très complexes de calcul neuronique.

1.3.4 Traitement intégré au plan focal

Le traitement intégré au plan focal constitue une approche intéressante pour le traitement d'images. On tente d'imiter d'une certaine façon l'architecture biologique des systèmes de perception visuelle. L'organe de la vue est généralement beaucoup plus complexe qu'un simple arrangement bidimensionnel de photo-détecteurs. La rétine de l'oeil humain, par exemple, est constituée d'une surface de cellules photo-sensibles à laquelle est couplée un arrangement multi-couches complexe de neurones spécialisés qui effectuent une forme de traitement spatial à plusieurs niveaux de résolution [26] [30].

L'intérêt majeur du traitement d'images intégré au plan focal est qu'il réduit considérablement la quantité d'information à transmettre au système de traitement et de reconnaissance. Cette affirmation est justifiée pour les systèmes biologiques comme pour d'éventuels systèmes électroniques. On remarque d'ailleurs que le nerf optique du système de vision humain devrait être de taille plus grande si aucune forme de compression ou de pré-traitement n'était réalisée au niveau de la rétine. Le traitement intégré au plan focal est traité de façon générale dans la section suivante. Les chapitres qui suivent font la description complète d'un tel système de traitement d'images intégré.

1.4 Opérateurs intégrés au plan focal

Il existe plusieurs type d'opérateurs qu'on voudrait pouvoir appliquer aux images originales avant même de passer aux algorithmes de haut niveau. L'image originale d'intensité est presque inutile lorsqu'on dispose, par exemple, de la liste et de l'image des arêtes d'une scène ainsi que de leurs relations hiérarchiques. Il est essentiel de tenir compte du besoin des algorithmes de reconnaissance de haut niveau avant de penser réaliser une forme de traitement au plan focal.

Dans l'optique du développement d'un capteur spécialisé dans le traitement des images, on doit se concentrer sur le type d'opérateurs qu'il convient d'intégrer, la grille qui sera utilisée pour placer les éléments photo-sensibles, ainsi que de la façon dont on doit générer puis extraire les résultats de l'opération désirée. Il est de plus essentiel de prévoir une multiplicité d'opérateurs afin que le système d'acquisitions soit flexible et facile à programmer en fonction de l'application visée.

1.4.1 Filtrage spatial

Le développement d'une rétine artificielle implique au minimum l'intégration d'une opération de filtrage spatial au plan focal [24] [42] [37] [44]. Soit $I(x, y)$ l'image de

la scène projetée sur le plan focal, $R(x, y)$ l'image résultant de la convolution de cette image d'entrée par un opérateur quelconque $H(x, y)$ s'écrit:

$$R(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} H(\tau, \zeta)\, I(x-\tau, y-\zeta)\, d\tau d\zeta \qquad (1\text{-}1)$$

Puisqu'on ne réalise que des capteurs discrets de $n \times m$ pixels, et que le masque de convolution $H[x, y]$ a généralement une région d'application limitée de dimension $(2X + 1)$ par $(2Y + 1)$ tel que:

$$H[x, y] = 0 \qquad \{x, y \mid (x \leq -X \cup x \geq X \cap y \leq -Y \cup y \geq Y)\} \qquad (1\text{-}2)$$

on obtient alors le résultat de la convolution discrète:

$$R[x, y] = \sum_{i=-X}^{X} \sum_{j=-Y}^{Y} H[i, j]\, I[x-i, y-j]\, \Delta S \qquad (1\text{-}3)$$

où ΔS est la surface d'un pixel de l'image d'entrée.

Il est prévisible que certaines contraintes de réalisation physique d'un capteur spécialisé impliquent des restrictions sur le type de filtrage possible comme par exemple le rayon maximal du filtre (X et Y), ou la liberté totale sur le choix des poids du filtre $H[x, y]$. Un compromis s'impose entre la complexité de chaque cellule d'un capteur spécialisé et la souplesse des opérateurs qu'on désire appliquer à l'image. Plusieurs autres considérations comme le type de balayage, la sensibilité, le genre de traitement analogique et/ou numérique à effectuer sur les signaux du capteur sont autant de points à prendre en compte pour la réalisation d'un filtrage rétinien.

1.4.2 Topologie hexagonale ou cartésienne

Un algorithme de vision est généralement développé pour des images échantillonnées selon une grille cartésienne. Cette propriété découle directement de la nature même du signal vidéo standard et de la fabrication des capteurs CCD qui sont développés selon une grille cartésienne. La façon de stocker l'image dans un ordinateur est un autre facteur qui favorise l'utilisation d'une topologie cartésienne. On associe facilement les indices d'une table bidimensionnelle d'éléments comme étant directement reliés à la position des pixels dans l'image.

Le développement d'un capteur spécialisé pour la vision artificielle n'a pas à utiliser préférablement une topologie plutôt qu'une autre. Il incombe donc d'analyser soigneusement quelle topologie est préférable pour chaque opération. La grille d'échantillonnage hexagonale possède plusieurs propriétés très intéressantes tant pour le traitement d'images que pour la réalisation physique de capteurs rétiniens. On retrouve même dans la littérature des capteurs à topologie irrégulière comme ceux à densité radiale logarithmique [39] [42] imitant la densité rétinienne des photo-détecteurs qui est plus grande au centre qu'en périphérie. Cette approche qui se veut une forme implicite de compression d'images est peu intéressante pour la réalisation de capteurs sur silicium puisqu'elle implique un gaspillage considérable de la surface utilisée [13] sous forme d'espacement excessif entre les photo-détecteurs de la périphérie du capteur.

La principale propriété de la topologie hexagonale est que tous les voisins immédiats d'un pixel donné sont à la même distance de ce dernier et occupent une position relative parfaitement symétrique. Cette symétrie simplifie la réalisation physique du capteur et fournit la possibilité d'élaborer des algorithmes de traitement d'images avec balayage sans orientation préférentielle. Il est également intéressant de noter que des éléments de surface triangulaires peuvent être ajustés en n'importe quel point de l'image pour fins d'interpolation et que les lignes obliques sont tracées de façon plus naturelle.

On doit cependant prévoir la nécessité d'utiliser des algorithmes de conversion entre les topologies hexagonales et cartésiennes lorsqu'il s'agit de tracer, imprimer ou afficher les images résultantes car l'ensemble des périphériques de sortie sont conçus pour accepter des images avec une grille cartésienne comme support. Ce détail a peu d'influence si on considère la vision artificielle comme un outil d'automatisation et que l'objectif premier est la représentation interne et non la production d'une image vidéo pour l'observateur humain.

1.4.3 Traitement analogique

L'utilisation des propriétés linéaires et non-linéaire des semi-conducteurs est essentielle pour l'implantation de traitements arithmétiques au plan focal. On doit utiliser de simples transistors comme amplificateurs linéaires ou non-linéaires, les grilles de transistor comme condensateurs, ces mêmes condensateurs comme intégrateurs de courant, les fils de poly-silicium comme résistance, etc.. Ce sont ces artifices de design qui permettent d'intégrer de petits calculateurs analogiques à l'intérieur de quelques μ^2.

Un calculateur analogique périphérique peut accompagner le capteur pour en augmenter la puissance de calcul et la flexibilité sans pour autant accroître la complexité électronique de chaque pixel à l'intérieur du capteur. Dans le cas d'un calculateur spécialisé externe au capteur, on peut utiliser des composantes discrètes ou encore réaliser un circuit dédié avec une technologie différente de celle du capteur. Dans cette optique, la nouvelle technologie d'assemblage de modules hybrides sur substrat de silicium est une solution attrayante. Dans ce cas, l'ensemble des circuits du système d'acquisitions s'intègre dans un même boîtier et cela permet d'augmenter considérablement la performance de même que le rapport signal à bruit des signaux analogiques. Il demeure essentiel d'exploiter au maximum les avantages du traitement analogique puisque l'élément photo-sensible fournit essentiellement un signal de ce type.

1.4.4 Extraction des résultats

La façon dont on prévoit extraire l'information visuelle d'un capteur à traitement intégré au plan focal doit faire l'objet d'une attention particulière. Par exemple, un capteur à traitement parallèle qui intègre un élément de calcul pour chaque pixel ne peut pas être exploité au maximum lorsqu'il s'agit d'extraire les résultats. La limitation du nombre possible de broches d'un circuit intégré constitue une barrière à l'extraction simultanée des différents signaux calculés en parallèle. La conception du système doit donc prévoir une stratégie d'extraction simple, cohérente et réalisable de la multitude de signaux générés par le traitement au plan focal.

Dans le cas d'un traitement en parallèle, il peut s'effectuer à la fin de chaque ligne et les données sont alors extraites colonne par colonne [16]. Cette approche pose cependant un problème de propagation des signaux en périphérie lors de la réalisation du dessin en VLSI. On peut difficilement imaginer l'extraction simultanée de 256 signaux analogiques distincts et la propagation de ces données vers 256 plots de contacts. Il est de plus obligatoire de générer une image résultante numérique pour éviter d'avoir à traiter un trop grand nombre de signaux analogiques en périphérie du capteur.

Une autre solution consiste à extraire les données d'un bon nombre de points d'une même région de l'image et d'effectuer les opérations de calcul analogique de façon complètement externe au capteur [37] [44]. Cette technique augmente la qualité et la densité de l'information générée par le système d'acquisitions d'images sans pour autant accroître de façon significative la vitesse de balayage puisqu'on se limite alors à l'analyse de l'image point par point. L'objectif poursuivi consiste à générer des données plus riches en information même si la synthèse de celles-ci ne se fait pas à un rythme accéléré. Le

projet proposé dans cette thèse utilise cette approche qui, grâce à son capteur à architecture multi-port, peut extraire simultanément l'illuminance d'une région de pixels et de les rendre disponibles en parallèle sur plusieurs canaux analogiques. On verra aux chapitres suivants que cette architecture, couplée à un traitement analogique massivement parallèle, augmente de façon significative la richesse de l'information générée par le capteur.

CHAPITRE 2

LE SYSTÈME MAR

2.1 Définition conceptuelle

On présente dans ce chapitre un système d'acquisitions d'images qui intègre un traitement au plan focal et qui répond le mieux possible aux besoins des algorithmes de vision de moyen et haut niveau. On veut s'assurer que la structure de données qu'il produit soit à la fois compacte et adaptée aux besoins des algorithmes de reconnaissance. Ce nouveau système de vision se veut compact, peu coûteux et souple au niveau de sa programmation, permettant son utilisation à une multitude d'applications. La motivation qui guide la conception du système est de produire un flot modeste de données de haut niveau plutôt qu'une suite rapide de données brutes.

Le système MAR rencontre en partie ces objectifs en proposant un capteur original qui permet d'extraire simultanément l'illuminance de plusieurs pixels situés à l'intérieur de la région avoisinante d'un point d'intérêt. Aucun traitement de signal n'est effectué sur le capteur. Un module analogique spécialisé placé en périphérie immédiate du capteur MAR se charge des opérations arithmétiques. La Figure 2.1 présente le diagramme bloc du système MAR. Ses principales caractéristiques sont la structure en boucle fermée, les

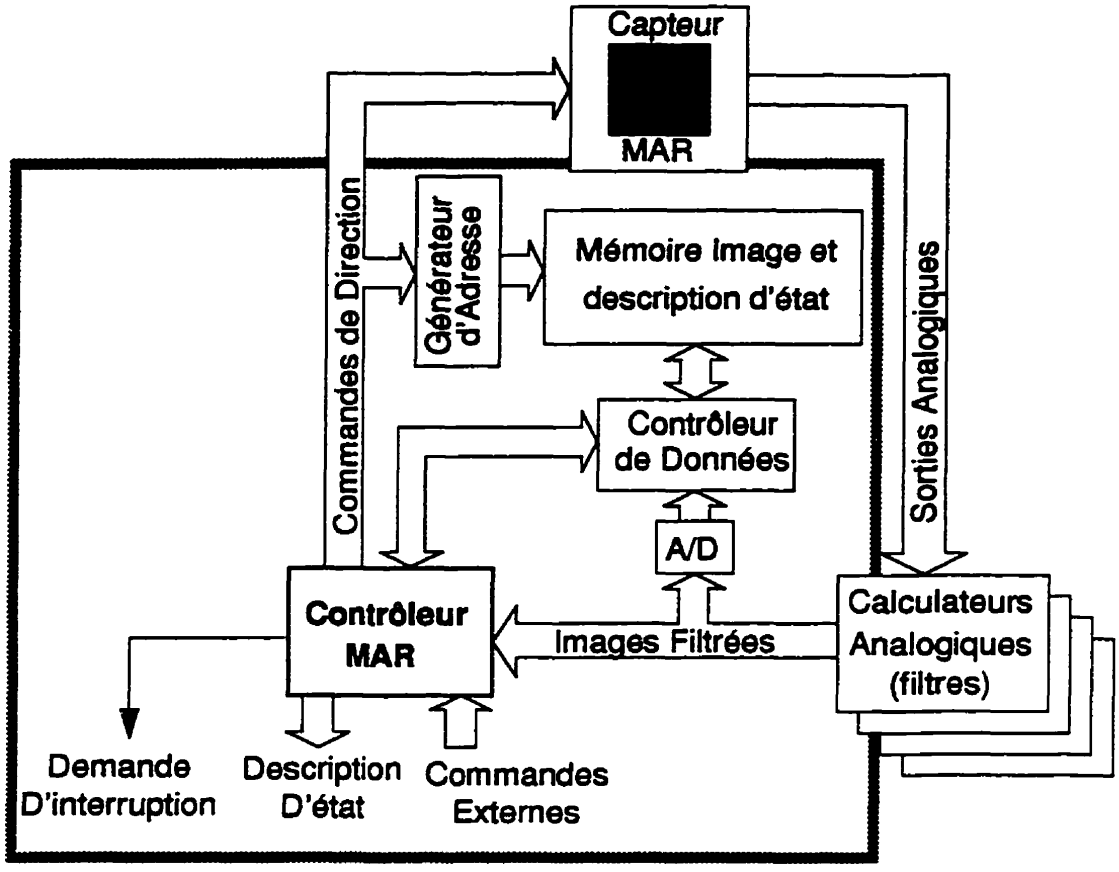


Figure 2.1 *Diagramme bloc du système MAR. Le capteur à accès multi-port est commandé par un contrôleur micro-programmé et ses sorties analogiques sont utilisées par plusieurs modules en parallèle pour le filtrage spatial. Les images filtrées peuvent être converties numériquement et stockées dans la mémoire image ou être directement utilisées par le contrôleur afin d'en extraire les passages par zéro. Le système fonctionne en boucle fermée pour effectuer le suivi d'arêtes et l'extraction des segments de droites.*

sorties analogiques en parallèle provenant du capteur, le filtrage simultané avec différents opérateurs (ou filtres) et la présence d'une mémoire image qui est balayée de la même façon que le capteur. Le capteur MAR est défini selon une grille de pixels en topologie hexagonale et un code de direction identifie le déplacement du pixel d'intérêt parmi les six directions définies par la topologie hexagonale.

La Figure 2.1 montre également la présence d'une unité de commande numérique qui est responsable des échanges d'information avec le monde extérieur. Cette unité détecte

la présence, la position et l'orientation des passages par zéro des images filtrées par l'opérateur laplacien de gaussienne [29]. Ce module supervise l'exécution d'instructions microcodées qui définissent différents modes de balayage du capteur en plus de réaliser des opérations morphologiques [2] [18]. Puisque le système calcule plusieurs images d'arêtes en parallèle, et que le suivi d'arêtes ne peut se faire que selon une seule de ces images à la fois, il est possible de dédoubler la partie encadrée à la Figure 2.1 si un plus haut niveau de performance est requis. Cette architecture est d'autant plus intéressante que l'annonce de la détection de segments de droites au système hôte se fait par voie d'interruptions et que le système externe peut potentiellement supporter plusieurs niveaux d'interruptions.

2.1.1 Jeu d'instructions spécialisé

Le système MAR doit être vu de l'extérieur comme un micro-processeur spécialisé en vision numérique dont le contenu du registre d'instructions définit le mode de fonctionnement. Chaque instruction, qui comprend différents champs, compte un total de 16 bits. Les caractéristiques de ce jeu d'instructions spécialisé sont présentées en détail au chapitre 4 (section 4.2). La séquence particulière des différentes instructions qui sont soumises à l'unité de commande par le système hôte définissent le programme d'exploitation du système MAR. Ce programme peut être une séquence rigide et prédéfinie d'instructions qui est générée par le système hôte en vue d'une tâche spécifique. On peut aussi rendre le programme d'exploitation conditionnel et ainsi réaliser une séquence adaptée à certaines caractéristiques visuelles décodées au cours même du traitement d'images. Cette procédure adaptative peut servir à isoler les régions de l'image qui seront étudiées en détail et donc diminuer le temps de traitement

Le fonctionnement global du système repose sur plusieurs registres de contrôle. Certains définissent la ou les conditions qui activent une demande d'interruption au système hôte, d'autres commandent la sélection et le routage des données générées par la caméra. Un autre ensemble de registres est utilisé pour configurer les fonctions de base du système MAR comme la position et la polarité horizontale et verticale du référentiel image, les fonctions d'arrêt temporaire, d'exécution pas à pas, etc.. Certains paramètres qui régissent la détection des passages par zéro de même que les opérations morphologiques sont également programmés par le système hôte. La fonction de chacun des registres est présentée en détail à la section 4.3. On verra également que certains de ces registres servent d'extension au registre d'instructions pour l'exécution du déplacement du pixel d'intérêt selon des modes très particuliers.

2.1.2 Structure de données hiérarchisée et description d'état

L'architecture de la caméra est définie de façon à ce que l'information générée conduise à une description détaillée de l'environnement visuel focalisé sur le capteur. Cette information peut être divisée en deux parties distinctes: l'information visuelle analogique et l'information numérique de description d'état du pixel d'intérêt. L'information analogique constitue l'image filtrée selon plusieurs résolutions spatiales et est représentée à la Figure 2.1 sous forme de bus de données analogiques.

Une caractéristique principale du système MAR est la génération de variables numériques représentant l'état du pixel d'intérêt. Cette description d'état correspond à un calcul inconditionnel de certaines propriétés dérivées des caractéristiques visuelles des trois derniers pixels visités. La description d'état comprend entre autre la direction du déplacement courant, la provenance du déplacement précédent, un code d'orientation d'arête par tranche de 30 degrés, un code de discontinuité d'arête ainsi que le sens de cette discontinuité. Certaines propriétés dérivées peuvent compléter cette description d'état comme, par exemple, la position interpolée de l'arête détectée entre les deux derniers pixels visités ou la longueur des segments d'arêtes continus. Cet ensemble de variables permet de créer une base de données décrivant hiérarchiquement les objets détectés dans la scène. Cette base de données organisée en graphe relationnel pourrait être interrogée par une application externe et ainsi guider une décision dérivée des caractéristiques visuelles de la scène. De façon générale, ces variables d'état sont redondantes et peuvent être calculées à partir d'autres informations connues mais leurs utilisation convient entre autre à une évaluation rapide et efficace de certaines variables de test dans les programmes de moyen et de haut niveau. On envisage typiquement utiliser la description d'état des trois ou quatre derniers pixels visités lors d'une requête d'interruption à l'ordinateur hôte pour définir la prochaine action à prendre.

L'ensemble des variables d'état peut être utilisé pour ajouter de l'information supplémentaire à chaque noeud de description hiérarchique dérivé de l'étude multi-résolution de la scène. Le principal problème rencontré lors de l'étude multi-résolution des images d'arêtes ou de l'intégration d'échelle est d'identifier une relation d'appartenance unique entre les arêtes détectées à deux niveaux résolution spatiales successives [36]. On observe habituellement ce problème lorsque deux filtres de résolution consécutives ont des bandes passantes de valeur trop différentes et qu'il existe trop d'arêtes à haute résolution qui peuvent être associées à une arête unique à basse résolution. Ce problème est rencontré parce qu'on essaie de limiter le nombre de convolution à effectuer lorsqu'on exécute ces

algorithmes sur un ordinateur séquentiel. Dans le cas du système MAR, la vitesse de fonctionnement est invariante en fonction du nombre d'images filtrées ce qui permet d'extraire assez d'information pour lever cette indétermination lors de la procédure d'intégration d'échelle. Il est évident qu'on ne cherchera pas à utiliser systématiquement l'ensemble complet des images filtrées pour effectuer l'intégration d'échelle car, dans bien des régions de l'image, la redondance de l'information d'un niveau de résolution à l'autre est énorme et la décision est facile à prendre.

2.2 Architecture générale du système MAR

On décrit ici les avantages et les inconvénients d'un tel système en boucle fermée ainsi que le contrôle et le mode d'accès par un système hôte. L'architecture proposée constitue une configuration de base qui pourrait être modifiée et améliorée en fonction de l'application proprement dite. Certaines instructions pouvant être exécutées de façon microcodées par la caméra réalisent leur fonction indifféremment du contenu de l'image. C'est le cas des instructions de balayage de l'image ligne par ligne et des déplacements rectilignes, hexagonaux et triangulaires. D'autre fonctions, plus évoluées, utilisent l'information extraite de la région précédemment visitée pour asservir la direction du déplacement au pixel suivant. Le suivi d'arêtes utilise entre autre cette propriété pour décrire les arrêtes comme étant une liste continue de points d'arêtes. Cette approche de suivi en boucle fermée permet de visiter un nombre restreint de points mais surtout d'établir des critères relationnels entre les arêtes détectées comme l'inclusion, l'intersection, les contours fermés, etc..

L'inconvénient majeur d'un système en boucle fermée est la limite de la fréquence d'opération. En effet, pour décider de la direction à prendre après la visite d'un point de l'image, on doit attendre que l'ensemble des variables analogiques et numériques mises en cause soient stables. Cette condition exclut toute forme de traitement en pipe-line. La vitesse de fonctionnement est donc strictement liée au temps de stabilisation des signaux. Dans le cas du système MAR, ce temps de stabilisation implique le délai de propagation du capteur entre la commande de déplacement du pixel et la sortie analogique des filtres réalisés à l'aide d'amplificateurs opérationnels. A ce délai, on doit ajouter celui des organes numériques de détection d'arêtes combiné à l'exécution du microcode.

Puisque certaines fonctions utilisent une configuration en boucle fermée et d'autre non, on ajoute à l'unité de commande un degré de flexibilité sur la fréquence de balayage. Pour ajuster dynamiquement la fréquence de balayage du capteur, on utilise la fréquence d'opération maximale de l'organe microcodé de façon permanente (environ 20 MHz)

comme horloge de base à partir de laquelle on ajoute un compteur d'attente programmable. On divise ainsi l'horloge de base ce qui laisse un temps suffisant au signal analogique pour se stabiliser. Un discussion plus approfondie sur la fréquence maximale de balayage du capteur MAR est reprise à la section 3.4.2.

L'unité de commande MAR est définie comme un périphérique esclave qui répond aux directives d'un système hôte. Son contenu adressable consiste en une trentaine de registres de 16 bits en mode d'écriture qui définissent son mode d'opération. Autant de registres sont accessibles en lecture pour informer l'ordinateur hôte de l'état du système et du contenu de l'image aux derniers points visités. Le mode de transfert des données entre le contrôleur MAR et son système maître est essentiellement asynchrone. L'interface d'entrée/sortie est d'ailleurs similaire aux conventions d'un bus de type VME. Une broche nommée CS identifie la plage adressable pour accéder au périphériques, une broche R/W définit la direction d'accès, 5 lignes d'adresses sont réservées pour accéder aux registres de contrôle et finalement, deux lignes (DS0 et DS1) identifient la période de validité du transfert des données de l'octet le moins significatif et de celui le plus signification. Les broches de contrôle servent à informer le système hôte d'une ou plusieurs demandes d'interruption en accord avec la programmation du système MAR.

La gestion des interruptions constitue une partie importante de la logique d'arbitration qui est incluse dans le contrôleur MAR. Le design du contrôleur prévoit un mode d'arrêt temporaire qui s'active automatiquement lorsqu'une demande d'interruption est signalée au système maître. Cette fonction, qui bloque la séquence normale de l'instruction en cours, est essentielle afin de permettre au gestionnaire d'interruption de lire l'information pertinente à cette interruption en provenance de la caméra avant de commander la poursuite de l'instruction en cours. On doit en effet s'assurer que la totalité du système MAR est statique lors de la lecture d'un de ses registres.

2.3 Architecture du système hôte

De façon générale, l'architecture du système hôte, de même que son programme d'exploitation, doivent être développés spécifiquement à une tâche de reconnaissance bien que certaines parties ou fonctions peuvent être généralisées. Une configuration typique comprend un microprocesseur local (ou un micro-contrôleur) configuré comme à la figure Figure 2.2 avec un bloc de mémoire dynamique pour le programme de même que pour les données temporaires et un bloc de mémoire morte pour l'initialisation du système. Un autre bloc de mémoire est partagé entre le contrôleur MAR et l'ordinateur hôte pour servir de tampon à l'information visuelle brute. Cette mémoire est d'ailleurs représentée à la Figure

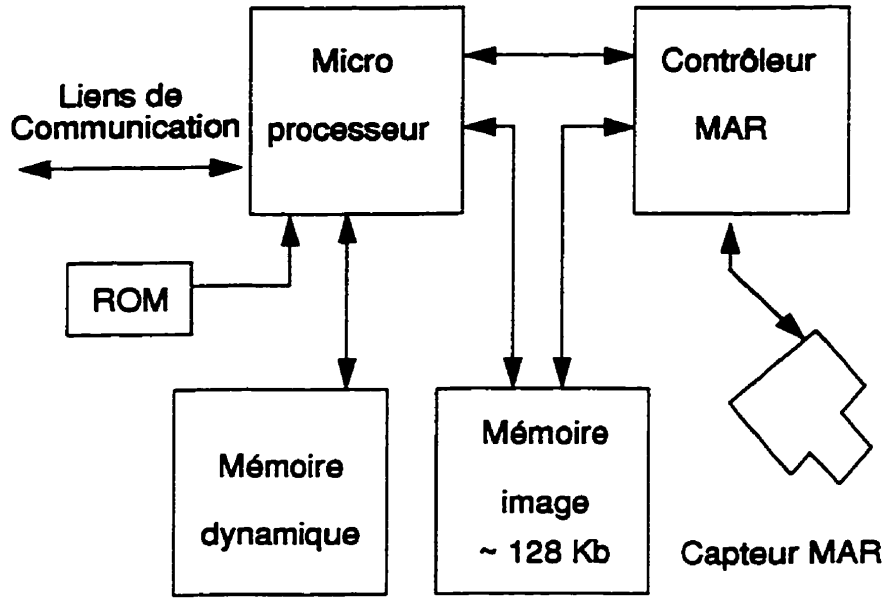


Figure 2.2 Architecture typique d'un système exploitant la caméra MAR.

2.1 et devrait avoir idéalement la même dimension que le capteur sur 16 bits de large. On choisit préférablement une mémoire à deux ports pour cette partie bien qu'on puisse utiliser une mémoire partagée dans le temps. L'avantage d'une mémoire à deux ports est que le système externe peut lire l'état des pixels mémorisés en même temps que la caméra fonctionne. Ceci implique que le système hôte peut, après une demande d'interruption signalée par la caméra, lire l'information provenant du contrôleur MAR puis relancer la recherche d'un nouveau point caractéristique sans temps mort.

La configuration proposée ici est un exemple qui peut être modifié et/ou enrichi en fonction de l'application visée. Un bon nombre de sous-systèmes qui augmentent la performance globale de la caméra peuvent être greffés au contrôleur au niveau de la supervision du transfert de données. On peut imaginer, par exemple, une queue de données (FIFO) qui mémorise l'information pertinente lors d'une demande d'interruption et qui relance automatiquement l'exécution du microcode. On obtiendrait ainsi, dans le cas d'interruptions commandées par la discontinuité des arêtes, un rythme de traitement beaucoup plus régulier et pratiquement indépendant de la longueur des segments d'arêtes.

Bien que l'ensemble des unités périphériques de commande puissent être configurées et programmés en fonction de l'application visée, le capteur MAR est l'élément central du système proposé d'acquisitions et de traitement d'images. La réalisation VLSI ainsi que la configuration électronique interne de ce capteur à traitement intégré au plan focal sont invariantes en fonction de l'utilisation qu'on veut en faire. On présente donc au chapitre suivant la description détaillée de ce capteur à accès multi-port de photo-récepteurs de même que ses limitations et des détails relatifs à son mode d'opération.

CHAPITRE 3

LE CAPTEUR MAR

3.1 Photo-capteur spécialisé

Le capteur MAR est l'élément de base du système de traitement d'images intégré au plan focal. C'est un circuit VLSI conçu pour effectuer la prise d'images 2D tout en combinant un pré-traitement sous forme de filtrage spatial. Le design du capteur MAR vise, à l'aide de la technologie actuelle, une caméra ayant une résolution spatiale acceptable de 256 x 256 pixels. On présente ici les grandes lignes du design du capteur MAR réalisé à l'aide de la technologie CMOS ainsi que ses caractéristiques opérationnelles. On explique comment les objectifs essentiels de sensibilité, de linéarité et de la possibilité d'effectuer une lecture non-destructive du signal vidéo sont rencontrés. On décrit le mode d'accès, le mode de balayage, les limitations physiques et les détails relatifs à la réalisation VLSI. On analyse également les contraintes, les inconvénients et les avantages de la topologie hexagonale pour la réalisation d'un tel capteur.

3.1.1 Compromis résolution complexité

Plusieurs études portant sur le développement de capteurs à traitement intégré au plan focal négligent d'analyser les contraintes de réalisation d'un capteur avec une résolution spatiale acceptable [16] [21] [34]. La technologie CMOS actuelle ne permet pas de fabriquer un circuit intégré dont les dimensions dépassent celles d'un carré de 2 cm de côté. Si on dépasse cette limite, on obtient un taux de circuits sans défauts (taux de rendement) négligeable et on risque fort, par le rejet d'un trop grand nombre de circuits, de tomber sous le seuil de la rentabilité ou encore de forcer une étape de test trop lente. Il est difficile d'envisager l'ajout d'une unité de calcul trop élaborée sur chaque pixel puisque la surface d'intégration disponible est limitée. En fait, si on se limite à la fabrication d'un capteur de 256x256 pixels sur un dé de silicium de 1.5 cm à 2 cm de côté, chaque pixel ne peut occuper une surface carrée dépassant 50 à 70 microns de côté.

Pour ces raisons, nous avons conçu un capteur dont l'emphase a été mise sur la complexité du mode d'extraction de l'information visuelle permettant un calcul de complexité variable en périphérie plutôt que de définir un système dont le traitement est intégré au niveau du pixel mais dont la souplesse de fonctionnement est limitée.

3.1.2 Opto-électronique et technologie CMOS

La fabrication de prototypes se fait grâce à la *Société Canadienne de Microélectronique* et donne accès aux technologies CMOS 3 μ et CMOS 1.2 μ, à l'Arséniure de Galium, et à la technologie bipolaire. Il est donc impératif de définir, dans le cas d'une recherche universitaire exploratoire, le design de circuits opto-électroniques dans l'une ou l'autre de ces technologies. Dans le cas précis de capteurs bidimensionnels à haute résolution, on doit intégrer un grand nombre de photo-détecteurs (jusqu'à 400 K transistors) sur un même circuit intégré sans drainer un trop grand courant d'alimentation. La technologie CMOS est donc le seul choix rencontrant ces critères de faible consommation de puissance et de haute capacité d'intégration [47].

La technologie CMOS permet de réaliser des photo-diodes et des photo-transistors de qualité satisfaisante sur substrat de silicium dans la partie visible du spectre électromagnétique de même que dans celle de l'infrarouge proche. Le silicium a une énergie d'activation Eg = 1.2 eV [3] et est donc transparent à toute lumière de longueur d'onde plus grande que 1.1 μm (ex.: infrarouge lointain) et opaque pour la lumière de longueur d'onde inférieure à 1.1 μm. La courbe d'absorption du silicium [43] est expliquée par la transformation d'un photon incident d'énergie supérieure à Eg (absorption) en une

paire électron-trou. Comme on peut le voir à la Figure 3.1, dans le silicium intrinsèque, la recombinaison est très probable car la paire électron-trou reste voisine et le photon est réémis après un temps de vie moyen des porteurs τ. Cette propriété est facilement

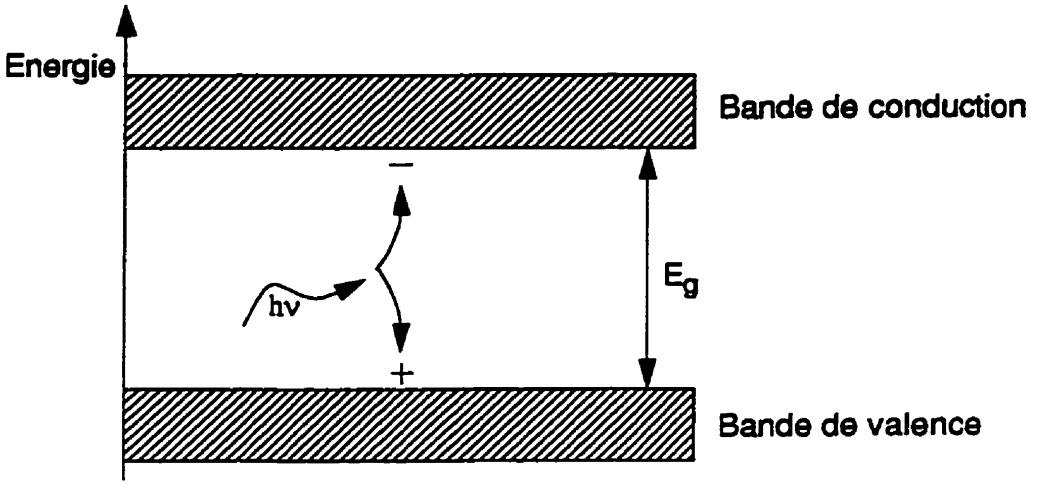


Figure 3.1 *Création d'une paire électron-trou dans un substrat de silicium intrinsèque. Un photon de longueur d'onde λ inférieure a 1.1 μm est absorbé mais, après un temps de vie moyen des porteurs τ, la recombinaison électron-trou se produit et un photon est réémis.*

observable à l'aide d'une lunette d'observation pour infrarouge. En projetant une lumière composite au travers d'une pastille de silicium, on ne voit rien traverser à l'oeil nu alors qu'on voit clairement le faisceau à l'aide de la lunette infrarouge. La Figure 3.2 illustre le cas d'une jonction PN où la création d'une paire électron-trou causée par l'incidence d'un photon d'énergie suffisante est suivie d'une migration due à la présence du champ électrique présent à la jonction. Puisque les charges sont éloignées dès leur création, la recombinaison est peu probable et il s'en suit un courant électrique de la région N vers la région P (conduction inverse). La Figure 3.2 explique également la forme non-linéaire de la bande de conduction autours de la jonction. Puisque les charges fixes autours de la jonction impliquent un champ électrique qui est la primitive du patron de charges même lorsque la jonction n'est pas polarisée, il s'en suit une variation spatiale du potentiel électrique qui est la primitive du patron de champ électrique. Il est intéressant de constater que la barrière de potentiel V_γ représente la limite de conduction en direct d'une diode au silicium soit environ 0.6V. Evidemment, la répartition des charges fixes ne varie pas de

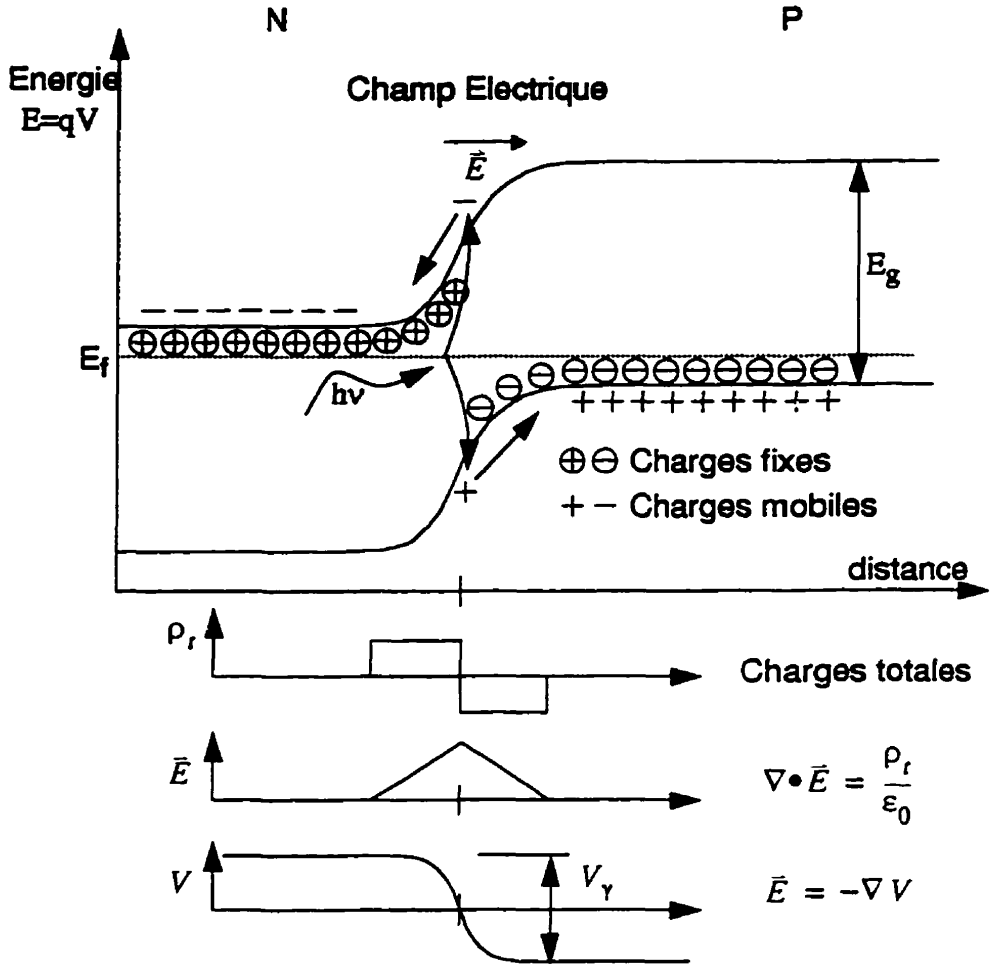


Figure 3.2 Visualisation des charges mobiles près d'une jonction PN non polarisée pour le silicium. L'existence du champ électrique à la jonction explique la migration dans des directions opposées (courant électrique) des paires électrons-trou crées dans la région de charges d'espace. Les équations de Maxwell expliquent la forme approximativement parabolique des bandes de valence et de conduction des deux types de dopant.

façon aussi simple mais cette analogie qualitative est suffisante pour les besoins explicatifs de cette section.

La région de charges d'espace représentée par w varie avec la tension de polarisation à l'inverse appliquée aux bornes de la jonction PN. Plus la jonction est polarisée, plus le champ électrique est fort et plus la région de charges d'espace w est grande. La Figure 3.3 représente graphiquement l'allure des bandes de conduction et de

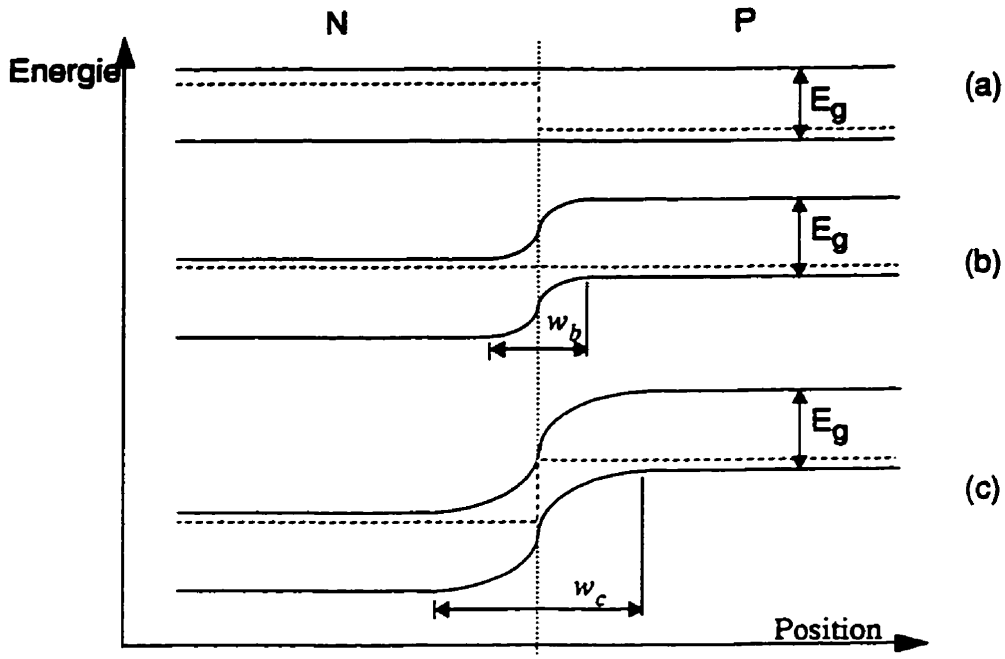


***Figure 3.3** Variation de la largeur de la région de charges d'espace pour une jonction PN polarisée (a), non-polarisée (b), et polarisée inversement (c). Les niveaux de Fermi (E_f) sont représentés par des lignes pointillées. On constate que w est relié à la valeur de la tension inverse de polarisation.*

valence d'une jonction PN polarisée de différentes façons. Les niveaux de Fermi E_f sont représentés par les lignes pointillées et la différence entre le niveau de Fermi du type N et celui du type P est principalement associé à la façon dont la jonction PN est polarisée. Dans le cas d'une jonction polarisée directement (Figure 3.3 (a)), la disparité des niveaux de Fermi est relié à la tension du seuil de conduction V_γ comme il est d'ailleurs montré à la Figure 3.2. Si la jonction est non polarisée (Figure 3.3 (b)), les niveaux de Fermi coïncident alors que dans le cas d'une jonction polarisée inversement (Figure 3.3 (c)), on éloigne d'autant l'alignement des niveaux de Fermi et la région de charges d'espace se trouve alors élargie proportionnellement à la tension inverse de polarisation.

Les exemples présentés à la Figure 3.2 et à la Figure 3.3 illustrent le cas d'une même concentration de dopant de part et d'autre de la jonction et qui est de l'ordre de $10^{16} particules/\text{cm}^3$. Afin d'assurer la création d'un contact ohmique entre la couche de

métallisation et les régions de diffusion, on doit doper fortement la région des contacts [43]. Ces régions fortement dopées atteignent des concentration de l'ordre de $10^{20} particules/cm^3$. Pour ces raisons, les courbes de la Figure 3.2 prennent une allure différente pour une jonction PN^+et sont représentées à la Figure 3.4

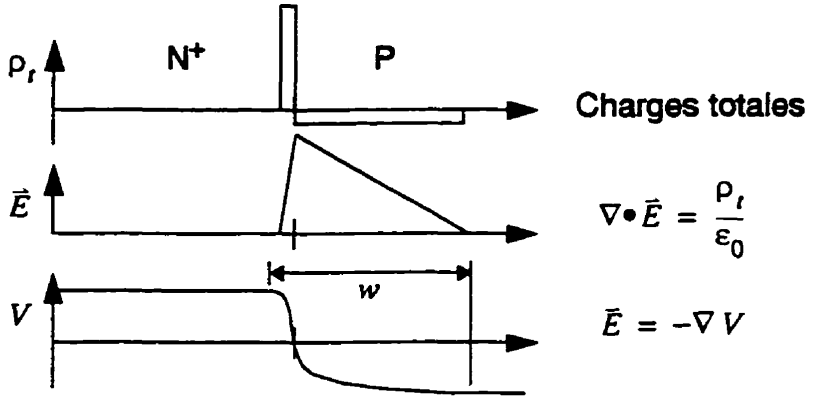


Figure 3.4 Représentation graphique des charges fixes et la variation spatiale de potentiel pour une jonction PN^+

Pour la réalisation pratique d'une photo-diode en technologie CMOS, on utilise une région de diffusion N^+ dans un substrat de type P faiblement dopé. On désigne ce type de circuit photosensible par diode PIN (pour P-Intrinsèque-N). Par l'emploi d'un substrat de type P, on permet l'intégration voisine de transistors de type N pour la sélection et l'amplification. On évite ainsi d'utiliser les deux types de substrats dans la région de la matrice de capteurs ce qui minimise la taille des pixels. La Figure 3.5 donne le détail en coupe de la réalisation d'une photo-diode PIN. La région de charges d'espace w représentée est fortement photo-sensible bien que la recombinaison locale est peu probable à l'intérieur de la longueur de diffusion L ce qui rend cette région également photo-sensible. Il est de toute façon essentiel de garder une distance minimale L entre la diffusion N^+ et la diffusion P^+ de façon à garantir la réalisation d'une diode PIN. Le cas échéant, on réalise une jonction tunnel de type P^+-N^+ qui a la propriété de conduire le courant comme une résistance lorsqu'elle est polarisée à l'inverse [3].

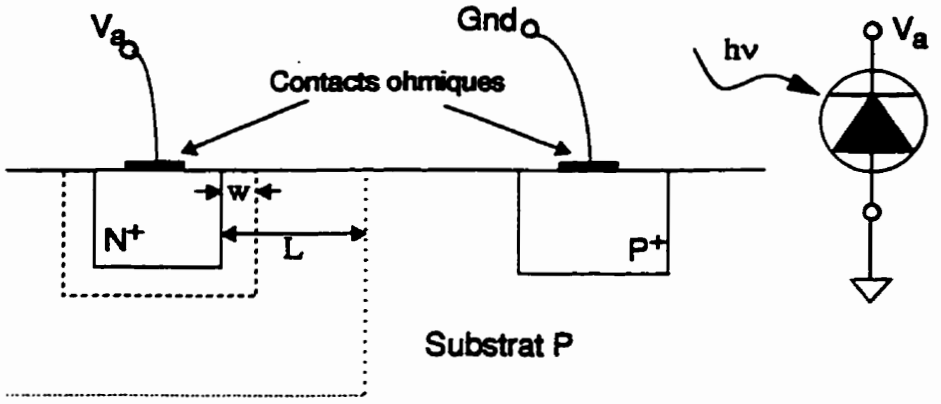


Figure 3.5 *Photo diode PIN en technologie CMOS. Les diffusions P^+ et N^+ sont utilisées pour effectuer un contact ohmique. la région photo-sensible est délimitée par la région de charges d'espace w bien que la création d'un photo-courant est probable à l'intérieur de la longueur de diffusion L.*

Il est essentiel de connaître les dimensions physiques de la région de charges d'espace qui est photo-sensible pour une diode PIN. Ces dimensions correspondent à la distance maximale d'où peut être créée une paire électron-trou pour ensuite être attirée par le champ électrique de la jonction PN. Il faut prendre en considération ces valeurs d'espacement minimales définies par w et de L lors de la réalisation du dessin des masques du pixel. On définit la profondeur de la région de charges d'espace w selon la relation [43][35]:

$$w = \sqrt{\frac{2\varepsilon (V_a + V_\gamma)}{|q| N_d}} \tag{3-1}$$

où V_γ est la tension en polarisation inverse, ε est la constante diélectrique du silicium soit $11.9\varepsilon_0$, q est la charge de l'électron et N_d est la concentration de dopant de la partie intrinsèque. La longueur de diffusion L est définie selon la relation suivante [35]:

$$L = \sqrt{D\tau} \tag{3-2}$$

avec D la constante de diffusion définie par l'équation (3-3) et τ le temps de vie moyen des porteurs qui est de l'ordre de 4 ns pour le silicium faiblement dopé.

$$D \cong \frac{\mu_n kT}{q} = \frac{\mu_n}{40} \tag{3-3}$$

Pour la technologie CMOS4S de Northern Telecom à grille de 1.2 µ on a $N_d = 1.5x10^{16}/cm^3$ et $\mu_n = 378.4\ cm^2/V\text{-}s$ [12] ce qui donne $w \cong 0.7$ µm et $L \cong 1.9$ µm. On doit donc s'assurer que la région bordant la frontière N^+ soit bien exposée à la lumière sur environ 2 µm du côté du substrat (région faiblement dopée). Il est de plus impératif de garder une distance minimale de 2 µm entre le contact de polarisation du substrat N et la jonction P^+N bien que les règles de design de la technologie CMOS4S à grille de 1.2 µ de Northern Telecom exigent de toute façon un espacement minimum de 3.2 µm entre les deux types de diffusion dans un même substrat [12].

Le photo-transistor réalisé avec la technologie CMOS est en fait une jonction base-émetteur de type PIN placée dans un substrat jouant le rôle de collecteur. Comme on peut le voir à la Figure 3.6, on réalise un photo-transistor NPN en plaçant une diffusion N^+ dans un puits P flottant jouant le rôle de la base, le tout est placé dans un substrat de type N polarisé positivement. Deux désavantages sérieux nous forcent à rejeter ce type de photo-

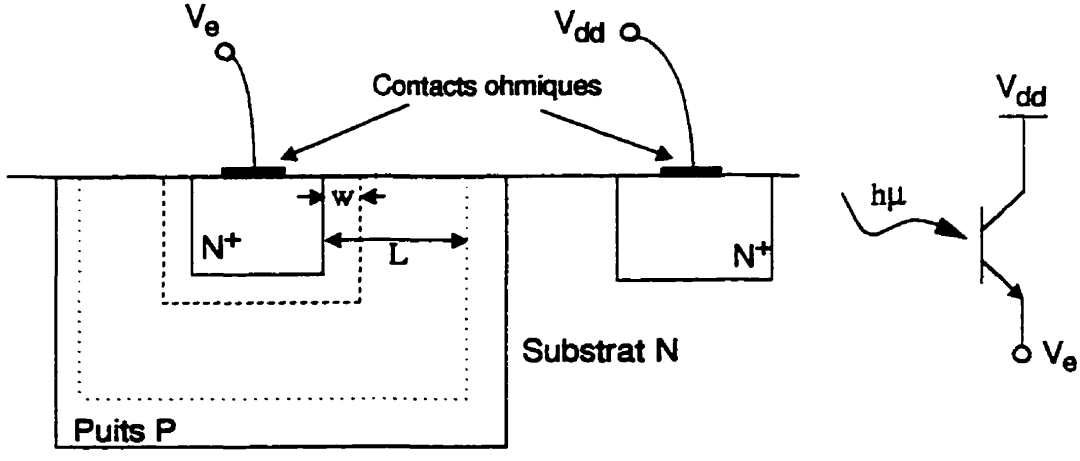


Figure 3.6 Photo-transistor NPN réalisé en technologie CMOS à puits P dans un substrat N. Le puits P flottant joue le rôle de la base flottante et le substrat N le rôle du collecteur.

détecteur dans le cas de la réalisation d'un capteur bidimensionnel. Premièrement, la dimension d'un photo-transistor est beaucoup plus grande que celle d'une photo-diode parce que les règles de design forcent un grand espacement entre les deux types de substrat pour éviter l'apparition de tyristors parasites [48]. Deuxièmement, le gain en courant entre le photo-courant de la base et le courant de sortie à l'émetteur n'est habituellement pas très élevé puisque la technologie n'est pas prévue pour ces applications. Une technologie BiCMOS serait peut-être plus favorable. Cependant, pour obtenir la plus haute densité possible, on n'utilisera que des photo-diodes NMOS dans la réalisation des pixels du capteur MAR.

3.1.3 Photo-capteur à accès multi-port non destructif

Le principe exploité par le capteur MAR consiste à extraire simultanément l'information provenant de plusieurs pixels à l'intérieur d'une région d'intérêt. Dans cette optique, lors d'un balayage complet du capteur, chaque pixel est lu autant de fois qu'il y a de pixels dans le masque de convolution. Il est donc essentiel que la lecture soit non destructive, ce qui est très différent d'une caméra CCD qui opère en initialisant le processus d'intégration de chaque ligne immédiatement après une lecture. On réalise l'extraction de l'information d'un pixel selon un masque qui accède aux trois diagonales principales d'une matrice bidimensionnelle de détecteurs organisés selon une topologie hexagonale à la manière d'une mémoire multi-ports.

La Figure 3.7 nous montre l'architecture d'adressage multi-ports d'une source de courant unique de valeur I_S dont l'amplitude est proportionnelle à l'intensité lumineuse. A l'aide de trois signaux de sélection différents S_Y, S_{X1} et S_{X2}, on dirige le courant de la source I_s sur l'un des trois bus de données analogiques distincts Y, $X1$ et $X2$. La définition de la dimension des transistors est l'une des tâches principales de la réalisation VLSI d'un circuit, qu'il soit analogique ou numérique. Dans le cas du design du pixel MAR, la taille des trois transistors de sélection influence directement les caractéristiques tension-courant du circuit ainsi que la linéarité de sa réponse au signal lumineux. Ce choix important est guidé par la valeur maximale du courant I_S débité par le photo récepteur. Une étude plus approfondie de ce détail d'implantation est présentée à la section 3.1.4.

Pour effectuer une lecture non destructive, le courant I_S d'un pixel donné doit rester constant peu importe le nombre d'accès qu'on y fait et doit être invariant quant à la durée des lectures effectuées. C'est d'ailleurs la raison qui nous pousse à utiliser un signal en courant plutôt qu'un signal en tension. Il est en effet très difficile, à l'aide de la technologie

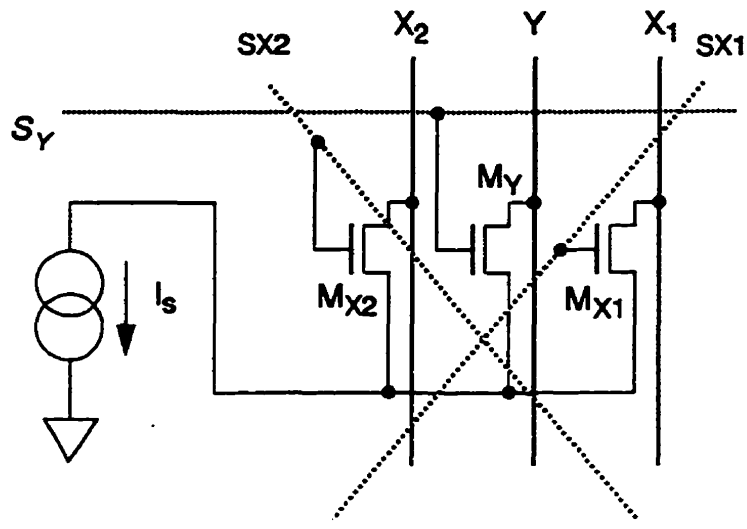


Figure 3.7 Architecture multi-port du capteur MAR. Chaque pixel comprend une source de courant d'amplitude proportionnelle à l'intensité lumineuse. Le courant I_S peut être dirigé sur l'un ou l'autre des trois bus de données analogiques Y, X1 ou X2. Les trois lignes de sélection accèdent aux trois diagonales principales de l'organisation hexagonale du capteur MAR à une orientation relative de 60 degrés.

CMOS, de générer un bon amplificateur de tension alors qu'un simple transistor NMOS est, de façon intrinsèque, un amplificateur à transconductance et donc une source de courant commandée par une tension.

La Figure 3.8 donne le détail de deux configurations possibles pour réaliser, de façon simple et compacte une source de courant commandée par un signal lumineux. Le circuit (a) est un photo-détecteur à intégration d'illuminance alors que le circuit (b) est un photo-détecteur à valeur instantanée d'illuminance. Comme on peut le voir à la Figure 3.8, le circuit à intégration d'illuminance utilise le photo-courant I_E pour varier la tension V_g en déchargeant la capacité de la grille du transistor M_1. Le transistor M_2 est utilisé pour initialiser le processus d'intégration d'illuminance en plaçant la tension V_{Reset} moins la tension de seuil V_{SN} d'un transistor N [40] sur la grille du transistor M_1. Cette tension de seuil V_{SN} a une valeur d'environ 2V lorsque la tension V_{Reset} a une valeur fortement positive (~5V). Le signal de tension V_{Reset} peut d'ailleurs être propagé avantageusement par l'un des bus de données analogiques puisque ceux-ci ne sont pas en opération lors d'une procédure de remise à zéro de l'intégration d'illuminance.

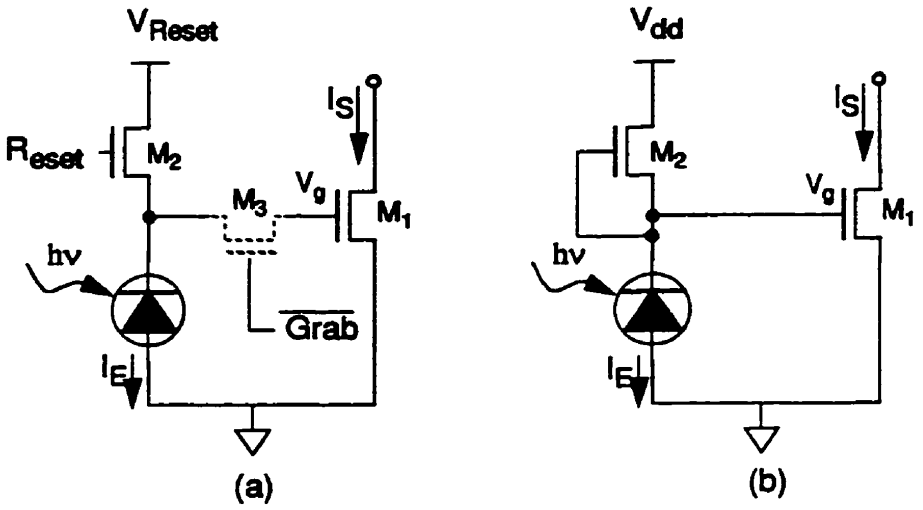


Figure 3.8 *Différentes configurations de sources de courant commandées par un signal lumineux. Le circuit (a) intègre le photo-courant I_E pendant une période de temps donnée grâce à la capacité de la grille du transistor M_1 alors que le circuit (b) utilise le transistor M_2 en mode linéaire comme convertisseur courant-tension. Le transistor M_1 est utilisé comme amplificateur de transconductance qui convertit la tension de grille V_g en courant I_S.*

Dans le cas de la Figure 3.8 (a), plus la lumière est intense, plus le photo-courant I_E est fort, plus la tension du noeud V_g baisse rapidement et plus le courant I_S final est faible. Après un temps d'intégration t_i commun à tous les pixels, chaque pixel peut commander un courant I_S correspondant à son illuminance locale. Il est de plus important de noter que le noeud V_g est électriquement isolé des transistors de sélection ce qui rend la lecture non destructive. Le transistor M_3, qui est optionnel, peut être utilisé pour stopper le processus d'intégration lumineuse durant le temps requis pour effectuer la lecture complète du capteur. Dans le cas contraire, on peut utiliser un obturateur électromécanique placé dans le chemin optique de la caméra pour bloquer l'arrivée de la lumière.

Le circuit (b) de la Figure 3.8 utilise le transistor $M2$ en mode linéaire pour agir comme résistance et ainsi définir une tension V_g proportionnelle au photo-courant [15], et ce de façon continue et instantanée. Le transistor M_2 et la photo-diode agissent alors en mode de diviseur de tension. Ce circuit a l'avantage de donner la valeur immédiate du signal lumineux mais il est impossible d'ajuster dynamiquement la sensibilité du capteur. Il faut en fait agir sur l'ouverture de la lentille et l'éclairage de la scène de façon à calibrer le niveau de blanc. Ce capteur à lecture instantanée utilise un transistor N en mode non-

linéaire qui, combiné à l'effet de substrat, implique une forte non-linéarité de la tension V_g en fonction de l'illuminance. Le circuit (a) de la Figure 3.8 a l'avantage d'effectuer cet ajustement en modifiant le temps d'intégration t_i de façon à calibrer le niveau de blanc maximum correspondant à la saturation d'un pixel. On peut également prévoir un circuit spécialisé qui automatise cette calibration du niveau de blanc associé à la saturation du pixel.

La Figure 3.9 présente le détail temporel de fonctionnement du capteur à intégration d'illuminance. On peut y distinguer les deux cas extrêmes de luminosité: le noir et le blanc. On effectue la calibration du circuit en choisissant la région la plus blanche et en fixant le temps d'intégration t_i de façon à obtenir un courant I_S pratiquement nul pour cette intensité lumineuse. Dans cette condition, on a une tension de grille légèrement supérieure à la limite de conduction d'un transistor MOS $V_t + \varepsilon$. On obtient ainsi le meilleur rapport signal à

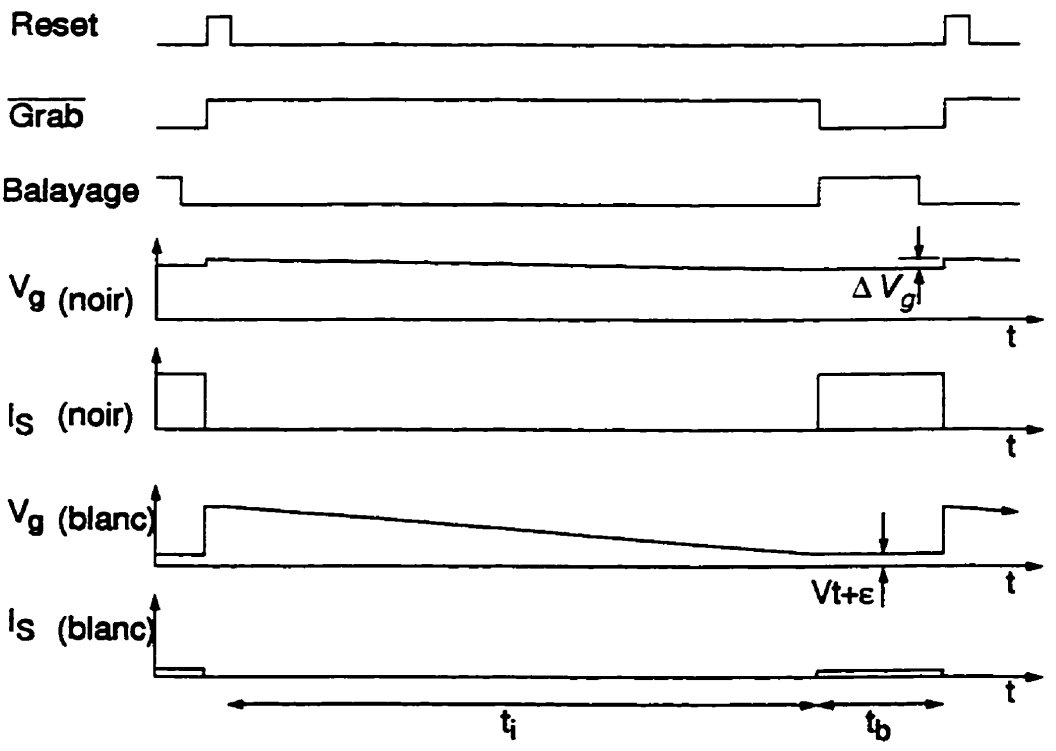


Figure 3.9 Diagramme temporel représentant la procédure complète de la prise d'images et de son extraction pour un capteur à intégration d'illuminance comme celui de la Figure 3.8 (a). On distingue la forme du signal analogique pour les cas avec (blanc) et sans (noir) lumière.

bruit possible pour une situation donnée bien qu'il puisse être favorable de limiter la tension de grille V_g à une valeur minimum légèrement supérieure à $(V_t + V_{min})$ afin d'utiliser une portion plus linéaire de la relation tension courant du transistor M_1. Cette approche est d'ailleurs présentée à la section suivante par l'équation (3-4). Avec la configuration de la Figure 3.8 (a), la procédure de la calibration du niveau de blanc peut être rendue adaptative. On peut même laisser saturer volontairement certains pixels qui reçoivent une lumière trop intense comme c'est le cas pour une source lumineuse dans la scène qui pointe vers la caméra.

La Figure 3.9 présente également le cas où on utilise le transistor M_3 de la Figure 3.8 (a) pour bloquer le processus d'intégration pendant la durée t_b du balayage du capteur. L'effet du courant de fuite de la photo-diode qui cause une variation de tension ΔV_g même dans le noir total est également illustré [21]. La durée de l'intégration t_i est d'ailleurs limitée par la valeur relative de ce courant de fuite. On doit en effet effectuer la prise d'images ainsi que le balayage du capteur dans un intervalle de temps le plus court possible afin de minimiser l'effet du courant de fuite. Cette condition est généralement respectée lorsque l'éclairage de la scène est suffisamment intense et que l'ouverture de la lentille est grande. Le fonctionnement du prototype de la caméra à permis de démontrer que le temps requis par le courant de fuite pour décharger complètement le noeud V_g dans le noir absolu est de l'ordre de 3 à 4 secondes. Il faut donc que la durée de la prise d'images, qui est de l'ordre de 150 ms, soit suffisamment courte pour considérer comme négligeable les effets négatifs du courant de fuite.

3.1.4 Linéarité de la réponse du pixel en fonction du signal lumineux

Plusieurs effets non-linéaires sont associés à la configuration électronique d'un pixel du capteur MAR. Le courant qui est drainé par chaque pixel sélectionné passe par un multiplexer analogique (section 3.3.5) avant de joindre la broche de sortie du capteur. Si on combine la Figure 3.7 et la Figure 3.8, et qu'on ajoute le transistor requis par le multiplexer analogique de sortie, on obtient le schéma de la Figure 3.10 où V_{pp} est la tension de polarisation des sorties analogiques du capteur. Un premier effet non-linéaire provient du fait que la sensibilité de la photo-diode dépend de la tension de polarisation V_g appliquée à l'inverse selon la relation (3-1). Comme condition initiale, la tension V_g est fixée par la référence globale $(V_{Reset} - V_{SN})$ puis, à mesure que le photo-courant I_E décharge la capacité de grille du transistor M_1, la largeur de la région de charges d'espace w diminue

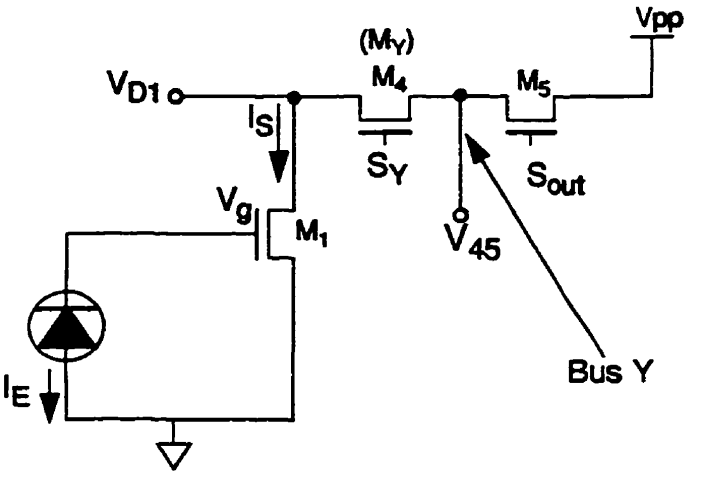


Figure 3.10 Configuration électronique d'un pixel adressé par le bus Y. Les transistors M_4 et M_5 sont utilisés comme transistors de sélection et leurs signaux de grille respectifs S_Y et S_{out} sont des commandes numériques à 5 V. Le courant I_S est une fonction approximativement quadratique de Vg mais cette relation peut être modifiée en ajustant la dimension des transistors M_1, M_4 et M_5

de même que la sensibilité du photo-détecteur. Le transistor M_1 qui joue le rôle de convertisseur tension-courant réagit en mode saturé ($V_{D1} > V_g - V_t$) selon la relation suivante:

$$I_S = \frac{\beta}{2}(V_g - V_t)^2 \tag{3-4}$$

Dans l'équation (3-4), β est défini par la forme géométrique du transistor M_1 qui est proportionnelle au rapport longueur/largeur (w/l) du canal alors que V_t est la tension de seuil du transistor N soit 0.75 V [12].

On présente dans ce qui suit l'expression analytique du courant de sortie I_S en fonction de l'illuminance E_{in}. On considère le photo-courant comme une fonction linéaire de la largeur de la région de charges d'espace w tel que discuté à la section 3.1.2. Puisque la relation courant-tension de la capacité de grille s'écrit:

$$V_g = (-\frac{1}{C_g})\int I_E dt \tag{3-5}$$

ou encore

$$I_E = -C_g \frac{dV_g}{dt} \tag{3-6}$$

et que l'équation (3-1) peut être récrite dans ce cas particulier sous la forme:

$$I_E \propto E_{in} w = E_{in} \sqrt{\frac{2\varepsilon (V_g + V_\gamma)}{q N_d}} \tag{3-7}$$

ou

$$I_E = E_{in} K \sqrt{V_g + V_\gamma} \tag{3-8}$$

où E_{in} représente l'illuminance au pixel considéré et $K = \sqrt{\frac{2\varepsilon}{q N_d}}$, la constante de sensibilité de la photo-diode au silicium, on peut calculer la valeur finale de la tension de grille V_g après un temps d'intégration t_i du photo-courant I_E. En combinant les équations (3-5) et (3-7), on obtient l'équation différentielle suivante:

$$\frac{dV_g}{dt} = (-\frac{1}{C}) I_E = (-\frac{K}{C}) E_{in} \sqrt{V_g + V_\gamma} \tag{3-9}$$

d'où

$$\int_{(V_{Reset} - V_{SN})}^{V_f} \frac{dV_g}{\sqrt{V_g + V_\gamma}} = (-\frac{K}{C}) E_{in} \int_0^{t_i} dt \tag{3-10}$$

On déduit la relation de V_f qui est la tension de grille finale après le temps d'intégration t_i en fonction du degré d'illuminance E_{in} soit:

$$2 (\sqrt{V_f + V_\gamma} - \sqrt{V_{Reset} - V_{SN} + V_\gamma}) = (-\frac{K}{C}) E_{in} t_i \tag{3-11}$$

donc:

$$V_f = (\sqrt{V_{Reset} - V_{SN} + V_\gamma} - \frac{K}{2C} E_{in} t_i)^2 - V_\gamma \tag{3-12}$$

En plaçant l'équation (3-12) dans l'équation (3-4) avec $V_g = V_f$ on obtient la fonction qui relie le courant de sortie final du pixel I_S à l'illuminance E_{in} et qui s'écrit:

$$I_S = \frac{\beta}{2}\left(\left[\left(\sqrt{V_{Reset} - V_{SN} + V_T} - \frac{K}{2C}E_{in}t_i\right)^2 - V_T\right] - V_t\right)^2 \qquad (3\text{-}13)$$

L'équation (3-13) démontre bien les effets non-linéaires de la réponse du capteur MAR en fonction de l'intensité lumineuse. On retrouve des termes en E_{in}^4 et en E_{in}^2 alors que l'ensemble des autres termes sont communs à tous les pixels en supposant une uniformité du procédé de fabrication. On peut également remarquer que dans le cas particulier du noir $(E_{in} = 0)$, on obtient comme prévu, le courant maximal drainé par le transistor M_1 soit:

$$I_S = \frac{\beta}{2}\left(\left[V_{Reset} - V_{SN}\right] - V_t\right)^2 \qquad (3\text{-}14)$$

L'équation (3-13) n'est valide que si la tension V_{DS} du transistor M_1 (V_{D1}) est assez grande pour considérer ce dernier en mode saturé. Ceci est rendu possible en définissant la dimension physique des transistors M_4 et M_5 ($(w/l)_4$ et $(w/l)_5$) avec leur largeur plus grande que celle du transistor M_1. Puisque, à la Figure 3.10, c'est le même courant I_S qui traverse les trois transistors M_1, M_4 et M_5, et que la somme des trois tensions V_{ds} doit être forcément égale à V_{pp}, on aura des familles de courbes I_{ds} fonction de V_{ds} et V_{gs} comme celles montrées à la Figure 3.11. On peut voir que le transistor M_1 est bel et bien opéré en mode saturé.

Il est possible de modifier la forme quadratique de l'équation (3-13) en rajustant légèrement la dimension du transistor M_4 de façon à garder le transistor M_1 en mode de saturation pour V_g faible et en mode non-linéaire pour V_g grand et ainsi rendre sigmoïdale la réponse du courant I_S en fonction de la tension de grille V_g. Dans ce cas particulier, on peut voir à la Figure 3.12 le lieu des régions d'opération qui gardent la somme des tensions V_{ds} égales à V_{pp}.

Une façon de limiter les effets non-linéaires associés à la configuration de la Figure 3.10 est de n'utiliser qu'une partie restreinte de la plage dynamique disponible. Évidemment, ce choix a pour conséquence de réduire le rapport signal à bruit mais demeure

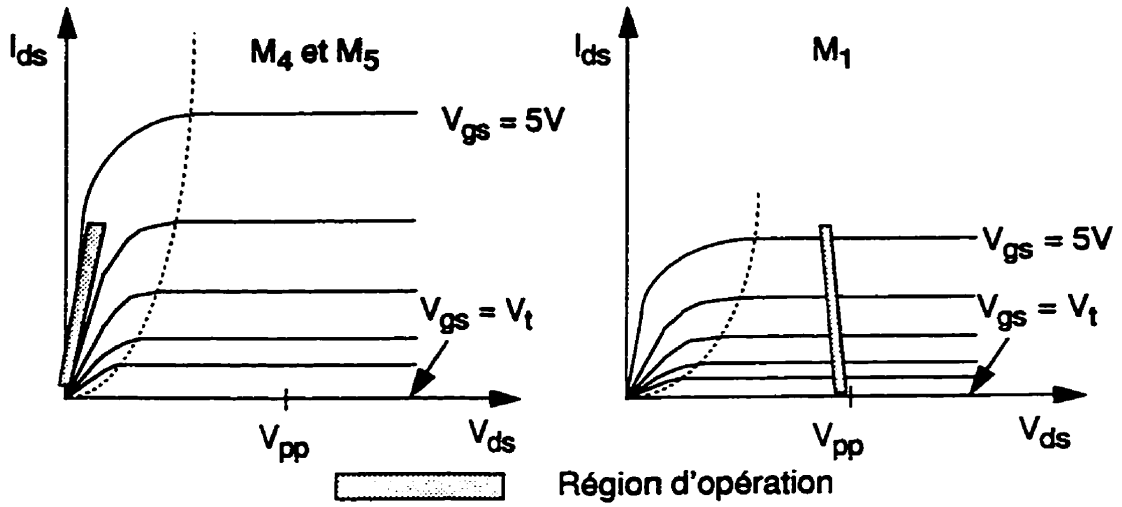


Figure 3.11 Familles de courbes de I_{ds} en fonction de V_{ds} et V_{gs} pour les trois transistors de la Figure 3.10. Puisque la largeur des transistors M_4 et M_5 est plus grande que celle du transistor M_1, que le courant est identique pour les trois et que la tension V_{gs} des transistors M_4 et M_5 est pratiquement à 5 V, on constate que la région d'opération du transistor M_1 est limitée à la zone de saturation.

un compromis acceptable dépendant de l'application visée. De toute façon, on peut effectuer la détection d'arêtes tant sur l'image originale que son logarithme et obtenir sensiblement les mêmes résultats. Il est de plus connu que la plupart des systèmes visuels biologiques ont une réponse non-linéaire, souvent même logarithmique, en fonction de l'amplitude de l'intensité lumineuse captée [9].

3.2 Modes de balayage

Le capteur MAR comprend une partie centrale composée d'une matrice bidimensionnelle de photo-capteurs. Chaque pixel a une capacité d'adressage multi-port organisée selon une topologie hexagonale. Un circuit numérique périphérique commande correctement les différentes lignes de sélection. Cette section décrit le mode de balayage, l'architecture des registres à décalage ainsi que la procédure d'initialisation du capteur. La Figure 3.13 donne l'organisation générale des quatre différents registres à décalage requis pour effectuer la sélection multi-port ainsi que la commande du multiplexer analogique de sortie. Par convention, la direction de décalage est définie comme étant positive dans le sens des flèches en gras pour chacun des quatre registres à décalage de la Figure 3.13.

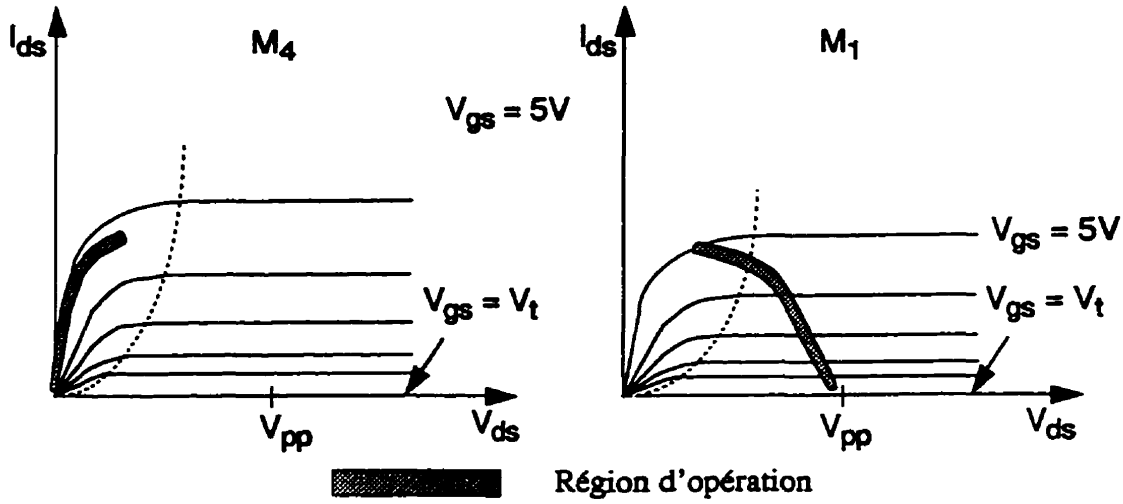


Figure 3.12 Modification de la réponse du pixel par l'utilisation d'un transistor M_4 plus petit. On observe le déplacement de la région d'opération du transistor M1 dans la zone non-linéaire et on rend ainsi la réponse sigmoïdale.

L'adressage de la région d'intérêt permet d'extraire l'information analogique de plusieurs pixels simultanément grâce à l'architecture à accès multiport. Le pixel d'intérêt, qui est au centre de cette région, peut être déplacé d'une position à la fois dans l'une des six directions définies par la topologie hexagonale. Les déplacements relatifs du pixel d'intérêt se résument en commandes de décalage spécifiques pour chacun des quatre registres du capteur afin de garder un synchronisme spatial des lignes de sélections de même que pour la commande du multiplexer analogique de sorties. Les sous-sections qui suivent traitent du fonctionnement et de l'initialisation des registres périphériques du capteur.

3.2.1 Registres à décalage

Le mode de balayage du capteur MAR est conçu pour permettre au système externe de déplacer le pixel d'intérêt dans n'importe laquelle des six direction définies par la topologie hexagonale. Pour ce faire, on doit conserver les trois diagonales de sélection actives et concurrentes au pixel d'intérêt. Afin de déplacer le pixel d'intérêt dans une direction ou de le laisser stationnaire pendant une période de temps indéterminé, chaque registre à décalage doit pouvoir avancer, reculer ou rester sur place. La Figure 3.14 donne

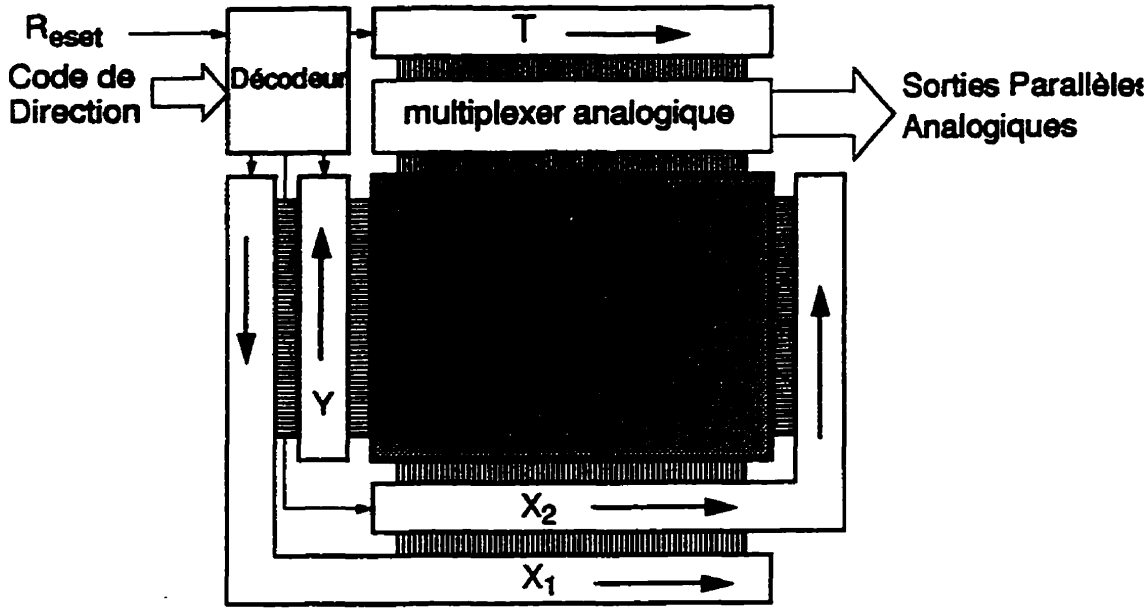


Figure 3.13 Diagramme bloc du capteur MAR. Trois registres à décalage sont utilisés pour effectuer la sélection multi-port (X_1, X_2 et Y) alors qu'un quatrième (T) sert à contrôler le multiplexer analogique de sortie. Les flèches indiquent, pour chaque registre, la direction positive de décalage.

le détail de la réalisation VLSI d'un module de registre à décalage bidirectionnel. On utilise un style de design dynamique avec une horloge à deux phases sans recouvrement afin de réaliser un module très compact. Chaque registre à décalage comprend une bascule dynamique et un multiplexer trois à un commandé par les deux signaux d'aiguillage: MOVE et DIR. Le signal MOVE indique s'il y a déplacement ou non et le signal DIR commande la direction du décalage dans le cas où le signal MOVE est actif. Les signaux de commande MOVE et DIR doivent être stables quelques nanosecondes avant la descente de l'horloge ϕ_1 et la sortie Q_i change après la montée de l'horloge ϕ_2.

La sortie Q_i de chaque registre à décalage commande, à travers un étage tampon, les quelques 300 transistors de sélection de la matrice de photo-détecteurs. La Figure 3.14 donne un exemple de fonctionnement d'un bloc de trois modules de registre à décalage sous forme de diagramme temporel. On peut voir que le "1" logique qui rend la sortie Q active pour le module i se déplace de façon synchrone vers le module $i + 1$, puis vers le module $i - 1$ et finalement au module i conformément aux signaux de commande de déplacement

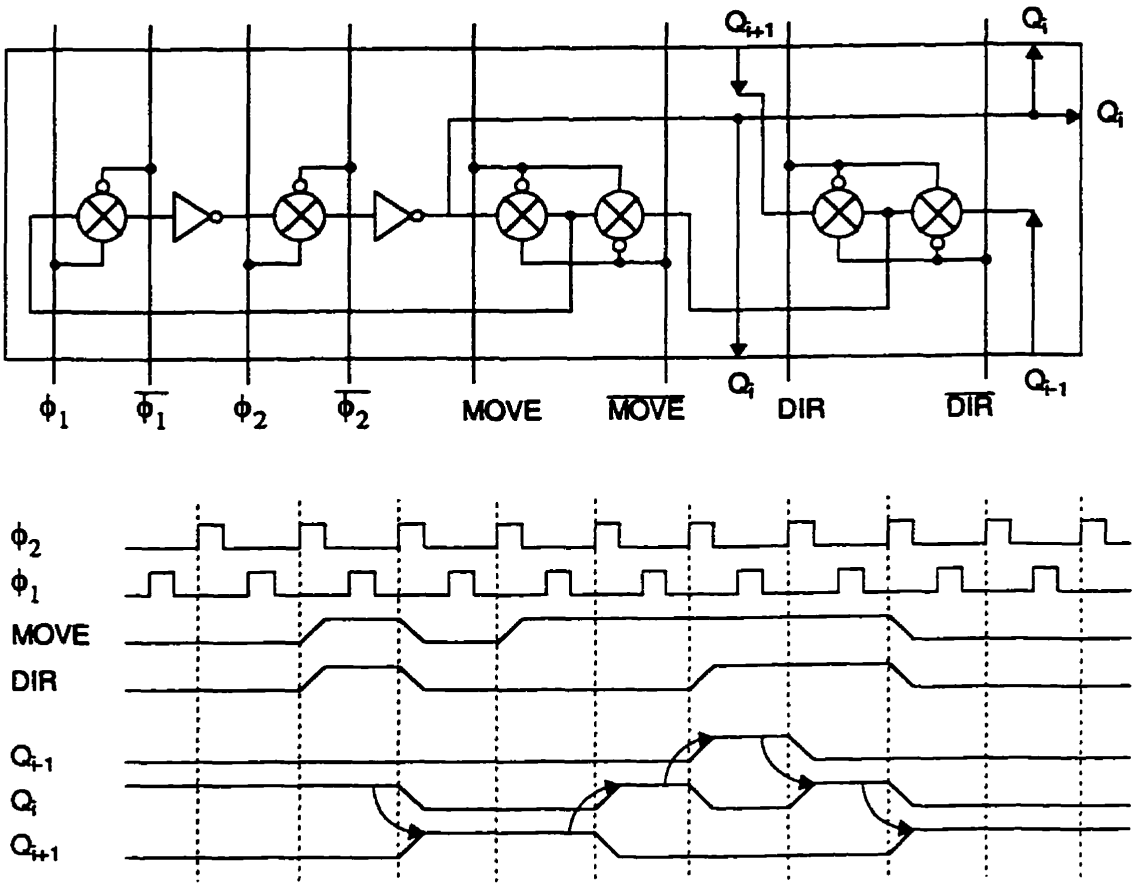
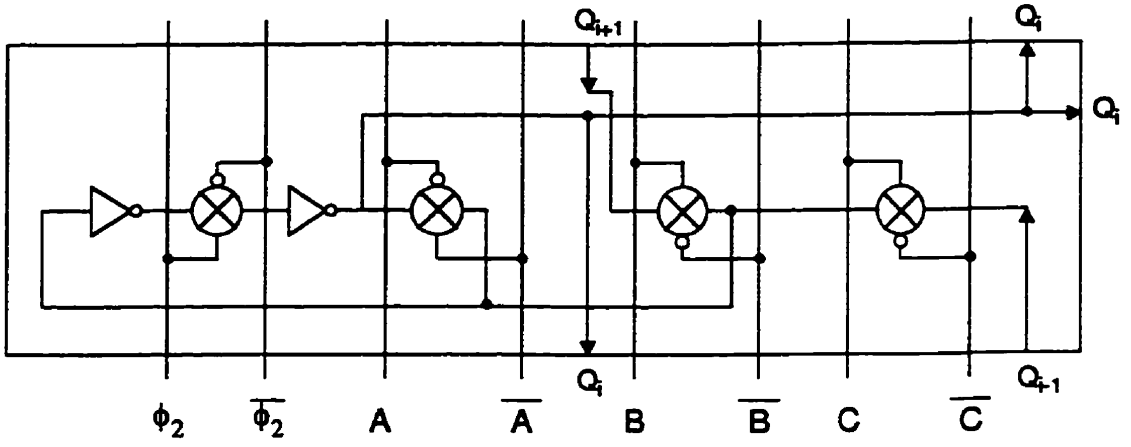


Figure 3.14 Architecture générale d'un module de registre à décalage utilisé en périphérie du capteur MAR. Un exemple d'utilisation montre une séquence de commandes qui déplace le contenu du registre en fonction des valeurs des signaux MOVE et DIR.

(*MOVE*) et de direction (*DIR*). On peut également remarquer que l'état des registres à décalage reste inchangé lorsque le signal *MOVE* est inactif.

L'architecture proposée à la Figure 3.14 utilise un multiplexer statique séparé du registre à décalage dynamique qui est commandé par les deux phases d'horloge. Il aurait été possible de regrouper, dans un même bloc de logique, le multiplexage et l'horloge ϕ_1 en utilisant un design dynamique avec horloges conditionnées [40] tel que montré à la Figure 3.15. Cette approche aurait permis de diminuer le nombre de transistors requis pour la réalisation d'un module de registre à décalage mais cela aurait compliqué le design du circuit de génération et de propagation des horloges en laissant le nombre de signaux de commande inchangé. La principale raison pour laquelle cette approche fût rejetée est qu'il



$$A = \phi_1 \cdot \overline{MOVE}$$

$$B = \phi_1 \cdot MOVE \cdot \overline{DIR}$$

$$C = \phi_1 \cdot MOVE \cdot DIR$$

Figure 3.15 *Module de registre à décalage utilisant le style de design avec horloges conditionnées. Le nombre de transistors requis est diminué mais le nombre de signaux de commande est le même qu'à la Figure 3.14.*

aurait fallu générer l'ensemble des horloges conditionnées dans la même région du circuit alors qu'on ne dispose que des quatre coins pour placer toute la logique de décodage et de génération des horloges. De plus, lors de la réalisation VLSI du capteur, la totalité de cet espace est requise et exploitée pour placer cette logique de contrôle. Pour plus de détails sur l'utilisation des quatre coins du circuit VLSI du capteur, se référer à la section 3.4.1.

3.2.2 Vérification de débordement et testabilité du capteur

La sortie Q_i d'un module de registre à décalage est propagée à l'étage tampon qui effectue la sélection et à ses deux voisins immédiats comme il est montré à la Figure 3.14. On utilise les sorties Q_i des deux modules de bout de chacun des quatre registres à décalage du capteur pour effectuer, à l'aide d'une fonction "ou" à huit entrées, la vérification d'une condition de débordement. Ce signal valide le balayage du capteur en signalant au système maître toute erreur de transmission ou d'exécution des signaux de commande de direction. Cette condition survient, par exemple, lors d'une commutation électrique violente près de la zone d'utilisation de la caméra.

Le signal de débordement a cependant un rôle majeur dans le test des circuits fabriqués. Pour effectuer le test de la partie numérique d'un capteur MAR, on initialise chaque registre à décalage de la caméra puis on utilise le signal de débordement pour vérifier la propagation bidirectionnelle d'un "1" ou d'un "0" dans chaque registre en comparant le signal observé avec la séquence prévue. La partie analogique doit être vérifiée par la suite avec un signal lumineux de deux intensités différentes. La réponse de chaque pixel devrait être relativement uniforme lors d'un balayage complet du capteur dans les deux cas. Cette séquence de test nous permet de détecter la défectuosité ponctuelle d'un pixel ou encore un problème plus global (colonne, ligne ou diagonale non-fonctionnelle).

3.2.3 Code de déplacement et définition de la parité en topologie hexagonale

On a défini le code de direction de trois bits selon la convention du cercle trigonométrique comme montré à la Figure 3.16 (a). On applique cette convention sur

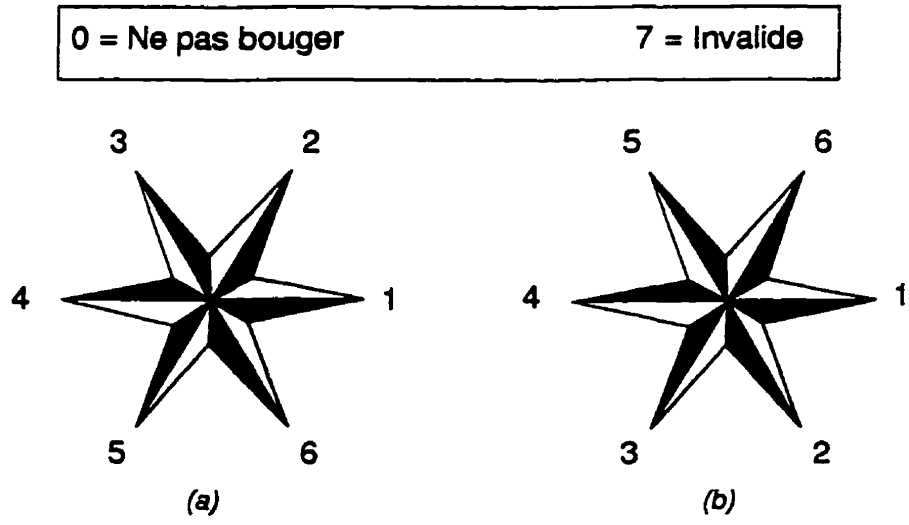


Figure 3.16 Convention du code de direction sur l'image (a) et pour le capteur (b). On utilisera, par convention, le code de direction sur l'image (a) pour toute description ultérieure.

l'image vue de face comme si un observateur se trouvait à la place de la caméra. Pour appliquer ce déplacement au référentiel du capteur, on doit tenir compte de l'inversion totale de l'image (haut-bas et gauche-droite) causée par la lentille convergente ainsi qu'une inversion gauche-droite due au fait qu'on dessine le circuit VLSI vu de dessus alors qu'on

interprète l'image comme si on regardait derrière le capteur. Cela a pour conséquence qu'on doit considérer la convention (b) de la Figure 3.16 pour le design VLSI du capteur. On utilisera cependant le code de direction sur l'image (Figure 3.16 (a)) comme convention officielle pour toute explication ultérieure puisque c'est le déplacement apparent au système d'acquisitions qui compte.

Lorsqu'on traite une image en topologie hexagonale, on doit stocker les données dans une matrice cartésienne et utiliser une convention qui permette d'accéder à chaque pixel en lui associant une position spatiale particulière. On utilise par convention un signal de parité qui est associé à chaque ligne et qui définit si elle est décalée ou non. La Figure 3.17 donne les détails d'un champ de 4 X 4 pixels en topologie hexagonale et de son équivalent pour une matrice de données cartésienne. On définit la convention suivante pour

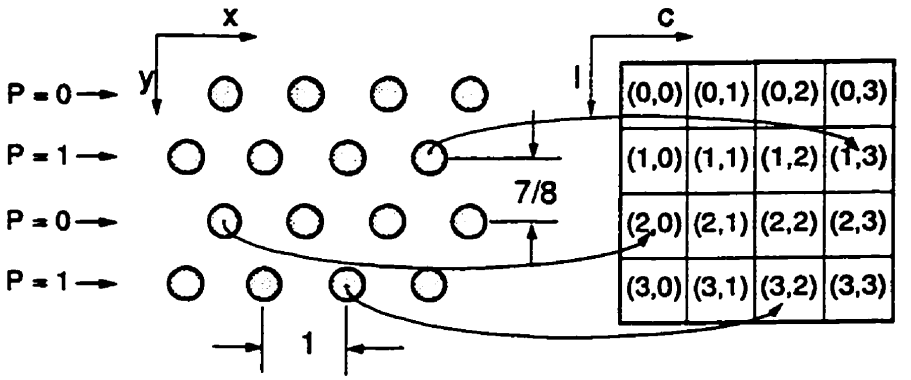


Figure 3.17 Définition de la parité d'une ligne en topologie hexagonale et utilisation d'une matrice 2D cartésienne pour le stockage des données hexagonales. Une ligne paire (P=1) est non-décalée alors qu'une ligne impaire est décalée vers la droite d'un demi pixel.

le signal de parité P: $P = 0$ pour les lignes paires décalées vers la droite de $1/2$ pixel et $P = 1$ pour les lignes impaires non décalées. De façon plus formelle, on définit P comme étant le reste d'une division entière de l'indice de ligne l par 2 et qui s'écrit:

$$P = modulo\,(l, 2) \tag{3-15}$$

La Figure 3.17 donne également une idée de l'espacement horizontal et vertical entre les pixels sur le capteur MAR. Une topologie hexagonale exacte implique une

distance unitaire entre chaque pixel soit une géométrie triangulaire équilatérale avec des angles de 60° entre chaque diagonale. Dans cette optique, la distance y entre deux lignes consécutives devrait être de $\cos(60°)$ ce qui correspond approximativement à 0.866. Puisqu'on doit définir, lors de la réalisation VLSI du capteur MAR, une géométrie selon une grille de résolution au 0.1 µm [12] et que le calcul en nombre entier représente un avantage marqué pour le traitement ultérieur des images provenant du système MAR, on place les lignes selon un rapport exact de $7/8$ (= 0.875). Dans cette condition, comme on peut le voir à la Figure 3.18, la distance entre deux pixels voisins en diagonale sur deux lignes consécutives est de 1.0078 pixels ce qui donne à peine 0.8% d'erreur lorsqu'on

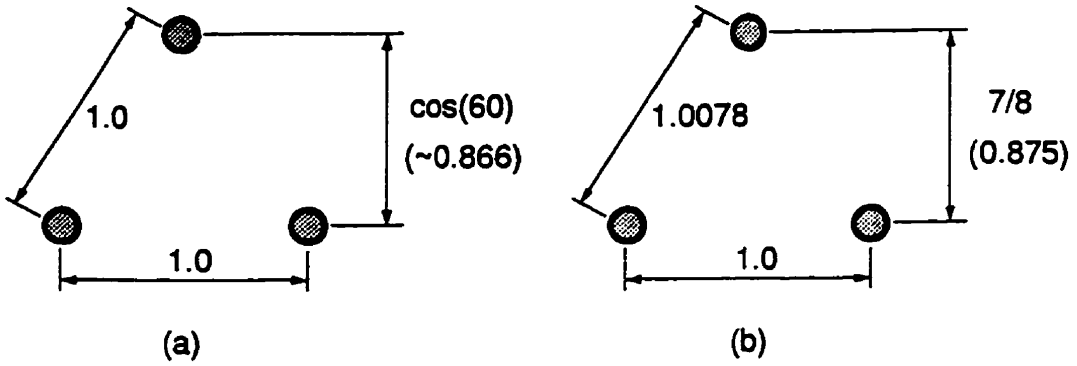


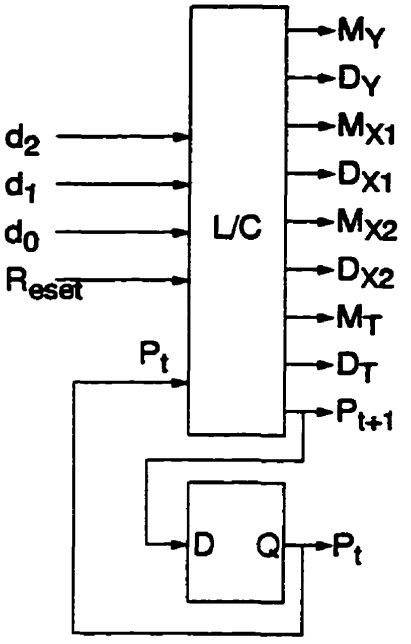
Figure 3.18 Comparaison entre une topologie hexagonale exacte (a) et une approximation par un rapport 7/8 (b) réalisée sur le capteur MAR.

considère tous les pixels voisins équidistants. Un avantage à l'utilisation du rapport de $7/8$ entre les lignes est la relation simple qui existe entre les indices (l, c) et la position (x, y) d'un pixel en topologie hexagonale. En définissant une distance de 8 entre les pixels d'une même ligne plutôt que 1, on peut écrire la position $(x, y) = f(l, c)$ selon les relations suivantes:

$$y = 7l \tag{3-16}$$

$$x = 8c + 4(\overline{modulo(l, 2)}) \tag{3-17}$$

Le contrôle des registres à décalage du capteur MAR est réalisé par un module de décodage intégré au circuit du capteur comme montré à la Figure 3.13. Ce module, décrit en détail à la Figure 3.19, transforme le code de direction de trois bits en quatre paires de signaux MOVE et DIR servant à commander les quatre différents registres à décalage. Les



Dir $d_2d_1d_0$	Image	Capteur	T P_t=0	T P_t=1	Y	X_1	X_2	P_{t+1}
0 0 0	•	•	0	0	0	0	0	P_t
0 0 1	→	→	+1	+1	0	+1	+1	P_t
0 1 0	↗	↘	+1	0	-1	+1	0	$\overline{P_t}$
0 1 1	↖	↙	0	-1	-1	0	-1	$\overline{P_t}$
1 0 0	←	←	-1	-1	0	-1	-1	P_t
1 0 1	↙	↖	0	-1	+1	-1	0	$\overline{P_t}$
1 1 0	↘	↗	-1	0	+1	0	+1	$\overline{P_t}$

+1 ⟶ $D_i = 1$ et $M_i = 1$
0 ⟶ $D_i = X$ et $M_i = 0$
+1 ⟶ $D_i = 0$ et $M_i = 1$

Figure 3.19 Décodeur de direction du capteur MAR. Le décodeur comprend une partie de logique combinatoire pour contrôler les registres à décalage et une partie séquentielle pour définir le signal de parité en topologie hexagonale. Le bloc de logique combinatoire est défini par les équations (3-18) à (3-26). La direction de commande +1 est dans la direction des flèches en gras sur la Figure 3.13.

équations (3-18) à (3-25) résument les solutions, réduites à l'aide de tables de Karnaugh, des signaux de commandes en fonction des signaux de direction d_2, d_1 et d_0, du signal de

remise à zéro R_{eset} ainsi que du signal interne de parité P_t comme il est d'ailleurs montré à la Figure 3.19.

$$M_Y = d_0 d_2 + d_1 \tag{3-18}$$

$$D_Y = d_2 \tag{3-19}$$

$$M_{X1} = d_2\overline{d_1} + \overline{d_1}d_0 + \overline{d_2}d_1\overline{d_0} \tag{3-20}$$

$$D_{X1} = \overline{d_2} \tag{3-21}$$

$$M_{X2} = \overline{d_1}d_0 + d_1\overline{d_0} = d_1 \oplus d_0 \tag{3-22}$$

$$D_{X2} = d_2 d_1 + (\overline{d_2})\,(\overline{d_1}) = \overline{d_2} \oplus d_1 \tag{3-23}$$

$$M_{Top} = d_0 P_t + \overline{d_2}d_0\overline{d_1} + \overline{d_1}d_2\overline{d_0} + \overline{P_t}d_1\overline{d_0} \tag{3-24}$$

$$D_{Top} = \overline{P_t}d_1 + (\overline{d_2})\,(\overline{d_1}) \tag{3-25}$$

$$P_{t+1} = \overline{(\overline{P_t} \oplus M_Y) + (R_{eset}d_2 d_1 \overline{d_0})} \tag{3-26}$$

Comme on peut le voir à la Figure 3.19, le signal P_t, qui représente la parité actuelle, provient de la sortie d'une bascule D qui effectue une mise à jour du signal parité en y assignant la valeur P_{t+1} à chaque coup d'horloge. Le signal P_{t+1}, comme le montre l'équation(3-26), inverse le signal parité lorsqu'on change de ligne ($M_Y = 1$) et force la valeur initiale de cette bascule à $P_t = 0$ lorsqu'un déplacement dans la direction 6 est effectué en même temps que le signal R_{eset} est activé. Cette condition spéciale est associée à la procédure d'initialisation du capteur et est discutée en détail à la section suivante. On remarque également à la Figure 3.19 que le code de direction 0 est utilisé pour signifier que le pixel d'intérêt reste sur place et que le code de direction 7 est invalide.

3.2.4 Initialisation du capteur

L'initialisation du capteur MAR consiste à remettre à zéro tous les modules de registres à décalage puis, de placer un "1" dans chaque registre au bon endroit de façon à obtenir un synchronisme spatial des trois lignes de sélection et du multiplexer de sortie. Le

pixel pointé à ce moment est appelé pixel d'intérêt. On utilise le signal de contrôle R_{eset} pour initialiser les registres à décalage et une courte séquence de déplacement du pixel d'intérêt afin d'obtenir l'initialisation voulue. La Figure 3.20 est un exemple générique d'un capteur de 4 X 4 pixels et montre le détail des signaux de commande et du signal R_{eset} qui permettent l'initialisation du capteur. On effectue l'initialisation dans le coin inférieur gauche du capteur (coin supérieur gauche de l'image) ce qui implique que le registre X_2 et T sont initialisés à l'extrême gauche, le registre Y, en bas, et le registre X_1 dans sa partie centrale au coin inférieur gauche. Le registre X_1 ne peut être initialisé que dans la direction 5 ou dans la direction 4. Pour des raisons de régularité de conception, un module de registre à décalage est ajouté aux registres X_2 et T (en pointillé sur la Figure 3.20) ce qui définit une colonne de pixels virtuels. C'est d'ailleurs dans cette région qu'est effectuée l'initialisation du capteur MAR.

L'état des différents registres à décalage de la Figure 3.20 est montré après qu'on ait exécuté la séquence d'initialisation du capteur MAR. Le registre T commande le multiplexer analogique de sortie. La Figure 3.20 ne montre qu'une partie restreinte des bus de données et du multiplexer lui-même. On peut voir que le module actif du registre T commande une diagonale de commutateurs qui font circuler le courant des pixel adressés dans les bons canaux de sorties analogiques. On distingue le courant commandé par le pixel d'intérêt qui est placé sur le noeud de sortie I_0 par le commutateur en trait gras. Le détail complet du multiplexer analogique de sortie incluant les signaux des diagonales X_1 et X_2 est présenté plus loin à la section 3.3.5.

La séquence d'initialisation qui nous permet d'obtenir la configuration finale de la Figure 3.20 est présentée à la Figure 3.21. Les quatre modules des registres à décalage qui sont actifs à la fin de la séquence d'initialisation sont ombragés à la Figure 3.20. Cette séquence de commandes de directions doit être étudiée en relation avec la Figure 3.20 en tenant bien compte s'il s'agit d'une commande de déplacement simple (MOVE) ou d'un déplacement avec le signal R_{eset} actif (SET). Les commentaires ajoutés identifient quels registres sont actifs après l'exécution de chaque commande d'initialisation et la lettre majuscule de A à F fait référence aux pixels marqués de la Figure 3.20. La première opération de l'initialisation du capteur MAR consiste à vidanger tous les registres à décalage. On réalise cette opération en effectuant au minimum 385 déplacements (pour un capteur de 256 X 256) simples dans deux directions consécutives (par exemple: 1 et 2).

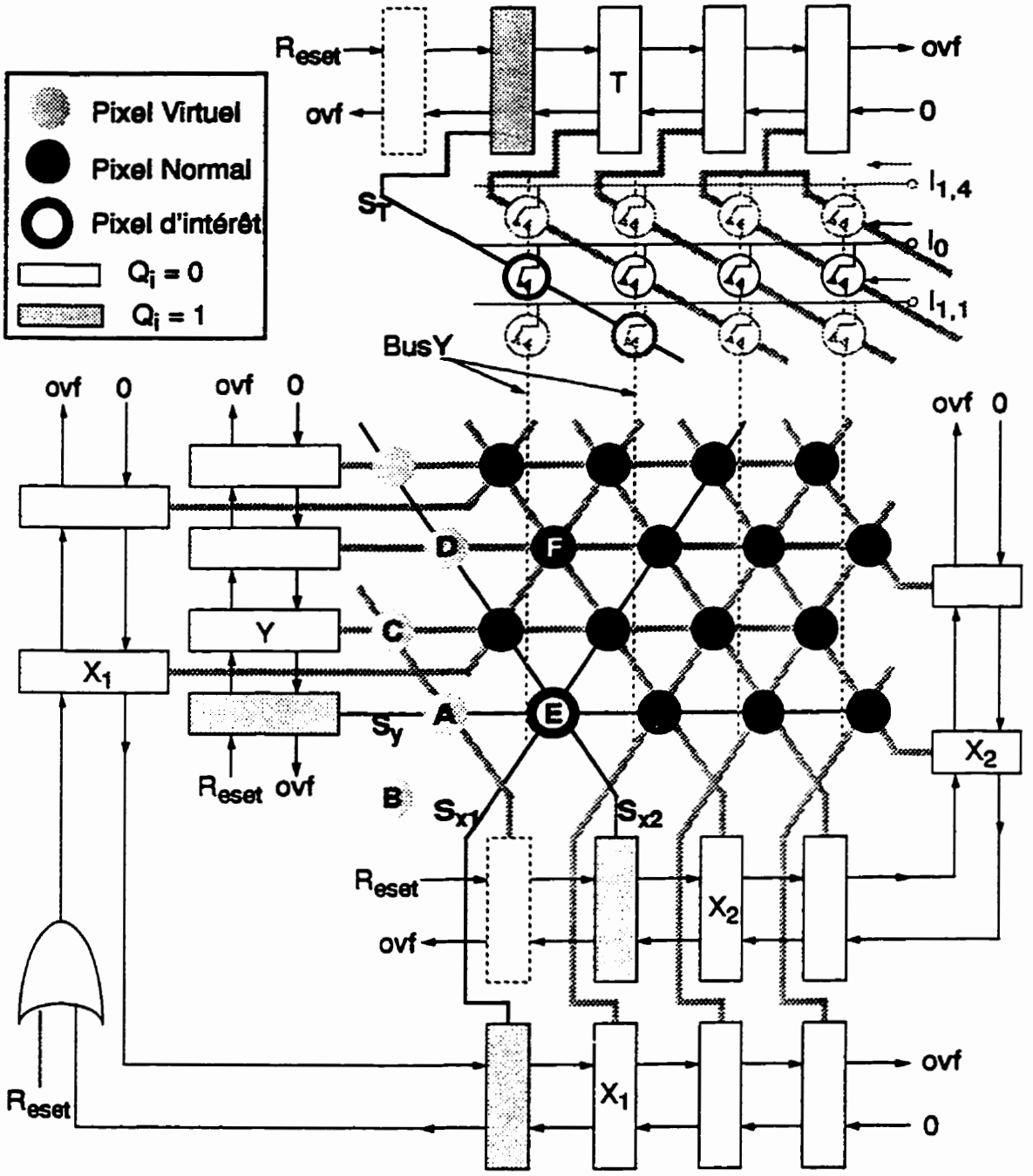


Figure 3.20 Exemple d'un capteur de 4 X 4 pixels. On montre ici l'état du contenu des quatre registres à décalage après la séquence d'initialisation. Les modules foncés indiquent une sélection active alors que les modules blancs indiquent une sélection inactive. Les pixels marqués d'une lettre de A à F font référence à la Figure 3.21

	#	Instruction	DIR	R_{eset}	P_t	Registres Initialisés	Figure 3.20
	1	SET	6	1	0	Y, X_2, (T)	-
	2	MOVE	3	0	1	(T)	-
	3	MOVE	3	0	0		-
	4	SET	5	1	1	Y, X_1	A
	5	MOVE	3	0	0	X_1	B
	6	SET	6	1	0	$(Y, X_1, X_2, T)_1$	A
OPTION	7	SET	5	1	1	$(Y, X_1, X_2, T)_1+(Y, X_1)_2$	C
OPTION	8	MOVE	2	0	0	$(Y, X_1, X_2, T)_1+(X_1)_2$	A
OPTION	9	MOVE	5	1	1	$(Y, X_1, X_2, T)_1+(X_1)_2$	C
OPTION	10	SET	6	1	0	$(Y, X_1, X_2, T)_1+(Y,X_1,X_2)_2$	A+D
	11	MOVE	1	0	0	Pixel(s) d'intérêt	E(+F)

Figure 3.21 *Séquence d'initialisation du capteur MAR. La séquence 1 à 6 définit une sélection concurrente au pixel A de la Figure 3.20. Les lignes 7 à 10 sont optionnelles et définissent des masques de convolution additionnels. La ligne 11 sert à déplacer le (ou les) pixel(s) d'intérêt(s) de la région virtuelle à la région normale.*

Les instructions 1 à 3 de la Figure 3.21 ne servent qu'à initialiser correctement la bascule de parité. Aux instructions 4 et 5, on initialise le registre X_1 puis, à la ligne 6, on complète l'initialisation des trois autres registres. Il est à noter que le registre X_1 doit déjà être initialisé à l'étape 6 puisque $M_{X1} = 0$ lorsque la direction de déplacement est 6. La valeur du signal P_t doit nécessairement être à 0 à l'étape 5 puisqu'on doit s'assurer que $M_t = 1$ à l'étape 6 pour initialiser correctement le registre T. Après l'étape 6, les quatre registres à décalage sont correctement initialisés et pointent le pixel A de la Figure 3.20.

On verra à la section 3.3.2 qu'il est possible de faire la sélection simultanée de plusieurs pixels d'intérêt pour des applications spécifiques. Cette technique pallie en partie au problème associé à un masque de convolution unique qui a la caractéristique d'effectuer un sous-échantillonnage de l'image et qui peut causer des problèmes lors de l'étude de surfaces spéculaires ou texturées. La contrainte principale pour effectuer correctement ce type d'initialisation multiple, provient du registre T qui doit absolument contenir un seul module actif dû à l'architecture du multiplexer analogique sortie. Cette procédure optionnelle représentée par les lignes 7 à 10 de la Figure 3.21 peut être effectuée une ou deux fois afin de définir respectivement deux ou trois pixels d'intérêts simultanément.

3.3 Adressage de la région d'intérêt

Cette section présente les avantages offerts par l'adressage multi-port du capteur MAR. On présente précisément le masque de convolution ainsi que le contexte d'application typique d'opérateurs à symétrie de révolution de même que d'autres opérateurs spatiaux plus généraux. On donne également les détails architecturaux du multiplexer analogique de sortie qui place chaque photo-courant sur le bon canal analogique. On termine cette section en présentant la méthode de compensation du courant de fuite des photo-diodes.

3.3.1 Masque de convolution en topologie hexagonale

L'adressage multi-port effectué sur le capteur MAR extrait, sous forme de signal analogique, l'information d'un grand nombre de pixels simultanément. On utilise la nomenclature spécifique $P_r(\delta)$ pour distinguer chaque pixel du masque de convolution où r est la distance le séparant du pixel d'intérêt et δ l'indice de l'angle représenté en nombre entier de 60° selon la même convention que le code de direction de la Figure 3.16 (a). La liste LP des pixels du masque de convolution s'écrit donc de façon formelle:

$$LP = \{P_0 \cup P_r(\delta) \mid (r \in 1, 2, \ldots, 15)\ et\ (\delta \in 1, 2, \ldots, 6)\} \qquad (3\text{-}27)$$

L'information analogique consiste en une série de courants associés à l'intensité lumineuse captée par le pixel d'intérêt P_0 et celle de ses 90 voisins $P_r(\delta)$ activés par le masque de sélection dont le rayon maximum r_{max} est fixé à 15 par la dimension physique du multiplexer analogique de sortie. Le masque de convolution en topologie hexagonale est présenté à la Figure 3.22. De façon interne, le multiplexer analogique de sortie place le photo-courant de chaque pixel sélectionné sur un bus distinct mais, pour des raisons de simplification et pour limiter le nombre de sorties analogiques, le signal de tous les pixels

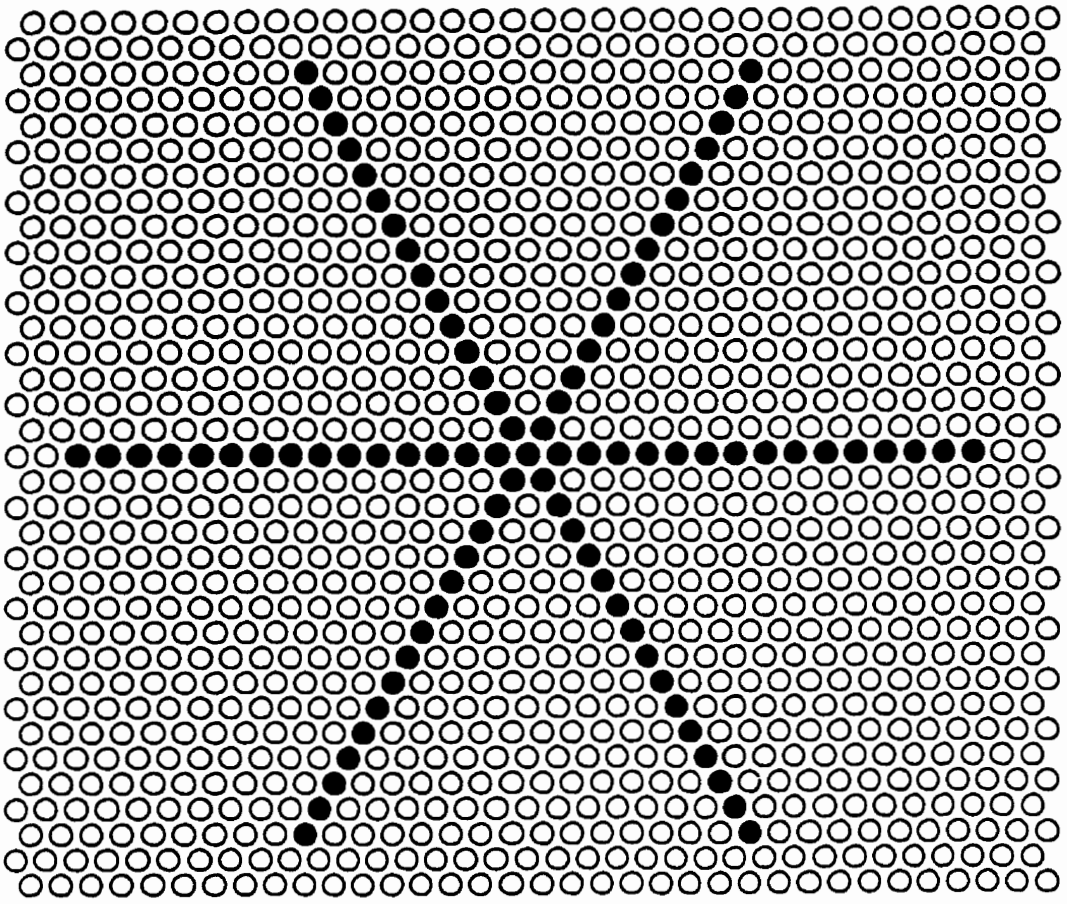


Figure 3.22 *Masque de convolution en topolologie hexagonale. L'information analogique des 91 pixels marqués en noir est extraite simultanément grâce à l'adressage multi-port.*

situés à une même distance du centre du masque de sélection et dont le rayon dépasse 2 sont regroupés en sextuplets en réservant ces canaux à des opérateurs à symétrie de révolution.

Il en résulte donc 26 canaux analogiques distincts: celui du pixel d'intérêt I_0, les 12 premiers voisins immédiats situés aux rayons 1 et 2 $\{I_1(1),...,I_1(6),I_2(1),...,I_2(6)\}$ et les 13 canaux restants, pour les rayons 3 à 15 $\{I_3, I_4,...,I_{14}, I_{15}\}$ qui sont réservés aux opérateurs à symétrie de révolution. La Figure 3.23 donne le détail du centre du masque de convolution et met en évidence la différence entre les pixels du centre ($r < 3$) et ceux réservés aux opérateurs à symétrie de révolution. On obtient pour une image dont l'intensité

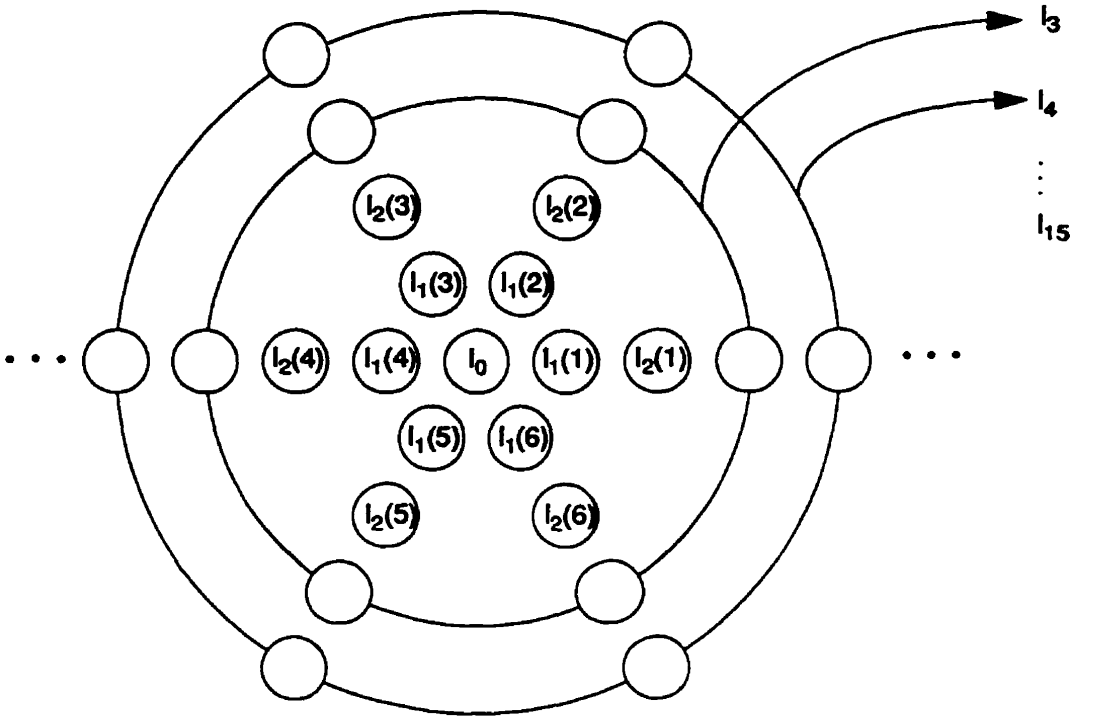


Figure 3.23 Représentation des signaux analogiques distincts présents à la sortie du capteur MAR. Les 13 pixels centraux sont accédés individuellement alors qu'on forme un canal unique, réservé aux opérateurs à symétrie de révolution, avec la somme des courants des 6 pixels équidistants pour les rayons 3 à 15

est uniforme en tous points, un courant d'amplitude 6 fois plus grande pour les canaux avec $r > 2$ puisque chaque canal correspond à la somme des courants générés par les six pixels de même rayon.

3.3.2 Masque de convolution à plusieurs pixels d'intérêt

Comme il a été mentionné à la section 3.2.4, il est possible d'effectuer une procédure d'initialisation multiple qui définit plusieurs masques de convolution superposés. Par exemple, en effectuant la sélection de trois pixels d'intérêt superposés, on obtient une extraction de tous les pixels marqués à la Figure 3.24. Dans ce cas, on effectue une moyenne en Y (orientation des bus de données analogiques) entre l'ensemble des paires de pixels adressés sur des lignes consécutives de même parité par la simple addition des courants sur un bus commun comme par exemple les paires de pixels A-D ou E-F de la Figure 3.20. On y remarque le problème de la sur-sélection de certains pixels dans la région

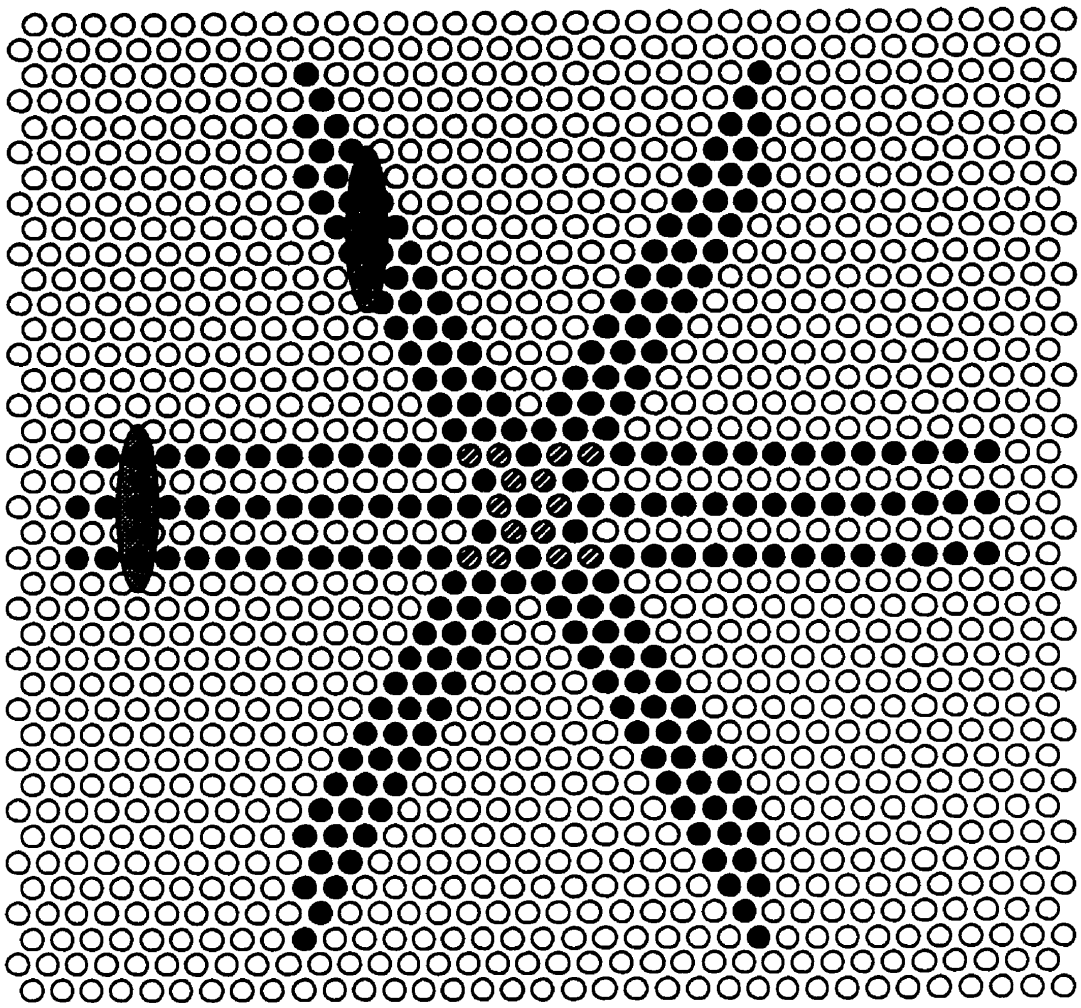


Figure 3.24 Masque de convolution à trois pixels d'intérêt. L'ensemble des signaux des 259 pixels marqués sont extraits de la matrice de photo-capteurs mais la somme est faite verticalement comme le montre les encadrés ovaloïdes. On peut remarquer que 14 pixels situés dans la région centrale (en hachuré) sont sélectionnés deux fois.

centrale du masque de convolution. Dans le cas de l'exemple à trois pixels d'intérêt montré à la Figure 3.24 on compte 259 pixels extraits simultanément dont 14 sont sélectionnés plus d'une fois. Ces quatorze pixels sont marqués d'un patron hachuré. Concrètement, on doit ajuster les coefficients associés à ces pixels afin d'assurer la pondération conforme au filtre désigné.

Il est important de mentionner que ce mode d'opération ne peut s'appliquer qu'à des opérateurs à symétrie de révolution et à filtrage grossier. Un opérateur grossier a la

caractéristique d'avoir les coefficients de même signe et de valeur similaire dans la région centrale (r<4). Par l'application d'un masque de convolution à plusieurs pixels d'intérêt, on échantillonne une plus grande partie de la région d'intérêt mais on applique un plus grand facteur de distorsion à l'image filtrée bien que les opérateurs grossiers ont la caractéristique de ne pas bien localiser les arêtes. On verra au chapitre 5 que l'étude comparative dans le domaine des fréquences de ces différents filtres révèle que le masque de convolution à plusieurs pixels d'intérêt a des avantages marqués pour l'atténuation des hautes fréquences.

3.3.3 Opérateurs à symétrie de révolution appliqués au capteur MAR.

L'objectif principal qui motive la réalisation d'un capteur donnant accès à un si grand nombre de pixels est l'application d'opérateurs grossiers faisant ressortir les contours les plus significatifs de la scène. Ces opérateurs demandent généralement un grand effort de calcul pour les processeurs numériques classiques. Il est de plus très intéressant d'obtenir le résultat de la convolution effectuée selon plusieurs résolutions spatiales simultanément ce qui donne l'opportunité d'envisager une architecture massivement parallèle pour le traitement de ces données. Tel que discuté à la section 1.4.1, le résultat de la convolution spatiale $R(x, y)$ d'une image $I(x, y)$ par un filtre $H(x, y)$ s'écrit dans le domaine continu:

$$R(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} H(\tau, \zeta)\, I((x-\tau), (y-\zeta))\, d\tau d\zeta \tag{3-28}$$

Pratiquement, on effectue la convolution de façon discrète avec une image échantillonnée selon une grille cartésienne de n lignes par m colonnes avec un masque de convolution limité radialement comme décrit à la section 1.4.1. Cette convolution discrète s'écrit de la façon suivante:

$$R[x, y] = \sum_{i=-X}^{X} \sum_{j=-Y}^{Y} H[i, j]\, I[x-i, y-j]\, \Delta S \tag{3-29}$$

avec ΔS représentant la surface d'un pixel de l'échantillonnage spatial (habituellement mesuré en pixel et égal à 1). Pour le cas particulier du capteur MAR où l'on effectue un sous échantillonnage spatial de l'image, on doit évaluer la "zone d'influence" de chaque pixel adressé par le masque de convolution de la Figure 3.22. La Figure 3.23 illustre la

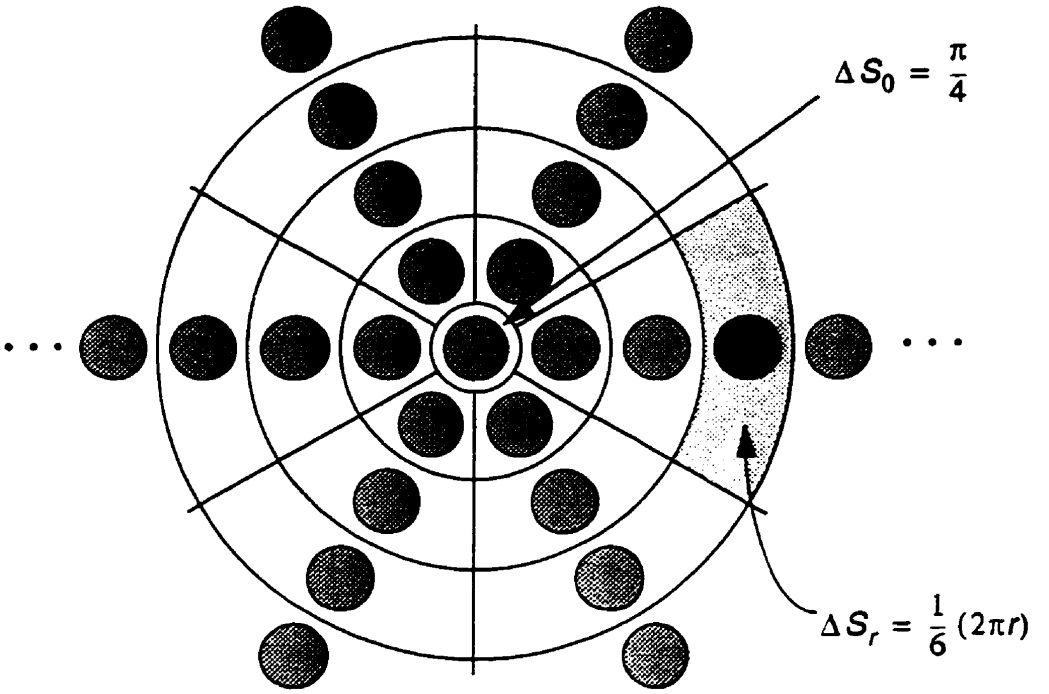


Figure 3.25 *Représentation de la zone d'influence des pixels du masque de convolution. On utilise un coefficient modifié afin de compenser le sous-échantillonnage pour le calcul du produit de convolution.*

région échantillonnée par le pixel marqué en noir. On définit l'aire de cette région comme étant le sixième de l'anneau tracé entre $r+\frac{1}{2}$ et $r-\frac{1}{2}$ soit:

$$\Delta S_r = \frac{1}{6}(\pi(r+\frac{1}{2})^2 - \pi(r-\frac{1}{2})^2) = \frac{1}{6}(2\pi r) \qquad (3\text{-}30)$$

On définit la surface du disque central de rayon $\frac{1}{2}$ comme étant l'aire associé au pixel central P_0 soit:

$$\Delta S_0 = \frac{\pi}{4} \tag{3-31}$$

A partir des définitions (3-30) et (3-31), on récrit l'équation du produit de convolution (3-29) spécifiquement au capteur MAR pour un opérateur $F[r]$ à symétrie de révolution:

$$R[x, y] = \frac{\pi}{3}\left[\frac{3}{4}F[0]I_0 + \sum_{r=1}^{15}\sum_{\delta=1}^{6} rF[r]I_r(\delta)\right] \tag{3-32}$$

Puisqu'on utilise des signaux dont la somme sur δ est pré-calculée et qu'on leur applique le même coefficient à rayon constant, on obtient:

$$R[x, y] = \frac{\pi}{3}\left[\frac{3}{4}F[0]I_0 + \sum_{r=1}^{15} rF[r]I_r\right] \tag{3-33}$$

ou

$$I_r = \sum_{\delta=1}^{6} I_r(\delta) \tag{3-34}$$

L'équation (3-33) montre qu'on doit ajuster la pondération par un facteur r pour tenir compte du sous-échantillonnage de l'image pour les pixels qui sont loin du pixel d'intérêt.

Deux observations permettent de minimiser les effets négatifs de ce sous-échantillonnage spatial. Premièrement, les coefficients en périphérie du masque de convolution ont généralement des valeurs très faibles alors que c'est cette région qui est la moins représentée. Deuxièmement, les opérateurs fins qui donnent une bonne localisation des arêtes ont pratiquement tous les pixels à leurs disposition et ne souffrent pas du sous-échantillonnage. Les filtres grossiers pour leur part, déforment naturellement les contours ce qui justifie un surplus de distorsion spatiale par rapport à un filtrage effectué avec un masque de convolution où tous les pixels de la région d'intérêt sont utilisés.

Seules les régions de l'image fortement texturées sont susceptibles de causer certains problèmes de fausse détection puisque l'échantillonnage spatial risque fort de ne pas être représentatif. C'est précisément dans ces cas qu'on préconise l'emploi d'un masque de convolution à plusieurs pixels d'intérêt comme discuté à la section 3.2.4. Il est

à prévoir également que la convolution sera erronée pour les contours en forme de pointes dans le cas de géométries fortement aiguës.

3.3.4 Opérateur spatial généralisé

Les treize pixels au centre du masque de convolution sont accessibles individuellement pour l'application de filtres directionnels. Un masque de Kirsch [5] ou encore le filtre de Canny [14] sont des opérateurs intéressants pour le calcul du gradient de l'image. Un exemple de ces filtres appliqué à la topologie hexagonale est montré à la Figure

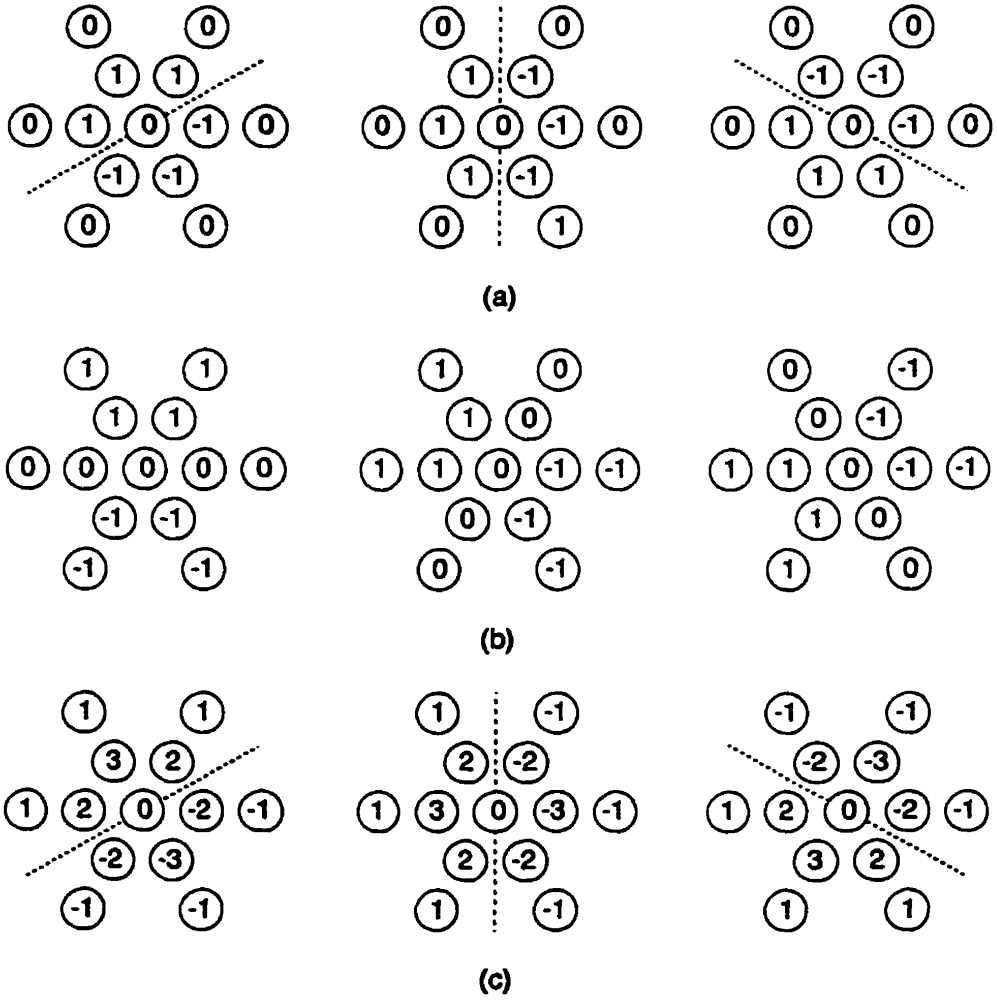


Figure 3.26 Exemple de filtres directionnels connus appliqués à la topologie hexagonale. L'opérateur gradient (a), le masque de Kirsch (b) et le filtre de Canny (c).

3.26. Il s'agit ici d'une solution possible et comme le calcul est analogique, la pondération des masques n'a pas à être restreinte aux valeurs entières. Ces opérateurs ont l'avantage de réagir à l'orientation des portions d'arêtes. De plus, ils indiquent l'importance relative de ces arêtes selon l'amplitude du résultat de la convolution. On peut évidemment calculer simultanément les filtres pour chaque orientation et disposer du résultat en parallèle ce qui nous permet de définir rapidement la tendance de l'orientation d'arête.

Dans le domaine du traitement d'images sur des machines séquentielles, on reconnaît habituellement l'avantage des filtres directionnels par la simplicité de leurs masques de convolution car le temps de traitement est proportionnel au nombre de coefficients distincts lors de la convolution. Dans le cas du système MAR, le temps de calcul de la convolution est constant peu importe les dimensions du filtre et, puisqu'il est plus facile de détecter un passage par zéro qu'un maximum, il est avantageux d'utiliser le laplacien de gaussienne plutôt que des opérateurs directionnels. Comme on le verra à la section 4.3.11, le contrôleur réalise, grâce à un jeu d'instructions microcodé, un algorithme de suivi qui extrait les portions rectilignes d'arêtes tout en fournissant des attributs relatifs à l'orientation et à la discontinuité de celles-ci.

3.3.5 Multiplexer analogique des sorties

Sans être la partie la plus complexe du capteur MAR, le multiplexer analogique de sortie est certes celle qui demande la plus grande attention puisqu'il place les bons signaux, préalablement sélectionnés grâce à l'adressage multi-port, sur les bons canaux de sortie. Comme on l'a déjà mentionné, le nombre de multiplexers placés côte à côte définit le rayon maximal du masque de convolution. La Figure 3.27 donne le détail de cette partie du capteur pour un exemple simpliste d'un masque de convolution de rayon maximal égal à 1 pixel. La fonction de multiplexage est remplie par l'action d'un simple transistor NMOS en mode de commutateur commandé. On remarque que le multiplexer est séparé en deux parties distinctes: un multiplexer simple (I à K) pour l'ensemble des bus Y (sélection horizontale) et un multiplexer double (C et E) pour l'ensemble des bus des deux diagonales X_1 et X_2.

La Figure 3.27 met en évidence le pixel d'intérêt P_0 (A-1) et quatre de ses six voisins immédiats, $P_1(1)$, $P_1(2)$, $P_1(3)$ et $P_1(4)$, qui sont convenablement sélectionnés comme montré à la Figure 3.20. Les signaux de sélection actifs sont marqués d'un encadré. Les pixels $P_1(5)$ et $P_0(6)$ ne sont pas montrés pour alléger le dessin mais sont également activés par les signaux S_{X1} et S_{X2} respectivement. Le registre à décalage

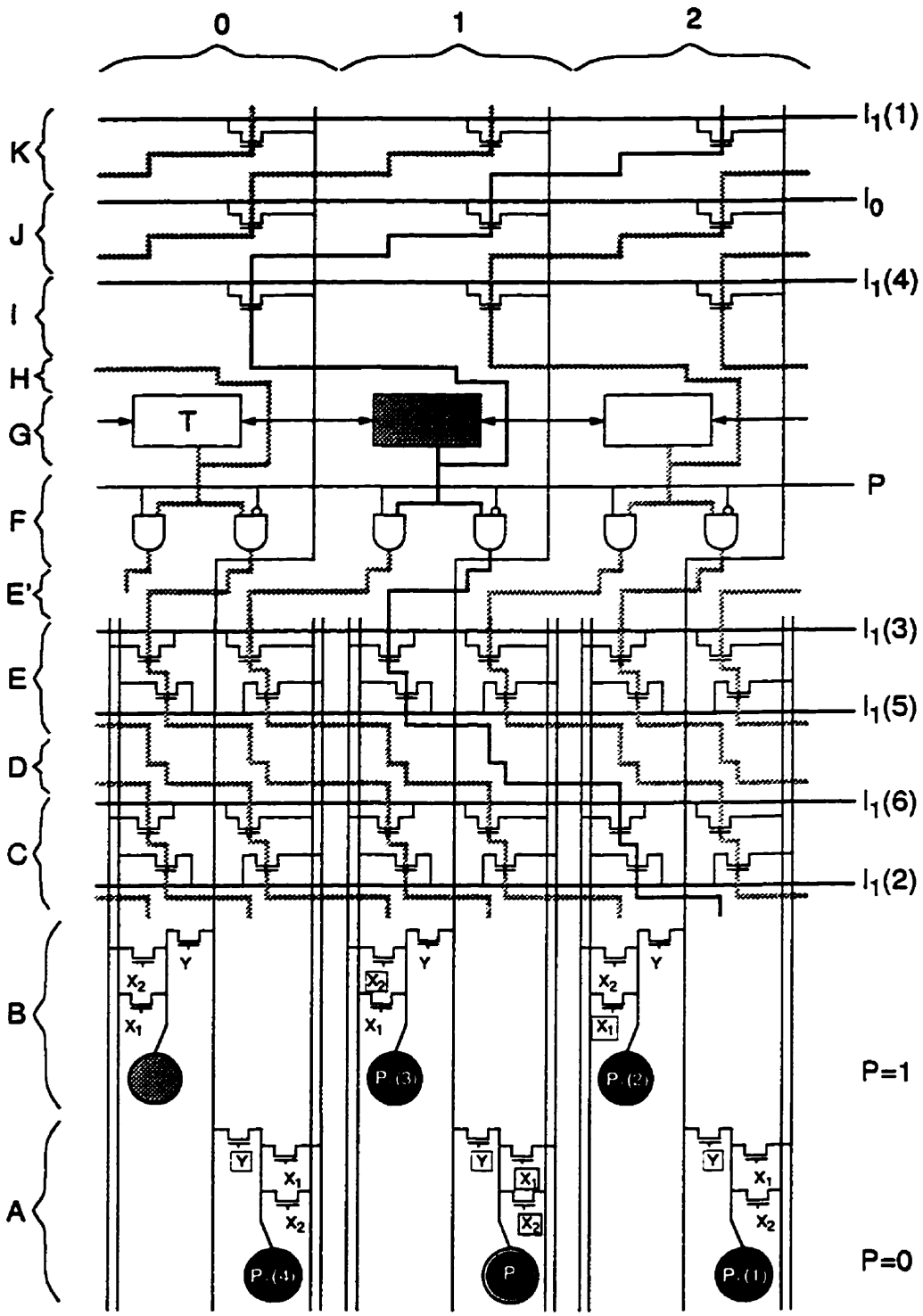


Figure 3.27 Architecture interne du multiplexer analogique de sortie. On montre un exemple générique d'un capteur ayant un masque de convolution de rayon maximal égal à 1 pixel. La partie A à B correspond à une portion de la région de pixels alors que la partie C à K représente le multiplexer.

T, qui est correctement sélectionné en synchronisme avec le pixel d'intérêt (G-1), commande les modules de multiplexers simples I-0, J-1 et K,2 afin d'extraire les signaux des pixels A-0, A-1 et A-2 respectivement. Le même signal de sortie actif du registre à décalage T (G-1) est utilisé pour contrôler la bonne diagonale (en gras) des modules E-1 et C-2 pour l'extraction des signaux des pixels 2,3,5 et 6. Un petit démultiplexer (F-1) réagit au fait que le pixel d'intérêt est situé sur une ligne paire (B) ou sur une ligne impaire (A) et sa présence est une conséquence du déplacement conditionnel du registre T en fonction du signal parité lors des déplacement en diagonale comme montré à la Figure 3.19. La série de modules fictifs (C-0, C-1 et C-2) propagent correctement la diagonale de sélection des multiplexers doubles sans effectuer l'extraction des signaux. Cet artifice permet d'éviter la sélection multiple qu'on retrouve au pixel d'intérêt. Même si les trois signaux de sélection sont actifs au pixel P_0, les bus analogiques X_1 et X_2 sont laissés flottants et la totalité du courant circule dans le bus Y et est extraite par le multiplexer simple J-1.

On peut suivre sur la Figure 3.27 le chemin parcouru par le courant de chaque pixel vers le canal de sortie analogique qui est propre à la sélection courante. Dans le cas d'un masque de convolution complet comme celui de la Figure 3.22, on répète les lignes C, E, I et K quinze fois plutôt qu'une et on obtient ainsi 91 sorties analogiques qui sont toutes disponibles aux extrémités gauche ou droite du capteur. Pour réaliser la somme des signaux des six pixels de même rayon des canaux réservés à des opérateurs à symétrie de révolution, on n'a qu'à connecter ensemble les six signaux en courant $I_r(1)$ à $I_r(6)$ disponibles à l'une ou l'autre des extrémités du multiplexer analogique de sortie.

Pour assurer un synchronisme spatial entre le masque de convolution et le multiplexer de sortie, on effectue un décalage de la sortie du registre T vers la bonne diagonale des deux parties du multiplexer parallèle. Les modules H et E' de la Figure 3.27 servent à décaler le signal de sélection des multiplexers de façon à s'assurer que le signal du pixel P_0 se retrouve sur le canal I_0. Lorsqu'on réalise le multiplexer complet avec $r_{max} = 15$, on doit prévoir un décalage de 15 (r_{max}) colonnes vers la gauche à la ligne H et de 7.5 ($r_{max}/2$) colonnes vers la gauche à la ligne E'. Ce décalage est mis en évidence pour l'exemple simpliste de la Figure 3.27 où l'on peut distinguer clairement, dans le cas d'un filtre ayant un rayon maximal (r_{max}) de 1 pixel, un décalage de 1 pixel à la ligne H et de 1/2 pixel à la ligne E'. Il est à noter que le décalage de $r_{max}/2$ de la ligne E' aurait pu être effectué entre la ligne F et G sur la sortie Q_i du registre à décalage T. Dans ce cas, on aurait eu un seul signal à décaler plutôt que deux. Un module de compensation du courant

de sortie est placé au bout du multiplexer analogique de sorties. On présente à l'ANNEXE A les détails de ce module qui effectue la compensation du courant de fuite des photo-détecteurs de même que la correction de la composante continue des photo-courants.

3.3.6 Remise à zéro du processus d'intégration d'illuminance

Lors de la remise à zéro du processus d'intégration d'illuminance au début de la prise d'images, la tension V_{Reset} de la Figure 3.8 (a) doit être appliquée à l'ensemble des pixels du capteur MAR. Puisque les bus de données ne sont pas utilisés pendant cette procédure, les bus Y sont monopolisés pour propager la tension V_{Reset} à l'ensemble des pixels du capteur MAR. Comme on l'a mentionné à la section 3.1.3, puisqu'on utilise des transistors N pour propager des signaux de valeurs de tensions positives ($V_{Reset} \geq 5.0V$), il en résulte une perte de tension de seuil combinée à un effet de substrat de l'ordre de 2 V. On retrouve donc, après le processus d'initialisation du capteur, un tension sur la grille du transistor $M_1(V_g)$ égale à $V_{Reset} - V_{SN}$. La Figure 3.28 montre le détail complet de l'architecture d'un pixel correspondant à l'assemblage de la Figure 3.7 avec la Figure 3.8 (a). On remarque l'utilisation conjointe du bus analogique Y pour la lecture des pixels sélectionnés par S_Y de même que pour l'initialisation de la grille du transistor M_1 à travers le transistor M_2.

Puisque le bus Y est utilisé pour initialiser les pixels, on doit prévoir un module qui prend possession de ce bus lorsque le signal R_{eset} est actif. On place une rangée de petits modules comprenant un transistor N comme montré à la Figure 3.29. Lorsque le signal R_{eset} est actif, le signal de tension V_{Reset} est placé sur le bus Y en même temps que le transistor M_2 de la Figure 3.28 est activé, ce qui a pour conséquence de précharger la jonction PN de la photo-diode.

3.4 Réalisation VLSI

Cette section décrit les outils qui ont permis la réalisation du dessin des masques du circuit intégré du capteur MAR. Plusieurs détails d'implantation physique de ce circuit VLSI sont également abordés. Le capteur MAR comprend plus de 400,000 transistors pour réaliser, sur un même dé de silicium, un total de 68,000 pixels. On présente le concept de compilateur de structure qui est un outil qui génère, à partir de quelques cellules de base, le dessin des masques d'un capteur de dimension quelconque. On discute des limitations physiques du capteur MAR comme la fréquence maximale de balayage ainsi que les détails

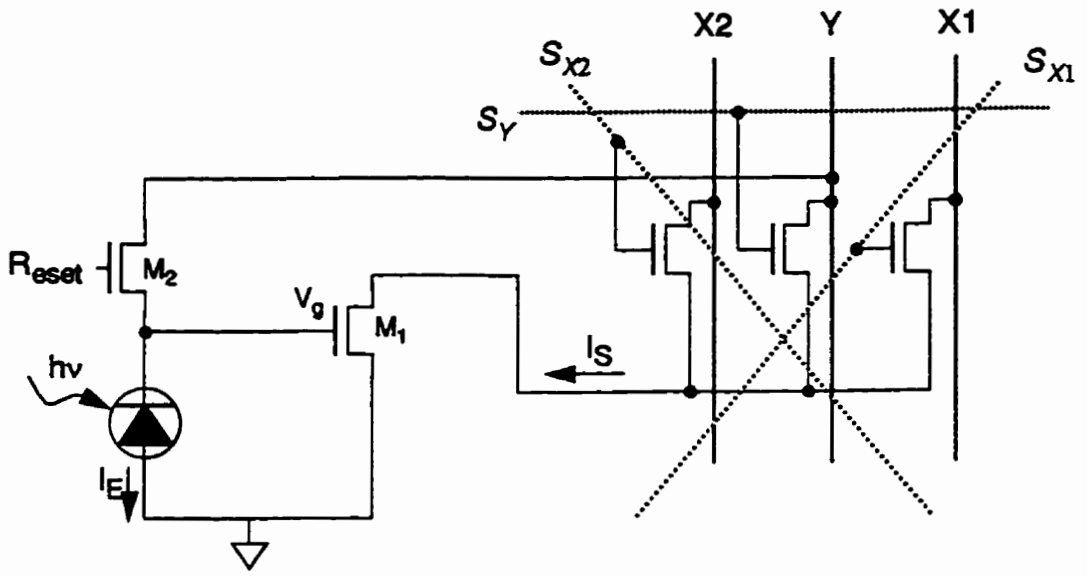


Figure 3.28 Pixel MAR complet tel que fabriqué dans la version courante du capteur 256 X 256. La tension V_{Reset} est propagée par le bus Y lors de la remise à zéro du capteur.

géométriques pour la réalisation de diagonales à 60° à partir de sections de conducteurs essentiellement orthogonaux. On traite également de la génération et de la distribution des signaux de contrôles internes et des horloges. On termine cette section par un court historique de la réalisation VLSI du capteur MAR de même que par la description des plots de contact.

3.4.1 Compilateur de structures

Le premier prototype du capteur MAR a été développé sur le logiciel ELECTRIC qui permet de réaliser des arrangement répétitif de cellules. L'assemblage des différents modules demeure cependant une tâche manuelle interminable où la moindre modification sur les cellules de base implique un réarrangement presque total du circuit. Le logiciel CADENCE (initialement nommé SDA) qui a été utilisé pour réaliser les versions subséquentes du capteur MAR met au service du concepteur un module de compilation de

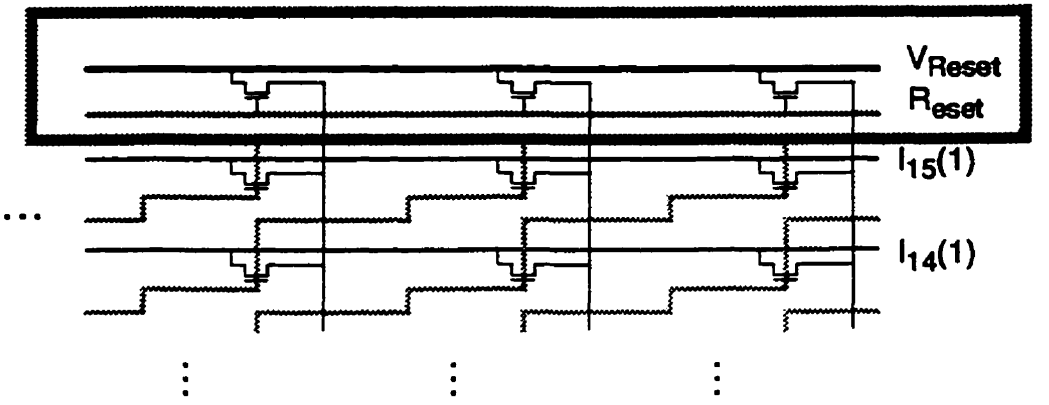


Figure 3.29 Module servant à prendre possession du bus Y pour effectuer le remise à zéro des pixels.

structures qui réalise l'assemblage d'un circuit répétitif (tel de la mémoire, un PLA ou un capteur 2D) à partir de certaines spécifications relationnelles.

La Figure 3.30 donne l'allure du plan de description réalisé afin d'alimenter le compilateur de structure pour la génération d'un capteur de 268 lignes par 256 colonnes (les termes anglais réfèrent au plan de description utilisé durant le projet). Un tel plan de description ("template" dans CADENCE) définit seulement les relations d'alignement des différents modules qui composent le circuit final alors que chaque élément distinct de la Figure 3.30 comprend une liste de propriétés qui définit le nombre, le nom et l'orientation des cellules de base qui doivent être assemblées par le compilateur. La liste décrivant sommairement le contenu descriptif de chaque module ainsi que la référence au texte du chapitre 3 est présentée au Tableau 3.1 On retrouve à l'ANNEXE E (Figure E.2) la représentation template du capteur MAR tel qu'éditée par le logiciel CADENCE.

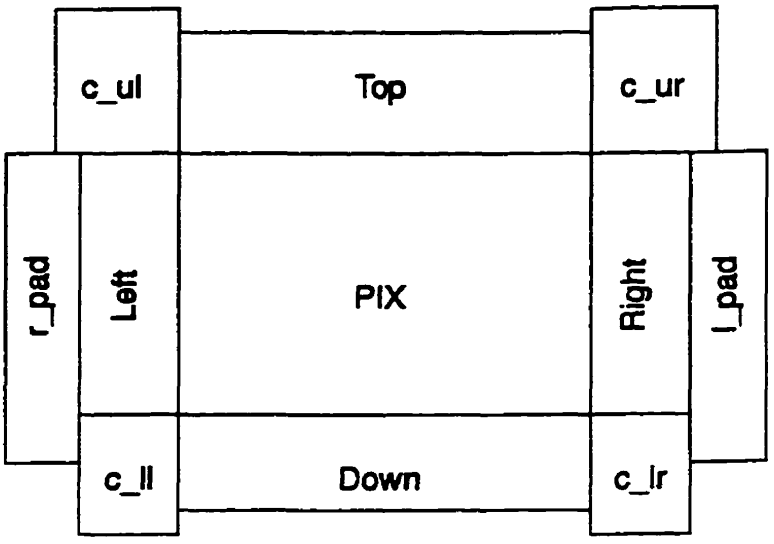


Figure 3.30 *Plan de description utilisé par le compilateur de structure pour la génération automatique du circuit intégré du capteur MAR.*

Tableau 3.1 *Description sommaire des cellules de base du capteur MAR*

Macro	Cellule de base	lignes	colonnes	Commentaires	Réf.
PIX:	2_pixels_MAR	#lig=134	#col=256	matrice de photo-capteurs	3.4.3
Top	registre_top	#lig=1	#col=256	multiplexer de sorties	3.3.5
Left	registre_left	#lig=128	#col=1	2 reg. Y et 1 reg. X_1	3.2.4
Right	registre_right	#lig=128	#col=1	1 registre X_2 (par 2 lignes)	3.2.4
Down	redistre_down	#lig=1	#col=256	1 reg. X_1 et 1 reg. X_2	3.2.4
c_ul	c_ul	#lig=1	#col=1	génération d'horloges	3.4.4
c_ll	c_ll	#lig=1	#col=1	décodage du la direction	3.2.3
c_ur	c_ur	#lig=1	#col=1	extraction des signaux analogiques	3.3.5
c_lr	c_lr	#lig=1	#col=1	calcul du débordement	3.2.2
r_pad	r_pad	#lig=1	#col=1	plots de contacts analogiques	3.4.6
l_pad	l_pad	#lig=1	#col=1	plots analogiques et numériques.	3.4.6

La représentation "layout" de chaque cellule de base du Tableau 3.1, qui est le dessin des masques de la cellule en question, comprend une couche d'alignement spécifique au logiciel CADENCE qui est utilisée pour abouter l'ensemble des différentes cellules du circuit complet. Le design détaillé n'est pas présenté ici car le nombre de cellules différentes est trop imposant. On décrit cependant aux paragraphes suivants les lignes directrices qui ont guidé la réalisation du dessin des masques des cellules de base de même que la fonction respective de leur contenu logique et analogique. On fait de plus référence à l'ANNEXE E pour une vue d'ensemble des différentes parties du dessin des masques du capteur.

Premièrement, le pixel doit être extrêmement compact puisque c'est la taille de ce dernier qui définit la résolution du capteur (la dimension maximale d'un dé de silicium limite la résolution). Le rapport hauteur sur largeur d'un pixel doit être exactement 7/8. On doit s'assurer que la cellule du multiplexer analogique double comme à la Figure 3.27 (cellules E-0 à E-2 et C-0 à C-2) est réalisable à l'intérieur de la dimension horizontale d'un pixel car, la densité des signaux verticaux et des contacts est très élevée. On doit viser la plus haute densité possible lors de la réalisation du dessin des masques de cette cellule de multiplexage car elle se répète 32 fois (verticalement) dans le module de base "Top". La section 3.4.3 est réservée à la description détaillée du contenu du pixel MAR.

Le design des autres sous modules qui composent les modules Left, Right, Top et Down est plus simple. Chaque registre à décalage qui y est inclus est identique à celui de la Figure 3.14. La contrainte sur la surface de silicium requise pour la réalisation de ces cellules est moins grande car elles se répètent une seule fois par ligne ou par colonne du capteur. Le design des quatre modules de coin représente une partie très importante de la définition des cellules de base du capteur MAR. Cet espace est le seul disponible pour intégrer toute la logique périphérique de décodage, de génération d'horloges, d'amplification et d'extraction des signaux analogiques.

Le coin inférieur gauche (c_ll) comprend le circuit de décodeur de direction comme celui de la Figure 3.19 ainsi que la bascule de parité. Cette partie du capteur (Figure E.3) comprend les modules d'amplification des signaux de contrôle pour les registres X_1 et Y ainsi que pour une partie des signaux de contrôle du registre X_2. Les signaux de débordement provenant des registres Y, X_1 et T sont dirigés vers le coin inférieur droit. Un énorme tampon régénère le signal R_{eset} qui est propagé à l'ensemble des pixels photo-sensibles.

Le module du coin supérieur gauche (c_ul) comprend principalement le module de génération des deux phases de l'horloge ainsi que les tampons qui leurs sont associés (Figure E.4). Les détails de ce circuit sont présentés à la section 3.4.4. On y réalise l'amplification des signaux de contrôle du registre T en plus d'effectuer l'extraction des 14 signaux analogiques du centre du masque de convolution (I_0, $I_1(1)$, ..., $I_1(6)$, $I_2(1)$, ..., $I_2(6)$ et I_3) qui sont envoyés vers les plots de contact du côté gauche du capteur. Les signaux analogiques sont entourés d'une gaine métallique reliée à la masse du circuit afin de limiter au maximum les interactions parasites causées par les commutations numériques.

Le module du coin supérieur droit (c_ur) effectue l'extraction des 12 signaux analogiques restants (I_4 à I_{15}) qui sont réservés pour des opérateurs à symétrie de révolution et les dirige vers les plots de contacts du côté droit (Figure E.5). Ce module comprend également les 91 transistors de correction de courant de fuite avec ses deux signaux de contrôle (V_{Goff} et V_{Doff}) de même que le signal global V_{Reset}. C'est dans cette partie du circuit qu'est placé le plot de contact pour la référence de tension numérique V_{SS}.

Le coin inférieur droit (c_lr) contient le module d'évaluation du signal de débordement (ovf) de même que l'amplificateur de sortie pour cet unique signal de sortie numérique (Figure E.6). On y retrouve l'autre partie des amplificateurs des signaux de contrôle du registre X_2. On place également dans cette partie du circuit le plot de contact de la tension d'alimentation numérique V_{DD} (5 V) ainsi que le signal de masse analogique, connecté sur un plot de contact indépendant. La masse analogique est propagée à l'ensemble de la matrice de pixels par le module du coin inférieur droit afin de réduire au maximum l'interaction du bruit de commutation numérique. Finalement, le signal R_{eset} est propagé dans toutes les parties utiles du circuit (c_ll et c_ul) afin d'assurer l'initialisation des quatre registres à décalage et la remise à zéro du processus d'intégration d'illuminance.

3.4.2 Fréquence maximum de balayage

Le mode de balayage du capteur MAR est essentiellement séquentiel. Dû à la complexité du processus d'initialisation du capteur, le déplacement du pixel d'intérêt est limité à un seul pixel à la fois bien qu'on puisse orienter ce déplacement dans n'importe quelle des six directions définies par la topologie hexagonale. La fréquence de balayage du capteur MAR est physiquement limitée par le module de registre à décalage de la Figure 3.14. Ce circuit étant très simple, le principal délai entre les signaux de commandes et la réponse du registre à décalage est causé par le décodeur de direction et les étages tampons.

En pratique, pour un capteur réalisé avec technologie CMOS 1.2 µm, la fréquence maximum de fonctionnement est de 45 MHz. Puisque les commandes de direction proviennent du contrôleur MAR et que ce dernier ne peut fonctionner avec une horloge de base dépassant 20 MHz, la vitesse de déplacement du pixel d'intérêt est donc limitée à cette fréquence.

Même si les signaux de sélection du capteur ont la possibilité de changer d'état à la cadence de 20 M cycles, la réponse analogique transitoire du signal de sortie des pixels dure quelques centaines de nanosecondes. La réponse analogique est limitée par la bande passante et la variation maximum de la tension de sortie ("slew rate") des amplificateurs analogiques externes responsables de la conversion courant-tension (Figure A.2) de même que du filtrage analogique. En pratique, lors du déplacement d'un pixel, si on utilise des amplificateurs externes ayant une bande passante unitaire de 6 MHz et une pente maximale de 40 V/µs, on obtient un temps de stabilisation complet (95%) des signaux analogiques de l'ordre de 500 ns ce qui implique une fréquence maximale d'environ 2 MHz. Pour les besoins de certaines applications telles que la détection de passages par zéro, une valeur approximative du résultat à environ 80% de sa valeur finale est satisfaisante. Puisque la réponse analogique à un changement de position du pixel d'intérêt a essentiellement la forme d'une exponentielle décroissante le balayage peut se faire à la fréquence de 5 MHz. La Figure 3.31 résume graphiquement ces remarques sur la fréquence maximum de balayage du capteur MAR.

En conclusion, on peut donc effectuer un déplacement rapide du pixel d'intérêt d'un endroit à un autre à la fréquence maximum définie par le contrôleur numérique jusqu'à concurrence de 45 MHz. Dans ce cas, on ne fait que changer le pixel d'intérêt d'une région à une autre et on ne s'intéresse nullement aux valeurs des sorties analogiques (Figure 3.31 (a)). L'extraction des données visuelles analogiques est limitée par la bande passante des amplificateurs externes et requiert des temps d'attente entre chaque commande de déplacement pour que les sorties du circuit analogique se stabilisent. La fréquence de balayage se situe alors entre 2 et 5 MHz (Figure 3.31 (c) et (b) respectivement), dépendant du niveau de stabilisation désiré pour les sorties analogiques.

3.4.3 Détails géométriques d'un pixel

Les caractéristiques électroniques du pixel MAR sont décrites dans les sections précédentes. On présente maintenant les principaux concepts géométriques du pixel MAR. Le schéma de la Figure 3.32 donne le détail de la réalisation VLSI d'un pixel avec une représentation symbolique du dessin des masques. La forme, la dimension et l'espacement

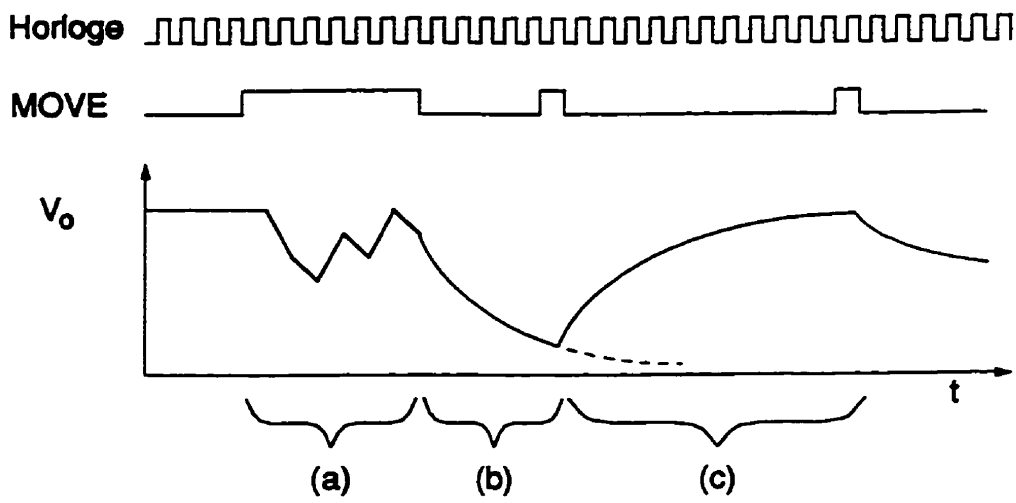


Figure 3.31 Réponse analogique des sorties du capteur MAR et limitations de la fréquence maximum de balayage. On peut effectuer un déplacement rapide (a), attendre un niveau de stabilisation partiel (b) ou complet (c) des signaux analogiques. Le mode de balayage est contrôlé en modifiant le nombre de cycles d'attente entre chaque déplacement du pixel d'intérêt.

ne sont pas respectés mais le sens, l'alignement et les inter-connexions concordent avec les détails de la réalisation physique. Un seul contact de polarisation du substrat P est placé par pixel. Cette région de diffusion P+ est à la fois l'anode de la photo-diode de la Figure 3.5 et le contact de polarisation de substrat des cinq transistors N. L'ensemble de ces contacts de polarisation de substrat est relié au noeud commun de référence analogique qui est polarisé à 0 V. La longueur du transistor d'amplification M_1 est plus grande que celle des transistors de sélection (voir la section 3.1.4). La Figure E.7 de l'ANNEXE E donne l'allure réelle d'un pixel MAR.

Le pixel de la Figure 3.32 est un pixel impair (P = 1) non décalé vers la droite. Dans ce cas, les signaux de sélection S_{X1} et S_{X2} sont propagés tel que montré. Dans le cas des pixels pairs (P = 0), le signal S_{X1} est propagé verticalement (à la place de S_{X2} pour la Figure 3.32) et S_{X2} est propagé en diagonale. La géométrie à 60° de ces deux signaux de sélection est réalisée en propageant chacun d'eux verticalement au travers d'une cellule, puis

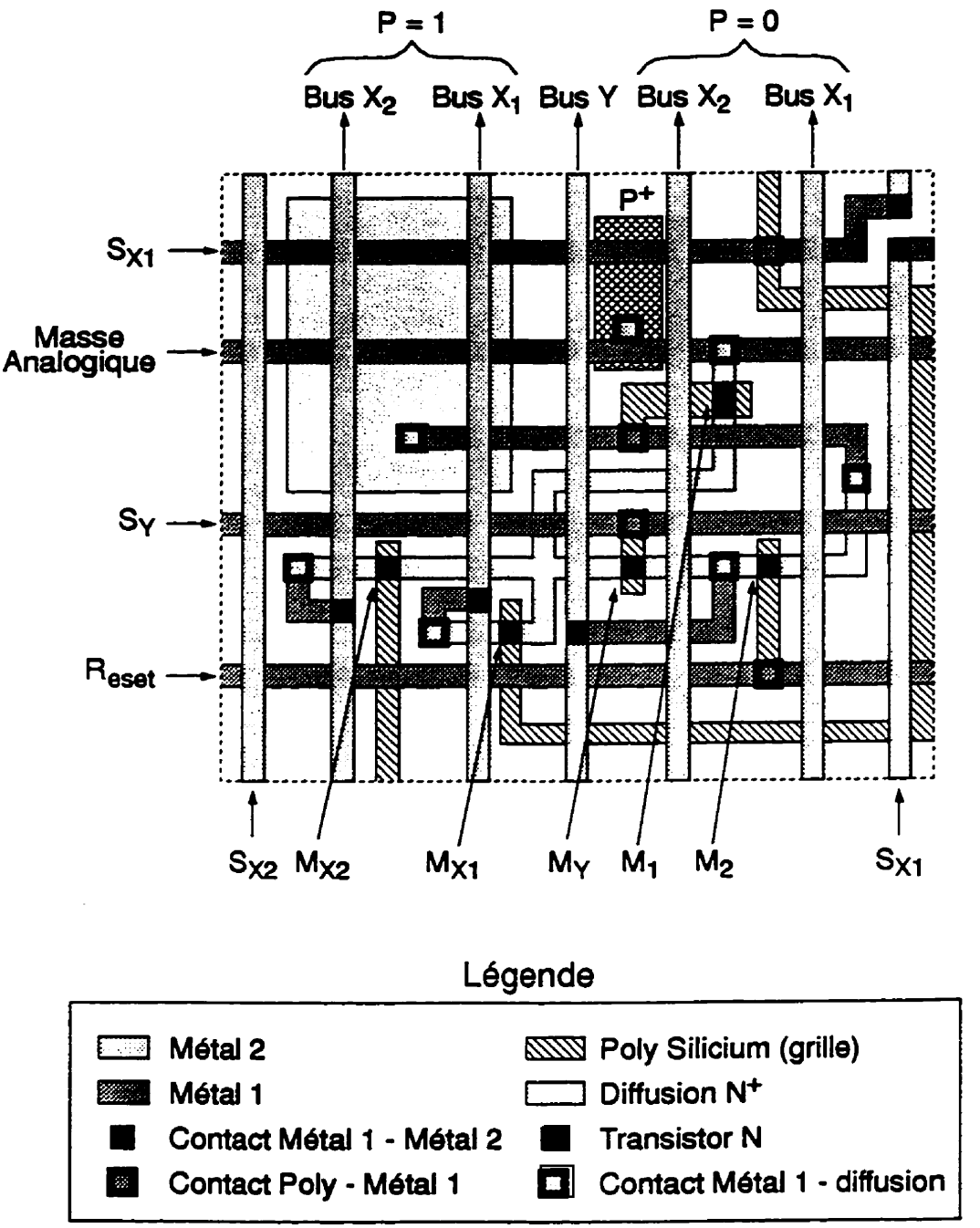


Figure 3.32 Détails géométriques de la réalisation VLSI du capteur MAR. La grande partie de diffusion N^+ est la photo-diode et la polarisation du substrat se fait par le contact à une région P^+ unique. Les étiquettes des transistors renvoient à la Figure 3.7 et à la Figure 3.8 (a).

verticalement et horizontalement (diagonalement) au travers de la cellule suivante. On obtient alors un pas de deux cellules en y et de une cellule en x ce qui définit bien un angle de 60°. En réalité, c'est un angle qui s'approche de cette valeur dû à l'approximation de $\sin 60°$ par le rapport 7/8 comme le montre l'équation (3-35).

$$\text{atan}\,(2\,(\frac{7}{8})) \;=\; 60.25° \qquad (3\text{-}35)$$

On peut remarquer à la Figure 3.32 que la dimension externe du pixel a un rapport 7/8 et que le centre de masse de la région de diffusion N+ qui forme la région photo-sensible est placé à 25% du pixel (horizontalement). On réalise l'arrangement hexagonal en plaçant les pixels des lignes impaires tel que montré à la Figure 3.32 et leurs miroirs verticaux pour les lignes paires. Un exemple de 3 X 3 pixels est montré à la Figure 3.33 où l'on a marqué en gras les trois lignes de sélection concurrentes ainsi que le pixel d'intérêt. On peut remarquer la façon dont les signaux de sélection se propagent en diagonale comme décrit au paragraphe précédent. Il est intéressant de faire le parallèle entre la Figure 3.33 et les lignes de sélection de la Figure 3.20 ou encore avec la propagation des bus de donnés analogiques de la Figure 3.27. On a volontairement supprimé le transistor M_2 et le signal R_{eset} pour alléger la Figure 3.33.

Un dernier détail géométrique concerne la forme de la jonction P-N^+ de la photo-diode de la Figure 3.32. Comme il a été mentionné à la section 3.1.2, le silicium est relativement opaque à la lumière visible ce qui fait que la périphérie de la jonction diode est plus photo-sensible que le centre de la région N^+. On dessine donc la photo-diode selon une géométrie en arbre plutôt qu'un simple carré comme montré à la Figure 3.32. La Figure 3.34 qui utilise la même convention que la Figure 3.32 montre les détails d'une telle géométrie. Il est cependant essentiel de prendre en considération les déplacement relatifs des différents masques lors de la fabrication du circuit intégré. Les règles de design sont définies afin de garantir un bon fonctionnement global même si l'alignement des masques de couches différentes n'est pas parfait [40]. Dans le cas d'un photo-détecteur, le déplacement relatif de quelques dixièmes de micron des couches de métallisation par rapport à la couche de diffusion a pour effet de déplacer la région d'exposition de la lumière au photo-détecteur. Dans le cas où tous les pixels sont identiques et qu'on considère ces déplacement relatifs comme étant identiques à la grandeur du circuit, on obtient une variation de sensibilité qui est également commune à l'ensemble des pixels. On retrouve à

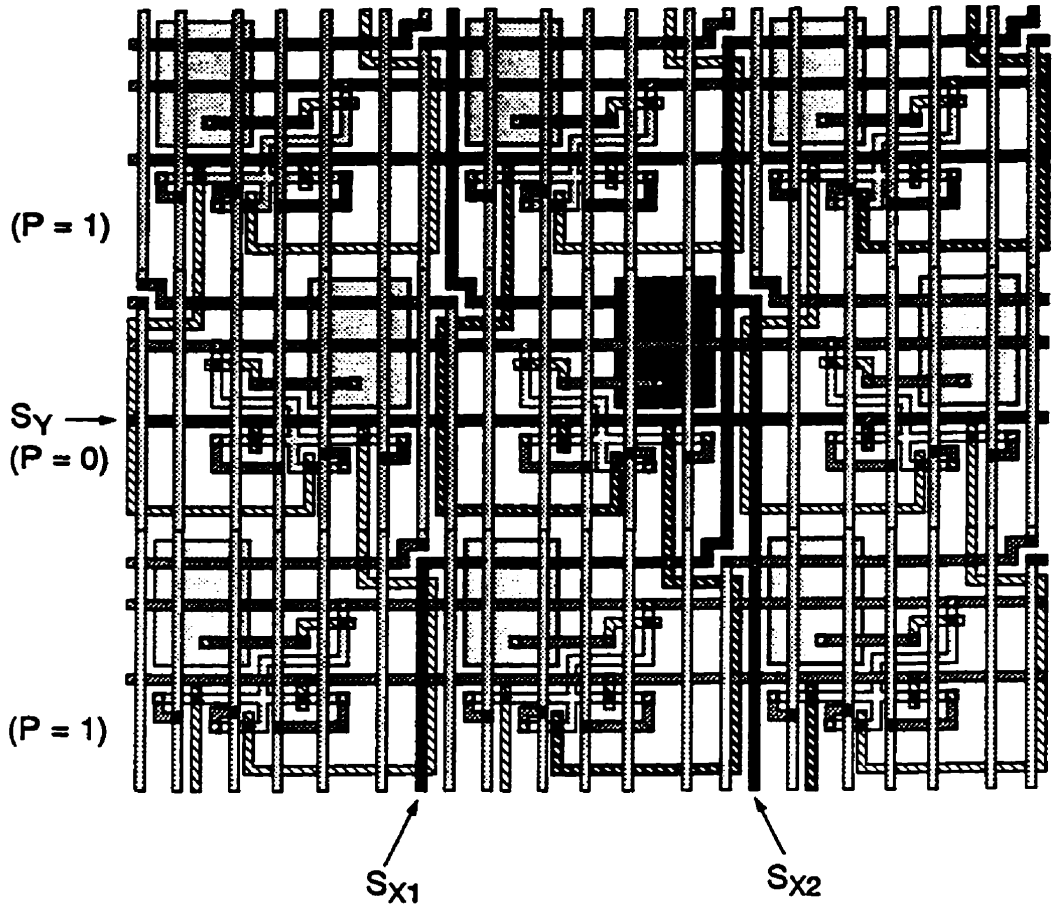


Figure 3.33 Exemple d'une matrice de 3 X 3 pixels MAR. On peut distinguer les trois lignes de sélection actives dessinées en gras de même que le pixel d'intérêt.

l'ANNEXE E (Figure E.8) un sous-ensemble de la matrice de capteurs selon les dimensions réelles tel que dessiné à l'aide du logiciel d'édition des masques.

Le capteur MAR utilise un pixel unique mais qui est inversé de la gauche à la droite une ligne sur deux pour créer l'effet de la topologie hexagonale comme montré à la Figure 3.33. Il en résulte qu'un déplacement vers la gauche des couches de métallisation pour un pixel pair est perçu comme un déplacement vers la droite pour les pixels impairs. Ceci à pour effet de causer une différence d'exposition à la lumière, donc de sensibilité, entre les pixels pairs et les pixels impairs. On règle ce problème en plaçant toutes les géométries

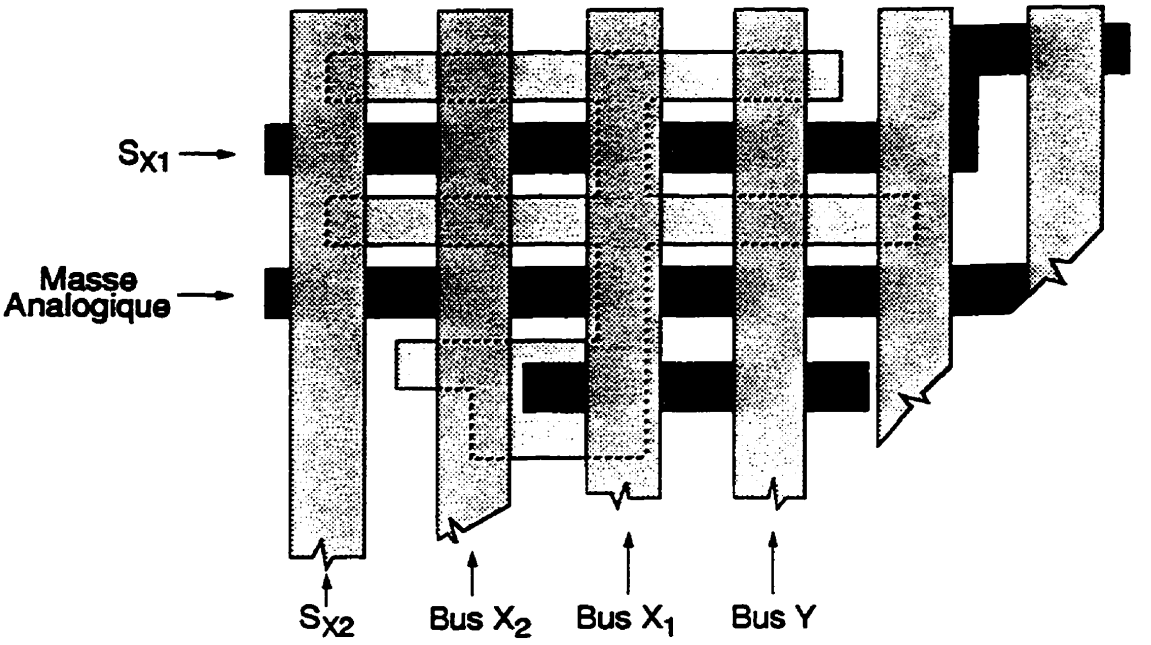


Figure 3.34 Géométrie du photo-détecteur en forme d'arbre. Les parties de la jonction PN qui sont verticales sont centrées sous un conducteur métallique pour garantir l'uniformité de la photo-sensibilité entre les pixels pairs et les pixels impairs.

verticales de la jonction PN de façon à ce qu'elles soient centrées sur un conducteur métallique vertical ou encore centrées entre deux conducteurs verticaux. Ce détail géométrique est présenté à la Figure 3.34 où la partie photo-sensible du pixel est montrée en détail. Il est facile de voir qu'un léger décalage horizontal de la deuxième couche de métallisation (conducteurs verticaux) par rapport à la couche de diffusion a le même effet sur la variation de sensibilité du pixel peu importe que ce décalage soit vers la gauche ou vers la droite. Tout décalage vertical est identique peu importe la parité de la ligne et n'implique pas de restriction spéciale sur la géométrie du photo-détecteur.

Il est de plus relativement important de recouvrir toutes les parties électroniques d'une des deux couches de métal car chaque portion de diffusion N^+ est intrinsèquement photo-sensible. Ce photo-courant parasite est faible bien que, dans le cas des drains des transistors de sélection, toutes les sorties des pixels d'une même colonne se retrouvent en parallèle (256) pour un bus de données analogiques unique et que leurs effets peuvent ne pas être négligeables. Il n'est pas vraiment important de protéger la partie numérique puisque celle-ci est commandée et régénérée à la cadence de l'horloge soit entre 5 et 45

MHz et que la période est trop courte pour que le photo-courant modifie de façon significative la valeur des noeuds dynamiques des registres à décalage. On observe en pratique une interférence causée par faisceau lumineux intense à une fréquence d'horloge de 1 KHz.

3.4.4 Circuits de génération et de propagation des horloges.

Le circuit de génération d'horloge à deux phases sans recouvrement est l'une des parties du capteur qui demande un design minutieux. Chaque phase de l'horloge ainsi que son inverse doit être propagée à l'ensemble des quatre registres à décalage comme montré à la Figure 3.14. Il en résulte donc une charge capacitive imposante sur chaque noeud puisqu'on doit alimenter pas moins de 1250 modules de registres à décalage. On utilise un circuit comme celui de la Figure 3.35 pour générer les deux phases d'horloge requises. La

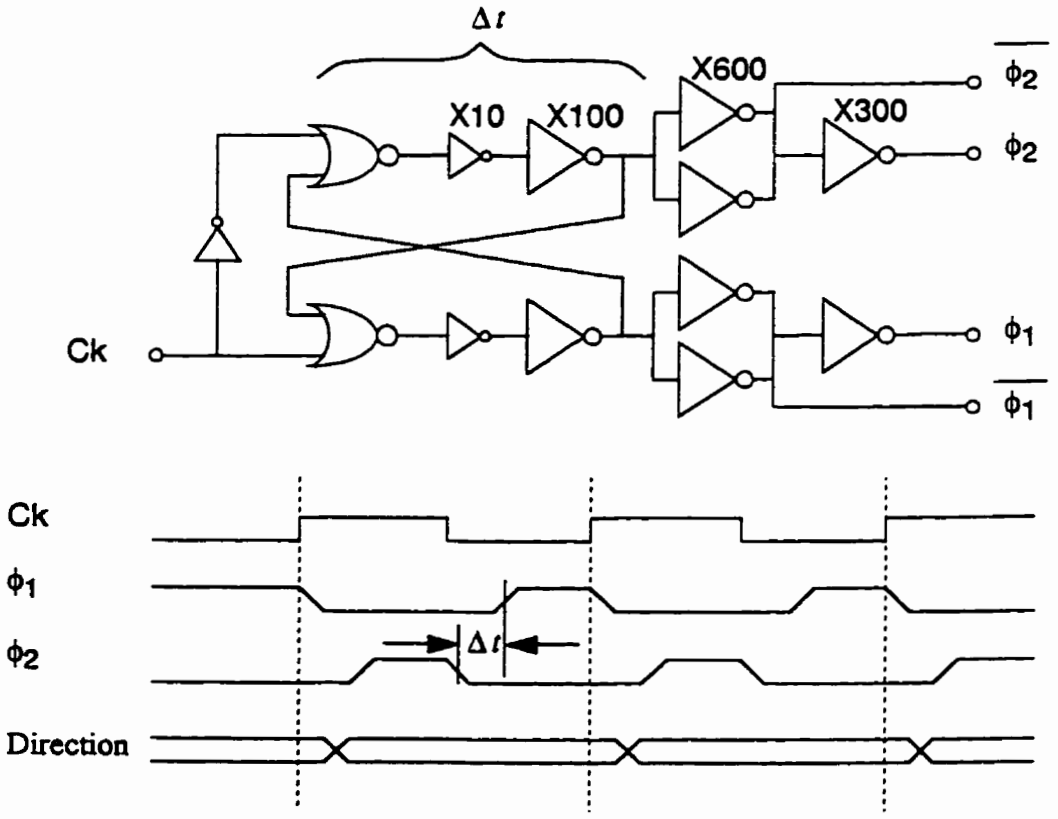


Figure 3.35 Circuit de génération des l'horloges à deux phases sans recouvrement et leurs circuits d'amplification.

dimension relative des transistors de chaque inverseur de cette chaîne d'amplification doit être croissante [40] afin d'éviter que les temps de montée et de descente des signaux d'horloge se détériorent.

Cette séquence d'inverseurs implique cependant un délai de propagation mais le même type d'amplificateurs est utilisé pour régénérer les signaux *MOVE* et *DIR* des registres à décalage qui seront eux aussi retardés par un même facteur. Il est de toute façon normal d'observer un changement d'état des signaux de commande synchrones (*Direction*) quelques nanosecondes après la montée de l'horloge de base Ck. En pratique, pour un design en technologie CMOS 1.2 µm, le délai de non-recouvrement (Δt) est de l'ordre de 5 ns et le temps de propagation des horloges est de 12 ns. On exigera donc que les signaux de commande (*Direction*) restent stables au moins 5 ns après une transition positive de l'horloge de base Ck. On peut d'ailleurs voir à la Figure 3.35 que la durée du non-recouvrement entre les deux phases Δt est fixé par les trois premières portes logiques et que ce délai est essentiel au bon fonctionnement des registres à décalage et au synchronisme de l'exécution des commandes de direction.

3.4.5 Historique de la réalisation VLSI

Plusieurs versions de capteurs MAR ont été soumises pour fabrication à la Société Canadienne de Microélectronique au cours des trois dernières années. Les premières versions ont été soumises pour la technologie cmos3dlm (CMOS 3 µm) vers la fin de l'année 1988. On a par la suite adapté le design à la technologie cmos4s (CMOS 1.2 µm) dès que cette technologie a été rendue accessible par la Société Canadienne de Microélectronique en novembre 1989. Au début du projet, plusieurs circuits de test ont été fabriqués dans les deux technologies afin de valider individuellement les concepts associés aux parties analogiques et numériques. Le premier capteur complet et fonctionnel a été soumis en juillet 1990 et fût reçu en novembre 1990. C'est d'ailleurs cette version de capteur de 128 X 128 pixels qui a été montée dans un boîtier de caméra et qui a permis les premières acquisitions d'images d'arêtes à multiples résolutions selon une topologie hexagonale. Le masque de convolution de cette première version effectue l'extraction de 49 pixels pour un rayon maximal de 8 pixels.

Au moment de parution de la thèse, la toute nouvelle version du capteur MAR soumise en janvier 1991 et reçue en juin 1991 est parfaitement fonctionnelle et est sur le point de remplacer l'ancienne version qui comportait certains défauts optiques et électroniques dont la différence de sensibilité entre les pixels des lignes paires et impaires.

Cette nouvelle version comprend une matrice de 268 lignes de 256 pixels et réalise une convolution sur un total de 91 pixels comme on l'a décrit tout au long de ce chapitre. Elle permet l'application de filtres spatiaux généralisés car le signal des 13 pixels au centre du masque de convolution est extrait sur des canaux individuels.

3.4.6 Description des plots de contact

On présente sous forme de référence la description des plots de contact pour la dernière version du capteur MAR qui devra être utilisée pour plusieurs autres travaux de recherche. La Figure 3.36 qui suit complète la description des différents modules du capteur MAR de la section 3.4.1. Le dé de silicium de ce circuit a des dimensions approximatives de 9 mm X 9 mm et est monté sur un boîtier de céramique de type PGA 68. Ce type de boîtier est le plus petit acceptable même si plusieurs broches ne sont pas utilisées. Ceci est dû au fait que tous les plots de contact sont situés sur le côté gauche ou droit du circuit.

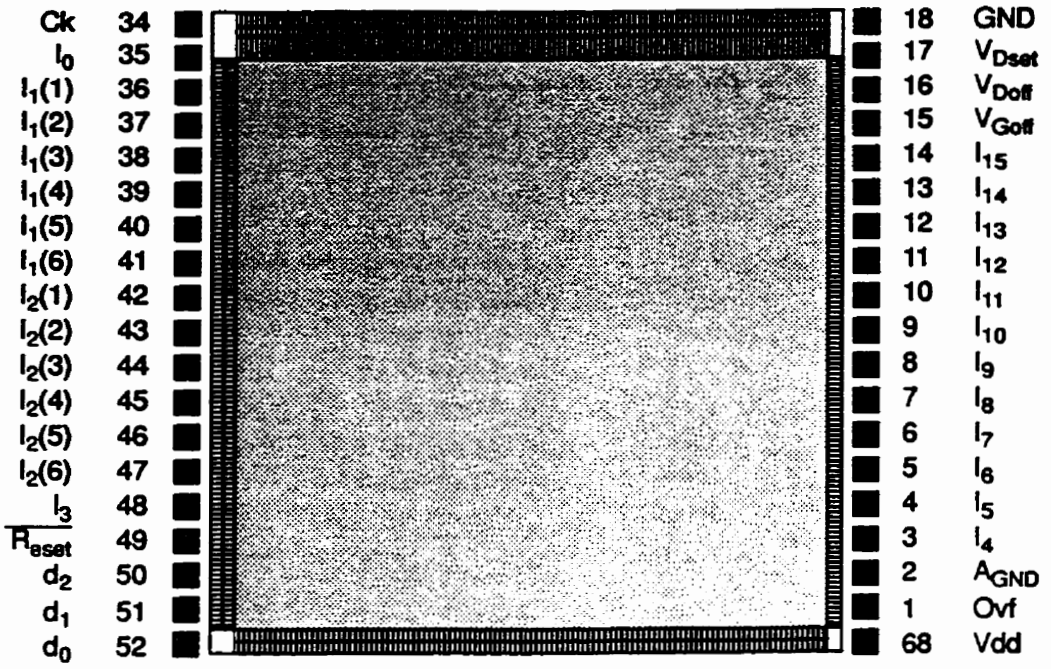


Figure 3.36 *Description des plots de contact pour la dernière version du capteur MAR.*

CHAPITRE 4

LE CONTRÔLEUR DE CAMÉRA

4.1 Description fonctionnelle

Le contrôleur de caméra est un circuit numérique CMOS spécialisé qui sert d'interface entre le capteur MAR et le système hôte en plus de gérer l'exécution autonome d'une gamme d'instructions de traitement d'images de bas niveau exécutable à l'intérieur même de la caméra. La description fonctionnelle du contrôleur MAR se divise en plusieurs sous-blocs qui contribuent de façon spécifique à la tâche globale du système MAR, comme par exemple l'extraction d'arêtes selon plusieurs résolutions spatiales. On présente, dans ce chapitre, la description détaillée de chaque élément qui compose cet organe microcodé dont le schéma de haut niveau est montré à la Figure 4.1. On peut remarquer que le contrôleur est conçu pour fonctionner avec une mémoire périphérique adressée conformément à la convention de la section 3.2.3 (Figure 3.18). Cette mémoire d'état fait l'acquisition, pour chaque pixel, d'un sous-ensemble programmable d'un grand nombre de variables de description d'état dont le choix revient au module de sélection des données. On remarque également à la Figure 4.1 que, conformément à la description de haut niveau de la Figure 2.1, un port numérique de sortie fournit les commandes de directions qui sont utilisées par

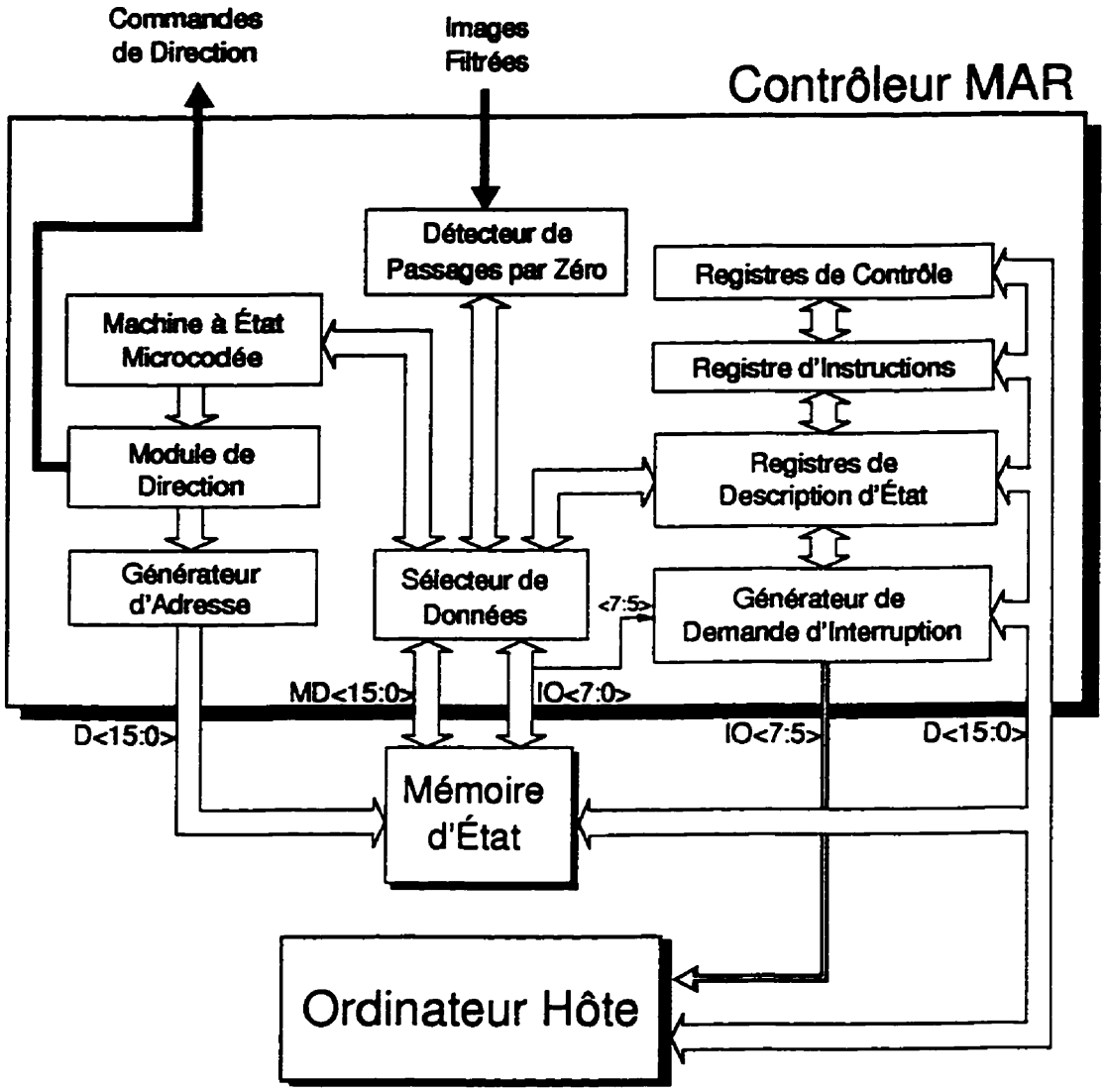


Figure 4.1 Diagramme bloc du contrôleur de caméra MAR.

le capteur de même qu'un port d'entrée analogique qui accepte les différentes images filtrées pour y extraire les passages par zéro, pré-traitement indispensable à la détection d'arêtes. Comme il a été mentionné à la section 2.2, le contrôleur MAR comprend un ensemble de registres de configuration et de description d'état qui sont tous accessibles par l'ordinateur hôte. Les signaux générés par le module de demande d'interruption complètent la série de canaux de communications entre le contrôleur et son environnement.

Il est a noter que, pour limiter le nombre de broches du circuit, le bus de données numérique *D* (à droite sur la Figure 4.1) est confondu avec le bus d'adresse de la mémoire d'état (à gauche sur la Figure 4.1). Cette utilisation partagée du bus de données entraîne une contrainte de fonctionnement qui interdit l'accès à la mémoire d'état par le contrôleur lorsque l'ordinateur hôte fait accès aux registres internes du contrôleur et vice versa. Cette contrainte est acceptable puisque le contrôleur est normalement inopérant lorsque l'ordinateur hôte programme la prochaine opération à être exécutée par le système MAR ou encore, il est en mode d'arrêt temporaire lors d'une requête d'interruption. Les données de description d'état se propagent à travers le bus nommé *MD* (Mémoire Données). C'est un bus de 16 bits qui est utilisé pour propager la description d'états sélectionnée par le contrôleur de données. On retrouve également un bus numérique de 8 bits nommé *IO* qui peut véhiculer de l'information descriptive ou contrôler de façon externe certaines fonctions du contrôleur. En plus de permettre la vérification fonctionnelle du circuit, les trois bits les plus significatifs de ce bus (IO<7:5>) identifient les trois niveaux possibles de requête d'interruption. Les autres broches de ce circuit intégré de 68 entrées/sorties sont utilisées pour propager les adresses des registres internes et les signaux d'accès à ces registres tel que discuté à la section 2.2. Les sections qui suivent présentent la description détaillée de chaque sous-module de la Figure 4.1.

4.2 Microcode et jeu d'instructions

Cette section décrit les aspects logiques et fonctionnels de la machine à états microcodée qui constitue le principal organe intelligent du contrôleur MAR. On y présente la façon dont le contrôle de la séquence des micro-instructions s'effectue en incluant un aperçu du contenu du microcode lui-même. Le jeu d'instructions du système MAR est présenté de même que les différentes façons offertes à l'usager pour déplacer le pixel d'intérêt. Le contexte d'utilisation de chacun de ces modes de balayage de même que le mode de poursuite d'arêtes sont également décrits. On doit toujours garder à l'esprit que chaque instruction implique un déplacement du masque de convolution sur le capteur MAR de même qu'une mise à jour cohérente du pointeur de la mémoire périphérique. Il est par conséquent possible d'effectuer le traitement désiré sur l'image provenant du capteur, tout comme sur les données présentes dans la mémoire périphérique ou même sur une combinaison des deux.

4.2.1 Unité de logique programmable (PLA)

La machine à états microcodée se compose d'un bloc de logique programmable (PLA) constitué de 158 lignes d'un microcode spécifique au contrôleur MAR. La

description électronique de ce PLA, qui compte 18 entrées et 24 sorties, est présentée à la section 4.5.1. On peut voir à la Figure 4.2 le plan d'ensemble de l'utilisation d'un PLA pour réaliser une machine à états. On réserve 5 signaux d'entrée et 5 signaux de sortie pour définir une variable d'état qui joue le rôle d'un pointeur servant à contrôler la séquence de l'exécution du microcode. On remarque qu'une partie du registre d'instructions est dirigée directement aux entrées du PLA alors que d'autres sont utilisées par l'unité arithmétique de direction pour effectuer le calcul de la commande de direction, ou par le reste de la logique du contrôleur MAR.

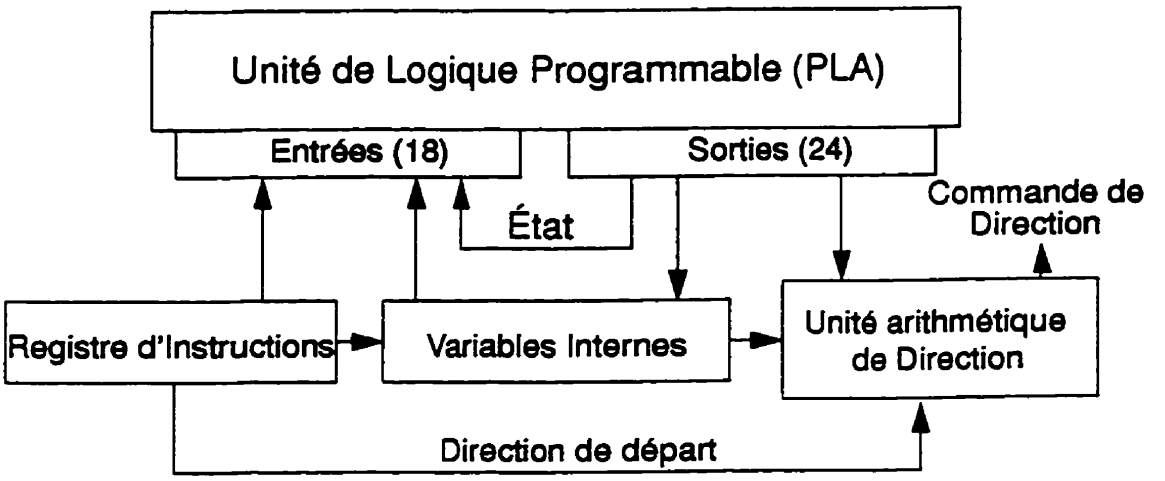


Figure 4.2 Plan d'ensemble de l'utilisation d'un PLA comme machine à états. On utilise 5 entrées et 5 sorties pour définir une variable d'état qui joue le rôle d'un pointeur de programme lors de l'exécution du microcode.

Un ensemble de variables internes provenant des autres modules du contrôleur complète la liste des entrées et des sorties du PLA. Des signaux de sorties sont utilisés comme variables de description d'état du pixel d'intérêt alors que d'autres servent à conditionner, diversifier ou stopper l'exécution du microcode. Certains de ces modules internes sont des compteurs d'événements. Deux de ces compteurs d'événements servent à conditionner l'exécution de plusieurs instructions: le compteur N et le compteur E. De façon générale, le compteur N est utilisé pour compter le nombre de déplacements effectués depuis le début de l'exécution d'une instruction et le compteur E sert à répertorier le nombre d'exceptions rencontrées durant l'exécution de cette instruction. Les deux

compteurs ont une valeur limite programmable qui indique généralement la fin de l'exécution d'une instruction lorsqu'une de ces limites est atteinte. Pour certaines instructions, l'exécution se termine lorsque les deux compteurs N et E atteignent leur limite. Un troisième compteur, nommé C, gère le temps d'attente de la stabilisation des signaux analogiques conformément à la discussion de la section 3.4.2. Ce cycle d'attente, qui est de l'ordre de 5 à 20 cycles d'horloge, est exécuté après chaque déplacement sauf dans le cas des instructions de déplacement rapide.

Plusieurs versions du microcode ont été produites depuis le début du projet mais celle qui a été utilisée pour la fabrication du plus récent prototype du contrôleur MAR (septembre 1991) est présentée à l'ANNEXE B. Cette annexe contient également la description et la nomenclature des entrées et des sorties du module de logique programmable utilisé par le contrôleur pour effectuer la séquence microcodée des déplacements du pixel d'intérêt.

4.2.2 Registre d'instructions

Le principal registre du système MAR est le registre d'instructions. Il comprend 16 bits divisés en différents champs comme montré à la Figure 4.3. Le code de l'instruction utilise un champ de quatre bits complété d'un champ de trois bits donnant la direction de

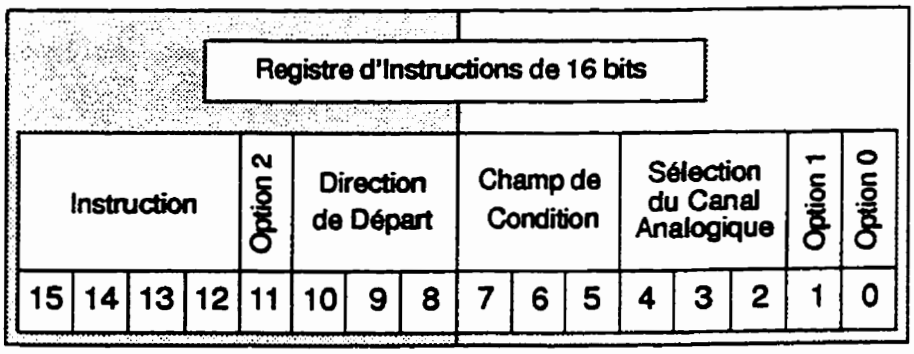


Figure 4.3 Registre d'instructions du contrôleur MAR. Le système hôte utilise ce registre pour définir le mode de balayage du capteur.

départ pour cette instruction. Trois bits non contigüs définissent certaines options ou variantes de l'instruction. Un champ de trois bits rend l'exécution de l'instruction conditionnelle à certains paramètres d'état. Finalement, un champ sert à choisir lequel des 8 canaux analogiques sera utilisé pour effectuer le traitement, l'acquisition d'images ou le suivi d'arêtes.

Le champ Instruction de même que les trois bits d'option sont utilisés directement comme entrées du PLA, alors que le champ de condition jumelé à certaines variables internes du contrôleur définissent le signal de détection de la frontière de balayage *C* (entrée #6 du PLA du Tableau B.1). Le champ de direction de départ (*DIR_R<2:0>*) est utilisé par l'unité arithmétique de direction décrite à la section 4.3.7. Finalement, le champ de sélection du canal analogique est utilisé exclusivement par les instructions de recherche d'arêtes et commande un multiplexer 8 à 1. L'opération d'écriture à l'adresse correspondant au registre d'instructions entraîne l'exécution d'une instruction. Un module d'arbitration des instructions gère les temps morts entre l'exécution de deux instructions et sa description fait l'objet de la section 4.3.1.

Pour simplifier la programmation du registre d'instructions de même que certains registres associés à la séquence des instructions comme par exemple les compteurs d'événements, on a défini un langage spécifique facilitant la programmation et la documentation des logiciels d'exploitation. Les détails de ce pré-processeur au langage C de même que la syntaxe choisie sont présentés à l'ANNEXE C. Les sections qui suivent présentent le jeu d'instructions du contrôleur MAR et donnent les principales caractéristiques des différents modes de déplacement du pixel d'intérêt. Le code spécifique de programmation de chaque instruction se trouve à l'ANNEXE C.

4.2.3 Instructions de contrôle

Les instructions de contrôle ne sont définies que pour initialiser le capteur ou le mode d'opération du contrôleur MAR.

4.2.3.1 L'instruction: *RESET_POSITION*

Cette instruction qui est décrite à la section C.5.11 fixe l'origine du référentiel image. Son action concrète consiste à remettre à zéro le pointeur horizontal, le pointeur vertical, ou encore les deux dépendant de la valeur du champ option. On peut envisager une calibration dynamique du référentiel de l'image en fonction d'un repère actif qui serait volontairement placé dans la scène, ou encore placer la référence de position à l'une ou l'autre des 9 positions suggérées par la Figure 4.4.

Figure 4.4 Position possibles de l'origine du référentiel image du contrôleur MAR sur la surface photosensible du capteur.

4.2.3.2 L'instruction: *SET_CAPTEUR*

L'exécution de cette instruction active le signal R_{esti} qui initialise les registres à décalage du capteur et qui remet à zéro le processus d'intégration de l'illuminance conformément aux discussions des sections 3.2.4 et 3.3.6. Bien que l'initialisation du capteur entraîne inévitablement une nouvelle prise d'images, la remise à zéro du processus d'intégration de l'illuminance peut être déclenchée sans modifier l'état des registres à décalage en effectuant l'instruction *SET_CAPTEUR* (section C.5.13) avec la direction de départ *0* (aucun déplacement).

4.2.3.3 L'instruction de remise à zéro du contrôleur

Cette instruction n'est exécutée qu'une seule fois lors de la mise sous tension du système et garantit le bon fonctionnement de la machine à états. En effet, il est possible qu'à la mise sous tension, l'état initial du PLA jumelé à celui des différents registres de contrôle conduise à l'exécution d'une boucle sans fin. Cette instruction force le retour inconditionnel à un état interne stable en attente d'une nouvelle instruction. On réalise une remise à zéro du contrôleur en plaçant l'instruction *SET_CAPTEUR* dans le registre d'instructions puis forçant une interruption d'instruction (le signal *STOP* de la section 4.3.3).

4.2.4 Modes de balayage local du pixel d'intérêt

Les instructions de déplacement du pixel d'intérêt se répartissent selon trois modes: le mode rectiligne, le mode zigzag et le mode hélicoïdal. Comme on peut le voir à la Figure 4.5, le mode rectiligne progresse d'un pixel à la fois dans la direction désirée alors que les

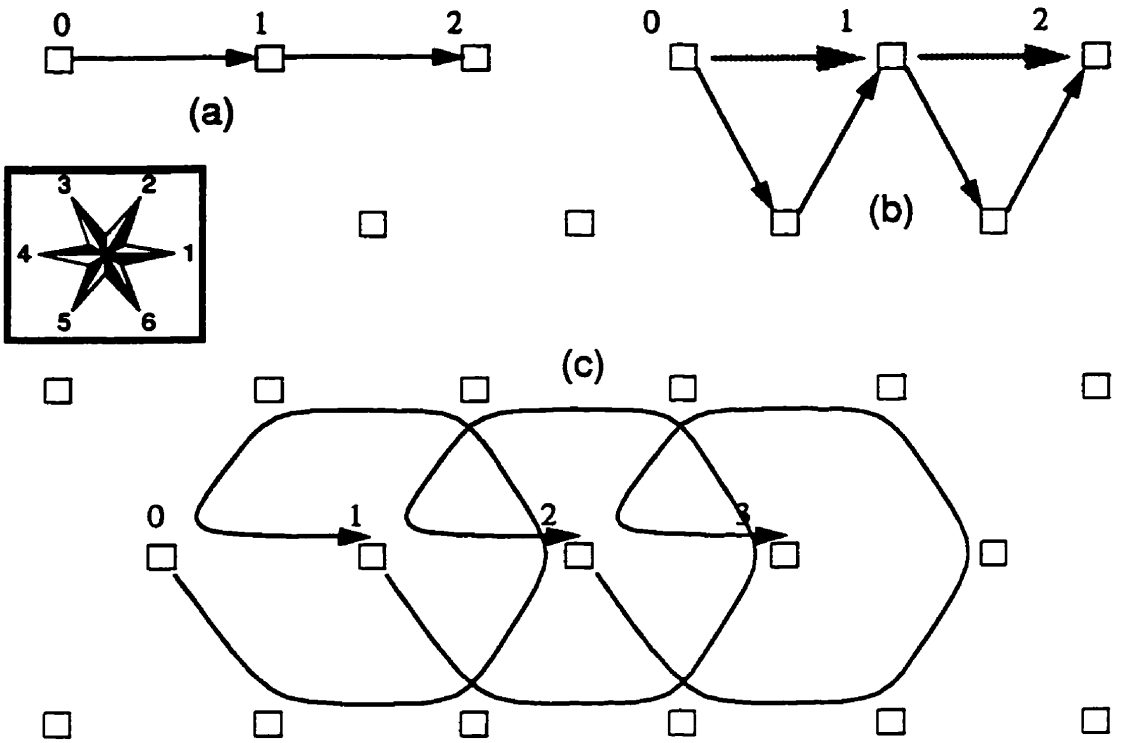


Figure 4.5 Différents modes de déplacement applicables à la plupart des instructions de déplacement du pixel d'intérêt. On effectue deux déplacements rectilignes en (a), deux déplacements en mode zigzag en (b) et trois déplacements selon le mode hélicoïdal en (c). Tous les déplacements de cet exemple sont effectuées dans la direction 1.

modes en zigzag et hélicoïdal impliquent une trajectoire plus ou moins complexe avant de terminer le cycle de déplacement dans la direction choisie.

Les deux modes complexes de déplacement peuvent être exploités par un système dédié, placé en périphérie du système MAR, et qui requiert des données dans un ordre propice au calcul pour lequel il est défini. Le mode de déplacement en zigzag est utile pour un sous-système qui calcule le recouvrement de surfaces douces comme il est discuté à la section 6.3.5. Le mode de déplacement hélicoïdal sert à plusieurs fonctions qui doivent visiter les six voisins immédiats avant de calculer une propriété associée au pixel central. Il est important de noter que le test de détection de la frontière de l'image ne s'effectue qu'à

la fin d'une séquence de déplacement complexe. De la même façon, un compteur d'événements associé à un déplacement est incrémenté après la séquence complète de déplacement complexe. On ne peut interrompre une instruction que lorsque la séquence associée à un mode de déplacement est complétée.

4.2.5 Direction de déplacement généralisé

L'unité arithmétique de direction comprend deux signaux de contrôle nommés *L* et *R* qui modifient la trajectoire prévue du pixel d'intérêt vers la gauche ou vers la droite respectivement. Ces signaux sont commandés par deux compteurs qui activent de façon cyclique le signal *L* puis le signal *R* afin de définir un déplacement selon un angle quelconque comme on peut le voir à la Figure 4.6. Le déplacement présenté dans cet

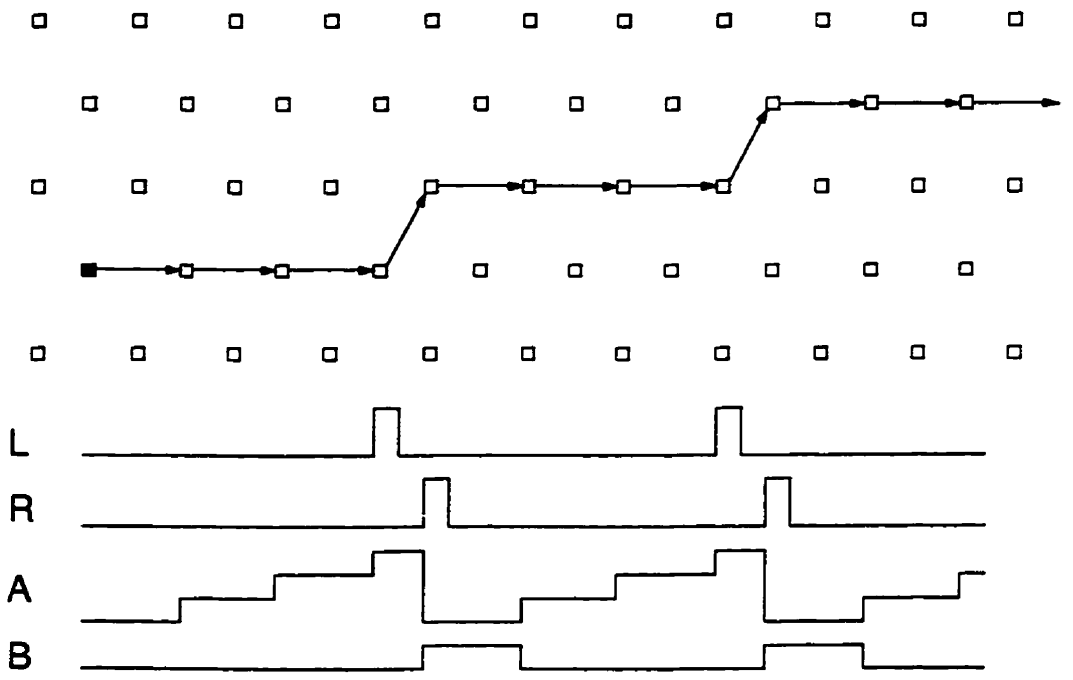


Figure 4.6 *Exemple de déplacement généralisé. On montre la période d'activation des deux signaux de contrôle L et R. Les valeurs limites respectives placées dans les deux compteurs d'événement A et B sont 3 et 1.*

exemple est exécuté trois fois dans la direction de départ (en occurrence la direction 1) puis une fois dans la direction de départ+1. Un registre de contrôle spécifie si la première correction se fait vers la gauche (comme l'exemple de le Figure 4.6) ou vers la droite et on doit également programmer la valeur limite des deux compteurs d'événements A et B. Dans l'exemple de la Figure 4.6, les compteurs A et B sont initialisés à 3 et à 1 respectivement. Ce type de déplacement généralisé modifie uniquement la trajectoire prévue pour une instruction donnée. On peut appliquer, à chaque instruction en mode de déplacement généralisé, les trois modes de balayage local du pixel d'intérêt soit: rectiligne, en zigzag et hélicoïdal.

4.2.6 Instructions de déplacement du pixel d'intérêt

Le reste du jeu d'instructions du système MAR est utilisé pour déplacer de diverses façons le pixel d'intérêt à l'intérieur de la frontière de l'image. Ces instructions sont séparées en trois classes: les instructions de déplacement simple, les instructions de déplacement complexe et l'instruction intelligente de poursuite d'arêtes.

4.2.7 Instructions de déplacement simple

Les instructions de déplacement simple sont définies soit pour déplacer le pixel d'intérêt d'une région de l'image à une autre, soit pour définir une séquence très particulière de balayage d'une région de l'image. Chacune de ces instructions peut être exécutée selon l'un ou l'autre des mode de balayage présentés à la section 4.2.4 bien que dans certain cas, cette définition n'a aucun sens comme c'est le cas des déplacement rapides ou du déplacement unique.

4.2.7.1 Déplacement rapide: *FAST_MOVE* et *FAST_MOVE_N*

Les instructions de déplacement rapide sont utilisées exclusivement pour faire passer le pixel d'intérêt d'un endroit de l'image à un autre. On utilisera par conséquent le mode de déplacement rectiligne et il sera possible de se déplacer d'un pixel à chaque coup d'horloge. Pour des raisons d'architecture interne du module de déplacement généralisé, on ne pourra déplacer le pixel d'intérêt que dans l'une ou l'autre des six directions de base pour ces deux instructions. L'instruction *FAST_MOVE* (section C.5.1) déplace le pixel d'intérêt d'une position dans la direction de départ précisée par le registre d'instructions alors que l'instruction *FAST_MOVE_N* (section C.5.2) effectue autant de déplacements dans cette même direction qu'il est spécifié par la valeur limite du compteur d'événement N et ce, à la cadence d'un déplacement par cycle d'horloge du contrôleur.

De façon générale, on pourra déplacer le pixel d'intérêt d'un point de l'image à un autre grâce à une séquence de deux instructions *FAST_MOVE_N* orientées dans des directions différentes. Le système ne fait pas d'écriture à la mémoire d'état lors des déplacements rapides puisque plusieurs variables de description d'état internes au contrôleur de même que l'état des signaux analogiques ne sont pas valides. Pour cette raison toute instruction exécutée immédiatement après une instruction de déplacement rapide démarre d'abord par une petite procédure d'exception qui s'assure que les variables internes sont valides au point de départ. Cette procédure ne fait que déplacer le pixel d'intérêt d'une position dans la direction opposée à la direction de départ puis, revient au point de départ. Ces deux déplacements se font à la manière d'un déplacement de type *MOVE* sans écriture à la mémoire d'état puis, l'instruction débute immédiatement après.

4.2.7.2 Déplacement linéaire: *MOVE*, *MOVE_N* et *MOVE_ZIGZAG_N*

Les instructions de déplacement linéaires sont utilisées pour programmer une trajectoire simple par opposition aux trajectoires complexes traitées à la section 4.2.8. De façon similaire aux instructions de déplacement rapide, l'instruction *MOVE* (section C.5.4) déplace le pixel d'intérêt d'une position dans la direction de départ accompagné d'une écriture à la mémoire MAR. L'instruction *MOVE_N* (section C.5.7) effectue *N* déplacements dans la direction de départ. L'instruction *MOVE_ZIGZAG_N* (section C.5.10) commande un déplacement de *N* pixels selon le mode zigzag tel que montré à la Figure 4.5 (b). L'instruction *MOVE_N* avec l'option *HELIX* commande un déplacement linéaire en mode hélicoïdal comme à la Figure 4.5 (c).

4.2.7.3 Rebondissement aux frontières d'une région de l'image

Chaque instruction comprend un champ qui définit quelle variable de description d'état ou combinaison de variables est utilisée pour définir la frontière d'une région de l'image. Cette frontière peut être simplement carrée et être associée à la dimension externe du capteur MAR ou encore elle peut correspondre à un contour fermé. Cette frontière peut également être associée à un contour d'arête combiné à la fenêtre d'étude ou même provenir d'un signal externe au contrôleur. Chaque instruction exécutée par le contrôleur MAR vérifie si le pixel d'intérêt se situe sur la frontière ou non. Les instructions de déplacement simple utilisent la définition de cette frontière pour réagir sous forme d'un rebondissement lorsque le déplacement du pixel d'intérêt active cette condition limite. Pour les instructions de déplacements complexes (section 4.2.8) de même que pour l'instruction de poursuite de frontières (section 4.2.9), l'exécution de l'instruction se termine lorsque la frontière est atteinte.

On définit un mode de rebondissement programmable à l'aide du champ *Option* pour les instructions *MOVE_N* et *MOVE_ZIGZAG_N*. On peut forcer l'interruption de l'instruction immédiatement après avoir atteint une frontière ou encore forcer un certain nombre de rebondissements vers la gauche ou vers la droite provoquant ainsi une forme de balayage aléatoire comme on peut le voir à la Figure 4.7. Ce mode de balayage correspond

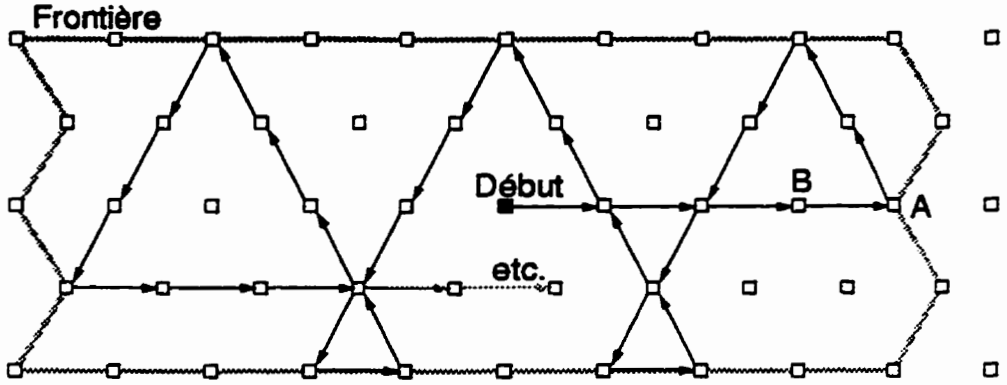


Figure 4.7 Exemple d'exécution de l'instruction "MOVE_N" avec activation de l'option de rebondissement. Le pixel noir représente le point de départ. On effectue ici les rebondissement vers la gauche soit la direction précédente +2. Le contour gris correspond aux points où la condition est active (ou frontière).

en fait à une généralisation de la fonction de rebondissement utilisée par d'autres instructions. Il peut quand même être intéressant d'utiliser, dans certain cas, ce mode de balayage avec rebondissement aux frontières pour effectuer un balayage de façon aléatoire.

Les instructions sont définies de façon à ce qu'une instruction ne se termine jamais avec le pixel d'intérêt qui réside sur la frontière. Lorsque, par exemple, on désactive l'option de rebondissement pour l'instruction *MOVE_N* et que la frontière est atteinte (comme au point A de la Figure 4.7), l'algorithme revient sur ses pas d'une position (point B de la Figure 4.7) avant de terminer l'instruction. Cette caractéristique est essentielle puisqu'une instruction ne peut pas être exécutée si, au premier déplacement, la condition à la frontière est active. Le cas échéant, on doit forcer un déplacement inconditionnel pour ramener le pixel d'intérêt à l'intérieur de la frontière ou encore modifier la frontière pour assurer une désactivation de la condition.

4.2.8 Instructions de déplacement complexes

Les instructions de déplacement complexes servent à programmer des trajectoires de balayage nécessaires à l'application de certains algorithmes de traitement d'images. Chacune des instructions complexes présentée dans cette section peut forcer un déplacement du pixel d'intérêt en mode rectiligne, zigzag ou hélicoïdal qu'il s'agisse d'une trajectoire triangulaire ou hexagonale ou encore d'une fonction de balayage de l'image.

4.2.8.1 Déplacement triangulaire: *MOVE_TRI* et *MOVE_TRI_N*

Les instructions de déplacement triangulaires ne sont pas implantés volontairement au jeu d'instructions du contrôleur. Ils découlent plutôt de la généralisation du mode de balayage hexagonal (section 4.2.8.2) où l'ajout d'un nombre restreint de lignes au microcode permet de définir le mode de balayage triangulaire. Le trajet suivi par le pixel d'intérêt est montré à la Figure 4.7 lors de l'exécution des l'instructions *MOVE_TRI* (section C.5.8) en (a) et *MOVE_TRI_N* (section C.5.9) en (b) et (c).

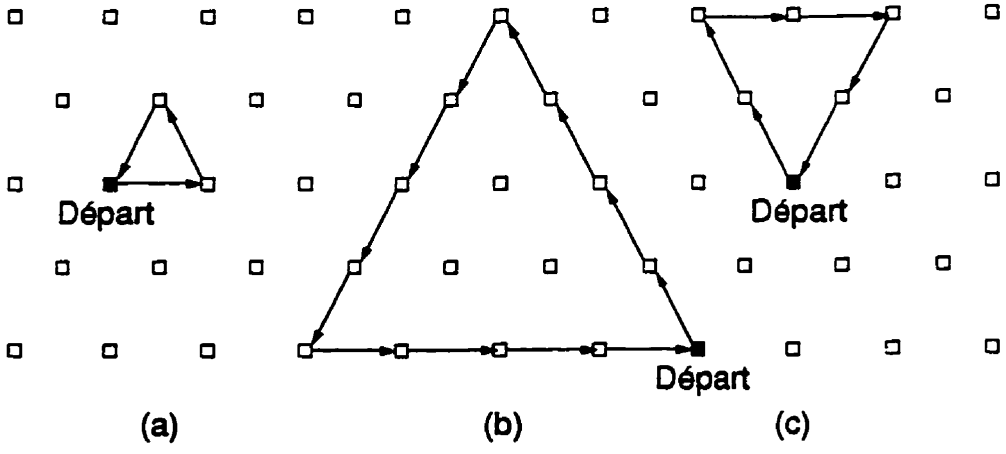


Figure 4.8 Exemple d'exécution des instructions "MOVE_TRI" (a) et "MOVE_TRI_N" (b) avec l'option de virage à gauche active. On utilise l'option de virage à droite en (c). La longueur des côtés des triangles est définie par le compteur N qui vaut 4 en (b) et 2 en (c). La direction de départ est 1 en (a) et 3 en (b) et (c). Le compteur E doit avoir une valeur minimum de 3 pour effectuer le balayage d'un triangle complet.

La direction de départ est programmable de même que le sens de rotation. Pour l'instruction *MOVE_TRI_N*, le compteur d'événements N définit la longueur des arêtes de la trajectoire triangulaire et le compteur E définit le nombre de côtés qui seront parcourus

avant de terminer l'instruction. Les modes de balayage en zigzag et hélicoïdal sont valides pour l'instruction *MOVE_TRI_N* et l'exécution d'une instruction de déplacement triangulaire est avortée lorsque la condition qui définit la frontière est activée.

4.2.8.2 Déplacement hexagonal: *MOVE_HEX* et *MOVE_HEX_N*

Les instructions de déplacement hexagonaux sont similaires aux instructions de déplacement triangulaires sauf pour le changement de direction qui survient aux coins de la géométrie de la trajectoire. Comme on peut le voir à la Figure 4.7 le balayage d'un

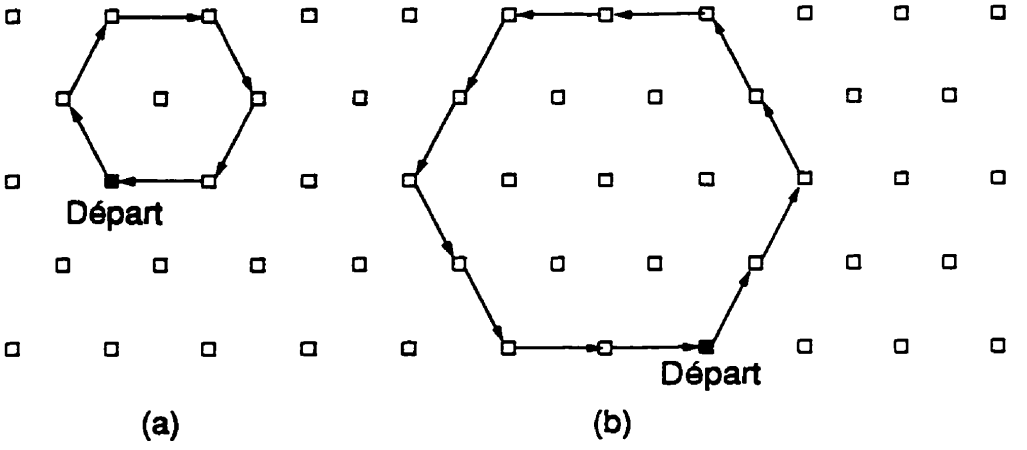


Figure 4.9 Exemple d'exécution des instructions "MOVE_HEX" (a) et "MOVE_HEX_N" avec N = 2 (b). Le nombre de côtés tracés par chacune de ces instructions est défini par le compteur E qui doit avoir une valeur minimale de 6 pour tracer une fois un hexagone complet. Le mode de déplacement rectiligne est montré ici mais les modes zigzag et hélicoïdal peuvent aussi être utilisés.

hexagone unitaire s'exécute à l'aide de l'instruction *MOVE_HEX* (section C.5.5) comme à la Figure 4.7 (a) ou selon une trajectoire hexagonale dont la longueur des côtés est programmable par le compteur d'événement N (section C.5.6).

Le principal champ d'application du balayage hexagonal est le recouvrement de l'information tridimensionnelle à partir de l'illuminance. Comme on le verra à la section 6.3.5 et à l'ANNEXE G, cet algorithme fait croître une solution selon des hexagones concentriques. Les instructions de déplacement hexagonal sont également utiles pour la reconnaissance de patrons ou le suivi d'objets mobiles.

4.2.8.3 L'instruction de balayage de région: *SCAN_AREA*

Le jeu d'instructions du système MAR comprend une instruction de balayage séquentiel de l'image. Cette instruction déplace le pixel d'intérêt de façon à visiter tous les pixels qui sont situés à l'intérieur de la frontière. Ce mode est similaire à celui d'une caméra vidéo standard. L'instruction peut débuter dans l'une ou l'autre des six directions possibles, ce qui autorise un balayage diagonal. Les modes de déplacement en zigzag ou hélicoïdal s'appliquent également à cette instruction.

Pour balayer l'image au complet, on programme la frontière à la dimension du capteur. Il existe différentes options à l'instruction *SCAN_AREA* (section C.5.12) comme le balayage d'une ligne avec retour rapide nommé *RASTER*, le balayage boustrophédonique nommé *COIL* et le balayage double nommé *TWICE*. Ces trois variantes de l'instruction de balayage de région sont montrées à la Figure 4.10. Le balayage dédoublé sert à l'implantation d'opérateurs récursifs séparables impliquant le balayage d'une ligne dans une direction puis, un second balayage dans la direction opposée. Certains opérateurs morphologiques séparables tirent profit de l'option *TWICE*.

Le nombre de pixels visités lors de l'exécution de l'instruction *SCAN_AREA* est un excellent estimateur de l'aire de la région définie par le champ condition. Lors du balayage d'une région avec une frontière de forme arbitraire, on peut estimer l'aire à l'intérieur de celle-ci ou encore recueillir des statistiques sur l'ensemble des pixels qui la composent (la densité des arêtes, des histogrammes, calcul de moyennes, etc.).

4.2.9 L'instruction intelligente de poursuite d'arêtes: *FIND*

L'instruction de poursuite d'arêtes constitue la partie essentiellement intelligente du microcode implantée dans le système MAR. Cette partie de microcode représente environ 25% des 160 lignes de code intégrées dans la machine à états. La Figure 4.11 illustre un exemple du trajet effectué par le pixel d'intérêt lors de l'exécution de l'instruction *FIND* (section C.5.3). L'algorithme de cette instruction débute par la recherche d'une arête dans la direction de départ. Une fois un point d'arête trouvé, la poursuite se fait en mode zigzag de part et d'autre de l'arête tant que la détection est validée. Lorsque la détection de l'arête est interrompue (comme au déplacement #10 de la Figure 4.11), l'algorithme déplace alors le pixel d'intérêt de façon à fermer le triangle (déplacement 11 de la Figure 4.11) en plus de signaler la présence d'une discontinuité d'arête. A ce point, on doit nécessairement retrouver l'arête et poursuivre la recherche dans cette nouvelle direction (déplacements 12 à 13 de la Figure 4.11).

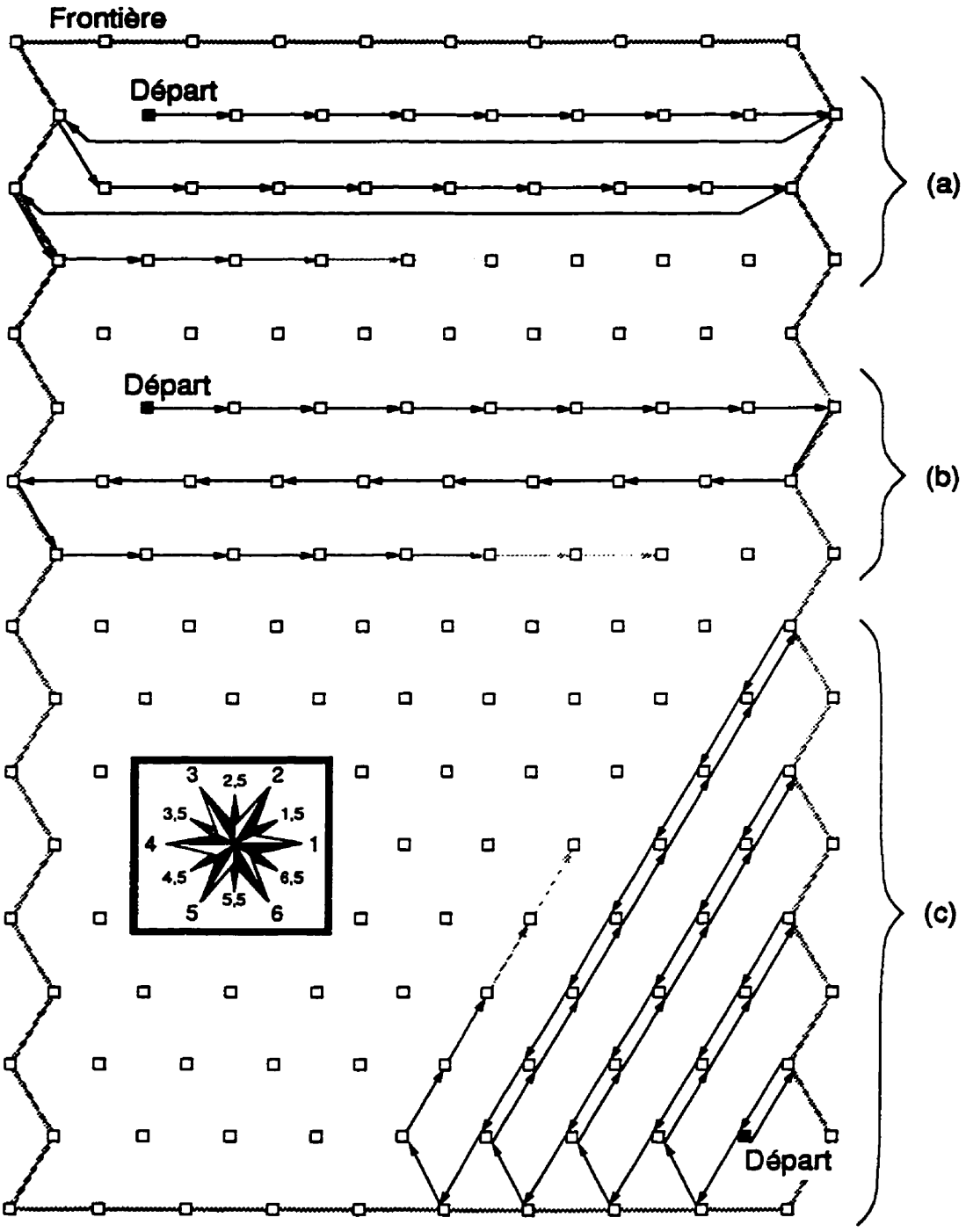


Figure 4.10 *Instruction de balayage de régions. On peut effectuer le balayage d'une ligne avec un retour rapide (a) comme celui d'un signal vidéo standard ou un balayage boustrophédonique (b). On montre en (c) un exemple de balayage dédoublé avec une direction de départ diagonale. Ces trois exemples utilisent le mode de déplacement rectiligne.*

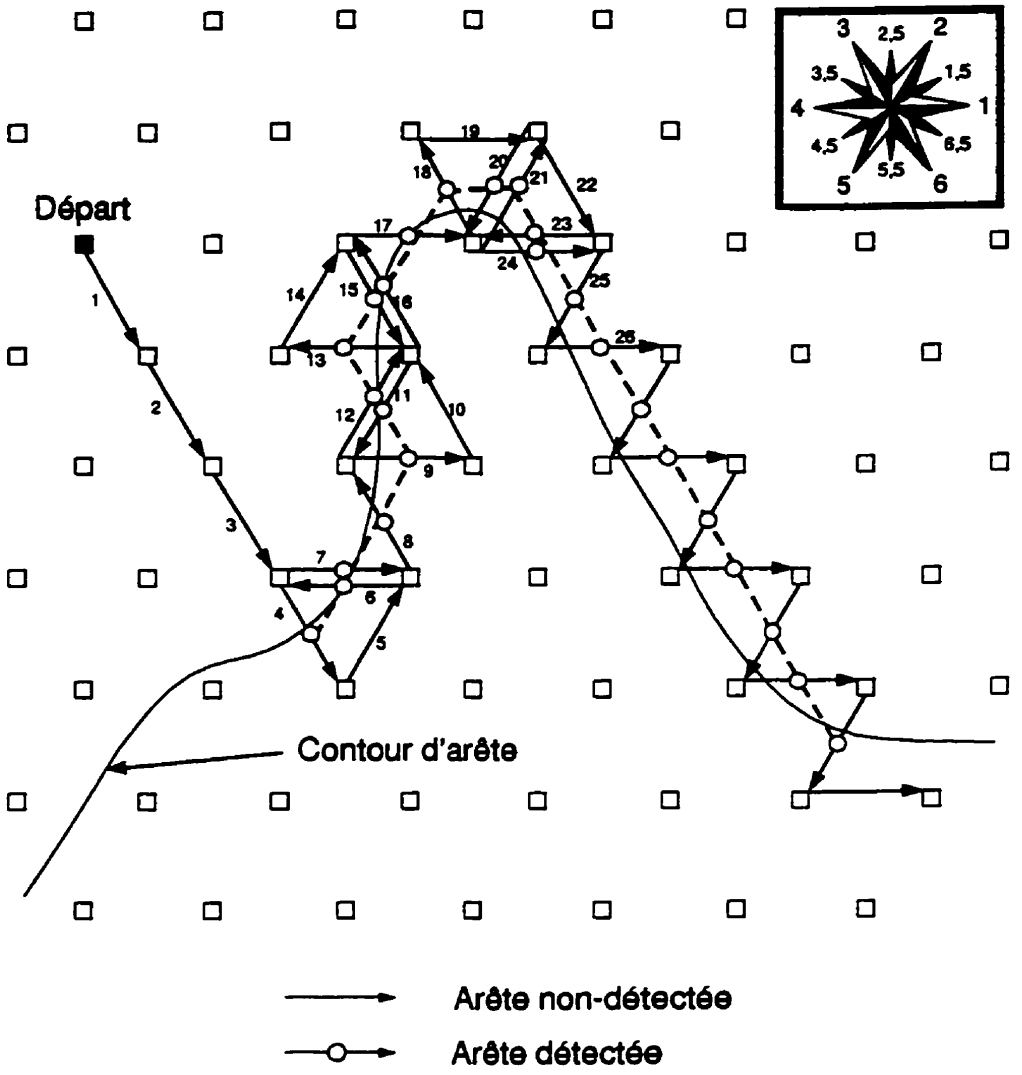


Figure 4.11 *Exécution de l'instruction de la poursuite d'une arête. L'algorithme débute par la recherche de la première occurrence d'arrête (4) puis entreprend la poursuite vers la gauche (pour cet exemple) (les déplacements 5 et 6). Le suivi s'effectue en se déplaçant en mode zigzag de part et d'autre de l'arête (7,8,9). Lorsque l'arête est perdue (10), l'algorithme ferme le triangle (11) puis poursuit dans une nouvelle direction (12,13).*

L'algorithme de poursuite d'arêtes comprend une procédure d'exception qui tente de continuer la poursuite lorsque le signal d'arêtes est localement bruité. Cette procédure est définie comme étant une poursuite d'arête en mode aveugle. On présente à la Figure 4.12 un exemple de suivi d'arêtes dont l'exécution est complétée en mode aveugle dans une

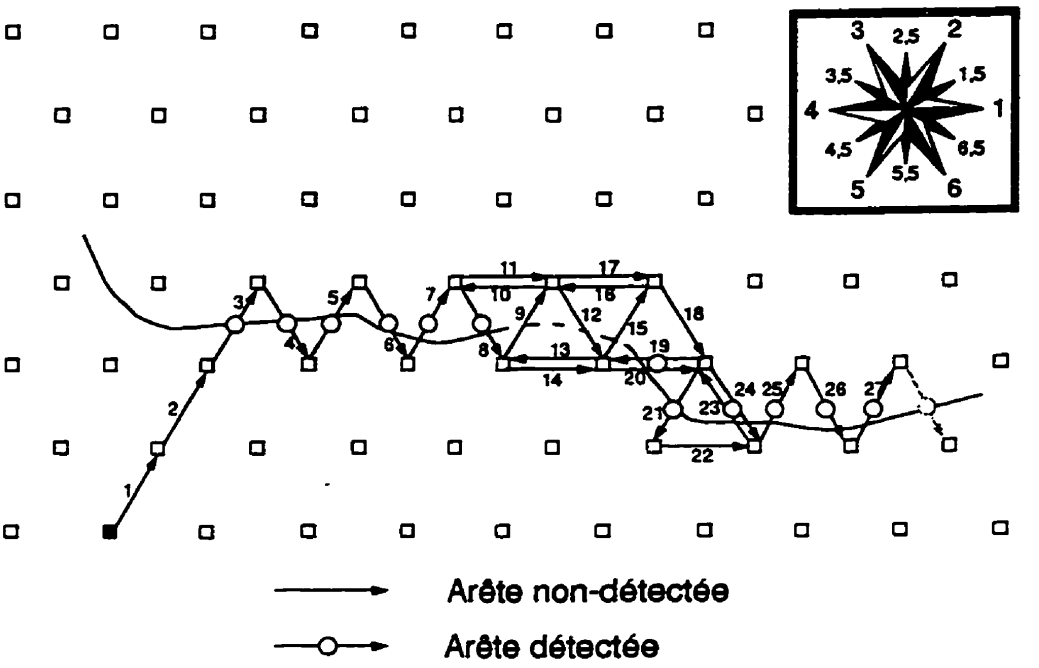


Figure 4.12 Exemple de poursuite d'arêtes en mode aveugle. La poursuite débute normalement (1-8) puis la détection est perdue en (9). On forme normalement le triangle (10) mais comme l'arête n'est pas détectée dû à un signal bruité, on poursuit en mode aveugle (11-18) dans la dernière direction valide jusqu'à ce qu'on détecte l'arête à nouveau (19). L'algorithme poursuit alors normalement (20-27...). Le nombre d'itérations en mode aveugle est limité par le compteur d'événements E.

région bruitée. Lorsqu'une discontinuité d'arête survient et que l'arête n'est pas détectée après avoir fermé le triangle (déplacement 10 de la Figure 4.12), on continue la recherche dans la dernière direction de poursuite d'arête valide (déplacements 11 à 19 de la Figure 4.12). La prochaine détection d'arête implique le retour à l'exécution normale de l'instruction. On doit évidemment utiliser le mode aveugle avec prudence puisqu'il peut provoquer un saut à une autre arête, particulièrement dans les zones de jonction. Pour cette raison, on utilise le compteur d'événement E pour programmer le maximum d'itérations permises en mode aveugle. Dans le cas où le mode aveugle est inutilisé (E=0), l'instruction de recherche d'arêtes se termine immédiatement après la perte de détection (déplacement #10 de la Figure 4.12). Il est important de noter que le retour au mode normal de recherche, qui survient lorsqu'on détecte à nouveau le signal d'arête, peut se produire sur un des côtés (au déplacement #19 de la Figure 4.12), ou dans la même direction que la dernière direction valide (par exemple le déplacement #18 de la Figure 4.12).

Les exemples de la Figure 4.11 et de la Figure 4.12 débutent par la recherche d'une première occurrence d'arête. Ceci correspond à un mode d'opération programmable par le champ option. Un autre mode nommé *IMMEDIATE*, impose que le premier déplacement doit se faire sur une arête. Dans ce cas, le déplacement #4 à la Figure 4.11 ou le déplacement #3 à la Figure 4.12 correspond à la condition de départ du mode *IMMEDIATE*.

En plus des différentes options présentées aux paragraphes précédents, l'instruction de poursuite d'arêtes peut évoluer de deux façons différentes. Lorsque survient une discontinuité dans la détection de l'arête, la direction de recherche peut être incrémentée par pas de 30° ou par pas de 60°. Les exemples présentés à la Figure 4.11 et à la Figure 4.12 utilisent le mode de recherche par pas de 60°. La Figure 4.13 montre deux portions d'arêtes identiques dont la poursuite est effectuée selon ces deux modes. On remarque à la Figure 4.13 (a) que la partie d'arête entre les points 21 et 24 fait l'objet d'une recherche continue, que la position interprétée de l'arête est plus réaliste et que l'orientation de cette portion d'arête est constante à 1,5. La même région est traitée différemment dans le cas du mode de recherche par pas de 60°. On remarque dans ce cas que les discontinuités d'arêtes sont nombreuses (entre les points *P* à *Z*) et que l'orientation d'arête oscille entre les directions 1 et 2.

Le mode de recherche par pas de 30° implique cependant une série de déplacements inutiles lorsque le changement d'orientation de l'arête est brusque comme c'est le cas pour une géométrie aiguë (représentée entre les points 4 et 14). Dans ce cas, l'algorithme modifie son orientation de recherche de la direction 5 (points 1 à 3) à la direction 5,5 (verticale vers le bas aux déplacements #6 et #7). L'algorithme poursuit dans la direction 6 (déplacement #9), puis dans la direction 6,5 (déplacements #11 et #12) pour finalement retrouver l'arête dans la direction 1 (aux points 15 à 18). Le mode de recherche par pas de 30° peut poser un problème si la densité des arêtes est grande. La recherche peut sauter d'une arête à une autre si, par exemple, une autre arête était présente lors du déplacement #7 de la Figure 4.13 (a). Pour cette raison, ce mode de recherche d'arêtes est prohibé pour les images d'arêtes provenant d'un filtrage fin. On sait de toute façon que les filtrages grossiers arrondissent les géométries aiguës et que l'intervalle entre deux arêtes consécutives est limité par la bande passante du filtre.

La détection de discontinuité de la direction de poursuite d'arêtes est incluse à même l'instruction de poursuite d'arêtes. L'occurrence d'un changement d'orientation de l'arête active une sortie de PLA nommée *D* (pour discontinuité) en plus d'indiquer le sens de cette discontinuité *SD* qui indique que la nouvelle orientation de la poursuite se fait vers

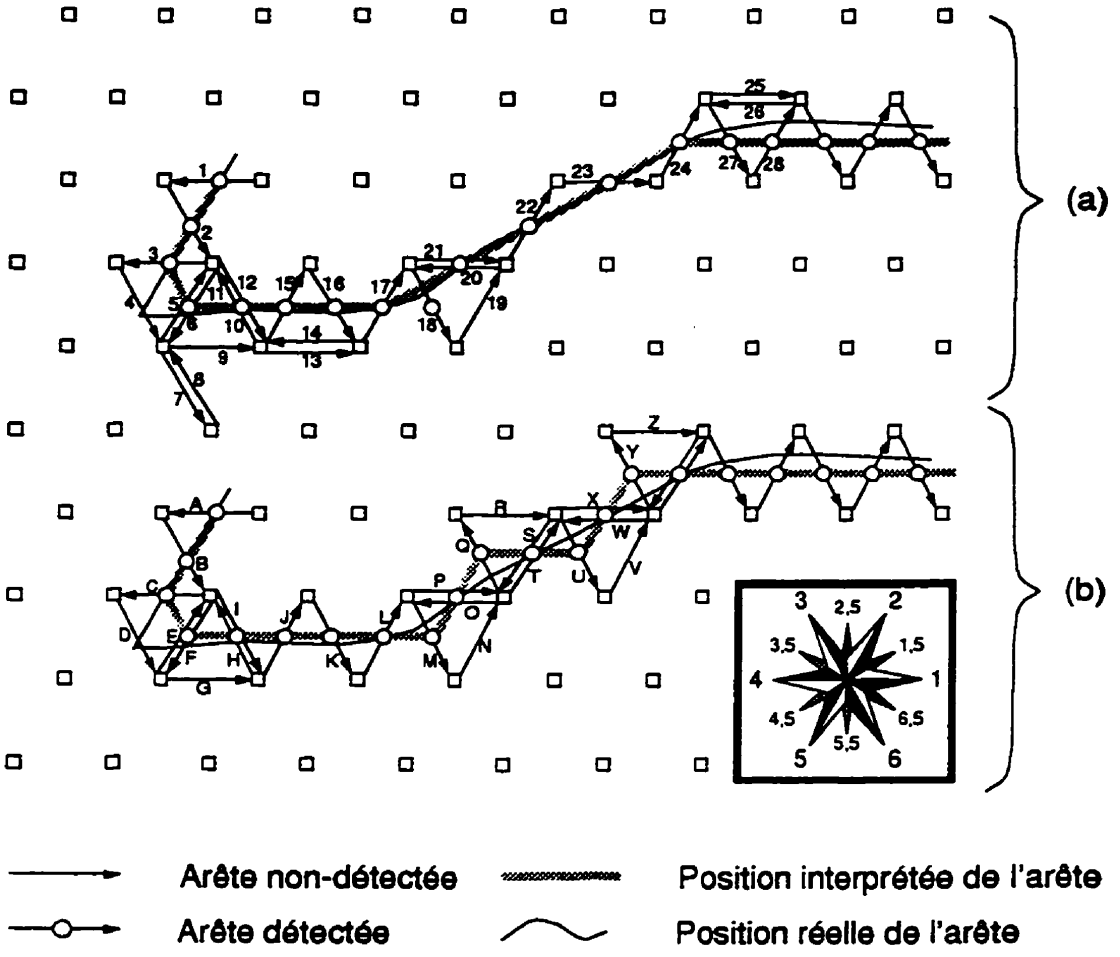


Figure 4.13 Comparaison entre l'instruction de poursuite d'arêtes par pas de 30 degrés (a) et par pas de 60 degrés (b). Les changements brusques de l'orientation de l'arête impliquent un nombre important de déplacements inutiles (4 à 14) pour le mode à 30 degrés. Par compte, les diagonales douces sont détectées sans discontinuité d'arête (20 à 23) alors que le mode à 60 degrés identifie plusieurs discontinuités d'arête pour ces diagonales (N à Z).

la gauche (*SD=1*) ou vers la droite (*SD=0*). Par exemple, à la Figure 4.13 (a), les déplacement #21 et #27 impliquent l'activation du signal *D* avec *SD=1* et *SD=0* respectivement. Dans le cas de la Figure 4.13 (b), le signal de discontinuité d'arête est activé aux déplacements *P* et *X* avec *DS=1* et au déplacement 'T' avec *DS=0*. Cette information génère une interruption au système hôte en ne lui signalant que les points de discontinuités d'arêtes. Ces deux bits d'information servent également à identifier des

discontinuités qui sont détectées à une cadence régulière et dont le sens de discontinuité alterne lors de la poursuite d'une arête oblique. On peut faire une analogie entre ce genre de séquence régulière de discontinuité et le déplacement généralisé montré à la Figure 4.6. C'est d'ailleurs le même module qui réalise ces deux fonctions et qui est décrit à la section 4.3.11.

On termine cette section en précisant que l'algorithme de recherche d'arêtes s'applique à toute image binaire en définissant le changement d'état comme étant une arête lors du passage d'une région blanche (bit=1) à une région noire (bit=0). Le compteur d'événements *N* fixe le nombre maximal de points d'arêtes détectés qu'on veut suivre avant que l'instruction ne se termine. C'est en quelque sorte une limite sur la longueur de l'arête dont on veut faire la poursuite. Puisque les lieux d'arête forment généralement un contour fermé, un circuit spécialisé du contrôleur conserve en mémoire le point de départ de la recherche afin de détecter le retour au point d'origine du pixel d'intérêt.

4.3 Description détaillée du contrôleur MAR

Après avoir fait la description fonctionnelle du contrôleur MAR et présenté le jeu d'instructions commandant le déplacement du pixel d'intérêt selon plusieurs modes, on présente dans cette section la description détaillée de ses différentes composantes. On rappelle que le contrôleur est un module intégré en un seul circuit VLSI de 68 broches. On discute dans cette section l'arbitration de l'accès au registre d'instructions, le détail de la réalisation des compteurs d'événements, le module d'interruption des instructions, le générateur d'adresses pour la mémoire externe et la définition de la fenêtre d'étude. On poursuit la description détaillée du module d'exécution conditionnelle, de l'unité arithmétique de direction, de l'évaluation de l'orientation d'arête, de la détection des passages par zéro d'un signal analogique, du sélecteur de données et de l'extraction des sections rectilignes d'arêtes. On présente finalement le module de détection du retour au point de départ, l'archivage de l'information pertinente, et les registres de configuration.

4.3.1 Arbitration des instructions

L'exécution d'une instruction débute immédiatement après le cycle d'écriture au registre d'instructions. Pour cette raison, on doit d'abord programmer l'ensemble des registres de configuration relatifs à celle-ci puis, programmer l'instruction en question. Pour ne pas lier la fréquence d'opération du système hôte à celle du système MAR, le transfert des données s'exécute de manière asynchrone entre ces deux unités. Les deux systèmes peuvent donc utiliser des horloges de base de fréquences différentes. Le module

d'arbitration des instructions régit le déclenchement de l'exécution en synchronisme avec l'horloge du système MAR.

Comme on peut le voir à la Figure 4.14 le module d'arbitration des instructions reçoit les entrées suivantes: l'horloge du système MAR (*H*), le signal d'écriture au registre d'instructions (*W*) et le signal annonçant la fin de l'instruction précédente ou le début de la prochaine (*N*). Ce circuit séquentiel génère deux signaux internes de contrôle soit le signal synchrone d'activation du registre tampon (*Ready*) et le signal qui force des cycles d'attente lorsqu'aucune instruction n'est prête à être exécutée (*Force_NOP*). Le signal *F* est combiné à la condition qui définit la frontière de l'image (*CF*) pour activer un signal commun de condition d'exécution (*C*). On utilise un ligne commune simplement pour limiter le nombre d'entrées du PLA car les deux signaux *F* et *CF* sont utilisés à des moments différents. En effet, le signal *F* n'est activé que lorsque le contrôleur est en attente d'une nouvelle instruction (*N=1*) alors que l'activation du signal *CF* ne survient que lorsque le pixel d'intérêt atteint la frontière lors de l'exécution d'une instruction.

Puisque le signal de condition qui définit la frontière joue un rôle dans le processus de requête de cycles d'attente, l'exécution d'une instruction ne peut pas débuter lorsque le pixel d'intérêt réside sur une frontière (*CF=1*). Dans ce cas particulier, on doit forcer un déplacement inconditionnel du pixel d'intérêt vers l'intérieur de la frontière avant de poursuivre toute exécution conditionnelle. L'explication précédente donne la raison pour laquelle toutes les instructions de déplacement du système MAR sont programmées de façon à éviter qu'une instruction se termine alors que le pixel d'intérêt réside sur la frontière.

La Figure 4.14 montre un diagramme temporel de l'exécution de trois instructions différentes. Le nombre de cycles d'horloge pour exécuter les instructions *A*, *B* et *C* est 1, 6 et 1 respectivement. Le registre d'instructions comprend une mémoire tampon pour le chargement synchrone des commandes utilisées par le PLA. Ce registre supplémentaire donne la possibilité au système hôte de programmer la prochaine instruction avant la fin de l'instruction en cours d'exécution comme c'est le cas pour la programmation de l'instruction *C* durant l'exécution de l'instruction *B*. On a déjà mentionné qu'on doit programmer l'ensemble des registres de configuration avant le registre d'instructions. La programmation anticipée du registre d'instructions implique l'utilisation de la même configuration que pour l'instruction *B*. Le cas échéant, on doit attendre le fin de l'instruction courante avant de modifier les registres de configuration à moins que ces modifications ne gênent pas l'exécution de l'instruction courante (comme ce serait le cas si l'instruction *B*

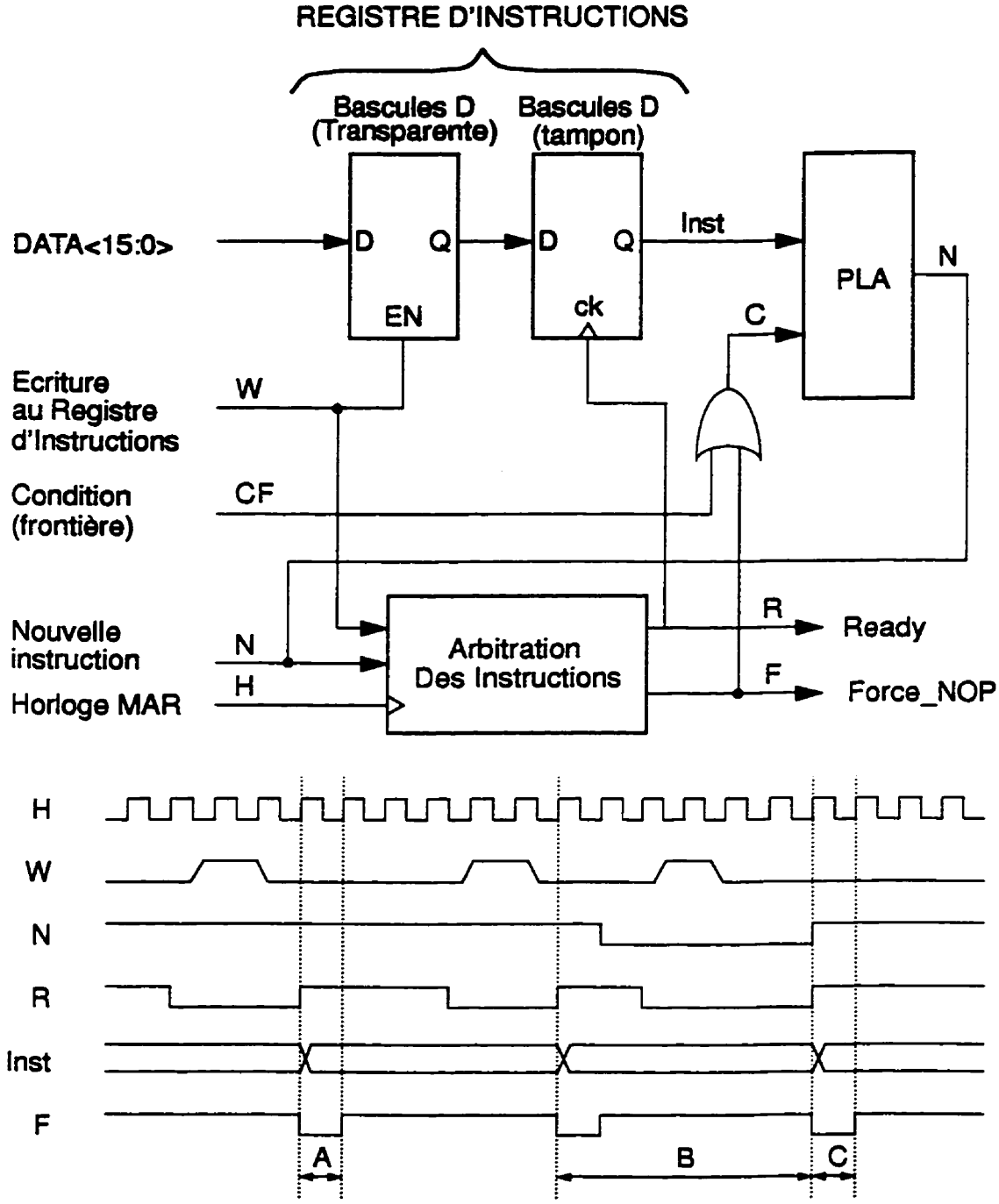


Figure 4.14 Module d'arbitration des instructions. La commande d'écriture (W) au registre d'instructions apparent se fait de façon asynchrone par rapport à l'horloge (H) du système MAR. Le signal de fin d'instruction (N) active le signal Ready (R) et implique un transfert synchrone de la prochaine instruction à travers la cache si, à ce moment, une instruction a été programmée. Le signal Force-NOP (F) commande des cycles d'attente lorsqu'aucune instruction n'est programmée (comme après l'instruction "A"). Le signal Ready indique au système hôte la fin de l'instruction précédente.

ne fait que déplacer inconditionnellement le pixel d'intérêt d'une région de l'image à une autre).

La Figure 4.15 donne le détail électronique du module d'arbitration des instructions générant les signaux *Ready* (*R*) et *Force_NOP* (*F*). Le signal *Ready*, qui commande un transfert du registre d'instructions aux entrées du PLA, est activé lorsque l'instruction précédente est terminée (*N=1*). Il est remis à zéro de façon asynchrone lorsque le système hôte programme la prochaine instruction (*W=1*). Puisque le cycle d'écriture est régi par le système hôte et que sa duré est inconnue, il peut s'écouler un ou plusieurs cycles d'horloge du système MAR (*H*) durant cette écriture au registre d'instructions. Un signal d'inhibition est donc activé durant cette période et force des cycles d'attente (*F=1*) dans le cas où

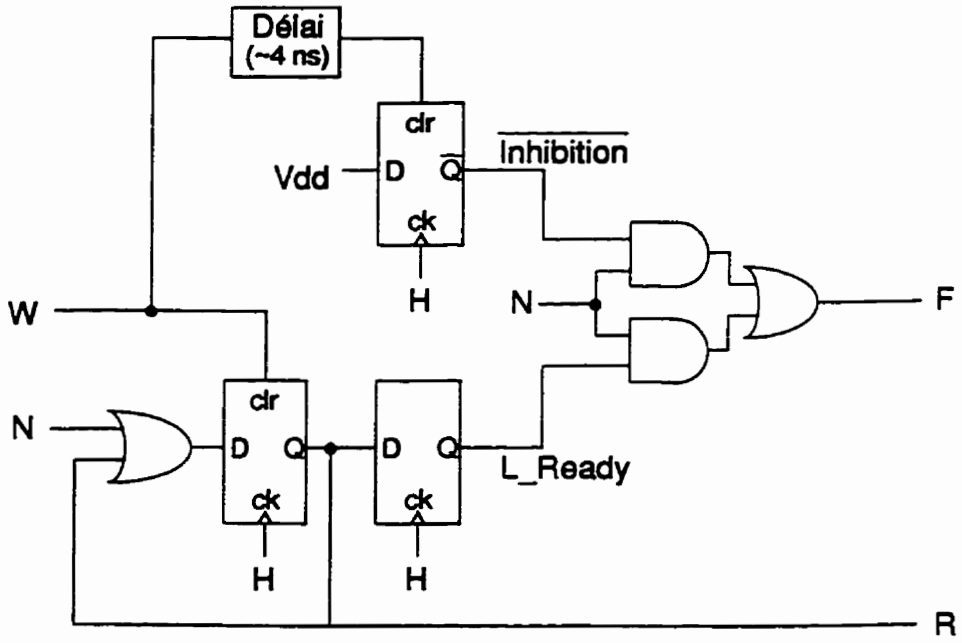


***Figure 4.15** Détail logique du module d'arbitration des instructions. Le signal Ready (R) est mis à zéro de façon asynchrone lors de l'écriture (W) au registre d'instructions et ne peut être activé que de façon synchrone au prochain cycle d'horloge (avec W=0). Durant toute la duré de l'écriture au registre d'instructions par le système hôte, le signal d'inhibition force des cycles d'attente (F=1) si l'instruction précédente est terminée (N=1).*

l'instruction précédente est terminée (*N=1*). On attend donc la fin du cycle d'écriture pour

entreprendre le début de l'exécution d'une nouvelle instruction. Le délai imposé au signal *W* pour l'activation du signal d'inhibition est prévu pour garantir un cycle d'attente supplémentaire lorsque le signal d'écriture *W* est désactivé au même moment où survient la transition montante de l'horloge du système MAR (*H*). On s'assure ainsi que le signal *Ready* a bien changé d'état (*R:0->1*) lorsque l'exécution de la nouvelle instruction (*F=0*) est commandée.

4.3.2 Contrôle de boucles externes au PLA

La plupart des instructions offrent des options référant à deux compteurs d'événements: *N* (16 bits) et *E* (8 bits). Ces deux compteurs oeuvrent indépendamment dans le cas des instructions *MOVE_N*, *SCAN_AREA* et *FIND* mais sont dépendants l'un de l'autre pour les instructions *MOVE_TRI_N* et *MOVE_HEX_N*. Les commandes d'incrémentation et de remise à zéro de ces compteurs sont exécutées par quatre sorties distinctes du PLA: *RESET_N*, *EN_N*, *RESET_E* et *EN_E* (sorties #15, 16, 17 et 18 du PLA (voir Tableau B.2)). La tâche du système hôte consiste à programmer les valeurs limites de ces compteurs d'événements en fonction des paramètres désirés pour l'exécution d'une instruction. Le dépassement de cette limite est signalé au PLA pour chaque compteur par un signal de débordement: *N_ZERO* et *E_ZERO* (entrées #5 et 3 du Tableau B.1). Le système hôte peut également lire l'état des compteurs, par exemple lors d'une requête d'interruption.

L'architecture des compteurs d'événements de même qu'un diagramme temporel montrant un exemple typique de leur utilisation sont présentés à la Figure 4.16. La programmation de la valeur limite (*W*) et la lecture de l'état d'un compteur (*L*) sont asynchrones via le bus de données du système hôte (*D*). Les signaux de remise à zéro (*R*) et d'incrémentation des compteurs d'événements (*E*) contrôlent, en synchronisme avec l'horloge du système MAR (*H*), les changements d'état interne. Le signal de débordement (*Z*) détecte quant à lui la fin du cycle du compteur.

L'exemple de la Figure 4.16 montre le cas où le système hôte utilise le compteur N pour détecter la cinquième occurrence d'un événement. Puisque le compteur est de type incrémental, on le charge avec le complément vrai de la valeur limite préalablement programmée (FFFA) lorsque le signal *R* est actif pour ensuite annoncer la fin du compte (FFFF) à l'aide du signal *Z*. La table d'adressage des différents registres de configuration incluant les compteurs d'événement est donnée à l'ANNEXE D (Figure D.1).

4.3.3 Générateur de demandes d'interruptions et auto-arrêt d'exécution

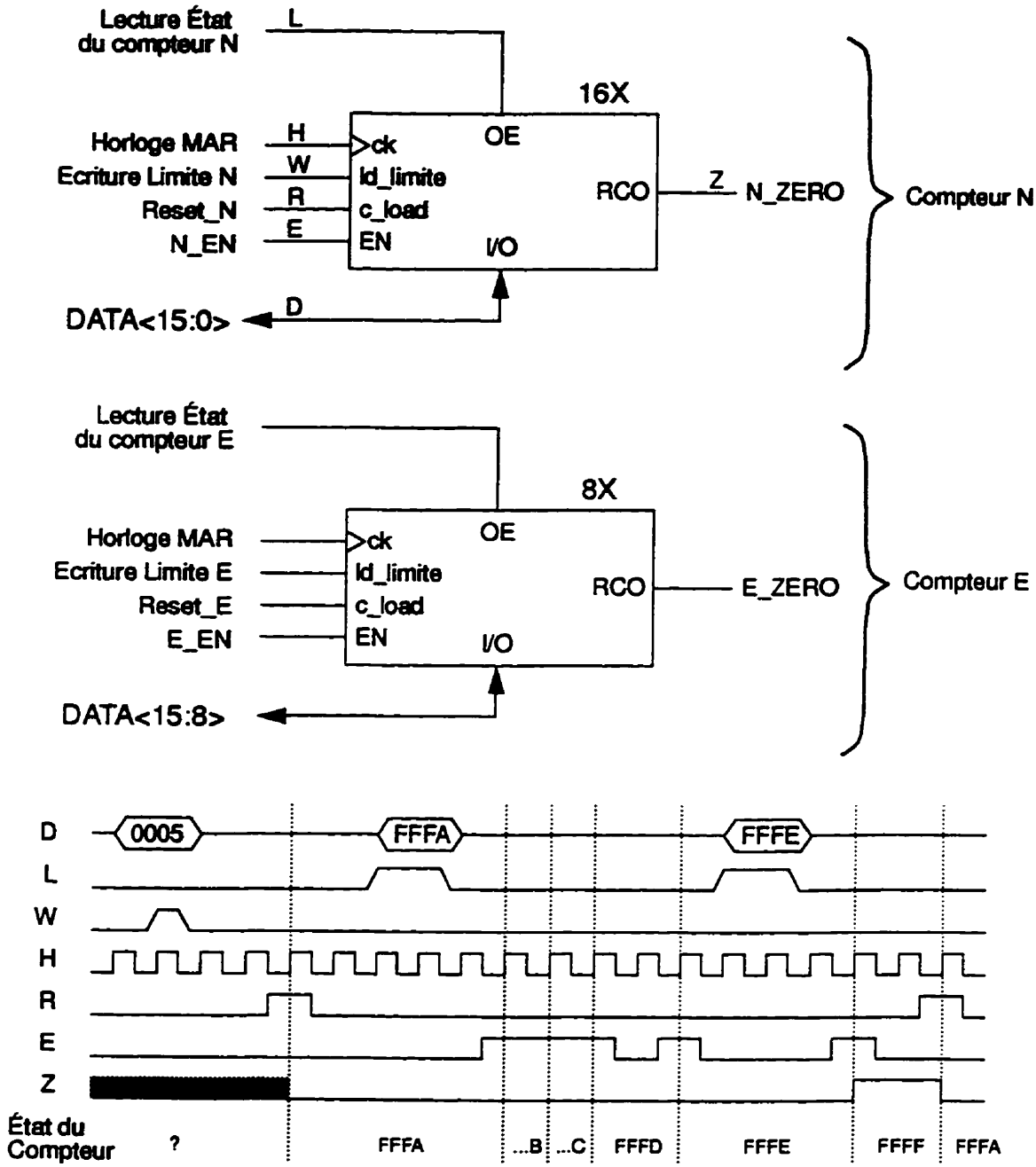


Figure 4.16 Fonctionnement des compteurs d'événements utilisés par le PLA. Le système hôte écrit à l'adresse du compteur (W=1) la valeur limite (5) du compteur d'événement. Il peut également lire l'état interne du compteur en effectuant un accès (L=1) à la même adresse. Chaque compteur est incrémental de type synchrone. Lorsque le signal Reset est actif (R=1), le compteur charge le complément vrai de la valeur limite préalablement emmagasinée dans un registre (FFFA). Le signal de débordement est activé (Z=1) après cinq commandes d'incrémentation (E=1).

Le contrôleur MAR comprend trois broches qui peuvent être programmées pour signaler au système hôte une requête d'interruption (broches I/O<7:5> représentées à la Figure 4.1). Chaque signal de requête d'interruption est programmable individuellement et peut être activé par un signal interne unique ou par une combinaison de plusieurs signaux. Le sélecteur de données présenté en détail à la section 4.3.10 effectue un *OU* logique parmi un sous-ensemble de 12 différentes variables internes du système MAR. La liste détaillée des variables internes commandant chacune des trois lignes d'interruption est présentée en ANNEXE D (Figure D.3, Tableau D.6 et Tableau D.7). On peut par exemple programmer la ligne I/O<7> pour qu'elle génère un demande d'interruption lorsque:

- *E_EN* == 1;
- *E_EN* == 1 OU *Arête détectée* == 1 OU *Discontinuité d'arête* == 1;
- *LONG_Discontinuité d'arête* == 1 OU *E_EN* = 1;
- etc...

Les variables programmables pour générer une requête d'interruption peuvent être choisies de façon à pouvoir détecter un vaste ensemble de cas particuliers. Certaines variables sont susceptibles de générer des interruptions uniquement pour synchroniser le transfert de l'information ou pour contrôler la séquence d'exécution du programme d'exploitation, comme la fin de l'exécution d'une instruction ou le retour du pixel d'intérêt au point de départ.

Un bloc de logique combinatoire jumelé à une bascule spéciale définit l'état d'un signal d'arrêt nommé *BREAK* qui interrompt temporairement l'exécution d'une instruction lors d'une requête d'interruption du système MAR vers le système hôte. Cette condition d'auto-arrêt d'exécution peut être désactivée pour chacun des trois signaux d'interruption afin de signaler uniquement l'occurrence d'une condition puis de poursuivre immédiatement l'instruction. De façon générale, lors de l'activation d'un signal de requête d'interruption, le système hôte doit lire l'état de certains registres du contrôleur MAR avant de commander la poursuite de l'instruction courante.

La Figure 4.17 présente le détail électronique de la réalisation du circuit de génération du signal d'auto-arrêt d'exécution *BREAK*. Le signal *BREAK* est activé dès qu'un signal de requête d'interruption est présent. Chaque ligne d'interruption peut être masquée pour inhiber le mode d'auto-arrêt grâce à trois variables de configuration *INT_BREAK_EN<2:0>*. Le trait sous le nom de ces variables à la Figure 4.17 signifie qu'elles proviennent d'un des registres statiques de configuration présentés à la section 4.3.14. Cette convention sera respectée tout au long du texte décrivant l'architecture

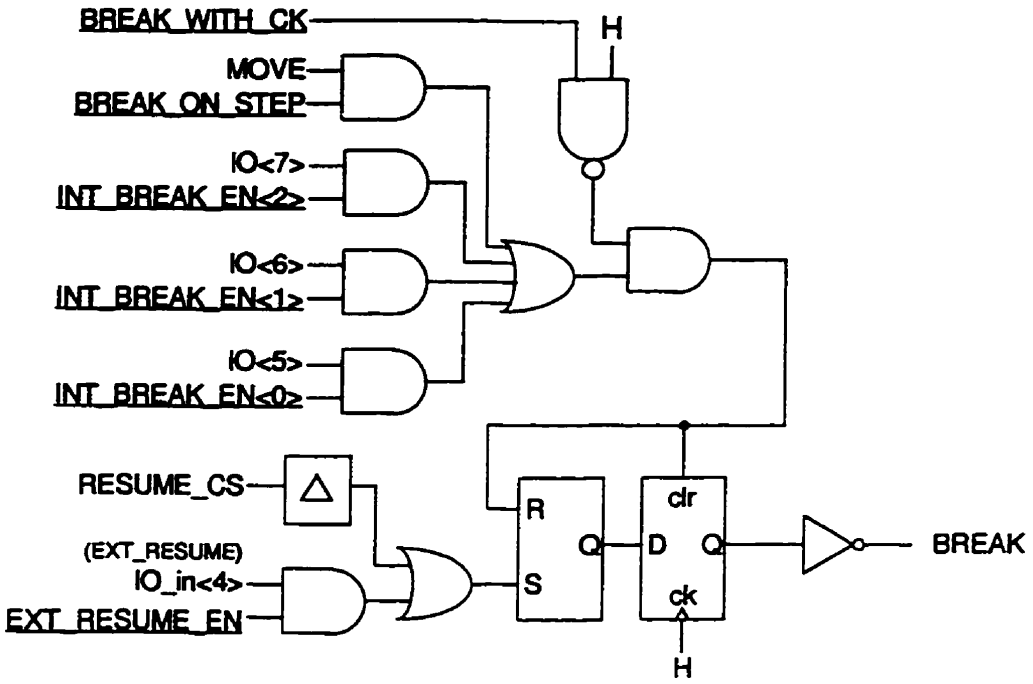


Figure 4.17 Génération du signal d'auto-arrêt "BREAK". Puisque ce signal influence les entrées du PLA, il doit changer d'état en synchronisme avec l'horloge du système MAR (H). Puisque les signaux de requêtes d'interruption sont synchrones, on les utilise pour activer le signal "BREAK". Lorsque le système hôte écrit à l'adresse "RESUME", la bascule SR change d'état et le signal "BREAK" est désactivé sur la prochaine monté de l'horloge (H).

détaillée du contrôleur. On peut également configurer le système pour qu'il exécute une instruction pas à pas lorsque la variable *BREAK_ON_STEP* est active. Pour contrer le cas où le signal d'interruption n'est pas synchrone, on peut activer la variable de configuration *BREAK_WITH_CK* et ainsi masquer les signaux d'interruption en dehors du demi-cycle où *H=1*.

Pour relancer l'exécution de l'instruction on désactive le signal *BREAK* en effectuant une écriture à l'adresse *RESUME* ou en utilisant un signal externe via le canal *IO<4>* préalablement configuré en entrée. Dans le cas où la remise à zéro de la bascule *SR* est effectué par une écriture à l'adresse *RESUME*, cette action est synchronisée avec *H* grâce à la bascule *SR* suivie d'une bascule *D* synchrone. Ainsi, le signal *BREAK* sera désactivé au prochain coup d'horloge (*H*). On ajoute un délai de quelques nanosecondes à

la fin du cycle d'écriture afin de s'assurer que le bus partagé *D<15:0>* est libéré par le système hôte avant de relancer le fonctionnement actif du contrôleur.

Tel que mentionné à la section 3.4.2, on utilise un compteur d'événements similaire au compteur E de la Figure 4.16 pour générer une boucle d'attente après chaque déplacement du pixel d'intérêt afin de stabiliser les signaux dérivés de la partie analogique. Comme on peut le voir à la Figure 4.18, le compteur C compte sans arrêt sur une période

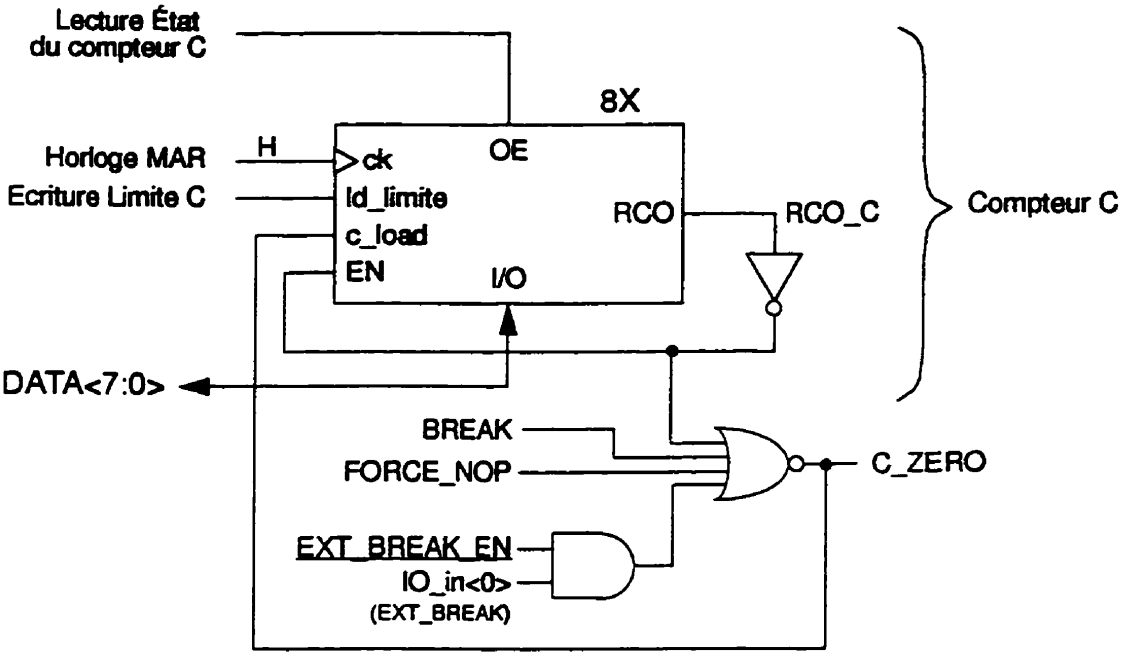


Figure 4.18 Compteur de cycles de stabilisation des signaux analogiques "C" et description de la logique générant des boucles d'attentes virtuelles dans le cas où le signal "BREAK" est actif. En mode d'opération normal, le compteur compte sans arrêt avec une période définie par la valeur limite préalablement programmée par le système hôte.

définie par la valeur limite lorsqu'aucun des signaux d'inhibition n'est actif. Les signaux *BREAK*, *FORCE_NOP* et *EXT_BREAK* forcent l'exécution d'une boucle sans fin tant qu'au moins un des trois signaux est actif. Le signal *EXT_BREAK*, qui correspond en fait à la ligne *IO<0>* configurée en mode entrée, peut servir à un module externe au contrôleur MAR pour interrompre la séquence normale des opérations. Pour utiliser ce mode externe d'arrêt temporaire d'exécution, on doit activer la variable de configuration *EXT_BREAK_EN*.

On peut également forcer une instruction à se terminer avant l'occurrence des conditions normales de la fin d'une instruction. Pour avorter l'exécution d'une instruction, le signal d'entrée du PLA nommé *INST_INTR (entrée #8 du PLA (Tableau B.1))* doit être activé. La Figure 4.19 montre le circuit qui génère ce signal. Une instruction peut être stoppée en écrivant à l'adresse *STOP_CS* ou en activant le signal *IO_in<3>* préalablement configurée en entrée (et *EXT_STOP_EN*=1). Cette possibilité d'avorter une instruction devrait être exploitée typiquement au retour d'une interruption conséquemment à la décision du programme d'exploitation. Après cette action, le contrôleur revient automatiquement en attente d'une nouvelle instruction.

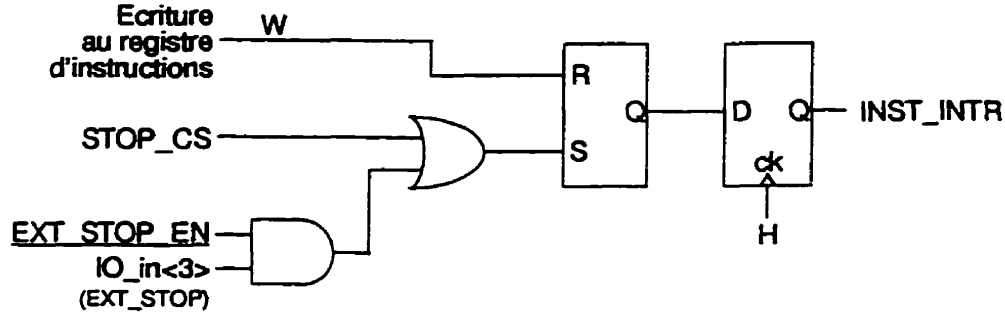


Figure 4.19 Circuit générateur du signal qui permet d'avorter l'exécution d'une instruction. On peut utiliser l'adresse STOP_CS ou un signal externe pour terminer prématurément l'exécution d'une instruction. Le signal INST_INTR est remis à zéro lors du chargement de l'instruction suivante. La bascule D assure un changement d'état synchrone.

4.3.4 Registres de position et pointeur de mémoire MAR

Le contrôleur de caméra utilise les commandes de direction fournies au capteur MAR pour mettre à jour l'indice de position X et Y du pixel d'intérêt dans l'image. Cette fonction est implantée grâce à deux compteurs synchrones bidirectionnels avec remise à zéro synchrone. Ces compteurs sont initialisés lors de l'exécution de l'instruction *RESET_POSITION* (section 4.2.3.1). L'indice de position sert de pointeur de mémoire MAR conformément à ce qui a été discuté à la section 3.2.3. La Figure 4.20 résume la réalisation du pointeur de mémoire MAR de même que celle du générateur des coordonnées de la position du pixel d'intérêt. Les deux modules de calcul de la position du

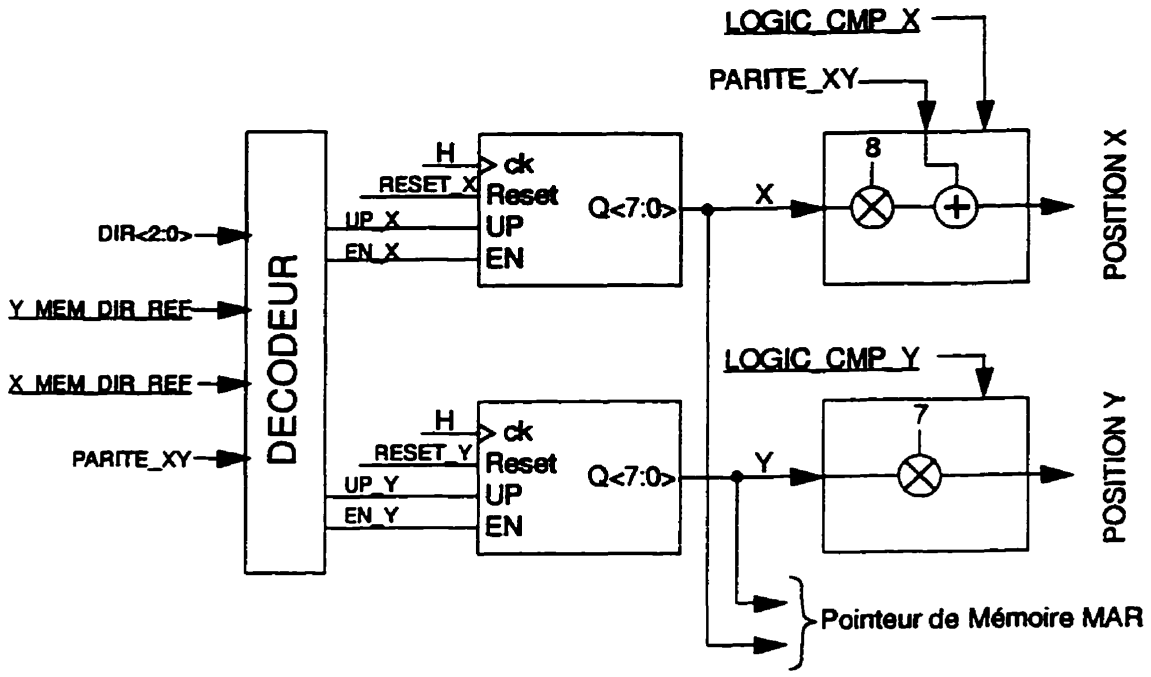


Figure 4.20 *Pointeur de mémoire MAR et générateur de position. Les valeurs des deux compteurs X et Y sont concaténées pour définir un pointeur de 16 bits qui adresse la mémoire MAR. Les deux variables de configuration "X_MEN_DIR_REF" et "Y_MEM_DIR_REF" servent à indiquer quel type d'arithmétique utiliser pour le calcul de la position. Deux modules de logique combinatoire calculent les coordonnées de la position X et Y du pixel d'intérêt.*

pixel d'intérêt sont construits conformément à la discussion de la section 3.2.3 (équations (3-16) et (3-17)).

Le type d'arithmétique binaire utilisée pour effectuer le calcul de la position du pixel d'intérêt est importante. Les deux variables de configuration *LOGIC_CMP_Y* et *LOGIC_CMP_X* servent à le définir. Ce choix est directement lié à la position de l'origine du référentiel image lors de l'exécution de la commande *RESET_POSITION*. Si la référence verticale est au centre du capteur, l'interprétation de l'indice de position Y doit être faite en complément de deux et la variable *LOGIC_CMP_Y* doit être désactivée. Par contre, si l'origine du référentiel image est choisie à l'un des coins, la représentation des positions est alors non-signée et on doit le signaler en activant les deux variable de configuration. Cette propriété est importante car les registres de position, qui ont une largeur de 16 bits pour la représentation de nombres de 11bits, répètent le bit de signe sur les bits les plus significatifs du registre lorsqu'on évalue la position en complément à deux.

On utilise un décodeur pour contrôler les deux compteurs synchrones. Le code de déplacement *DIR<2:0>* (d_2, d_1, d_0), la parité de la ligne courante *PARITE_XY* (P) de même que deux variables de configuration associées à la polarité des axes du référentiel image *X_MEM_DIR_REF* (X_R) et *Y_MEM_DIR_REF* (Y_R), génèrent les signaux de commandes des deux compteurs. Les équations logiques (J-1) à (J-4) de l'ANNEXE J résument la logique combinatoire comprise dans ce décodeur:

4.3.5 Définition de la fenêtre d'étude

On utilise un bloc de 4 comparateurs pour définir une fenêtre d'étude qui correspond à un sous-ensemble de l'image du capteur MAR. Les coordonnées X_MAX, Y_MAX, X_MIN et Y_MIN sont des registres de configuration programmables par le système hôte. Ils limitent le traitement d'images à une région particulière de l'image. Il est possible de détecter individuellement chaque signal de débordement comme le montre la Figure 4.21. Chaque comparateur d'égalité de la Figure 4.21 est réalisé grâce à huit portes

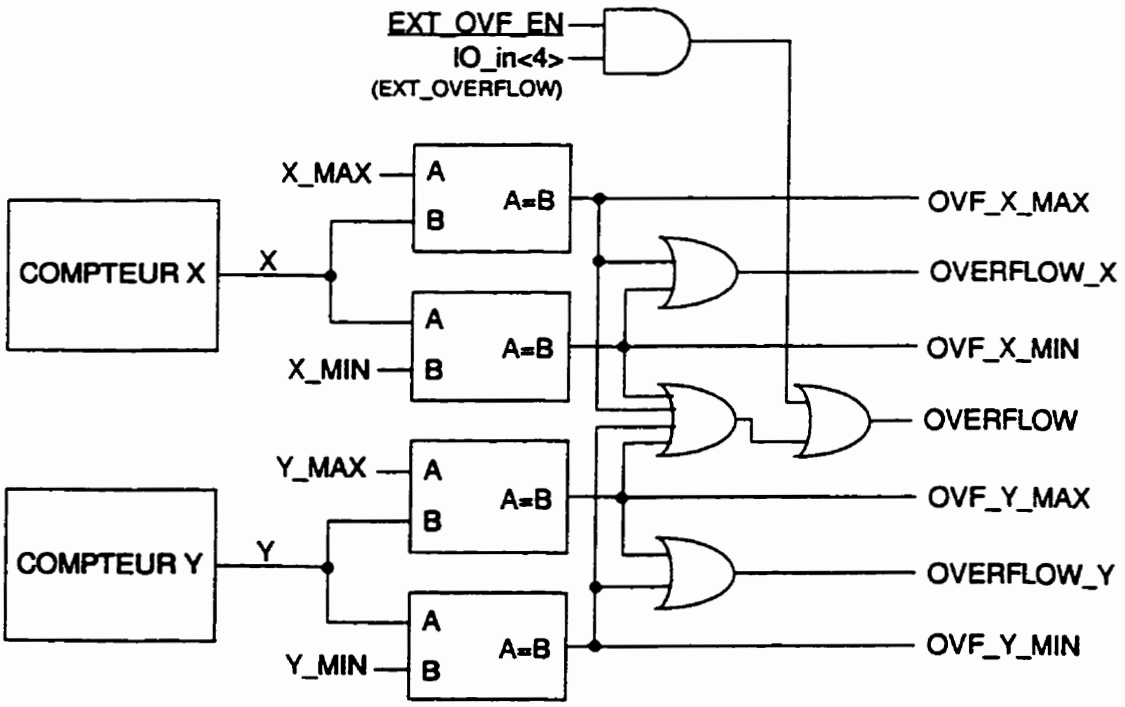


Figure 4.21 *Détection du débordement de la fenêtre d'étude par le signal "OVERFLOW". On génère également des variables intermédiaires pour identifier laquelle des limites est franchie par le pixel d'intérêt.*

OU Exclusif qui comparent un à un les bits de la variable de position à la valeur limite. On effectue une fonction *ET* entre chacun des blocs de comparaison et la sortie du module de comparaison (*A=B*) est activée uniquement lorsque l'entrée *A* est identique à l'entrée *B*. Si la frontière de l'image est atteinte lors de l'exécution de l'instruction de poursuite d'arêtes, on peut savoir rapidement quelle frontière a été franchie. Le cadre gris de la Figure 4.10 est un exemple typique de définition de la fenêtre d'étude. On remarque que les frontières verticales sont toujours de forme zigzag. Il est possible de combiner un signal externe (IO_in<4>) au signal de débordement de la fenêtre d'étude en activant la variable de configuration *EXT_OVF_EN*. On permet ainsi à un module externe de contrôler la forme et la dimension de la fenêtre d'étude.

4.3.6 Module d'exécution conditionnelle

Le champ de trois bits nommé *COND* du registre d'instructions (*INST_REG<7:5>*) choisit quelle variable, ou combinaison de variables internes, définissent la frontière. On retrouve à la section C.4.2 la description de ce code de condition. De façon générale, on peut forcer un déplacement inconditionnel en choisissant le code *0*. Si le code *1* est choisi, seules les bornes de la fenêtre d'étude serviront à déterminer la frontière. Pour les autres codes, la fenêtre d'étude est combinée à d'autres variables par une fonction OU. On peut entre autre définir la détection d'un passage par zéro combiné à la fenêtre d'étude comme frontière. Le Tableau 4.1 résume la définition des variables externes *EXT_COND<2:0>* en fonction des variables de configuration *EXT_COND_EN<2:0>*. La variable *VISITE_INT* est décrite à la section 4.3.13

***Tableau 4.1** Définition des variables externes du module de condition pour la frontière*

EXT_COND_EN<2> = 1	EXT_COND<2> = IO_in<2>
EXT_COND_EN<2> = 0	EXT_COND<2> = VISITE_INT
EXT_COND_EN<1> = 1	EXT_COND<1> = IO_in<2>
EXT_COND_EN<1> = 0	EXT_COND<1> = VISITE_INT
EXT_COND_EN<0> = 1	EXT_COND<0> = IO_in<2>
EXT_COND_EN<0> = 0	EXT_COND<0> = VISITE_INT

4.3.7 Unité arithmétique de direction

Les principales sorties de l'unité de logique programmable (PLA) servent à définir la direction du déplacement transmise au capteur MAR. La direction de déplacement (*DIR*) est calculée à partir d'une direction de base (*BASE*) et d'une valeur de déviation

(*MOD_OFFSET*) comme on peut le voir à la Figure 4.22. La direction courante (*DIR*) est

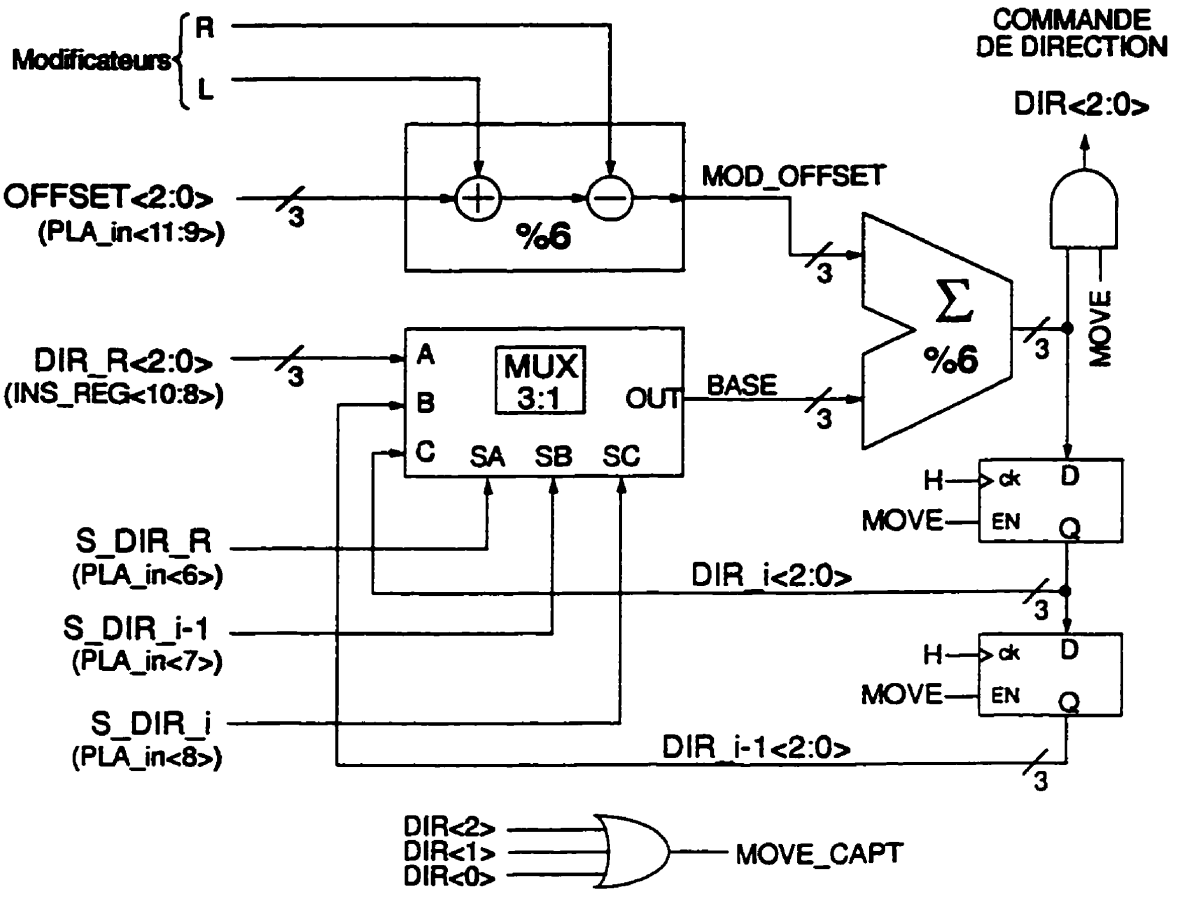


Figure 4.22 Unité Arithmétique de direction. La direction de base est sélectionnée par le PLA et est additionnée à la déviation (MOD_OFFSET) afin de définir la prochaine commande de déplacement. Les signaux de modification "L" et "R" incrémentent ou décrémentent respectivement la valeur de déviation fournie par le PLA (OFFSET). Tous les calculs sont effectués en modulo 6 pour rester conforme au code de déplacement.

mémorisée par une queue (*DIR_i* et *DIR_i-1*) et est retournée comme entrée à l'unité arithmétique de direction pour définir des déplacements relatifs à la direction précédente ou antérieure à la précédente. C'est un multiplexer trois à un qui définit le type de calcul à effectuer et c'est le PLA qui commande ce multiplexer grâce aux signaux *S_DIR_R*, *S_DIR_i* et *S_DIR_i*-1 (voir Tableau B.2). Le champ *Direction de Départ* du registre d'instructions (*DIR_R*) donne la direction du premier déplacement du pixel d'intérêt lors de l'exécution d'une instruction.

Dans le cas des déplacements relatifs, c'est la machine à état (*OFFSET*) qui fixe la direction du déplacement (en modulo 6) du pixel d'intérêt. Puisque le code de direction *0* garde le pixel d'intérêt immobile, l'arithmétique diffère légèrement de l'arithmétique en complément de deux conventionnelle. Le code de déviation (*MOD_OFFSET*) peut prendre les valeurs {*0, 1, 2, 3, 4 et 5*} pour un déplacement relatif de {*0, +1, +2, +3, -2 et -1*} respectivement. Le Tableau 4.2 donne l'arithmétique adaptée au système MAR.

***Tableau 4.2** Règles arithmétiques modulo 6 modifiées en fonction du code de direction du système MAR. Cette table de vérité est utilisée par le module d'addition %6 de la Figure 4.22.*

BASE	MOD_OFFSET	DIR	BASE	MOD_OFFSET	DIR
1	0 (+0)	1	4	0 (+0)	4
1	1 (+1)	2	4	1 (+1)	5
1	2 (+2)	3	4	2 (+2)	6
1	3 (+3)	4	4	3 (+3)	1
1	4 (-2)	5	4	4 (-2)	2
1	5 (-1)	6	4	5 (-1)	3
2	0 (+0)	2	5	0 (+0)	5
2	1 (+1)	3	5	1 (+1)	6
2	2 (+2)	4	5	2 (+2)	1
2	3 (+3)	5	5	3 (+3)	2
2	4 (-2)	6	5	4 (-2)	3
2	5 (-1)	1	5	5 (-1)	4
3	0 (+0)	3	6	0 (+0)	6
3	1 (+1)	4	6	1 (+1)	1
3	2 (+2)	5	6	2 (+2)	2
3	3 (+3)	6	6	3 (+3)	3
3	4 (-2)	1	6	4 (-2)	4
3	5 (-1)	2	6	5 (-1)	5

La Figure 4.23 donne le détail de la réalisation du module de calcul en base 6 qui est utilisé à la Figure 4.22 et qui respecte la table de vérité du Tableau 4.2. Un bloc de

logique combinatoire modifie la sortie du sommateur lorsqu'elle dépasse la valeur *6* en lui retranchant le nombre *6*. Ce bloc de logique est simplifié car on exploite le fait que la sortie du sommateur ne dépasse jamais la valeur *11*. Le circuit de la Figure 4.23 est utilisé en grande partie par le circuit de calcul de l'orientation d'arête et décrit à la section 4.3.8.

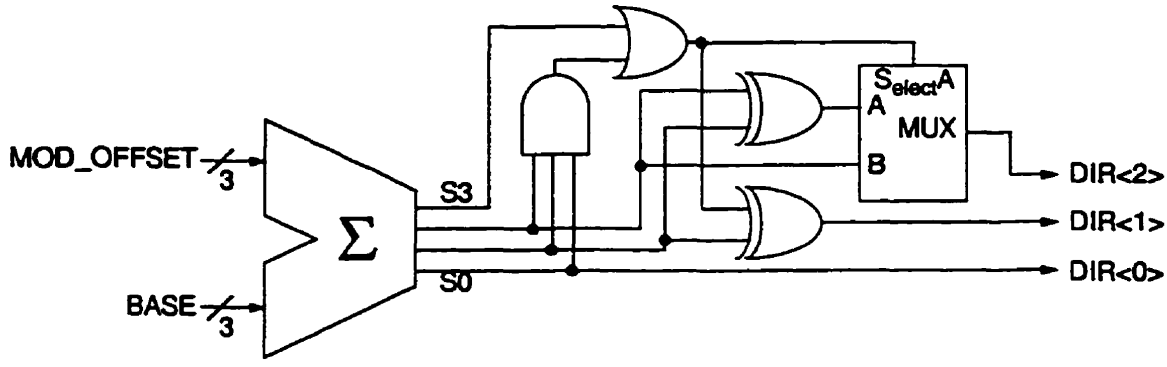


Figure 4.23 Module d'addition modulo 6 utilisé à la Figure 4.22 et qui respecte la table de vérité du Tableau 4.2. L'étendue des valeurs de la sortie du sommateur (S3, S2, S1 et S0) est limitée entre "0" et "11" puisque les plus grandes valeurs possibles aux entrées "BASE" et "MOD_OFFSET" sont "6" et "5" respectivement.

L'unité arithmétique de direction comprend un module qui modifie la valeur de déviation calculée par le PLA en soumettant cette dernière à une opération additionnelle, ce qui permet le déplacement généralisé discuté à la section 4.2.5. Le signal de modification de la direction *L* force l'incrément de la déviation originale (*OFFSET*) avant le calcul de la commande de direction modifie le déplacement prévu d'un cran vers la gauche (60°). Le signal *R* a le même effet, mais vers la droite. On ne doit évidemment pas activer simultanément les deux signaux *L* et *R*. Le Tableau 4.3 résume le comportement du module de modification de la déviation et les équations logiques (3-36) à (3-38) présentent la solution simplifiée à partir de la table de vérité du Tableau 4.3. On utilise une notation abrégée pour les variables *OFFSET* (O_2, O_1, O_0) et MOD_OFFSET (MO_2, MO_1, MO_0) afin d'alléger les équations.

Tableau 4.3 Table de vérité du module de modification de la déviation.

L	R	OFFSET	MOD_OFFSET	OFFSET	MOD_OFFSET
0	0	0	0	3	3
0	0	1	1	4	4
0	0	2	2	5	5
0	1	0	5	3	2
0	1	1	0	4	3
0	1	2	1	5	4
1	0	0	1	3	4
1	0	1	2	4	5
1	0	2	3	5	0
1	1	X	INVALIDE	X	INVALIDE

$$MO_2 = \overline{RO_2}O_1O_0L + O_2\overline{O_1}O_0\bar{L} + \bar{R}O_2\overline{O_1}\overline{O_0} + R\overline{O_2}\overline{O_1}\overline{O_0}\bar{L} \quad (3\text{-}36)$$

$$MO_1 = \overline{RO_2}\overline{O_1}O_0L + \overline{O_2}O_1O_0\bar{L} + \bar{R}\overline{O_2}O_1\overline{O_0} + RO_2\overline{O_1}\overline{O_0}\bar{L} \quad (3\text{-}37)$$

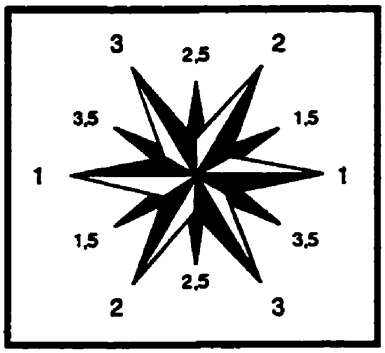
$$MO_0 = \overline{RO_2}\overline{O_0}L + \overline{RO_2}O_0\bar{L} + RO_2\overline{O_1}\overline{O_0}\bar{L} + \bar{R}\overline{O_1}\overline{O_0}L + \bar{R}\overline{O_1}O_0\bar{L} \quad (3\text{-}38)$$

Les signaux de déviation L et R peuvent être activés par un circuit interne qui commande un déplacement généralisé comme celui de la Figure 4.6. Ils peuvent également être contrôlés de l'extérieur par un module spécialisé quelconque. Le détail de la logique qui génère les signaux de modification de la déviation est présenté à la section 4.3.11.

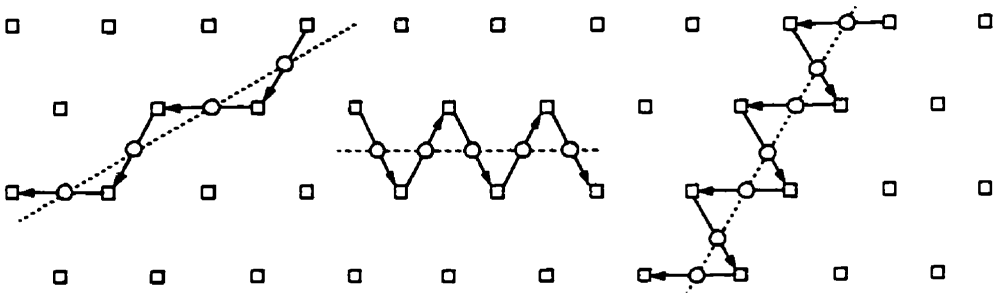
4.3.8 Module d'évaluation de l'orientation d'arête

L'orientation d'arête est une des variable d'état générée par le contrôleur MAR. Le code d'orientation d'arête est cohérent avec le code de direction et sa valeur est indépendante du sens de poursuite de l'arête. Puisqu'il est possible d'effectuer la poursuite d'arêtes en mode 30 et 60 degrés, on doit définir l'orientation d'arête selon un pas de 30 degrés (Figure 4.24). Une arête horizontale, par exemple, peut être parcourue de droite à

gauche ou de gauche à droite mais le code d'orientation d'arête *OA* est constant et égal à *1* comme c'est le cas à la Figure 4.24 (c). L'orientation d'arête est codée sur 3 bits



(a) Code d'orientation d'arête (OA)



(b) OA = 1,5 (c) OA = 1 (d) OA = 2

Figure 4.24 Code d'orientation d'arête (a). Le code est identique peu importe le sens du balayage utilisé lors du suivi d'arêtes. Trois exemples simples (en b, c et d) montrent des portions de poursuite d'arêtes ayant différentes orientations.

(OA<2:0>). La convention de la Figure 4.24 utilise les deux bits les plus significatifs pour la partie entière et le moins significatif comme partie fractionnaire. Les orientations ayant une partie fractionnaire proviennent de déplacements en mode 30°. On peut ignorer le point décimal au besoin et utiliser un code entier compris entre *1* et *7*. Le code d'orientation *0* signifie que l'orientation d'arrête est ignorée pour le cas considéré.

L'orientation d'arête est calculée en prenant la moyenne entre la direction des deux derniers déplacements du pixel d'intérêt selon une arithmétique modulo 6. Le Tableau 4.4

démontre que, pour les exemples de la Figure 4.24 (b), (c) et (d), ce calcul simple donne effectivement le bon code d'orientation d'arête.

Tableau 4.4 Calcul l'orientation d'arêtes pour les exemples de la Figure 4.29.

Cas de la Figure 4.24	Direction 1	Direction 2	Somme	Somme %6	OA
(b)	4	5	9	3	1,5
(c)	6	2	8	2	1
(d)	4	6	10	4	2

Un cas particulier survient lorsque la somme des deux directions du déplacement en zigzag donne 7 (séquence 1-6 ou 3-4) où on n'effectue pas le calcul modulo 6. La Figure 4.25 donne le détail logique de la réalisation du module d'évaluation de l'orientation d'arête. On remarque qu'une partie similaire à la Figure 4.23 est utilisée pour le calcul en modulo 6 à l'exception qu'il laisse le résultat du sommateur inchangé lorsque celui-ci est égal à 7. Le reste du circuit force le résultat à zéro selon l'occurrence de quatre conditions:

- La commande de déplacement courante (*DIR*) est nulle.
- La commande de déplacement courante (*DIR*) est identique la précédente (*DIR_i*).
- La déviation (*OFFSET*) est égale à 3 lors d'un déplacement relatif (*S_DIR_i = 1*).
- Il n'y a pas de détection d'arête entre les deux derniers points alors que la variable de configuration *P2_OA_EN* est active.

Le module d'évaluation de l'orientation d'arête sépare donc rapidement les séquences de déplacements réguliers relatifs à une détection d'arête ($OA \neq 0$) et les autres modes de déplacements.

Pour détecter l'occurrence d'une orientation d'arête particulière, on place un petit circuit de comparaison entre le code de l'orientation d'arête et une variable de configuration *OA_CMP<2:0>* tel que montré à la Figure 4.26. On peut effectuer la comparaison sur l'ensemble des bits du code d'orientation d'arête ou sur un sous-ensemble de ceux-ci. Ces variables sont présentes à l'entrée du sélecteur de données pour les signaux de requête d'interruption (Tableau D.7) et peuvent donc être combinées à volonté.

4.3.9 Détection des passages par zéro

La détection des passages par zéro fait partie intégrante du processus de détection d'arêtes. Lorsque l'image est convoluée avec l'opérateur laplacien de gaussienne, les arêtes

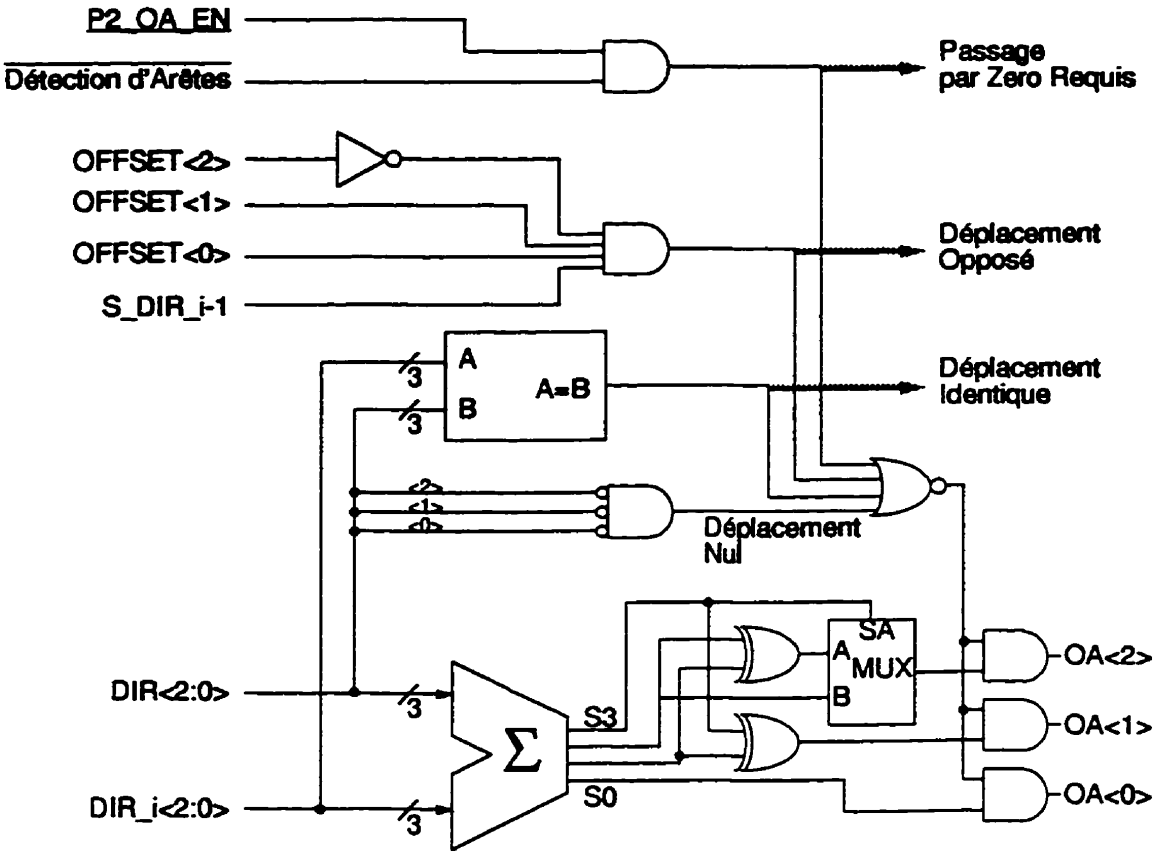


Figure 4.25 Module de calcul de l'orientation d'arête. On calcule la somme des deux derniers déplacements en mode modulo 6 lorsque la sortie du sommateur dépasse 7. La sortie est forcée à zéro lorsqu'on détecte un déplacement nul, opposée ou identique à la direction précédente. On peut configurer le module pour valider l'orientation d'arête même si la détection d'arête n'est pas effective.

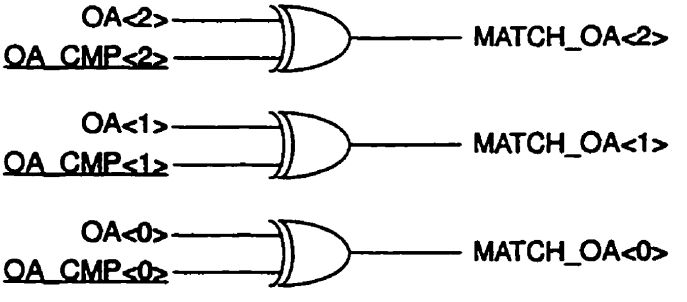


Figure 4.26 Détection de l'occurrence d'une orientation d'arête spécifique. On peut effectuer le test sur l'ensemble des bits ou sur un sous-ensemble de ceux-ci.

détectées correspondent aux passages par zéro du résultat de la convolution. Les multiples filtres du système MAR fournissent chacun un signal analogique dont il faut extraire les passages par zéro. Le contrôleur de caméra comprend 6 entrées analogiques et de la logique mixte qui transforment, pour chaque pixel visité, le signal analogique en deux bits d'information pertinente: le signe du signal (*S*) et la valeur absolue du signal auquel on applique un seuil et qu'on nomme hystérésis (*H*).

La Figure 4.27 montre un exemple en coupe des différents signaux mis en cause pour la détection d'un passage par zéro de l'image convoluée par l'opérateur $\nabla^2 G_\sigma (x, y)$. On peut voir que l'illuminance (a) change globalement autour de la ligne verticale pointillée et que cette variation implique un passage par zéro brusque de l'image convoluée (c). Le signe du résultat de la convolution (d) est une variable essentielle à la détection des passages par zéro. On doit cependant séparer les passages par zéro significatifs des faibles variations autour de l'origine du résultat de la convolution (Figure 4.27 (c)). On utilise la variable *Hystérésis* (e) pour détecter que la valeur absolue du signal a une amplitude significative. On définit le signal *H* comme suit:

$$H(x, y) = (R(x, y) > Vs+) \vee (R(x, y) < Vs-) \qquad (3\text{-}39)$$

Le signal *H* est généralement actif de part et d'autre d'un passage par zéro ce qui nous force à effectuer un traitement morphologique sur le signal *H* afin d'obtenir le signal *H**. La détection d'un passage par zéro est alors définie entre deux pixels par un changement de signe dans une région où *H** est actif. On peut voir à la Figure 4.27 (g) que cette condition est remplie à l'endroit pointé par le cône noir. Il est intéressant de remarquer qu'un point isolé d'amplitude élevée n'est pas reconnu par le détecteur d'arêtes comme c'est le cas dans la région ombragée de la Figure 4.27 puisque le signal *H** n'est pas actif des deux côtés de la région du changement de signe. Une variable de configuration nommée *EXT_P2_EN* permet au contrôleur d'utiliser une variable externe (*IO_in<2>*) comme signal de détection de passage par zéro.

Le module de génération des signaux *H* et *S* est présenté à la Figure 4.28. On compte un de ces modules pour chaque entrée analogique. Il s'agit de trois inverseurs CMOS (*I0, I1 et I2*) dont on a ajusté le seuil à des valeurs différentes en modifiant la taille des transistors qui les constituent. Le signal d'entrée est comparé à *2.2 V* et définit ainsi le bit signe (*S*). Les deux autre inverseurs (*I0 et I1*) servent à détecter si la valeur absolue de l'amplitude du signal analogique dépasse *200 mV*. On doit donc ajuster la composante continue de chaque entrée analogique (le zéro) à la limite de commutation de l'inverseur *I1*

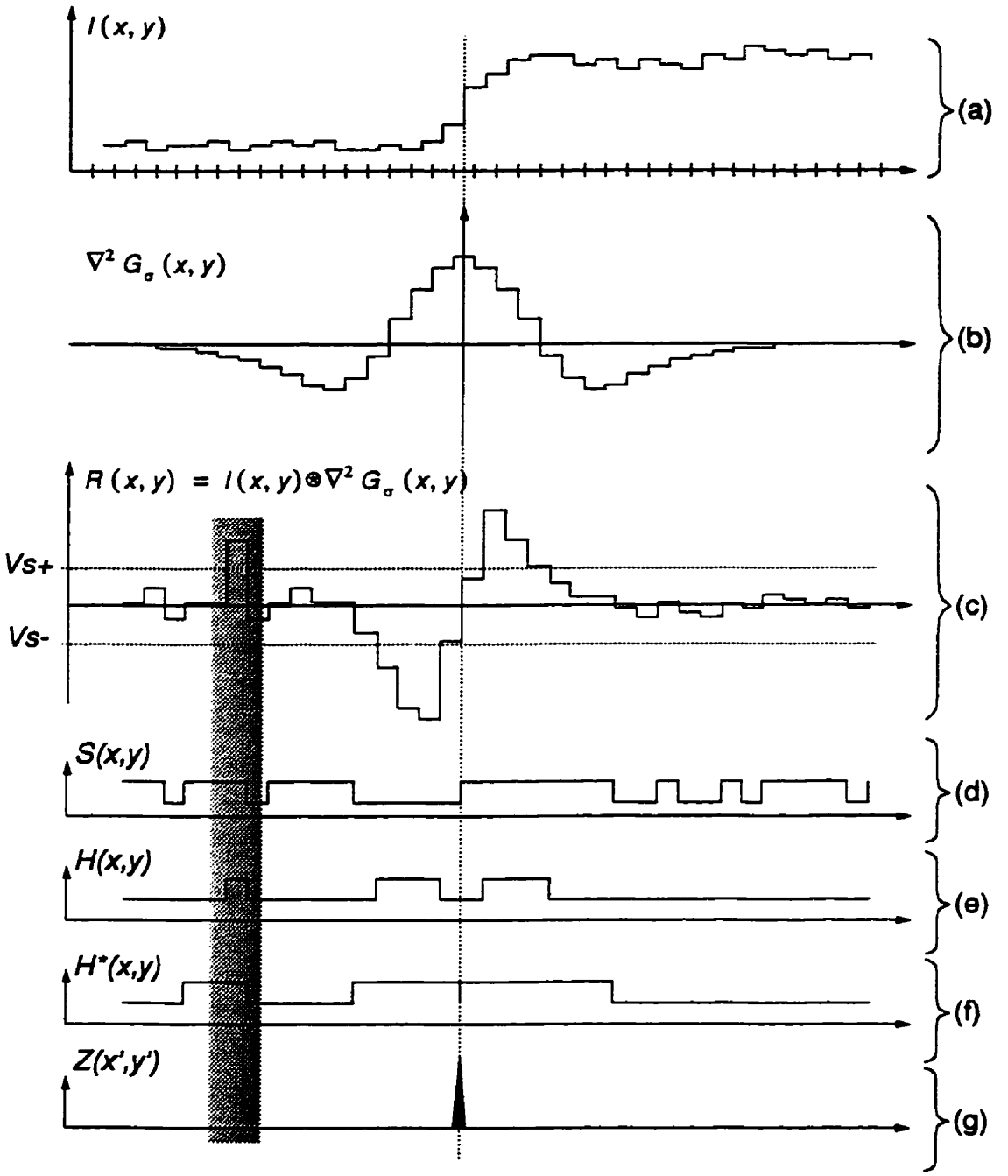


Figure 4.27 *Détection des passages par zéro du signal R(x,y). Une discontinuité dans l'image d'illuminance (a) se traduit par un passage par zéro du résultat de la convolution (c) avec l'opérateur laplacien de gaussienne (b). Les deux bits S (d) et H (e) sont générés par des opérations de seuillage. On dérive le signal H* (f) en dilatant le signal H conditionnellement à une continuité du signal S. Le passage par zéro (g) correspond à un changement de signe (S-1 != S+1) dans une région d'hystérésis valide (H*-1 = H*+1 = 1).*

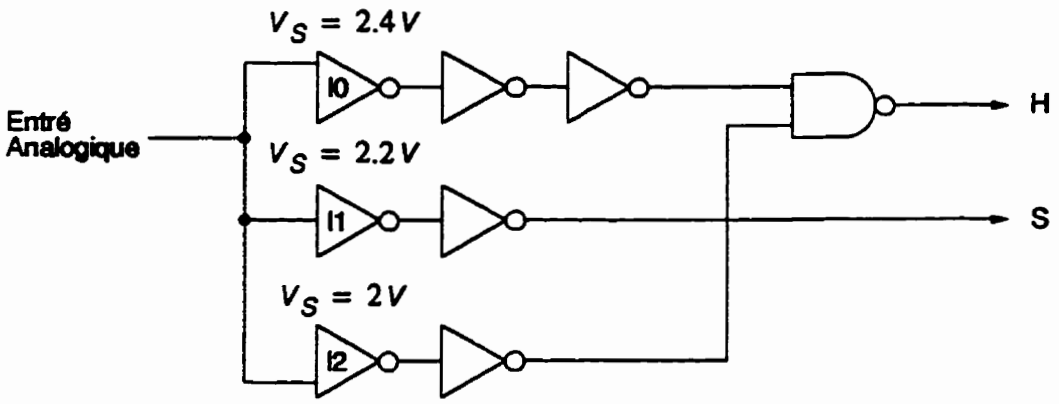


Figure 4.28 *Module intégré de génération des bits "S" et "H" à partir du signal d'entrée analogique. Les seuil de commutation des inverseurs I0 à I2 sont ajustés à des valeurs différentes en modifiant la taille des transistors qui les composent.*

soit *2.2 V*. Les inverseurs additionnels servent à régénérer le signal numérique lorsque l'entrée des inverseurs *I0* à *I2* est très près de la valeur du seuil de commutation.

Le signal dérivé H^* est obtenu par l'application répétée d'une opération morphologique de dilatation. La dilatation est effectuée en deux dimensions sur le signal H et est conditionnelle à une continuité du signe. Ceci simplifie l'application de l'opération de dilatation et rend le résultat insensible au filtre utilisé ou au type d'image qui est traitée. La Figure 4.29 montre le masque de dilatation de même que l'équation logique qui régit le passage de H à H_R. Le signal final H^* est obtenu en effectuant l'opération de dilatation un certain nombre de fois. On a avantage à utiliser cet opérateur conditionnel plutôt qu'une simple dilatation sur le signal H car on peut l'appliquer récursivement un bon nombre de fois sans risquer que la région d'hystérésis valide n'atteigne une région bruitée du signal signe. L'opération de dilatation devient alors pratiquement indépendante de la résolution spatiale de même que du type de scène analysée.

On peut effectuer la dilatation de deux façons différentes, soit par l'application récursive de l'opérateur de dilatation, soit en effectuant une opération de remplissage tout en conservant la même règle morphologique. La Figure 4.30 montre un exemple simple de l'image d'un disque en (a) qui, une fois passée par un module de seuillage comme celui de la Figure 4.28, donne, en (b), une image binaire des deux signaux H et S. On peut remarquer que le changement de signe est très clair autour du disque et que plusieurs passages par zéro non-significatifs se retrouvent dans les régions d'iso-illuminance. On peut remarquer

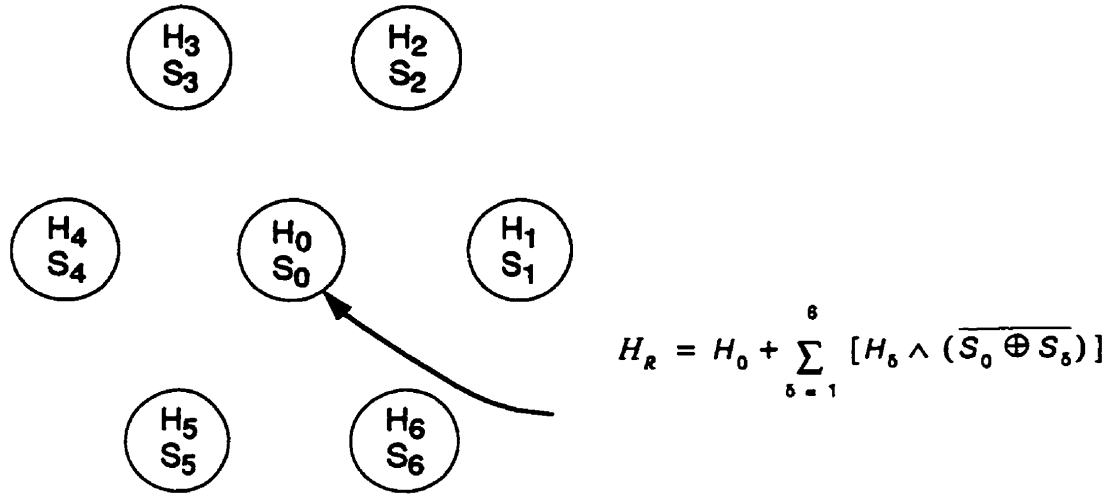


***Figure 4.29** Opérateur de dilatation appliqué au signal H. Le résultat de H_R au centre du masque reste actif si $H_0 = 1$ et est activée dans le cas où au moins un des voisins immédiat à le signal "H" actif et qu'il est de signe identique au pixel central.*

également que les lieux de forte amplitude ($H = 1$) forment deux anneaux de part et d'autre de la région d'arête (comme la courbe (e) de la Figure 4.27) et qu'un pixel de détection bruitée du signal H est présent à l'extrême gauche de l'image.

L'image (c) de la Figure 4.30 montre les modifications apportées au bit H après un certain nombre de dilatations récursives appliquées à l'image de (b). Le terme récursif indique qu'on calcule l'image résultat à partir de l'image source non-modifiée puis, qu'on applique à nouveau l'opérateur sur le résultat précédent. De cette façon, la dilatation est isotropique et la croissance de la région est limitée au nombre d'applications de l'opérateur. Par contre, l'image (e) de la Figure 4.30 montre le résultat d'une dilatation effectuée à la manière d'un remplissage en remplaçant la valeur de H par H_R directement sur l'image source dès qu'elle est calculée. On peut pratiquement remplir les régions affectées de l'image en deux ou trois balayages dans des orientations différentes à l'exception des cas pathologiques comme pour une région ayant la forme d'un colimaçon. L'image d'arêtes en (f) provenant de cette procédure de remplissage révèle cependant une fausse détection autour de la tache qui renfermait en (b) la petite détection bruitée.

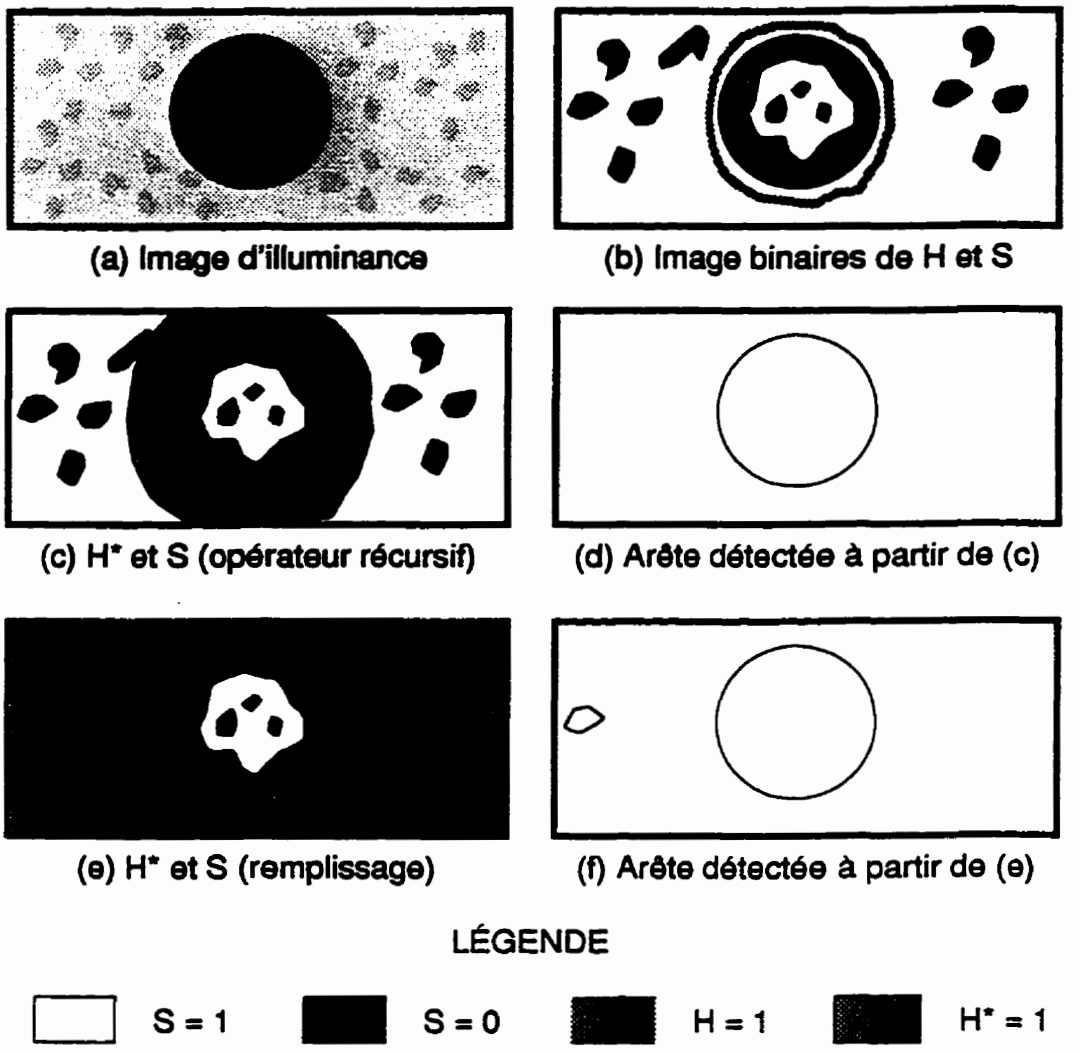


Figure 4.30 *Exemple 2D de l'opération de détection des passages par zéro. L'image d'illuminance (a) produit l'image brute de signe et d'hystérésis (b) similairement aux courbes (d) et (e) de la Figure 4.27. On applique récursivement l'opérateur morphologique de la Figure 4.29 un nombre limité de fois en (c) alors qu'on utilise une fonction de remplissage en (e). On représente en (d) et (f) les images d'arêtes produites à partir de (c) et (e) respectivement.*

La solution récursive favorise la réalisation d'un module en pipe-line pour le traitement morphologique lors de l'acquisition d'images en mode ligne par ligne. On obtient ainsi le résultat avec deux lignes de délai par module et cette architecture peut être réalisée en parallèle. Le désavantage majeur de la solution par remplissage est la sensibilité à la fausse détection lorsqu'un point unique de bruit se dissimule dans l'image d'hystérésis.

On doit porter une attention particulière aux cas où l'illuminance décroît en escalier. Une fausse détection peut survenir lorsqu'on dilate trop le signal H comme c'est le cas à la Figure 4.31 (f). On aura donc avantage à utiliser la méthode récursive pour une, deux ou trois itérations afin de limiter l'occurrence de fausse détections.

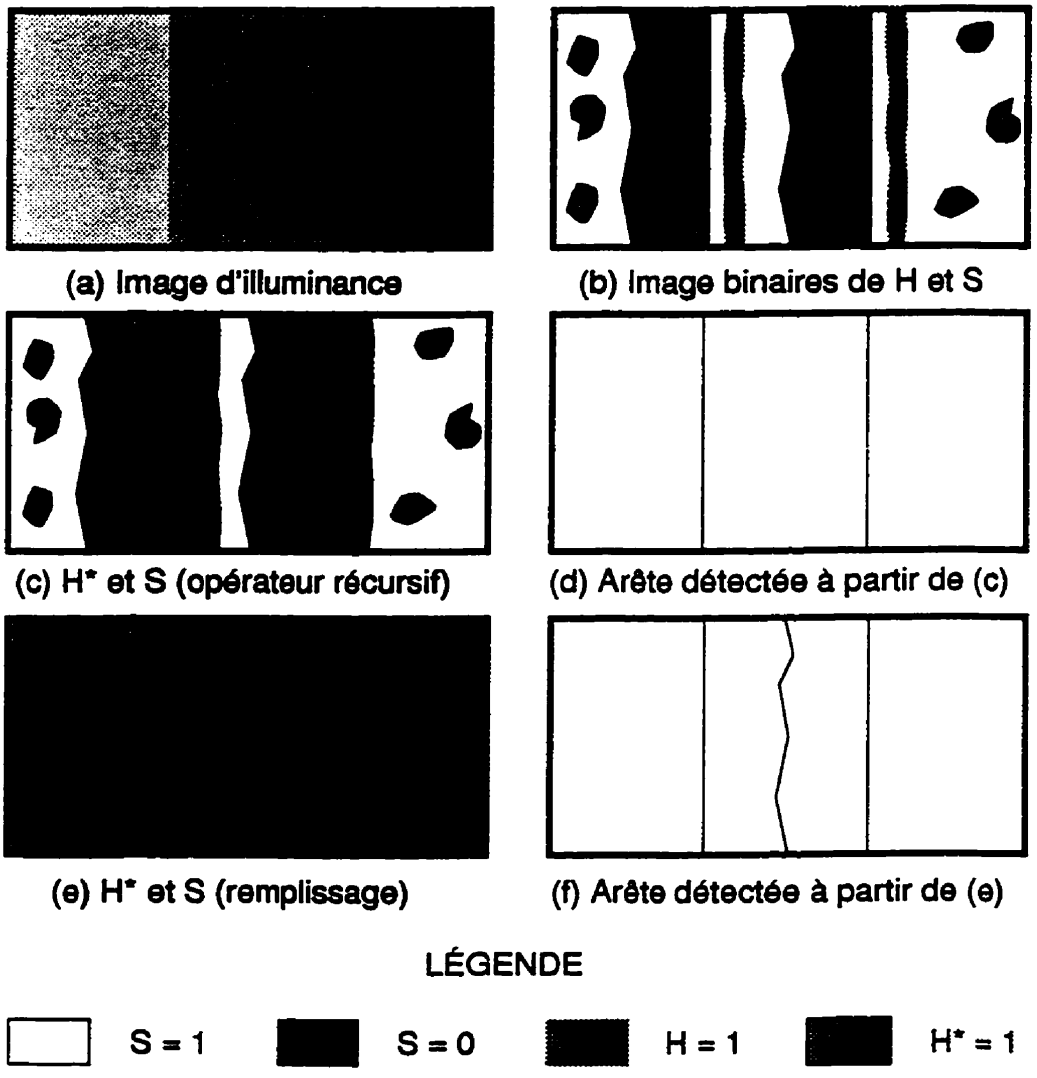


Figure 4.31 *Illustration d'une séquence de changements d'illuminance graduels (a) et du problème associé à la détection d'une fausse arête si on dilate trop les régions actives d'hystérésis.*

A partir de ce qui précède, on voit qu'une détection d'arête se fait entre deux pixels et non sur un pixel. Ceci implique que la grille qui sert à identifier les passages par zéro est plus dense que la matrice de pixels elle-même. La Figure 4.32 illustre bien ce phénomène

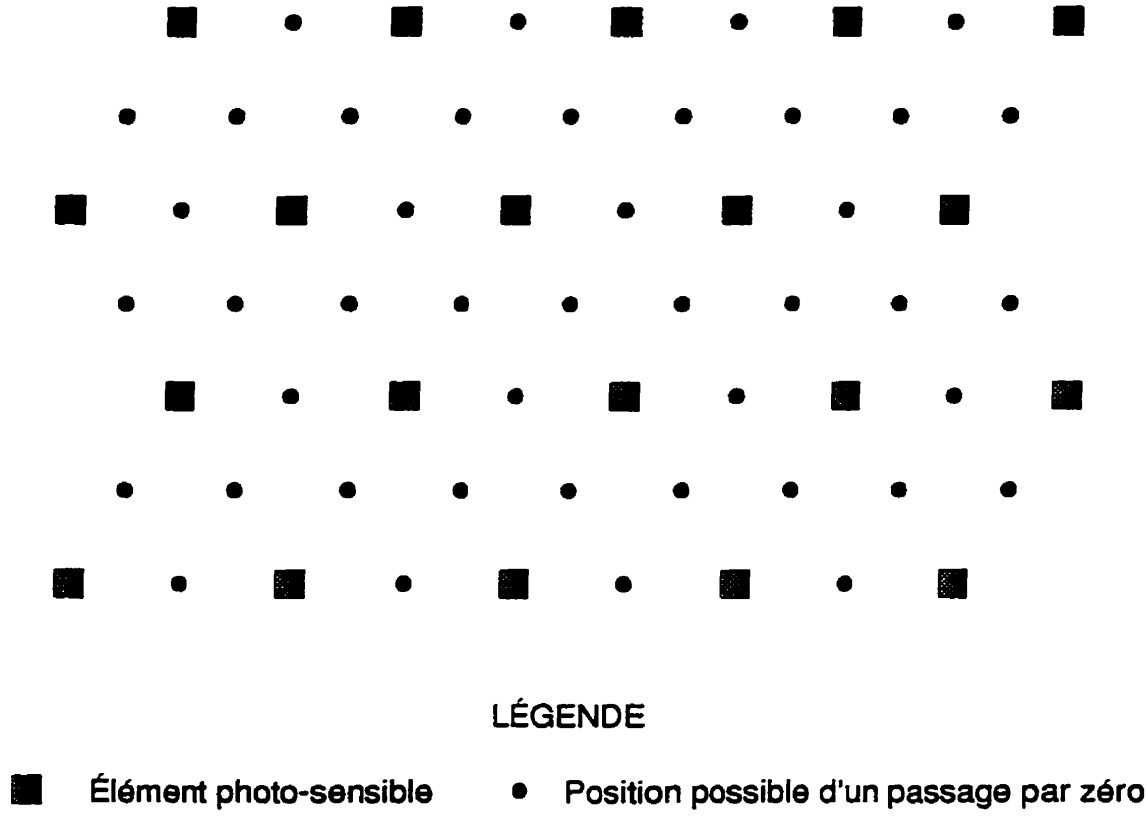


Figure 4.32 Grille des positions possibles des détections d'arêtes. Puisqu'un passage par zéro est définit entre deux pixels la grille résultante est donc quatre fois plus dense.

où on peut voir qu'une détection possible d'arête entre chaque paire de pixels voisins définit une grille quatre fois plus dense que la matrice de pixels photo-sensibles. En posant pour hypothèse qu'on effectue la détection des passages par zéro uniquement à partir des signaux H et S, on élimine le pixel comme position possible du passage par zéro et on obtient finalement une densité des lieux d'arêtes trois fois plus élevée que celle de la grille de pixels. Cette observation est la conséquence de l'augmentation de la dimension des matrices requises pour représenter les images d'arêtes extraites par le système MAR.

4.3.10 Routage et sélection des données

Le contrôleur génère un bon nombre de variables d'état pour la création de la description d'une scène et susceptibles d'influencer le déroulement du processus de commande de la caméra MAR. Le bus de données *MD<15:0>* qu'on peut voir à la Figure 4.1 sert à véhiculer cette information. Chaque bit est branché à la sortie d'un multiplexer programmable pour choisir la variable à stocker dans la mémoire d'état. Une des tâches du programme exploitant le système MAR consiste à configurer correctement ces multiplexers. Une sélection similaire est effectuée au niveau du bus *IO<4:0>*. On utilise cependant le bus *IO* principalement pour effectuer le test fonctionnel du contrôleur ou comme signal de contrôle externe. On retrouve à l'ANNEXE D la liste des variables qui peuvent être programmées pour les bits du bus de données *MD<15:0>* (Tableau D.4 et Tableau D.5) et pour le bus *IO<4:0>* (Tableau D.6 et Tableau D.7). On retrouve également à l'ANNEXE D l'adresse des registres de configuration de chaque sélecteur de données (Tableau D.1).

Le module de sélection des données comprend, en plus du multiplexer, un registre de quatre bits pour configurer le mode de transfert de chaque canal. Chaque bit des bus *MD* et *IO* peut être configuré en mode entrée uniquement, en mode sortie uniquement ou en mode entrée/sortie. La Figure 4.33 présente l'arrangement du sélecteur de données ainsi que de la logique d'échantillonnage en mode de lecture. Les champs du registre de configuration de 8 bits (qu'on retrouve en souligné à la Figure 4.33) sont décrits en détail à l'ANNEXE D (Figure D.2). L'adresse des registres de configuration pour chaque bit des bus *MD* et *IO* est présentée au Tableau D.1. Les différents modes d'accès qu'on peut programmer pour chaque module de sortie sont résumés au Tableau 4.5. La configuration

Tableau 4.5 Principales configurations possibles pour chaque canal des bus MD<15:0> et IO<4:0>.

OE_ON_ MOVE	EXT_OE	OE_ ACTIF	GRAB_ IN_EN	CONFIGURATION
0	0	1	X	Sortie seulement
0	0	0	0	Entrée seulement
1	0	0	1	Sortie lors d'une commande de déplacement; Entrée autrement.
0	1	0	1	Sortie lorsque le signal IO_in<4> est actif; Entrée autrement.
1	1	0	1	Sortie si MOVE ou IO_in<4> est actif; Entrée autrement

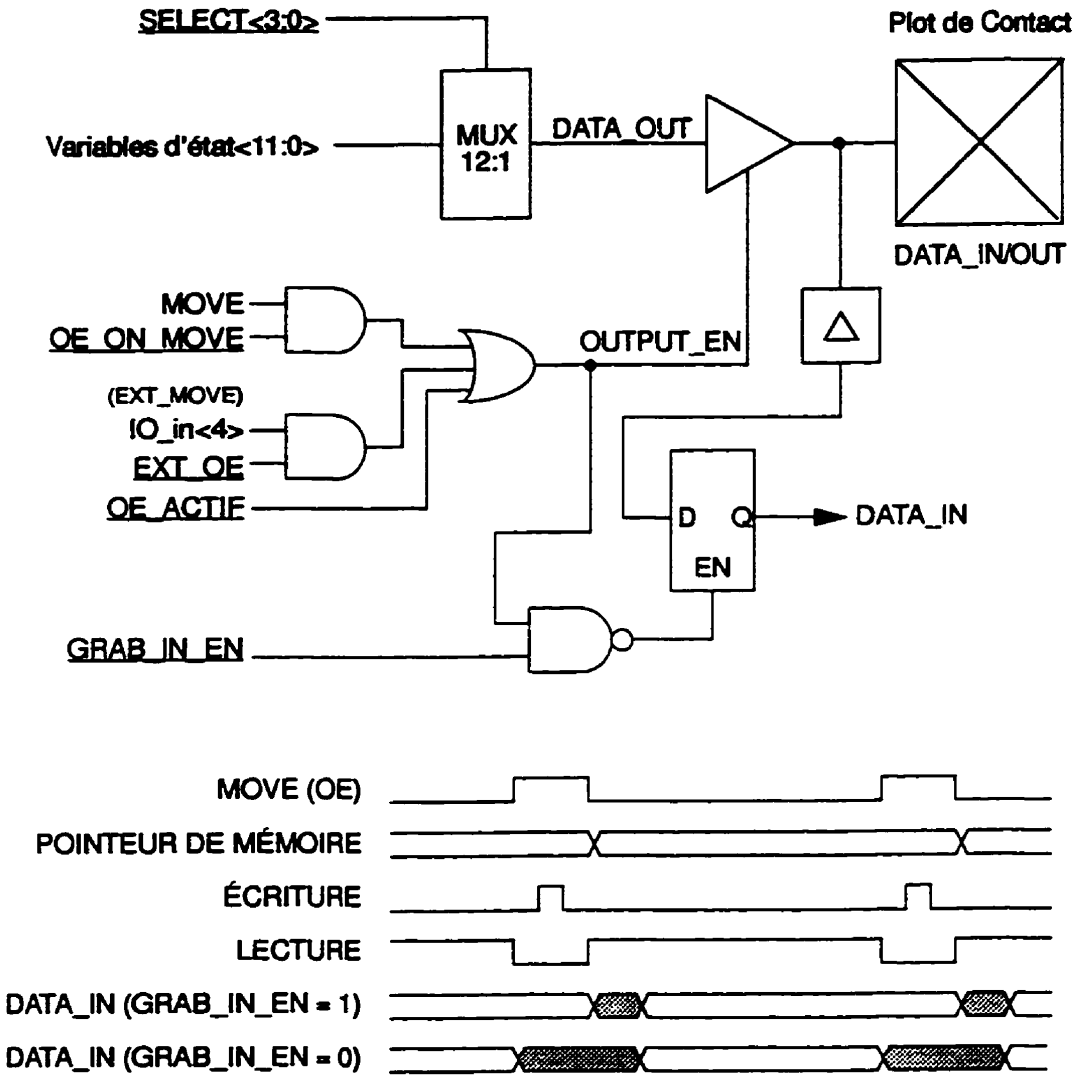


Figure 4.33 Configuration du transfert de données pour chacun des bits des bus MD<15:0> et IO<4:0>. Le multiplexer choisit laquelle des variables d'état est écrite lorsque le canal est en mode SORTIE. La bascule transparente sert à échantillonner la valeur lue au moment ou le contrôleur effectue une écriture en mode MOVE ou EXT_OE. Le diagramme temporel montre la synchronisation des signaux lorsque le canal est en mode combiné. Chaque canal peut également être configuré en entrée ou en sortie seulement.

choisie pour chaque canal de communication dépend du contexte d'utilisation du contrôleur MAR et des circuits périphériques qui sont utilisés.

Le mode mixte d'entrée/sortie est prévu typiquement pour le bus *MD* afin de permettre un cycle de lecture suivi d'un cycle d'écriture à chaque déplacement du pixel d'intérêt. On voit d'ailleurs à la Figure 4.33 le diagramme temporel de l'utilisation de ce mode d'accès à la mémoire périphérique. On peut donc rendre une écriture fonction du contenu précédemment stocké dans la mémoire avant d'y écrire une nouvelle valeur. En pratique, on peut modifier uniquement certains bits de la mémoire d'état et en laisser d'autres inchangés ou encore utiliser un champ de quelques bits de chaque case de la mémoire MAR comme compteur. Dans le premier cas, on doit lire le contenu de la mémoire pendant la période de latence entre deux commandes de déplacement, mémoriser cette valeur puis, la récrire dans la case mémoire. La procédure est similaire pour le cas du compteur sauf qu'on insère un bloc de logique combinatoire entre la valeur lue et le résultat à écrire. On verra à la section 4.3.13 comment on exploite cette propriété pour l'archivage du balayage.

Les canaux réservés à la génération des signaux de requête d'interruption (*IO<7:5>*) utilisent le même genre de circuit qu'à la Figure 4.33 sauf pour le module de multiplexage. La Figure 4.34 donne le détail du circuit qui permet de configurer un signal de demande d'interruption comme une combinaison de plusieurs variables. La fonction de ce circuit est cohérente avec la discussion de la section 4.3.3 et les champs du registre de configuration de 16 bits (qu'on retrouve en souligné à la Figure 4.34) sont décrits en détail à l'ANNEXE D (Figure D.3). La liste des variables qui peuvent être programmées pour les différents signaux de demande d'interruption est présentée au Tableau D.4 et au Tableau D.5.

4.3.11 Segments rectilignes d'arêtes et déplacements généralisés

Puisqu'on est intéressé à extraire l'information sur les arêtes dans le format le plus compact possible, on a prévu un sous système qui extrait les segments rectilignes d'arêtes. La diminution du nombre de points d'arêtes rend la structure de données plus compacte mais elle limite surtout le nombre de transferts entre le système hôte et la caméra tout en réalisant l'extraction de primitives de plus haut niveau. En plus du signal de discontinuité d'arête qui est une sortie du PLA, un module spécialisé est intégré au contrôleur pour détecter les longues discontinuités (*Long_DISC*). Puisque le signal de discontinuité d'arête (*DISC*) est actif lorsque la poursuite change d'orientation, une arête oblique rectiligne génère des discontinuités cycliques comme on peut le voir à la Figure 4.35. Ce cas est

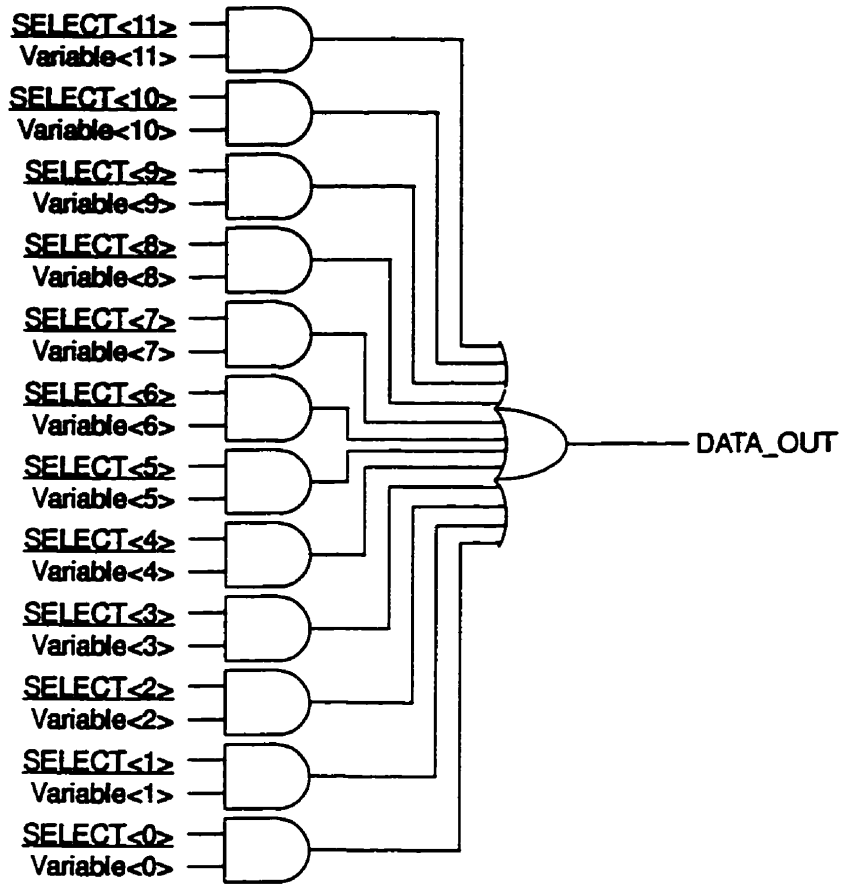


Figure 4.34 *Multiplexer spécialisé pour la configuration des signaux de requête d'interruption. Lorsqu'une des variables sélectionnées est active le signal de sortie "DATA_OUT" est alors actif.*

principalement rencontré lors de l'analyse de scènes polygonales. La détection d'une longue discontinuité (*Long_DISC*) se fait à la suite de la rupture du cycle régulier des deux signaux primaires de discontinuité d'arête (*DISC*) et du sens de la discontinuité (*Sens_DISC*).

On détecte un cycle régulier des signaux primaires de discontinuité lorsque les paires consécutives des segments d'arêtes détectés ont des longueurs identiques et que le changement de la direction de recherche alterne régulièrement à chaque segment comme c'est le cas à la Figure 4.35. On remarque que la rupture de la régularité des signaux de discontinuité signifie que la discontinuité précédente est le point de changement d'un long

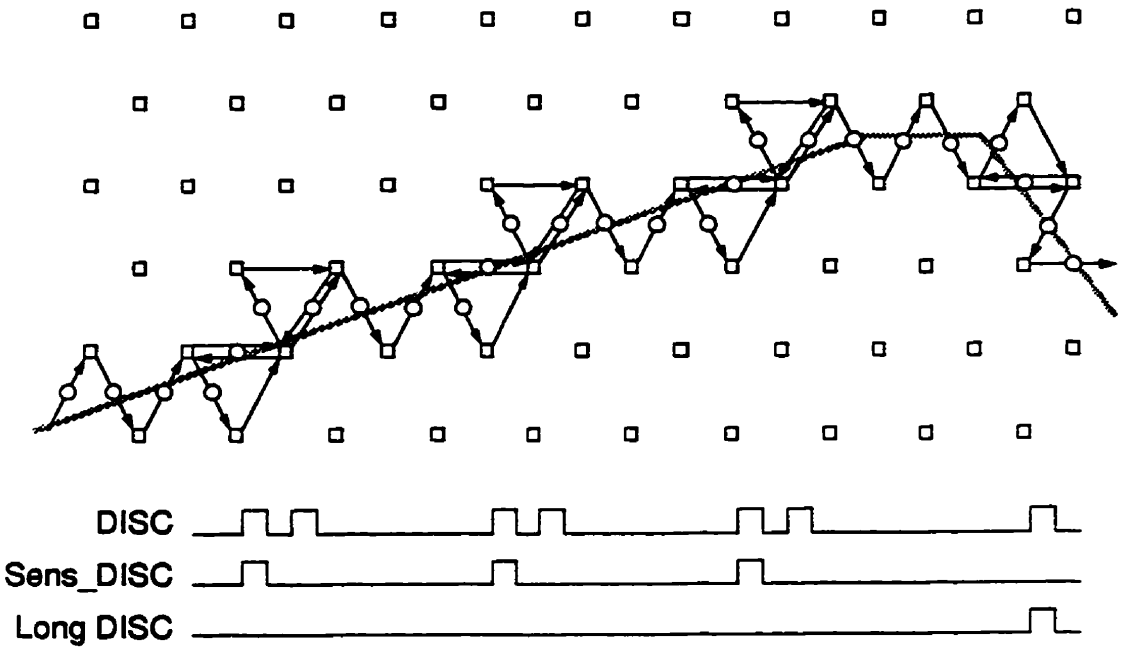


Figure 4.35 Procédure de détection d'un segment rectiligne d'arête. Les séquences cycliques de discontinuités avec des changements d'orientation de recherche opposées peuvent être considérées comme une arête régulière et rectiligne. Une discontinuité de longueur ou d'orientation différente indique que la discontinuité qui précède est la dernière de la séquence régulière.

segment d'arête. Il faut par conséquent mémoriser la position de ce point pour décrire correctement les lieux de discontinuité d'arêtes (voir la section 4.3.13 pour plus de détails à ce sujet). Le circuit qui sert principalement à détecter une séquence régulière de changement d'orientation d'arêtes sert également à générer de façon similaire les signaux *LEFT* et *RIGHT* pour le mode de déplacement généralisé (section 4.2.5, Figure 4.6 et Figure 4.22). Le texte, les figures et les équations logiques qui suivent donnent la description de ce module à fonction double.

Le signal de discontinuité est redéfini localement en fonction de la discontinuité d'arête lorsqu'on exécute l'instruction de poursuite d'arêtes (*FIND = 1*) et en fonction de la valeur des compteurs *A* et *B* autrement (*FIND = 0*). L'équation logique suivante décrit le comportement du signal *Local_DISC*:

$$Local\_DISC = \overline{FIND} \cdot MOVE\,(MATCH\_A + MATCH\_B) + FIND \cdot DISC \quad (3\text{-}40)$$

Une bascule bistable définit deux cycles distincts soit le *CYCLE_A* et le *CYCLE_B* ($\overline{CYCLE\_A}$). Cette bascule qu'on retrouve au haut de la Figure 4.36 change d'état lorsqu'on détecte une condition locale de discontinuité (*Local_DISC*) et est remise à zéro lorsqu'on initialise le compteur d'événement N. Le processus débute donc toujours par le cycle *A*. On utilise deux compteurs synchrones de 8 bits nommés *A* et *B* qui comptabilisent le nombre de déplacements consécutifs dans une même direction. Chaque compteur peut être remis à zéro de façon synchrone (*RESET*) et ce signal est prioritaire sur la commande d'incrémentation (*EN*). Lorsqu'on est en mode de poursuite d'arêtes, ces deux compteurs mémorisent alternativement la longueur des segments de base. Lorsqu'on rencontre une discontinuité d'arête, la valeur du compteur actif est stockée dans un registre (*GRAB_A = 1* si *CYCLE_A = 1*; *GRAB_B = 1* si *CYCLE_A = 0*) et sert de valeur de comparaison pour le prochain cycle. Les équations (3-41) à (3-46) résument les principaux signaux engendrés par le bloc de logique combinatoire de la Figure 4.36 qui commandent les deux compteurs de même que ceux qui activent la bascule du comparateur (*GRAB_A* et *GRAB_B*).

$$Reset\_A = RESET\_N + (Local\_DISC \cdot CYCLE\_A) \tag{3-41}$$

$$EN\_A = (N\_EN \cdot CYCLE\_A) + (Local\_DISC \cdot \overline{CYCLE\_A}) \tag{3-42}$$

$$Reset\_B = RESET\_N + (Local\_DISC \cdot \overline{CYCLE\_A}) \tag{3-43}$$

$$EN\_B = (N\_EN \cdot \overline{CYCLE\_A}) + (Local\_DISC \cdot CYCLE\_A) \tag{3-44}$$

$$GRAB\_A = Local\_DISC \cdot FIND \cdot CYCLE\_A \tag{3-45}$$

$$GRAB\_B = Local\_DISC \cdot FIND \cdot CYCLE\_B \tag{3-46}$$

On remarque que les signaux *GRAB_A* et *GRAB_B* ne peuvent s'activer que lors de l'exécution de poursuite d'arêtes (*FIND = 1*). Dans le cas d'un déplacement simple ou complexe avec direction généralisée (*FIND = 0*), les bascules sont programmées de façon asynchrone par le système hôte (*MAX_A_WRITE* et *MAX_B_WRITE*) et ces valeurs correspondent aux deux valeurs limites pour un déplacement généralisé (comme c'est le cas pour les valeurs *A_MAX =3* et *B_MAX = 1* de l'exemple de la Figure 4.6). On peut voir à la Figure 4.36 qu'une autre bascule mémorise le sens de la discontinuité précédente. Cette dernière doit être de signe opposé à la valeur courante pour valider un long segment d'arête. L'équation (3-47) définit les conditions de détection de la fin d'une séquence régulière de discontinuités.

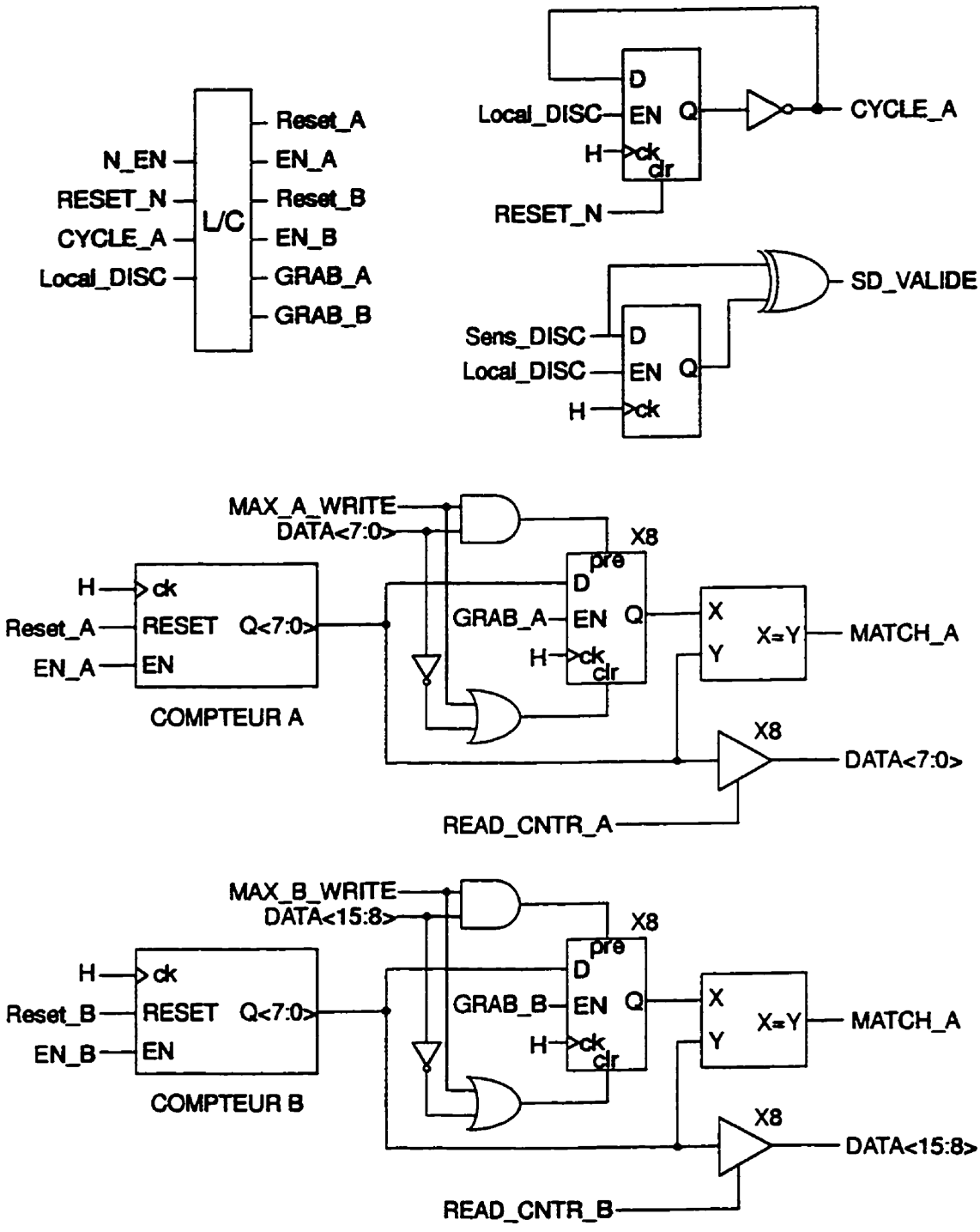


Figure 4.36 *Circuit de détection des segments rectilignes d'arêtes. Deux compteurs synchrones munis d'un comparateur d'égalité mémorisent alternativement la longueur des discontinuités d'arêtes. Deux bascules définissent le cycle (A ou B) de même que la validité des changements de directions de recherche. Ce circuit génère les signaux LEFT et RIGHT pour le déplacement généralisé. Les équations logiques (3-41) à (3-46) résument la logique combinatoire qui régit les compteurs et les comparateurs.*

$$Long\_DISC = ((CYCLE\_A \cdot\cdot \overline{MATCH\_A}) + (\overline{CYCLE\_A} \cdot\cdot \overline{MATCH\_B}) + \overline{SD\_VALIDE}) \cdot\cdot Local\_DISC \quad (3\text{-}47)$$

Une longue discontinuité d'arête (*Long_DISC*) survient uniquement lors de la détection d'une discontinuité (*Local_DISC = 1*) en combinaison avec l'un des cas suivant:

- La longueur du segment de base lors du cycle *A* n'est pas la même que celui du cycle *A* précédent.
- La longueur du segment de base lors du cycle *B* n'est pas la même que celui du cycle *B* précédent.
- Les deux derniers changement d'orientation de la poursuite d'arêtes sont effectués dans le même sens (*SD_VALIDE == 0*).

Les compteurs *A* et *B* sont utilisés pour le mode de déplacement généralisé (ou déplacement à angle). Dans ce cas, on programme la valeur limite des deux compteurs de même que la direction du premier changement d'orientation (*START_LEFT*) avant de lancer l'instruction. Si la variable commandant ce mode de déplacement (*ANGLE_EN*) est active, les signaux *LEFT* et *RIGHT* provoqueront alternativement une modification de la déviation pour le calcul du prochain déplacement tel que décrit à la section 4.3.7 (Figure 4.22). La Figure 4.37 donne un aperçu des variables à configurer pour effectuer un

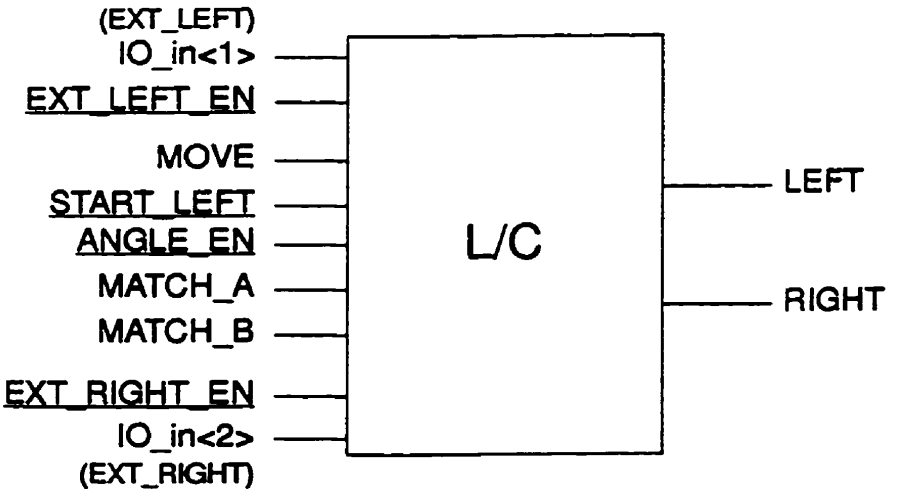


Figure 4.37 Liste des signaux de configuration qui composent les signaux de modification de la déviation LEFT et RIGHT. Ils sont utilisés par l'unité arithmétique de direction afin de définir le mode de déplacement généralisé. Pour que ce module soit actif, la variable de configuration ANGLE_EN doit être actif.

déplacement généralisé. Les équations (3-48) et (3-49) résument les équations du bloc de logique combinatoire de la Figure 4.37.

$$LEFT = (IO\_in\langle 1\rangle \cdot EXT\_LEFT\_EN) + ((MATCH\_A \cdot START\_LEFT) + (MATCH\_B \cdot \overline{START\_LEFT})) \cdot MOVE \cdot ANGLE\_EN \quad (3\text{-}48)$$

$$RIGHT = (IO\_in\langle 2\rangle \cdot EXT\_RIGHT\_EN) + ((MATCH\_B \cdot START\_LEFT) + (MATCH\_A \cdot \overline{START\_LEFT})) \cdot MOVE \cdot ANGLE\_EN \quad (3\text{-}49)$$

On peut également activer de façon externe les signaux *LEFT* et *RIGHT* par les canaux IO<2:1> lorsqu'on configure correctement les variables *EXT_LEFT_EN* et *EXT_RIGHT_EN*. Le niveau de complexité de ce module qui combine la détection de longue discontinuité d'arête et le mode de déplacement généralisé pourrait justifier le design d'un PLA limitant le délai de propagation à travers les nombreux blocs de logique combinatoire. Dans cette éventualité, on pourrait ajouter d'autres capacités de détection comme un degré de flexibilité sur la régularité des segments d'arête (variation locale de la longueur des segments de base) ou encore la détection de courbure régulière.

4.3.12 Détection de retour au point de départ

La détection de retour au point de départ est une propriété essentielle pour éviter que l'algorithme de poursuite d'arêtes ne fasse plusieurs cycles lors du suivi d'un contour fermé. Une paire de comparateurs détecte cette condition à partir des indices de position *X* et *Y* de la Figure 4.20. La Figure 4.38 donne le détail de la réalisation de ce module qui active le signal *X0* et *Y0* uniquement lorsque la position courante du pixel d'intérêt est identique à la position de départ (lors du début de l'exécution d'une instruction). Les signaux intermédiaires *X=X0* et *Y=Y0* permettent de détecter individuellement l'occurrence du croisement des axes principaux d'un système de coordonnées centré au point de départ par le pixel d'intérêt.

On prévoit utiliser ces signaux comme requête d'interruption au système hôte. On les retrouve d'ailleurs comme variables potentielles à l'entrée des multiplexers des canaux *IO<7:5>* (Tableau D.7).

4.3.13 Archivage du balayage et des points de discontinuité d'arête

Les principales variables d'état, l'adresse du pixel d'intérêt de même que sa position sont des données susceptibles de contenir des informations pertinentes à la description de la scène. Une procédure d'archivage conserve systématiquement l'ensemble

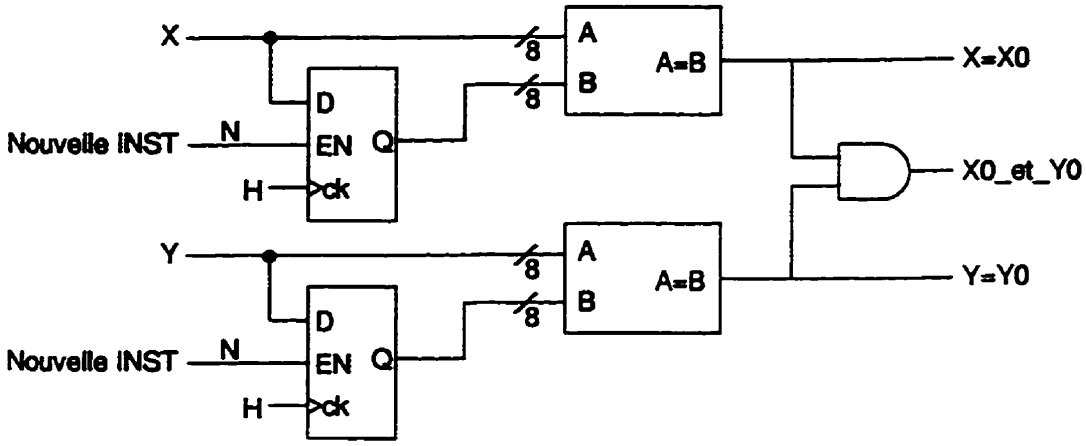


Figure 4.38 Circuit de détection du retour au point de départ. L'indice de position du pixel d'intérêt est mémorisé avant l'exécution de toute instruction. On peut également détecter si le pixel d'intérêt traverse l'un ou l'autre des axes définis par le point de départ.

de ces informations pour le pixel courant de même que pour les quatre pixels précédemment visités. L'archivage du balayage se fait simplement à l'aide d'une queue de données dont chaque étage est adressable par le système hôte en lecture comme montré à la Figure 4.39. On utilise un bloc semblable pour l'archivage de plusieurs variables d'état. Le décalage de l'information stockée se fait lorsque le signal de déplacement (*MOVE*_CAPT) est actif. Le Tableau 4.6 résume les variables mémorisées. Le Tableau 4.7 et le Tableau 4.8 décrivent le contenu des deux registres de description d'état de 16 bits

***Tableau 4.6** Liste des variables d'état qui sont archivées*

Nom	Variables	Description
pos<15:0>	Y<7:0>,X<7:0>	Indice de position (pointeur de mémoire)
pos_y<15:0>	pos_y<11:0>	position réelle en Y du pixel avec signe étendu à 16 bits
pos_x<15:0>	pos_x<11:0>	position réelle en X du pixel avec signe étendu à 16 bits
Etat_1<15:0>	voir Tableau 4.7	Variables binaires de description d'état
Etat_0<15:0>	voir Tableau 4.8	Variables binaires de description d'état

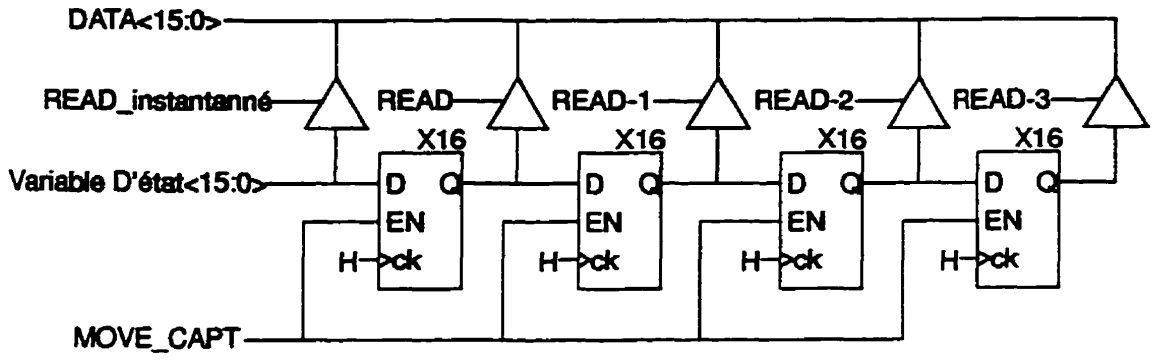


Figure 4.39 Registre à décalage utilisé pour l'archivage des variables d'état. Le signal de déplacement MOVE_CAPT active le décalage de l'information. Le système hôte peut extraire de ces registres l'information des quatre derniers pixels visités. Ce circuit est répété plusieurs fois pour différentes variables d'état.

Etat_1 et *Etat_0*. Ces deux registres réunissent les principales variables binaires internes du contrôleur MAR nécessaires au système hôte pour l'exécution rapide d'une requête d'interruption.

La position des deux dernières discontinuités d'arête est également mémorisée par le contrôleur afin de permettre au système hôte de connaître la coordonnée du dernier point valide d'un segment rectiligne. Le circuit alors utilisé est similaire à celui de la Figure 4.39 à la seule différence que le signal qui active le registre à décalage (*EN=MOVE_*CAPT) est remplacé par le signal de détection d'arêtes *DISC*. Ce circuit mémorise l'indice de position X et Y et indique ainsi au système hôte la coordonnée des deux derniers points de discontinuité d'arête. Les registres d'archivage de l'information accessible en lecture par le système hôte ont tous une adresse spécifique. La description de la plage d'adressage est disponible à l'ANNEXE D (Tableau D.1).

Tableau 4.7 Contenu du registre de description d'état (Etat_1<15:0>) mémorisé en mode d'archivage

Etat_1	Variable	Description
<15>	RESET_E	Remise à zéro du compteur d'événement E
<14>	E_EN	Occurrence d'un événement pour le compteur E
<13>	E_ZERO	Limite du compteur E atteinte
<12>	RESET_N	Remise à zéro du compteur d'événement N
<11>	N_EN	Occurrence d'un événement pour le compteur N
<10>	N_ZERO	Limite du compteur N atteinte
<9>	Condition	Condition définissant la frontière
<8>	P2	Détection d'un passage par zéro entre les deux derniers pixels
<7>	Sens_DISC	Sens de la discontinuité d'arête
<6>	DISC	Discontinuité d'arête
<5>	Long_DISC	Discontinuité d'un long segment d'arête rectiligne
<4:1>	Visite<3:0>	Nombre de visite par le pixel d'intérêt
<0>	DIR_i<2>	Direction de provenance (suite dans Etat_0<15:14>)

Un module nommé VISITE est réalisé pour compiler l'achalandage de chaque pixel. Ce module utilise quatre bits de la mémoire externe pour identifier le nombre de fois qu'un pixel a été visité. Cette information est particulièrement utile en mode de poursuite d'arêtes pour savoir si une arête rencontrée a déjà été visitée ou non. On exploite la capacité de lecture/écriture avec capacité de mémorisation des modules de sélection de données (section 4.3.10) pour additionner "1" au compte courant d'un pixel chaque fois qu'il est visité par le pixel d'intérêt. La Figure 4.40 présente la description de ce circuit. Une porte est placée à l'entrée du circuit pour détecter la valeur limite (15) et la laisser inchangée si ce cas se présente. On effectue une comparaison avec une valeur limite (*Visite_CMP<3:0>*) pour savoir si l'achalandage dépasse cette limite. On peut changer la polarité du test en modifiant la variable de configuration *Visite_LT_CMP_TEST* et ainsi détecter une limite inférieure.

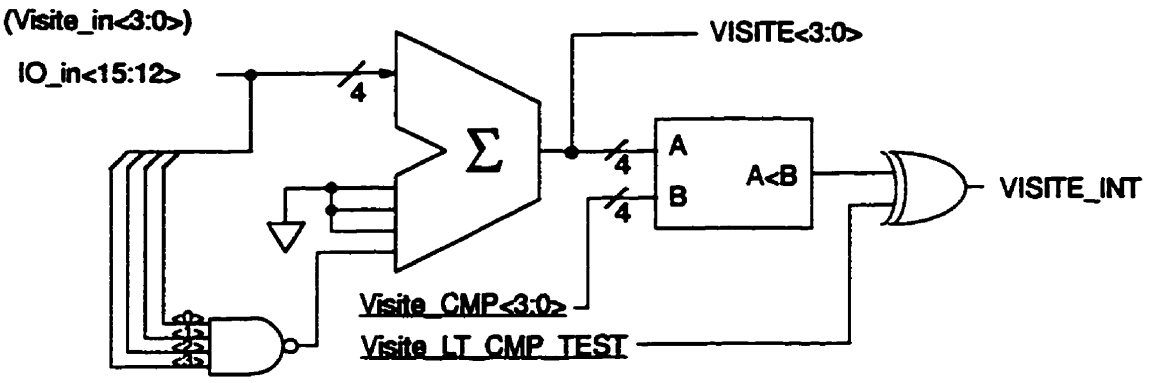


Figure 4.40 *Evaluation de l'achalandage des pixels. Le module VISITE additionne "1" à un champ de la mémoire MAR à chaque fois que le pixel d'intérêt visite un pixel. Le compte est limité à 15 et est laissé inchangé lorsque cette valeur limite est atteinte. Un comparateur permet de générer un signal d'interruption si une certaine valeur est dépassée.*

Tableau 4.8 *Contenu du registre de description d'état (Etat_0<15:0>) mémorisé en mode d'archivage*

Etat_0	Variable	Description
<15:14>	DIR_i<1:0>	Direction de provenance (suite de Etat_1<0>)
<13:11>	OA<2:0>	Orientation d'arête
<10:8>	DIR<2:0>	Direction du prochain déplacement
<7>	MAX_Y	Débordement de la frontière supérieure de la fenêtre d'étude
<6>	MIN_Y	Débordement de la frontière inférieure de la fenêtre d'étude
<5>	MAX_X	Débordement de la frontière droite de la fenêtre d'étude
<4>	MIN_X	Débordement de la frontière gauche de la fenêtre d'étude
<3>	Y=Y0	La position du pixel à la même coordonnée verticale que celle de la position de départ
<2>	X=X0	La position du pixel à la même coordonnée horizontale que celle de la position de départ
<1>	H0	Bit Hystérésis du pixel considéré (pour le filtre sélectionné INPUT_i)
<0>	S0	Bit Signe du pixel considéré (pour le filtre sélectionné INPUT_i)

4.3.14 Description des registres de configuration

Un bon nombre de modules du contrôleur de caméra requièrent la programmation d'une ou de plusieurs variables de configuration. Certaines de ces variables servent à configurer des signaux de commande externes, d'autres servent à choisir des variantes pour les modes de balayage de certaines instructions. Ces variables de configuration correspondent aux signaux dont le nom est souligné tout au long de la section 4.3. Le Tableau 4.9, le Tableau 4.10 et le Tableau 4.11 donnent la description de ces registres de configuration qui doivent être correctement programmés par le système hôte avant l'exécution d'une instruction. On ajoute pour chaque cas la référence au texte et aux Figures où on retrouve la description de l'utilisation de chacune de ces variables. La section qui suit décrit sommairement les circuits placés en périphérie du contrôleur MAR pour l'utilisation adéquate de ces caractéristiques.

Tableau 4.9 Description du registre de contrôle #2.

Control _reg_2	Variable	Description
<6>	PHI2_PATH_SELECT	Ajuste la durée du non-recouvrement des phases de horloge entre $\phi 1$ et $\phi 2$ (plus long si == 0)
<5>	PHI1_PATH_SELECT	Ajuste la durée du non-recouvrement des phases de horloge entre $\phi 2$ et $\phi 1$ (plus long si == 0)
<4>	VISITE_LT_CMP_TEST	Polarité du test d'achalandage: 1-> (plus grand que) (Figure 4.40)
<3:0>	VISITE_CMP<3:0>	Valeur pour la détection d'un maximum d'achalandage (Figure 4.40)

Tableau 4.10 Description du registre de contrôle #1.

Control _reg_1	Variable	Description
<15:13>	INT_BREAK_EN<2:0>	Arrêt temporaire lors d'une interruption (Figure 4.17)
<12>	EXT_BREAK_EN	Arrêt commandé par un signal externe (Figure 4.17)
<11>	BREAK_WITH_CK	Synchronise des signaux d'interruption (Figure 4.17)
<10>	BREAK_ON_STEP	Mode pas a pas (Figure 4.17)
<9>	EXT_LEFT_EN	Force une déviation à gauche par un signal externe (Figure 4.37)
<8>	EXT_RIGHT_EN	Force une déviation à droite par un signal externe (Figure 4.37)
<7>	P2_ETENDU	Mode de dilatation morphologique particulier (non documenté)
<6>	EXT_P2_EN	Utilise un signal externe pour la détection d'arêtes (section 4.3.9)
<5>	ANGLE_EN	Active le mode de déplacement généralisé (Figure 4.37)
<4>	START_LEFT	Définit le sens du premier changement de direction si ANGLE_EN =1 (Figure 4.37)
<3>	P2_OA_EN	Force le code de direction à zéro si P2=0 (Figure 4.25)
<2:0>	EXT_COND_EN<2:0>	Définit la provenance des signaux pour l'exécution conditionnelle (Tableau 4.1)

Tableau 4.11 Description du registre de contrôle #0.

Control _reg_0	Variable	Description
<15>	X_POS_PARITE_POL	Définit la parité des lignes pour le calcul de la position (même que le capteur si == 0)
<14>	PARITE_MEM_INIT	Parité de la position du pixel d'intérêt après l'exécution du dernier "*SET_CAPTEUR 6*"
<13>	Y_MEM_DIR_REF	Polarité du référentiel image verticale (Figure 4.20)
<12>	X_MEM_DIR_REF	Polarité du référentiel image horizontal (Figure 4.20)
<11>	LOGIC_CMP_Y	Calcule de la position verticale d'un pixel en arithmétique non-signée (Figure 4.20)
<10>	LOGIC_CMP_X	Calcule de la position horizontale d'un pixel en arithmétique non-signée (Figure 4.20)
<9>	POS_PTR_U_EN	Réserve le bus MD<15:8> pour propager l'indice de position Y lorsque le système hôte ne fait pas accès aux registre de configuration
<8>	POS_PTR_L_EN	Réserve le bus MD<7:0> pour propager l'indice de position X lorsque le système hôte ne fait pas accès aux registre de configuration
<7>	EXT_OVF_EN	Utilise un signal externe pour définir une fenêtre d'étude superposée à la variable OVERFLOW (Figure 4.21)
<6>	EXT_STOP_EN	Utilise un signal externe pour interrompre l'exécution d'une instruction (Figure 4.19)
<5>	EXT_RESUME_EN	Utilise une variable externe pour relancer l'exécution d'une instruction après l'activation du signal BREAK (Figure 4.17)
<4:3>	ETAT_CMP<4:3>	Utilisé en concaténation avec OA<2:0>. Permet de générer un signal d'interruption (MATCH_ETAT) lorsque l'état interne du PLA a cette valeur.
<2:0>	OA_CMP<2:0>	Génère un signal d'interruption composite (MATCH_OA<2:0>) lorsque l'orientation d'arête à cette valeur.

4.4 Circuits périphériques

Des circuits périphériques au contrôleur MAR sont prévus comme configuration de base afin de définir un système ayant un minimum de fonctionnalité. Cette configuration de base comprend la mémoire MAR de description d'état, un générateur d'histogramme spécialisé et un circuit de conditionnement des signaux d'arêtes. Comme il est discuté à la section 6.3, chaque application spécialisée peut nécessiter l'ajout de circuits périphériques pour une réalisation efficace.

4.4.1 Mémoire MAR de description d'état

Le bloc de mémoire qui est représenté à la Figure 4.1 est prévu initialement pour être de la mémoire statique bien que l'emploi de mémoire dynamique soit possible. Dans ce cas, on utiliserait les signaux externes d'arrêt temporaire (*EXT_BREAK* et *EXT_RESUME*) lors des périodes de rafraîchissement. On peut également utiliser de la mémoire à deux ports pour une plus grande efficacité d'opération. La Figure 4.41 donne le détail des circuits de commande nécessaires pour l'exploitation des différents modes d'accès dans le cas de la mémoire statique.

- Lecture ou écriture de la mémoire MAR par le système hôte (contrôleur en attente, *MEM_CS=1*)
- Lecture ou écriture des registres internes au contrôleur (*CTR_CS=1*)
- Lecture ou écriture de la mémoire MAR par le contrôleur (*CTR_CS=0*, *MEM_CS=0*)

De façon générale, le système hôte ne fait accès au contrôleur que lorsque celui-ci est en mode d'arrêt temporaire (*BREAK = 1*) ou entre l'exécution de deux instructions. On peut par contre annuler l'exécution d'une instruction sans conflit en écrivant à l'adresse *STOP* puisque le bus *D<15:0>* est programmé en écriture lorsque le signal *CTR_CS* est inactif. Avec une configuration de mémoire partagée comme celle de la Figure 4.41, la programmation de la prochaine instruction ne peut être anticipée tel que discuté à la section 4.3.1 car le bus *D<15:0>* adresse alors la mémoire d'état. Seule une instruction de déplacement rapide (*FAST_MOVE*) fait exception à cette règle car le contrôleur n'accède pas à la mémoire MAR lors de ce type de déplacement.

Les équations logiques (3-50) et (3-51) définissent les signaux qui commandent la lecture et l'écriture pour la mémoire MAR. Ces signaux devraient être générés normalement par un circuit de logique programmable (PAL) placé en périphérie.

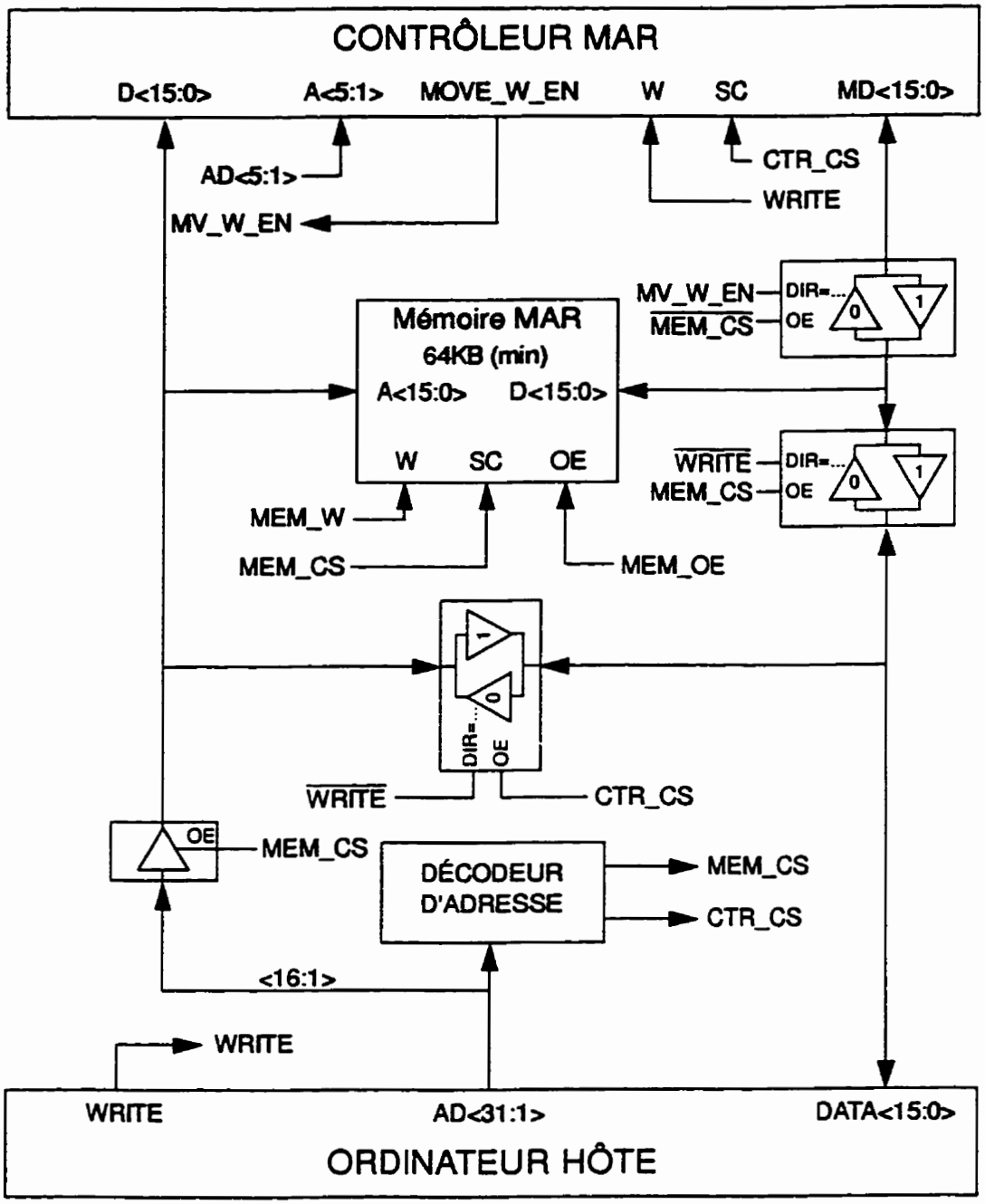


Figure 4.41 Circuits périphériques requis pour la mémoire d'état. L'accès à cette mémoire peut être fait par le contrôleur MAR ou par le système hôte en temps partagé. Lorsque le système hôte n'accède pas à la mémoire MAR ni au contrôleur, le bus D est utilisé comme bus d'adresse pour la mémoire d'état. Les équations (3-50) et (3-51) résument les signaux d'interfaces pour la mémoire.

$$MEM\_W = (MEM\_CS \cdot WRITE) + (\overline{MEM\_CS} \cdot \overline{CTR\_CS} \cdot M\_W\_EN) \quad (3\text{-}50)$$

$$MEM\_OE = (MEM\_CS \cdot \overline{WRITE}) + (\overline{MEM\_CS} \cdot \overline{CTR\_CS} \cdot \overline{M\_W\_EN}) \quad (3\text{-}51)$$

4.4.2 Générateur d'histogrammes spécialisé

Un deuxième circuit périphérique particulièrement important est un générateur d'histogrammes pour plusieurs signaux caractéristiques comme la détection d'arêtes avec les différents filtres. Il serait de plus intéressant de générer une série d'histogrammes pour différentes régions de l'image. La Figure 4.42 donne un aperçu de l'organisation de ces histogrammes pour lesquels on utilise les bits les plus significatifs des indices de position X et Y pour séparer l'image de façon pyramidale. Le programme d'exploitation peut alors connaître rapidement la densité des arêtes pour différentes parties de l'image et ainsi guider la procédure de suivi d'arêtes.

4.4.3 Conditionnement des signaux d'arêtes

Un module spécialisé pour le conditionnement des signaux d'arêtes est pratiquement indispensable pour atteindre un niveau de performance adéquat pour la détection d'arêtes en temps réel. Ce module doit réaliser l'opération de dilatation morphologique sur le signal H tel que discuté à la section 4.3.9 pour obtenir le signal H^*. Un circuit avec une architecture en pipeline peut réaliser ce traitement en imposant un délai constant de deux lignes de pixels par cycle de dilatation. La réalisation d'un tel module est particulièrement justifiable puisque deux à trois traitements morphologiques sur l'ensemble des images filtrées sont nécessaires. Ce type de traitement a l'avantage de fonctionner de façon autonome et ne requiert aucun seuil.

4.5 Réalisation VLSI

On termine ce chapitre par quelques brefs commentaires sur la réalisation VLSI du contrôleur de caméra. On discute de la génération automatique de bloc de logique programmable (PLA), de l'outil automatisé de description schématique, de la génération du circuit selon l'approche de cellules normalisées, de la testabilité d'un tel circuit et de l'historique de la réalisation du contrôleur.

4.5.1 Générateur de PLA

Le PLA est assemblé grâce au compilateur de structure d'une manière similaire à la création du capteur MAR (section 3.4.1) à l'exception du plan ET et du plan OU qui sont des matrices de cellules hétérogènes. La description des matrices hétérogènes est présentée

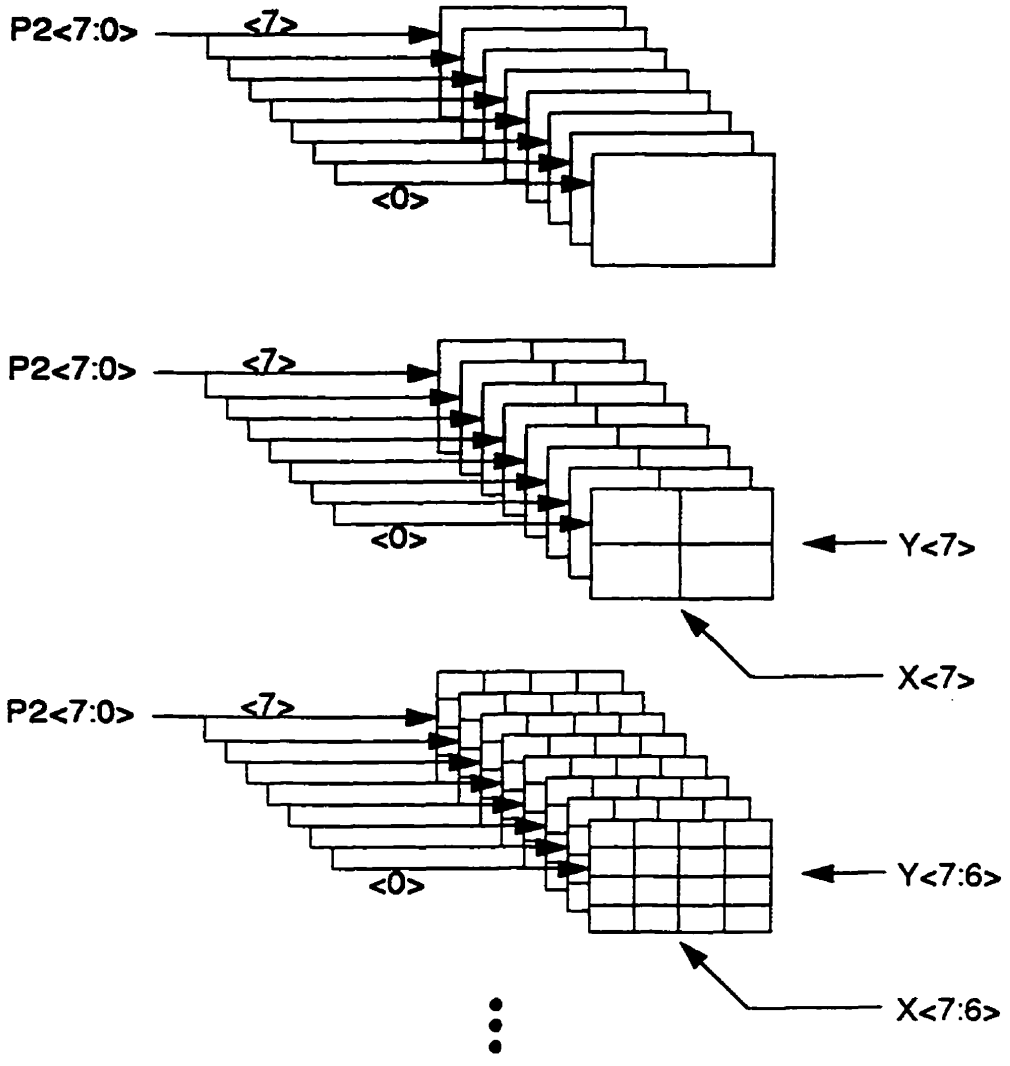


Figure 4.42 Histogrammes pyramidaux. Les bits les plus significatifs de l'indice de position définissent des sous ensembles de l'image. On peut rapidement connaître la densité des arêtes pour chaque région de l'image.

à l'ANNEXE B (section B.2) et la Figure 4.43 donne l'allure de l'arrangement des blocs composant le PLA. On retrouve à l'ANNEXE E (Figure E.10) l'exemple simpliste du dessin des masques d'un PLA réalisé à l'aide du compilateur de structure.

On utilise un style de design dynamique de type domino N avec cycles de précharge et d'évaluation pour le PLA [40]. Le Tableau 4.12 donne la description et la fonction de chaque partie du PLA représenté à la Figure 4.43. Pour la réalisation efficace du PLA, la

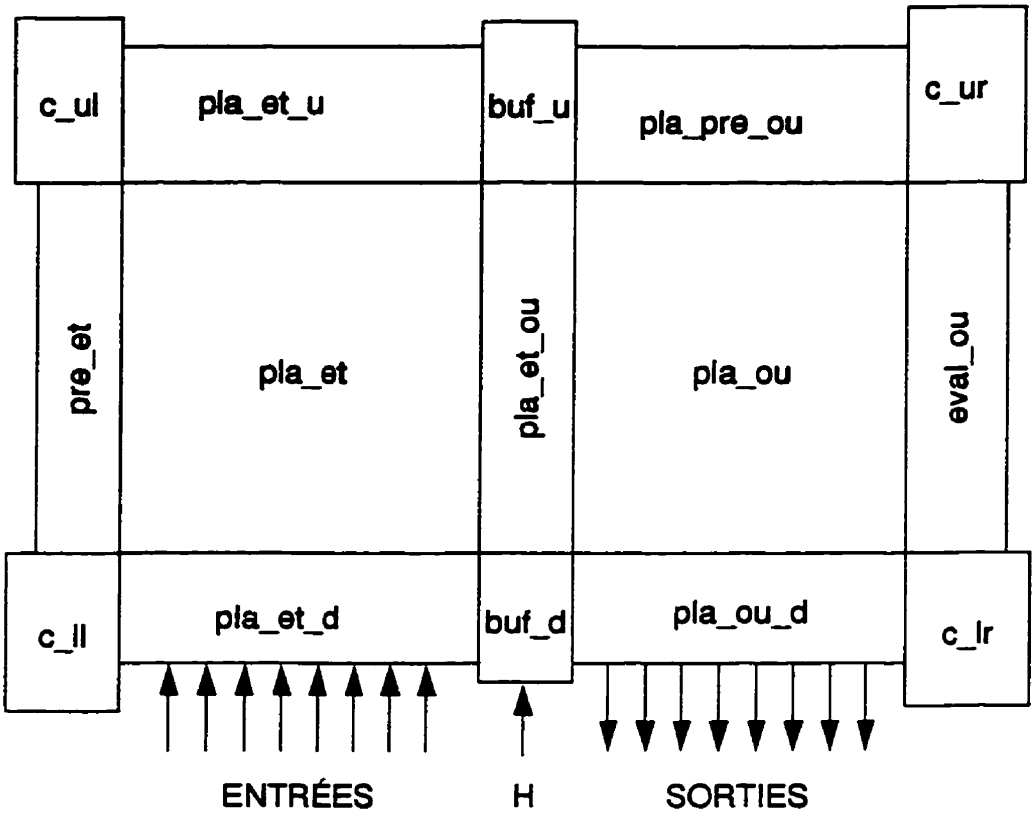


Figure 4.43 Organisation des principaux blocs pour la génération automatique du PLA. Les entrées sélectionnent dans le plan ET quelles lignes sont activées et le plan OU active conséquemment les sorties. Le fonctionnement est synchrone avec l'horloge de base H et le style de design est dynamique de type domino.

liste des lignes du microcode est divisée en trois sections. On réalise trois PLAs indépendants et, à l'aide d'une porte OU CMOS, on recombine chaque trio et on reconstruit ainsi le signal comme s'il ne s'agissait d'un seul grand PLA. Cette modification est requise dû au trop grand nombre de lignes du microcode qui impliquent des délais trop longs pour l'évaluation des noeuds du plan OU (une centaine de transistors en parallèle sur le même noeud). Grâce à cette modification, on peut quadrupler la fréquence d'opération par rapport à un circuit qui utilise un seul grand PLA. Le Tableau 4.12 présente, pour des raisons de simplification, la description d'un PLA unique de 158 lignes.

La structure dynamique et synchrone du design du PLA implique un délai d'un cycle d'horloge entre les entrées et les sorties. On peut donc retourner directement des sorties vers les entrées pour construire une machine à état. Les tests effectués sur le plus récent prototype du contrôleur ont démontré une fréquence maximum d'opération de l'ordre de 17 MHz.

Tableau 4.12 Description des cellules de base du PLA de la Figure 4.43.

cellule de base	# ligne	# colonne	Description
c_ul	1	1	Transistor d'évaluation du plan ET
c_ll	1	1	Transistor d'évaluation du plan ET
c_ur	1	1	Aucune logique
c_lr	1	1	Aucune logique
pla_et	158	18	Matrice hétérogène de sélection des lignes pour un état donné des entrées
pre_et	158	1	Transistors de précharge des lignes du plan ET
pla_et_d	1	18	Inverseurs tampons des entrées non-inversées du PLA
pla_et_u	1	18	Inverseurs tampons des entrées inversées du PLA
pla_et_ou	158	1	Registre qui mémorise les lignes sélectionnées par le plan ET
buf_u	1	1	Inverseurs tampons pour les signaux d'horloges à deux phases
buf_d	1	1	Générateur d'horloges à deux phases sans recouvrement et tampons
pla_ou	158	24	Matrice hétérogène de description du plan OU. Commande d'activation des sorties en fonction des lignes sélectionnées. Le plan OU effectue, pour chaque sortie, un OU câblé entre toutes les lignes sélectionnées par le plan ET.
pla_pre_ou	1	24	Précharge des sorties du PLA
pla_ou_d	1	24	registre dynamique pour les sorties du PLA
eval_ou	158	1	Transistors d'évaluation du plan OU. Les 158 transistors sont en parallèle.

4.5.2 Entrée schématique du circuit

La réalisation VLSI du contrôleur de caméra est faite à l'aide de la technologie CMOS 1.2 µm de Northern Telecom. On utilise une banque de cellules normalisées qui intègre l'ensemble des modules présentés à la section 4.3 à l'intérieur d'une même puce. Cette banque de cellules comprend des portes NOR et NAND à 2,3 et 4 entrées, des inverseurs simples et à sortie haute impédance, des bascules synchrones et transparentes, des plots de contact (pads) d'entrée et de sortie et des multiplexers.

On effectue la description sous forme de dessin schématique en incluant les plots de contact de même que les plots d'alimentation. On doit également créer une représentation schématique pour les macro-cellules (PLA) en nommant, de façon cohérente avec le circuit physique, les points d'entrées et de sorties afin de permettre aux utilitaires de placement et routage automatique d'inclure et de câbler correctement cette partie du circuit. Le dessin schématique est réalisé à partir du logiciel CADENCE selon un format segmenté en plusieurs plans dû à la grande dimension du design. Le Tableau 4.13 donne un aperçu du

Tableau 4.13 Liste des dimensions et des propriétés du circuit du contrôleur MAR

Propriété pour le contrôleur MAR	Valeur
Nombre de rangées de cellules standards	27
Largeur de la partie centrale	6036.63 µ
Hauteur de la partie centrale	6036.63 µµ
Largeur du circuit complet (Incluant les plots de contact)	7592.6 µ
Hauteur du circuit complet (Incluant les plots de contact)	7592.6 µ
Hauteur des cellules standards	75 µ
Longueur totale des cellules standards	141656 µ
Nombre total de cellules standards dans le design	4137
Nombre total de macro-cellules dans le design	3
Nombre total de plots de contacts dans le design	68
Nombre total de points de connexions utilisés dans le design	13210
Nombre total de noeuds dans le design	3328
Nombre total de connexions dans le design	9882
Surface totale des cellules standards	1.06242e+09 μ^2
Surface totale des macro-cellules	2.4282e+08 μ^2

nombre de cellules, noeuds et connexions du design complet lorsqu'il est prêt pour le placement et le routage automatique. On trouve également à l'ANNEXE E (Figure E.11) une vue très générale de l'ensemble des plans de description schématique du contrôleur MAR.

4.5.3 Placement et routage automatique

Une fois que la description du circuit est complétée, que toutes les cellules du circuit ont une représentation "layout" et "abstract" et qu'on a effectué une extraction du design schématique sans aucune erreur, on peut passer aux étapes de design automatisé qui complètent le dessin des masque du circuit. On commence par placer manuellement les macro-cellules de même que les plots de contacts. Un fichier de texte précise l'emplacement désiré pour chacun d'eux. Le placement des plots de contact peut être automatique mais il est avantageux d'imposer un ordre préalable aux nombreuses broches du circuit. On peut de plus forcer le placement des plots de contact des alimentation à des coins diamétralement opposés afin de forcer le routage de ces signaux critiques à la manière de deux peignes imbriqués et ainsi limiter les croisements.

On invoque par la suite l'utilitaire de placement automatique. Cette procédure, qui requiert quelques heures de traitement pour un circuit de cette taille (sur un SUN 3), procède au placement des cellules standards en minimisant la longueur des interconnexions et en uniformisant l'achalandage des canaux de routage. On génère par la suite les canaux de routage et on procède au routage global et ensuite au routage détaillé. Les étapes de routage nécessitent en tout 5 à 6 heures de traitement. On a choisi un routage compact à 45° vu la haute densité des canaux de routages et le grand nombre de cellules standards. La Figure E.8 de l'ANNEXE E présente le dessin des masques du contrôleur MAR avec ses 68 plots de contact et ses trois modules de PLA placés à l'extrémité supérieure du circuit.

4.5.4 Testabilité

Le contrôleur est conçu afin de pouvoir valider rapidement son fonctionnement. Les sélecteurs de données du bus *IO<7:0>* donnent accès à toutes les variables internes propres aux circuits séquentiels afin de détecter rapidement un mauvais fonctionnement de la logique qui n'est pas directement accessible. Le bloc de logique programmable (PLA) est testé en exécutant tour à tour les principales instructions et en commandant les signaux importants comme la détection d'arêtes ou la détection de la frontière d'étude par des canaux externes. On peut ainsi obtenir rapidement un diagnostique du circuit en

développant un programme d'exploitation spécifiquement pour le test. Le test du contrôleur MAR est complètement indépendant du capteur et se fait entièrement de façon numérique.

4.5.5 Historique de la réalisation VLSI

Le premier circuit intégré pour la réalisation du contrôleur a été développé à l'hiver 1990 à l'aide de la technologie CMOS 3 µm et comprenait uniquement le bloc de logique programmable (PLA). Le premier circuit fonctionnel a été fabriqué à l'été 1990 et a été testé avec succès à la fin de l'automne 1990. Il comprenait, outre le PLA, l'unité arithmétique de direction et le module d'orientation d'arête. On a immédiatement greffé ces trois sous-modules à un ensemble de composantes discrètes et de PAL en logique bipolaire (TTL) pour créer le premier prototype du contrôleur MAR. Les plans de ce circuit sont d'ailleurs présentés à l'ANNEXE F.

La banque de cellules requise par le générateur de PLA a été convertie à la technologie CMOS 1.2 µm au début de l'année 1991 et le circuit complet, tel que décrit à la section 4.3, a été soumis pour la fabrication à la fin de l'hiver 1991. Des tests effectués durant l'été 1991 (ANNEXE D) ont démontré le fonctionnement partiel du circuit. Les problèmes qui étaient relativement simples à corriger ont été résolus au début de l'automne 1991. Une nouvelle version du contrôleur MAR est présentement en cours de test et le prototype du circuit sera disponible sous peu pour se substituer à la version réalisée à l'aide de composantes discrètes.

4.5.6 Description des plots de contact

La description des plots de contact du contrôleur MAR est présentée à l'ANNEXE D (Figure D.1). On remarque que les bus *D<15:0>* et *MD<15:0>* utilisent chacun une rangée de broches latérales au circuit. On a pris soin de regrouper les signaux d'accès mémoire, le bus *IO<7:0>* et surtout les entrées analogiques qui sont bordées par des broches d'alimentation pour limiter le bruit numérique.

Ce chapitre termine la description formelle des parties du système MAR réalisées à l'aide de la technologie VLSI. On présente au prochain chapitre les détails relatifs au module de calcul analogique pour l'opération de filtrage spatial ainsi qu'un survol des simulations réalisées.

CHAPITRE 5

TRAITEMENT ANALOGIQUE ET SIMULATIONS

Ce chapitre présente les détails de la réalisation des filtres analogiques et la simulation des principales parties du système MAR. Plusieurs points relatifs au traitement analogique ont déjà été traités aux chapitres 3 et 4. Les sections qui suivent fusionnent cette information et précisent certaines particularités de ce type de circuit. On présente également des variantes possibles pour la réalisation efficace d'un opérateur analogique de convolution ainsi que la configuration choisie lors de la réalisation du premier prototype du système MAR.

Le modèle proposé pour le capteur MAR a été simulé, selon une grille hexagonale, pour faire la preuve que ce type de capteur spécialisé effectue la détection d'arêtes même avec un masque qui sous-échantillonne fortement l'image. La partie numérique du contrôleur a également fait l'objet de simulations pour la mise au point du microcode lors de la définition du jeu d'instructions. Un bon nombre de simulations effectuées à l'aide du logiciel SPICE ont permis de valider le fonctionnement des modules de base des circuits intégrés qui ont été fabriqués en plus d'estimer la vitesse de fonctionnement de certain d'entre eux.

5.1 Opérateur analogique de convolution

Le capteur MAR fournit une multitude de signaux analogiques relatifs à l'illuminance d'une région de pixels tel que décrit à la section 3.3.1. Chaque signal en courant provenant des sorties analogiques du capteur $I_r(\delta)$ est converti en tension $V_r(\delta) = R_g I_r(\delta)$ comme montré à la Figure 5.1. On dirige l'ensemble de ces signaux en

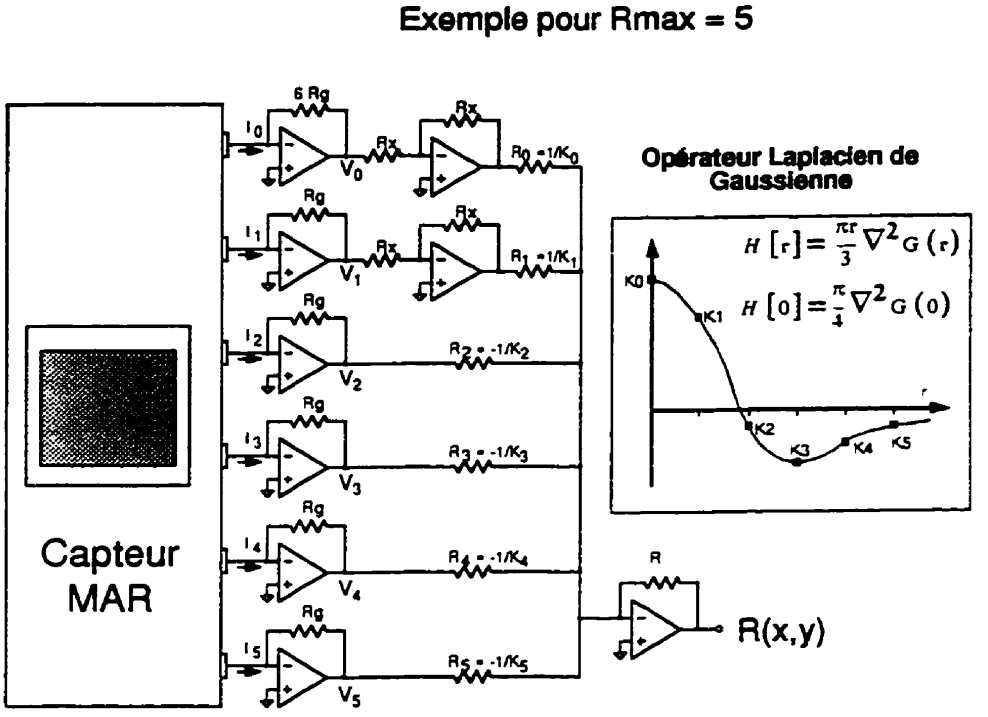


Figure 5.1 Opérateur analogique de convolution. Les signaux en courant provenant du capteur sont convertis en tension par la première colonne d'amplificateurs opérationnels. Certains canaux sont inversés pour tenir compte du signe des coefficients de la somme de produits qui est réalisée par un réseau de résistances.

tension vers un sommateur dont les résistances d'entrées sont fixées relativement à la valeur échantillonnée du filtre qu'on désire appliquer à l'image originale. Ce sommateur effectue le filtrage désiré du signal représentant l'image (V) au point d'intérêt avec un opérateur donné (H) comme montré à l'équation (5-1).

$$R[x,y] = R_g\{H[0]V_0 + \sum_{r=1}^{15}\sum_{\delta=1}^{6} H[r]V_r(\delta)\} \qquad (5\text{-}1)$$

On doit cependant respecter le signe négatif de certains coefficients $H[r]$ lors de la somme pondérée. La Figure 5.1 montre une solution qui consiste à inverser les signaux dont les coefficients sont de même signe. Puisque plusieurs filtres utilisent les mêmes signaux en tension, on doit prévoir l'inversion de la presque totalité de ceux-ci. Une deuxième solution nous permet d'éliminer cette série d'inverseurs. On peut ajouter une composante continue commune aux coefficients de $H[r]$ de façon à les rendre tous positifs pour ensuite effectuer la correction à l'aide d'un signal unique. La Figure 5.2 montre

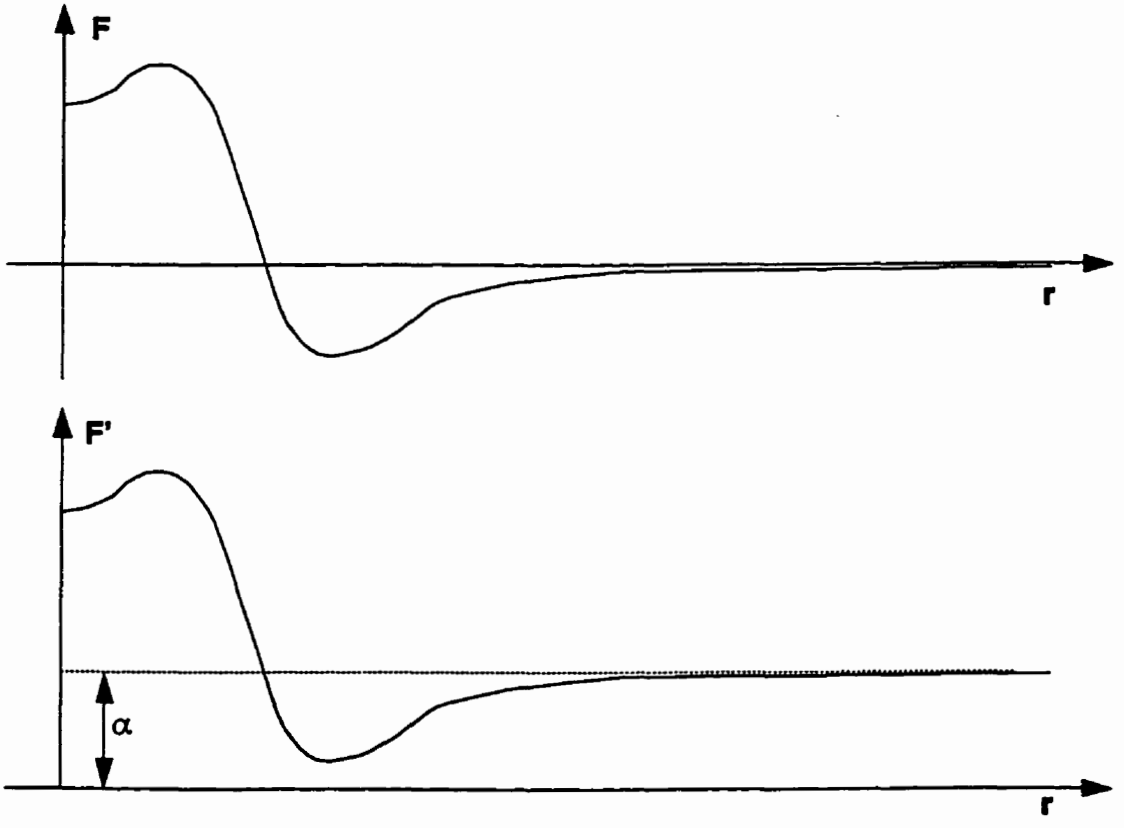


***Figure 5.2** Ajout d'une composante continue au filtre pour rendre l'ensemble des coefficients de même signe. L'équation (5-3) démontre qu'on doit soustraire la somme des signaux de l'image pour retrouver le résultat équivalent.*

graphiquement comment on rend l'ensemble des coefficients positifs en leur ajoutant une valeur commune α et ainsi obtenir $H'[r]$. Dans ce cas, le produit de convolution présenté à l'équation (5-1) se récrit alors:

$$R[x,y] = R_g\{(H'(0)-\alpha)V_0 + \sum_{r=1}^{15}\sum_{\delta=1}^{6}(H'[r]-\alpha)V_r(\delta)\} \qquad (5\text{-}2)$$

ou encore:

$$R[x,y] = R_g\left[\{H'[0]V_0 + \sum_{r=1}^{15}\sum_{\delta=1}^{6}H'[r]V_r(\delta)\} - \alpha\{V_0 + \sum_{r=1}^{15}\sum_{\delta=1}^{6}V_r(\delta)\}\right] \qquad (5\text{-}3)$$

Le terme de droite de l'équation (5-3) correspond à la somme de l'ensemble des signaux primaires et est très facilement réalisable à l'aide d'un simple sommateur. Cette approche a l'avantage de combiner la calibration du zéro du filtre en modifiant la valeur du coefficient α. La calibration des filtres fait d'ailleurs l'objet de la section 5.1.3.

5.1.1 Circuit à miroirs de courant

Une approche par sommation de courants est possible pour réaliser la convolution de façon analogique. On peut concevoir cette architecture aussi bien à l'aide de la technologie CMOS qu'avec la technologie Bipolaire. La Figure 5.3 montre l'arrangement de plusieurs blocs de miroirs de courant servant à calculer le produit de convolution par sommation de courants. L'exemple qui est montré utilise des transistors Bipolaires mais un arrangement similaire est réalisable avec des transistors à effet de champ.

Cette approche est désavantageuse si on veut réaliser le filtrage à plusieurs résolutions différentes. En effet, le courant de base qui polarise chaque transistor est fourni par le courant d'entrée et constitue une forme de distorsion. La génération d'un grand nombre de copies du signal source implique inévitablement une erreur d'évaluation qui sera de plus fonction de la taille de chacun d'eux. Un autre problème est associé à la variation des paramètres d'un transistor bipolaire. Le gain relatif d'un transistor bipolaire, qui est fixé par la dimension physique de ce dernier, peut difficilement être ajusté uniquement par les paramètres géométriques du dessin des masques. L'utilisation d'un tel circuit est donc moins intéressant du point de vue de la calibration.

La technologie CMOS permet une plus grande uniformité des propriétés électroniques des transistors d'un tel circuit mais le temps de stabilisation est relativement

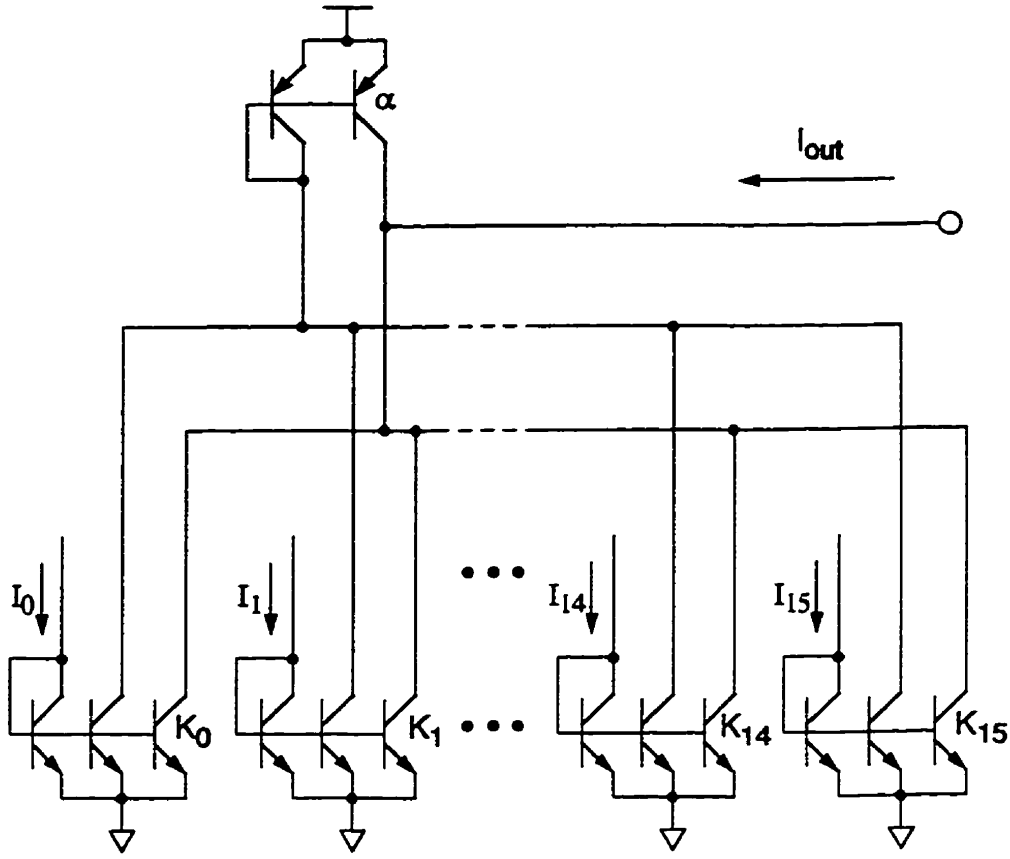


Figure 5.3 Circuit à miroir de courant pour le calcul analogique du produit de convolution. Le courant de sortie Iout est proportionnel à la somme pondérée des signaux d'entrée d'une façon similaire à l'équation (5-3). L'ajustement des coefficients est effectué en variant la dimension des différents transistors.

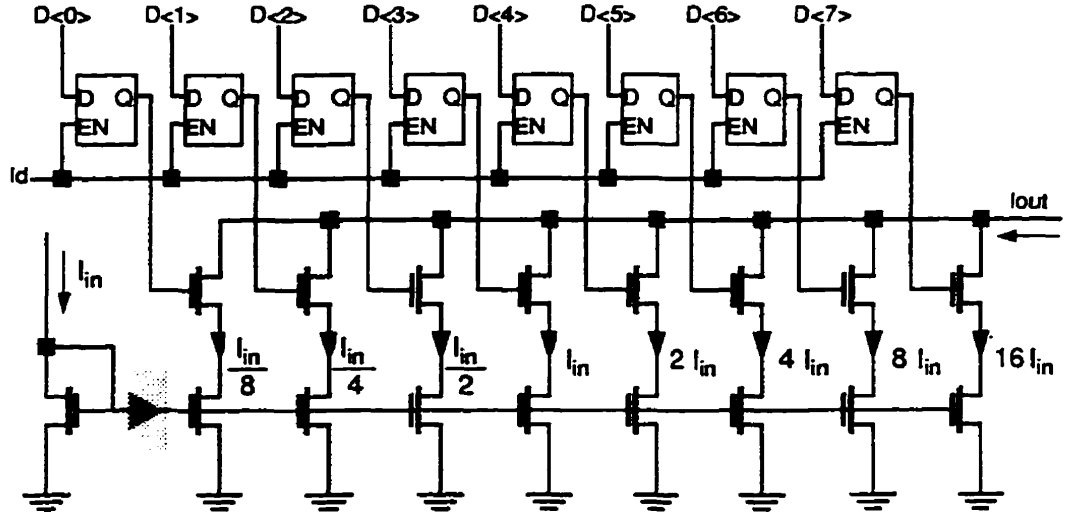
long dû à la forte charge capacitive que représentent les nombreuses grilles en parallèle compte tenu de la faible amplitude du courant de source. L'ANNEXE H présente d'ailleurs les performances d'un circuit exploratoire basé sur le miroir de courant et dont les coefficients sont programmables numériquement. La section suivante décrit le détail de la réalisation du circuit de l'ANNEXE H qui utilise la technologie CMOS.

5.1.2 Contrôle numérique des coefficients

Puisqu'il est possible d'appliquer plusieurs filtres sur l'image, on envisage l'utilisation de certains filtres dont les coefficients sont fixés de façon matérielle comme c'est le cas pour l'exemple de la Figure 5.1, alors que d'autres pourront être choisis dynamiquement selon l'application. Dans ce cas, on doit pouvoir programmer, grâce à une

variable numérique, les poids des coefficients d'un filtre donné. Un prototype de circuit à miroirs de courant avec contrôle numérique des coefficients a été fabriqué à l'aide de la technologie CMOS 3 µm. Ce circuit a prouvé qu'il peut fonctionner à une fréquence d'environ 200 KHz et le détail du fonctionnement de ce circuit de même que des tests qui ont été effectués sont présentés à l'ANNEXE H.

Chaque signal source est dirigé vers un circuit à miroirs de courant comme celui de la Figure 5.4. Le courant de sortie de chaque module (I_{out}) correspond à la somme de



$$I_{out} = I_{in}(16D_7 + 8D_6 + 4D_5 + 2D_4 + D_3 + \frac{D_2}{2} + \frac{D_1}{4} + \frac{D_0}{8})$$

Figure 5.4 Contrôle numérique des coefficients pour un circuit de convolution analogique utilisant une structure avec miroir de courant. Le registre de configuration active certains transistors et bloque les autres. Le courant de sortie I_{out} est alors fonction du produit entre l'entrée analogique I_{in} et le coefficient numérique D<7:0>.

plusieurs sources de courant dont la valeur croît selon un facteur deux. Un transistor en mode de commutation est placé en série dans chaque branche pour l'inclure ou non au signal de sortie. On place en parallèle plusieurs de ces modules afin d'obtenir la fonction somme de produits. L'ajout d'un amplificateur opérationnel en suiveur de tension dans la région ombragée de la Figure 5.4 pourrait améliorer grandement les performances

dynamiques du circuit car il permet de diminuer la charge capacitive du signal d'entrée et donc de diminuer le temps de stabilisation requis pour passer d'un état à l'autre. L'utilisation de la technologie BiCMOS pourrait probablement améliorer la performance de ce type de circuit en utilisant un arrangement hybride de transistors MOS pour la partie numérique et Bipolaires pour le miroir de courant.

5.1.3 Calibration des filtres

La calibration des filtres est un point important car, peu importe la solution retenue pour l'opérateur analogique de convolution, la tolérance des composantes utilisées implique forcément une variation de la réponse du filtre par rapport à sa valeur théorique. Un circuit et une procédure de calibration doivent être prévus pour chaque filtre analogique. Dans le cas de l'opérateur $\nabla^2 G_\sigma(r)$, la procédure de calibration consiste à garantir une réponse nulle lorsqu'on applique le filtre sur une région d'iso-illuminance et ce, quelle que soit l'intensité du signal.

Différents cas de calibration sont illustrés à la Figure 5.5 pour un opérateur laplacien de gaussienne. Lorsque la composante continue du filtre est trop élevée (a) ou trop basse (c), le résultat de la convolution n'est pas nul dans des régions d'iso-illuminance pour des valeurs différentes de l'image source. On remarque que le cas (b) correspond à une calibration parfaite. On utilise une image source à haut contraste pour effectuer cet ajustement comme par exemple un demi-plan noir sur un fond blanc. Dans le cas où l'ensemble des coefficients est positif comme discuté à la section 5.1, l'ajustement de la composante continue α de l'équation (5-3) est utilisée pour calibrer le filtre.

5.2 Acquisition numérique des données

La détection d'arêtes effectuée par le système MAR (section 4.3.9) ne nécessite pas l'image numérisée ni celle de la sortie des filtres. Le premier prototype de la caméra a d'ailleurs permis d'effectuer l'extraction multi-résolution des arêtes d'une scène sans qu'aucun convertisseur analogique/numérique n'y soit inclus. Il est cependant impératif d'intégrer un ou plusieurs convertisseurs analogiques/numériques pour faire l'acquisition de l'image d'illuminance (I_0) ou de la sortie des filtres à des fins d'affichage et de traitement ultérieur. L'amplitude du résultat de la convolution de l'image avec des opérateurs fins (arêtes à haute résolution) est essentiel pour l'évaluation sub-pixel de la position des passages par zéro dont on fait la description à la section 6.3.2.

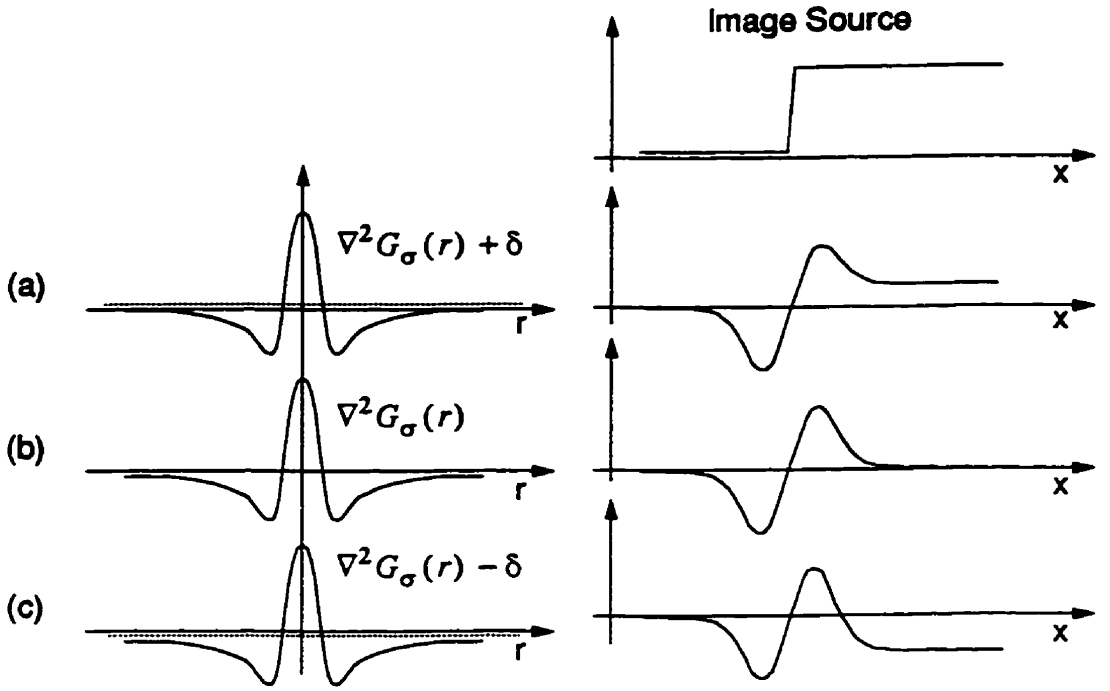


Figure 5.5 Calibration des opérateurs laplacien de gaussienne. Une variation de la composante continue du filtre permet d'ajuster le zéro du résultat de la convolution dans les régions d'iso-illuminance. On utilise typiquement une scène moitié blanche, moitié noire pour calibrer les filtres analogiques. On représente le cas où la partie positive du filtre est prédominante (a), celui où c'est la partie négative qui est trop imposante (c) et un filtre parfaitement calibré (b).

L'acquisition numérique implique l'ajout d'un bloc de mémoire et d'un ou plusieurs convertisseurs A/N en périphérie du contrôleur MAR. L'adressage de la mémoire se fait en écriture grâce au bus *D<15:0>* d'une façon similaire à la mémoire de description d'état. Le signal de déplacement du pixel d'intérêt *MOVE_*CAPT est utilisé comme signal d'activation d'une conversion. Le même signal sert à commander une écriture à la mémoire d'acquisitions. On pourrait de plus inclure un multiplexer analogique à l'entrée de chaque convertisseur A/N pour rendre programmable la sélection du canal analogique dont on veut faire l'acquisition.

5.3 Simulation de l'opérateur de convolution

La première étape du développement d'un système de traitement d'images au plan focal consiste à vérifier, par simulation, l'ensemble des étapes de traitement pour anticiper les résultats du prototype expérimental. La simulation du système MAR a été réalisée par un programme en langage C sur un poste de travail SUN. L'ANNEXE I présente les sources

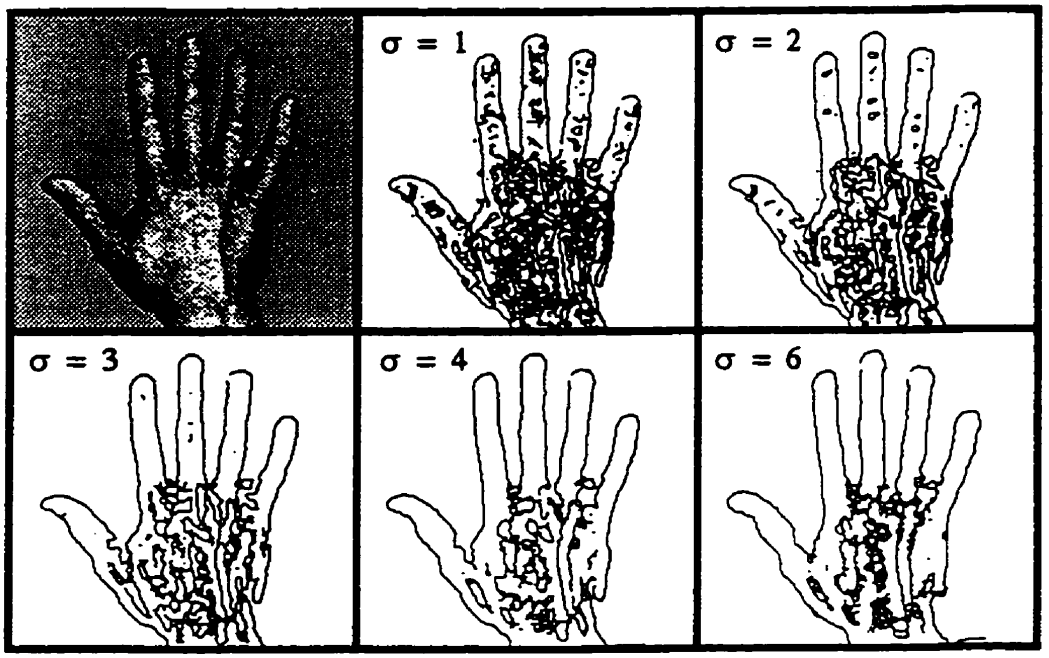
de ce programme de simulation de l'opérateur de convolution selon la philosophie du système MAR. Les images sources, en topologie cartésienne, sont d'abord converties selon une grille hexagonale (section I.1) et l'opération de filtrage est simulée en respectant le masque de convolution du capteur MAR (section I.2). On simule également l'algorithme de suivi d'arêtes (section I.3) et on utilise une procédure spéciale pour déplacer le pixel d'intérêt en topologie hexagonale de façon cohérente au code de direction compris entre 1 et 6 (section I.4). Le programme principal de simulation est finalement présenté à la section I.7. On montre à la Figure 5.6 le résultat de la simulation effectuée sur deux images différentes et ce, pour 5 niveaux de résolution spatiale.

On remarque, pour les filtres grossiers, les effets du sous-échantillonnage spatial. Le système peut en effet détecter certains points d'arêtes à basse résolution alors qu'aucune détection n'est faite par des filtres plus fins à cet endroit. Il est important de préciser que les images de la Figure 5.6 sont représentées sur une grille cartésienne alors que les résultats d'arêtes sont calculés en topologie hexagonale. Il en découle donc une distorsion à l'affichage puisqu'aucun travail n'avait été fait, au moment de la simulation de l'opérateur de convolution, sur la façon de représenter correctement des images d'arêtes en topologie hexagonale.

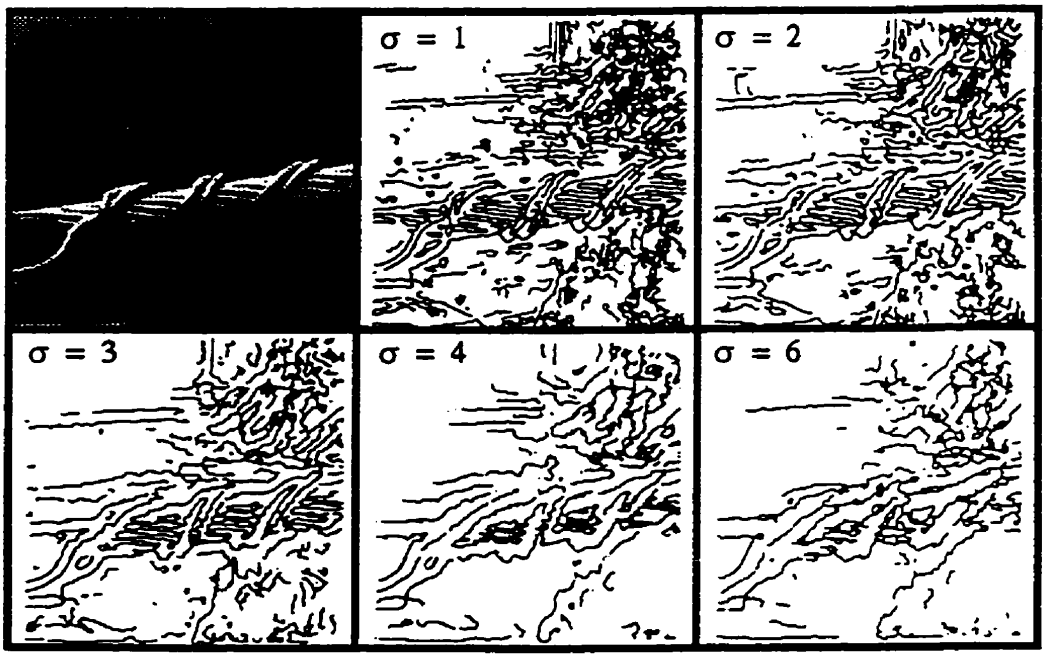
5.4 Analyse de Fourier du masque de convolution

L'analyse de Fourier du masque de convolution du capteur MAR révèle certaines caractéristiques propres à ce type d'opérateur. On doit s'attendre à voir apparaître des composantes à haute fréquence pour l'opérateur MAR comparativement au masque complet du laplacien de gaussienne. La Figure 5.7 présente la transformée de Fourier pour trois différents masques de convolution. L'image d'intensité de l'opérateur laplacien de gaussienne est présentée à la Figure 5.7 (a) pour une valeur de σ=5. Sa transformée de Fourier se retrouve en (b). On remarque que l'opérateur est bien un filtre passe bande et que, sur l'image (b), la composante continue tout comme les hautes fréquences sont atténuées complètement.

La Figure 5.7 (c) est le représentation 2D du masque de convolution du capteur MAR. On remarque que la valeur des coefficients correspond bien à $r\left(\nabla^2 G_\sigma(r)\right)$ puisque la valeur du pixel central est plus faible que celle des rayons 1 à 4. Cette caractéristique est d'ailleurs représentée par la courbe de la Figure 5.2. La Figure 5.7 (d) nous montre la transformée de Fourier du masque de convolution du capteur où l'on peut voir les effets du sous-échantillonnage spatial par la présence de composantes hautes fréquence dans les orientation définies entre deux diagonales du filtre MAR. Comme on pouvait s'y attendre,



(a)



(b)

***Figure 5.6** Résultats de simulation pour deux images. Les images d'arêtes détectées par l'opérateur du capteur MAR sont présentées selon plusieurs résolutions spatiales. On remarque que certains détails sont visibles à haute résolution (σ petit) mais disparaissent à basse résolution (σ grand). On peut également voir les effets du sous-échantillonnage*

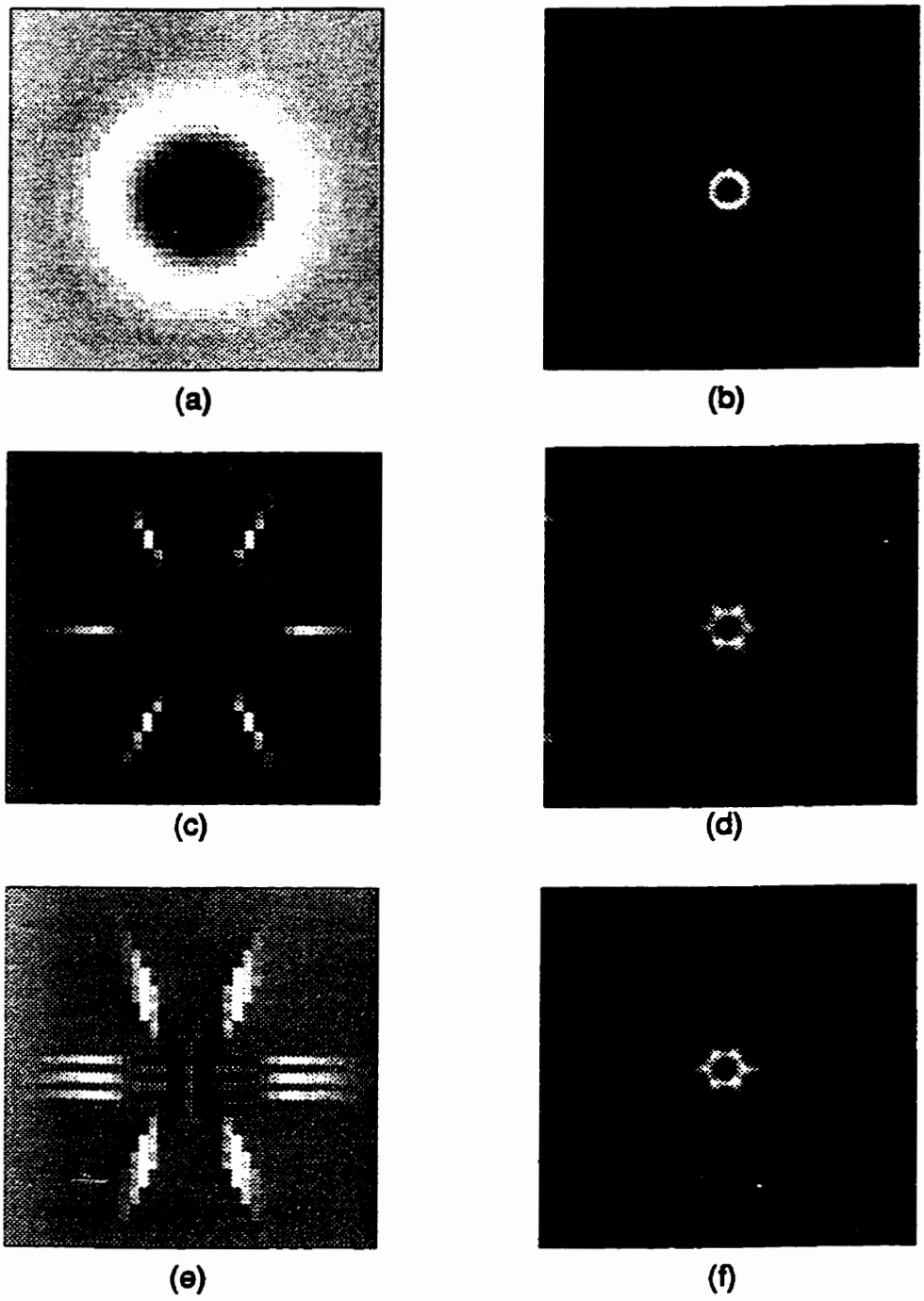


***Figure** 5.7 Analyse de Fourier de l'opérateur de convolution du capteur MAR. On voit en (a) l'image d'intensité de l'opérateur laplacien de gaussienne avec σ=5 et sa transformée de Fourier en (b). Le masque de convolution du capteur MAR est montré en (c) et sa transformée de Fourier en (d). On peut voir l'effet d'atténuation des hautes fréquences dans le cas du masque de convolution à plusieurs pixels d'intérêt en (e) et (f).*

** Reproduit avec la permission de Florent Parent*

la composante continue est parfaitement atténuée et une bonne partie du beigne central (passe bande) est présente.

Les images (e) et (f) de la Figure 5.7 permettent de faire la comparaison entre un masque de convolution simple (c) et un masque à plusieurs pixels d'intérêt (e). Dans ce dernier cas, la transformée de Fourier (f) montre clairement qu'une plus grande partie des composantes aux hautes fréquences est atténuée.

5.5 Simulation numérique du contrôleur MAR

Un bon nombre de simulations numériques ont été effectuées afin de valider le microcode lors de sa conception ainsi que le design logique des principaux modules du contrôleur. Ces simulations ont permis également de vérifier le fonctionnement de l'algorithme de suivi d'arêtes. Ces deux étapes de vérification du design du contrôleur de caméra ont été réalisés par Yannick Tremblay à l'été 1989 [45] et sont brièvement résumés ici.

Le design de base du contrôleur a été complètement décrit dans un langage de description matérielle pour des fins de simulation numérique à l'intérieur du logiciel ENDOT. Cette description de base comprenait le PLA en entier, l'unité arithmétique de direction, le générateur d'adresses, le module d'arbitration des instructions et la logique d'exécution conditionnelle. Ces simulations ont permis de corriger plusieurs détails du microcode tout en validant le reste de la logique d'arbitrage et de contrôle. L'ensemble du jeu d'instructions a été vérifié sur des images génériques de 8 x 8 pixels pour assurer le bon fonctionnement du PLA une fois fabriqué à l'aide de la technologie CMOS.

5.6 Émulateur MAR sur station graphique SUN

Une partie du projet de doctorat entrepris par Florent Parent consiste à développer un émulateur logiciel sur station graphique SUN du système MAR. Cet outil permet de développer le logiciel d'exploitation de la caméra de façon interactive tout en visualisant la progression du balayage. La Figure 5.8 donne une vue d'ensemble de la fenêtre graphique de l'émulateur du système MAR. On peut à l'aide de cet environnement mémoriser la séquence des instructions effectuées afin de la reproduire ultérieurement sur la vraie caméra. On devrait également pouvoir y exécuter un programme d'exploitation dont le déroulement est conditionné par les caractéristiques visuelles extraites de l'image tel que discuté à la section 2.2. L'émulateur logiciel du système MAR est présentement utilisé pour développer un algorithme de suivi d'arêtes selon plusieurs résolutions spatiales en vue de créer une structure de données hiérarchisée à partir de l'information extraite de la scène.

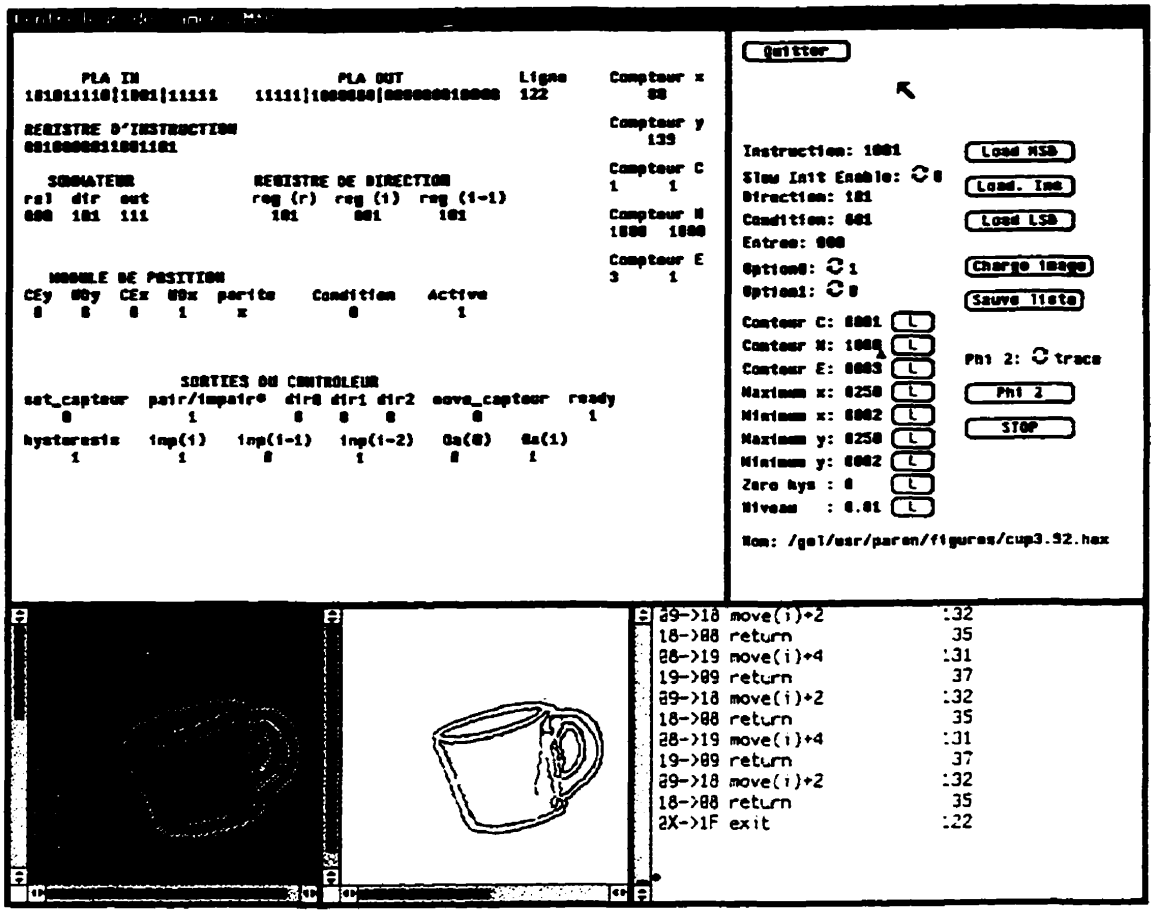


Figure 5.8 Vue d'ensemble de la fenêtre graphique de l'émulateur logiciel du système MAR. On programme le registre d'instructions à l'aide de menus et la progression du balayage de l'image est affiché en bas. On peut également observer une trace de l'exécution du microcode.

** Reproduit avec la permission de Florent Parent*

Le prochain et dernier chapitre décrit le prototype #2 de la caméra MAR (256 x 256 pixels) et présente les résultats obtenus à l'aide du prototype #1 (128 x 128 pixels). On y présente également une vue globale des applications typiques qui pourraient tirer profit de cette caméra à traitement intégré au plan focal et à topologie hexagonale.

CHAPITRE 6

MONTAGE EXPÉRIMENTAL, RÉSULTATS ET APPLICATIONS

6.1 Prototype de la caméra MAR

Le premier prototype du capteur MAR a d'abord été testé sur un banc d'essai pour photo-capteurs spécialisés. C'est un montage qui permet de vérifier rapidement le fonctionnement d'un circuit intégré photo-sensible. Ce même capteur est présentement monté dans un boîtier spécialement conçu pour les besoins spécifiques du système MAR. Cette section décrit globalement l'environnement de la mise en fonction du premier capteur MAR ainsi que la description sommaire de la version 2 de la caméra qui est présentement en cours de réalisation.

6.1.1 Banc d'essai pour photo-capteurs spécialisés

Il est intéressant de prévoir un montage pour tester les circuits photo-sensibles sans à avoir à investir un trop grand effort d'assemblage et de calibrage. Un agrandisseur photographique modifié est utilisé pour générer des images au plan focal à l'aide de négatifs 35 mm. Ce montage, qu'on peut voir à la Figure 6.1, facilite les tests qu'on fait subir à un photo-capteur en le connectant simplement sur une plaquette de montage. Les

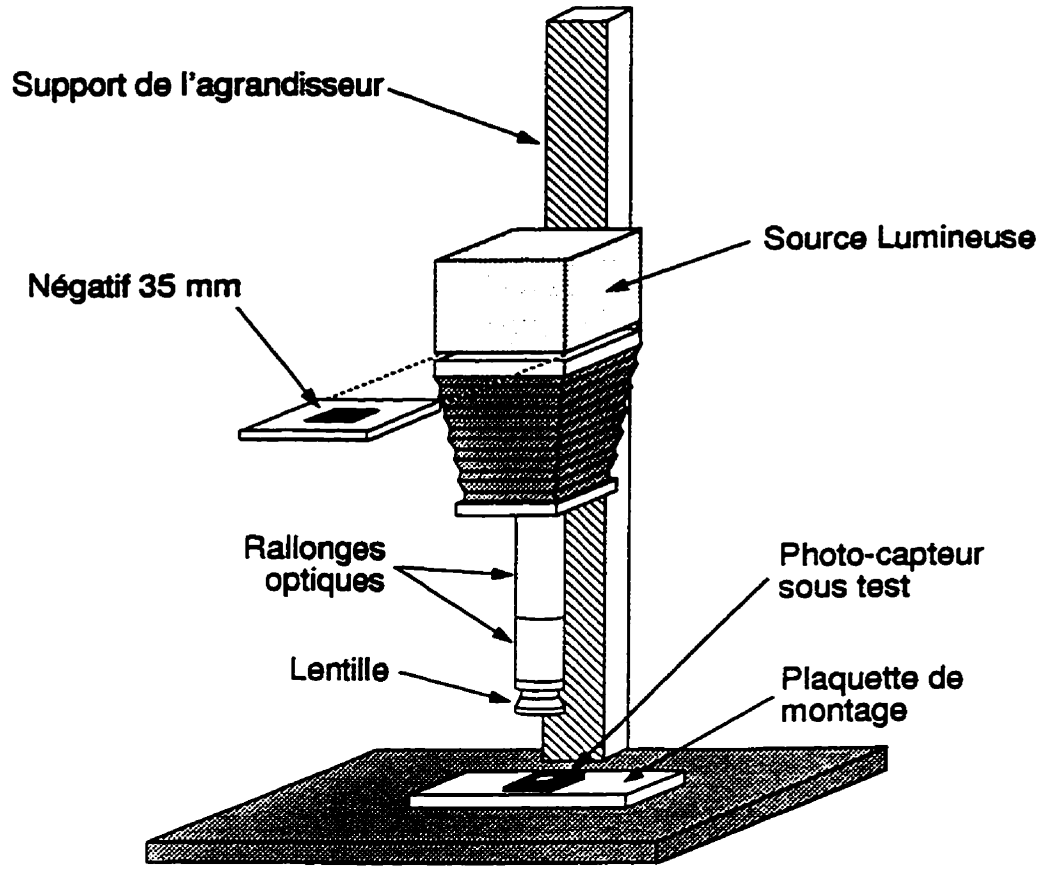


Figure 6.1 Banc d'essai pour photo-capteur spécialisé. On utilise un agrandisseur photographique auquel on ajoute des rallonges optiques pour focaliser l'image d'un négatif 35 mm sur un carré d'environ 5 mm de côté. Le capteur sous analyse est placé simplement sur la base de l'agrandisseur

agrandisseurs photographiques conventionnels sont conçus pour produire une image entre 4 et 60 cm de côté à partir d'un négatif de 35 mm. On a donc fabriqué des rallonges optiques qu'on a placées au niveau du support de la lentille afin de diminuer la taille des images focalisées. On a ainsi obtenu des images aussi petites que 5mm de côté, ce qui correspond à la taille typique des photo-capteurs.

On a effectué les premiers tests à l'aide du montage de la Figure 6.1 en utilisant des négatifs de visages humains. Les résultats obtenus à l'aide de ce montage sont montrés à la Figure 6.5. La qualité de la formation des images n'est cependant pas optimale car elle découle de la qualité et de l'ajustement de l'appareil photographique d'où proviennent les négatifs et de l'ajustement au foyer de l'agrandisseur. On a cependant réussi à obtenir des résultats très satisfaisants avec ce montage.

6.1.2 Composantes mécaniques de la caméra

Après avoir prouvé le fonctionnement du capteur MAR, on a fabriqué les pièces mécaniques requises pour le boîtier de la caméra de même que les circuit imprimés servant à connecter le capteur MAR et les composantes du traitement analogique. Une première version du système a été réalisée selon le procédé de fils discrets enroulés ("wire wrap"). Le circuit analogique des filtres était externe au boîtier et était câblé sur une plaquette de montage. C'est d'ailleurs ce premier prototype utilisant le capteur de 128 x 128 pixels qui a servi à générer les premières acquisitions d'images hexagonales.

Le schéma d'assemblage de la nouvelle version de la caméra est montré à la Figure 6.2. Le plan mécanique de chaque composante qu'on y retrouve est présenté à l'ANNEXE K. Cette version comprend un premier circuit de conditionnement qui convertit chaque courant provenant du capteur en un signal de tension. Les plans électriques de ce circuit de conditionnement du capteur sont présentés à ANNEXE L. Deux étages de filtres produisent, grâce à un réseau bidimensionnel de résistances, l'ensemble des images filtrées. La totalité des signaux analogiques est propagée sur un bus analogique vertical. Ce connecteur de 60 broches sert également à propager les alimentations de même que les commandes de déplacement du pixel d'intérêt en provenance du contrôleur. L'arrangement en sandwich des différents circuits facilite l'ajout de circuits supplémentaires.

6.1.3 Mise au foyer de l'image sur le capteur

La mise au foyer de l'image sur le capteur est une tâche ardue. En effet, il est difficile de connaître précisément la position de la surface photo-sensible par rapport au circuit imprimé qui supporte le capteur. Cette imprécision est due au fait que l'assemblage des dés de silicium n'est pas prévue pour une utilisation opto-électronique du circuit intégré. On doit de plus insérer le boîtier de céramique dans une base d'assemblage qui est elle même soudée au circuit imprimé du conditionnement des signaux analogiques. Cette séquence d'assemblages produit plusieurs effets parasites. On doit corriger ceux-ci en ajustant la définition d'une image sur le capteur à l'aide du montage de la Figure 6.3.

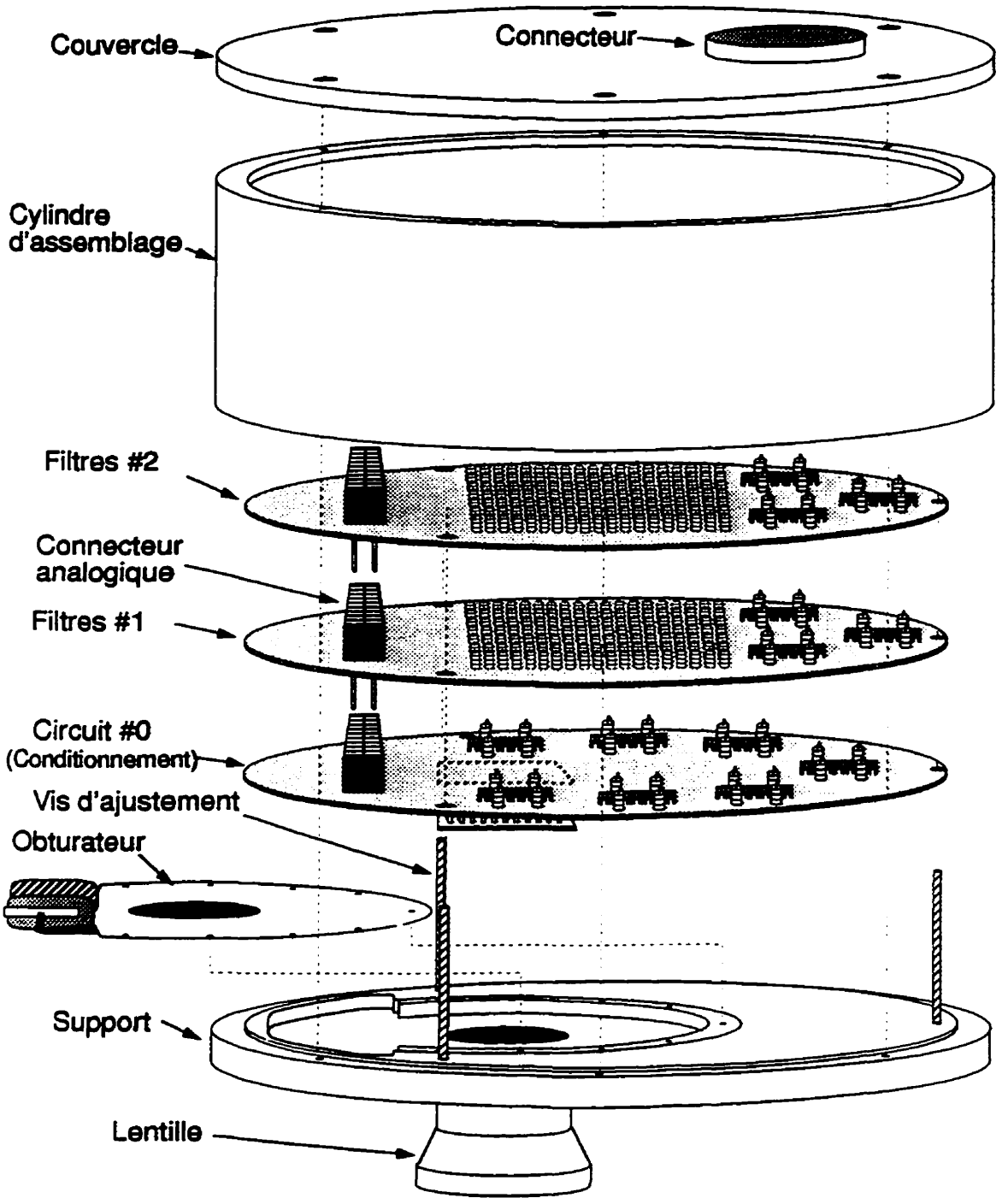


Figure 6.2 *Schéma d'assemblage de la caméra MAR. La lentille est fixée au support principal de même que le circuit #0 qui comprend le capteur sur le côté opposé aux autres composantes. Les vis d'ajustement servent à modifier l'orientation du capteur pour qu'il soit au foyer. Un connecteur analogique propage l'ensemble des signaux du capteur et des filtres pour tous les circuits analogiques de la caméra.*

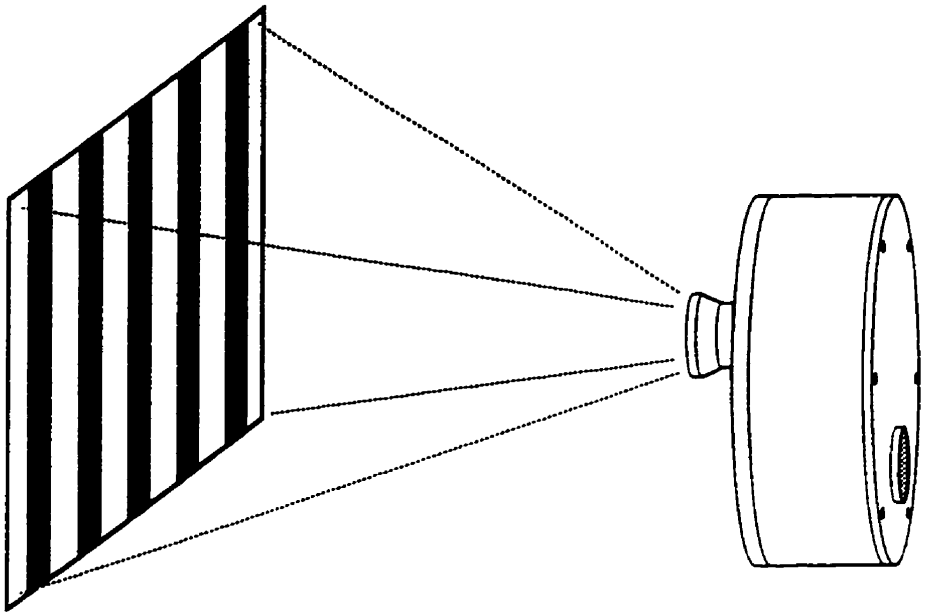


Figure 6.3 Montage requis pour la calibration de l'orientation du capteur. Après avoir aligné la cible de calibration parallèlement au boîtier de la caméra et à une distance connue de celui-ci, on ajuste la position et l'orientation du capteur afin d'obtenir la plage dynamique maximum pour l'amplitude des passages par zéro d'un des filtres sur toute l'étendue de l'image.

On utilise une cible à haut contraste composée de larges bandes verticales. La sortie des filtres produit une série régulière de passages par zéro lors d'un balayage en mode ligne par ligne. Comme on peut le voir à la Figure 6.4, Lorsqu'on observe la sortie des filtres laplaciens de gaussiennes alors que le capteur n'est pas parfaitement placé au plan focal, seules certaines régions de l'image se trouvant au foyer génèrent un signal d'amplitude optimale. Les autres régions du capteur voient une image embrouillée et l'amplitude du signal provenant des filtres est plus faible. La Figure 6.4 (b) montre le cas où la partie centrale du capteur est au foyer mais la normale au plan du capteur n'est pas parallèle à l'axe optique de la caméra par rapport à l'axe vertical. Le cas de la Figure 6.4 (c) est similaire sauf que la déviation angulaire est observée par rapport à l'axe horizontal.

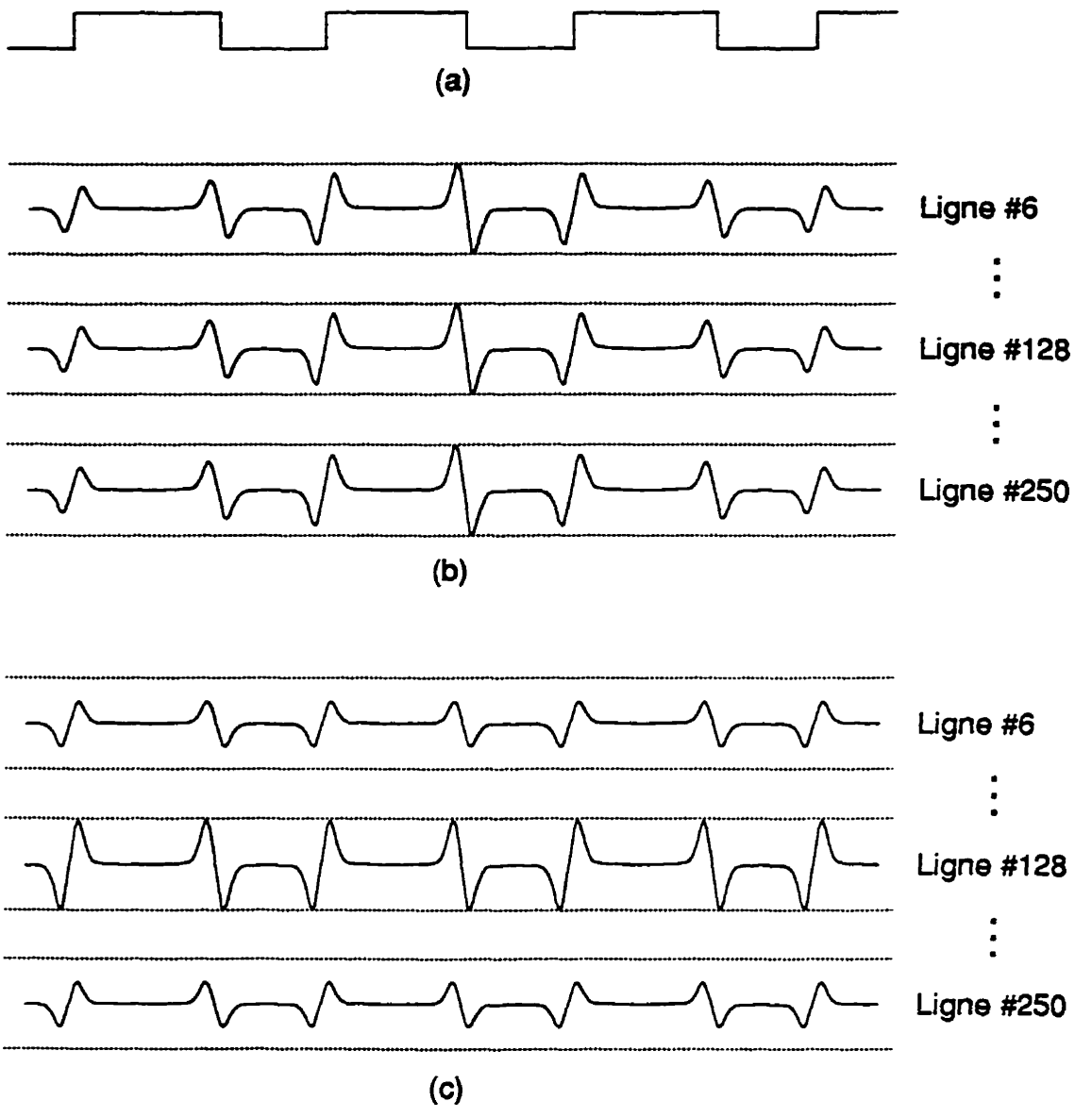


***Figure 6.4** Patrons des passages par zéro observés lorsque le centre du capteur est au foyer mais qu'il a un angle de déviation par rapport à: l'axe vertical en (a) et l'axe horizontal en (b) dans le montage de la Figure 6.3. La coupe du signal d'illuminance de la cible de calibration est montrée en (a). La calibration du capteur au foyer est optimale lorsque l'amplitude du résultat des filtres est maximale et uniforme sur toute l'image.*

On utilise alors les trois vis qui sont montrées à la Figure 6.2 pour ajuster l'orientation de la normale et la focale du capteur afin d'obtenir une réponse analogique

maximale et uniforme sur toute l'image lorsqu'on observe une cible comme celle de la Figure 6.3.

6.2 Résultats expérimentaux

Les résultats expérimentaux disponibles ont tous été obtenus à l'aide du premier capteur fonctionnel qui a une résolution de 128 x 128 pixels. Certaines images ont été captées grâce au montage de la Figure 6.1 à partir de négatifs de 35 mm alors que d'autres proviennent de la sortie de la caméra munie d'une lentille et d'un obturateur optique. L'acquisition d'images provenant du capteur MAR est la preuve du fonctionnement de ce dernier. Les images d'arêtes qui sont représentées ici utilisent une grille de 512 x 512 en topologie cartésienne

Les premières images obtenues par le capteur MAR sont montrées à la Figure 6.5. La Figure 6.5 (a) montre le visage d'un homme. On peut remarquer que chaque segment d'arête est soit horizontal, soit dans l'une ou l'autre des diagonales à 60°. La région de la chevelure est un exemple flagrant de l'extraction d'arêtes selon plusieurs résolution spatiales où on observe un ensemble de contours à haute résolution (σ=0.7 et σ=1) alors qu'on identifie uniquement deux régions distinctes à la sortie des filtres grossiers (σ=2.5 et σ=3). La région des yeux, de la bouche et le motif sur le bras gauche sont des exemples similaires.

Les résultats montrés à la Figure 6.5 (b) proviennent d'un visage de femme avec un store vénitien en arrière plan. On a effectué une rotation vers la gauche des résultats pour l'impression car la photo avait été prise dans une orientation différente. Les parties du visage et de la chevelure sont extraites similairement à l'exemple précédent mais on peut voir en plus que les patrons horizontaux du store sont très présents à haute résolution et fortement atténués à basse résolution. Il est intéressant de remarquer qu'un des transistors du multiplexer de sortie du capteur est inopérant, ce qui provoque l'apparition de certaines fausses détections. Cette anomalie est présente au premier quart à gauche du capteur pour la Figure 6.5 (a) et en bas de la Figure 6.5 (b). Bien que ces images soient importantes pour le projet puisqu'elles sont les premières qui furent captées, elles comportent des défauts majeurs dans la qualité de la formation des images au plan focal qui ne peut être ajusté précisément sur tout le capteur. De plus, certains filtres n'étaient pas parfaitement calibrés au moment de la prise de ces images.

La Figure 6.6, la Figure 6.7 et la Figure 6.7 présentent des résultats typiques obtenus à l'aide du capteur une fois assemblé dans son boîtier. Ces images ont été choisies pour

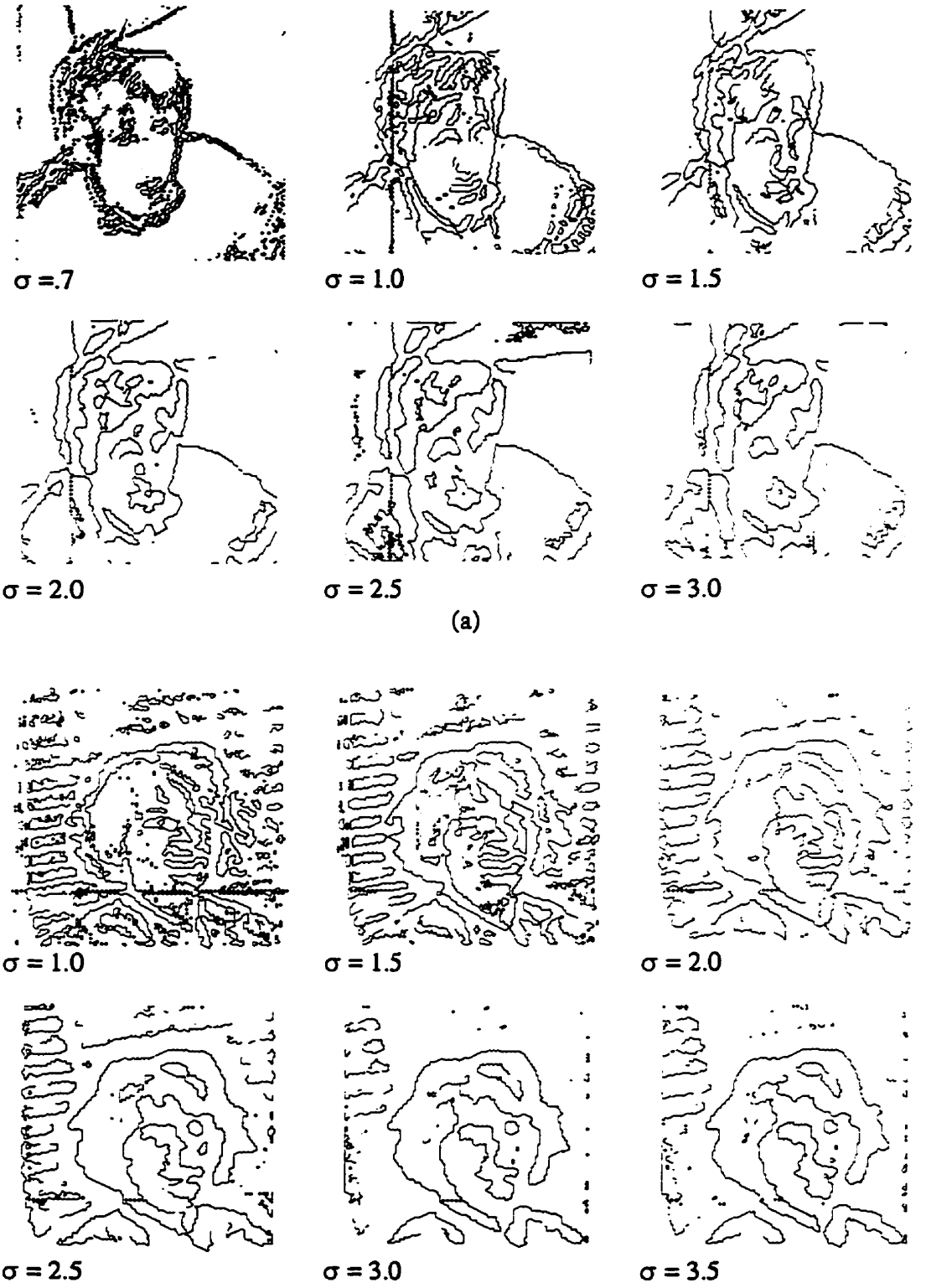
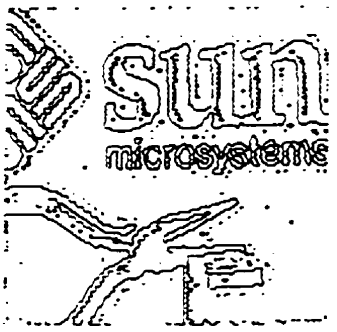
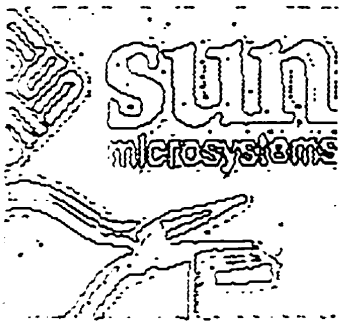


Figure 6.5 *Résultats obtenus à l'aide du banc d'essai pour photo-capteurs spécialisés. Les sources de ces images proviennent de négatifs des visages de Robert (a) et de Roxane (b). Les images de la partie (b) sont montrées avec une rotation de 90 degrés vers la gauche.*

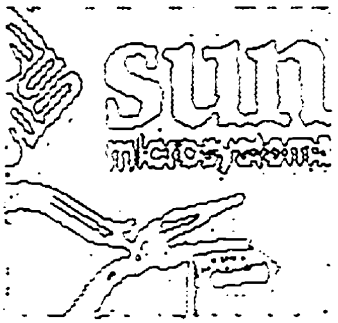
Image d'illuminance
(à partir d'une caméra CCD)



σ = 0.7

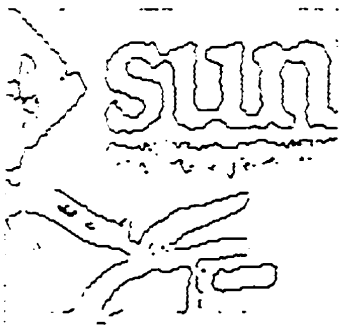


σ = 1.0

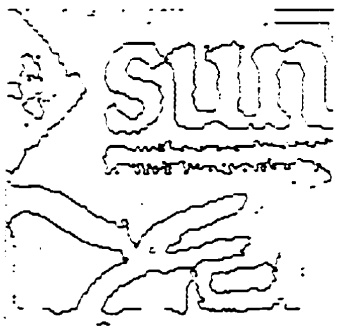


σ = 1.5

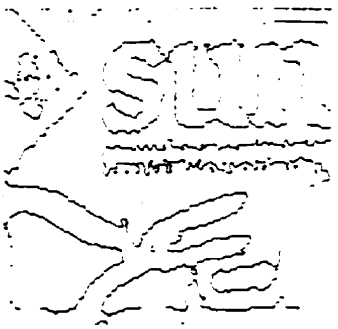
σ = 2.0



σ = 2.5



σ = 3.0



σ = 3.5

Figure 6.6 *Images d'arêtes résultantes pour la scène de "SUN". L'image d'illuminance montre bien les variations de contraste à la grandeur de l'image et même une région de réflexion spéculaire de la source lumineuse en bas à droite de l'image et sur les pinces. Les résultats des filtres grossiers (σ = 3 et σ = 3.5) montrent bien la capacité de segmentation intrinsèque au capteur MAR. Le symbole de "SUN" et les petits caractères y sont regroupés en un seul contour fermé alors qu'on identifie les éléments discrets de ces ensembles à mesure qu'on raffine la détection d'arêtes à l'aide de filtres plus fins.*

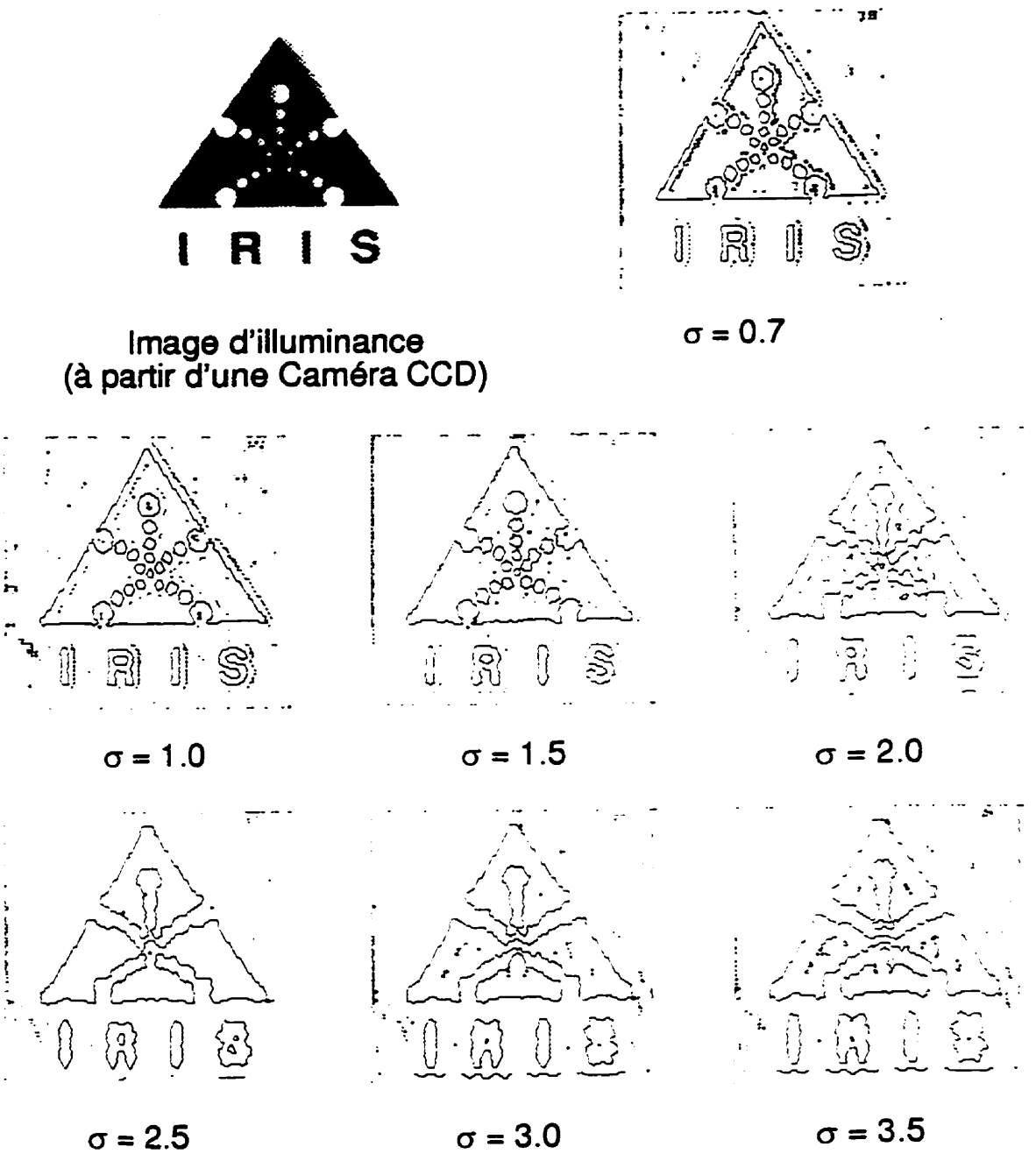


***Figure 6.7** Résultats de l'extraction d'arêtes à multiples résolutions sur l'image du symbole d'IRIS. Cette scène est intéressante car les petits cercles du centre présentent une bande de fréquences spatiales qui est révélée graduellement par les différents opérateurs.*

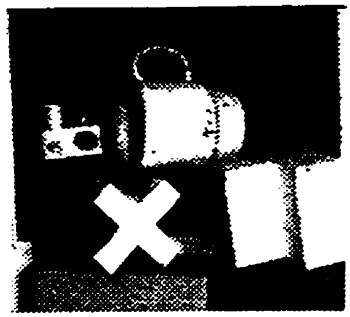
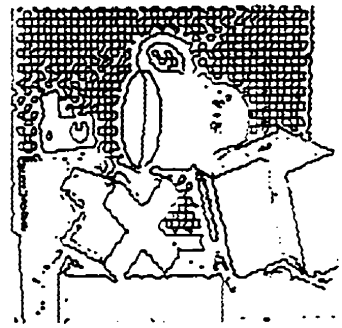
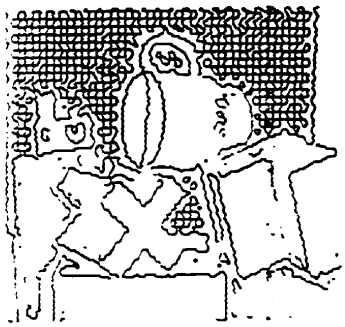


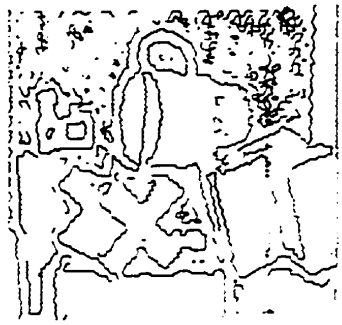
Image d'illuminance
(à partir d'une Caméra CCD)



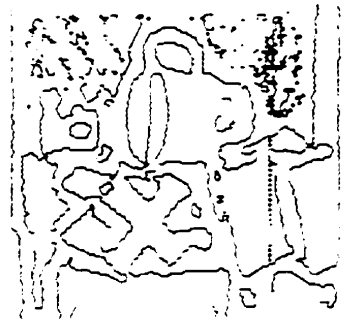
$\sigma = 0.7$



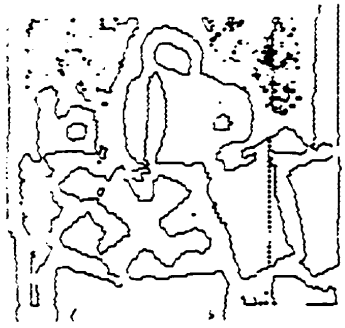
$\sigma = 1.0$



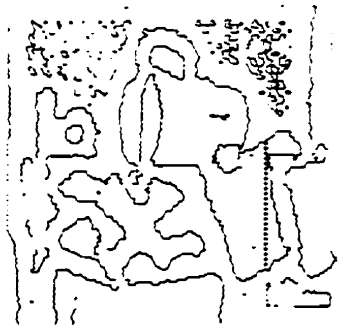
$\sigma = 1.5$



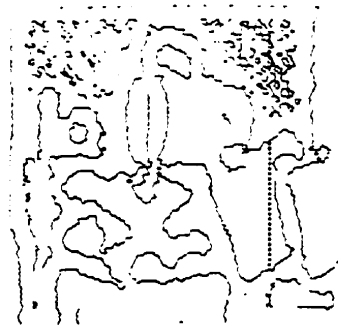
$\sigma = 2.0$



$\sigma = 2.5$



$\sigma = 3.0$



$\sigma = 3.5$

***Figure 6.8** Résultats de l'extraction d'arêtes à multiples résolutions à partir d'une scène polyédrique. On a placé volontairement un motif en damier à l'arrière-plan de la scène pour mettre en évidence la discrimination fréquentielle des filtres qui ont une valeur de σ élevée. La source d'éclairage se trouve à droite de la caméra et on peut remarquer la détection des contours due à l'ombrage des objets à l'avant-plan.*

démontrer la discrimination fréquentielle des opérateurs implantés sur le système MAR. Elles reflètent de plus le fonctionnement sur des images à niveaux de gris par opposition à une opération sur des images binaires. Les images d'illuminance sont fournies par une caméra CCD car, au moment de la rédaction de la thèse, le module d'acquisitions numérique de l'illuminance n'avait pas encore été implanté.

La Figure 6.6 montre les résultats obtenus pour une scène qui comprend des caractères typographiques ("SUN" et "microsystems"), le symbole de "SUN", une paire de pinces et un circuit intégré fixé sur un morceau de mousse rose. Les résultats obtenus sont plus clairs car il n'y a qu'une lentille comme intermédiaire entre la scène et le capteur. La capacité de segmentation est particulièrement mise en évidence pour le cas des caractères où le contour des lettres est très précis à haute résolution alors que le mot "microsystems" est perçu comme un objet unique à résolution plus grossière. Le même effet se produit sur le symbole de "SUN" qui est vu comme un objet unique en forme de losange à faible résolution alors qu'il s'agit de l'enchaînement de huit lettres "U" distinctes lors de la détection d'arêtes à l'aide de filtres plus fins. La détection d'arêtes effectuée sur la pince montre comment le système réagit sur un objet spéculaire.

Les résultats de la Figure 6.7 sont obtenus de la même façon que pour l'exemple précédent sur une scène qui représente le symbole du programme IRIS. L'image d'illuminance est fournie par une caméra CCD afin de présenter au lecteur la scène analysée. La discrimination fréquentielle est observable sur les ronds au centre du triangle qui sont détectés individuellement à haute résolution et qui sont regroupés à plus faible résolution.

La Figure 6.7 démontre le fonctionnement du système à partir d'une scène plus réelle. On a placé un motif en damier à l'arrière-plan de la scène pour mettre en évidence la discrimination fréquentielle des filtres qui ont une valeur de σ élevée. Les objets présents dans la scène de la Figure 6.7 sont tous blancs et les changements de contraste correspondent uniquement à un changement de l'orientation des surfaces par rapport à la source lumineuse. On peut d'ailleurs remarquer que de très faibles changements d'illuminance sont relativement bien détectés par la caméra. L'effet d'ombrage est mis en évidence car la source d'éclairage est placée volontairement à droite de la caméra. On peut donc remarquer le détection des contours due à l'ombrage à gauche des objets qui se retrouvent à l'avant-plan.

Une observation attentive du facteur d'échelle vertical montre une légère différence entre l'image vue par la caméra CCD et la représentation de l'image provenant du capteur

MAR. Cette distorsion n'existe pas en fait car elle est simplement générée par l'algorithme de représentation de l'image hexagonale sur une grille cartésienne. C'est que l'image hexagonale qui a un rapport 7/8 entre la hauteur d'une ligne du capteur et la distance inter-pixel est projetée sur une grille 8/8. L'ensemble des résultats qui sont présentés ici proviennent d'un montage d'acquisitions rudimentaire sans éclairage optimal ni artifices pour améliorer la qualité de la formation des images. L'ajustement au foyer du capteur pourrait certainement être amélioré par une procédure systématique de calibrage. On pourrait également ajuster de façon plus optimale la valeur des coefficients, le gain des différents filtres de même que l'unique seuil (V_s) qui définit l'activation du signal hystérésis (H) tel que discuté à la section 4.3.9.

6.3 Algorithmes de commande et applications spécialisées

Un certain nombre d'applications ont été initialement pensées de façon à influencer la description et le design de la caméra. Ce travail d'anticipation a permis de planifier la mise en place de quelques options intéressantes qui ont été intégrées au jeu d'instructions ou ajoutées comme modules connexes au contrôleur MAR. On présente ici quelques-unes des applications possibles. On insiste aussi sur la notion de topologie hexagonale et de communication bilatérale entre le système hôte et la caméra MAR. Plusieurs applications nécessiteraient idéalement des circuits électroniques connexes, sous forme de circuits VLSI spécialisés ou de composantes programmables. Cette section propose des avenues pour la conception de ces modules spécifiques. Elle ne vise pas une description détaillée de leur réalisation.

6.3.1 Détection d'arêtes multi-résolution et segmentation hiérarchisée

Pour une application typique d'extraction d'arêtes à de multiples résolutions, le système hôte commande un balayage de l'image filtrée à très basse résolution. Il extrait ensuite une ou plusieurs fenêtres d'étude. Chaque structure principale est étiquetée et le suivi d'arêtes à des résolutions de plus en plus fines est effectué[36]. A mesure que la description se raffine, il est possible d'estimer plus précisément la position des contours en plus d'ajouter des caractéristiques de forme, de jonction d'arêtes ou d'inclusion d'une arête par rapport à un autre contour fermé. Cette description se prête bien à la création d'un arbre simple dont les noeuds représentent la description d'une arête alors que les arcs décrivent les relations entre les différentes arêtes. On peut également envisager la création de relations entre les graphes d'une même image, à des résolutions différentes, afin de définir une étiquette unique pour les arêtes correspondant au même contour dans la scène.

Une variante intéressante consiste à utiliser les contours fermés détectés à basse résolution pour définir une frontière à l'intérieur de laquelle on effectue un balayage ligne par ligne en repérant tous les points d'arêtes détectés par des filtres plus fins. Il est certain que le type d'objet à reconnaître et le contexte de la prise d'images influenceraient le choix de l'algorithme d'exploitation. De plus, la performance de la reconnaissance sera une conséquence de la complexité du programme d'exploitation. Plus l'échange bilatéral d'information entre la caméra et le système hôte sera efficace et orienté sur des actions conditionnelles, moins on aura à extraire de l'information inutile pour atteindre le but visé.

L'extraction multi-résolution d'arêtes implique fréquemment l'analyse des textures. Une région délimitée par un contour fermé à basse résolution qui dévoile de nombreuses arêtes montrant une organisation régulière à plus haute résolution appartient souvent à une surface texturée. L'analyse d'histogrammes associés aux points inclus dans cette région contribue à l'étiquetage des textures et devrait être considéré dans ce cas.

6.3.2 Interpolation sub-pixel

L'opération de détection d'arêtes effectuée par le système MAR consiste à extraire les passages par zéro de l'image convoluée avec l'opérateur laplacien de gaussienne $(\nabla^2 G)$. Les passages par zéro sont normalement détectés entre deux pixels dont le signe du résultat de la convolution est opposé et dont l'amplitude de la dérivée première du signal est suffisamment élevée. Lorsqu'un passage par zéro est détecté entre deux pixels avec un opérateur à haute résolution (donc avec une précision considérable sur sa position), il peut être intéressant de calculer la position de l'arête avec une précision supérieure à la résolution même du capteur hexagonal. On utilise alors l'interpolation sub-pixel pour localiser les contours avec plus de précision. Cette unité d'interpolation peut être réalisée soit par l'ordinateur hôte ou par un module VLSI spécialisé plus rapide. Posons R_0 et R_1 l'amplitude de l'image convoluée aux deux pixels situés de part et d'autre d'un passage par zéro et ΔP la distance inter-pixel, l'approximation de la partie fractionnaire Υ_{01} de la position du passage par zéro à l'aide de l'interpolation linéaire s'exprime comme:

$$:\Upsilon_{01} = \frac{R_0 \Delta P}{|R_0 - R_1|} \tag{6-1}$$

La Figure 6.9 illustre l'équation (6-1) et montre que la partie fractionnaire est définie par rapport à la position du pixel P_0. Cette relation simple augmente la résolution spatiale de plusieurs bits et peut être calculée par un module numérique relativement

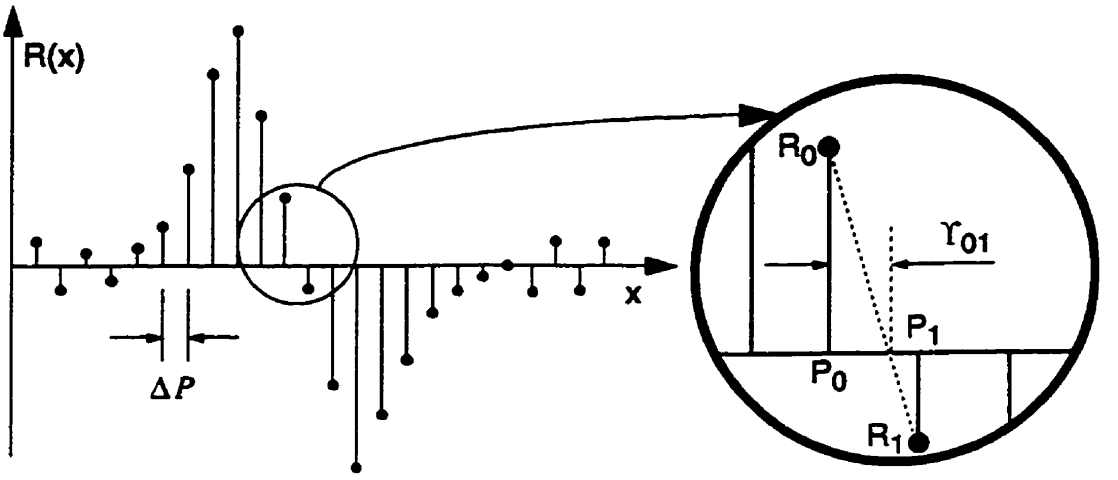


Figure 6.9 Interpolation sub-pixel de la position d'un passage par zéro pour un signal analogique échantillonné spatialement.

simple. Cette précision accrue sert avantageusement à augmenter la fiabilité de l'évaluation 3D en stéréo vision passive (section 6.3.4) par l'appariement des passages par zéro.

6.3.3 Opérateurs morphologiques

Les opérateurs morphologiques tels que l'érosion et la dilatation s'implantent facilement à l'aide de la caméra MAR. Le capteur donne accès à l'illuminance des 12 voisins immédiats du pixel d'intérêt permettant ainsi de programmer un opérateur directionnel. Certaines instructions de la caméra prévoient, par le biais du microcode, un mode de déplacement hélicoïdal permettant d'appliquer un opérateur morphologique sur l'image convoluée [2] [18]. Dans ce cas, on doit compter 7 coups d'horloge pour évaluer le résultat d'un pixel puisqu'on doit visiter les 6 voisins immédiats d'abord, puis, le pixel central auquel on assigne le résultat de l'opérateur morphologique. L'architecture du système MAR prévoit une mémoire image rendant plus efficace une telle procédure. La durée globale de traitement est réduite en balayant l'image du capteur une seule fois puis en effectuant le traitement morphologique sur la mémoire image à une fréquence supérieure.

Les fonctions d'érosion et de dilatation binaires sont utilisées comme pré-traitement dans le cas de l'extraction des arêtes à multiples résolutions. Comme nous l'avons précisé

à la section 4.3.9, l'opération de détection d'arêtes requiert deux bits d'information par pixel pour chaque filtre: un bit représentant le signe du résultat de la convolution avec l'opérateur $\nabla^2 G$ et un bit pour la valeur absolue de son amplitude seuillée (hystérésis). Un pré-traitement morphologique est requis afin d'effectuer la dilatation du bit hystérésis. En considérant qu'on effectue le filtrage de l'image selon plusieurs résolutions, il est alors intéressant d'effectuer en parallèle les opérations morphologiques sur l'ensemble des images filtrées.

6.3.4 Stéréo-vision passive

On envisage la possibilité de coupler deux caméras MAR pour réaliser une acquisition 3D par stéréo-vision passive [25] [28]. Puisque le système MAR donne accès simultanément à plusieurs images d'arêtes, on peut effectuer l'appariement stéréo sur l'ensemble des données disponibles fournies par deux caméras MAR fixes et ainsi augmenter le degré de confiance accordé à l'appariement d'un point caractéristique. Cette approche est basée sur l'utilisation de deux caméras complètes chacune munie d'un contrôleur et d'un module d'acquisitions. Une autre configuration moins conventionnelle exploiterait également les propriétés du système MAR tout en diminuant la quantité de matériel électronique requis. Celle-ci est présentée à la Figure 6.10. où l'on peut voir que chaque capteur fournit de l'information sur toute une région de pixels mais que l'un des deux est mobile. Un système d'asservissement recherche la position du capteur mobile qui offre un maximum de corrélation locale entre l'ensemble des signaux de l'image de gauche et de l'image de droite. On n'appliquerait cette procédure que dans les régions de l'image qui sont fortement contrastées où l'on aurait préalablement détecté des arêtes avec un opérateur grossier. Une simple relation géométrique par triangles semblables donne la profondeur 3D d'une région d'arête Y:

$$Y(3D) = \frac{fd}{\Delta X} \tag{6-2}$$

On constate à partir de l'équation (6-2) de même qu'à la Figure 6.10 que $Y(3D)$ tend vers l'infini lorsque ΔX tend vers zéro (ou que les rayons de la projection de l'objet sont parallèles). Puisque l'appariement ne se fait pas selon des valeurs entières de pixel comme les systèmes conventionnels de vision, la ligne de base d peut être réduite au minimum et conserver une bonne précision sur l'estimation de la profondeur 3D. On réduit ainsi les inconvénients causés par l'étude de deux images prises selon deux points de vue différents lorsque la ligne de base est trop grande (éclairage irrégulier, ombre, parties

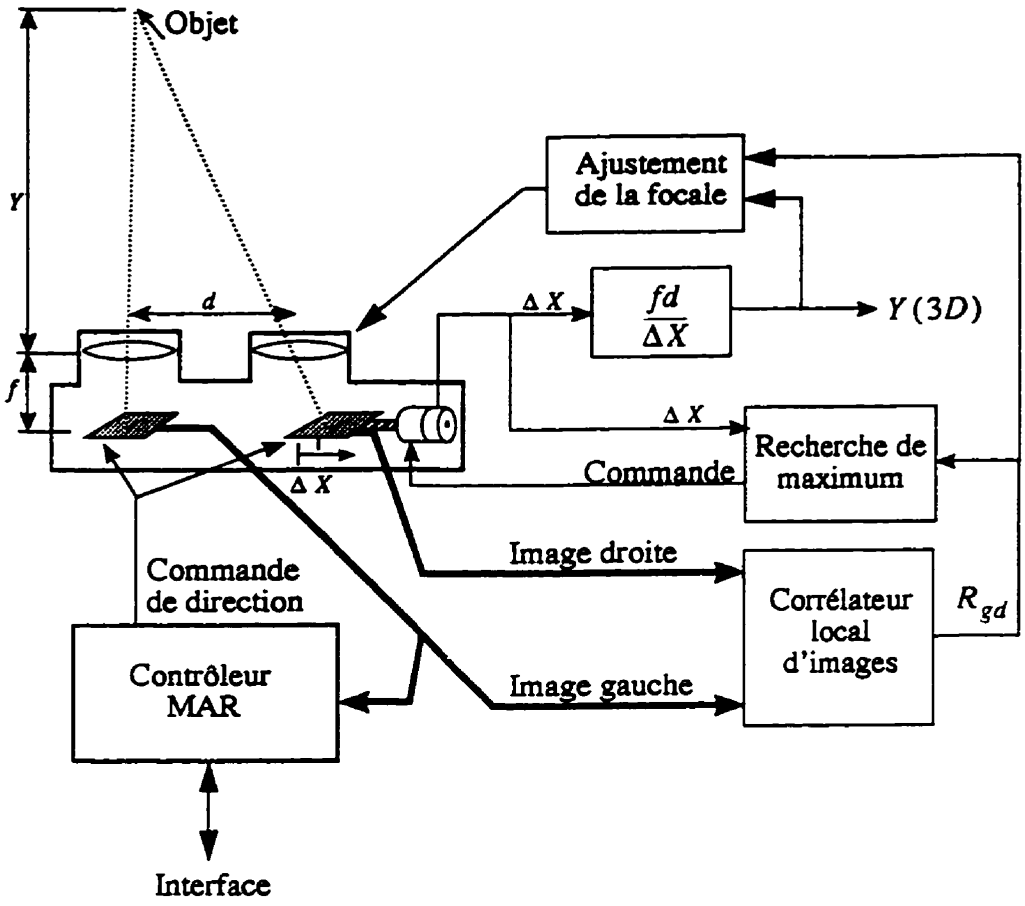


Figure 6.10 Modèle de caméra stéréo utilisant la propriété d'extraction parallèle de signaux analogiques pour l'évaluation 3D de la scène

cachées, etc.) L'utilisation d'un montage mécanique pour déplacer le capteur mobile limite le temps de réponse du système et ajoute un niveau de complexité appréciable qui se traduirait par une augmentation du coût de réalisation.

Il est cependant possible de procéder d'une autre façon. L'interpolation sub-pixel des passages par zéro peut être utilisée pour calculer la profondeur 3D à l'aide de deux capteurs fixes dont la ligne de base d est également très courte. Puisque la disparité stéréo est définie selon un seul axe, cette solution s'implante facilement par un sous système spécialisé qui paire les passages par zéro des images filtrées grossièrement lors d'un balayage ligne par ligne. On a avantage à utiliser une ligne de base très courte pour que les

deux images ne diffèrent pas beaucoup et ainsi reporter la précision du calcul de la profondeur sur la procédure d'interpolation sub-pixel. L'appariement est alors plus robuste que dans le cas d'une ligne de base plus large où on risque de se fier à une arête vue par une caméra alors qu'elle ne l'est par l'autre.

La valeur précise de la profondeur est calculée en effectuant l'intégration d'échelle locale puis en mesurant la disparité stéréo à l'aide des images d'arêtes denses obtenue par filtrage fin auxquelles on appliquerait l'interpolation sub-pixel. Plusieurs avantages intrinsèques à ce type de stéréo effectué à l'aide de deux caméras MAR peuvent être identifiés:

- En utilisant une ligne de base d très courte, on diminue au maximum l'étendue du déplacement du capteur mobile.
- La masse du capteur seul étant très faible, le temps de réponse du système mécanique demeure acceptable.
- Puisque la ligne de base d est très courte, on diminue les zones mortes de la paire stéréo.
- En utilisant un filtrage grossier, la carte de passages par zéro est moins dense et les arêtes sont plus longues ce qui rend l'appariement plus facile et fiable. L'image traitée est distordue mais la précision est disponible en utilisant des images à plus haute résolution et l'interpolation sub-pixel pour le calcul de la profondeur.
- la précision obtenue n'est pas limitée à un saut d'un pixel mais est plutôt définie par la résolution du mécanisme de l'appariement pour le capteur mobile et par la résolution de la partie fractionnaire associée à la procédure d'interpolation de la position des passages par zéro pour les deux capteurs fixes
- La valeur calculée de la distance $Y(3D)$ couplée à l'estimateur de corrélation R_{gd} peuvent être utilisés pour régler dynamiquement la focale des lentilles sur l'objet étudié.

6.3.5 Recouvrement de surfaces douces

Le recouvrement de surfaces à partir de l'image d'intensité communément appelé ("shape from shading") [20] [19] est une technique qui s'implante bien sur une topologie hexagonale. Cette technique appliquée sur une scène dont on connaît la carte de réflectance permet d'estimer le type de surface entre les arêtes (ex.: sphérique, selle, cylindrique, conique, etc.). L'idée est de faire croître une solution à partir d'un point singulier selon des hexagones concentriques [10]. Un avantage de la topologie hexagonale réside dans la possibilité de placer un morceau de surface triangulaire entre trois pixels voisins, de

calculer la normale à ce morceau de surface, puis d'évaluer la profondeur 3D d'un de ces points à partir de la profondeur connue des deux autres.

On développe ici le cas particulier (bien que très réaliste) où la scène est éclairée par une source confondue avec l'observateur (carte de réflectance à symétrie de révolution). On peut poser une relation simple entre l'illuminance d'un point de la scène $E_{triangle}$ et l'angle θ mesuré entre l'axe optique y et la normale à la surface $\vec{n}$:

$$E_{triangle} = f(\theta) \tag{6-3}$$

On associe une illuminance moyenne à chaque morceau de surface triangulaire définie entre trois points voisins P_0, P_1 et P_2 par la relation:

$$E_{triangle} = \frac{E_0 + E_1 + E_2}{3} = f(\theta) \tag{6-4}$$

Supposons que la profondeur (y_o et y_1) est connue pour les points P_0 et P_1 de la Figure 6.11, et qu'on cherche à évaluer la profondeur y_2 au point P_2. Puisque la normale au morceau de surface triangulaire (équilatéral) s'écrit:

$$\vec{n} = \begin{bmatrix} \frac{|y_1 - y_0|}{\Delta x} & 1 & \frac{\frac{y_1 + y_0}{2} - y_2}{\Delta z} \end{bmatrix} \tag{6-5}$$

où Δx est la distance entre deux pixels voisins et Δz est la distance entre deux lignes en topologie hexagonale soit: $\Delta z = \Delta x \cos 30°$ et que l'axe optique est représenté par le vecteur unitaire:

$$\vec{j} = [0\ 1\ 0] \tag{6-6}$$

On peut écrire à partir de la définition du produit scalaire que:

$$\vec{n} \cdot \vec{j} = \|\vec{n}\| \|\vec{j}\| \cos\theta = \|\vec{n}\| \cos\theta = 1 \tag{6-7}$$

d'où:

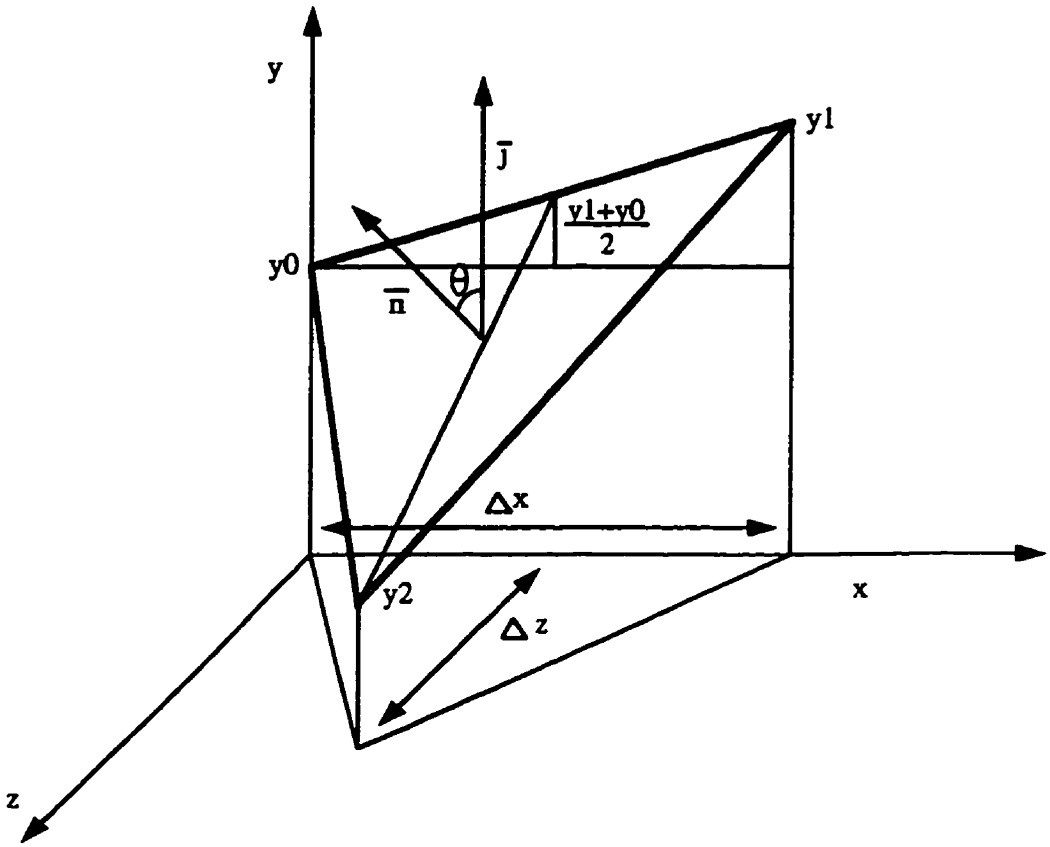


Figure 6.11 *Organisation géométrique d'un morceau de triangle équilatéral pour le "shape from shading" en topologie hexagonale. L'axe y est confondu avec l'axe optique.*

$$\cos\theta = \frac{1}{\|\vec{n}\|} = \frac{1}{\sqrt{1 + \left(\frac{y_1 - y_0}{\Delta x}\right)^2 + \left(\frac{\frac{y_1 + y_0}{2} - y_2}{\Delta z}\right)^2}} \qquad (6\text{-}8)$$

En remplaçant θ par sa valeur dans l'équation (6-4) et en solutionnant pour y_2 on obtient:

$$y_2 = \left(\frac{y_0 + y_1}{2}\right) \pm \Delta z \sqrt{\left(\frac{1}{\cos\left[f^{-1}\left(\frac{E_0 + E_1 + E_2}{3}\right)\right]}\right)^2 - \left(\frac{y_0 - y_1}{\Delta x}\right)^2 - 1} \qquad (6\text{-}9)$$

L'équation (6-9) met en évidence certaines caractéristiques intrinsèques de la technique de "shape from shading". Premièrement, on doit lever l'indétermination du signe de la croissance de la solution. La profondeur y_2 correspond bien à la moyenne de la profondeur des deux points connus plus ou moins une valeur fonction de l'illuminance de cette région. Deuxièmement, il est possible de calculer une valeur négative sous la racine carrée dans le cas où la différence de profondeur entre les deux points connus P_1 et P_0 est trop grande pour justifier la mesure locale de l'illuminance. C'est un cas particulier que l'algorithme devrait prévoir.

Il est intéressant de noter que dans le cas de surfaces de type lambertienne, l'équation (6-4) s'écrit:

$$E_{triangle} = \frac{E_0 + E_1 + E_2}{3} = E_{MAX}\cos\theta \qquad (6\text{-}10)$$

L'équation (6-9) se formule donc comme suit pour une surface de type lambertienne:

$$y_2 = \left(\frac{y_0 + y_1}{2}\right) \pm \Delta z \sqrt{\left(\frac{3E_{MAX}}{E_0 + E_1 + E_2}\right)^2 - \left(\frac{y_0 - y_1}{\Delta x}\right)^2 - 1} \qquad (6\text{-}11)$$

Il est certain qu'une recherche plus approfondie est nécessaire pour mettre au point les algorithmes de recouvrement de surface douce à partir de l'illuminance. Une variante intéressante serait de redéfinir la grille de pixels 3D de façon décalée par rapport à la grille de pixels d'illuminance. En plaçant le centre de chaque morceau de surface triangulaire exactement sur un pixel d'illuminance, on élimine le moyennage ($(E_0 + E_1 + E_2)/3$) et on augmente la précision de l'algorithme qui a le défaut d'accumuler les erreurs d'évaluation à mesure que la solution progresse. Dans ces conditions l'équation (6-11) se récrit sous la forme suivante:

$$y_2 = \left(\frac{y_0 + y_1}{2}\right) \pm \Delta z \sqrt{\left(\frac{E_{MAX}}{E_{centre}}\right)^2 - \left(\frac{y_0 - y_1}{\Delta x}\right)^2 - 1} \qquad (6\text{-}12)$$

La réalisation logicielle de cet algorithme de recouvrement de la forme par morceaux de triangles est présentée à l'ANNEXE G. On présente un résultat obtenu à l'aide de ce logiciel à la Figure 6.12 qui montre la croissance selon des hexagones concentriques

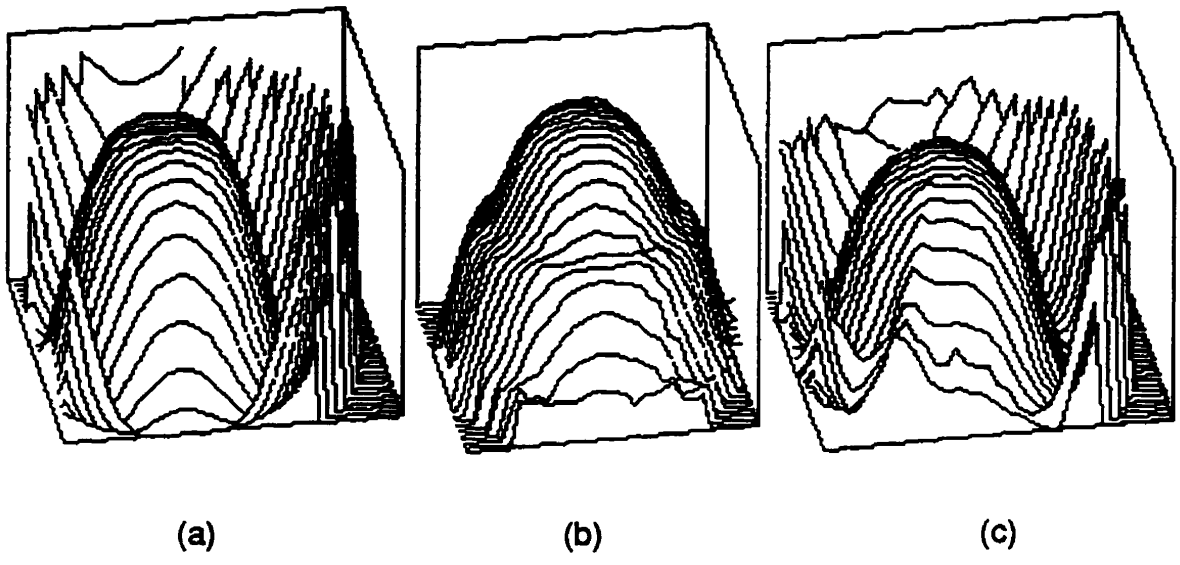


Figure 6.12 Exemple de recouvrement de l'image à partir de morceaux de surface triangulaires. L'image source est une cosinuoïde de révolution (a) et les résultats sont montrés en (b) où le sens de croissance est constant et en (c) où on évalue le signe de cette croissance à partir d'une fonction heuristique.

sur une image synthétique (une branche de cosinusoïde de révolution). L'algorithme utilise le modèle de l'équation (6-11). Il est intéressant de remarquer sur l'image résultante (c) que le sens de croissance de la solution est approximé par une fonction heuristique qui suppose qu'un lieu continu de singularités (dans cet exemple c'est un anneau à $r = \pi$) correspond à un changement de signe de la croissance de la solution. Le résultat montré en (b) calcule la solution en gardant un signe de croissance constant.

On remarque également que les erreurs d'estimation se font principalement dans les coins des hexagones ce qui nous porte à revoir de plus près cette partie de l'algorithme. Un dernier point concerne le critère de continuité qui devrait être respecté lorsqu'on effectue une révolution complète sur un hexagone lors de la croissance de la solution. En redistribuant l'erreur qui est effectuée entre le point de départ et le point d'arrivée aux autres points du parcours hexagonal, on pourrait probablement améliorer les résultats. Cette

correction pourrait être appliquée prioritairement aux régions plus sensibles au bruit comme les six coins du parcours hexagonal et les régions d'illuminance faible.

6.3.6 Poursuite dynamique

Une application intéressante pour une machine de vision est la poursuite dynamique d'un objet mobile dans une scène. Le système MAR s'acquitte de ce genre de tâche par la redéfinition dynamique de la fenêtre d'étude. La première phase de l'approche consiste à identifier l'objet cible et à le situer dans l'image. Cette procédure se réalise par la détection de contours à multiples résolutions comme à la section 6.3.1. Une fois l'objet situé, on l'encadre en définissant la fenêtre d'étude aux limites de son contour d'occlusion. En utilisant seulement un sous-ensemble de l'image, le nombre de pixels à traiter reste faible ce qui accélère le traitement. Puisque l'algorithme de suivi automatique d'arêtes se termine lorsque le contour traverse la fenêtre d'étude, la direction de déplacement de l'objet peut être estimée grossièrement par la détection de cette condition et ainsi forcer l'ajustement d'une nouvelle fenêtre d'étude pour continuer la poursuite. Il est évident que l'algorithme décrit ici doit s'adapter au contexte de la scène à analyser et que certains artifices spécifiques à chaque application seraient nécessaires pour valider la détection et la reconnaissance de l'objet visé.

La principale limitation de la poursuite dynamique d'objet est associée à la prise d'images elle même. Celle-ci implique l'intégration de la lumière pendant une certaine période de temps (environ 35 ms pour des conditions d'éclairage normales). Tout objet se déplaçant rapidement crée une image fantôme. La caméra MAR étant définie avec un temps d'intégration programmable, on aura donc avantage, comme on le fait avec un appareil photographique, à avoir une scène très bien éclairée pour limiter au maximum la durée d'exposition et ainsi optimiser la qualité de l'image d'objets mobiles.

6.3.7 Reconnaissance de patrons et de textures

L'extraction parallèle des signaux analogiques réalisée par le capteur MAR peut certainement être mise à profit pour effectuer la reconnaissance de patrons locaux ou encore pour la classification de textures. En effectuant une forme de corrélation entre l'information extraite du capteur et un patron donné, il est possible d'identifier la correspondance à ce patron. La génération d'histogrammes locaux renforce les hypothèses de reconnaissance de patrons et de textures. Aucune étude exhaustive n'a été entreprise sur ce domaine précis bien qu'on puisse envisager une recherche exploratoire pour ce genre d'application.

CONCLUSION

La vision artificielle est certainement l'un des domaines où on peut tirer pleinement profit de la puissance de calcul du traitement massivement parallèle qu'offrent certaines architectures spécialisées. Le traitement d'images à l'aide du système MAR constitue une forme de traitement en parallèle et permet d'extraire des caractéristiques qui sont compatibles avec les algorithmes de plus haut niveau du processus de reconnaissance. Grâce à l'extraction multi-port des valeurs d'illuminance faite par le capteur bidimensionnel et au calcul analogique massivement parallèle effectué en périphérie de l'organe photo-sensible, on dispose simultanément des images d'arêtes à différentes résolutions spatiales ainsi que du résultat de la convolution avec tout genre d'opérateur tel que gaussien, directionnel ou autre. Il est particulièrement intéressant de calculer le nombre d'opérations mathématiques équivalentes que le système MAR effectue en une seconde dans le cas d'une configuration typique de 16 filtres à une cadence de 30 images par seconde. On obtient alors:

$$\frac{30 \cdot images}{seconde} \times \frac{65536 \cdot pixels}{image} \times 16 \cdot filtres \times \frac{91 \cdot accès}{filtre} \times \frac{2 \cdot opérations}{accès}$$

soit:

$$5.7 \times 10^9 \ \frac{operations}{seconde} !$$

Cette impressionnante capacité de calcul n'est pas toujours utilisée au maximum car plusieurs données sont générées inconditionnellement et ne serviront que pour discriminer, à l'occasion, des cas ambigu lors de l'intégration d'échelle.

Pour être compatible avec l'information échantillonnée selon une grille hexagonale, on a développé des algorithmes spécifiques à ces images. Une unité numérique microcodée effectue la poursuite d'arêtes en mode hexagonal et commande le balayage du pixel d'intérêt selon une multitude de modes. Cette unité supervise également le transfert de l'information avec l'ordinateur hôte en plus de gérer l'archivage des variables d'état et la mise à jour d'un bon nombre de registres qui contiennent des données propices à la description de l'information visuelle extraite par le capteur.

Les images obtenues à l'aide du capteur MAR démontrent bien le fonctionnement global du système comme il avait été prévu lors des simulations qui ont précédé la réalisation du premier prototype. Les images d'arêtes générées par le système peuvent très clairement servir à une segmentation hiérarchisée des éléments qui composent une scène. De plus, il est particulièrement intéressant de constater que l'architecture proposée pour le capteur a permis la fabrication de prototypes ayant une résolution spatiale de 256 x 256 pixels sur un circuit de taille raisonnable. Il est possible d'envisager la fabrication de capteurs à accès multi-port de photo-récepteurs atteignant des résolutions de 512x512 pixels. C'est d'ailleurs la contribution la plus exceptionnelle de ce travail car un minimum de résolution est requis pour l'application d'un tel système intelligent d'acquisitions d'images à des fins d'automatisation de procédés.

Un grand nombre de travaux sont consécutifs au développement du système MAR et devront être sérieusement explorés en vue d'ajouter une forme d'intelligence à l'acquisition d'images. La calibration automatique de la caméra est une tâche qui doit être réalisée par le logiciel d'exploitation. La calibration comprend l'ajustement du niveau de blanc et la définition du référentiel image en fonction de repères dans la scène. Cette procédure devrait idéalement s'adapter aux changements de certaines conditions propres à l'environnement de la scène.

Puisque le système MAR génère plusieurs images d'arêtes en parallèle, on doit prévoir un degré de parallélisme pour le processus d'interprétation de cette information. On peut penser à une architecture numérique parallèle qui effectue l'extraction des segments d'arêtes continus pour chaque résolution. Le développement d'algorithmes de commande cohérents et efficaces pour l'exploitation de l'ensemble de ces architectures spécialisées est un des thèmes importants au chapitre des travaux futurs. Énumérons à titre d'exemple les études portant sur la vision stéréo et sur le recouvrement de surfaces douces à partir de l'ombrage. Plusieurs projets relatifs à l'amélioration du concept de la caméra MAR font appel à la technologie d'intégration à très grande échelle. La plupart des projets requiert le

développement et la mise au point de nouveaux circuits VLSI pour rendre ces idées réalisables, compactes et performantes.

L'amélioration des propriétés optiques du capteur MAR et l'ajout de nouvelles propriétés de fonctionnement sont des niches de recherche qui doivent être explorées afin de rendre le système encore plus efficace. Certaines améliorations ont été suggérées dans la thèse mais d'autres sont envisageables comme l'utilisation de la technologie BiCMOS, l'insertion des circuits de conditionnement des signaux analogiques, l'ajout d'une capacité de mémorisation dynamique pour chaque pixel ou le micromachinage de lentilles locales pour concentrer la lumière incidente sur la partie photo-sensible du capteur. L'intégration du calculateur analogique est déjà amorcée et constitue une orientation de recherche particulièrement importante visant à réduire la taille des circuits périphériques au capteur et par conséquent, celle du boîtier de la caméra.

L'aboutissement final du projet consiste en l'intégration de l'ensemble des composantes microélectroniques du système MAR dans un même boîtier de céramique sous la bannière du procédé d'assemblage des pastilles de silicium sur un plus grand substrat de silicium. ("multi-chip modules"). Cette nouvelle technologie d'assemblage offre des avantages marqués par rapport aux techniques conventionnelles en ce qui a trait aux délais de communications locales entre les modules et au niveau de la dimension finale du système. Elle permet de restreindre l'assemblage uniquement à des dés de silicium fonctionnels en plus d'offrir la possibilité de combiner des modules qui utilisent différentes technologies d'intégration. Dans le cas du système MAR, on peut penser à un assemblage des quatre principaux modules dans un même boîtier soit: le capteur et le contrôleur réalisés à l'aide de la technologie CMOS, le calculateur analogique en technologie Bipolaire ou BiCMOS et un bloc de mémoire pour la description d'état et l'acquisition numérique des données. Cette configuration particulièrement intéressante et compacte est réalisable et envisageable à court terme puisque la Société canadienne de microélectronique projette de mettre en service cette technologie d'assemblage multiple au cours de la prochaine année.

La venue de ce nouveau genre de caméra intelligente munie d'une capacité de traitement d'images intégrée au plan focal devrait permettre de rendre les systèmes d'acquisitions et de pré-traitement moins coûteux, plus performants et mieux adaptés aux besoins spécifiques des algorithmes de segmentation et de reconnaissance.

RÉFÉRENCES

[1] Anderson, S., W.H. Bruce, P.B. Denyer, D. Renshaw, G. Wang, *"A Single Chip Sensor & Image Processor for Fingerprint Verification"*, IEEE 1991 Custom Integrated Circuit Conference, San Diego, California, May 1991.

[2] Archibald, C.C., S.R. Sternberg, *"Mathematical Morphology Applicated to Range Image Processing"*. Machine Vision International Ltd., 280 Albert St., Ottawa, Canada. K1P 5G8.

[3] Ashcroft, N.W., N.D. Mermin, *"Solid State Physics"*, Saunder College HRW Ed., 1976.

[4] Bair, M.E., R. Sampson, R. Zuk, *"Three-Dimensional Imaging and Applications,"* Proc. SPIE 1986 Cambridge Symposium on Optics and Optoelectronics Advances in Intelligent Systems, vol. 726, Cambridge MA Oct. 28-31, 1986 pp. 264-274.

[5] Ballard, D.H., C.M. Brown, *"Computer Vision"*, Prentice -Hall,Inc., 1982

[6] Ballard, D.H., G.E. Hinton, and T.J. Sejnowski, *"Parallel visual computation"*,. Nature, Vol 306: pp. 21-26. 1983.

[7] Barnard, E., D. Casasent, *"Image processing for image understanding with neural nets"*, in IJCNN - IEEE INNS International Joint Conference on Neural Networks. Washington: IEEE Neural Networks Council. pp. I-111 - I-115. 1989.

[8] Batcher, K.E., "Design of Massively Parallel Processor, IEEE Transaction on Comp., Vol. C29, pp. 836-840, 1980.

[9] Baylor, A., "Photoreceptor Signal and Vision", Invest. ophtamologiy vision SCI., Vol. 28, pp. 34-49, 1987.

[10] Bélanger, J., J.P. Cormier, R. Houde, J.F. Méthot, M. Parizeau, Z. Shen, M. Tremblay, *"Expérimentation de la méthode Shape from Shading"*, Complément de vision numérique GEL-63938, mai 1989.

[11] Blais, F., M. Rioux, J. Domey, *"Compact Three-Dimensional Camera for Robot and Vehicle Guidance"*, Optics and Lasers in Engineering, Vol 10, pp. 227-239, 1989.

[12] Brown, D., A. Scott, *"Design Rules and Process Parameters for the Northern*

Telecom CMOS4S Process", Report IC90-01, Société Canadienne de Microélectronique, février 1990.

[13] Burt, P.J., "*Smart Sensing in Machine Vision*", Machine Vision, H. Freeman Editor, Academic Press, 1988.

[14] Canny, J.F., "*Finding Edge and Lines in Images*", Ph.D. Theses, Massachusetts Institute of Technologie, Artificial Intelligence Laboratory, June 1983.

[15] Chamberlain, S.G., J.P.Y. Lee, "*A Novel Wide Dynamic Range Silicon Photodetector and Linear Imaging Aray*" IEEE Journal of Solid-State Circuit, VOL. Sc-19, N0.1, February 1984.

[16] Chen, K., A. Aström, P.-E. Danielson, "*PASIC. A Smart Sensor for Computer Vision*", Proc. 10th International Conference on Pattern Recognition, V.2, pp. 286 - 291, Atlantic City, juin 1990.

[17] Datacube, 4 Dearborn Road, Peabody, MA. "*Max Video family Image Processors*"

[18] Haralick, R.M., S.R. Sternberg, X. Zhuang, "*Image Analysis Using Mathematical Morphology*". IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. PAMI-9, #4, July 1987.

[19] Horn, B.K.P., "*Robot Vision,*" McGraw-Hill MIT Press, Cambridge MA, 1986.

[20] Horn, B.K.P., "*Obtaining Shape from Shading Imformation.*" Chap. 4 dans The Psychology of Computer Vision, P.H. Winston (ed.), McGraw-Hill, New York, pp. 115-155.

[21] Knight, T.M., "*Design of an Integrated Optical Sensor with On-Chip Preprocessing*", Ph.D. Theses, Massachusetts Institute of Technologie, June 1983.

[22] Koch, C., *et al.*, "*Computing motion using resistive networks*, in *Neural Information Processing Systems*", D.Z. Anderson, Editor. American Institute of Physics: New York. pp. 422-431. 1988.

[23] Liu, S.-C., J.G. Harris. "*Generalized smoothing networks in early vision*" in the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'89). San Diego, California, June 4-8: IEEE. pp. 184-191. 1989.

[24] Mahowald, M.A., C. Mead, "*The Silicon Retina*",. Scientific American, Vol 264(5): pp. 76-82. 1991.

[25] Marr, D., T. Poggio, "*Cooperative Computation of Stereo Disparity*", Science, Vol 194: pp. 283-287. 1976.

[26] Marr, D., "*Early Processing of Visual Information*", Philosophical Transactions of the Royal Society of London B, Vol 245: pp. 483-519. 1976.

[27] Marr, D.C., T. Poggio, "*From understanding computation to understanding neural circuitry*", Neurosciences Res. Prog. Bull., Vol 15(3): pp. 470-488. 1977.

[28] Marr, D., T. Poggio, "*A computational theory of human stereo vision*", Proceedings of the Royal Society of London, Series B, Vol 204: pp. 301-328. 1979.

[29] Marr, D., E. Hildreth, "*Theory of edge detection*",. Proceedings of the Royal

Society of London, Series B, Vol 207: p. 187-217. 1980.

[30] Marr, D., *"Vision - A Computational Investigation into the Human Representation and Processing of Visual Information"*, San Francisco: Freeman. 1982.

[31] Mayer, M.A.C., *et al.*, *"Implementing Neural Architectures Using Analog VLSI Circuit"*, IEEE Trans. Circuits & Syst., Vol 36(5): pp. 643-652. 1989.

[32] Mead, C.A., *"Silicon Models of Neural Computing"*, IEEE First International Conference on Neural Networks, Vol 1, pp. 91-106, Juin 1987.

[33] Mead, C.A., M.A. Mahowald, *"A Silicon Model of Early Visual Processing*. Neural Networks", Vol 1: pp. 91-97. 1988.

[34] Mead, C.A., *"Analog VLSI and Neural Systems"* Addison-Wesley Publishing Company, 1989.

[35] Plummer, B.T., *"Quick Reference Manuel for Silicon Integrated Circuit Technology"*, Wiley-Interscience Publication, 1985.

[36] Pogio, T.A., A.L. Yuille, "Scaling Theorem for Zero Crossing", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol PAMI-8, NO. 1, Jan. 1986.

[37] Poussart, D., M., Tremblay, *"VLSI Implementation of Focal Plane Processing for Smart Vision Sensing"*, Pattern Recognition: Architectures, Algorithms and Applications, Plamondon R.et Cheng H., éditeurs, World Scientific Publishing Co., 1991.

[38] Rioux, M., F. Blais, *"Compact Three-Dimensional Camera for Robotic Application,"* Journal of the Optical Society of America A, vol. 3, pp.1518-1521, Sept. 1986.

[39] Rojer, A.S., E.L. Schwartz, *"Design Consideration for a Space-Variant Visual Sensor with Complex-Logarithmic Geometry"*, Proc. 10th International Conference on Pattern Recognition, V.2, pp. 278 - 285, Atlantic City, juin 1990.

[40] Savaria, Y., *"Conception et Vérification des Circuits VLSI"* Éditions de l'École Polytechnique de Montréal, 1988.

[41] Sivilotti, M.A., M.A. Mahowald, C.A. Mead. *"Real-time visual computations using analog CMOS processing arrays"*, in Advanced Ressearch in VLSI: Proceedings of the 1987 Stanford Conference. Stanford: The MIT Press. pp. 295-312. 1987.

[42] Spiegel, J., F. Kreider, C. Claiys, I. Debusschere, G. Sandini, P. Dario, F. Fantini, P. Belluti, G. Soncini, *"A Foveated retina-like sensor using CCD technology"* Analog VLSI Implementation of Neural Networks, C. Mead and M. Ismail editors, Kluwer, Boston, 1989.

[43] Sze, S.M., *"Physics of Semiconductor Devices"*, Second Edition, Wiley-Interscience Publication, 1981.

[44] Tremblay, M., D. Poussart, *"MAR: An Integrated System for Focal Plane Edge-Tracking with Parallel Analog Filtering and Built-in Primitives for Image Acquisition and Analysis"*, Proc. 10th International Conference on Pattern Recognition, V.2, pp. 292 - 298, Atlantic City, juin 1990.

[45] Tremblay, Y., *"Émulateur logiciel et microcode pour système MAR"*, Projet d'étudiant d'été CRSNG, Septembre 1989.

[46] Tucker, L.W., G.G. Robertson, *"Architecture and Applications of the Connection Machine"*, IEEE computer, pp. 26-38, Août 1988.

[47] Wang, G., D. Renshaw, P.B. Denyer, M.Lu, *"CMOS Video Camera"*, EURO ASIC '91, 1991

[48] Weste, N., K. Eshraghian, *"Principles of CMOS VLSI Design A System Perspective"*, Addison Wesley, 1985.

[49] Yachida, M., S. Tsuji, X.-Q. Huang, *"WIRESIGHT - A Computer Vision System for 3-D Measurement and Recognition of Flexible Wire Using Cross-Stripe Light,"* Proc. 6th IEEE International Conference on Pattern Recognition, Munich, West Germany, pp. 220-222, Oct. 19-22, 1982.

[50] Zornetzer, S.F., J.L. Davis, C. Lau, ed. *"An Introduction to Neural and Electronic Networks"*, Academic Press: San Diego, California. 1990.

ANNEXE A

Conditionnement du Signal Analogique

A.1 Compensation du courant de fuite et de la composante continue

La dernière version du capteur MAR comprend un module de compensation du courant de fuite et de la composante continue qui est ajouté à la sortie de chaque canal analogique distinct (91). Tel que présenté à la section 3.1.4, une photo-diode dans l'obscurité laisse passer un courant de fuite non-nul appelé "courant noir". Ce courant a une intensité proportionnelle à la tension de polarisation inverse aux bornes de la diode PIN [21][43]. On utilise un transistor environ 5 fois plus large que celui de la source de courant d'un pixel (le transistor M_1 de la Figure 3.8) pour ajouter une composante continue sur chaque canal comme montré à la Figure A.1.

On remarque que les deux signaux de commande de ce module V_{Doff} et V_{Goff} sont communs à tous les drains et grilles des transistors de compensation. Tout comme pour les pixels, on assume que le procédé de fabrication est suffisamment homogène et que les transistors de compensation sont identiques. On peut donc, de l'extérieur du capteur, réaliser un circuit qui commande ces deux signaux en plaçant une tension de référence sur

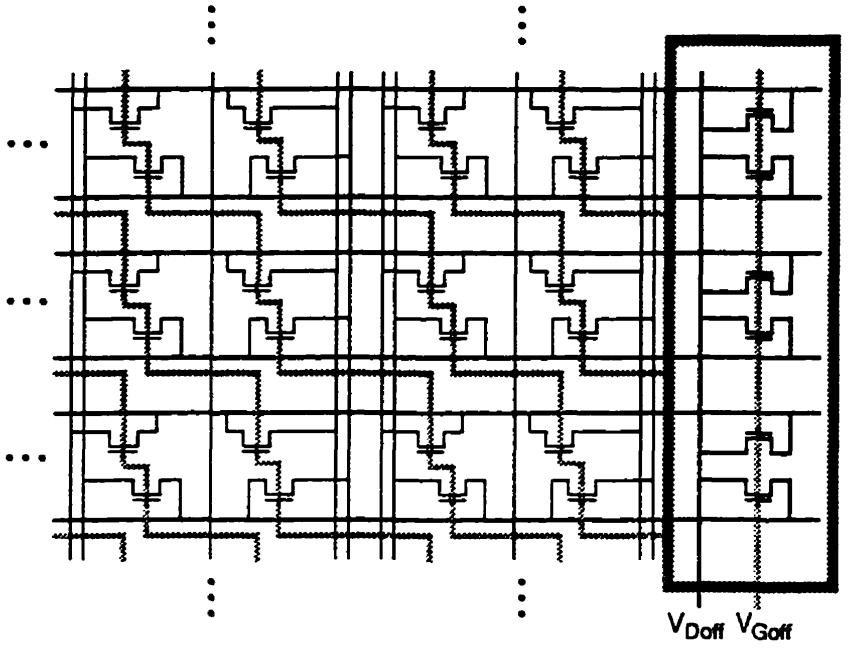


Figure A.1 *Compensateur de courant de fuite et correction de la composante continue. Un transistor N est ajouté pour chaque sortie analogique. Les tensions de grille et de drain de l'ensemble de ces 91 transistors sont connectées afin d'assurer une compensation uniforme.*

le noeud V_{Doff} et en ajustant la tension V_{Goff} pour provoquer une réponse nulle pour un pixel dans le noir.

On peut d'ailleurs voir à la Figure A.2 l'ensemble des circuits externes nécessaire au fonctionnement correct du capteur MAR. On doit initialement fixer les deux tensions de polarisation V_{Doff} et V_{pp} pour garantir qu'on utilise transistor M_1 en mode de saturation comme il est décrit à la section 3.1.4, de même que pour le transistor M_6 de la Figure A.2 (typiquement, $V_{pp} \approx 4.0V$ et $V_{Doff} \approx 7.0V$). On ajuste la valeur de V_{pp} à l'aide du potentiomètre P_1. On doit par le suite définir la valeur continue du signal V_{Goff} à l'aide du potentiomètre P_3 en s'assurant de bloquer le circuit intégrateur ($R_{eset} = 1$). On ajuste la composante continue du signal V_{Goff} de façon à avoir (dans le noir) $V_o = 0.0V$ soit:

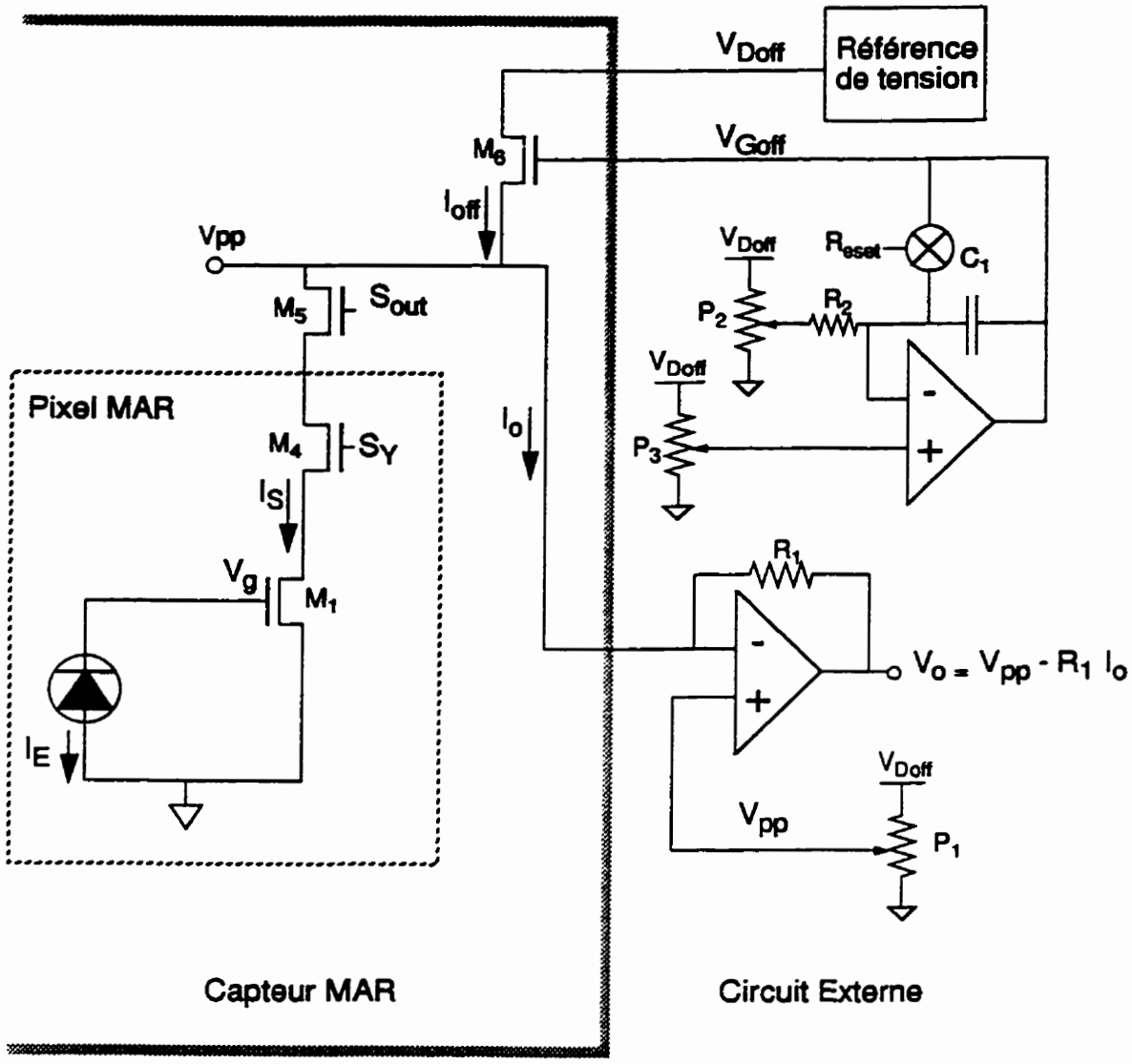


Figure A.2 *Conversion courant tension et illustration du processus de compensation du courant de fuite pour un pixel dans le noir. On montre un seul pixel et un seul canal analogique avec son propre convertisseur courant-tension. Le circuit de génération des signaux V_{Doff}, V_{Goff} et V_{pp} est unique.*

$$I_o = I_{off} - I_S = \frac{V_{pp}}{R_1} \qquad \text{(A-1)}$$

Le potentiomètre P_2 ajuste la pente du signal de compensation de courant de fuite V_{Goff}. Le signal V_{Goff} prend alors la forme de la Figure A.3. On doit rappeler que cette

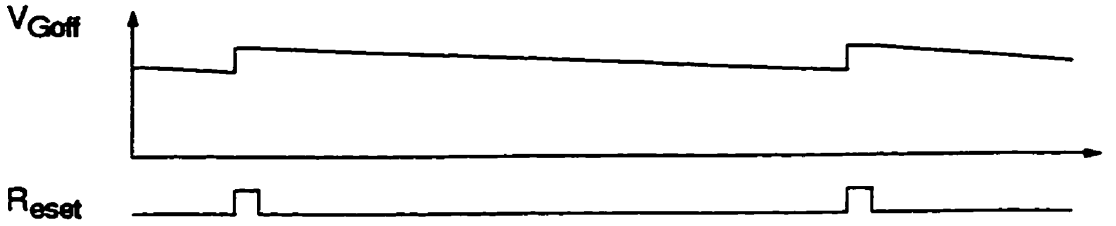


Figure A.3 Forme d'onde temporelle du signal V_{Goff}. Le circuit intégrateur est réinitialisé en même temps que la matrice de pixels et diminue le courant I_{off} de façon à compenser virtuellement le courant de fuite.

procédure de correction du courant de fuite est essentielle car tous les pixels du capteur MAR sont initialisés simultanément et que, après la période d'intégration écoulée, les pixels doivent rester figés pour la durée entière du balayage du capteur. Pendant le balayage du capteur, la valeur de la tension de grille V_g de chaque pixel dérive légèrement, ce qui implique une diminution du courant I_S des derniers pixels visités. Pour utiliser la compensation du courant de fuite afin de définir une référence uniforme, on doit limiter le niveau maximum de blanc avant que le pixel MAR sature (par exemple $V_g > 1.0V$), puis assumer que des signaux V_g de tous les pixels dérivent ensembles (ce qui n'est pas rigoureusement exact puisque le courant de fuite est proportionnel à la tension V_g [21]). On compense alors la diminution du courant I_S par une diminution du courant de correction I_{off}.

Ce circuit de compensation du courant de fuite de même que la correction de la composante continue n'est pas requis si on analyse les passages par zéro de l'image convoluée avec le laplacien d'une gaussienne car la réponse à ce type de filtre est essentiellement insensible au niveau de la composante continue de l'image.

ANNEXE B

Microcode du Système MAR

B.1 Format du microcode et généralité sur la logique programmable

Le microcode qui est présenté à la section B.2 requiert un minimum d'explications préliminaires afin de pouvoir en comprendre le contenu. On doit d'abord préciser que le format global de la représentation texte du microcode est le suivant:

EEEEEEEEEEEEEEEE SSSSS|SSSSSSSS|SSSSSSSSS Commentaire

où "E" représente l'état des variables d'entrées, "S" l'état à chacune des sorties lorsque cette ligne est sélectionnée et que le symbole "|" n'est qu'un simple séparateur. L'espace sert à délimiter les trois principaux champs d'une ligne de microcode. On remarque qu'un court commentaire est ajouté à chaque ligne du microcode et qu'une ligne débutant par le caractère "#" sert a ajouter un commentaire long et est ignorée.

A chaque coup d'horloge, l'état des sortie "S" du PLA est mis à jour et chaque sortie peut prendre la valeur logique "1" ou "0". L'état de l'ensemble des variables d'entrée détermine la ligne du microcode qui sera exécutée. On permet à certaines variables d'entrée de prendre n'importe quelle valeur lorsque le caractère "x" est utilisé pour décrire cet état des variables d'entrée. Par exemple, la ligne codée "01x11" sera sélectionnée lorsque l'état des entrées est "01111" ou encore "01011".

A chaque coup d'horloge, une ou plusieurs lignes de microcode sont sélectionnées et les sorties sont placées à "0" ou à "1" conformément à cette (ces) ligne(s) de code. Si plusieurs lignes sont sélectionnées, chaque sortie réagit comme un "ou" câblé et elle prendra la valeur "1" si au moins une des lignes sélectionnées place un "1" pour cette sortie. Cela a pour conséquence que si aucune ligne n'est sélectionnée, toutes les sorties seront à "0". Cette caractéristique est également utilisée par le module d'arbitration des instruction afin de définir une façon de stopper une instruction ou d'inclure des cycles d'attente entre l'exécution de deux instructions. On choisit donc l'état "11111" (1F) comme première ligne de microcode à être exécutée de façon à ce que la ligne spéciale d'interruption d'une instruction puisse forcer l'état 1F et ainsi commander le début d'une nouvelle instruction.

On présente au Tableau B.1 la description des entrées du PLA de même que celle de ses sorties au Tableau B.2. La section B.2 donne la liste des 158 instructions qui définissent le microcode du contrôleur MAR. Un minimum de commentaires est inclus à ce code et sa compréhension requiert un travail considérable. Une version étendue du même microcode est présentée à la section B.2 où le même code est repris instruction par instruction. Dans cette version, qui est utilisée exclusivement à des fin explicatives, certaines lignes sont répétées plusieurs fois mais la compréhension du microcode peut se faire pour une seule instruction à la fois

Tableau B.1 Description des entrées du microcode du contrôleur MAR

Numéro Entrée	Code Commentaire	Description
0	##	option #2 (déplacement hélicoïdal)
1	o1	option #1
2	o2	option #0
3	eZ	détection de débordement du compteur externe E
4	cZ	détection de débordement du compteur externe C
5	nZ	détection de débordement du compteur externe N
6	C	Exécution conditionnelle et détection des frontières du déplacement. Force un cycle d'attente si il n'y a pas d'instruction à exécuter
7	P2	Détection d'un passage par zéro entre les deux derniers points visités
8	II	Interruption de l'instruction (retourne à l'état 1F)
9:12	In	Instruction proprement dite: OPCOCE<0,3,2,1>
13:17	Ein	Etat de la machine microcodée: lire de droite à gauche. Etat_in<0,1,2,3,4>

Tableau B.2 Description des sorties du microcode du contrôleur MAR

Numéro Sorties	Code Commentaire	Description
4:0	out	Etat de la machine après le coup d'horloge. Etat<4,3,2,1,0>
5	M*	$\overline{\text{MOVE}}$: utilisé pour annuler un déplacement si M* = 1
8:6	D	définit la direction de base: 678 -> (S_DIR_R, S_DIR_I, S_DIR_i-1) 100 -> absolu (direction du registre d'instruction (r)), 010 -> relatif à la dernière direction (i), 001 -> relatif à la direction sub-précédente (i-1)
11:9	R	OFFSET<2:0>: ajouter ce nombre (en base 6) à la direction de base pour définir la nouvelle direction de déplacement (calcul en modulo 6)
12	SM	Déplacement lent: implique un cycle d'attente avant le prochain déplacement
13	RA	Actif lorsque le microcode boucle en attente d'une nouvelle instruction
14	mv	Move Write Enable: Indique un déplacement régulier où on doit mémoriser la description d'état
15	Nr	Remettre le compteur externe N à zéro
16	Ne	Incrémenter le compteur externe N
17	Er	Remettre le compteur externe E à zéro
18	Ee	Incrémenter le compteur externe E
19	ld	Load Instruction: Exécuter une nouvelle instruction
20	D	Détection d'une discontinuité d'arête lors de la poursuite d'arêtes
21	RP	Reset Position: Définition du centre du référentiel image
22	SC	Set Capteur: Active le signal Reset pour le capteur MAR
23	SD	Définit le sens de la discontinuité d'une arête

B.2 Microcode du contrôleur MAR (version intégrale)

```
#18 nombre d'entrees
#24 nombre de sorties
#158 nombre de lignes
# modifie le 27 mars 1991 pour ICALVMRC
#
# Routines generales de rebondissement et d'interruption
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ |*    |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
0xxxxxxxxxxxx11111 11110|0100011|101000000000 1F->1E slow_init move(r)+3
xxxxxxxxxxxxx01111 11011|0100000|101000000000 1E->1B move(r)+0
xxxxxx0x1xxxxxxx00 11111|1000000|100000010000 00-07->1F Ins.Interruption
0xxxxx0x1xxxx0xx10 11111|1000000|100000010000 08-0E->1F Ins.Interruption
0xxxxx0x1xxxx10010 11111|1000000|100000010000 09->1F Ins.Interruption
xxxxxxxx1xxxx11111 11111|1000000|111000010000 1F->1F Ins.Interruption
1xxxxx1xxxxxx11111 11111|1000000|111000010000 1F->1F Force_NOP; ld_instr
x0xxxxxxx0xxx00111 11010|0010011|100000100000 1C->1A move(i)+3; exit
x00xxxxxx1xxx00111 11010|0001010|100000100000 1C->1A move(i-1)+2; exit
x01xxxxxx1xxx00111 11010|0001100|100000100001 1C->1A move(i-1)+4; exit
x10xxxxxx1xxx00111 10100|0001010|100000100000 1C->14 move(i-1)+2 (S+2)
x11xxxxxx1xxx00111 10100|0001100|100000100001 1C->14 move(i-1)+4 (S+4)
x10xxxxxx0xxx00111 10000|0010010|100000100000 1C->10 move(i)+2 (reb+2)
x11xxxxxx0xxx00111 10000|0010100|100000100001 1C->10 move(i)+4 (reb-2)
xxxx0xxxxxxxx11011 11011|1000000|101000000000 1B->1B Boucle_C
xxxx1xxxxxxxx11011 11111|1000000|101000000000 1B->1F continue 1F...
xxxx0xxxxxxxx01011 11010|1000000|101000000000 1A->1A Boucle_C
xxxx1xxxxxxxx01011 11111|1000000|101000010000 1A->1F exit; ld_instr.
# 17 lignes
# Routines generales de boucle C
#
#12345678901234567 01234|5678901|234567890123
# | | enenen
#10ZZZ PI _____ |*    |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
xxxx0xxxxxxxx00001 10000|1000000|101000000000 10->10 Boucle_C
xxxx1xxxxxxxx00001 00000|1000000|101000000000 10->00 return
xxxx0xxxxxxxx10001 10001|1000000|101000000000 11->11 Boucle_C
xxxx1xxxxxxxx10001 00001|1000000|101000000000 11->01 return
xxxx0xxxxxxxx01001 10010|1000000|101000000000 12->12 Boucle_C
xxxx1xxxxxxxx01001 00010|1000000|101000000000 12->02 return
xxxx0xxxxxxxx11001 10011|1000000|101000000000 13->13 Boucle_C
xxxx1xxxxxxxx11001 00011|1000000|101000000000 13->03 return
xxxx0xxxxxxxx00101 10100|1000000|101000000000 14->14 Boucle_C
xxxx1xxxxxxxx00101 00100|1000000|101000000000 14->04 return
xxxx0xxxxxxxx10101 10101|1000000|101000000000 15->15 Boucle_C
xxxx1xxxxxxxx10101 00101|1000000|101000000000 15->05 return
xxxx0xxxxxxxx01101 10110|1000000|101000000000 16->16 Boucle_C
xxxx1xxxxxxxx01101 00110|1000000|101000000000 16->06 return
xxxx0xxxxxxxx11101 10111|1000000|101000000000 17->17 Boucle_C
xxxx1xxxxxxxx11101 00111|1000000|101000000000 17->07 return
xxxx0xxxxxxxx00011 11000|1000000|101000000000 18->18 Boucle_C
xxxx1xxxxxxxx00011 01000|1000000|101000000000 18->08 return
xxxx0xxxxxxxx10011 11001|1000000|101000000000 19->19 Boucle_C
xxxx1xxxxxxxx10011 01001|1000000|101000000000 19->09 return
xxxx0xxxxxxxx11110 01111|1000000|101000000000 0F->0F Boucle_C
xxxx1xxxxxxxx11110 01110|1000000|101000000000 0F->0E return
xxxx0xxxxxxxx10110 01101|1000000|101000000000 0D->0D Boucle_C
xxxx1xxxxxxxx10110 01100|1000000|101000000000 0D->0C return
xxxx0xxxxxxxx11010 01011|1000000|101000000000 0B->0B Boucle_C
xxxx1xxxxxxxx11010 01010|1000000|101000000000 0B->0A return
#26 lignes
```

```
# routines: Miscellenous, f_move, f_move_N
# 1000 , 0000 , 0001
#
#12345678901234567 01234|5678901|234567890123
#
#10ZZZ PI          |*    |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
xxxxxxxx11000xxxxx 11111|1000000|101000010000 1F->1F Instr Interruption
1x0xxx0xx100011111 11111|1000000|101000010100 1F->1F Reset position
1x1xxx0xx100011111 11111|0100000|001000010010 1F->1F Set capteur
1xxxxx0xx000011111 11111|0100000|001000010000 1F->1F move(r)+0; exit
1xxxxx0xx000111111 00001|1000000|001100000000 1F->01 correct. decalage N
xxxxxx0x0000110000 00000|0010000|001010000000 01->00 f_move(i)+0 init.
xxxxx00x0000100000 00000|0010000|001010000000 00->00 f_move(i)+0 boucleN
xxxxxx1xx000100000 00010|0010011|001000000000 00->02 f_move(i)+3 rebond
xxxxxxxxx000101000 11111|0010000|001000010000 02->1F f_move(i)+0 rebond
xxxxx10x0000100000 11111|1000000|001000010000 00->1F exit
#10 lignes
# Move triangulaire et hexagonal
#
# Move_T&Move_H, Move_T_S&Move_H_S, Move_N_T_S&Move_N_H_S
# 1100 depart= (r)+1
# 0100 , 0101 , \ 1101 depart= (r)-1
# inst == 0100 -> option_2 = 0 : move_T & move_H
#
#12345678901234567 01234|5678901|234567890123
#
#10ZZZ PI          |*    |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1xxxxx0xx010x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
1xxxxx0xx110011111 10100|0100001|100101000000 1F->14 move(r)+1 (S_+1)
1xxxxx0xx110111111 10100|0100101|100101000001 1F->14 move(r)+5 (S_-1)
xxxxxx1xxx10xxxxx0 11010|0010011|100000000000 0X->1A move(i)+3; exit
xxx1xx0x0010000000 11111|1000000|100000010000 00->1F exit; ld_instr.
xxx1x10x0010100000 11111|1000000|100000010000 00->1F exit; ld_instr.
xxx1x10x0110x00000 11111|1000000|100000010000 00->1F exit; ld_instr.
xxx0xx0x0010000000 00001|1000000|101000000000 00->01 coin_atteind
xxx0x10x0010100000 00001|1000000|101000000000 00->01 coin_atteind
xxx0x10x0110x00000 00001|1000000|101000000000 00->01 coin_atteind
xxxxx00x0010100000 10000|0010000|100010000000 00->10 move(i)+0;boucle_N
xxxxx00x0110000000 10100|0010010|100010000000 00->14 move(i)+2;boucle_N
xxxxxx0x0110000100 10000|0010100|100000000001 04->10 move(i)+4;boucle_N
xxxxx00x0110100000 10100|0010100|100010000001 00->14 move(i)+4;boucle_N
xxxxxx0x0110100100 10000|0010010|100000000000 04->10 move(i)+2;boucle_N
x00xxx0x0110x10000 10100|0001001|100110100000 01->14 move(i-1)+1;Hex_+1
x01xxx0x0110x10000 10100|0001101|100110100001 01->14 move(i-1)+5;Hex_-1
x10xxx0x0110x10000 10100|0001010|100110100000 01->14 move(i-1)+2;Tri_+2
x11xxx0x0110x10000 10100|0001100|100110100001 01->14 move(i-1)+4;Tri_-2
x00xxx0x0010x10000 10000|0010001|100110100000 01->10 move(i)+1;Hex_+1
x01xxx0x0010x10000 10000|0010101|100110100001 01->10 move(i)+5;Hex_-1
x10xxx0x0010x10000 10000|0010010|100110100000 01->10 move(i)+2;Tri_+2
x11xxx0x0010x10000 10000|0010100|100110100001 01->10 move(i)+4;Tri_-2
#23 lignes
#
```

```
# Routines pour les instructions suivantes:
# Move,Move_N,find_1,Move_N_SetScan_Area(S_A,S_A_serp,S_A_S, S_A_serp_S)
# 0010, 001I , 1010 , 101I et x11x (0110, 0111 , 1110 , 1111 )
#
# ATT.: scan_area* [x11x] -> option_0 = 0 toujours (depart (r)+5)
# ATT.: scan_area* [x11x] -> registre E >= 1
#
#12345678901234567 01234|5678901|234567890123
#
#10ZZZ PI          |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
1xxxxx0xx0x1x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
1x0xxx0xx1x1x11111 10100|0100001|100101000000 1F->14 move(r)+1 (zigzag)
1x1xxx0xx101x11111 10100|0100101|100101000001 1F->14 move(r)+5 (zigzag)
1x1xxx0xx111x11111 11111|1000000|101000010000 1F->1F trap: S_A opt_0 = 1
10xxxx0xx101011111 11101|1100101|101101000000 1F->1D find_1 (not move)
11xxxx0xx101011111 11101|0000000|100101000000 1F->1D find_1 move(101x)
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xxx0xx1xxx01x00000 11100|1000000|101000000000 00->1C Move_N*
xxx1xx1xxx01x00000 11010|0010011|100000000000 00->1A move(i+3); exit
xxxxxx0x0001000000 11111|1010000|100000010000 00->1F exit; Move
xxxxxx0x0001100000 10000|0010000|100010000000 00->10 move(i)+0 move_N
xxxxxx0x0011x00000 10000|0010000|100000000000 00->10 move(i)+0 S_A
xx0xxxxx01x1x00100 10000|0010100|100000000001 04->10 move(i)+4
xx1xxxxx0101x00100 10000|0010010|100000000000 04->10 move(i)+2
xx0xxx0x0101x00000 10100|0010010|100010000000 00->14 move(i)+2 move_N_S
xx0xxx0x0111x00000 10100|0100001|100000000000 00->14 move(r)+1 S_A_ser_S
xx1xxx0x0101x00000 10100|0010100|100010000001 00->14 move(i)+4 move_N_S
#19 lignes
1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
#6 lignes
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI          |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
xxxxxx1xxx11000000 10011|0100011|101000000000 00->13 move(r)+3
00xxxx1xxx11100000 10011|0100010|100011000000 00->13 move(r)+2
01xxxx1xxx11100000 10011|0100100|100011000001 00->13 move(r)+4
1xxxxx1xxx11100000 00011|1000000|100001000000 00->03 not move (causal)
xxxxxx0x0x11011000 00011|0100011|001000000000 03->03 f_move(r)+3
xxxxxx1x0x11011000 10010|0100000|101000000000 03->12 move(r)+0
x0xxxx1xxx11x01000 10001|0100001|101011000000 02->11 move(r)+1
x1xxxx1xxx11x01000 10001|0100101|101011000001 02->11 move(r)+5
xxx0xx1xx011111000 10011|0100011|100000100000 03->13 move(r)+3
x0x1xx1xxx11x1x000 11010|0100100|100000000001 01,03->1A move(r)+4; exit
x1x1xx1xxx11x1x000 11010|0100010|100000000000 01,03->1A move(r)+2; exit
xxxxxx0x0x11111000 00010|1000000|101001000000 03->02
xxx0xx1xx111111000 10111|0100010|100000100001 03->17 move(r)+2
xxxxxxxxx111111100 10111|0100100|100000000000 07->13 move(r)+4
xxxxxx0x0011101000 10010|0100011|100000000000 02->12 move(r)+3
xxxxxx0x0111101000 10110|0100010|100000000000 02->16 move(r)+2
xxxxxxxxx111101100 10010|0100100|100000000001 06->12 move(r)+4
xxx0xx1xx011x10000 10001|0100000|100000100000 01->11 move(r)+0
xxxxxx0x0x11x10000 10000|0100000|100001000000 01->10 move(r)+0
xxx0xx1xx111x10000 10101|0100001|100000100000 01->15 move(r)+1
xxxxxxxxx111x10100 10001|0100101|100000000001 05->11 move(r)+5
#22 lignes
```

```
# Routines de recherche d'arête
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI __ ____ ___ |*  _ _  |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
110xxx0xx100111111 11101|0100001|100101000000 1F->1D move(r)+1
111xxx0xx100111111 11101|0100101|100101000000 1F->1D move(r)+5
10xxxx0xx100111111 11101|1000000|100101000000 1F->1D not_move
0xxxxx1xx1010xxx00 11010|0010011|100000000000 00-07->1A move(i)+3; exit
0xxxxx1xx1001xxx00 11010|0010011|100000000000 00-07->1A move(i)+3; exit
0xxxxx1xxxxxx0xx10 11010|0010011|100000000000 08-0E->1A move(i)+3; exit
0xxxxx1xxxxxx10010 11010|0010011|100000000000 09 ->1A move(i)+3; exit
xxxxx10x01010xxxx0 11111|1000000|000000010000 0X->1F exit
xxxxx10x01001xxxx0 11111|1000000|000000010000 0X->1F exit
xxx1xx0x01010xxxx0 11111|1000000|000000010000 0X->1F exit
xxx1xx0x01001xxxx0 11111|1000000|000000010000 0X->1F exit
00xxxx0xxxxxx10111 01111|0100000|101001000000 1D->0F move(r)+0
010xxx0xxxxxx10111 11000|1000000|101001001001 1D->18 not_move
011xxx0xxxxxx10111 11001|1000000|101001001000 1D->19 not_move
0xxxxx000xxxx01110 01111|0010000|101000100000 0E->0F move(i)+0
0x0xxx010xxxx01110 11000|0010010|100011001001 0E->18 move(i)+2
0x1xxx010xxxx01110 11001|0010100|100011001000 0E->19 move(i)+4
0xxxxx010xxxx00010 11001|0010100|100011000000 08->19 move(i)+4
0xxxxx010xxxx10010 11000|0010010|100011000000 09->18 move(i)+2
0xxxxx000xxxx00010 01011|0010010|101000000000 08->0B move(i)+2
xxxxxx00010xx10000 11001|0010101|100010100000 01->19 move(i)+5; E++
0xxxxx000xxxx10010 01101|0010100|101000000000 09->0D move(i)+4
xxxxxx00010xx10100 11000|0010001|100010100000 05->18 move(i)+1; E++
0xxxxx000xxxx01010 10001|0010011|101000000000 0A->11 move(i)+3
0xxxxx010101001010 11001|0010011|100010001001 0A->19 move(i)+3
0xxxxx010100101010 10111|0010011|100010001001 0A->17 move(i)+3
0xxxxx000xxxx00110 10101|0010011|101000000000 0C->15 move(i)+3
0xxxxx010101000110 11000|0010011|100010001000 0C->18 move(i)+3
0xxxxx010100100110 10011|0010011|100010001000 0C->13 move(i)+3
0xxxxx010100111100 10011|0010001|100011000000 07->13 move(i)+1
xxxxxx010100111000 10111|0010101|100011000000 03->17 move(i)+5
0xxxxx000100111100 10110|0010011|101000000000 07->16 move(i)+3
xxxxxx0x0100101100 11001|0010010|100010001000 06->19 move(i)+2
xxxxxx000100111000 10010|0010011|101000000001 03->12 move(i)+3
xxxxxx0x0100101000 11000|0010100|100010001000 02->18 move(i)+4
#35 lignes
```

B.3 Microcode du contrôleur MAR (version étendue)

```
18 nombre d'entrees
24 nombre de sorties
158 nombre de lignes
# modifie le 27 mars 1991 pour ICALVMRC
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _________ |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
0xxxxxxxxxxxx11111 11110|0100011|101000000000 1F->1E slow_init move(r)+3
xxxxxxxxxxxxx01111 11011|0100000|101000000000 1E->1B move(r)+0
xxxxxx0x1xxxxxxx00 11111|1000000|100000010000 00-07->1F Ins.Interruption
0xxxxx0x1xxxx0xx10 11111|1000000|100000010000 08-0E->1F Ins.Interruption
0xxxxx0x1xxxx10010 11111|1000000|100000010000 09->1F Ins.Interruption
xxxxxxxx1xxxx11111 11111|1000000|111000010000 1F->1F Ins.Interruption
1xxxxx1xxxxxx11111 11111|1000000|111000010000 1F->1F Force_NOP; ld_instr
x0xxxxxxx0xxx00111 11010|0010011|100000100000 1C->1A move(i)+3; exit
x00xxxxxx1xxx00111 11010|0001010|100000100000 1C->1A move(i-1)+2; exit
x01xxxxxx1xxx00111 11010|0001100|100000100001 1C->1A move(i-1)+4; exit
x10xxxxxx1xxx00111 10100|0001010|100000100000 1C->14 move(i-1)+2 (S+2)
x11xxxxxx1xxx00111 10100|0001100|100000100001 1C->14 move(i-1)+4 (S+4)
x10xxxxxx0xxx00111 10000|0010010|100000100000 1C->10 move(i)+2 (reb+2)
x11xxxxxx0xxx00111 10000|0010100|100000100001 1C->10 move(i)+4 (reb-2)
xxxx0xxxxxxxx11011 11011|1000000|101000000000 1B->1B Boucle_C
xxxx1xxxxxxxx11011 11111|1000000|101000000000 1B->1F continue 1F...
xxxx0xxxxxxxx01011 11010|1000000|101000000000 1A->1A Boucle_C
xxxx1xxxxxxxx01011 11111|1000000|101000010000 1A->1F exit; ld_instr.
# 17 lignes
# Routines generales de boucle C
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _________ |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
xxxx0xxxxxxxx00001 10000|1000000|101000000000 10->10 Boucle_C
xxxx1xxxxxxxx00001 00000|1000000|101000000000 10->00 return
xxxx0xxxxxxxx10001 10001|1000000|101000000000 11->11 Boucle_C
xxxx1xxxxxxxx10001 00001|1000000|101000000000 11->01 return
xxxx0xxxxxxxx01001 10010|1000000|101000000000 12->12 Boucle_C
xxxx1xxxxxxxx01001 00010|1000000|101000000000 12->02 return
xxxx0xxxxxxxx11001 10011|1000000|101000000000 13->13 Boucle_C
xxxx1xxxxxxxx11001 00011|1000000|101000000000 13->03 return
xxxx0xxxxxxxx00101 10100|1000000|101000000000 14->14 Boucle_C
xxxx1xxxxxxxx00101 00100|1000000|101000000000 14->04 return
xxxx0xxxxxxxx10101 10101|1000000|101000000000 15->15 Boucle_C
xxxx1xxxxxxxx10101 00101|1000000|101000000000 15->05 return
xxxx0xxxxxxxx01101 10110|1000000|101000000000 16->16 Boucle_C
xxxx1xxxxxxxx01101 00110|1000000|101000000000 16->06 return
xxxx0xxxxxxxx11101 10111|1000000|101000000000 17->17 Boucle_C
xxxx1xxxxxxxx11101 00111|1000000|101000000000 17->07 return
xxxx0xxxxxxxx00011 11000|1000000|101000000000 18->18 Boucle_C
xxxx1xxxxxxxx00011 01000|1000000|101000000000 18->08 return
xxxx0xxxxxxxx10011 11001|1000000|101000000000 19->19 Boucle_C
xxxx1xxxxxxxx10011 01001|1000000|101000000000 19->09 return
xxxx0xxxxxxxx11110 01111|1000000|101000000000 0F->0F Boucle_C
xxxx1xxxxxxxx11110 01110|1000000|101000000000 0F->0E return
xxxx0xxxxxxxx10110 01101|1000000|101000000000 0D->0D Boucle_C
xxxx1xxxxxxxx10110 01100|1000000|101000000000 0D->0C return
xxxx0xxxxxxxx11010 01011|1000000|101000000000 0B->0B Boucle_C
xxxx1xxxxxxxx11010 01010|1000000|101000000000 0B->0A return
#
```

```
#*********************************************************#
# INSTRUCTION = 0 #
# F_MOVE #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ ___ |*  _ _ |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1xxxxx0xx000011111 11111|0100000|001000010000 1F->1F move(r)+0; exit
#
#*********************************************************#
# INSTRUCTION = 1 #
# MISCELLANOUS #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ ___ |*  _ _ |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
xxxxxxxx11000xxxxx 11111|1000000|101000010000 1F->1F Instr Interruption
1x0xxx0xx100011111 11111|1000000|101000010100 1F->1F Reset position
1x1xxx0xx100011111 11111|0100000|001000010010 1F->1F Set capteur
#
#*********************************************************#
# INSTRUCTION = 2 #
# F_MOVE_N #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ ___ |*  _ _ |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1xxxxx0xx000111111 00001|1000000|001100000000 1F->01 correct. decalage N
xxxxxx0x0000110000 00000|0010000|001010000000 01->00 f_move(i)+0 init.
xxxxx00x0000100000 00000|0010000|001010000000 00->00 f_move(i)+0 boucleN
xxxxxx1xx000100000 00010|0010011|001000000000 00->02 f_move(i)+3 rebond
1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
xxxxxxxxx000101000 11111|0010000|001000010000 02->1F f_move(i)+0 rebond
xxxxx10x0000100000 11111|1000000|001000010000 00->1F exit
#
```

```
#*********************************************************#
# INSTRUCTION = 3 (opt_2 = 0) #
# FIND_1 (30 degre) #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI          |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
110xxx0xx100111111 11101|0100001|100101000000 1F->1D move(r)+1
111xxx0xx100111111 11101|0100101|100101000000 1F->1D move(r)+5
10xxxx0xx100111111 11101|1000000|100101000000 1F->1D not_move
0xxxxx1xx1001xxx00 11010|0010011|100000000000 00-07->1A move(i)+3; exit
0xxxxx1xxxxxx0xx10 11010|0010011|100000000000 08-0E->1A move(i)+3; exit
0xxxxx1xxxxxx10010 11010|0010011|100000000000 09 ->1A move(i)+3; exit
xxxxx10x01001xxxx0 11111|1000000|000000010000 0X->1F exit
xxx1xx0x01001xxxx0 11111|1000000|000000010000 0X->1F exit
00xxxx0xxxxxx10111 01111|0100000|101001000000 1D->0F move(r)+0
010xxx0xxxxxx10111 11000|1000000|101001001001 1D->18 not_move
011xxx0xxxxxx10111 11001|1000000|101001001000 1D->19 not_move
0xxxxx000xxxx01110 01111|0010000|101000100000 0E->0F move(i)+0
0x0xxx010xxxx01110 11000|0010010|100011001001 0E->18 move(i)+2
0x1xxx010xxxx01110 11001|0010100|100011001000 0E->19 move(i)+4
0xxxxx010xxxx00010 11001|0010100|100011000000 08->19 move(i)+4
0xxxxx010xxxx10010 11000|0010010|100011000000 09->18 move(i)+2
0xxxxx000xxxx00010 01011|0010010|101000000000 08->0B move(i)+2
xxxxxx00010xx10000 11001|0010101|100010100000 01->19 move(i)+5; E++
0xxxxx000xxxx10010 01101|0010100|101000000000 09->0D move(i)+4
xxxxxx00010xx10100 11000|0010001|100010100000 05->18 move(i)+1; E++
0xxxxx000xxxx01010 10001|0010011|101000000000 0A->11 move(i)+3
0xxxxx010100101010 10111|0010011|100010001001 0A->17 move(i)+3
0xxxxx000xxxx00110 10101|0010011|101000000000 0C->15 move(i)+3
0xxxxx010100100110 10011|0010011|100010001000 0C->13 move(i)+3
0xxxxx010100111100 10011|0010001|100011000000 07->13 move(i)+1
xxxxxx010100111000 10111|0010101|100011000000 03->17 move(i)+5
0xxxxx000100111100 10110|0010011|101000000000 07->16 move(i)+3
xxxxxx0x0100101100 11001|0010010|100010001000 06->19 move(i)+2
xxxxxx000100111000 10010|0010011|101000000001 03->12 move(i)+3
xxxxxx0x0100101000 11000|0010100|100010001000 02->18 move(i)+4

1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
#
```

```
#*********************************************************#
# INSTRUCTION = 4 #
# MOVE #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI __ ___ ___ |•  __  |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1xxxxx0xx0x1x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xxx0xx1xxx01x00000 11100|1000000|101000000000 00->1C Move N*
xxx1xx1xxx01x00000 11010|0010011|100000000000 00->1A move(i+3); exit
xxxxxx0x0001000000 11111|1010000|100000010000 00->1F exit; Move

1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
#
```

```
#*********************************************************#
# INSTRUCTION = 5 (opt_2 = 0) #
# FIND_0 (60 degre) #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI          |*    |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1x0xxx0xx1x1x11111 10100|0100001|100101000000 1F->14 move(r)+1 (zigzag)
1x1xxx0xx101x11111 10100|0100101|100101000001 1F->14 move(r)+5 (zigzag)
10xxxx0xx101011111 11101|1100101|101101000000 1F->1D find_1 (not_move)
11xxxx0xx101011111 11101|0000000|100101000000 1F->1D find_1 move(101x)
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xxx0xx1xxx01x00000 11100|1000000|101000000000 00->1C Move_N*
xxx1xx1xxx01x00000 11010|0010011|100000000000 00->1A move(i+3); exit
xx0xxxxx01x1x00100 10000|0010100|100000000001 04->10 move(i)+4
xx1xxxxx0101x00100 10000|0010010|100000000000 04->10 move(i)+2
xx0xxx0x0101x00000 10100|0010010|100010000000 00->14 move(i)+2 move_N_S
xx1xxx0x0101x00000 10100|0010100|100010000001 00->14 move(i)+4 move_N_S
0xxxxx1xx1010xxx00 11010|0010011|100000000000 00-07->1A move(i)+3; exit
0xxxxx1xxxxxx0xx10 11010|0010011|100000000000 08-0E->1A move(i)+3; exit
0xxxxx1xxxxxx10010 11010|0010011|100000000000 09 ->1A move(i)+3; exit
xxxxx10x01010xxxx0 11111|1000000|000000010000 0X->1F exit
xxx1xx0x01010xxxx0 11111|1000000|000000010000 0X->1F exit
00xxxx0xxxxxx10111 01111|0100000|101001000000 1D->0F move(r)+0
010xxx0xxxxxx10111 11000|1000000|101001001001 1D->18 not_move
011xxx0xxxxxx10111 11001|1000000|101001001000 1D->19 not_move
0xxxxx000xxxx01110 01111|0010000|101000100000 0E->0F move(i)+0
0x0xxx010xxxx01110 11000|0010010|100011001001 0E->18 move(i)+2
0x1xxx010xxxx01110 11001|0010100|100011001000 0E->19 move(i)+4
0xxxxx010xxxx00010 11001|0010100|100011000000 08->19 move(i)+4
0xxxxx010xxxx10010 11000|0010010|100011000000 09->18 move(i)+2
0xxxxx000xxxx00010 01011|0010010|101000000000 08->0B move(i)+2
xxxxxx00010xx10000 11001|0010101|100010100000 01->19 move(i)+5; E++
0xxxxx000xxxx10010 01101|0010100|101000000000 09->0D move(i)+4
xxxxxx00010xx10100 11000|0010001|100010100000 05->18 move(i)+1; E++
0xxxxx000xxxx01010 10001|0010011|101000000000 0A->11 move(i)+3
0xxxxx010101001010 11001|0010011|100010001001 0A->19 move(i)+3
0xxxxx000xxxx00110 10101|0010011|101000000000 0C->15 move(i)+3
0xxxxx010101000110 11000|0010011|100010001000 0C->18 move(i)+3

1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
#
```

```
#*********************************************************#
# INSTRUCTION = 6 #
# MOVE_N #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ _____ |*  _ _ |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
1xxxxx0xx0x1x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xxx0xx1xxx01x00000 11100|1000000|101000000000 00->1C Move_N*
xxx1xx1xxx01x00000 11010|0010011|100000000000 00->1A move(i+3); exit
xxxxxx0x0001100000 10000|0010000|100010000000 00->10 move(i)+0 move_N
1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX

#*********************************************************#
# INSTRUCTION = 7 (opt2 = 0) #
# MOVE_N_S #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ _____ |*  _ _ |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
1x0xxx0xx1x1x11111 10100|0100001|100101000000 1F->14 move(r)+1 (zigzag)
1x1xxx0xx101x11111 10100|0100101|100101000001 1F->14 move(r)+5 (zigzag)
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xxx0xx1xxx01x00000 11100|1000000|101000000000 00->1C Move_N*
xxx1xx1xxx01x00000 11010|0010011|100000000000 00->1A move(i+3); exit
xx0xxxxx01x1x00100 10000|0010100|100000000001 04->10 move(i)+4
xx1xxxxx0101x00100 10000|0010010|100000000000 04->10 move(i)+2
xx0xxx0x0101x00000 10100|0010010|100010000000 00->14 move(i)+2 move_N_S
xx1xxx0x0101x00000 10100|0010100|100010000001 00->14 move(i)+4 move_N_S
1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
#
```

```
#*********************************************************#
# INSTRUCTION = 8 (opt2 = 0) #
# MOVE_T & MOVE_H #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ |*  |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
1xxxxx0xx010x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
xxxxxx1xxx10xxxxx0 11010|0010011|100000000000 0X->1A move(i)+3; exit
xxx1xx0x0010000000 11111|1000000|100000010000 00->1F exit; ld_instr.
xxx0xx0x0010000000 00001|1000000|101000000000 00->01 coin_atteind
x00xxx0x0010x10000 10000|0010001|100110100000 01->10 move(i)+1;Hex_+1
x01xxx0x0010x10000 10000|0010101|100110100001 01->10 move(i)+5;Hex_-1
x10xxx0x0010x10000 10000|0010010|100110100000 01->10 move(i)+2;Tri_+2
x11xxx0x0010x10000 10000|0010100|100110100001 01->10 move(i)+4;Tri_-2

1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX

#*********************************************************#
# INSTRUCTION = 9 (opt2 = 0) #
# MOVE_N_T_S & MOVE_N_H_S #
#*********************************************************#
#************** depart = (r) + 1 ***********************#
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ |*  |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
1xxxxx0xx110011111 10100|0100001|100101000000 1F->14 move(r)+1 (S_+1)
xxxxxx1xxx10xxxxx0 11010|0010011|100000000000 0X->1A move(i)+3; exit
xxx1x10x0110x00000 11111|1000000|100000010000 00->1F exit; ld_instr.
xxx0x10x0110x00000 00001|1000000|101000000000 00->01 coin_atteind
xxxxx00x0110000000 10100|0010010|100010000000 00->14 move(i)+2;boucle_N
xxxxxx0x0110000100 10000|0010100|100000000001 04->10 move(i)+4;boucle_N
x00xxx0x0110x10000 10100|0001001|100110100000 01->14 move(i-1)+1;Hex_+1
x01xxx0x0110x10000 10100|0001101|100110100001 01->14 move(i-1)+5;Hex_-1
x10xxx0x0110x10000 10100|0001010|100110100000 01->14 move(i-1)+2;Tri_+2
x11xxx0x0110x10000 10100|0001100|100110100001 01->14 move(i-1)+4;Tri_-2

1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
#
```

```
#*********************************************************#
# INSTRUCTION = 10 #
# MOVE_N_T & MOVE_N_H #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI          |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
1xxxxx0xx010x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
xxxxxx1xxx10xxxxx0 11010|0010011|100000000000 0X->1A move(i)+3; exit
xxx1x10x0010100000 11111|1000000|100000010000 00->1F exit; ld_instr.
xxx0x10x0010100000 00001|1000000|101000000000 00->01 coin_atteind
xxxxx00x0010100000 10000|0010000|100010000000 00->10 move(i)+0;boucle_N
x00xxx0x0010x10000 10000|0010001|100110100000 01->10 move(i)+1;Hex_+1
x01xxx0x0010x10000 10000|0010101|100110100001 01->10 move(i)+5;Hex_-1
x10xxx0x0010x10000 10000|0010010|100110100000 01->10 move(i)+2;Tri_+2
x11xxx0x0010x10000 10000|0010100|100110100001 01->10 move(i)+4;Tri_-2
1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX

#*********************************************************#
# INSTRUCTION = 11 (opt2 = 0) #
# MOVE_N_T_S & MOVE_N_H_S #
#*********************************************************#
#************** depart = (r) - 1 ***********************#
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI          |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEE1DRSS
1xxxxx0xx110111111 10100|0100101|100101000001 1F->14 move(r)+5 (S_-1)
xxxxxx1xxx10xxxxx0 11010|0010011|100000000000 0X->1A move(i)+3; exit
xxx1x10x0110x00000 11111|1000000|100000010000 00->1F exit; ld_instr.
xxx0x10x0110x00000 00001|1000000|101000000000 00->01 coin_atteind
xxxxx00x0110100000 10100|0010100|100010000001 00->14 move(i)+4;boucle_N
xxxxxx0x0110100100 10000|0010010|100000000000 04->10 move(i)+2;boucle_N
x00xxx0x0110x10000 10100|0001001|100110100000 01->14 move(i-1)+1;Hex_+1
x01xxx0x0110x10000 10100|0001101|100110100001 01->14 move(i-1)+5;Hex_-1
x10xxx0x0110x10000 10100|0001010|100110100000 01->14 move(i-1)+2;Tri_+2
x11xxx0x0110x10000 10100|0001100|100110100001 01->14 move(i-1)+4;Tri_-2

1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX
#
```

```
#*********************************************************#
# INSTRUCTION = 12 #
# Scan_Area #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ |* ___ |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1xxxxx0xx0x1x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xxxxxx0x0011x00000 10000|0010000|100000000000 00->10 move(i)+0 S_A
xxxxxx1xxx11000000 10011|0100011|101000000000 00->13 move(r)+3
xxxxxx0x0x11011000 00011|0100011|001000000000 03->03 f_move(r)+3
xxxxxx1x0x11011000 10010|0100000|101000000000 03->12 move(r)+0
x0xxxx1xxx11x01000 10001|0100001|101011000000 02->11 move(r)+1
x1xxxx1xxx11x01000 10001|0100101|101011000001 02->11 move(r)+5
x0x1xx1xxx11x1x000 11010|0100100|100000000001 01,03->1A move(r)+4; exit
x1x1xx1xxx11x1x000 11010|0100010|100000000000 01,03->1A move(r)+2; exit
xxx0xx1xx011x10000 10001|0100000|100000100000 01->11 move(r)+0
xxxxxx0x0x11x10000 10000|0100000|100001000000 01->10 move(r)+0
1xx0x00x0xxxx00000 11000|0010101|100000000000 00->18 move(i)+5 S_A_HEX
1xxxxxxxxxxxx00010 11001|0010001|101000000000 08->19 move(i)+1 S_A_HEX
1xxxxxxxxxxxx10010 01011|0010001|101000000000 09->0B move(i)+1 S_A_HEX
1xxxxxxxxxxxx01010 01101|0010001|101000000000 0A->0D move(i)+1 S_A_HEX
1xxxxxxxxxxxx00110 01111|0010001|101000000000 0C->0F move(i)+1 S_A_HEX
1xxxxxxxxxxxx01110 10111|0010001|101000000000 0E->17 move(i)+1 S_A_HEX
1xxxxxxxxxxxx11100 10000|0010010|101000000000 07->10 move(i)+2 S_A_HEX

#*********************************************************#
# INSTRUCTION = 13 (opt2 = 0) #
# Scan_Area_S #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _____ |* ___ |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1x0xxx0xx1x1x11111 10100|0100001|100101000000 1F->14 move(r)+1 (zigzag)
1x1xxx0xx111x11111 11111|1000000|101000010000 1F->1F trap: S_A opt_0 = 1
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xx0xxxxx01x1x00100 10000|0010100|100000000001 04->10 move(i)+4
xx0xxx0x0111x00000 10100|0100001|100000000000 00->14 move(r)+1 S_A_ser_S
xxxxxx1xxx11000000 10011|0100011|101000000000 00->13 move(r)+3
xxxxxx0x0x11011000 00011|0100011|001000000000 03->03 f_move(r)+3
xxxxxx1x0x11011000 10010|0100000|101000000000 03->12 move(r)+0
x0xxxx1xxx11x01000 10001|0100001|101011000000 02->11 move(r)+1
x1xxxx1xxx11x01000 10001|0100101|101011000001 02->11 move(r)+5
x0x1xx1xxx11x1x000 11010|0100100|100000000001 01,03->1A move(r)+4; exit
x1x1xx1xxx11x1x000 11010|0100010|100000000000 01,03->1A move(r)+2; exit
xxxxxx0x0x11x10000 10000|0100000|100001000000 01->10 move(r)+0
xxx0xx1xx111x10000 10101|0100001|100000100000 01->15 move(r)+1
xxxxxxxxx111x10100 10001|0100101|100000000001 05->11 move(r)+5
#
```

```
#*********************************************************#
# INSTRUCTION = 14 (opt2 = 0) #
# Scan_Area_Serpentin #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _______ |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1xxxxx0xx0x1x11111 10000|0100000|100101000000 1F->10 move(r)+0 (rect.)
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xxxxxx0x0011x00000 10000|0010000|100000000000 00->10 move(i)+0 S_A
00xxxx1xxx11100000 10011|0100010|100011000000 00->13 move(r)+2
01xxxx1xxx11100000 10011|0100100|100011000001 00->13 move(r)+4
1xxxxx1xxx11100000 00011|1000000|100001000000 00->03 not move (twice)
x0xxxx1xxx11x01000 10001|0100001|101011000000 02->11 move(r)+1
x1xxxx1xxx11x01000 10001|0100101|101011000001 02->11 move(r)+5
xxx0xx1xx011111000 10011|0100011|100000100000 03->13 move(r)+3
x0x1xx1xxx11x1x000 11010|0100100|100000000001 01,03->1A move(r)+4; exit
x1x1xx1xxx11x1x000 11010|0100010|100000000000 01,03->1A move(r)+2; exit
xxxxxx0x0x11111000 00010|1000000|101001000000 03->02
xxxxxx0x0011101000 10010|0100011|100000000000 02->12 move(r)+3
xxx0xx1xx011x10000 10001|0100000|100000100000 01->11 move(r)+0
xxxxxx0x0x11x10000 10000|0100000|100001000000 01->10 move(r)+0

#*********************************************************#
# INSTRUCTION = 15 (opt2 = 0) #
# Scan_Area_Serpentin_S #
#*********************************************************#
#
#12345678901234567 01234|5678901|234567890123

#10ZZZ PI _______ |*     |MAvrered PCD
#ooecnC2I/In\/Ein\ /out\|M/D\/R\|SRmNNEElDRSS
1x0xxx0xx1x1x11111 10100|0100001|100101000000 1F->14 move(r)+1 (zigzag)
1x1xxx0xx111x11111 11111|1000000|101000010000 1F->1F trap: S_A opt_0 = 1
xxxxx10x0xx1x0x000 11111|1000000|100000010000 (00,02)->1F exit
xxx1xx0x0xx1x00000 11111|1000000|100000010000 00->1F exit; tout les cas
xx0xxxxx01x1x00100 10000|0010100|100000000001 04->10 move(i)+4
xx0xxx0x0111x00000 10100|0100001|100000000000 00->14 move(r)+1 S_A_ser_S
00xxxx1xxx11100000 10011|0100010|100011000000 00->13 move(r)+2
01xxxx1xxx11100000 10011|0100100|100011000001 00->13 move(r)+4
1xxxxx1xxx11100000 00011|1000000|100001000000 00->03 not move (twice)
x0xxxx1xxx11x01000 10001|0100001|101011000000 02->11 move(r)+1
x1xxxx1xxx11x01000 10001|0100101|101011000001 02->11 move(r)+5
x0x1xx1xxx11x1x000 11010|0100100|100000000001 01,03->1A move(r)+4; exit
x1x1xx1xxx11x1x000 11010|0100010|100000000000 01,03->1A move(r)+2; exit
xxxxxx0x0x11111000 00010|1000000|101001000000 03->02
xxx0xx1xx111111000 10111|0100010|100000100001 03->17 move(r)+2
xxxxxxxxx111111100 10111|0100100|100000000000 07->13 move(r)+4
xxxxxx0x0111101000 10110|0100010|100000000000 02->16 move(r)+2
xxxxxxxxx111101100 10010|0100100|100000000001 06->12 move(r)+4
xxxxxx0x0x11x10000 10000|0100000|100001000000 01->10 move(r)+0
xxx0xx1xx111x10000 10101|0100001|100000100000 01->15 move(r)+1
xxxxxxxxx111x10100 10001|0100101|100000000001 05->11 move(r)+5
```

ANNEXE C

Langage Assembleur pour le Système MAR

Réalisé par André Hamel

Hiver 1991

C.1 INTRODUCTION

Le système MAR est un processeur rétinien spécialisé dans le traitement d'images de bas niveaux pour la vision robotique. Le système comprend une machine à états qui exécute une séquence microcodée dans le but d'effectuer certaines séquences de balayage, détection et suivi d'arête, croissance de région, interpolation sub-pixel ainsi que d'autres fonctions plus simples. Le système MAR possède un registre d'instructions de 16 bits et un certain nombre de registres de configuration qui lui permettent d'être vu par un processeur hôte comme un périphérique esclave.

Le présent rapport résulte d'un projet de fin d'étude consistant en la réalisation d'un compilateur pour le Langage Assembleur. La syntaxe de ce dernier fut développé en collaboration avec M. Marc Tremblay et M. Florent Parent.

C.2 Le Langage Assembleur

Le Langage Assembleur a été conçu dans le but de simplifier la conception d'application utilisant le système MAR. Celui-ci par l'usage explicite de mnémoniques facilite le travail du concepteur en lui enlevant la lourde tâche qui était d'utiliser le microcode. Auparavant le concepteur devait configurer un registre de 16 bits au niveau binaire pour réaliser l'instruction voulue. Par exemple lorsqu'il voulait faire un MOVE

(INST=0100) avec ses options (direction, condition, ...), il devait initialiser le registre comme suit: INS_REG=0x4334; (en format hexadécimal).

INS_REG[15:0]:

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	1	0	0	0	0	1	1	0	0	1	1	0	1	0	0

Maintenant pour réaliser cette instruction en Langage Assembleur, on n'a qu'a inclure la ligne suivante dans un programme écrit en C:

MOVE INSIDE_WINDOW DIR3 INPUT5;

Comme on peut le voir, l'instruction ci-haut est beaucoup plus explicite que la précédente. Le Langage Assembleur fait partie intégrante de l'environnement logiciel du système MAR.

Les outils retenus pour la conception du compilateur sont lex (lexical analysis program generator) et le langage C.

C.3 Conception du compilateur

L'utilisation de lex pour concevoir le compilateur facilita grandement le travail. Le compilateur fut développé sur un poste de travail SUN et son compilateur C fut utilisé, ainsi que Dbxtool...

La fonction du compilateur ou du préprocesseur est de convertir les instructions du Langage Assembleur en instructions en code C, ainsi ces dernières pourront être compilées avec le compilateur C. La forme des instructions en code C est simplement une initialisation de registres servant à configurer le système MAR.

Trois registres sont utilisés. Le premier INS_REG[15:0] qui est le registre d'instruction du système MAR et les deux autres registres: COMPTEUR_N et COMPTEUR_E servant de compteur pour les différentes instructions. Dans le deuxième document ces registres sont présentés et documentés.

Exemple:

FAST_MOVE_N 234;

Lorsque le compilateur rencontrera l'instruction ci-haut il remplacera celle-ci par:

/* FAST_MOVE_N 234; */

COMPTEUR_N= -234;

INS_REG= 0x2020;

Comme on peut le voir, ces trois lignes sont compatibles avec la syntaxe C.

C.3.1 Compilation

La source du compilateur se nomme source.l et après avoir généré le fichier lex.yy.c (lex source.l <ENTER>) on la compile de la façon suivante:

tutorial% cc -ll -o preprocesseur lex.yy.c

Le fichier de sortie se nomme "preprocesseur" et c'est le fichier qui fait la conversion du Langage Assembleur en langage C.

C.3.2 Redirection de l'entrée et de la sortie

Un fichier de type "batch" fut utilisé pour rediriger l'entrée et la sortie standard. Son contenu est le suivant:

(preprocesseur <$1) >$2

$1 correspond au premier paramètre lors de l'appel.

$2 correspond au deuxième paramètre lors de l'appel.

Ainsi, lorsque ce fichier est exécuté il exécute le programme nommé "preprocesseur" (le compilateur) tout en redirigant l'entrée et la sortie selon les paramètres. Ce fichier se nomme martoc. Comme il fait la conversion d'instructions du Langage Assembleur (propre au système MAR) en code C, d'où l'appellation mar to C.

C.3.3 Utilisation

Lorsque l'application contenant des instructions du Langage Assembleur (nommé source.mar) et que l'on veut la compiler. On doit d'abord utiliser martoc de la façon suivante:

tutorial% martoc source.mar source.c <ENTER>

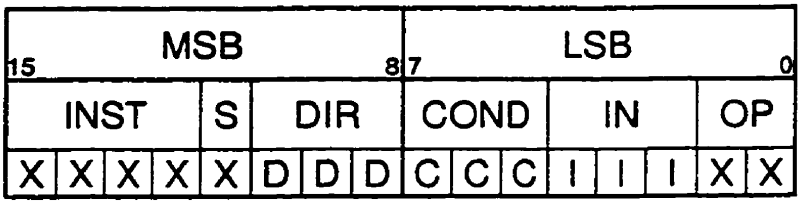
Le fichier "source.c" est maintenant compilable avec le compilateur C de unix;

tutorial% cc -o program source.c <ENTER>

et "program" est le nom du fichier exécutable.

C.4 DESCRIPTION DES CHAMPS

Le Langage Assembleur présenté dans ces pages servira a initialiser le registre d'instruction INS_REG[15:0] suivant:



Cette section donne les règles pour les options ou les champs d'instructions du Langage Assembleur.

Chaque instruction possède un mot clé, l'instruction elle-même, suivie de plusieurs champs optionnels. Exemple:

MOVE_N [DIR#d] [#displacement] [condition] [INPUT#i]

[[#max]bounce_option [bounce_direction]];

L'instruction ci-haut possède comme mot clé MOVE_N. Chaque champ supplémentaire lorsqu'il est entre crochet est optionnel et chaque option peut être imbriquée l'une dans l'autre. Un état ou une valeur numérique seront pris par défaut lorsqu'ils ne seront pas présents. L'ordre des options n'est pas important mais le point virgule à la fin de l'instruction est nécessaire.

Les mots clés sont en majuscule et ils doivent être écrits comme tel lorsqu'on veut les utiliser. Ces derniers ne peuvent commencer à la colonne 0 (i.e. ils doivent obligatoirement être séparés du début de la ligne d'instruction par un espace ou une tabulation) et une instruction peut prendre deux lignes pour être déclarée.

Lorsque l'option est écrite en minuscule cela signifie que le champ doit être remplacé par un mot clé (ce dernier peut être l'option d'un autre mot clé). Lorsque le champ écrit en minuscule est précédé par un dièse (#) ceci signifie qu'il doit être remplacé par une valeur entière sous forme décimal. Exemple [#displacement] est une constante numérique (i.e. représente un nombre). et [condition] est une constante alphabétique (i.e. représente un mot clé en majuscule).

De plus, les constantes numériques (i.e. celles précédées par un dièse) peuvent être remplacées par une variable ou une expression déclarées par l'utilisateur. Pour faire appel à une variable ou une expression, on doit les placer entre parenthèses là où la constante numérique devait se placer. Exemples:

FAST_MOVE DIR6;

FAST_MOVE DIR(nom_de_la_variable);

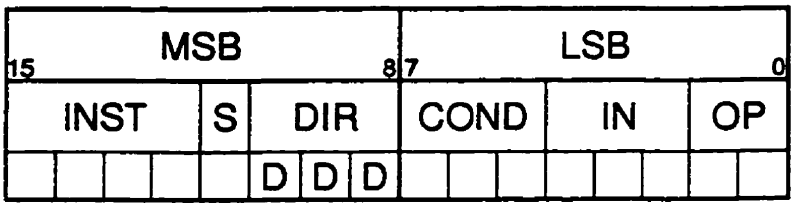
FAST_MOVE DIR(f/(a+5*c)+c/g);

Les espaces entrent chaque champ doivent être respectées lorsqu'elles sont présentes ou absentes. Les "underscore" (_) présents dans les instructions sont obligatoires.

Dans les pages qui vont suivent la définition de chaque champs est présentée sous forme de tableau dont le format est uniforme. La première colonne donne la valeur binaire du champs de l'instruction considéré. La deuxième colonne donne le mot clé à utiliser lorsqu'il est présent et la troisième colonne fournie les commentaires nécessaires. Ainsi, on peut faire facilement le lien entre le microcode et le Langage Assembleur. La valeur par défaut est toujours mise entre parenthèse.

C.4.1 Champ direction: [DIR#d]

Cette option possède comme mot clé DIR suivi immédiatement d'un chiffre allant de 0 à 6 (exemple: DIR4). Si cette option n'est pas présente la direction 0 sera prise comme état par défaut. L'état par défaut est toujours entouré de parenthèses.



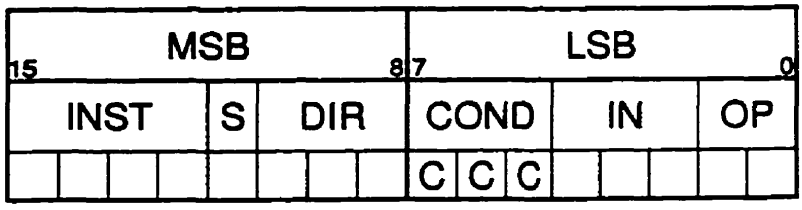
[DIR#d]

DDD	DIR#d	Commentaire	degré
(000)	DIR0	direction nulle, i.e. reste sur place, condition par défaut	
001	DIR1	direction 1, vers l'est.	0
010	DIR2	direction 2, vers le nord-nord-est	60
011	DIR3	direction 3, vers le nord-nord-ouest.	120
100	DIR4	direction 4, vers l'ouest.	180
101	DIR5	direction 5, vers le sud-sud-ouest.	240
110	DIR6	direction 6, vers le sud-sud-est.	300

INS_REG[10:8]

C.4.2 Champ condition: [condition]

Ce champ d'instruction laisse à l'utilisateur le choix des conditions limitant la fenêtre d'étude. Ils suivent la correspondance suivante:



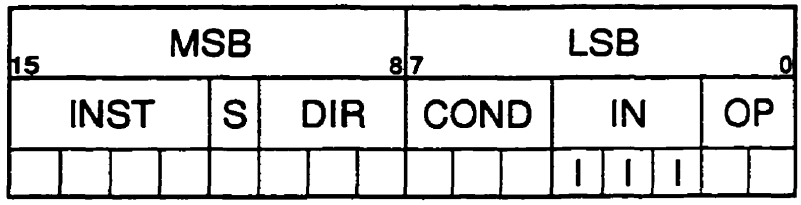
[condition]

CCC	condition	Commentaire
000	ALWAYS	inconditionnel.
(001)	INSIDE_WINDOW	dépassement des limites en x ou en y, condition par défaut.
010	NOT_EXTERNAL_0	EXT_COND<0> ou dépassement.
011	NOT_EXTERNAL_1	EXT_COND<1> ou dépassement.
100	NOT_ZERO_P2	Zero P2 ou dépassement.
101	NOT_EXTERNAL_2	EXT_COND<2> ou dépassement.
110	ZERO_P2	Zero P2* ou dépassement.
111	EXTERNAL_2	$\overline{\text{EXT\_COND<2>}}$ ou dépassement.

INS_REG[7:5]

C.4.3 Champ de sélection d'entrée: [INPUT#i]

Ce champ permet à l'utilisateur de sélectionner le filtre du module de calcul analogique qui sera utilisé par le contrôleur. Huit entrées peuvent être utilisées, elles sont numérotées de 0 à 7 et l'entrée 0 est prise par défaut. Exemple: INPUT4.



[INPUT#i]

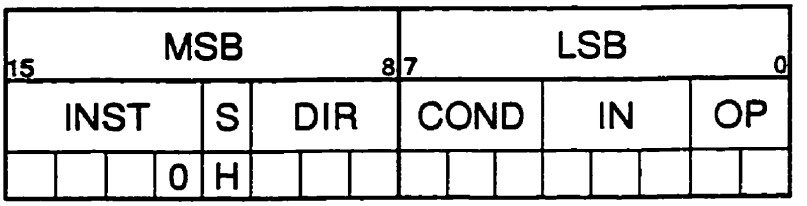
III	INPUT#i	Commentaire
(000)	INPUT0	entrée numéro 0, entrée par défaut.
001	INPUT1	entrée numéro 1
010	INPUT2	entrée numéro 2
011	INPUT3	entrée numéro 3
100	INPUT4	entrée numéro 4
101	INPUT5	entrée numéro 5
110	INPUT6	entrée numéro 6
111	INPUT7	entrée numéro 7

INS_REG[4:2]

C.4.4 champ S: [HELIX]

Cette option est valide pour les instructions MOVE_N, MOVE_HEX_N, MOVE_TRI_N et SCAN_AREA RASTER. Cette condition implique de façon générale que le bit 12 soit "0"(INS_REG[12]=0).

Lorsque cette option est présente elle fait précéder chaque déplacements linéaire par une visite des 6 voisins immédiat à la manière d'un déplacement hélicoïdale.



[HELIX]

H	HELIX	Commentaire
(0)		OPT2=0, le bit S=0.
1	HELIX	OPT2=1, le bit S=1.

INS_REG[11]

C.5 INSTRUCTIONS

Cette section fait la description détaillée du jeu d'instruction complet et des mnémoniques utilisées pour simplifier la programmation de même que la compréhension d'un programme d'exploitation du système MAR.

C.5.1 FAST_MOVE

[DIR#d] [condition] [INPUT#i];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	0	0	0	0	D	D	D	C	C	C	I	I	I	0	0

Déplacement unitaire rapide du point à l'étude.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.

REMARQUES:

- N'exécute pas de boucle C.
- Cette instruction peut amener le point d'intérêt sur la frontière (i.e. sur condition_active=1) et l'y laisser même si la condition d'exécution comprend la frontière de l'image.

C.5.2 FAST_MOVE_N

[DIR#d] [#displacement] [condition] [INPUT#i];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	0	1	0	0	D	D	D	C	C	C	I	I	I	0	0

Déplacement rapide du point à l'étude selon le nombre de pixels donnés par le paramètre #displacement.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.

[#displacement] Nombre entier de 16 bits. Le compteur N sera initialisé à cette valeur. Si l'option #displacement n'est pas présente, la dernière valeur sera prise par défaut (i.e. le compteur N ne sera pas modifié).

REMARQUES:

- N'exécute pas de boucle C. (déplacement d'un pixel par cycle d'horloge)
- Le signal Stop externe peut interrompre cette instruction.
- L'instruction ne se termine jamais sur une frontière.
- Si la condition sélectionnée est autre que le dépassement des limites x et y, le respect de la condition n'est pas assuré car les signaux ZéroP2, ZéroP3 ou les conditions externes ne sont probablement pas valides au moment où le contrôleur vérifie leur état.

C.5.3 FIND

[mode] [START_direction] [IMMEDIATE] [#numberCROSSING]

[DIR#d] [condition] [INPUT#i];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	M	M	M	0	D	D	D	C	C	C	I	I	I	E	T

Cette instruction permet la recherche et la poursuite d'arête ou de frontières.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.

[START_direction] Contrôle le sens du début de la recherche (option 0)

T	START_direction	Commentaire
(0)	START_ LEFT	tourne dans la direction (i)+2 lorsque l'arête est atteinte. Départ à gauche lorsque recherche d'arête.
1	START_RIGHT	tourne dans la direction (i)-2 lorsque l'arête est atteinte. Départ à droite lorsque recherche d'arête.

[mode] Détermine l'incrément de la direction de recherche à chaque discontinuité d'arête.

MMM	mode	Commentaire
(011)	30	recherche à 30 degré.
101	60	recherche à 60 degré.

[IMMEDIATE] Si cette option est présente le système fait une recherche automatique d'arête (le bit E=0). Débute l'instruction par la recherche de la première arête dans la direction de départ. La valeur par défaut est non immédiate (E=1).

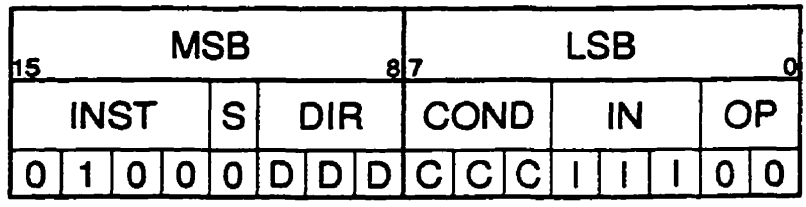
[#numberCROSSING] Nombre entier de 16 bits. Initialise le compteur N et donne le nombre de croisement d'arête que l'instruction devra effectuer avant de terminer l'instruction. Si ce paramètre n'est pas présent l'ancienne valeur sera prise (i.e. le compteur N ne sera pas modifié).

REMARQUES:

- L'instruction peut être interrompue par le signal Stop.
- Lorsque le point à l'étude atteint une frontière, il recule de un pixel et l'instruction prend fin même si le compteur N n'est pas échus.
- Le compteur E est utilisé pour contrôler le mode de recherche "aveugle". Sa valeur indique le nombre de passage par zéro qu'on effectue lorsque l'arête est interrompue. La recherche s'effectue alors dans la dans la dernière orientation d'arête valide un nombre E de point. si la frontière n'est pas de nouveau détectée à ce moment, l'instruction termine.

C.5.4 MOVE

[DIR#d] [condition] [INPUT#i];



Déplace le point à l'étude d'un pixel dans la direction donnée #d puis attend C+1 cycles d'horloge avant de charger une nouvelle instruction.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.

REMARQUES:

- Exécute les boucles C.
- L'instruction ne ce termine jamais sur une frontière.

C.5.5 MOVE_HEX

[DIR#d] [#side] [CORNER_direction] [condition] [INPUT#i];

15 MSB 8								7 LSB 0							
INST				S	DIR			COND			IN			OP	
1	0	0	0	0	D	D	D	C	C	C	I	I	I	0	R

Déplacement en forme d'hexagone de côté unitaire du pixel d'intérêt débutant dans la direction donnée #d.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.

[#side] Nombre entier de 8 bits. Détermine le nombre de côtés qui sont longés. Le compteur E sera initialisé à cette valeur. Si le paramètre #side n'est pas présent l'ancienne valeur utilisée sera prise (i.e. le compteur E ne sera pas modifié). Pour exécuter le déplacement d'un hexagone complet (une seule révolution), on utilise l'option "6".

[CORNER_direction] Indique le sens de rotation (rebondissement)aux coins

(frontières).

R	CORNER_direction	Commentaire
(0)	CORNER_LEFT	incrément de la direction, (i)+1.
1	CORNER_RIGHT	décrément de la direction, (i)-1.

REMARQUES:

- Exécute les boucles C.
- Les déplacement peuvent être arrêtés par le signal Stop externe.
- Si une frontière est rencontrée, le point à l'étude est reculé d'un pixel puis l'instruction est interrompue.
- L'instruction ne se termine jamais sur une frontière.

C.5.6 MOVE_HEX_N

[DIR#d] [condition] [#side_length] [#numberSIDE]

[CORNER_direction] [ZIGZAG [START_direction]] [HELIX] [INPUT#i];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
1	0	M	Z	H	D	D	D	C	C	C	I	I	I	0	R

Fait décrire un patron hexagonal au pixel d'intérêt. Les côtés sont parcourus en zigzag si l'option ZIGZAG est présente (le bit Z=1) sinon les côtés sont parcourus de façon rectiligne (M=1 et Z=0). Le départ du zigzag se fait dans la direction définie par l'option START_direction (défini le bit M), valide seulement si l'option ZIGZAG est présente (Z=1).

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.
HELIX	voir section C.4.4

[#side_length] Nombre entier de 16 bits. Donne la longueur en pixel du côté de l'hexagone. Le compteur N sera initialisé à cette valeur. Si ce paramètre n'est pas présent l'ancienne valeur sera prise (i.e. le compteur N ne sera pas modifié).

[#numberSIDE] Nombre entier de 8 bits. Donne le nombre de côté qui sont longés. Le compteur E sera initialisé à cette valeur. Si ce paramètre n'est pas présent l'ancienne valeur sera prise (i.e. le compteur E ne sera pas modifié). Par exemple, on utilise cette option pour effectuer le balayage complet d'un hexagone une seule fois: 6SIDE.

[CORNER_direction]

R	CORNER_direction	Commentaire
(0)	CORNER_LEFT	incrément de la direction au coin, (i)+1.
1	CORNER_RIGHT	décrément de la direction au coin, (i)-1.

[ZIGZAG] Si cette option est présente le déplacement s'effectue en zigzag sinon le mode de déplacement est rectiligne. Dans le cas d'un déplacement en mode zigzag, on doit définir la direction de départ à l'aide du code "START_direction".

Z	ZIGZAG	Commentaire
(0)		Déplacement en mode rectiligne
1	ZIGZAG	Déplacement en mode zigzag

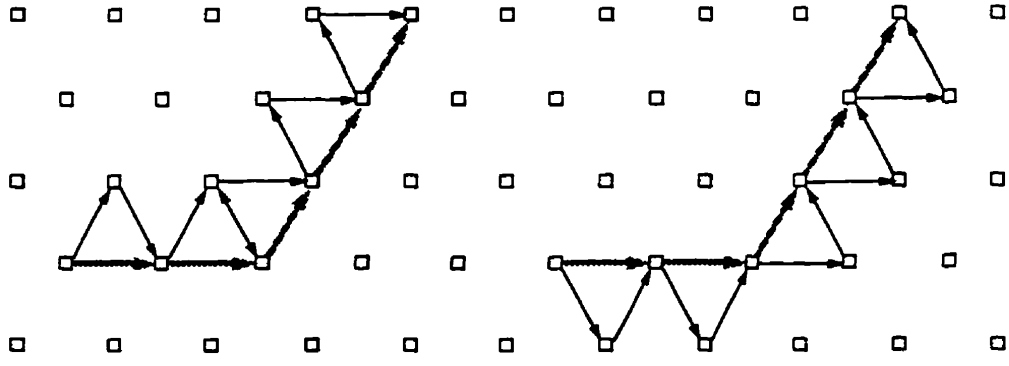
ZIGZAG [START_direction]

M	START_direction	Commentaire
(0)	START_LEFT	départ dans la direction (r)+1.
1	START_RIGHT	départ dans la direction (r)-1.

REMARQUES:

- Exécute les boucles C.
- Les déplacement peuvent être arrêtés par le signal Stop externe.
- Si une frontière est rencontrée, le point à l'étude est reculé d'un pixel puis l'instruction est interrompue.
- L'instruction ne ce termine jamais sur une frontière.
- Si le mode de déplacement en zigzag est choisi, la combinaison des options "START_direction" et "CORNER_direction" définit si le balayage se fait vers l'intérieur de l'hexagone ou vers l'extérieur. Cela a en plus des conséquences sur la façon dont est effectué le balayage dans les coins de l'hexagone.

START_direction	CORNER_direction	Déplacement du pixel d'intérêt p/r à la frontière
(START_LEFT)	(CORNER_LEFT)	INTÉRIEUR
START_LEFT	CORNER_RIGHT	EXTÉRIEUR
START_RIGHT	CORNER_LEFT	EXTÉRIEUR
START_RIGHT	CORNER_RIGHT	INTÉRIEUR



(a) INTÉRIEUR (b) EXTÉRIEUR

Figure C.1 Exemple de balayage d'un coin d'hexagone en mode zigzag. On voit les deux possibilités de déplacement du pixel d'intérêt par rapport à la frontière de déplacement.

C.5.7 MOVE_N

[DIR#d] [#displacement] [condition] [INPUT#i]

[#max] [bounce_option] [bounce_direction]] [HELIX];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	1	1	0	H	D	D	D	C	C	C	I	I	I	B	R

Déplace le pixel d'intérêt d'un nombre N de positions qui est défini par le paramètre #displacement dans la direction donnée #d. La trajectoire de balayage peut être modifiée lorsqu'on atteint la frontière qui est définie par le champ de condition.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.
HELIX	voir section C.4.4

[#displacement] Nombre entier de 16 bits. Le compteur N sera initialisé à cette valeur. Si l'option #displacement n'est pas présente, la dernière valeur sera prise par défaut (i.e. le compteur N ne sera pas modifié).

[bounce_option] Définit comment effectuer la poursuite de l'instruction si on atteint la frontière définie par le champs "condition". Les options "#maxBOUNCE" et "bounce_direction" ont une signification seulement si bounce_option=BOUNCE.

B	bounce_option	Commentaire
(0)	NO_BOUNCE	retour en arrière et interruption, état par défaut.
1	[#max]BOUNCE	rebondissement et poursuite de l'instruction.

[#max] nombre entier de 8 bits. La variable #max donne le nombre d'exception accepté avant d'interrompre la suite des déplacements par le système. Le compteur E sera initialisé à cette valeur. Si l'option #max n'est pas présente, la dernière valeur sera prise par défaut (i.e. le compteur E ne sera pas modifié).

[bounce_direction] (valide si bounce_option=BOUNCE) Définit le sens du rebondissement lorsque le pixel d'intérêt atteint la frontière

R	bounce_direction	Commentaire
(0)	LEFT	le rebond s'effectue dans la direction précédente plus 2, (i)+2.
1	RIGHT	le rebond s'effectue dans la direction précédente moins 2, (i)-2.

REMARQUES:

- Exécute les boucles C.
- Les déplacement de plus d'un pixel peuvent être arrêtés par le signal Stop externe.
- L'instruction peut se terminer sur une frontière si le compteur E est échus (#maxBOUNCE) à la suite de deux déplacement consécutifs qui activent chacun la condition de frontière.

C.5.8 MOVE_TRI

[DIR#d] [#side] [CORNER_direction] [condition] [INPUT#i];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
1	0	0	0	0	D	D	D	C	C	C	I	I	I	1	R

Déplacement en forme de triangle équilatéral de côté unitaire du pixel d'intérêt débutant dans la direction donnée #d.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.

[#side] Nombre entier de 8 bits. Détermine le nombre de côtés qui sont longés. Le compteur E sera initialisé à cette valeur. Si le paramètre #side n'est pas présent l'ancienne valeur utilisée sera prise (i.e. le compteur E ne sera pas modifié). Pour exécuter le déplacement d'un triangle complet (une seule révolution), on utilise l'option "3".

[CORNER_direction] Indique le sens de rotation (rebondissement)aux coins (frontières).

R	CORNER_direction	Commentaire
(0)	CORNER_LEFT	incrément de la direction, (i)+2.
1	CORNER_RIGHT	décrément de la direction, (i)-2.

REMARQUES:

- Exécute les boucles C.
- Les déplacement peuvent être arrêtés par le signal Stop externe.
- Si une frontière est rencontrée, le point à l'étude est reculé d'un pixel puis l'instruction est interrompue.
- L'instruction ne se termine jamais sur une frontière.

C.5.9 MOVE_TRI_N

[DIR#d] [condition] [#side_length] [#numberSIDE]

[CORNER_direction] [ZIGZAG [START_direction]] [INPUT#i] [HELIX];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	0	0	0	0	D	D	D	C	C	C	I	I	I	0	0

Fait décrire un patron triangulaire au pixel d'intérêt. Les côtés sont parcourus en zigzag si l'option ZIGZAG est présente (le bit Z=1) sinon les côtés sont parcourus de façon rectiligne (M=1 et Z=0). Le départ du zigzag se fait dans la direction définie par l'option START_direction (défini le bit M), valide seulement si l'option ZIGZAG est présente (Z=1).

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.
HELIX	voir section C.4.4

[#side_length] Nombre entier de 16 bits. Donne la longueur en pixel du côté du triangle. Le compteur N sera initialisé à cette valeur. Si ce paramètre n'est pas présent l'ancienne valeur sera prise (i.e. le compteur N ne sera pas modifié).

[#numberSIDE] Nombre entier de 8 bits. Donne le nombre de côté qui sont longés. Le compteur E sera initialisé à cette valeur. Si ce paramètre n'est pas présent l'ancienne valeur sera prise (i.e. le compteur E ne sera pas modifié). Par exemple on utilise cet option pour le balayage d'un triangle complet une seule fois: 3SIDE.

[CORNER_direction]

R	CORNER_direction	Commentaire
(0)	CORNER_LEFT	incrément de la direction au coin, (i)+2.
1	CORNER_RIGHT	décrément de la direction au coin, (i)-2.

[ZIGZAG] Si cette option est présente le déplacement s'effectue en zigzag sinon le mode de déplacement est rectiligne. Dans le cas d'un déplacement en mode zigzag, on doit définir la direction de départ à l'aide du code "START_direction".

Z	ZIGZAG	Commentaire
(0)		Déplacement en mode rectiligne
1	ZIGZAG	Déplacement en mode zigzag

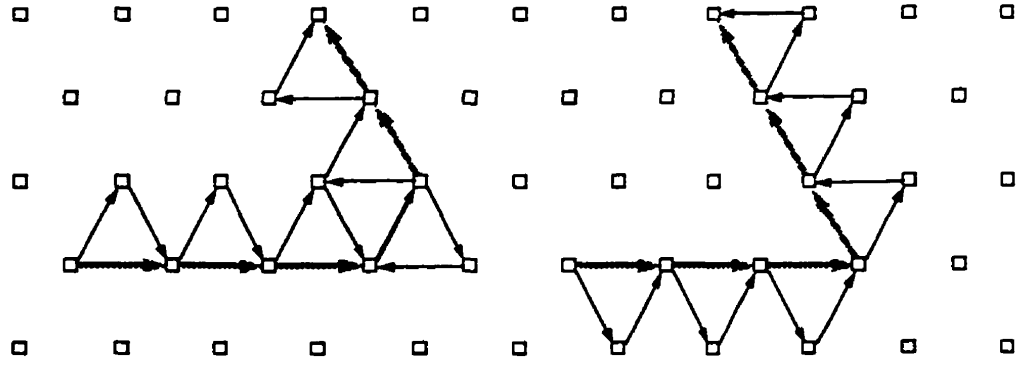
ZIGZAG [START_direction]

M	START_direction	Commentaire
(0)	START_LEFT	départ dans la direction (r)+1.
1	START_RIGHT	départ dans la direction (r)-1.

REMARQUES:

- Exécute les boucles C.
- Les déplacement peuvent être arrêtés par le signal Stop externe.
- Si une frontière est rencontrée, le point à l'étude est reculé d'un pixel puis l'instruction est interrompue.
- L'instruction ne ce termine jamais sur une frontière.
- Si le mode de déplacement en zigzag est choisi, la combinaison des options "START_direction" et "CORNER_direction" définit si le balayage se fait vers l'intérieur du triangle ou vers l'extérieur. Cela a en plus des conséquences sur la façon dont est effectué le balayage dans les coins du triangle

START_direction	CORNER_direction	Déplacement du pixel d'intérêt p/r à la frontière
(START_LEFT)	(CORNER_LEFT)	INTÉRIEUR
START_LEFT	CORNER_RIGHT	EXTÉRIEUR
START_RIGHT	CORNER_LEFT	EXTÉRIEUR
START_RIGHT	CORNER_RIGHT	INTÉRIEUR



(a) INTÉRIEUR (b) EXTÉRIEUR

Figure C.2 Exemple de balayage d'un coin dune triangle équilatéral en mode zigzag. On voit les deux possibilités de déplacement du pixel d'intérêt par rapport à la frontière de déplacement.

C.5.10 MOVE_ZIGZAG_N

[DIR#d] [#displacement] [condition] [INPUT#i]

[#max] [bounce_option] [direction];

15 MSB 8								7 LSB 0							
INST				S	DIR			COND			IN			OP	
0	1	1	1	0	D	D	D	C	C	C	I	I	I	B	R

Déplace le pixel d'intérêt en mode zugzag d'un nombre N de positions qui est défini par le paramètre #displacement dans la direction donnée #d. La trajectoire de balayage peut être modifiée lorsqu'on atteint la frontière qui est définie par le champ de condition. Le sens du zigzag lorsqu'on débute l'instruction est définit par le champs *B* (option 1).

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.
HELIX	voir section C.4.4

[#displacement] Nombre entier de 16 bits. Le compteur N sera initialisé à cette valeur. Si l'option #displacement n'est pas présente, la dernière valeur sera prise par défaut (i.e. le compteur N ne sera pas modifié). Le nombre de déplacement correspond à la distance parcourue dans la direction globale de déplacement soit un déplacement par zigzag complet.

[bounce_option] Définit comment effectuer la poursuite de l'instruction si on atteint la frontière définie par le champs "condition". Les options "#maxBOUNCE" et "bounce_direction" ont une signification particulière si bounce_option=BOUNCE.

B	bounce_option	Commentaire
(0)	NO_BOUNCE	retour en arrière et interruption, état par défaut.
1	[#max]BOUNCE	rebondissement et poursuite de l'instruction.

[#max] nombre entier de 8 bits. La variable #max donne le nombre d'exception accepté avant d'interrompre la suite des déplacements par le système. Le compteur E sera initialisé à cette valeur. Si l'option #Max. n'est pas présente, la dernière valeur sera prise par défaut (i.e. le compteur E ne sera pas modifié).

[direction] Définit le sens du rebondissement lorsque le pixel d'intérêt atteint la frontière. Cette option définit également la direction de départ.

R	direction	Commentaire
(0)	LEFT	le rebond s'effectue dans la direction sub-précédente plus 2, (i-1)+2. (si BOUNCE est actif) Débute avec le zigzag à gauche (départ +1)
1	RIGHT	le rebond s'effectue dans la direction sub-précédente moins 2, (i-1)-2. (si BOUNCE est actif) Débute avec le zigzag à droite (départ -1)

REMARQUES:

- Exécute les boucles C.
- Les déplacement de plus d'un pixel peuvent être arrêtés par le signal Stop externe.
- L'instruction peut se terminer sur une frontière si le compteur E est échus (#maxBOUNCE) à la suite de deux déplacement consécutifs qui activent chacun la condition de frontière.
- Lorsque l'option BOUNCE est choisie, la direction de départ est la même que le sens de rebondissement.

C.5.11 RESET_POSITION

[condition] [X] [Y];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	0	0	1	Y	0	0	0	C	C	C	0	0	0	X	0

Remise à zéro des compteurs de position.Cette instruction sert à déterminer l'origine du système de coordonnée pour en X, en Y ou les deux axes en même temps.

Champs	Commentaire
[condition]	voir section C.4.2.

[X] [Y]

X	Y	[X]	[Y]	Commentaire
(1)	(1)	X	Y	initialise les coordonnées x et y.
1	0	X		initialise la coordonnée x seulement.
0	1		Y	initialise la coordonnée y seulement.

C.5.12 SCAN_AREA

[[mode] [ZIGZAG] [HELIX]] [#number_LINE] [DIR#d] [BACKWARD]

[condition] [INPUT#i] [BORDER#max];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
1	1	M	M	H	D	D	D	C	C	C	I	I	I	B	0

Permet d'effectuer le balayage complet d'une région de l'image délimitée par la condition à la frontière et ce, selon différents modes possibles de balayage. La direction de départ peut être n'importe laquelle des six directions valides ce qui implique que le balayage peut se faire horizontalement (ligne par ligne) ou diagonalement. Plusieurs modes de balayages sont utilisables en passant par le mode "raster" typique du signal vidéo standard jusqu'au balayage hélicoïdal.

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.
[INPUT#i]	voir section C.4.3.

[mode] Définit quel mode de balayage sera effectué.

MM	mode	Commentaire
(00)	RASTER	mode conventionnel (OPT2=0). Déplacement lent dans la direction de départ jusqu'à la frontière définie par le champs condition, retour rapide sur la même ligne jusqu'à la frontière opposée puis, changement de ligne. Semblable au balayage vidéo.
00	RASTER HELIX	voir section C.4.4 (OPT2=1 ou H=1). Comme RASTER à l'exception que le mode de déplacement est hélicoïdal. Le retour rapide se fait en mode rectiligne.
01	RASTER ZIGZAG	mode zigzag (OPT2=0, H=0). Comme RASTER à l'exception que le mode de déplacement est en zigzag. Le retour rapide se fait en mode rectiligne.
10	COIL	mode serpentin (OPT2=0, H=0). Déplacement lent dans la direction de départ jusqu'à la frontière définie par le champs condition, changement de ligne et retour lent sur la ligne suivante jusqu'à la frontière opposée. Changement de ligne à nouveau.
11	COIL ZIGZAG	mode serpentin-zigzag (OPT2=0, H=0). Comme RASTER à l'exception que le mode de déplacement est en zigzag à l'allé et au retour.
10	TWICE	mode balayage dédoublé (OPT2=1, H=1). Déplacement lent dans la direction de départ jusqu'à la frontière définie par le champs condition, retour lent sur la même ligne jusqu'à la frontière opposée puis, changement de ligne.

[#number_LINE] Nombre entier de 16 bits. Donne le nombre de lignes à balayer. Initialise le compteur N. Si ce paramètre n'est pas présent la valeur précédente sera utilisée (i.e. le compteur N ne sera pas modifié).

[BACKWARD]

B	BACKWARD	Commentaire
0	BACKWARD	balayage vers le haut.
(1)		balayage vers le bas.

[BORDER#max] nombre entier de 8 bits. Donne le maximum de pixel consécutif rencontré qui activent la condition (sur la frontière définie par le code de condition). Lorsque ce maximum est atteint, on termine le balayage en replaçant le pixel d'intérêt à l'intérieur de la zone de balayage. Initialise le compteur E. Si ce paramètre n'est pas présent l'ancienne valeur sera prise (i.e. le compteur E ne sera pas modifié).

REMARQUES:

- Exécute les boucles C.
- Les déplacement de plus d'un pixel peuvent être arrêtés par le signal Stop externe.
- Lorsqu'on utilise un des modes RASTER le pixel d'intérêt dépasse la frontière de 1 position après le retour rapide
- En mode zigzag, l'option 0 est toujours à zéro et le départ se fait toujours vers la droite (direction de départ +1)
- L'instruction ne ce termine jamais sur une frontière à moins que cette frontière soit de forme quelconque. On ne peut prédire alors le comportement des différentes fonctions de balayage.
- Le balayage peut se faire dans n'importe laquelle des 6 directions possibles définissant un balayage horizontal ou en diagonal.

C.5.13 SET_CAPTEUR

[DIR#d] [condition];

MSB 15							8	LSB 7							0
INST				S	DIR			COND			IN			OP	
0	0	0	1	0	D	D	D	C	C	C	0	0	0	0	1

Met le signal R_{eset} actif pour un cycle d'horloge. est utilisé pour initialiser les différents registres à décalage du capteur MAR. Effectue également

Champs	Commentaire
[DIR#d]	voir section C.4.1.
[condition]	voir section C.4.2.

REMARQUES:

- N'exécute pas les boucles C.
- Utilisé avec la direction "DIR0" pour remettre à zéro le processus d'intégration de l'illuminance sur le capteur MAR.

ANNEXE D

Test Fonctionnel du Contrôleur MAR

Réalisé par Stéphane Dallaire

Été 1991

D.1 Introduction

Ce rapport présente un projet d'été réalisé à l'été 91 pour Marc Tremblay étudiant au doctorat au laboratoire de vision et systèmes numériques. Le projet consistait à vérifier le fonctionnement de circuits VLSI conçus par Marc Tremblay. Ceux-ci ont été conçus pour être intégrés au système M.A.R. (Multiport Array photoReceptor). La tâche plus spécifique était dans un premier temps de mettre au point un banc d'essai pour les tests et ensuite de l'utiliser pour la vérification des circuits.

La vérification proprement dite des circuits a consisté d'abord à vérifier leur fonctionnement général puis, s'ils fonctionnaient comme prévu, à établir leurs performances en évaluant certains paramètres comme la bande passante. Le but de cette vérification est de détecter les défauts de fabrication entraînant un mauvais fonctionnement du circuit ainsi que les erreurs de conception qui auraient pu survenir.

D.2 Description du circuit: Le contrôleur de caméra MAR

Le LVMRC est un circuit intégré qui a été réalisé en technologie cmos4s. Sa description des plots de contact est présentée à la Figure D.1.

1 -> MD<7>
2 -> D<6>
3 -> MD<6>
4 -> D<5>
5 -> MD<5>
6 -> D<4>
7 -> MD<4>
8 -> D<3>
9 -> MD<3>
10 -> D<2>
11 -> MD<2>
12 -> D<1>
13 -> MD<1>
14 -> D<0>
15 -> MD<0>

16 -> GND
17 -> Vdd ring
18 -> Analog<0>
19 -> Analog<1>
20 -> Analog<2>
21 -> Analog<3>
22 -> Analog<4>
23 -> Analog<5>
24 -> GND ring
25 -> R/W_
26 -> DS0_
27 -> DS1_
28 -> A<5>
29 -> A<4>
30 -> A<3>

31 -> A<2>
32 -> A<1>
33 -> CS_
34 -> MD<15>
35 -> D<15>
36 -> MD<14>
37 -> D<14>
38 -> MD<13>
39 -> D<13>
40 -> MD<12>
41 -> D<12>
42 -> MD<11>
43 -> D<11>
44 -> MD<10>
45 -> D<10>

46 -> MD<9>
47 -> D<9>
48 -> MD<8>
49 -> D<8>
50 -> Vdd
51 -> Vdd ring
52 -> DIR<2>
53 -> DIR<1>
54 -> DIR<0>
55 -> set_capteur_
56 -> IO<7>
57 -> IO<6>
58 -> IO<5>
59 -> IO<4>
60 -> IO<3>
61 -> IO<2>
62 -> IO<1>
63 -> IO<0>
64 -> GND ring
65 -> CK
66 -> move
67 -> move_w_en
68 -> D<7>

vue de dessus

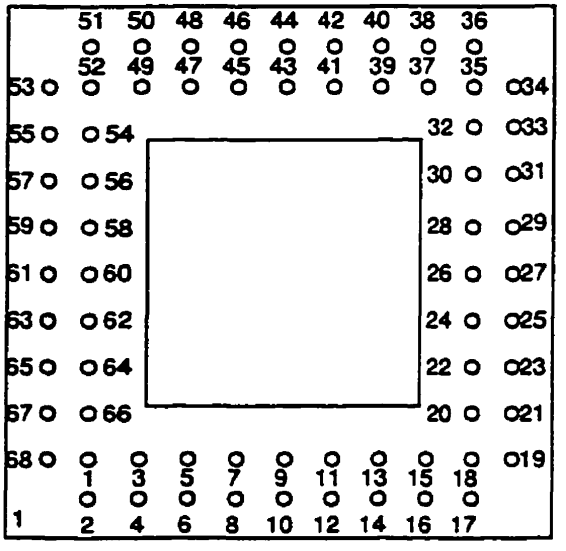


Figure D.1 Description des plots de contact du LVMRC

Le LVMRC est de loin le circuit le plus complexe qui a été vérifié au cours de ce projet. Il s'agit en fait du contrôleur du système M.A.R. au complet intégré sur une seule puce. Il comprend entre autre le PLA avec tout son microcode, les compteurs C, N et E, un

module contrôlant le pointeur de position, la position ainsi qu'un détecteur de dépassements un peu comme sur le LVCT4, un module traitant et détectant les discontinuités d'arêtes, un module contrôlant l'"arbitration" des instructions, etc.

Le circuit est aussi doté de 3 bus bidirectionnels, le bus "D<15:0>", le bus "MD<15:0>", et le bus "IO<7:0>". Le bus "D<15:0>" lorsqu'il est utilisé comme entrée permet de programmer les 31 registres de configuration. Lorsque le bus "D<15:0>" est configuré en sortie, il permet à l'utilisateur de lire des informations sur l'état des différentes parties du circuit, celles-ci étant "stockées" dans les 31 registres de lecture. Les détails concernant ces 31 registres en écriture et en lecture sont indiqués au Tableau D.1.

Tableau D.1 Plage d'adressage des Registres du LVMRC

REGISTRE	ÉCRITURE	LECTURE
<0>	Instruction	pos_index_instantané
<1>	Compteur N	Compteur N
<2>	Compteurs C & E	Compteurs C & E
<3>	Compteurs A & B	Compteurs A & B
<4>	Stop	pos1<15:0>
<5>	Resume	pos2<15:0>
<6>	Control0	pos3<15:0>
<7>	Control1	pos4<15:0>
<8>	MD_conf [1,0]	pos_x<11:0>
<9>	MD_conf [3,2]	pos_x_-1
<10>	MD_conf [5,4]	pos_x_-2
<11>	MD_conf [7,6]	pos_x_-3
<12>	MD_conf [9,8]	pos_y<11:0>
<13>	MD_conf [11,10]	pos_y_-1
<14>	MD_conf [13,12]	pos_y_-2
<15>	MD_conf [15,14]	pos_y_-3
<16>	IO_conf [1,0]	État0
<17>	IO_conf [3,2]	État0_-1
<18>	IO_conf [4]	État0_-2
<19>	int<5>	État0_-3
<20>	int<6>	État1
<21>	int<7>	État1_-1

Tableau D.1 Plage d'adressage des Registres du LVMRC

<22>	Erod [0]	État1_-2
<23>	Erod [1]	État1_-3
<24>	Erod [2]	pos_x_instantanée
<25>	Erod [3]	pos_y_intantanée
<26>	Erod [4]	État0_instantané
<27>	Erod [5]	État1_instantané
<28>	x_y_init	Last discontinuité_pos
<29>	x_y_MIN	Last discontinuité_pos_-1
<30>	x_y_MAX	
<31>	Control2	

Pour lire ou écrire de l'information sur le bus "D<15:0>", il faut dans un premier temps que l'entrée "CS_" (chip select_) soit active basse. Par la suite, il faut déterminer à l'aide de l'entrée "R/W_" s'il s'agit d'une opération de lecture ou d'écriture. La sélection du registre auquel on désire accéder s'effectue au moyen des 5 bits d'adresse "A<5:1>". En fait, le numéro du registre inscrit au tableau 3.1 correspond à son adresse. Finalement, pour accéder au "byte" le plus significatif (MSB), il suffit que l'entrée "DS1_" soit active basse et pour accéder au "byte" le moins significatif, il faut que l'entrée "DS0_" soit active basse.

Comme on peut le remarquer sur le Tableau D.1 chacun des bits du bus "MD<15:0>" et du bus "IO<7:0>" est complètement configurable. La configuration des 15 bits du bus "MD<15:0>" s'effectue en programmant adéquatement les registres "MD_conf" et la configuration des 5 bits de "IO<4:0>" s'effectue en programmant les registres "IO_conf". Pour ce qui est des bits "IO<7:5>", leur configuration s'effectue en programmant respectivement les registres "int<7>", "int<6>" et "int<5>".

Les bits du bus "MD<15:0>" ainsi que ceux de "IO<4:0>" sont configurables de la même façon. Leur configuration s'effectue en programmant adéquatement les 8 bits de configuration associés à chacun d'eux. La Figure D.2 indique la signification des 8 bits de configuration des registres "MD_conf" et "IO_conf".

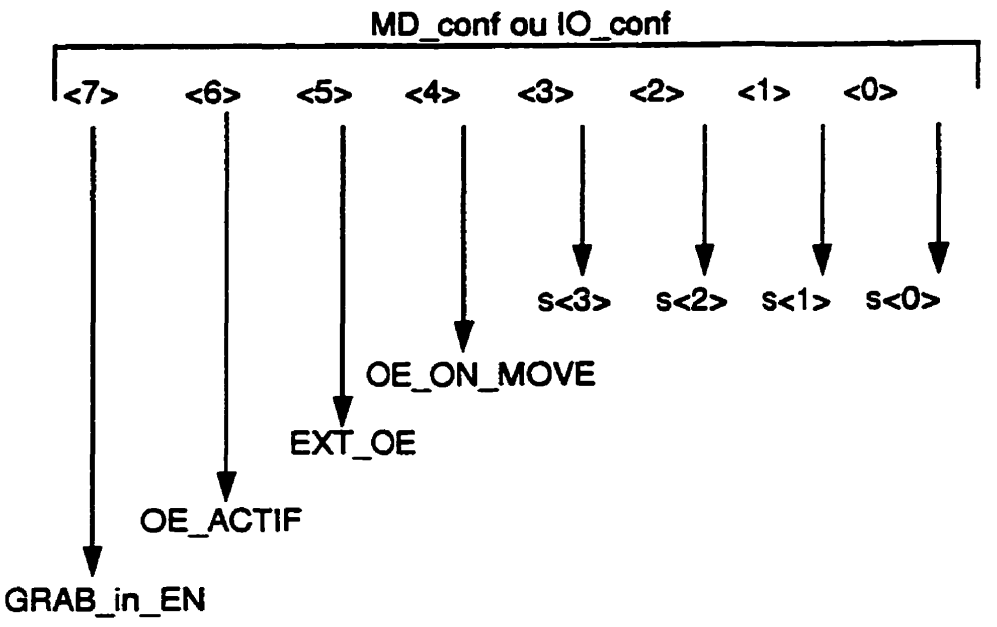


Figure D.2 Registres de configuration "MD_conf" et "IO_conf"

Les bits "s<3:0>" permettent de sélectionner, parmi un choix de 12, le signal de sortie que l'on désire obtenir sur ce bit. Les 3 bits "OE_ON_MOVE", "EXT_OE" et "OE_ACTIF" sont utilisés pour déterminer si le signal de sortie est présent seulement lorsque le signal "MOVE" est actif, si celui-ci est présent seulement lorsque le signal "L_IO_in<4>" est actif ou s'il est toujours présent sur ce bit. Finalement, le bit "GRAB_in_EN" est utilisé pour le signal "L_MD_in<>" ou pour le signal "L_IO_in<>" dépendant s'il s'agit d'un bit du bus "MD" ou du bus "IO". Ces signaux ("L_MD_in<>" et "L_IO_in<>") sont en fait les sorties de bascules D dont les entrées sont respectivement le signal "MD< >" et le signal "IO< >". Lorsque "GRAB_in_EN" = 0, l'entrée "ENABLE" de la bascule D est toujours égale à 1. Cependant lorsque "GRAB_in_EN" = 1, l'entrée "ENABLE" de la bascule est égale à 1 seulement lorsque le bit "MD< >"ou le bit "IO< >" est utilisé comme entrée.

La configuration des bits "IO<5>", "IO<6>" et "IO<7>" est toutefois quelque peu différente que celle décrite précédemment. Celle-ci s'effectue en programmant adéquatement les 16 bits de configuration associés à chacun de ces bits. La Figure D.3 indique la signification de chacun des bits de configuration.

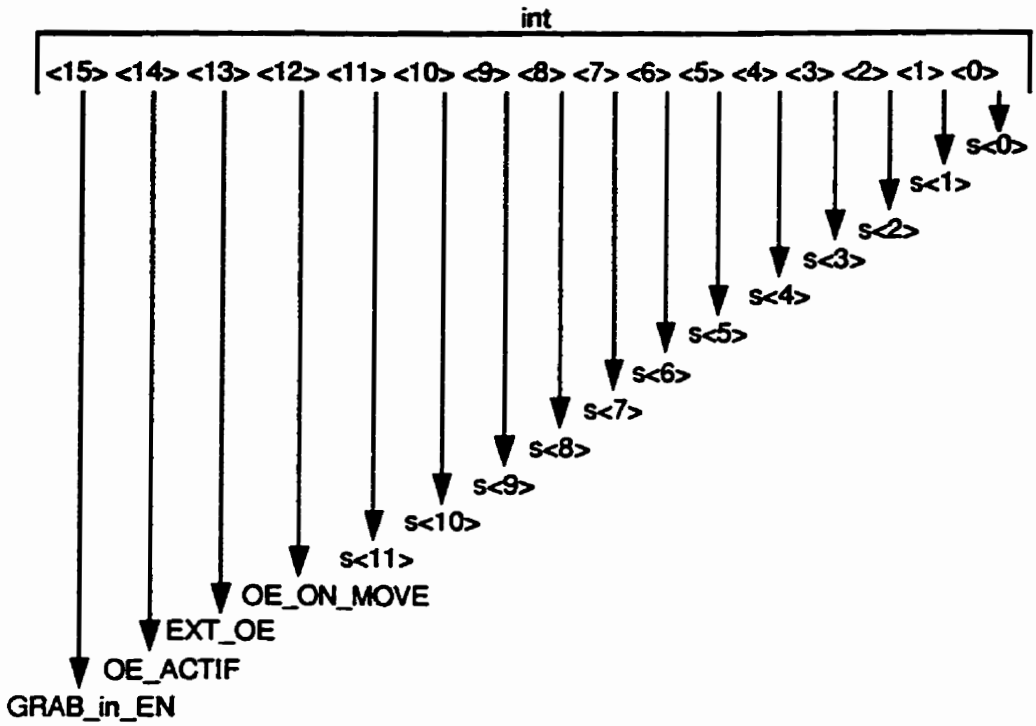


Figure D.3 Registres de configuration "int"

La différence entre ces 2 modes de configuration c'est-à-dire entre celui de laFigure D.2 et celui de la Figure D.3 est la façon de sélectionner le signal de sortie que l'on désire associer à ce bit. Dans ce cas-ci, pour sélectionner un des 12 signaux de sortie, il suffit d'activer le bit de sélection "s" (parmi les 12 de "s<11:0>") correspondant à son numéro. Une caractéristique intéressante est qu'il est possible d'en sélectionner plus d'un à la fois. Il est donc possible de former des "ou câblé" entre différents signaux du contrôleur.

La configuration du bus "MD<15:0>" comme entrée est prévue pour être utilisée à l'intérieur du module "ÉROSION" ainsi que pour le module "VISITE". Le bus "MD<15:0>" permet aussi, lorsqu'il est configuré en lecture, d'accéder à certains signaux internes au contrôleur. On peut observé auTableau D.2 les 12 signaux de sorties configurables sur chacun des bits de ce bus.

Tableau D.2 Signaux de sorties configurables sur le bus "MD<15:0>"

s<3:0>=:	11	10	9	8	7	6	5	4	3	2	1	0
MD<15>	1	12	13	4	7	8	16	29	26	23	57	20
MD<14>	0	11	12	4	6	9	15	28	25	22	56	19
MD<13>	1	12	13	4	5	10	14	27	24	21	55	18
MD<12>	0	11	13	4	7	8	16	29	26	23	54	17
MD<11>	1	12	13	4	6	9	15	28	25	22	53	41
MD<10>	0	11	13	4	5	10	14	27	24	21	52	35
MD<9>	1	12	13	4	7	8	16	29	26	23	51	40
MD<8>	0	11	13	4	6	9	15	28	25	22	50	34
MD<7>	1	12	13	4	5	10	14	27	24	21	49	39
MD<6>	0	11	13	4	7	10	16	29	26	23	48	33
MD<5>	1	12	13	4	6	8	15	28	25	22	47	38
MD<4>	0	11	13	4	5	9	14	27	24	21	46	32
MD<3>	1	12	13	4	7	9	16	29	26	20	45	37
MD<2>	0	11	13	4	6	10	15	28	25	19	44	31
MD<1>	1	12	13	4	5	6	14	27	24	18	43	36
MD<0>	0	11	13	4	3	2	58	12	1	17	42	30

Le Tableau D.3 donne le nom de chaque variable qui est associée au code numérique du Tableau D.2.

Tableau D.3 Variables d'états pouvant être programmées pour chaque bit du bus MD<15:0>

code	variable d'état	code	variable d'état
58	VISITE_INT	29	G_DIR<2>
57	L_MD_in<15>	28	G_DIR<1>
56	L_MD_in<14>	27	G_DIR<0>
55	L_MD_in<13>	26	DIR_PREC_I<2>
54	L_MD_in<12>	25	DIR_PREC_I<1>
53	L_MD_in<11>	24	DIR_PREC_I<0>
52	L_MD_in<10>	23	DIR_PREC_I_1<2>
51	L_MD_in<9>	22	DIR_PREC_I_1<1>

Tableau D.3 Variables d'états pouvant être programmées pour chaque bit du bus MD<15:0>

code	variable d'état	code	variable d'état
50	L_MD_in<8>	21	DIR_PREC_I_1<0>
49	L_MD_in<7>	20	VISITE<3>
48	L_MD_in<6>	19	VISITE<2>
47	L_MD_in<5>	18	VISITE<1>
46	L_MD_in<4>	17	VISITE<0>
45	L_MD_in<3>	16	OA<2>
44	L_MD_in<2>	15	OA<1>
43	L_MD_in<1>	14	OA<0>
42	L_MD_in<0>	13	DISC
41	H_out<5>	12	LONG_DISC
40	H_out<4>	11	SENS_D
39	H_out<3>	10	E_ZERO
38	H_out<2>	9	RESET_E
37	H_out<1>	8	E_EN
36	H_out<0>	7	RESET_N
35	S_out<5>	6	N_EN
34	S_out<4>	5	N_ZERO
33	S_out<3>	4	P2
32	S_out<2>	3	CONDITION
31	S_out<1>	2	OVERFLOW
30	S_out<0>	1	H0
		0	S0

Pour ce qui est des 5 bits les moins significatifs du bus "IO<7:0>", leur configuration comme entrées permet à certains signaux internes du contrôleur d'être remplacés par des signaux externes. Cela à été prévu de manière à ce que si un signal interne ne fonctionne pas correctement ou pas de tout, il puisse être remplacé par un signal externe. Cela n'a cependant pas été prévu pour tout les signaux mais seulement pour ceux qui semblaient les plus importants. On peut observer au Tableau D.4 les 12 signaux de sorties configurables sur chacun des bits de "IO<4:0>".

Tableau D.4 Signaux de sorties configurables sur "IO<4:0>"

s<3:0>=:	11	10	9	8	7	6	5	4	3	2	1	0
IO<4>	8	7	1	0	30	41	43	49	52	26	15	6
IO<3>	27	21	20	19	18	17	16	10	9	25	14	5
IO<2>	36	35	34	33	32	31	30	29	28	24	13	4
IO<1>	45	44	43	42	41	40	39	38	37	23	12	3
IO<0>	54	53	52	51	50	49	48	47	46	22	11	2

Le Tableau D.5 donne le nom de chaque variable qui est associée au code numérique du Tableau D.4.

Tableau D.5 Variables d'états pouvant être programmées pour chaque bit du bus IO<4:0>

code	variable d'état	code	variable d'état
54	C_ZERO	27	SD_VALIDE
53	RESET_E	26	CYCLE_A
52	E_EN	25	VISITE<3>
51	E_ZERO	24	VISITE<2>
50	RESET_N	23	VISITE<1>
49	N_EN	22	VISITE<0>
48	N_ZERO	21	MATCH_ETAT
47	EN_X	20	P2_
46	UP_X	19	CONDITION
45	UP_Y	18	BREAK
44	EN_Y	17	LONG_DISC
43	MIN_Y	16	DISC
42	MAX_Y	15	ETAT<4>
41	MIN_X	14	ETAT<3>
40	MAX_X	13	ETAT<2>
39	Y_EQ_Y0	12	ETAT<1>
38	X_EQ_X0	11	ETAT<0>
37	MATCH_1F_	10	RESET_POS
36	phi2_mod0_	9	S_MOVE
35	phi1_mod0_	8	VAR
34	LOCAL_DISC_CK	7	SENS_D

Tableau D.5 Variables d'états pouvant être programmées pour chaque bit du bus IO<4:0>

code	variable d'état	code	variable d'état
33	MV_CAPT_CK	6	F_NOP
32	EN_A_	5	READY
31	RESET_A_	4	OA<2>
30	MATCH_B	3	OA<1>
29	MATCH_A	2	OA<0>
28	MAX_A_CK	1	MAR_SYSTEM_RUN_
		0	PARITE_XY

Une autre caractéristique intéressante est que le 3 bits les plus significatifs du bus "IO<7:0>" c'est-à-dire les bits "IO<7>", "IO<6>" et "IO<5>" peuvent être utilisés comme des signaux d'interruption. Il suffit de configurer les registres de contrôle en conséquence. Lorsque ces bits sont configurés comme sorties, cela permet à certains signaux internes au contrôleur d'être assignés comme des signaux d'interruption. Aussi, en configurant ces bits comme entrées, les interruptions peuvent être générées à partir de signaux externes. On peut observer au Tableau D.6 les 12 signaux de sorties configurables sur "IO<7>", "IO<6>" et "IO<5>".

Tableau D.6 Signaux de sorties configurables sur "IO<7:5>"

s<3:0>=:	11	10	9	8	7	6	5	4	3	2	1	0
IO<7>	31	30	29	28	27	26	25	24	23	22	21	20
IO<6>	22	21	19	18	17	16	15	14	13	12	11	10
IO<5>	11	10	9	8	7	6	5	4	3	2	1	0

Le Tableau D.7 donne le nom de chaque variable qui est associée au code numérique du Tableau D.6.

Tableau D.7 Variables d'états pouvant être programmées pour chaque bit du bus IO<7:5>

code	variable d'état	code	variable d'état
31	RESET_E	15	CONDITION
30	E_EN	14	MATCH_ETAT
29	E_ZERO	13	MATCH_OA<2>
28	RESET_N	12	MATCH_OA<1>
27	N_EN	11	MATCH_OA<0>
26	N_ZERO	10	OVERFLOW
25	P2_	9	OVERFLOW_Y
24	P2	8	OVERFLOW_X
23	SD_VALIDE	7	MAX_Y
22	DISC	6	MIN_Y
21	LONG_DISC	5	MAX_X
20	VAR	4	MIN_X
19	H0	3	Y_EQ_Y0
18	F_NOP	2	X_EQ_X0
17	READY	1	X0_et_Y0
16	MAR_SYSTEM_RUN_	0	VISITE_INT

Pour obtenir plus de détails sur le contrôleur LVMRC, se référer au design du circuit sur le logiciel "CADENCE". Le fichier de conception du LVMRC s'intitule "MAR_controller". Ce fichier est situé dans le sous répertoire "control" du répertoire "cmos4s".

D.3 Procédure de test: du contrôleur MAR

La vérification du LVMRC s'est effectuée en plusieurs étapes dont chacune consistait à vérifier différentes parties du circuit. Ces étapes sont décrites brièvement ci-dessous. Mais avant de procéder à la vérification proprement dite, il a d'abord fallu programmer les registres de configuration du contrôleur. La programmation de ces registres ainsi que la plupart des tests ont été effectués à l'aide du générateur de données HP 8180A. Les mesures ont été effectuées dans la plupart des cas à l'aide de l'analysera logique HP 16500A.

D.3.1 Vérification des horloges phi1 et phi2

Les horloges "phi1" et "phi2" sont des signaux qui sont générés à partir du signal d'horloge "CK" qui est un signal provenant de l'extérieur du circuit. La vérification a consisté à vérifier si les 2 horloges sont belles et biens inverses (l'une étant l'inverse de l'autre) et s'il n'y a pas de recouvrement entre celles-ci. Ceci est essentielle au bon fonctionnement du circuit.

Ces 2 signaux n'étant pas directement accessibles de l'extérieur du circuit, il a fallu assigner à tour de rôle le signal "phi1" ("phi1_mod_0") et "phi2" ("phi2_mod_0") au bit "IO<5>" et les comparer avec le signal d'horloge "CK". Les résultats sont indiqués à laFigure D.4 à une fréquence de 1 MHz pour "phi1_path_select" = 0 et "phi2_path_select" = 0 (signaux de contrôle). Les différents délais qui ont été mesurés y ont aussi été inclus.

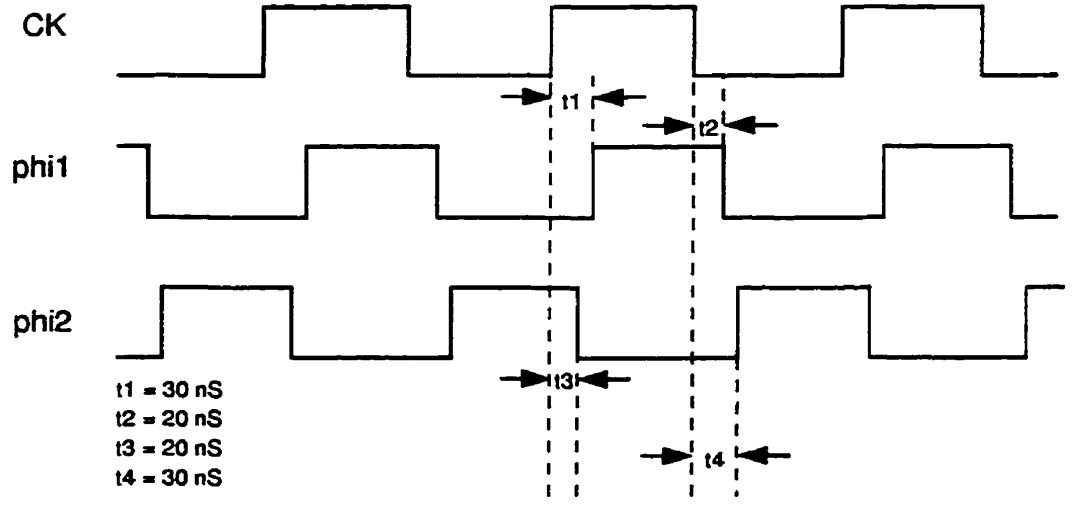


Figure D.4 Vérification des signaux "phi1" et "phi2"

On remarque sur la Figure D.4 que le 2 signaux sont belles et biens inverses et qu'il n'y a pas de recouvrement entre ceux-ci. Bien que les délais t_1, t_2, t_3 et t_4 soient un peu plus courts que prévus, ils sont tout-de-même acceptables pour le bon fonctionnement du circuit.

Les mêmes tests ont été répétés à la même fréquence mais avec "phi1_path_select" = 1 et "phi2_path_select" = 1. Aucune différence appréciable n'a cependant été remarquée. Les délais étaient à peu près les mêmes que précédemment.

D.3.2 Vérification du module d'arbitration des instructions

Parmi les signaux d'arbitration des instructions on compte le signal "READY", le signal "F_NOP" et le signal "CONDITION". Ceux-ci gèrent l'exécution des instructions. Ces signaux n'étant pas directement accessibles à l'extérieur du circuit, il a d'abord fallu configurer différents bits du bus "MD<15:0>" et du bus "IO<7:0>" de manière à les rendre accessibles. Par la suite, l'allure de ces signaux a été vérifiée lors de l'exécution d'une instruction simple telle une "instruction interruption", un "reset position" ou un "set capteur" (instructions du groupe instruction = 1). Les résultats sont indiqués à la Figure D.5

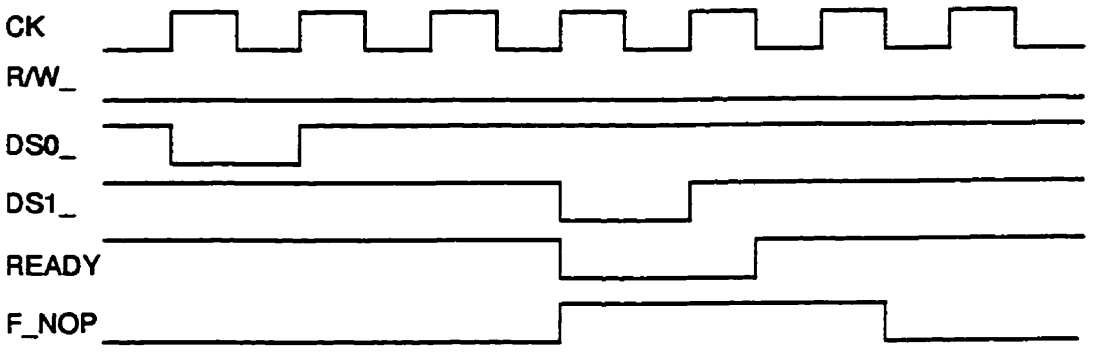


Figure D.5 Arbitration des instructions

Un examen approfondi de la Figure D.5 montre que le signal "F_NOP" n'est pas adéquat. Ce mauvais fonctionnement résulte d'une erreur de conception qui fait aussi en sorte que le signal "CONDITION" ne fonctionne pas correctement lui non plus car le signal "CONDITION" est généré à partir du signal "F_NOP". Aussi, à cause de ce mauvais fonctionnement, l'exécution d'une instruction telle un "MOVE" ne se termine jamais, celle-ci recommence toujours.

Quelques solutions ont été envisagées pour corriger le signal "CONDITION" en rendant celui-ci externe, mais pour le signal "F_NOP", il n'y avait rien à faire. Cette erreur

de conception a donc de très lourdes conséquences. La vérification a tout-de-même été poursuivie dans le but de vérifier s'il n'y avait pas d'autres erreurs de conception.

D.3.3 Vérification des compteurs C, E et N

La première étape de la vérification des compteurs C, E et N a été de leur assigner d'abord une certaine valeur en écrivant celle-ci dans les registres de configuration #1 et #2. Une lecture de ces mêmes registres a par la suite permis de vérifier si les valeurs ont été stockées correctement.

Différentes instructions telles des "MOVE", des "MOVE_N et des "MOVE_T" ont ensuite été exécutées de façon à vérifier si les 3 compteurs comptaient correctement.

Ces 2 vérifications ont permis de conclure que les compteurs C,E et N fonctionnaient correctement. Seul le signal "C_ZÉRO" n'est pas adéquat. Cela est toutefois évident puisque ce signal est généré à partir du signal "F_NOP".

D.3.4 Vérification des compteurs x et y

Comme dans le cas des compteurs C, E et N, la première étape a été d'assigner une certaine valeur aux compteurs puis d'aller la lire par la suite. Il a donc fallu écrire une certaine valeur de "x_y_init" dans le registre de configuration #28 et aller ensuite, sans exécuter aucune instruction, lire la valeur de "pos_index_instantané" dans le registre #0.

L'étape suivante a été d'écrire des valeurs de "x_y_MIN" et "x_y_MAX" dans les registres de configuration #29 et #30. Puis, en assignant différentes valeurs de "x_y_init", le pixel à l'étude a successivement été placé sur chacune de ces frontières (une à la fois). L'observation du signal "OVERFLOW" a permis de vérifier si les valeurs de "x_y_MIN" et "x_y_MAX" avaient été stockées correctement et si la détection des dépassements s'effectuait de façon adéquate.

La dernière étape a été d'exécuter des instructions "MOVE" dans différentes directions et de vérifier si l'indice de position instantanée "pos_index_instantané" variait en conséquence.

Les 2 premières parties de la vérification ont été très concluantes c'est-à-dire que le circuit a réagi comme prévu. Cependant, lors de la dernière partie, il a été impossible de faire varier l'indice de position instantané en effectuant des instructions "MOVE". L'examen des différents signaux de contrôle des compteurs a permis de conclure que le

signal "MV_CAPT_CK" ne fonctionnait pas. Il s'agit de l'horloge des compteurs de position mais ce signal est aussi utilisé à plusieurs autres endroits dans le circuit.

D.3.5 Vérification du microcode

Le microcode du contrôleur a été quelque peu modifié depuis sa première version. Un nouveau mode de déplacement a été implanté. Il s'agit du déplacement en mode hélicoïdal. Afin de sélectionner ce mode de déplacement, le PLA a été muni d'une entrée supplémentaire soit l'entrée "opt_2". Lorsque "opt_2" = 0, les déplacements s'effectuent de la même manière qu'avec la première version du microcode. Cependant, lorsque "opt_2" = 1, les déplacements s'effectuent hélicoïdalement.

Le vérification du microcode n'a pas consisté à vérifier le microcode en entier car sa première version lui ressemblant beaucoup a déjà été vérifiée très en détails avant d'être implantée au système MAR. La vérification a consisté plutôt à vérifier les instructions en mode hélicoïdal, c'est-à-dire lorsque "opt_2" = 1.

La procédure adoptée pour cette vérification est d'abord de vérifier sommairement, pour chaque instruction, son exécution en mode normal (lorsque "opt_2" = 0) puis de vérifier un peu plus en détail son exécution en mode hélicoïdal (lorsque "opt_2" = 1). Cependant, les instructions du type "Scan_Area" n'ont pu être vérifiées étant donné que les indices de position x et y ne varient pas lors de déplacements du pixel à l'étude.

Parmi les instructions qui ont été vérifiées, une seule ne s'exécutait pas correctement. Il s'agit de l'instruction "MOVE" (instruction = 4) avec "opt_2" = 1. Au lieu de passer de l'état 00 à l'état 18, le PLA passait plutôt de l'état 00 à l'état 1F.

D.3.6 Vérification du module VISITE

Le module VISITE est conçu pour interfacer le LVMRC avec une mémoire externe. Il devrait permettre de conserver dans cette mémoire le nombre de fois que chaque pixel de la fenêtre d'étude a été visité. Pour ce faire, la mémoire externe devrait comporter un espace de 4 bits pour chaque pixel de la fenêtre d'étude. La fonction du module VISITE est en fait d'incrémenter de 1 la valeur contenue dans un espace mémoire à chaque fois que le pixel correspondant à cet espace est visité.

Cependant, pour la vérification, aucun bloc mémoire n'a été utilisé mais plutôt un 74LS373 étant donné que son fonctionnement est à peu près semblable à celui d'une mémoire. On peut observer à la Figure D.6 le montage utilisé pour la vérification.

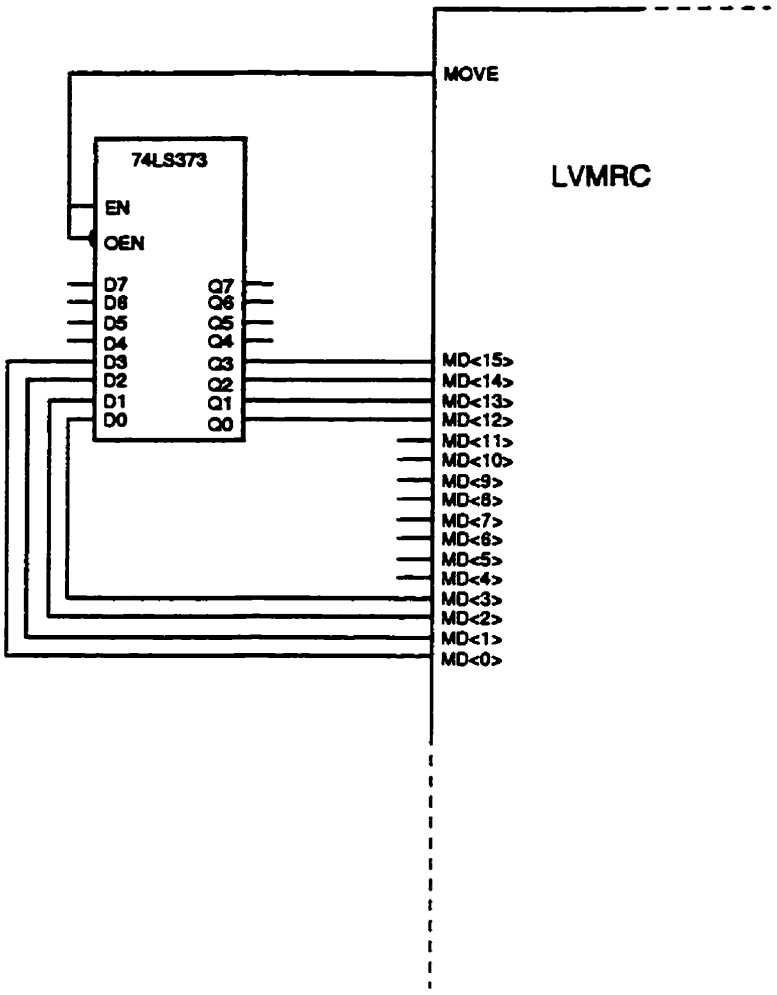


***Figure D.6** Montage utilisé pour vérifier le module VISITE*

Les 4 bits "MD<3:0>" ont été configurés de manière à fournir le signal "VISITE<3:0>" comme sorties. Les 4 bits "MD<15:12>" ont été configurés comme entrées lorsque "MOVE" = 0 et "flottants" lorsque "MOVE" =1 c'est-à-dire configurés comme sorties mais sans aucun signal assigné. De plus, ces 4 bits ont été configurés de manière à ce que les signaux "L_MD_in<15:12>" soient mis à jour seulement lorsque "MOVE" = 0 c'est-à-dire lorsque les 4 bits "MD<15:12>" sont des entrées.

La procédure utilisée pour la vérification a été d'exécuter des instructions "MOVE" et de vérifier à chaque fois si la valeur de "VISITE<3:0>" s'incrémentait de 1 jusqu'à ce

qu'elle atteigne la valeur finale de F. Ces tests ont permis de conclure que le module "VISITE" fonctionnait bel et bien comme prévu.

D.3.7 Vérification du module ÉROSION

Le module d'érosion est un module servant à traiter les 6 entrées analogiques "Analog<5:0>". Il permet d'effectuer des érosions ou des dilatations d'images (arêtes). Cependant, étant donné que le signal "MV_CAPT_CK" ne fonctionnait pas, plusieurs parties de ce module ne fonctionnaient pas non plus.

En fait seuls les signaux "H_out<0>" et "S_out<0>" ont été vérifiés. Ceux-ci sont associés à l'entrée analogique "Analog<0>". Le signal "S_out<0>" correspond au signe de l'entrée analogique. Un zéro analogique "S" a été fixé aux environs de 2.5 V. Lorsque la tension à l'entrée "Analog<0>" est inférieure à cette valeur, "S_out<0>" = 0. Par contre, lorsque la tension à l'entrée est supérieure à 2.5 V, "S_out<0>" = 1. Une certaine marge autour du zéro analogique a aussi été définie. La marge inférieure est "H-" et la marge supérieure est "H+". Lorsque la tension à l'entrée est à l'intérieur de cette marge, "H_out<0>" = 0. Cependant, lorsque celle-ci est à l'extérieur, "H_out<0>" = 1. Cette situation est illustrée à la Figure D.7.

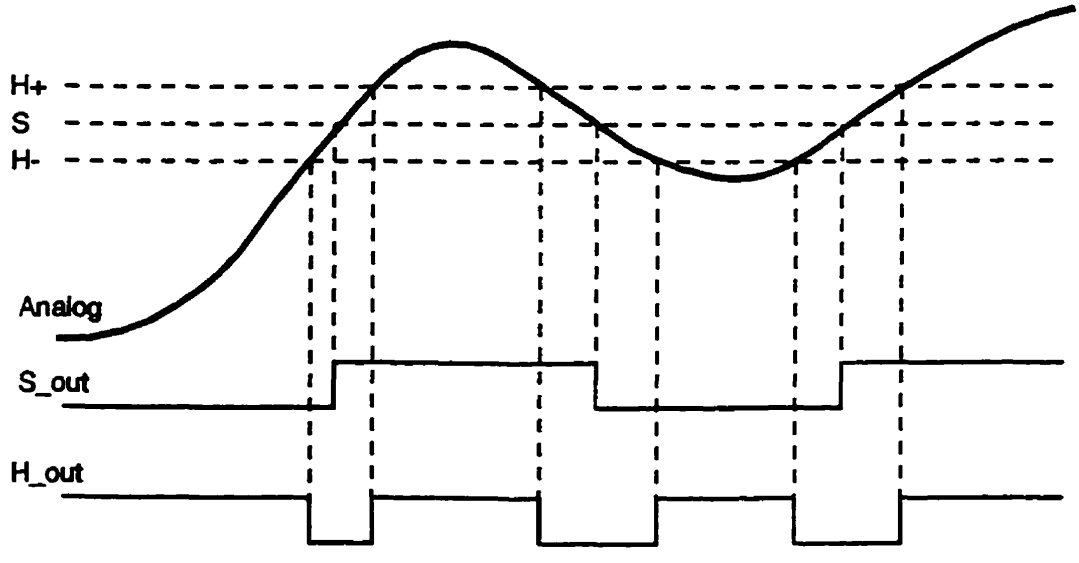


Figure D.7 Signification des signaux "S_out" et "H_out"

Ce que l'on désirait mesurer, c'était en fait les valeurs de "S", "H+" et "H-" pour vérifier si elles avaient à peu près les mêmes valeurs que prévue. La démarche utilisée pour mesurer ces valeurs a été de faire varier la tension à l'entrée "Analog<0>" entre 0 et 5 V et de noter les valeurs de tensions pour lesquelles les signaux "S_out<0>" et "H_out<0>" varient. La valeur de "H-" correspond à la tension pour laquelle le signal "H_out<0>" passe de 1 à 0 et que le signal "S_out<0>" demeure à 0. La valeur de "H+" correspond à la tension pour laquelle le signal "H_out<0>" passe de 0 à 1 et que le signal "S_out<0>" demeure à 1. Finalement, la valeur de "S" correspond à la tension pour laquelle le signal "S_out<0>" passe de 0 à 1 ou de 1 à 0 et que le signal "H_out<0>" demeure à 0. Ces valeurs ont aussi été mesurées pour l'entrée "Analog<0>" des 4 autres circuits. Les résultats sont illustrés au Tableau D.8.

Tableau D.8 Évaluation des valeurs de "S", "H-" et "H+"

# du spécimen	S (V)	H- (V)	H+ (V)
LVMRC #1	2.21	1.99	2.40
LVMRC #2	2.10	1.89	2.30
LVMRC #3	2.21	1.99	2.42
LVMRC #4	2.18	1.95	2.37
LVMRC #5	1.99	1.81	2.19

En examinant ce tableau on remarque que le valeur de "S" est bien située aux environs de 2.5 V (un peu plus bas pour être plus précis). On remarque aussi que la marge autour du zéro analogique est un peu étroite c'est-à-dire que les valeurs de "H-" et "H+" sont un peu trop près de la valeur de "S". Ceci aurait pour effet de rendre le système un peu plus sensible au bruit.Il serait souhaitable que dans la prochaine version du LVMRC les valeurs de "H-" et"H+" soient programmables c'est-à-dire qu'on puisse sélectionner différents "seuils" en programmant un multiplexer.

D.3.8 Évaluation de la fréquence maximale d'utilisation

À titre indicatif, la fréquence maximale d'utilisation a aussi été évaluée. La procédure utilisée a été la même que dans le cas du LVCT4. Dans un premier temps, il a fallu évaluer les délais de propagation des sorties afin d'en déterminer le plus grand. C'est celui-ci qui détermine la fréquence maximale d'utilisation.

Le délai maximum observé a été de 50 ns. Après quelques essais, il s'est avéré que la fréquence maximale d'utilisation était d'environ 17 MHz.

D.4 "Bugs" du LVMRC

Le Tableau D.9 dresse la liste des différents "bugs" du LVMRC accompagnés, à l'occasion, de suggestions de corrections à apporter pour la prochaine version du circuit. La liste qui suit présente les différentes modifications à apporter pour la prochaine version du LVMRC.

Tableau D.9 "Bugs" du LVMRC

"Bugs"	Suggestions de corrections
- Signal "F_NOP":	- Inverser le signal "READY" utilisé pour générer "F_NOP"
- Signal "MV_CAPT_CK":	- Corriger le délai de 3 ns (mettre un délai plus grand) - Utiliser "phi1" plutôt que "phi2_" pour générer le signal (c'est peut-être "EN_" qui est court) - Ajouter une entrée de secours afin de pouvoir rendre le signal externe si nécessaire
- Signal "MAX_A_CK_":	- Générer le signal à partir de "CYCLE_A" au lieu de "CYCLE_B"
- Signal "UP_Y":	- Mettre un inverseur au lieu d'un "buffer" entre "UP_Y_" et "UP_Y"
- Lecture de "POS_Y instantanée	- Corriger l'adresse du registre
- Instruction "MOVE" avec "opt_2" = 1	

Corrections suggérées:

- Ajouter au multiplexer MD la possibilité de ne pas modifier la mémoire utilisée pour le module "VISITE".
- Ajuster le seuil pour l'hystérésis (inverseurs avec des seuils plus grands et plus petits que 2.5 V) de façon à le rendre un peu plus grand. Prévoir peut-être la possibilité de rendre le seuil programmable en ajoutant un multiplexer.
- Remplacer l'horloge des compteurs de position x et y ("MV_CAPT_CK") par "phi2".

D.5 État du circuit LVMRC

La rédaction de ce document à été faite à la fin de l'été 1991 et une version corrigée a été soumise pour fabrication à la société canadienne de microélectronique à la fin de septembre 1991. La nouvelle version de ce circuit devrait être expédié vers la fin de 1991 et pourra être intégrée au système MAR si l'ensemble des corrections qui y ont été effectuées sont concluantes.

ANNEXE E

Dessin des Masques du Capteur et du Contrôleur MAR

On présente dans cette annexe le dessin des masques en noir et blanc des principaux circuits intégrés qui ont été fabriqués, soit le capteur et la contrôleur MAR. Le niveau de détail n'est pas élevé mais on peut voir l'allure générale des circuits.

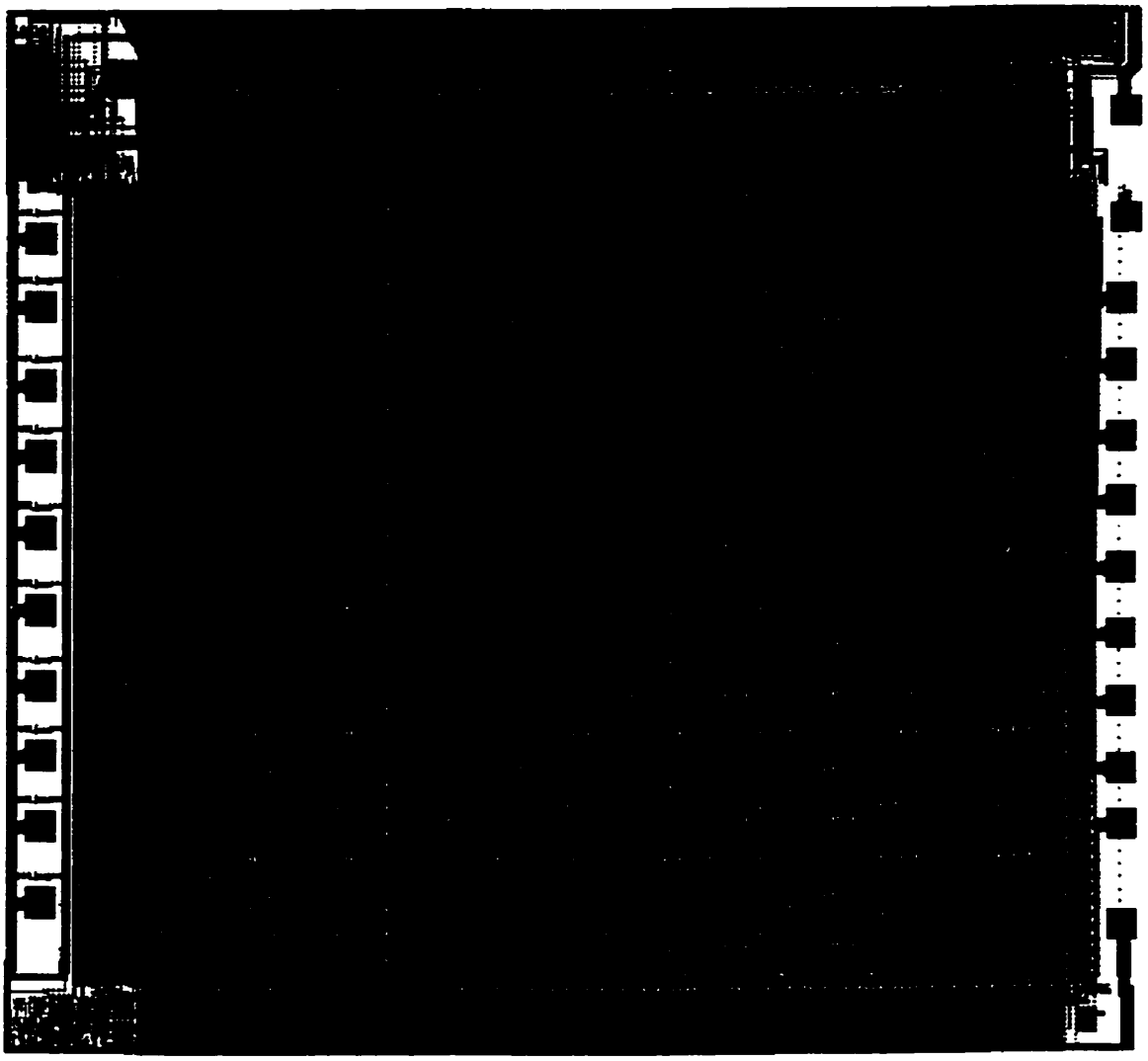


Figure E.1 Dessin des masques du capteur MAR de 128 x 128 pixels. On présente une version de résolution réduite pour mieux montrer les détails du design.

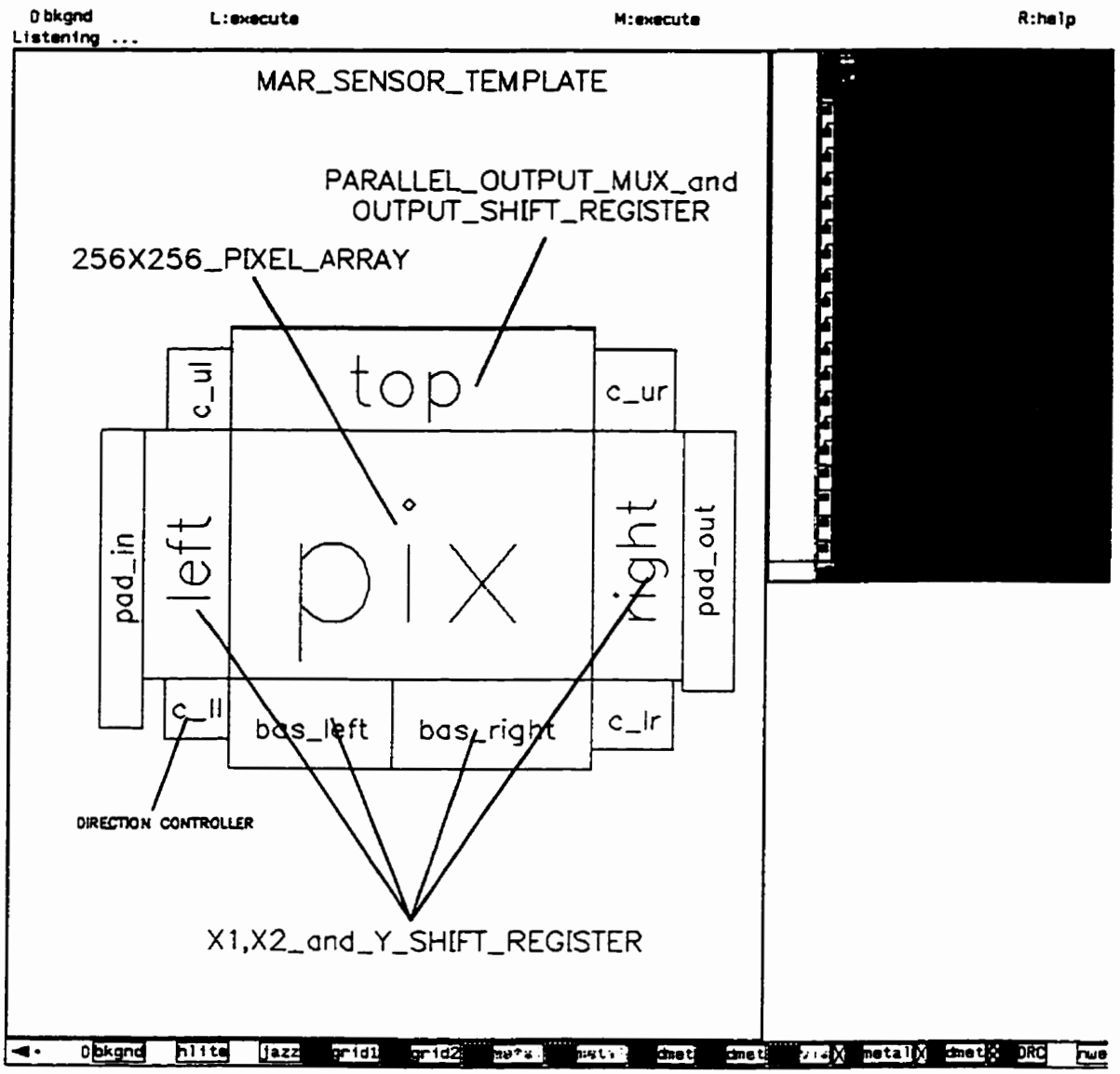


Figure E.2 *Représentation "template" qui est utilisée par le compilateur de structure pour créer le circuit de la Figure E.1.*

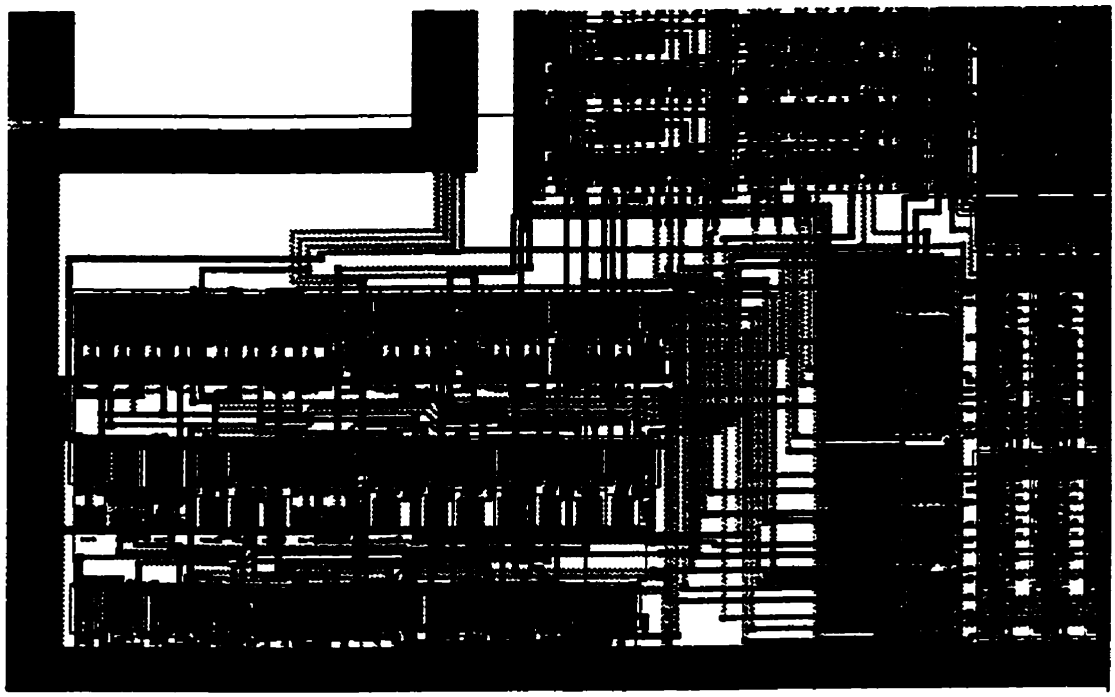


Figure E.3 Détail du coin inférieur gauche du capteur MAR

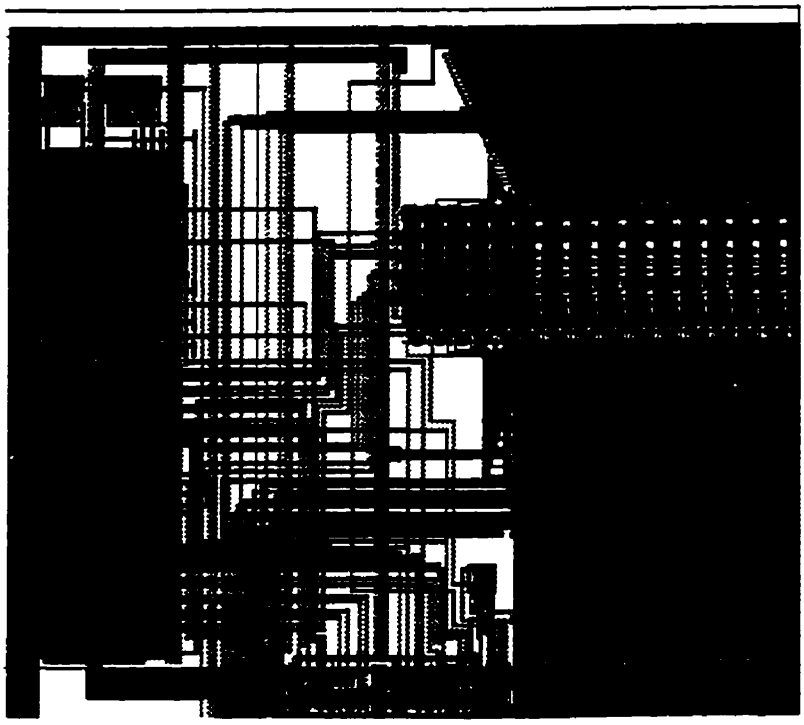


Figure E.4 Détail du coin supérieur gauche du capteur MAR

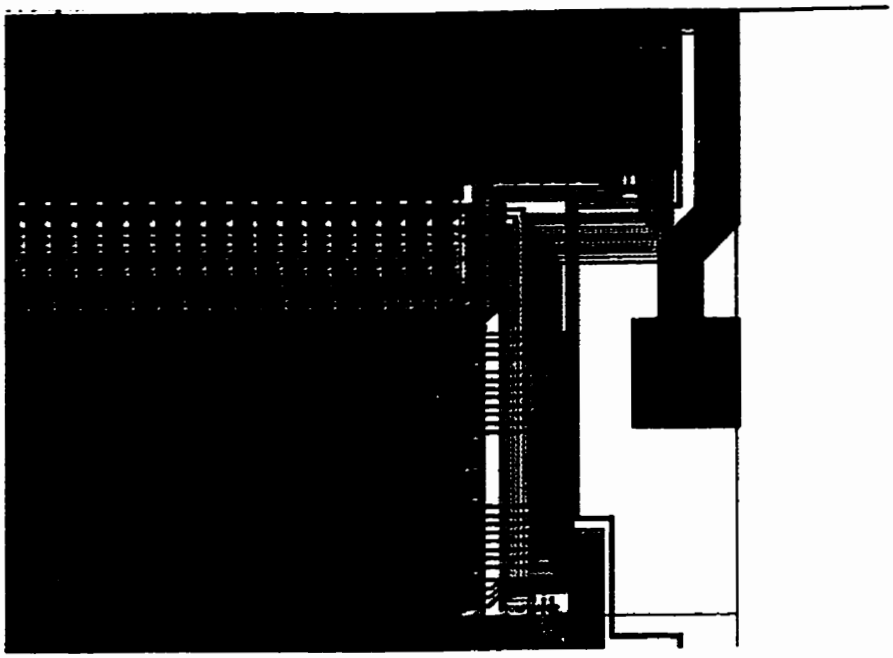


Figure E.5 Détail du coin supérieur droit du capteur MAR

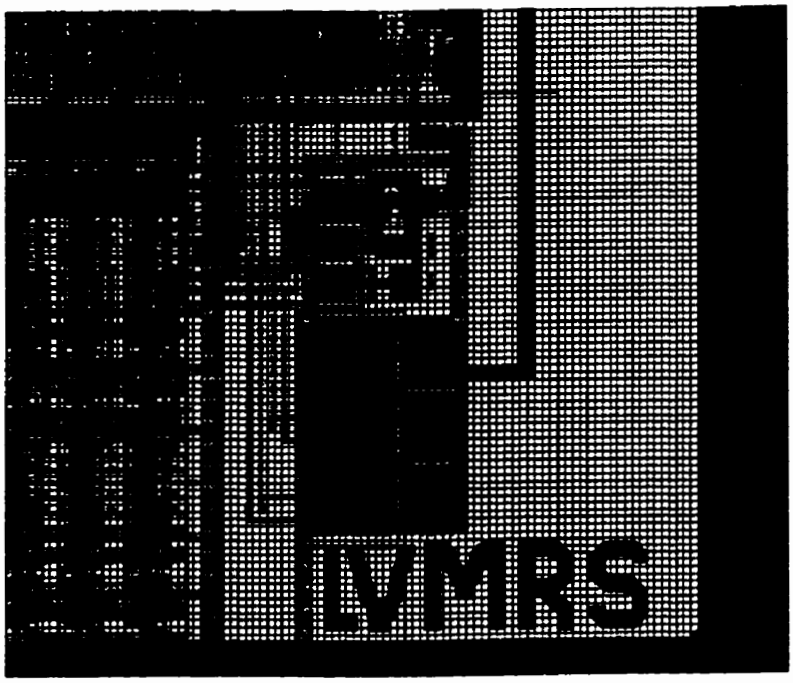


Figure E.6 Détail du coin inférieur droit du capteur MAR

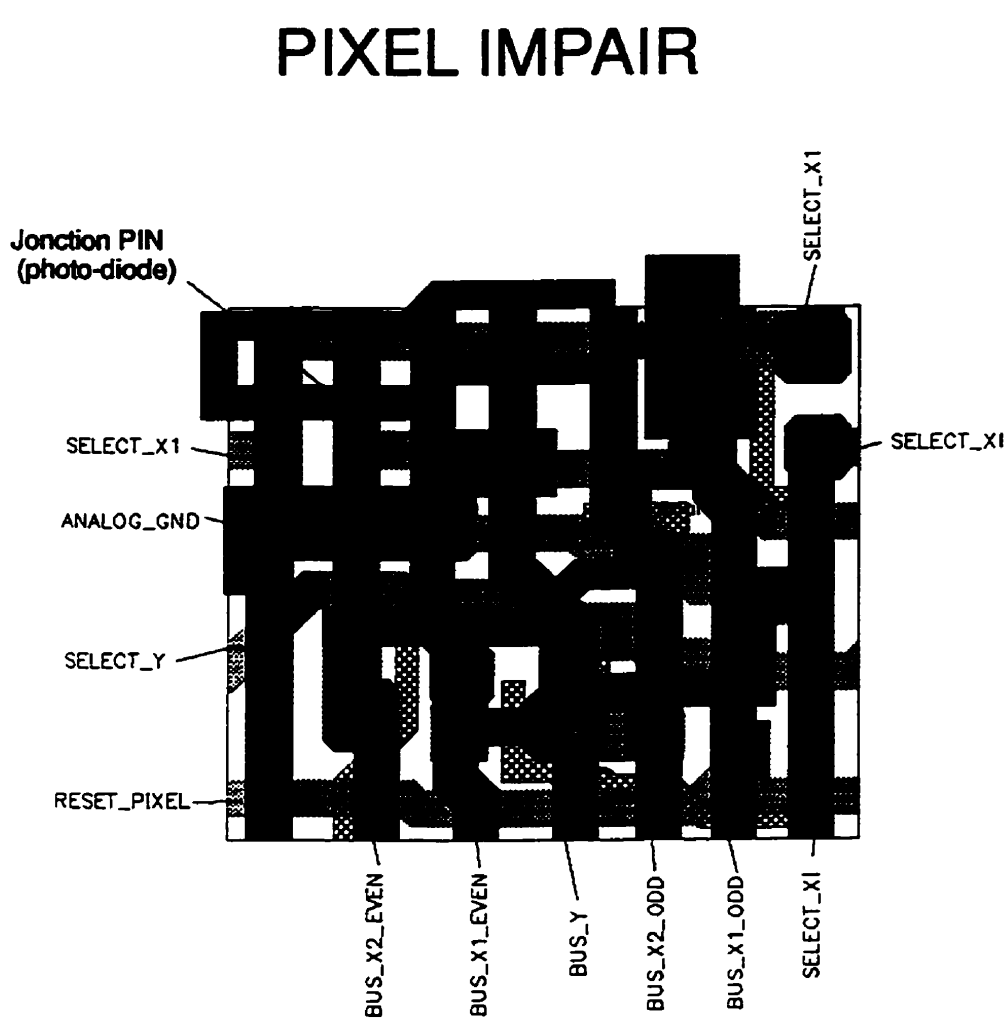


***Figure E.7** Détail d'un pixel du capteur MAR. La photo-diode se retrouve dans la partie supérieure gauche.*

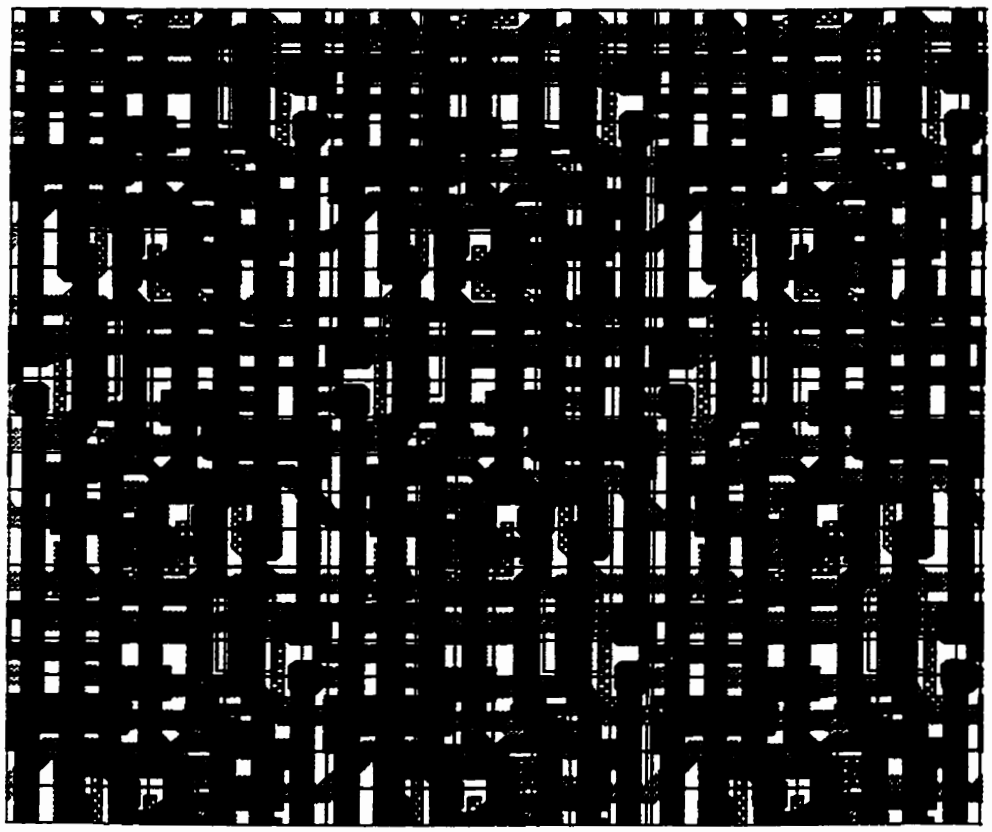


Figure E.8 Partie de la matrice de pixels de 3 x 3 éléments photo-sensibles.

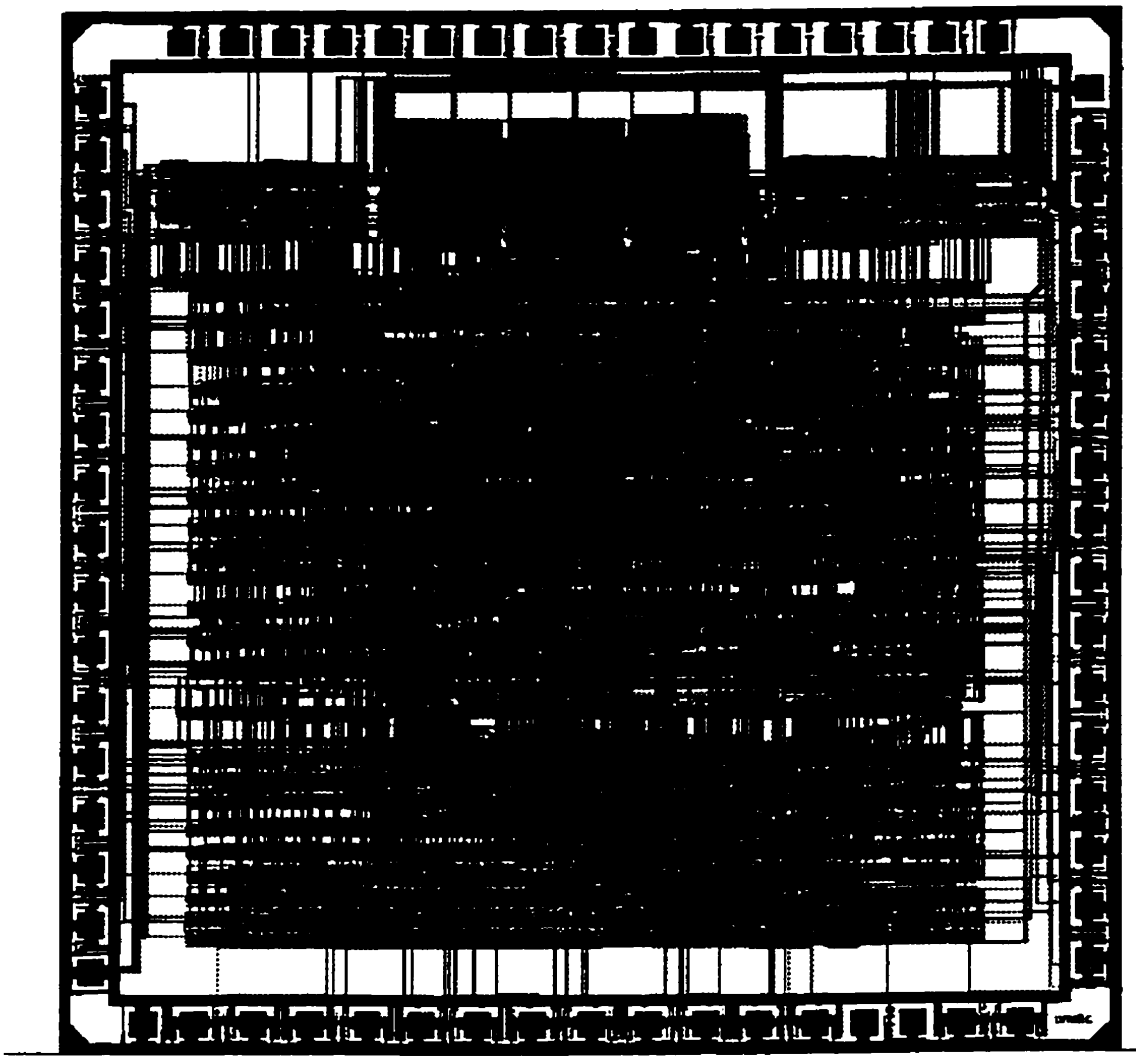


***Figure E.9** Circuit complet du contrôleur MAR. On remarque les trois parties de PLA au centre supérieur.*

EXEMPLE d'un PLA

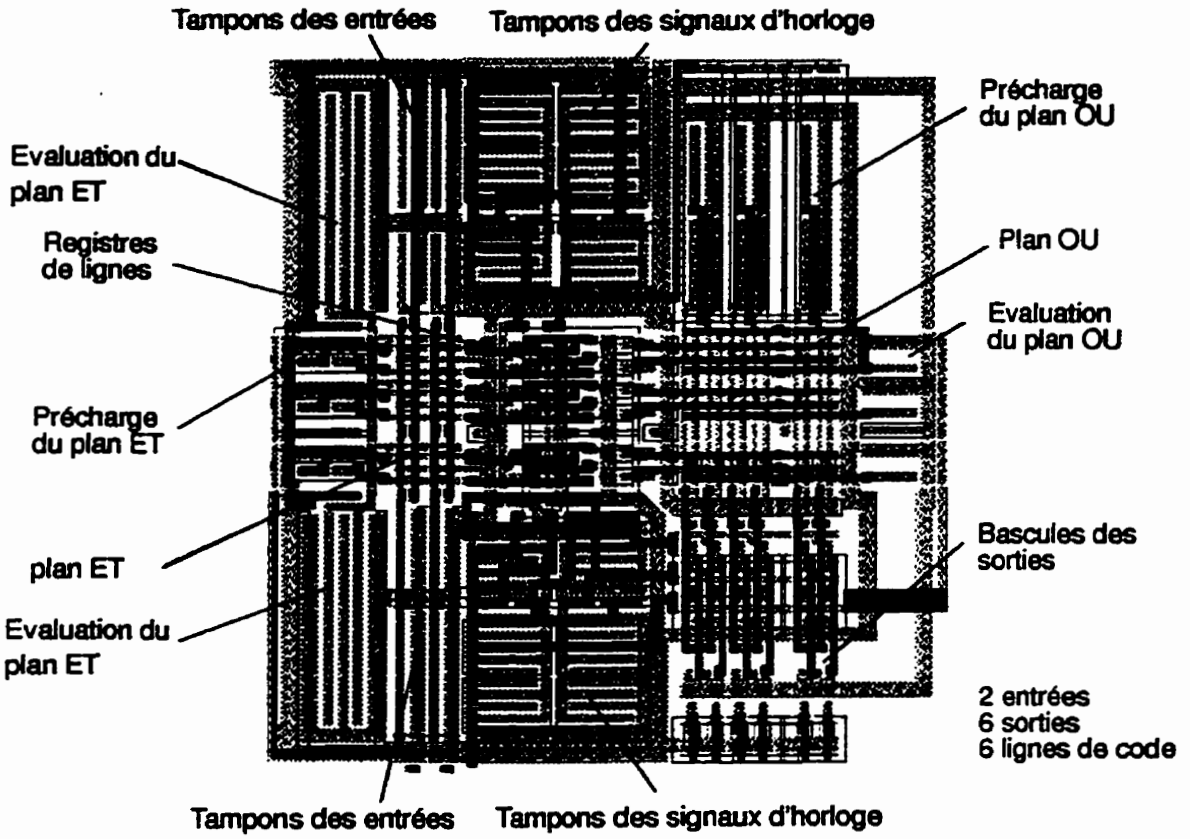


Figure E.10 *Dessin des masques d'un exemple d'un PLA de 2 entrées, 6 sorties et de 6 lignes de code.*

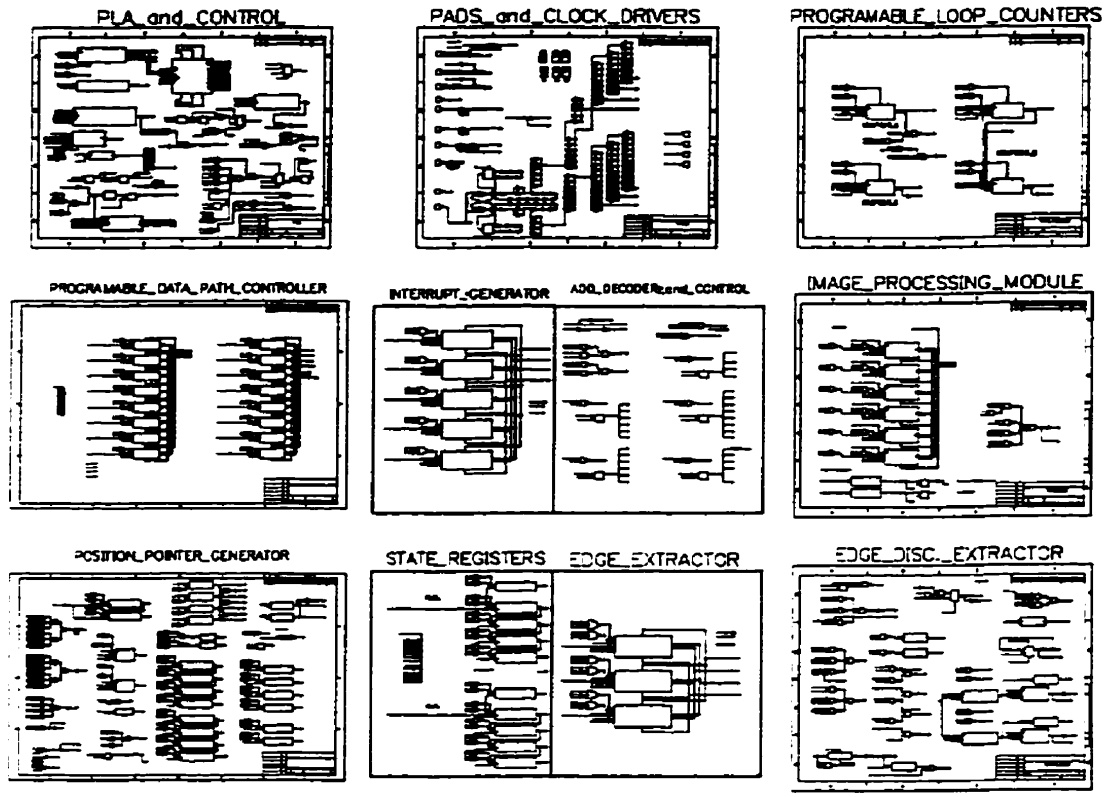


Figure E.11 Vue grossière de la description schématique du contrôleur MAR.

ANNEXE F

Plans schématiques du contrôleur réalisé à l'aide de composantes discrètes

Le contrôleur MAR à été initialement réalisé à l'aide de composantes discrètes et programmables. Cette version comprend les premier circuits intégrés du PLA qui se sont avérés fonctionnels. Le Tableau F.1 résume le contenu de chaque plan présenté dans cette annexe.

Tableau F.1 Description des différents plans du contrôleur MAR réalisé à l'aide de composantes discrètes.

NOM	Description	Référence
PLAN 1	PLA, unité arithmétique de direction et registre d'instruction	Figure F.1
PLAN 2	Module d'orientation d'arête, d'exécution conditionnelle et arbitration des interruptions	Figure F.2
PLAN 3	Compteurs d'événements N (16 bits), E et C (8bits)	Figure F.3
PLAN 4	Décodage d'adresse, Registre de configuration de base	Figure F.4
PLAN 5	Pointeur de mémoire MAR et définition de la fenêtre d'étude	Figure F.5
PLAN 6	Générateur d'horloges, Arbitration des instructions et aiguillage du bus de données local	Figure F.6
PLAN 7	Sélecteur de données et registres de description d'état	Figure F.7
PLAN 8	Sélecteur de données et détection de retour au point de départ	Figure F.8
PLAN 9	Module de dilatation et sélecteur des entrées analogiques	Figure F.9
PLAN 10	Mémoire MAR de description d'état (128 K x 16 bits)	Figure F.10
PLAN 11	Configuration des requêtes d'interruptions	Figure F.11
Connecteur	Connecteur du système hôte et du capteur MAR	Figure F.12

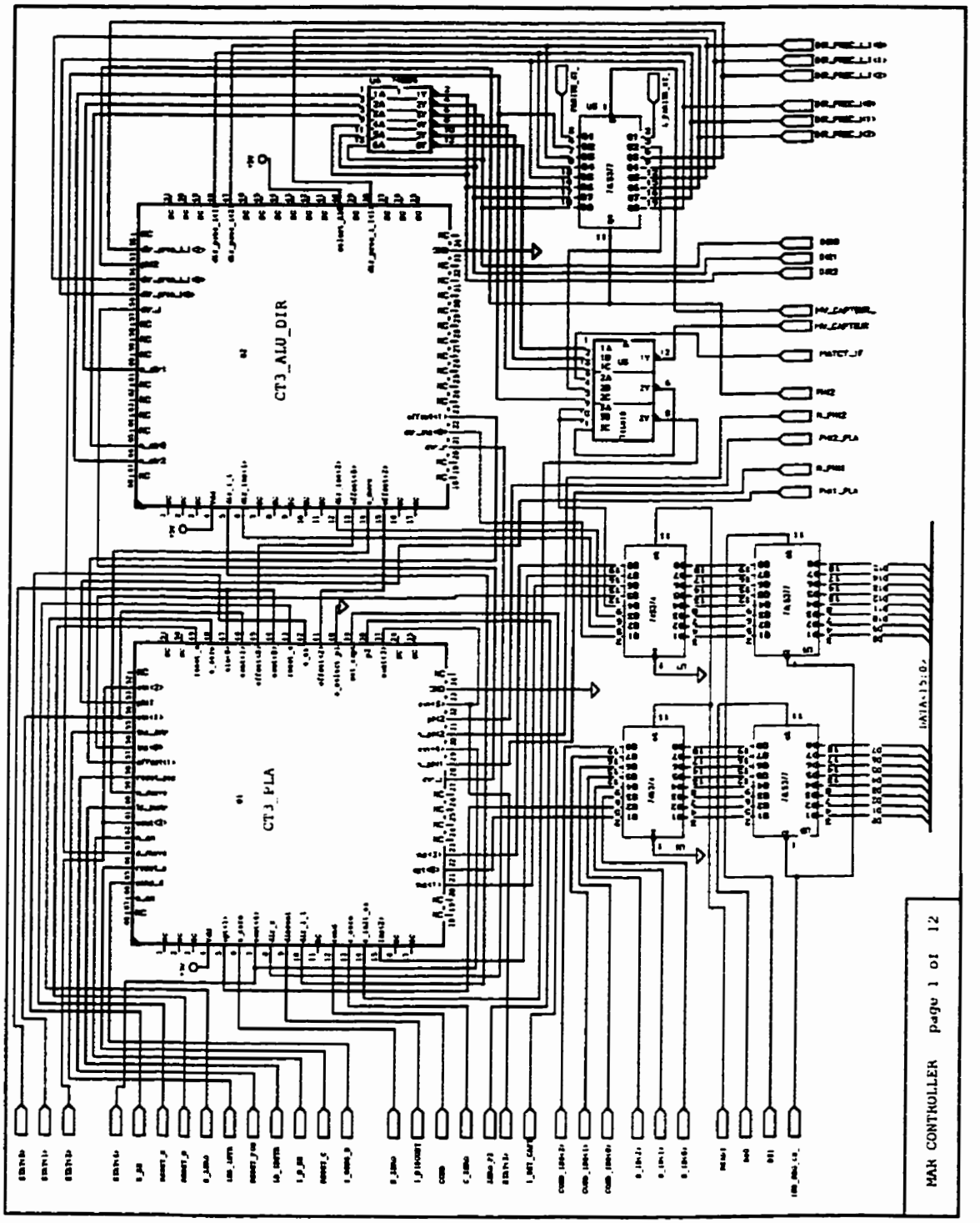


Figure F.1 PLA, unité arithmétique de direction et registre d'instruction.

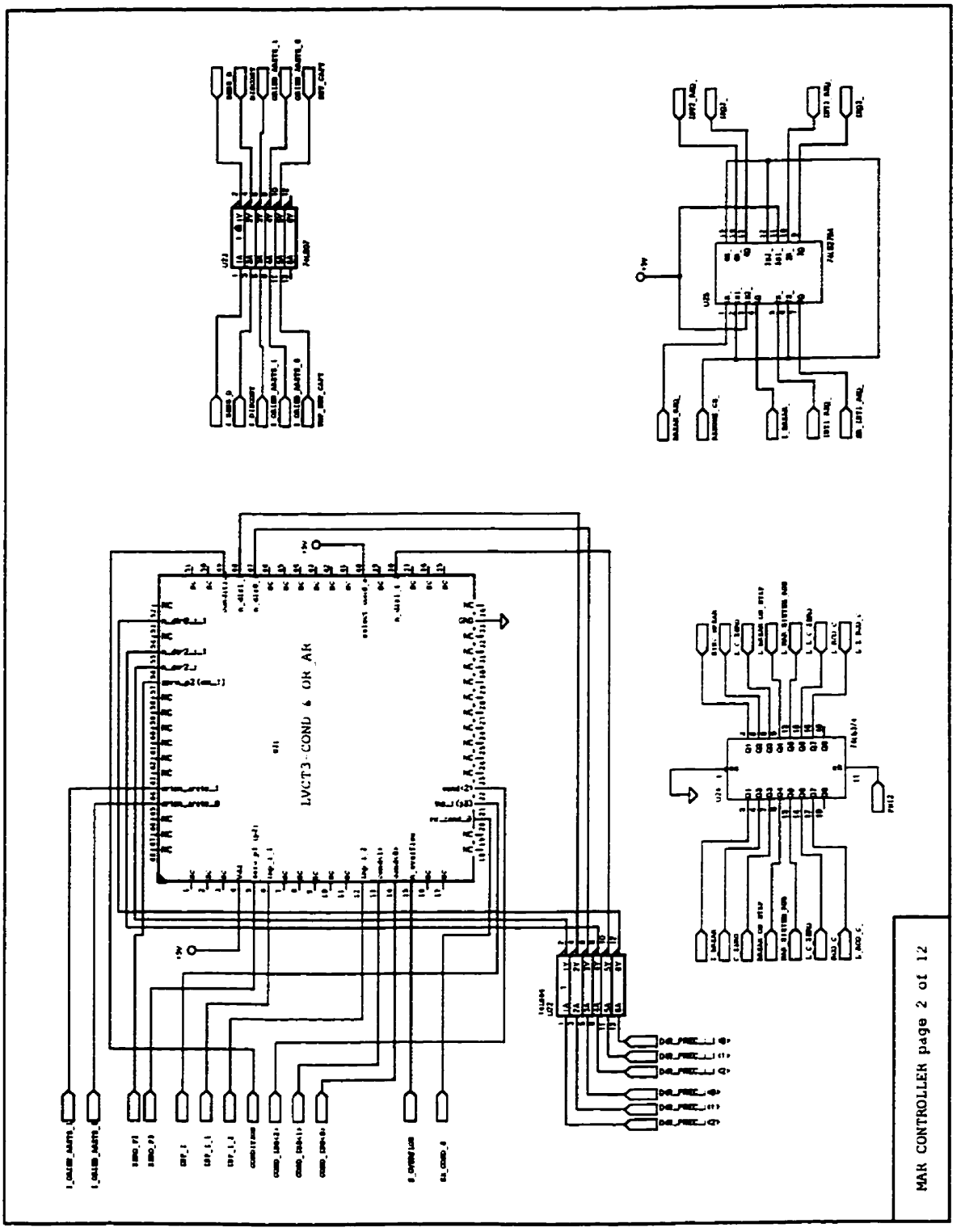


Figure F.2 *Module d'orientation d'arête, d'exécution conditionnelle et arbitration des interruptions.*

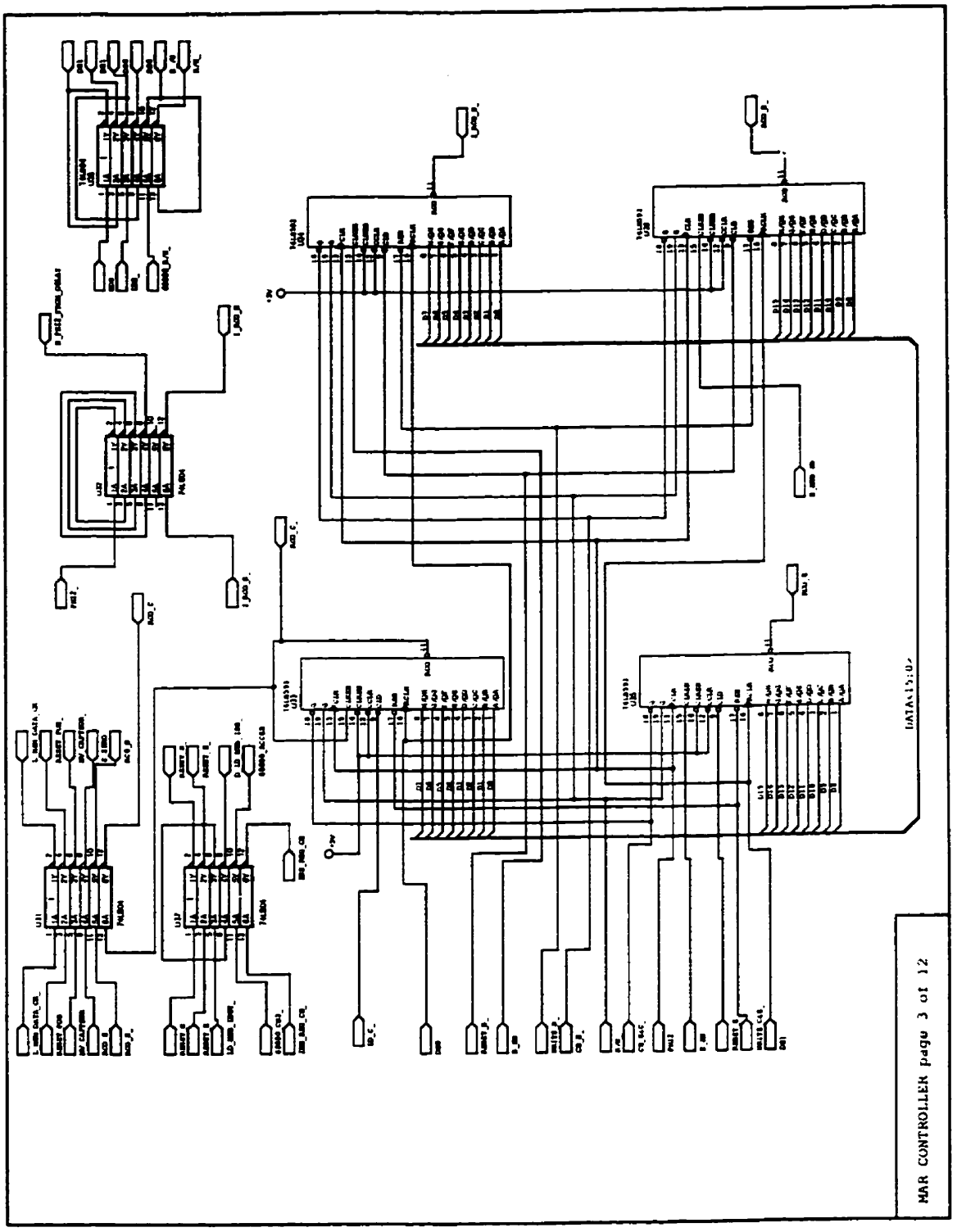


Figure F3 Compteurs d'événements N (16 bits), E et C (8bits).

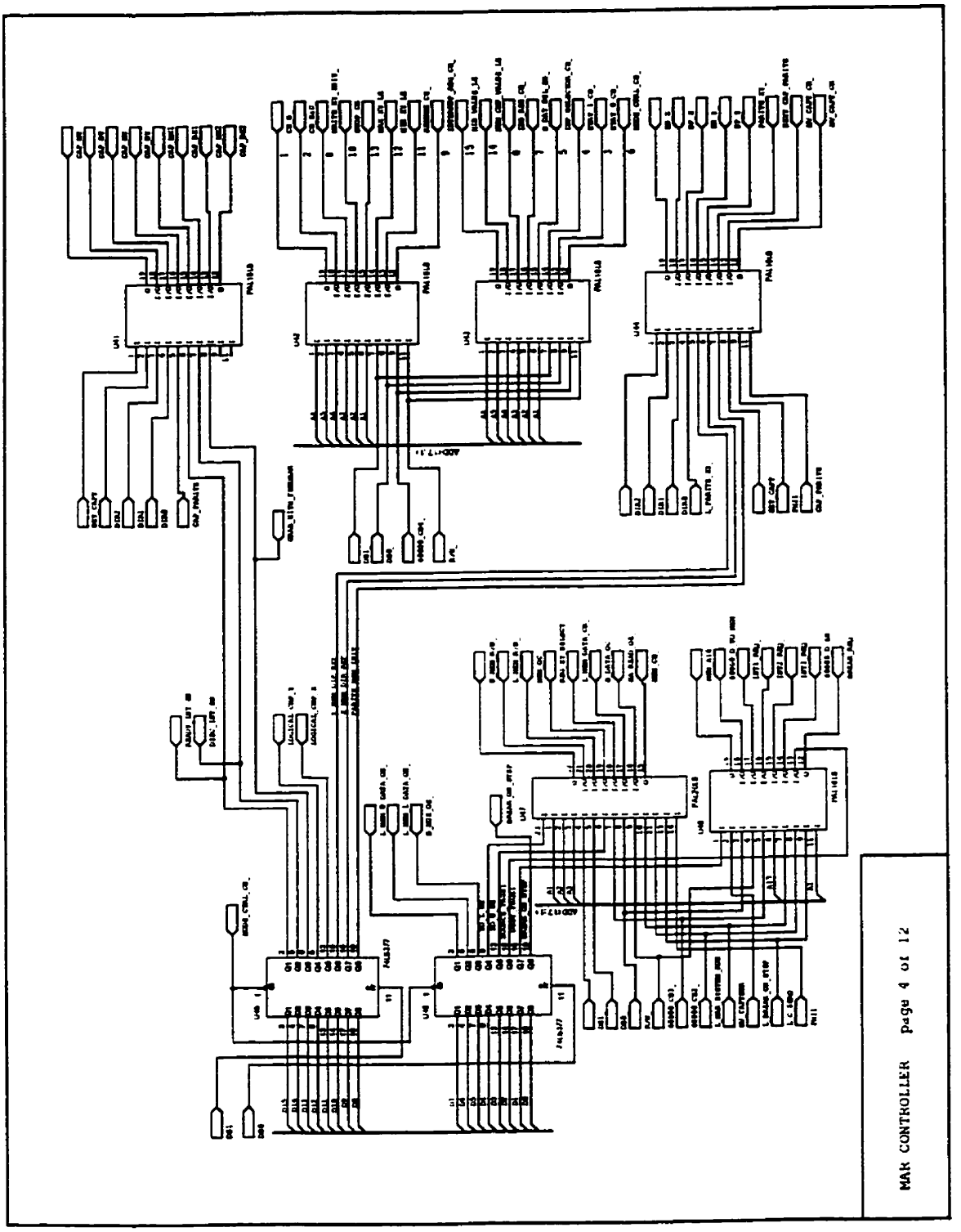


Figure F.4 Décodage d'adresse, Registre de configuration de base

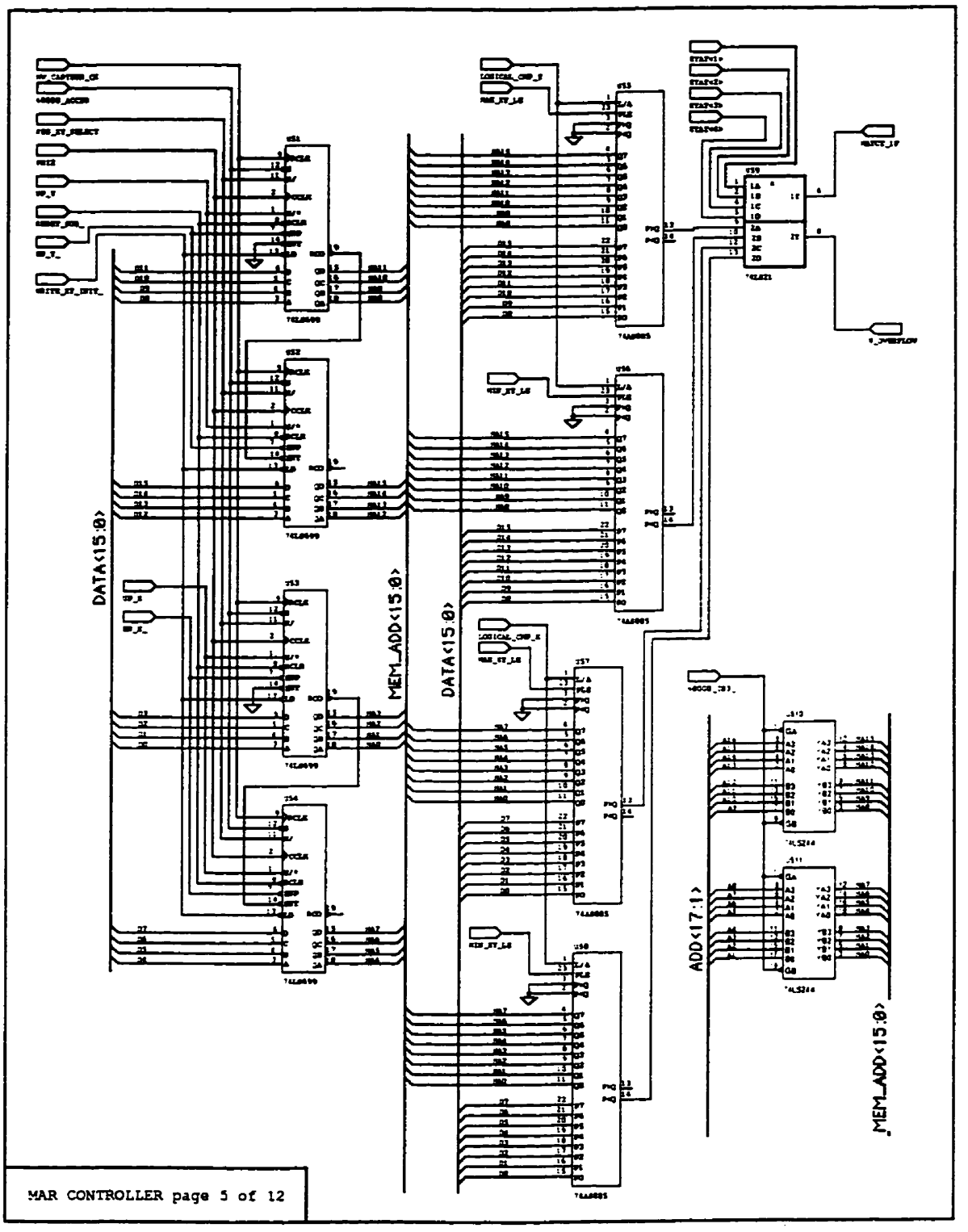


Figure F.5 Pointeur de mémoire MAR et définition de la fenêtre d'étude.

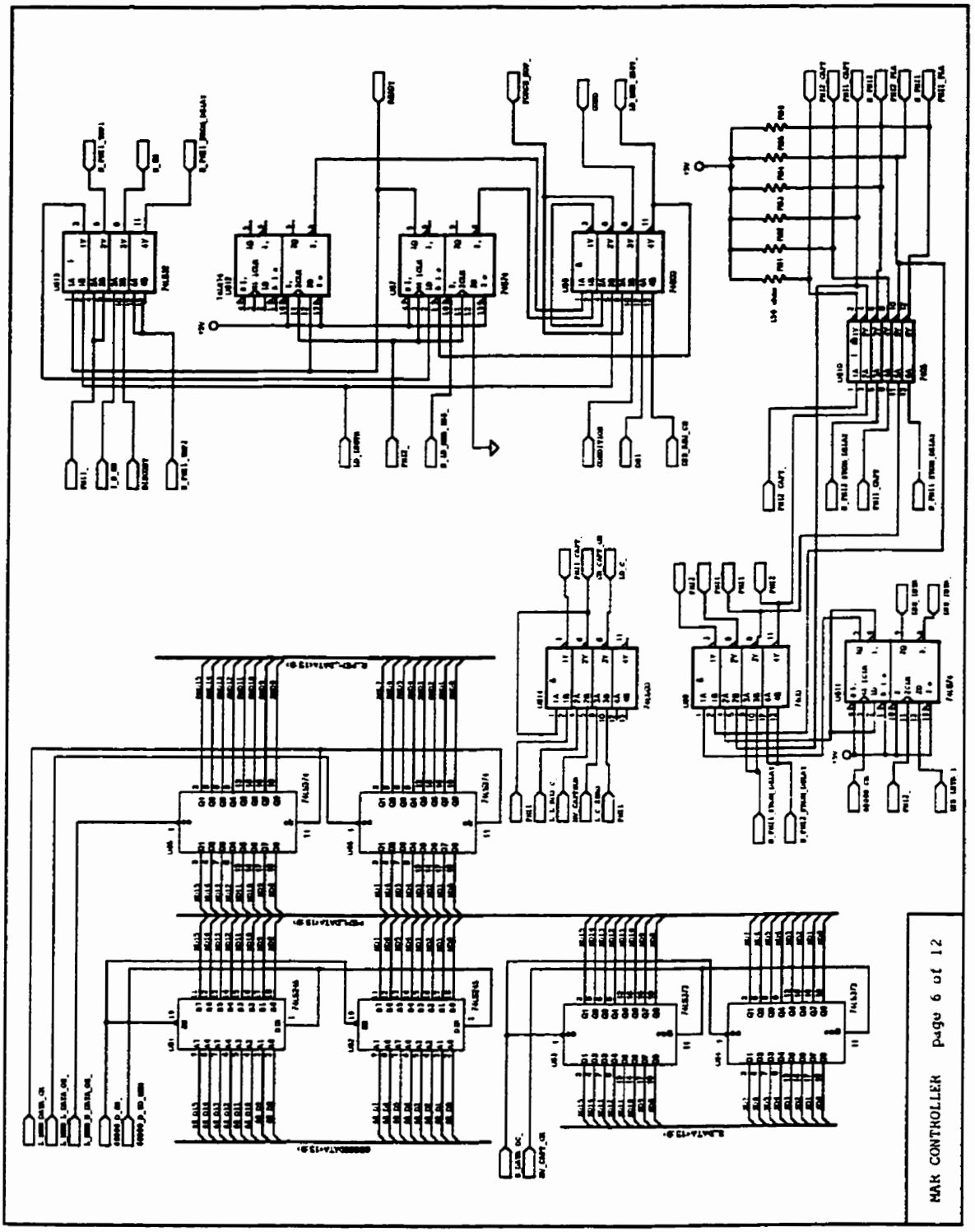


Figure F.6 Générateur d'horloges, Arbitration des instructions et aiguillage du bus de données local.

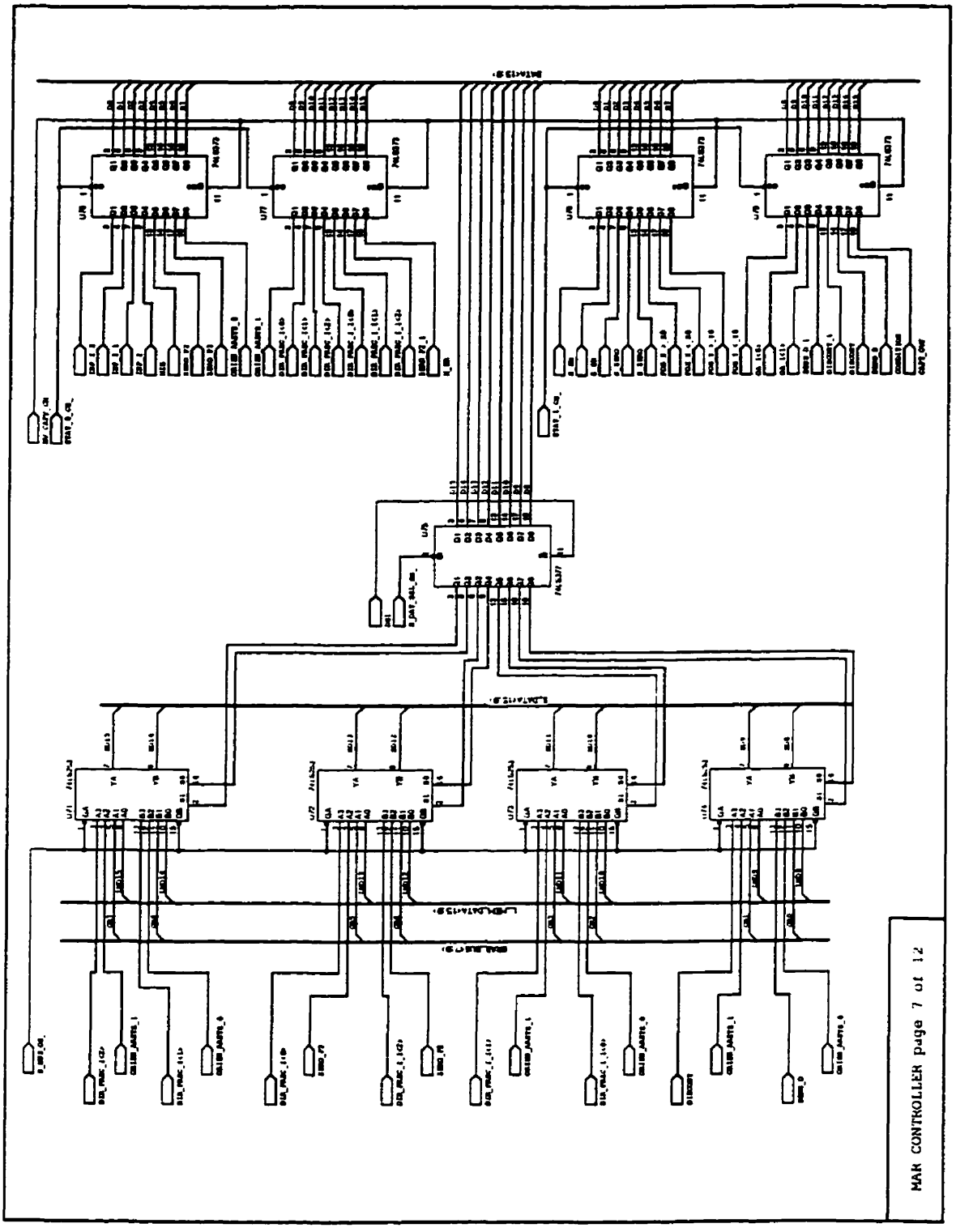


Figure F.7 Sélecteur de données et registres de description d'état.

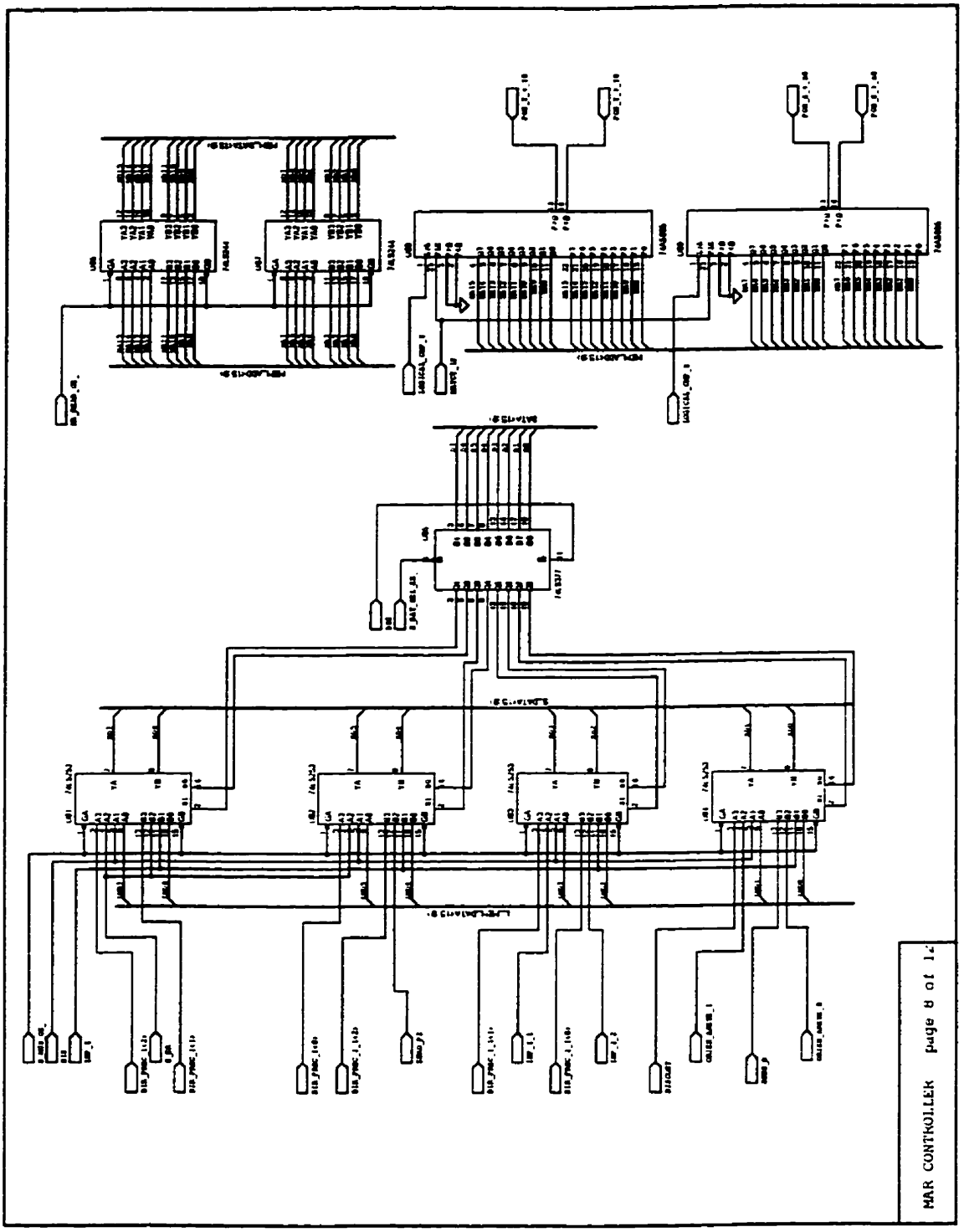


Figure F.8 Sélecteur de données et détection de retour au point de départ.

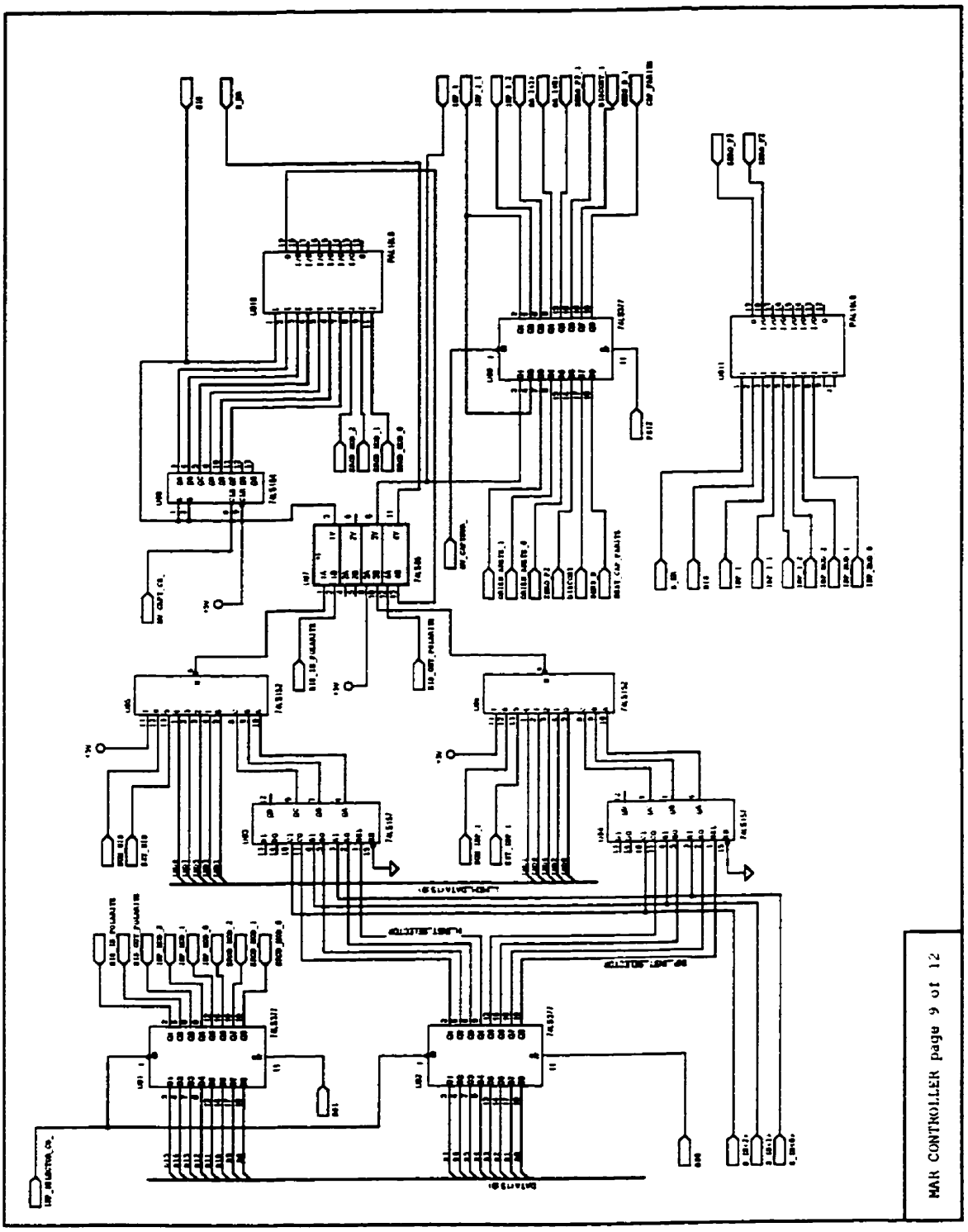


Figure F.9 Module de dilatation et sélecteur des entrées analogiques.

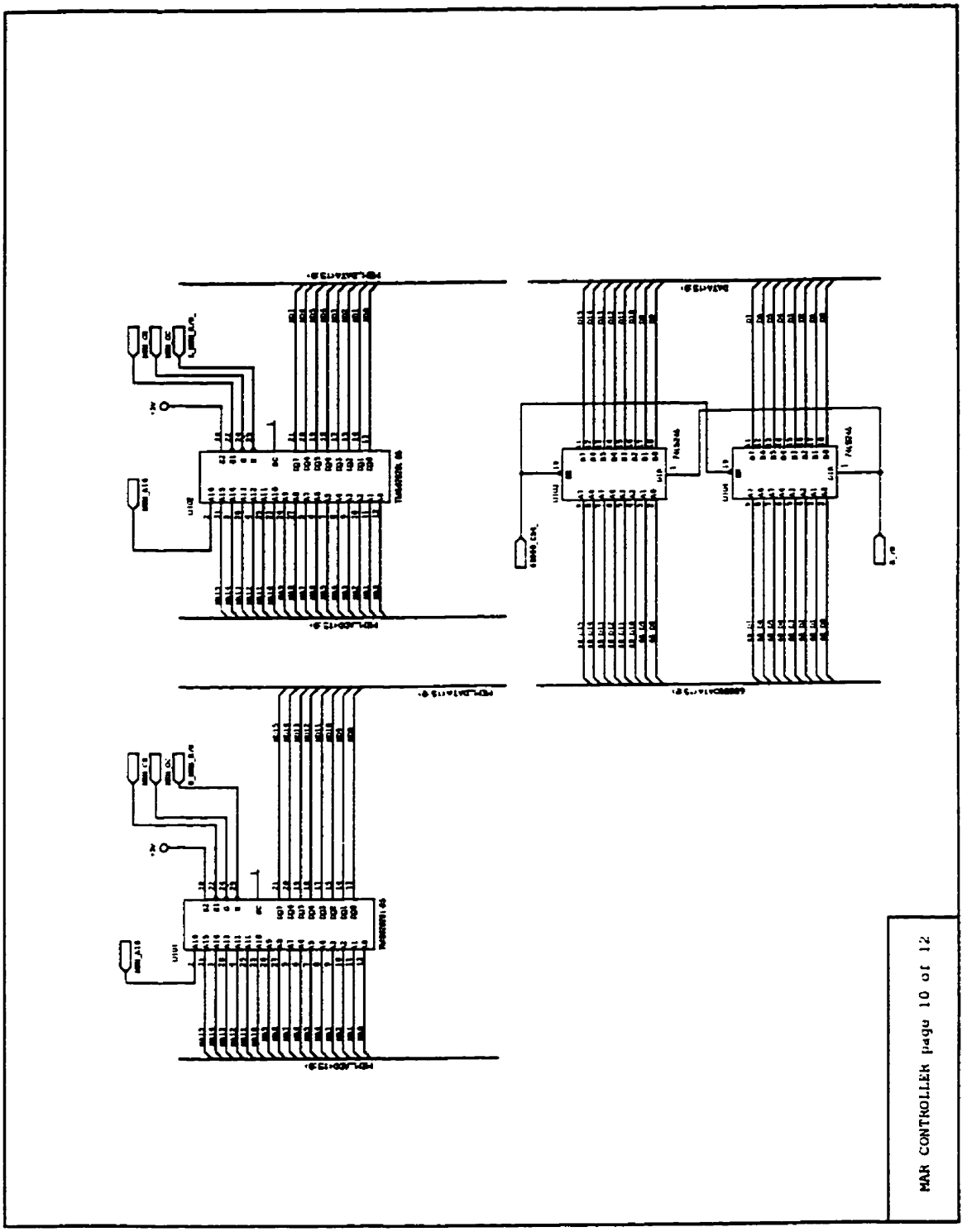


Figure F.10 Mémoire MAR de description d'état (128 K x 16 bits).

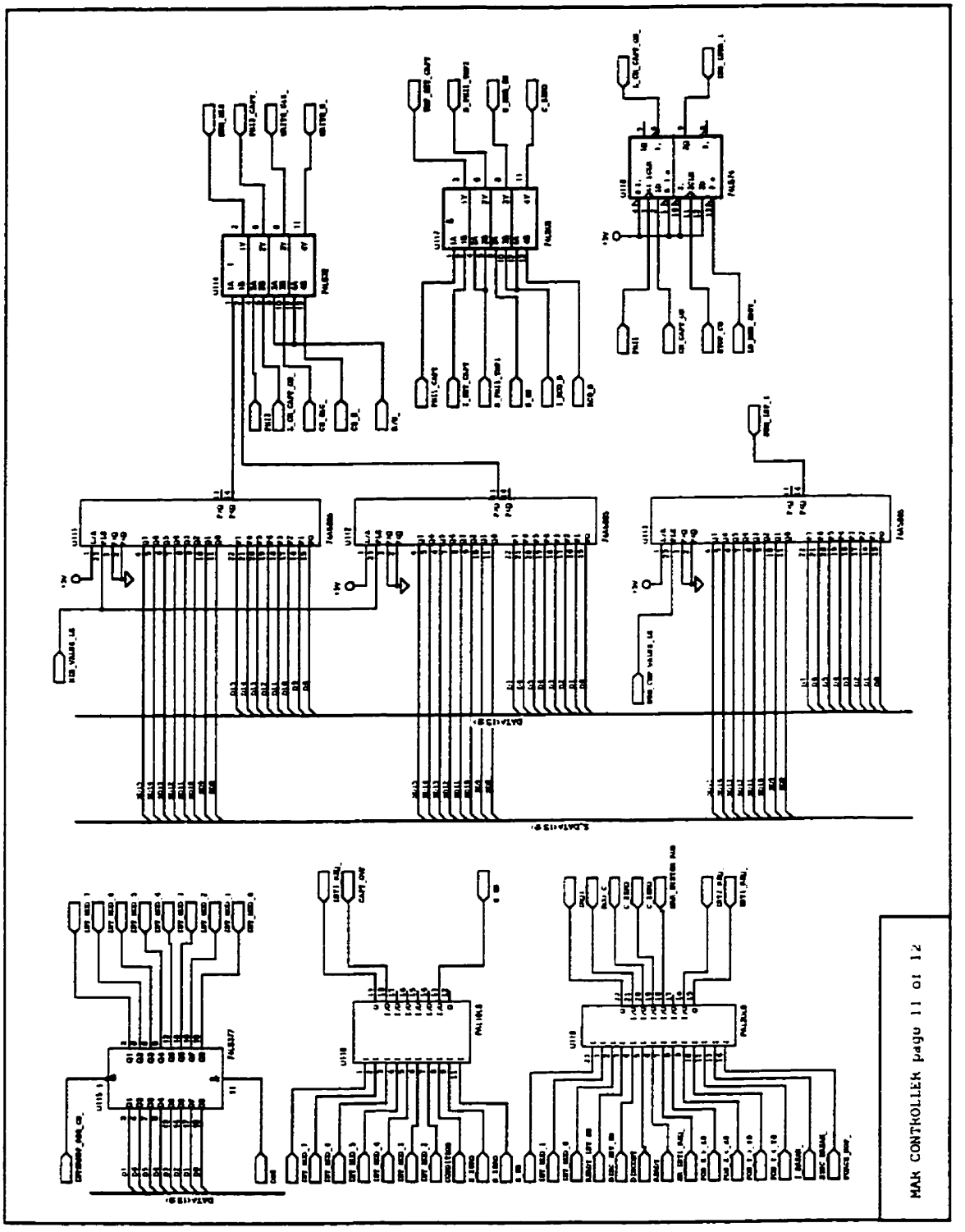


Figure F.11 Configuration des requêtes d'interruptions.

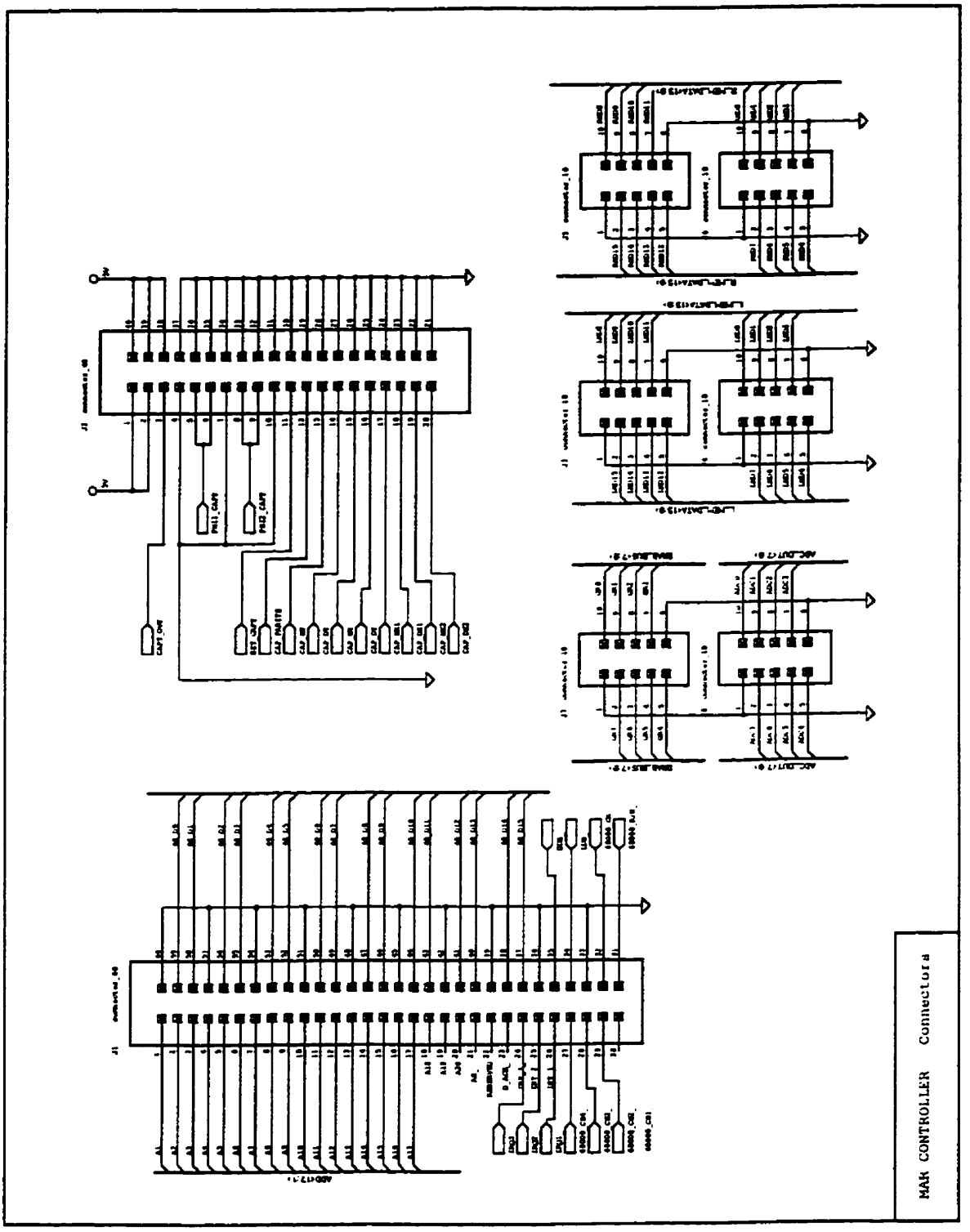


Figure F.12 Connecteur du système hôte et du capteur MAR.

ANNEXE G

Programme en C du recouvrement de surface à partir de morceaux de triangles

G.1 Programme principal "shape_main.c"

```
#include <stdio.h>
#include <rasterfile.h>
#include <math.h>
#define pi 3.14159

/* shape <image_in> <image_out> */

#define NB_COL_HEX 256+2
#define NB_LIGNE_HEX ((256+2)*8)/7

short image_cart[256][256];
char convexite[NB_LIGNE_HEX][NB_COL_HEX];
short image_hex[NB_LIGNE_HEX][NB_COL_HEX];
float image_float[NB_LIGNE_HEX][NB_COL_HEX];


main(argc,argv)
int argc;
char **argv;
{
FILE *fin;
```

```
FILE *fout;
int nb_ligne;
int nb_col;
struct rasterfile header;
short i,j;
char color_map[256];
int x_depart,y_depart;
short x_max,y_max;
int R;
float max_value,min_value;
int max_r;
int passe_1[6][2];
int passe_2[6][6][2];
int dummy[6][2];
char reponse[10];
int nb_pas;


if(argc != 3)
    {
    printf("usage: shape <f_name_in> <f_name_out>\n");
    exit(1);
    }
if ((fin = fopen(argv[1],"r")) == (FILE *)0)
    {
    printf("shape: can't open file %s\n",argv[1]);
    exit(1);
    }
if ((fout = fopen(argv[2],"w")) == (FILE *)0)
    {
    printf("shape: can't open file %s\n",argv[2]);
    exit(1);
    }
fread(&header,sizeof(struct rasterfile),1,fin);
fwrite(&header,sizeof(struct rasterfile),1,fout);
for(i=0;i<3;i++)
    {
    fread(color_map,sizeof(char),256,fin);
    fwrite(color_map,sizeof(char),256,fout);
    }
nb_ligne = header.ras_height;
nb_col = header.ras_width;
for(i=0;i<nb_ligne;i++)
    for(j=0;j<nb_col;j++)
        image_cart[i][j] = (short)getc(fin);
for(i=0;i<NB_LIGNE_HEX;i++)
    for(j=0;j<NB_COL_HEX;j++)
        {
        image_float[i][j] = -1.0;
        convexite[i][j] = (char)(0);
        }

printf("conversion cartesien to hexagonale\n");
cart2hex(image_cart,nb_ligne,nb_col,image_hex);
```

```
boucle:

x_depart = y_depart = 0;
printf("quel est le numero de la ligne de depart? :");
scanf("%d",&y_depart);
printf("quel est le numero de la colonne de depart? :");
scanf("%d",&x_depart);
printf("quel est le rayon d'observation? :");
scanf("%d",&R);
y_depart = y_depart*8/7;
printf("quel est le rayon maximal de l'hexagone? :");
scanf("%d",&max_r);
printf("Combien d'iteration desirez-vous? :");
scanf("%d",&nb_pas);


trouve_max((short)x_depart,(short)y_depart,image_hex,&x_max,&y_max,R,252);
trouve_max(x_max,y_max,image_hex,&x_max,&y_max,4,255);
/*
x_max=x_depart;
y_max=y_depart;
*/

printf("le point singulier est situe a la coordonnee cartesienne:\n");
printf("x = %15d y = %15d\n",x_max,(y_max*7)/8);

if(image_float[y_max][x_max] < 0.0)
    image_float[y_max][x_max] = 5000.0/90.0;
shape_region(convexite,y_max,x_max,image_hex,image_float,image_float[y_max][x_m
ax],max_r,passe_1);

if(nb_pas >= 2)
for(i=0;i<6;i++)
    {
    if(passe_1[i][0] == 0)
        continue;
    y_max = passe_1[i][0];
    x_max = passe_1[i][1];
    if(y_max < 0 || x_max < 0 || y_max > NB_LIGNE_HEX-1 || x_max > NB_COL_HEX-
1)
        continue;

shape_region(convexite,y_max,x_max,image_hex,image_float,image_float[y_max][x_m
ax],max_r*3/2,passe_2[i]);
    }
if(nb_pas >= 3)
for(j=0;j<6;j++)
    for(i=0;i<6;i++)
        {
        if(passe_2[j][i][0] == 0)
            continue;
        y_max = passe_2[j][i][0];
        x_max = passe_2[j][i][1];
```

```
        if(y_max < 0 || x_max < 0 || y_max > NB_LIGNE_HEX-1 || x_max >
NB_COL_HEX-1)
            continue;

shape_region(convexite,y_max,x_max,image_hex,image_float,image_float[y_max][x_m
ax],max_r*5/2,dummy);
        }


printf("Voulez vous effectuer une autre passe (y/n)? :");
scanf("%s",reponse);
if(strcmp(reponse,"y") == 0 ||strcmp(reponse,"o") == 0 || strcmp(reponse,"yes") == 0
||strcmp(reponse,"oui") == 0)
    goto boucle;

max_value = -1e30;
min_value = 1e30;

for(i=0;i<NB_LIGNE_HEX;i++)
    for(j=0;j<NB_COL_HEX;j++)
        {
        if( image_float[i][j] < 6000.0/90.0 && image_float[i][j] > max_value)
            max_value = image_float[i][j];
        if( image_float[i][j] >0.0 && image_float[i][j] < min_value)
            min_value = image_float[i][j];

}
printf("min = %f max = %f\n",min_value,max_value);
printf("conversion point flotant to entier\n");
float2char(image_hex,((nb_ligne+2)*8)/7,nb_col+2,image_float,max_value,0);

printf("conversion hexagonale to cartesien\n");
hex2cart(image_cart,nb_ligne,nb_col,image_hex);
for(i=0;i<nb_ligne;i++)
    for(j=0;j<nb_col;j++)
        putc(image_cart[i][j],fout);
return;
}
```

G.2 Fonctions requises par le programme principal: "shape_conv.c"

```
#include <math.h>
#define NB_LIGNE_HEX ((256+2)*8)/7
#define NB_COL_HEX 256+2
#define MIN_ILLUMINENCE 30
#define FACT_ECHELLE 1.0

float calcule_radical();
float calcule_prof();



float calcule_radical(e0,e1,e2,y0,y1,delta_x,e_max)
short e0,e1,e2,e_max;
float y0,y1,delta_x;
{
float valeur;

valeur = -(y0-y1)*(y0-y1)/delta_x/delta_x;
valeur -= 1.0;
valeur += ((float)e_max*3.0 / (float)(e0+e1+e2))*((float)e_max*3.0 / (float)(e0+e1+e2));
return (valeur);
}


trouve_max(x_depart,y_depart,image_hex,x_max,y_max,R,quota)
short x_depart;
short y_depart;
short image_hex[NB_LIGNE_HEX][NB_COL_HEX];
short *x_max;
short *y_max;
short R;
int quota;
{
int dx,dy;
int i,j;
int x,y;
int cond_arret = 0;
short valeur[6];
short max_valeur;
x = x_depart;
y = y_depart;
while (cond_arret == 0)
    {
    max_valeur = image_hex[y][x];
    i = y;
    j = x;
    for(dx = -R; dx <= R;dx++)
        {
        if(dx == 0) continue;
        for(dy = -R; dy <= R;dy++)
            {
```

```
            if(dy == 0) continue;
            if(x+dx > 0 && x+dx < NB_COL_HEX-1 &&
             y+dy > 0 && y+dy < NB_LIGNE_HEX-1 &&
             image_hex[y+dy][x+dx] <= quota &&
             image_hex[y+dy][x+dx] > max_valeur)
                {
                max_valeur = image_hex[y+dy][x+dx];
                i = y+dy;
                j = x+dx;
                }
            }
        }
    if (i == y && j == x)
        {
        if (max_valeur < 250)
            {
            if (y < NB_LIGNE_HEX - 2*R -1 && x < NB_COL_HEX - 2*R-1)
                {
                i+=2*R;
                j+=2*R;
                }
            else
                cond_arret = 1;
            }
        else
            cond_arret = 1;
        }
    x = j;
    y = i;
    }
*x_max = x;
*y_max = y;
}


calcule_convexite(direction,convexite,image_hex,image_float,y2,y1,y0,x2,x1,x0)
int direction;
char convexite[NB_LIGNE_HEX][NB_COL_HEX];
short image_hex[NB_LIGNE_HEX][NB_COL_HEX];
float image_float[NB_LIGNE_HEX][NB_COL_HEX];
int y2,y1,y0,x2,x1,x0;
{
int y3,y4,x3,x4;
int y5,y6,x5,x6;
int signe_35;
int signe_36;
int signe_45;
int signe_46;
int val_2;
int cmptr_3;
int cmptr_4;
int cmptr_5;
int cmptr_6;
```

```
int conv_val;

x3 = x4 = x2;
y3 = y4 = y2;
y5 = y2;
x5 = x2;
y6 = y2;
x6 = x2;

move(direction-1,&y4,&x4);
move(direction-2,&y3,&x3);
move(direction+1,&y6,&x6);
move(direction+2,&y5,&x5);
conv_val = convexite[y5][x5] + convexite[y6][x6];


for(cmptr_3 = 1;image_hex[y3][x3] == image_hex[y2][x2];cmptr_3++)
    move(direction-2,&y3,&x3);
for(cmptr_4 = 1;image_hex[y4][x4] == image_hex[y2][x2];cmptr_4++)
    move(direction-1,&y4,&x4);
for(cmptr_5 = 1;image_hex[y5][x5] == image_hex[y2][x2];cmptr_5++)
    {
    move(direction+2,&y5,&x5);
    conv_val += convexite[y5][x5];
    }
for(cmptr_6 = 1;image_hex[y6][x6] == image_hex[y2][x2];cmptr_6++)
    {
    move(direction+1,&y6,&x6);
    conv_val += convexite[y6][x6];
    }

val_2 = (int)image_hex[y2][x2];
signe_35 = (image_hex[y3][x3] - val_2) * (image_hex[y5][x5] - val_2);
signe_45 = (image_hex[y4][x4] - val_2) * (image_hex[y5][x5] - val_2);
signe_36 = (image_hex[y3][x3] - val_2) * (image_hex[y6][x6] - val_2);
signe_46 = (image_hex[y4][x4] - val_2) * (image_hex[y6][x6] - val_2);

if(signe_35 > 0 && signe_36 > 0 && signe_45 > 0 && signe_46 > 0 &&
image_hex[y3][x3] < val_2 && image_hex[y4][x4] < val_2 && _ABS(cmptr_4 -
cmptr_5) <= 1 && _ABS(cmptr_3 - cmptr_6) <= 1 && _ABS(conv_val)-(cmptr_5 +
cmptr_6) <= 1)



    {
/*
putchar('S');
*/
    if(conv_val > 0)
        convexite[y2][x2] = -1;
    else
        convexite[y2][x2] = 1;
```

```
    }
else if(convexite[y0][x0] != convexite[y1][x1])
    {
    y3 = y2;
    x3 = x2;
    move(direction-3,&y3,&x3);
    if(convexite[y3][x3] + convexite[y1][x1] + convexite[y0][x0] > 0)
        convexite[y2][x2] = 1;
    else
        convexite[y2][x2] = -1;
    }
else
    convexite[y2][x2] = convexite[y0][x0];


/*
if (convexite[y2][x2] == 0)
    {
    if(_ABS(conv_val) == cmptr_56 && _ABS(cmptr_56-cmptr_34) < 3 && signe > 0.9
&& _ABS(image_float[y0][x0] - image_float[y5][x5]) < 2.0 && image_hex[y2][x2] >
150)
        {
putchar('S');
        if(conv_val > 0)
            conv_val = -1;
        else
            conv_val = 1;
        while(y3 != y0 || x3 != x0)
            {
            if (y3 == y4 && x3 == x4)
                {
                move(direction+2,&y3,&x3);
                move(direction+1,&y4,&x4);
                }
            else
                {
                move(direction+1,&y3,&x3);
                move(direction+2,&y4,&x4);
                }
            convexite[y3][x3] = conv_val;
            convexite[y4][x4] = conv_val;
            }
        }
    else
        {
        if (convexite[y2][x2] == 0)
            convexite[y2][x2] = convexite[y0][x0];
        else if (conv_val > 0)
            convexite[y2][x2] = 1;
        else
            convexite[y2][x2] = -1;
        }
    }
*/
```

```
}



float
calcule_prof(direction,r_inibit,convexite,y0,y1,e0,e1,e2,e_max,ligne_2,ligne_1,ligne_0,c
ol_2,col_1,col_0,image_hex,image_float,erreur)
int direction;
int r_inibit;
char convexite[NB_LIGNE_HEX][NB_COL_HEX];
float y0,y1;
short e0,e1,e2,e_max;
int ligne_2,ligne_1,ligne_0,col_2,col_1,col_0;
short image_hex[NB_LIGNE_HEX][NB_COL_HEX];
float image_float[NB_LIGNE_HEX][NB_COL_HEX];
float *erreur;
{
float valeur;
float delta_z;
float delta_x;


if (e0+e1+e2 < MIN_ILLUMINENCE)
    return (-1.0);
if (r_inibit > 4)

calcule_convexite(direction,convexite,image_hex,image_float,ligne_2,ligne_1,ligne_0,co
l_2,col_1,col_0);
else
    convexite[ligne_2][col_2] = convexite[ligne_0][col_0];
if(ligne_0 == ligne_1)
    {
    delta_x = 1.0 ;
    delta_z =0.875;
    }
else
    {
    delta_x = 1.0077822 ;
    delta_z = 0.86375719;
    }
if ((valeur = calcule_radical(e0,e1,e2,y0,y1,delta_x,e_max)) < 0.0)
    {
    *erreur = (float)sqrt((double)(-valeur));
    return ((y0 + y1)/2.0);
    }
else
    {
    *erreur = 0.0;
    return((y0 + y1)/2.0 + convexite[ligne_2][col_2] * delta_z * (float)sqrt(valeur));
    }
}

/*****************************************************************************/
```

```
/* move dans la direction 1 a 5 */
/* */
/* 1 = a droite */
/* 2 = en haut a droite */
/* 3 = en haut a gauche */
/* 4 = a gauche */
/* 5 = en bas a gauche */
/* 6 = en bas a droite */
/* */
/*****************************************************************************/

move(direction,i_hex,j_hex)
int direction;
int *i_hex,*j_hex;
{
int return_value = 0;

while(direction < 0)
    direction+=6;
switch(direction%6)
    {
    case 1:(*j_hex)++;
        break;
    case 2:if(*i_hex & 1)
            (*j_hex)++;
        (*i_hex)--;
        break;
    case 3:if(!(*i_hex & 1))
            (*j_hex)--;
        (*i_hex)--;
        break;
    case 4:(*j_hex)--;
        break;
    case 5:if(!(*i_hex & 1))
            (*j_hex)--;
        (*i_hex)++;
        break;
    case 0:if(*i_hex & 1)
            (*j_hex)++;
        (*i_hex)++;
        break;
    }
if (*i_hex == -1)
    return_value = -1;
if (*j_hex == -1)
    return_value = -1;
if (*i_hex == NB_LIGNE_HEX)
    return_value = -1;
if (*j_hex == NB_COL_HEX)
    return_value = -1;
return(return_value);
}
```

```
float2char(image_hex,nb_ligne_hex,nb_col_hex,image_float,max_value,min_value)
short image_hex[NB_LIGNE_HEX][NB_COL_HEX];
short nb_ligne_hex;
short nb_col_hex;
float image_float[NB_LIGNE_HEX][NB_COL_HEX];
float max_value;
float min_value;
{
short i;
short j;

for(i=0;i<nb_ligne_hex;i++)
    for(j=0;j<nb_col_hex;j++)
        {
        if(image_float[i][j] < min_value)
            image_hex[i][j] = 0;
        else
            image_hex[i][j] = (short)((image_float[i][j] - min_value)*255.0/(max_value-
min_value));
        }
}



shape_region(convexite,y_depart,x_depart,image_hex,image_float,z0_depart,max_r,depa
rt)
char convexite[NB_LIGNE_HEX][NB_COL_HEX];
int y_depart,x_depart;
short image_hex[NB_LIGNE_HEX][NB_COL_HEX];
float image_float[NB_LIGNE_HEX][NB_COL_HEX];
float z0_depart;
int max_r;
int depart[6][2];
{

int i,j;
int cmptr,d;
int cond_arret = 0;
int r=0;
int x0,y0,x1,y1,x2,y2;
float erreur;
int deplacement, next_deplacement;
float z_cumulatif;
float z_precedent;
float z_tmp;
int direction;
int conv_val;

for(i=0;i<6;i++)
    for(j=0;j<2;j++)
        depart[i][j] = 0;
if(image_float[y_depart][x_depart] < 0.0)
    image_float[y_depart][x_depart] = z0_depart;
if(convexite[y_depart][x_depart] == 0)
```

```
    convexite[y_depart][x_depart] = -1;
x0 = x1 = x2 = x_depart;
y0 = y1 = y2 = y_depart;
for(direction = 1;direction < 7;direction ++)
    {
    move(direction,&y2,&x2) ;
    if(image_float[y2][x2] < 0.0)
        image_float[y2][x2]    =    FACT_ECHELLE    *    calcule_prof(direction-
1,r,convexite,z0_depart,z0_depart,image_hex[y0][x0],(image_hex[y0][x0]+image_hex[y
2][x2])                                                                    /
2,image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex,image_float,image_float,&
erreur);
    y2 = y0;
    x2 = x0;
    }
for(direction = 1;direction < 7;direction ++)
    {
    move(direction,&y1,&x1);
    move(direction+1,&y2,&x2);
    if(image_float[y2][x2] < 0.0)
        {
        z_tmp                =                FACT_ECHELLE                *
calcule_prof(direction,r,convexite,z0_depart,image_float[y1][x1],image_hex[y0][x0],ima
ge_hex[y1][x1],image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex,image_float,
image_float,&erreur);

/*
printf("direction=%1d          premiere_valeur=%6f          deuxieme_valeur=%6f
erreur=%6f\n",direction,z_tmp,image_float[y2][x2],erreur);
*/

        image_float[y2][x2] += z_tmp;
        image_float[y2][x2] /= 2.0;
        }
    y2 = y1 = y0;
    x2 = x1 = x0;
    }
move(direction,&y0,&x0);
direction += 2;
for(r=1;r < max_r;r++)
    {
    z_cumulatif = z_precedent = 0.0;
    for(cmptr = 0;cmptr < 6; cmptr++,direction++)
        {
        if (direction >= 7)
            direction -= 6;
        deplacement = r;
        for(d=0;d < deplacement;d++)
            {
            y2 = y1 = y0;
            x2 = x1 = x0;
            if (move (direction,&y1,&x1) != 0)
                {
                move(direction,&y0,&x0);
```

```
                continue;
                }

            if(r == max_r-3 && y2 > 0 && x2 > 0 && y2 < NB_LIGNE_HEX && x2 <
NB_COL_HEX && d > deplacement/4 && depart[cmptr][0] == 0 &&
image_float[y2][x2] > z0_depart/10.0 && image_hex[y2][x2] > 150)
            {
            depart[cmptr][0] = y2;
            depart[cmptr][1] = x2;
            }
            if (move (direction-1,&y2,&x2) != 0)
                {
                move(direction,&y0,&x0);
                continue;
                }
            if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] >= 0.0 &&
image_float[y2][x2] < 0.0)
                image_float[y2][x2] = FACT_ECHELLE *
calcule_prof(direction,r,convexite,image_float[y0][x0],image_float[y1][x1],image_hex[y
0][x0],image_hex[y1][x1],image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex,i
mage_float,&erreur);
            else if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] < 0.0)
                {
                move(direction,&y0,&x0);
                if(move(direction,&y1,&x1) != 0)
                    continue;
                if(move(direction,&y2,&x2) != 0)
                    continue;
                if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] >= 0.0 &&
image_float[y2][x2] < 0.0)
                    image_float[y2][x2] = FACT_ECHELLE *
calcule_prof(direction,r,convexite,image_float[y0][x0],image_float[y1][x1],image_hex[y
0][x0],image_hex[y1][x1],image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex,i
mage_float,&erreur);
                if(move(direction-2,&y1,&x1) != 0)
                    continue;
                if(move(direction+3,&y2,&x2) != 0)
                    continue;
                if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] >= 0.0 &&
image_float[y2][x2] < 0.0)
                    image_float[y2][x2] = FACT_ECHELLE * calcule_prof(direction-
2,r,convexite,image_float[y0][x0],image_float[y1][x1],image_hex[y0][x0],image_hex[y1
][x1],image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex,image_float,&erreur);
                move(direction+2,&y1,&x1);
                move(direction+3,&y0,&x0);
                }
            else if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] >= 0.0 &&
image_float[y2][x2] < 0.0)
                {
                y1 = y0;
                x1 = x0;
                move(direction-2,&y1,&x1);
                if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] >= 0.0)
                    image_float[y2][x2] = FACT_ECHELLE * calcule_prof(direction-
```

```
2,r,convexite,image_float[y0][x0],image_float[y1][x1],image_hex[y0][x0],image_hex[y1
][x1],image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex,image_float,&erreur);
                }

/*
printf ("x=%3d y=%3d prof= %6f r=%2d direct=%1d
depl=%2d\n",x2,y2,image_float[y2][x2],r,direction,d);
printf("(%3d,%3d)%f ",x2,y2,image_float[y2][x2]);
*/

            if(image_float[y2][x2] >= 0.0)
                {
                z_cumulatif += (image_float[y2][x2] - z_precedent);
                z_precedent = image_float[y2][x2];
                }
            move(direction,&y0,&x0);
            }

/*
printf("\n");
*/

        if(x0 < 0 || x0 >= NB_COL_HEX || y0 < 0 || y0 >= NB_LIGNE_HEX)
            continue;
        conv_val = 0;
/*
putchar('(');
*/
        y2 = y1 = y0;
        x2 = x1 = x0;
        if (move (direction,&y2,&x2) != 0)
            continue;
        if (move (direction+1,&y1,&x1) != 0)
            continue;
        if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] >= 0.0 &&
image_float[y2][x2] < 0.0)
            image_float[y2][x2] = FACT_ECHELLE *
calcule_prof(direction+1,r,convexite,image_float[y0][x0],image_float[y1][x1],image_hex
[y0][x0],image_hex[y1][x1],image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex
,image_float,&erreur);

/*
printf("direction: %f ",image_float[y2][x2]);
*/

        y2 = y1 = y0;
        x2 = x1 = x0;
        if (move (direction-1,&y2,&x2) != 0)
            continue;
        if (move (direction-2,&y1,&x1) != 0)
            continue;
        if(image_float[y1][x1] >= 0.0 && image_float[y0][x0] >= 0.0 &&
image_float[y2][x2] < 0.0)
            {
```

```
            image_float[y2][x2]          =          FACT_ECHELLE          *
calcule_prof(direction+1,0,convexite,image_float[y0][x0],image_float[y1][x1],image_he
x[y0][x0],image_hex[y1][x1],image_hex[y2][x2],(short)255,y2,y0,y1,x2,x0,x1,image_he
x,image_float,&erreur);
            conv_val = convexite[y2][x2];
            }

/*
printf("direction+1(premier): %f ",image_float[y2][x2]);
*/

        y1 = y0;
        x1 = x0;
        if (move (direction,&y1,&x1) != 0)
            continue;
        if(image_float[y1][x1]  >=  0.0  &&  image_float[y0][x0]  >=  0.0  &&
image_float[y2][x2] < 0.0)
            z_tmp                =                FACT_ECHELLE                *
calcule_prof(direction,0,convexite,image_float[y0][x0],image_float[y1][x1],image_hex[y
0][x0],image_hex[y1][x1],image_hex[y2][x2],(short)255,y2,y1,y0,x2,x1,x0,image_hex,i
mage_float,&erreur);
        else
            z_tmp = -1.0;

/*
printf("direction+1(deuxieme): %f \n",z_tmp);
*/

        if(z_tmp >= 0.0)
            {
            if(image_float[y2][x2] >= 0.0)
                image_float[y2][x2] = (image_float[y2][x2] + z_tmp) / 2.0;
            else
                image_float[y2][x2] = z_tmp;
            }

/*

        if(convexite[y2][x2] != conv_val)
            {
            conv_val = convexite[y0][x0] + convexite[y1][x1];
            move(direction+3,&y1,&x1);
            move(direction-2,&y1,&x1);
            conv_val += convexite[y1][x1];
            if(conv_val > 0)
{
putchar('s');
                convexite[y2][x2] = 1;
}
            else
{
putchar(':');
                convexite[y2][x2] = -1;
}
```

```
            }
*/
        if(image_float[y2][x2] >= 0.0)
            {
            z_cumulatif += (image_float[y2][x2] - z_precedent);
                z_precedent = image_float[y2][x2];
            }
        y2 = y1 = y0;
        x2 = x1 = x0;
        move(direction,&y2,&x2);
        move(direction-2,&y1,&x1);
        conv_val = convexite[y0][x0] + convexite[y1][x1] +convexite[y2][x2];
        y2 = y0;
        x2 = x0;
        move(direction-1,&y2,&x2);
        if(conv_val > 0)
            convexite[y2][x2] = 1;
        else
            convexite[y2][x2] = -1;
/*
putchar(')');
*/
        }
    z_cumulatif -= z_precedent;
/*
putchar('\n');
printf("r = %2d prof = %f variation = %f\n",r,image_float[y2][x2],z_cumulatif);
*/
    move(direction-2,&y0,&x0);
    }
}
```

ANNEXE H

Test Fonctionnel du Circuit à Miroir de Courant

Réalisé par Stéphane Dallaire

Été 1991

H.1 Vérification du LVMIR: Description du circuit

Le LVMIR est un circuit intégré essentiellement analogique qui permet d'effectuer des sommes de courant pondérées. Il a été réalisé en technologie cmos3dlm. La description des plots de contact pour ce circuit est présentée à la Figure H.1.

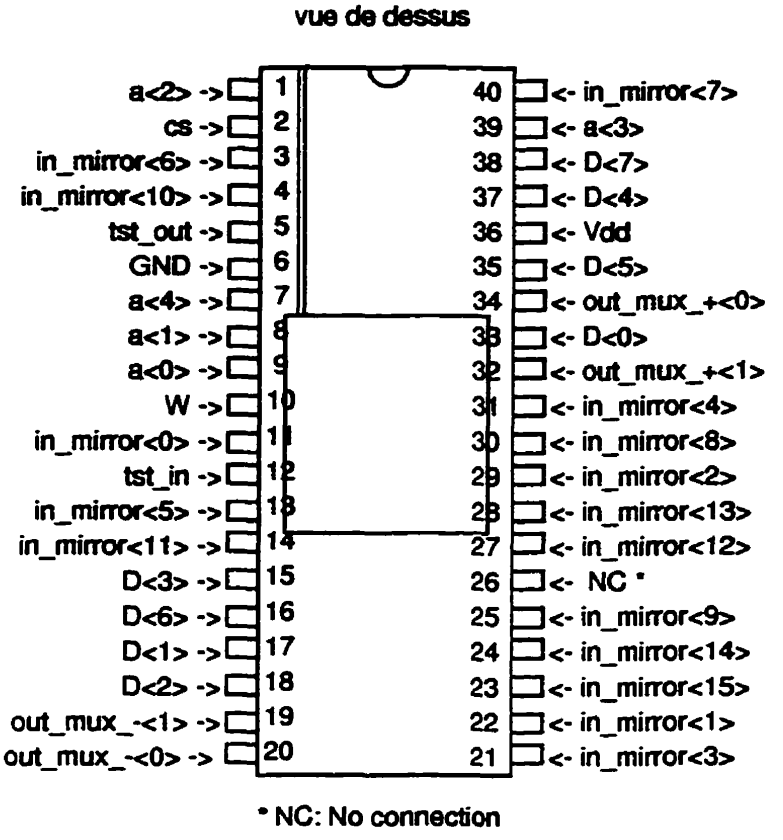
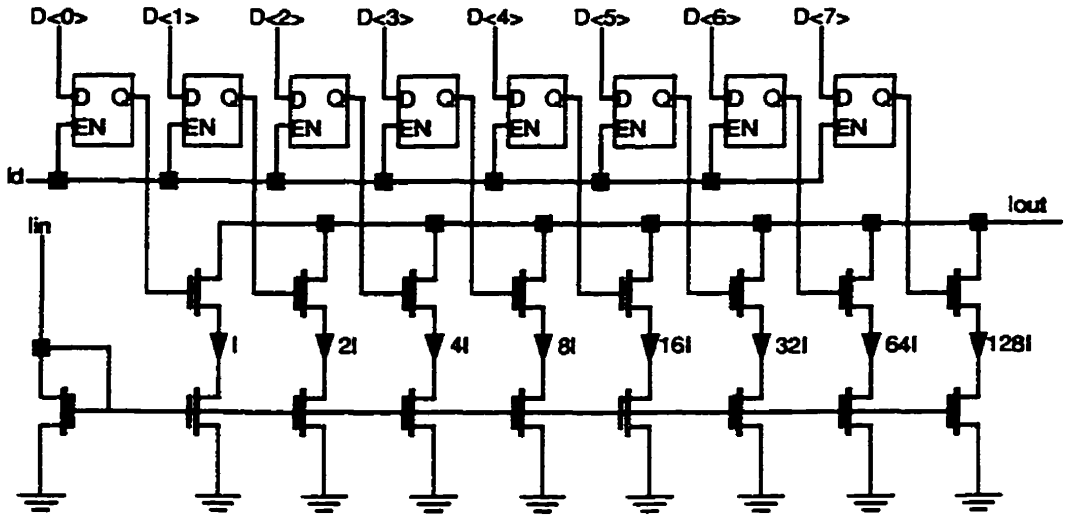


Figure H.1 Description des plots de contacts du LVMIR.

Chacune des entrées "in_mirror" et "tst_in" peut être adressée à l'aide des 5 bits d'adresse "a<4:0>" pour y programmer leur poids respectif. Le poids ou la pondération est programmé à l'aide des bits "D<7:0>". Ce système est basé sur le principe du miroir de courant un peu comme celui de la Figure H.2.



Iout = I(128 D<7> + 64 D<6> + 32 D<5> + 16 D<4> + 8 D<3> + 4 D<2> + 2 D<1> + D<0>)

Figure H.2 *Circuit miroir de courant.*

Ces entrées pondérées, à l'exception de "tst_in", sont connectées à 2 multiplexers analogiques, un pour le "byte" le plus significatif (MSB) c'est-à-dire pour les entrées "in_mirror<15:8>" et l'autre pour le "byte" le moins significatif (LSB) c'est-à-dire pour les entrées "in_mirror<7:0>". Ces multiplexers ont 2 sorties chacun ("out_mux_-" et "out_mux_+"), l'une étant le complément de l'autre. Là encore chacun des 2 multiplexers peut être adressé (avec "a<4:0>") pour y programmer, avec les bits "D<7:0>", les courants que l'on veut sommer. Le Tableau H.1 résume quel procédure utiliser pour effectuer la programmation des différents registres de configuration et le Tableau H.2 présente la carte d'adressage de chacun d'eux.

Tableau H.1 Adressage du LVMIR

	CS	W	a<4>	a<3>	a<2>	a<1>	a<0>
ld<18>	1	0	0	0	1	1	0
ld<17>	1	0	0	0	1	0	1
ld<16>	1	0	0	0	1	0	0
ld<15>	1	0	1	1	0	1	1
ld<14>	1	0	1	1	0	1	0
ld<13>	1	0	1	1	0	0	1
ld<12>	1	0	1	1	0	0	0
ld<11>	1	0	1	1	1	1	1
ld<10>	1	0	1	1	1	1	0
ld<9>	1	0	1	1	1	0	1
ld<8>	1	0	1	1	1	0	0
ld<7>	1	0	1	0	0	1	1
ld<6>	1	0	1	0	0	1	0
ld<5>	1	0	1	0	0	0	1
ld<4>	1	0	1	0	0	0	0
ld<3>	1	0	1	0	1	1	1
ld<2>	1	0	1	0	1	1	0
ld<1>	1	0	1	0	1	0	1
ld<0>	1	0	1	0	1	0	0

Tableau H.2 Plage d'adressage des registres de LVMIR

adresse	registre
ld<18>	tst_in
ld<17>	analog_mux<1>
ld<16>	analog_mux<0>
ld<15>	in_mirror<15>
ld<14>	in_mirror<14>
ld<13>	in_mirror<13>
ld<12>	in_mirror<12>
ld<11>	in_mirror<11>
ld<10>	in_mirror<10>
ld<9>	in_mirror<9>
ld<8>	in_mirror<8>
ld<7>	in_mirror<7>
ld<6>	in_mirror<6>
ld<5>	in_mirror<5>
ld<4>	in_mirror<4>
ld<3>	in_mirror<3>
ld<2>	in_mirror<2>
ld<1>	in_mirror<1>
ld<0>	in_mirror<0>

H.2 Procédure de test

Le montage de la Figure H.3 a été utilisé pour la vérification du LVMIR. Celle-ci s'est effectuée en 2 étapes soient la vérification du fonctionnement statique (en DC) et la vérification du fonctionnement dynamique (en AC).

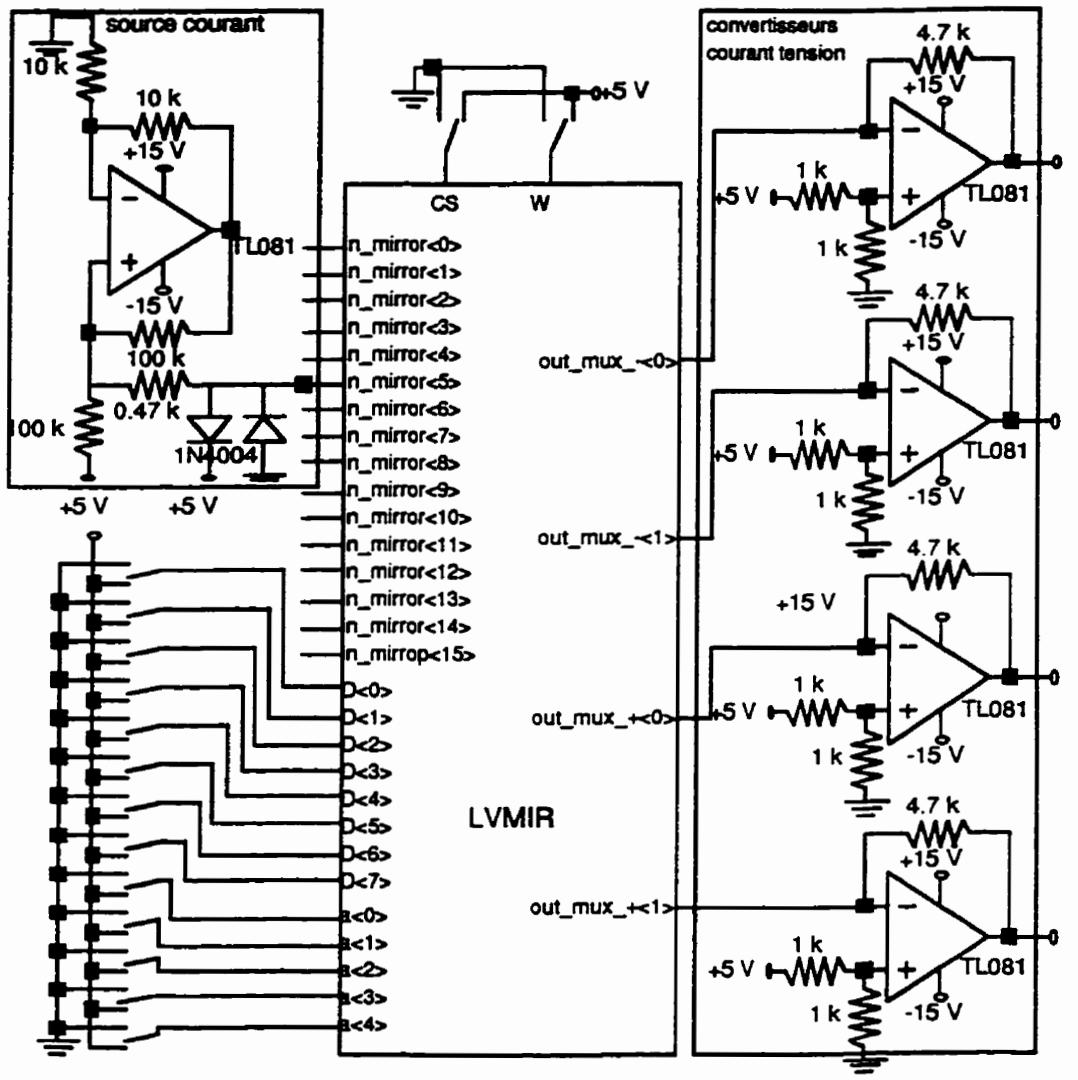


Figure H.3 Montage utilisé pour vérifier le LVMIR

H.2.1 Vérification du fonctionnement statique:

Cette partie des tests a consisté à vérifier la fonctionnalité de chacune des entrées "in_mirror<15:0>" et "tst_in", c'est-à-dire l'adressage de chacune d'elles, l'ajustement de la pondération (poids) à l'aide des entrées "D<7:0>", de l'action du signal "write" (w) et du signal "chip select" (CS) et enfin de la configuration (adressage et écriture) des 2 multiplexers analogiques.

À la suite de ces tests, il s'est avéré que seulement 1 circuit sur les 5 qui ont été vérifiés, était fonctionnelle. Il s'agit du LVMIR #4. Cependant, il y avait aussi le LVMIR #2 qui était fonctionnelle à l'exception d'une seule entrée qui ne fonctionnait pas correctement soit l'entrée "in_mirror<12>".

H.2.2 Vérification du fonctionnement dynamique:

Pour cette partie des tests, nous avons utilisé le circuit qui fonctionnait le mieux selon le test statique c'est-à-dire le LVMIR #4. La première étape a consisté à évaluer la réponse en fréquence du circuit. D'autres paramètres permettant de caractériser la vitesse de fonctionnement du circuit ont aussi été évalués. Il s'agit du temps de montée ("rise time") et du temps de décroissance ("fall time").

H.2.3 Évaluation de la réponse en fréquence:

La procédure adoptée fut la méthode point par point. Un courant sinusoïdal de fréquence et d'amplitude connues a été appliqué à l'une des entrées "in_mirror<15:0>". Puis, au moyen d'un oscilloscope, l'amplitude de la sortie ainsi que son déphasage par rapport à l'entrée ont été mesurés. Cette procédure a été répétée pour plusieurs fréquences différentes de façon à pouvoir tracer la courbe de réponse en fréquence.

Il est à noter que le courant sinusoïdal appliqué à l'entrée comportait un certain "offset". Cet "offset" était absolument nécessaire car le LVMIR n'est alimenté qu'avec une tension positive (+5 V). On ne peut donc lui appliquer un courant négatif, celui-ci ne polariserait pas le transistor à l'entrée (voir Figure H.2).

H.2.4 Remarques sur les deux courbes:

En examinant la courbe d'amplitude en fonction de la fréquence, on constate que le LVMIR se comporte plutôt comme un système du premier ordre (pour des fréquences inférieures à 1.6 MHz). Celui-ci a une fréquence de coupure à -3 dB d'environ 160 kHz.

Cependant, la courbe de phase en fonction de la fréquence n'est pas du tout comme celle d'un système du premier ordre. On peut tenter d'expliquer ce phénomène par la présence d'effets d'ordre supérieur (2^e ou 3^e ordre) qui ne sont pas ou presque pas perceptibles sur la courbe d'amplitude.

On peut remarquer la présence d'une autre fréquence de coupure aux environs de 1.6 MHz. Le LVMIR peut donc être vu comme un système du premier ordre ayant une fréquence de coupure de 160 kHz en cascade avec un autre système du premier ordre ou d'ordre supérieur ayant une fréquence de coupure de 1.6 MHz. Quoi qu'il en soit, cette fréquence de coupure de 1.6 MHz n'est pas d'un très grand intérêt pratique puisque lors de l'utilisation du circuit, il faudra se limiter à des fréquences bien inférieures à cause de la fréquence de coupure de 160 kHz. En fait, il faudra se limiter à des fréquences de l'ordre de 200 kHz.

H.2.5 Évaluation du temps de montée et du temps de décroissance:

Bien que ces paramètres soient surtout employés pour caractériser la vitesse ou rapidité des circuits numériques, ceux-ci ont tout-de-même été mesurés sur le LVMIR même si celui-ci est d'abord et avant tout un circuit analogique.

La vitesse ou rapidité de fonctionnement d'un circuit dépend entre autre du temps de transition entre les états. Le temps de montée ("rise time") et le temps de décroissance ("fall time") sont les temps de transition entre les états. Plus précisément, le temps de montée est le temps pris par la sortie d'un système pour passer de 10% à 90% de sa valeur finale lorsqu'on le soumet à une entrée en échelon. Pour ce qui est du temps de décroissance, celui-ci est le temps pris par la sortie d'un système pour passer de 90% à 10% de sa valeur initiale lorsqu'on le soumet à un essai de lâché.

La procédure utilisée pour effectuer ces mesures est d'appliquer une onde carrée d'une fréquence de 50 kHz à l'entrée (une des entrées "in_mirror<15:0>) et de mesurer les temps de montée et de décroissance de la sortie tels que définis au paragraphe précédent. Ces mesures ont été effectuées à l'aide de l'oscilloscope numérique de l'analyseur logique HP 16500A. On peut observer au Tableau H.3 les valeurs mesurées du temps de montée et du temps de décroissance de la sortie du circuit à mirroir de courrant.

Tableau H.3 Analyse transitoire de circuit à miroir de courant

Temps de monté:	1.66 µs
Temps de descente:	2.16 µs

ANNEXE I

Simulation Logicielle de l'Opérateur de Convolution

I.1 Fonction de conversion d'une image cartésienne en topologie hexagonale

```
#include <stdio.h>

/* image_cart est de nb_ligne lignes par nb_col colonnes */
/* image_hex est de (8*nb_ligne)/7+1 lignes par nb_col colonnes */


cart2hex(image_cart,nb_ligne,nb_col,image_hex)
short image_cart[256][256];
short nb_ligne;
short nb_col;
short image_hex[((256+2)*8)/7][256+2];

{
short nb_ligne_hex;
short i_hex,j_hex;
short i_cart,j_cart;
short hex_y;
short val_inf,val_sup;
```

```
nb_ligne_hex = (nb_ligne<<3)/7 + 1;

for(i_hex=0;i_hex<nb_ligne;i_hex++)
    {
    for(j_hex=0;j_hex<nb_col;j_hex++)
        {
        hex_y = i_hex;
        j_cart = j_hex;
        i_cart = hex_y;
        if(i_hex&1)
            image_hex[i_hex][j_hex] = (image_cart[i_cart][j_cart] +
image_cart[i_cart][j_cart-1]) >> 1;
        else
            image_hex[i_hex][j_hex] = image_cart[i_cart][j_cart];
        }
    }
}
```

I.2 Fonction qui effectue le filtrage d'une image en topologie hexagonale

```
filter(image_in,nb_ligne_hex,nb_col_hex,image_out,coeff,R,max_value,min_value)

short image_in[((256+2)*8)/7][256+2];
short nb_ligne_hex;
short nb_col_hex;
float image_out[((256+2)*8)/7][256+2];
float *coeff;/* tableau des coefficients du filtre de Marr */
short R;
float *max_value;
float *min_value;

{
short i,j,r;
float value;

*max_value = -9.0e20;
*min_value = 9.0e20;
for(i=0;i<nb_ligne_hex-1;i++)
    {
    for(j=0;j<nb_col_hex-1;j++)
        {
        if(i < R || i > nb_ligne_hex-R-1 || j < R || j > nb_col_hex-R-1)
            {
            image_out[i][j] = 0.0;
            continue;
            }
        value = coeff[0]*image_in[i][j];
        if(i&1)
            for(r=1;r<R;r++)
                value += coeff[r]*(image_in[i-r][j+(r+1)/2] +
```

```
                    image_in[i+r][j+(r+1)/2] + image_in[i-r][j-r/2] +
                    image_in[i+r][j-r/2] + image_in[i][j-r] +
                    image_in[i][j+r]);
        else
            for(r=1;r<R;r++)
                value += coeff[r]*(image_in[i-r][j-(r+1)/2] +
                    image_in[i+r][j-(r+1)/2] + image_in[i-r][j+r/2] +
                    image_in[i+r][j+r/2] + image_in[i][j-r] +
                    image_in[i][j+r]);
        if(value > *max_value) *max_value = value;
        if(value < *min_value) *min_value = value;
        image_out[i][j] = value;
        }
    }
}
```

I.3 Fonction qui effectue le suivi d'arête en topologie hexagonale

```
#include "def.h"



find_0(image_in,nb_ligne_hex,nb_col_hex,image_out,zero_value,hysteresis_value,
    x_depart,y_depart,dir_depart,max_pixel,max_e,instruction,option,max_circle)

float image_in[NB_LIGNE_HEX][NB_COL_HEX];
short nb_ligne_hex;
short nb_col_hex;
short image_out[NB_LIGNE_HEX][NB_COL_HEX];
float zero_value;
float hysteresis_value;
int *x_depart,*y_depart,*dir_depart;
int max_pixel,max_e,instruction,option,max_circle;
{
int n=0,e=0,i,j,r;
int x,y,x_1,y_1,x_2,y_2;
float value;
int trace[2][15];
int etat;
int p2;
int circle=0;
int old_dir,dir;

x=(*x_depart);
y=(*y_depart);
x_1=(*x_depart);
y_1=(*y_depart);
x_2=(*x_depart);
y_2=(*y_depart);
dir=old_dir=(*dir_depart);
while((p2 =
zero_detect(image_in[y][x],image_in[y_1][x_1],image_in[y_2][x_2],zero_value,hysteres
```

```
is_value)) == 0)
    {
    x_2=x_1; y_2=y_1;x_1=x; y_1=y;
    if(move(dir,&y,&x,nb_ligne_hex,nb_col_hex) != 0)
        {
        *x_depart = x;
        *y_depart = y;
        *dir_depart = dir;
        return(n);
        }
    }
if(image_out[y][x] != 0 || image_out[y_1][x_1] != 0)
    {
    *x_depart = x;
    *y_depart = y;
    *dir_depart = dir;
    return(n);
    }
image_out[y][x] = 255;
trace[0][n] = y;
trace[1][n] = x;
image_out[y_1][x_1] = 255;
/*
putchar('\n');
putchar('*');
*/
if(option == 0)
    etat = 9;
else
    etat = 8;
while (1)
    {
    if (n == 0)
        hysteresis_value /= 10;
    switch(etat)
        {
        case 0x1:
            dir -= 1;
            etat = 0x9;
            e++;
            n++;
            break;
        case 0x5:
            dir += 1;
            etat = 0x8;
            e++;
            n++;
            break;
        case 0x8:
            if(p2)
                {
                dir -= 2;
                etat = 0x9;
                n++;
```

```
        e = 0;
        }
    else
        {
        dir += 2;
        etat = 0xA;
        }
    break;
case 0x9:
    if(p2)
        {
        dir += 2;
        etat = 0x8;
        n++;
        e = 0;
        }
    else
        {
        dir -= 2;
        etat = 0xC;
        }
    break;
case 0xA:
    if(p2)
        {
        if (instruction == 0)
            etat = 0x7;
        else
            etat = 0x9;
        dir += 3;
        n++;
        }
    else
        {
        dir += 3;
        etat = 0x1;
        }
    break;
case 0xC:
    if(p2)
        {
        if (instruction == 0)
            etat = 0x3;
        else
            etat = 0x8;
        dir += 3;
        n++;
        }
    else
        {
        dir += 3;
        etat = 0x5;
        }
    break;
```

```
            case 0x3:
                if(p2)
                    {
                    dir -= 1;
                    etat = 0x7;
                    n++;
                    e = 0;
                    }
                else
                    {
                    dir += 3;
                    etat = 0x2;
                    }
                break;
            case 0x7:
                if(p2)
                    {
                    dir += 1;
                    etat = 0x3;
                    n++;
                    e = 0;
                    }
                else
                    {
                    dir += 3;
                    etat = 0x6;
                    }
                break;
            case 0x2:
                dir -= 2;
                etat = 0x8;
                break;
            case 0x6:
                dir += 2;
                etat = 0x9;
                break;
            }
        if (e == max_e || n == max_pixel || _ABS(circle) > 6*max_circle)
            {
            if (n < 4)
                for(i=0;i<4;i++)
                    image_out[trace[0][i]][trace[1][i]] = 0;
            *x_depart = x;
            *y_depart = y;
            *dir_depart = dir;
            return(n);
            }
        if(p2 /*&& (etat == 8 || etat == 9 || etat == 3 || etat == 7)*/)
            {
/*
            if(_ABS(image_in[y][x]) < _ABS(image_in[y_1][x_1]))
                {
*/
/*
```

```
            if(image_out[y][x] != 0)
                {
                *x_depart = x;
                *y_depart = y;
                *dir_depart = dir;
                return(n);
                }
*/
            image_out[y][x] = 255;
            if (n < 4)
                {
                trace[0][n] = y;
                trace[1][n] = x;
                }
/*
            }
        else
            {
*/
/*
            if(image_out[y_1][x_1] != 0)
                {
                *x_depart = x;
                *y_depart = y;
                *dir_depart = dir;
                return(n);
                }
*/
            image_out[y_1][x_1] = 255;
/*
            }
*/
        }
    x_2=x_1; y_2=y_1;x_1=x; y_1=y;
    if(move(dir,&y,&x,nb_ligne_hex,nb_col_hex) != 0)
        {
        *x_depart = x;
        *y_depart = y;
        *dir_depart = dir;
        return(n);
        }
    p2 = zero_detect(image_in[y][x],image_in[y_1][x_1],image_in[y_2][x_2],
                zero_value,hysteresis_value);
/*
printf("x=%3d y=%3d dir=%1d etat=%x %f %f
%f\n",x,y,dir,etat,image_in[y][x],image_in[y_1][x_1],image_in[y_2][x_2]);
*/
    circle += (dir - old_dir);
    if (dir > 6)
        dir-=6;
    if (dir < -5)
        dir += 6;
    old_dir = dir;
    }
```

```
}



zero_detect(image_0,image_1,image_2,zero_value,hysteresis_value)
float image_0,image_1,image_2;
float zero_value;
float hysteresis_value;
{
int val0,val1;

if ( (_ABS(image_0-zero_value) >= hysteresis_value &&
    _ABS(image_1-zero_value) >= hysteresis_value ||
    _ABS(image_0-zero_value) >= hysteresis_value &&
    _ABS(image_2-zero_value) >= hysteresis_value ||
    _ABS(image_1-zero_value) >= hysteresis_value &&
    _ABS(image_2-zero_value) >= hysteresis_value)
    && (image_0-zero_value)*(image_1-zero_value) < 0.0)
    return(255);
else
    return(0);
}
```

I.4 Fonction qui déplace le pixel d'intérêt en topologie hexagonale

```
#include <math.h>
#define NB_LIGNE_HEX ((256+2)*8)/7
#define NB_COL_HEX 256+2



/*************************************************************************/
/* move dans la direction 1 a 5 */
/* */
/* 1 = a droite */
/* 2 = en haut a droite */
/* 3 = en haut a gauche */
/* 4 = a gauche */
/* 5 = en bas a gauche */
/* 6 = en bas a droite */
/* */
/*************************************************************************/

move(direction,i_hex,j_hex,nb_ligne_hex,nb_col_hex)
int direction;
int *i_hex,*j_hex;
int nb_ligne_hex,nb_col_hex;
{
int i,j;
```

```
i = (*i_hex);
j = (*j_hex);
while(direction < 0)
    direction+=6;
switch(direction%6)
    {
    case 1:(*j_hex)++;
        break;
    case 2:if(*i_hex & 1)
            (*j_hex)++;
        (*i_hex)--;
        break;
    case 3:if(!(*i_hex & 1))
            (*j_hex)--;
        (*i_hex)--;
        break;
    case 4:(*j_hex)--;
        break;
    case 5:if(!(*i_hex & 1))
            (*j_hex)--;
        (*i_hex)++;
        break;
    case 0:if(*i_hex & 1)
            (*j_hex)++;
        (*i_hex)++;
        break;
    }
if (*i_hex == -1)
    {
    *i_hex = i;
    *j_hex = j;
    return(-1);
    }
if (*j_hex == -1)
    {
    *i_hex = i;
    *j_hex = j;
    return(-1);
    }
if (*i_hex == nb_ligne_hex)
    {
    *i_hex = i;
    *j_hex = j;
    return(-1);
    }
if (*j_hex == nb_col_hex)
    {
    *i_hex = i;
    *j_hex = j;
    return(-1);
    }
return(0);
}
```

I.5 Fonction qui convertit l'une image résultante en nombre entier

```
#include <stdio.h>

/* image_cart est de nb_ligne lignes par nb_col colonnes */
/* image_hex est de (8*nb_ligne)/7+1 lignes par nb_col colonnes */



hex2cart(image_cart,nb_ligne,nb_col,image_hex)
short image_cart[512][512];
int nb_ligne;
int nb_col;
short image_hex[((256+2)*8)/7][256+2];

{
int nb_ligne_hex;
int i_hex,j_hex;
int i_cart,j_cart;
int cart_y;
int val_inf,val_sup;


nb_ligne_hex = (nb_ligne+2<<3)/7;

for(i_hex=0;i_hex<nb_ligne_hex;i_hex++)
    for(j_hex=0;j_hex<nb_col+2;j_hex++)
        {
        if((i_hex*7)/4+1 >= 512 ||j_hex*2+(i_hex & 1) >= 512)
            continue;
        if(image_hex[i_hex][j_hex] != 0)
            {
            if((i_hex*7)%4 <= 2)
             image_cart[(i_hex*7)/4][j_hex*2+(i_hex & 1)] = 255;
            if((i_hex*7)%4 >= 2)
             image_cart[((i_hex*7)/4)+1][j_hex*2+(i_hex & 1)] = 255;
            }
        }

}
```

I.6 Fonction de conversion de la topologie hexagonale à une image cartésienne

```
float2hex(image_hex,nb_ligne_hex,nb_col_hex,image_float,max_value,min_value)
short image_hex[((256+2)*8)/7][256+2];
short nb_ligne_hex;
short nb_col_hex;
float image_float[((256+2)*8)/7][256+2];
float max_value;
float min_value;
{
short i;
```

```
short j;
float abs_max;

if(max_value > 0.0 && min_value < 0.0 && max_value)
    {
    if(max_value > -min_value)
        abs_max = max_value;
    else
        abs_max = -min_value;
    }
else
    {
    if(max_value < 0.0)
        abs_max = -min_value;
    else
        abs_max = max_value;
    }

for(i=0;i<nb_ligne_hex;i++)
    for(j=0;j<nb_col_hex;j++)
        image_hex[i][j] = (short)((image_float[i][j])*256.0/(2*abs_max));
}
```

I.7 Programme principal de simulation de l'opérateur de convolution

```
#include "def.h"
#include <stdio.h>
#include <rasterfile.h>
#include <math.h>
/*
double power();
long fact();
*/
#define pi 3.14159

/* simule <image_in> <image_out> */


short image_cart[256][256];
short image_hex[NB_LIGNE_HEX][NB_COL_HEX];
float image_float[NB_LIGNE_HEX][NB_COL_HEX];


main(argc,argv)
int argc;
char **argv;
{
FILE *fin;
FILE *fout;
int R;
float sigma;
int nb_ligne;
```

```
int nb_col;
struct rasterfile header;
short i,j;
char color_map[3*256];
float coeff[16];
float min_value;
float max_value;
double tmp;
int x,y,dir_depart,x_depart,y_depart;
float zero_value,hysteresis_value;
int N,E,max_circle;


if(argc != 3)
    {
    printf("usage: simule <f_name_in> <f_name_out>\n");
    exit(1);
    }
fin = fopen(argv[1],"r");
printf("quel est le rayon maximal du filtre en pixel? :");
scanf("%d",&R);
printf("quel est la valeur de sigma? :");
scanf("%f",&sigma);
fread(&header,sizeof(struct rasterfile),1,fin);
fread(color_map,sizeof(char),3*256,fin);

for(i=0,j=R;i < R;i++)
    {
    tmp = (double)(i*i) / (2*sigma*sigma);
    coeff[i] = (float)(-1.0/(pi*sigma*sigma*sigma*sigma) * (1.0 - tmp) * exp(-tmp));
/*
*/
    if(i>0)
        coeff[i] *= (float)i;
/*
*/
    if(_ABS(coeff[i]) < 0.000001)
        j = i;
    printf("coeff[%2d] = %f\n",i,coeff[i]);
    }
tmp = coeff[0];
for(i=1;i<R;i++)
    tmp += (6.0*coeff[i]);
printf("valeur moyenne = %f\n",tmp);

for(i=0;i<j;i++)
    coeff[i]-=(tmp/((j*6)-5));

tmp = coeff[0];
for(i=1;i<R;i++)
    tmp += (6.0*coeff[i]);
printf("valeur moyenne = %f\n",tmp);
```

```
nb_ligne = header.ras_height;
nb_col = header.ras_width;
for(i=0;i<nb_ligne;i++)
    for(j=0;j<nb_col;j++)
        image_cart[i][j] = (short)getc(fin);

printf("conversion cartesien to hexagonale\n");
cart2hex(image_cart,nb_ligne,nb_col,image_hex);

printf("filtre de Marr\n");
filter(image_hex,NB_LIGNE_HEX,NB_COL_HEX,image_float,coeff,R,&max_value,&
min_value);
printf("max_value = %15f ; min_value = %15f\n",max_value,min_value);

for(i=0;i<NB_LIGNE_HEX;i++)
    for(j=0;j<NB_COL_HEX;j++)
        image_hex[i][j] = (short)0;
printf("Find_0\n");
x_depart = 128;
y_depart = 128;
/*
printf("quel est la direction de depart? :");
scanf("%d",&dir_depart);
*/
printf("quel est le zero et l'hysteresis? :");
scanf("%f%f",&zero_value,&hysteresis_value);
printf("quel sont les valeurs de N ,E et max_circle? :");
scanf("%d%d%d",&N,&E,&max_circle);
    /*
while (1)
    {
    printf("quel est le point de depart? (x_depart,y_depart):");
    scanf("%d%d",&x_depart,&y_depart);
    if(x_depart < 0 || y_depart < 0) break;
    */
for(x=10;x< 250; x+=10)
for(y=10;y< 8*250/7; y+=10)
    for(i=1;i<7;i++)
        {
        dir_depart = i;
        x_depart = x;
        y_depart = y;
        find_0(image_float,NB_LIGNE_HEX,NB_COL_HEX,image_hex,zero_value,
            hysteresis_value,&x_depart,
            &y_depart,&dir_depart,N,E,1,0,max_circle);
        find_0(image_float,NB_LIGNE_HEX,NB_COL_HEX,image_hex,zero_value,
            hysteresis_value,&x_depart,
            &y_depart,&dir_depart,N,E,1,1,max_circle);
        }
/*
    }
*/
fout = fopen(argv[2],"w");
```

```
fwrite(&header,sizeof(struct rasterfile),1,fout);
fwrite(color_map,sizeof(char),3*256,fout);



/*
printf("conversion point flotant to entier\n");
float2hex(image_hex,((nb_ligne+2)*8)/7,nb_col+2,image_float,max_value,min_value);
*/

printf("conversion hexagonale to cartesien\n");
hex2cart(image_cart,nb_ligne,nb_col,image_hex);
for(i=0;i<256;i++)
    for(j=0;j<256;j++)
        putc(image_cart[i][j],fout);
return;
}
```

ANNEXE J

Equations logiques du décodeur pour le pointeur de mémoire MAR

Le décodeur utilisé pour contrôler les deux compteurs synchrones utilise les signaux suivants. On les retrouve à la Figure 4.20. Le code de déplacement *DIR<2:0>* (d_2, d_1, d_0), la parité de la ligne courante *PARITE_XY* (P) de même que deux variables de configuration associées à la polarité des axes du référentiel image *X_MEM_DIR_REF* (X_R) et *Y_MEM_DIR_REF* (Y_R), génèrent les signaux de commandes des deux compteurs. Les équations logiques (J-1) à (J-4) résument la logique combinatoire comprise dans ce décodeur:

$$EN_X = d_2\overline{d_1}\,\overline{d_0} + \overline{d_2}\,\overline{d_1}d_0 + \overline{d_2}d_1\overline{d_0}\,\overline{P} + \overline{d_2}d_1d_0P + d_2\overline{d_1}d_0P + d_2d_1\overline{d_0}\,\overline{P} \qquad \text{(J-1)}$$

$$\overline{UP_x} = d_2\overline{d_1}\,\overline{d_0}\,\overline{X_R} + \overline{d_2}d_1d_0\overline{X_R} + d_2\overline{d_1}d_0\overline{X_R} + \overline{d_2}\,\overline{d_1}d_0X_R + \overline{d_2}d_1\overline{d_0}X_R + d_2d_1\overline{d_0}X_R \qquad \text{(J-2)}$$

$$EN_Y = \overline{d_2}d_1d_0 + d_2\overline{d_1}d_0 + \overline{d_2}d_1\overline{d_0} + d_2d_1\overline{d_0} \qquad \text{(J-3)}$$

$$\overline{UP_Y} = Y_R\overline{d_2} + \overline{Y_R}d_2 \qquad \text{(J-4)}$$

ANNEXE K

Plans mécaniques des composantes de la caméra MAR

On présente ici les diverses composantes mécaniques requises pour le support et l'assemblage de la caméra MAR. On y retrouve le plan du support de lentille, le cylindre d'assemblage, le couvert arrière et celui de la forme des différents circuits imprimés.

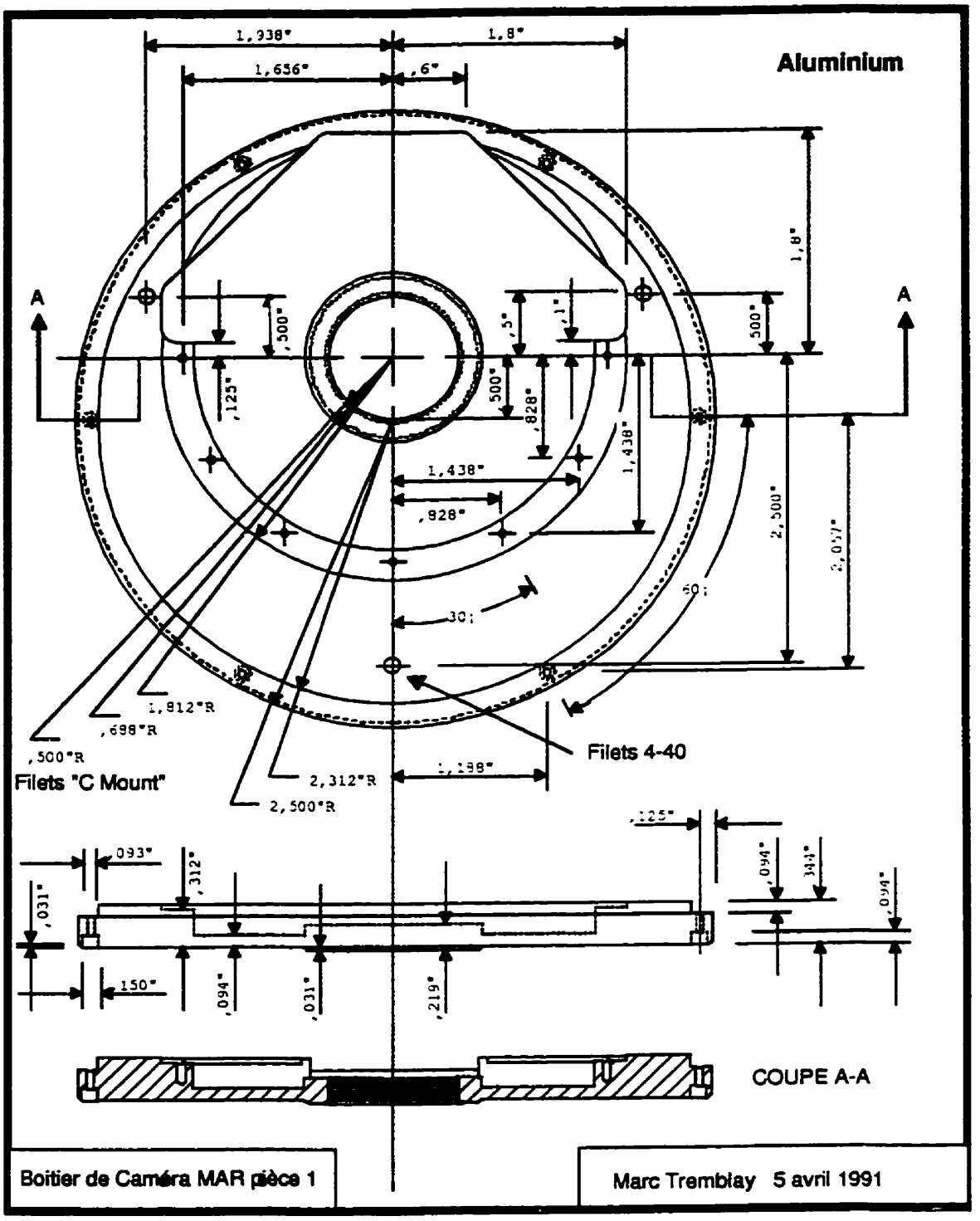


Figure K.1 Pièce principale boîtier de la caméra MAR. Elle permet de fixer la lentille, l'obturateur optique, et les circuits imprimés.

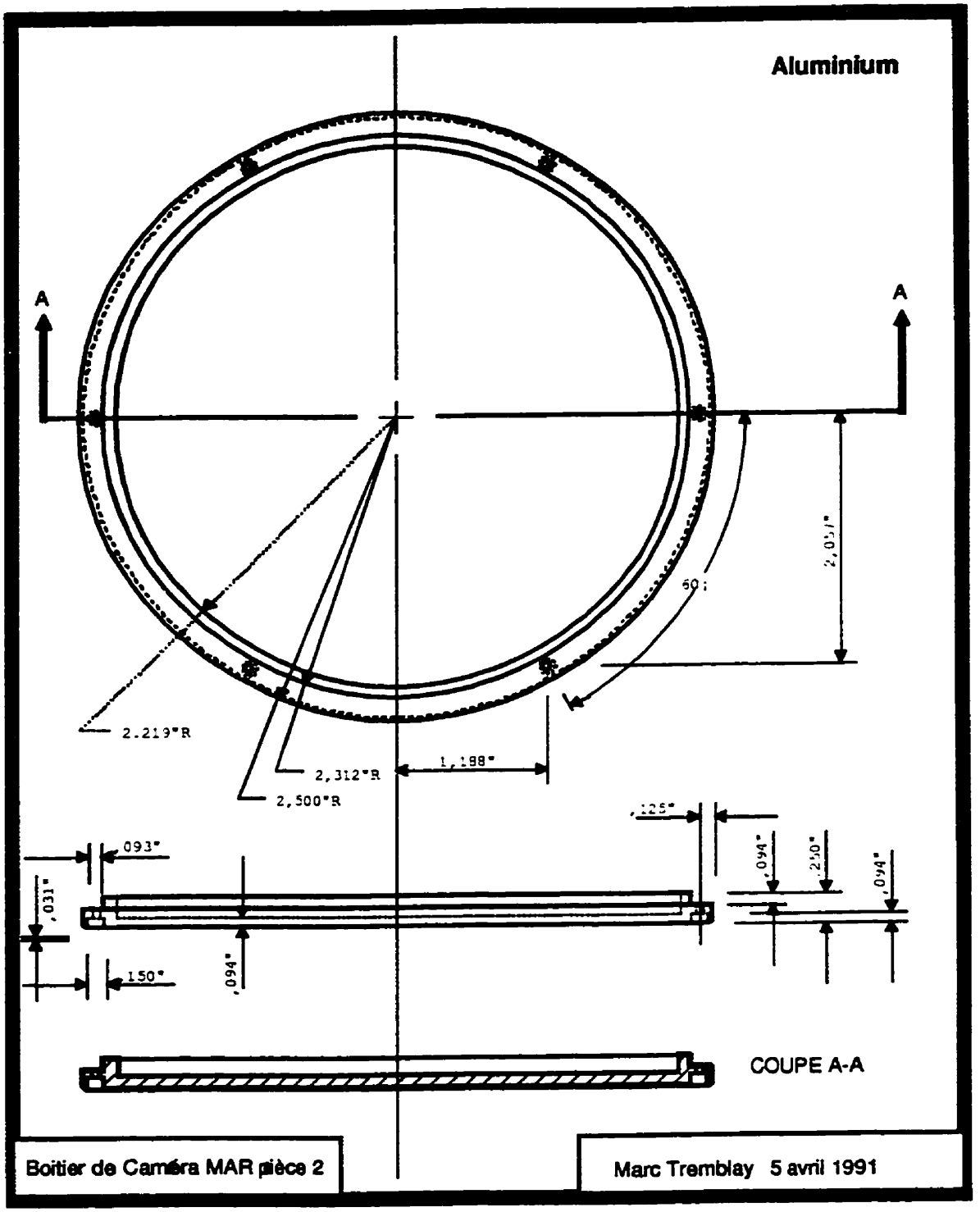


Figure K.2 Plan du couvert arrière de la caméra. On fixe sur cette pièce les éventuels connecteurs de communication et les fiches d'alimentations DC.

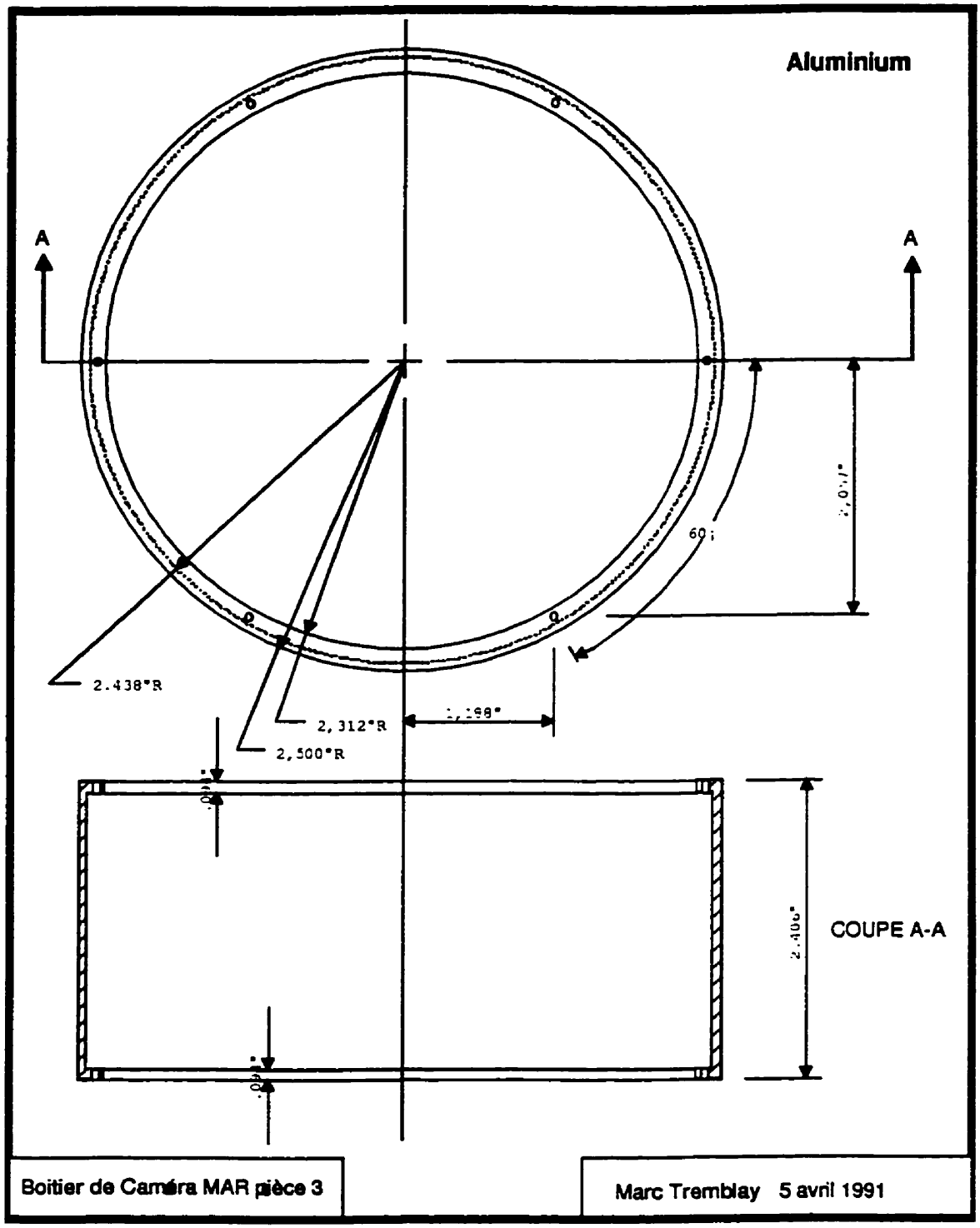


Figure K.3 *Plan du cylindre d'assemblage pour le boîtier de la caméra.*

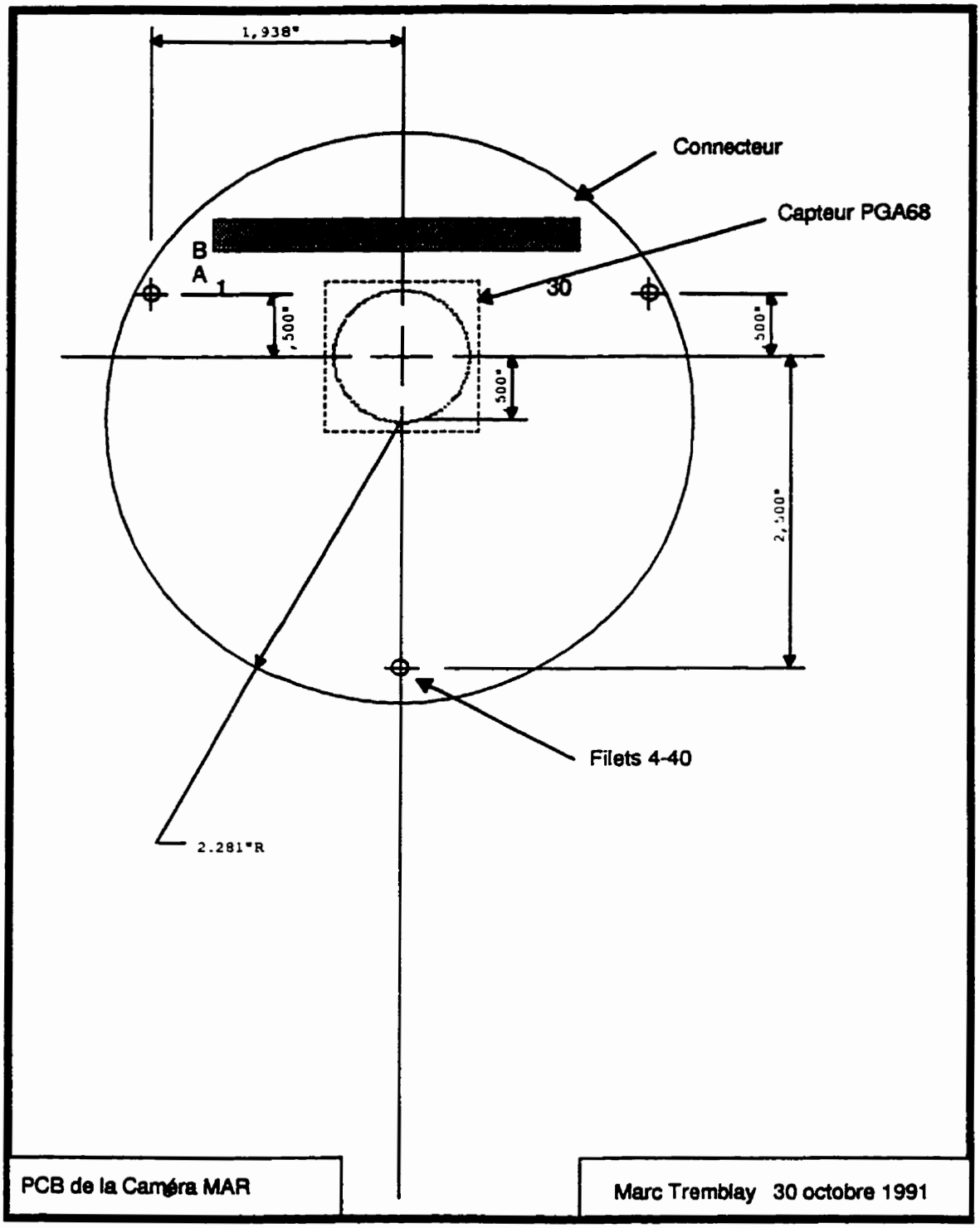


Figure K.4 Plan de la forme des circuits imprimés de la caméra. On représente l'emplacement exact des trous pour l'assemblage, celui du connecteur et la position du centre optique.

ANNEXE L

Plans Électriques du Circuit de Conditionnement pour le Capteur

On présente ici les plans électriques du circuit de conditionnement des signaux analogiques en provenance du capteur. Ils comprennent plusieurs convertisseurs courant/ tension de même que le circuit de commande pour la compensation du courant de fuite. On présente également la définition du connecteur.

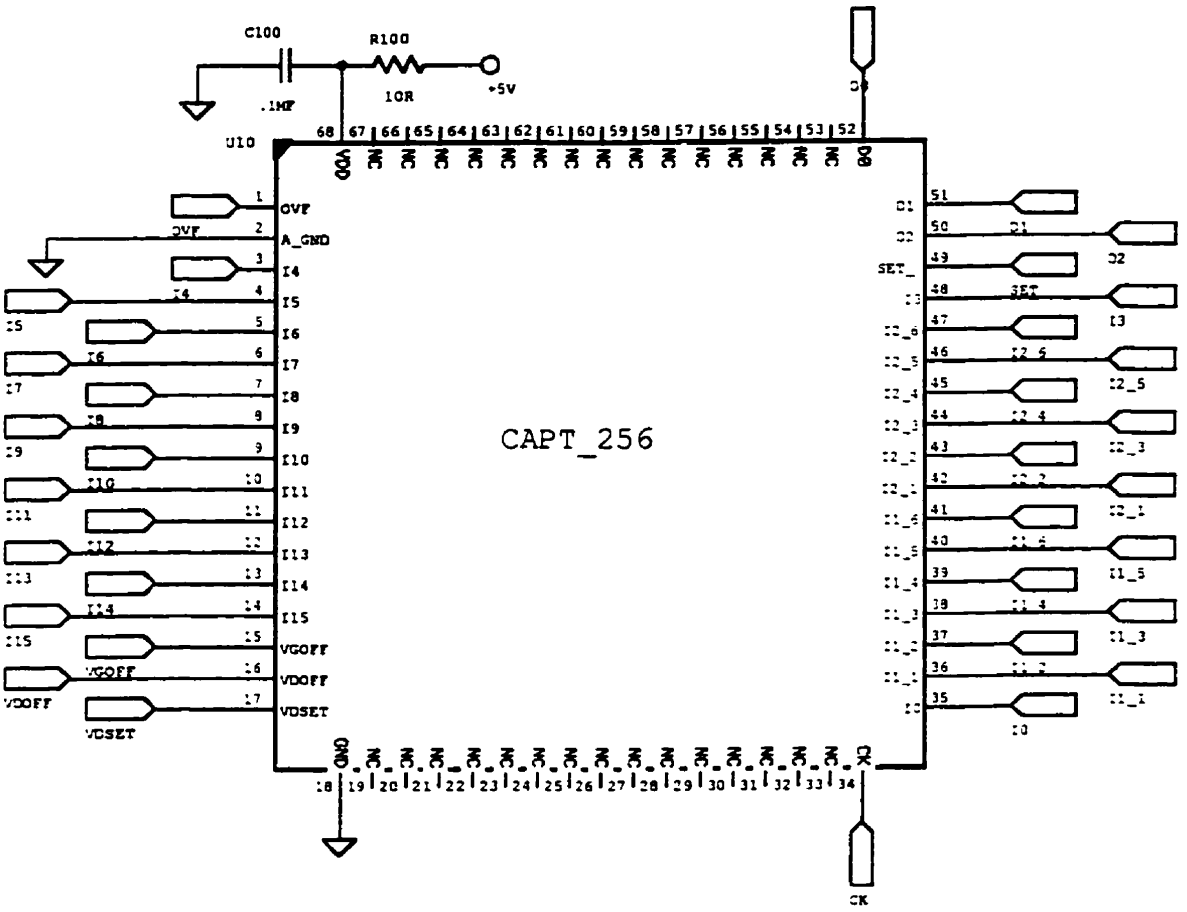


***Figure L.1** Plan de connexion pour le capteur MAR. Lors de la réalisation du circuit imprimé, le capteur est placé du côté opposé aux autres composantes.*

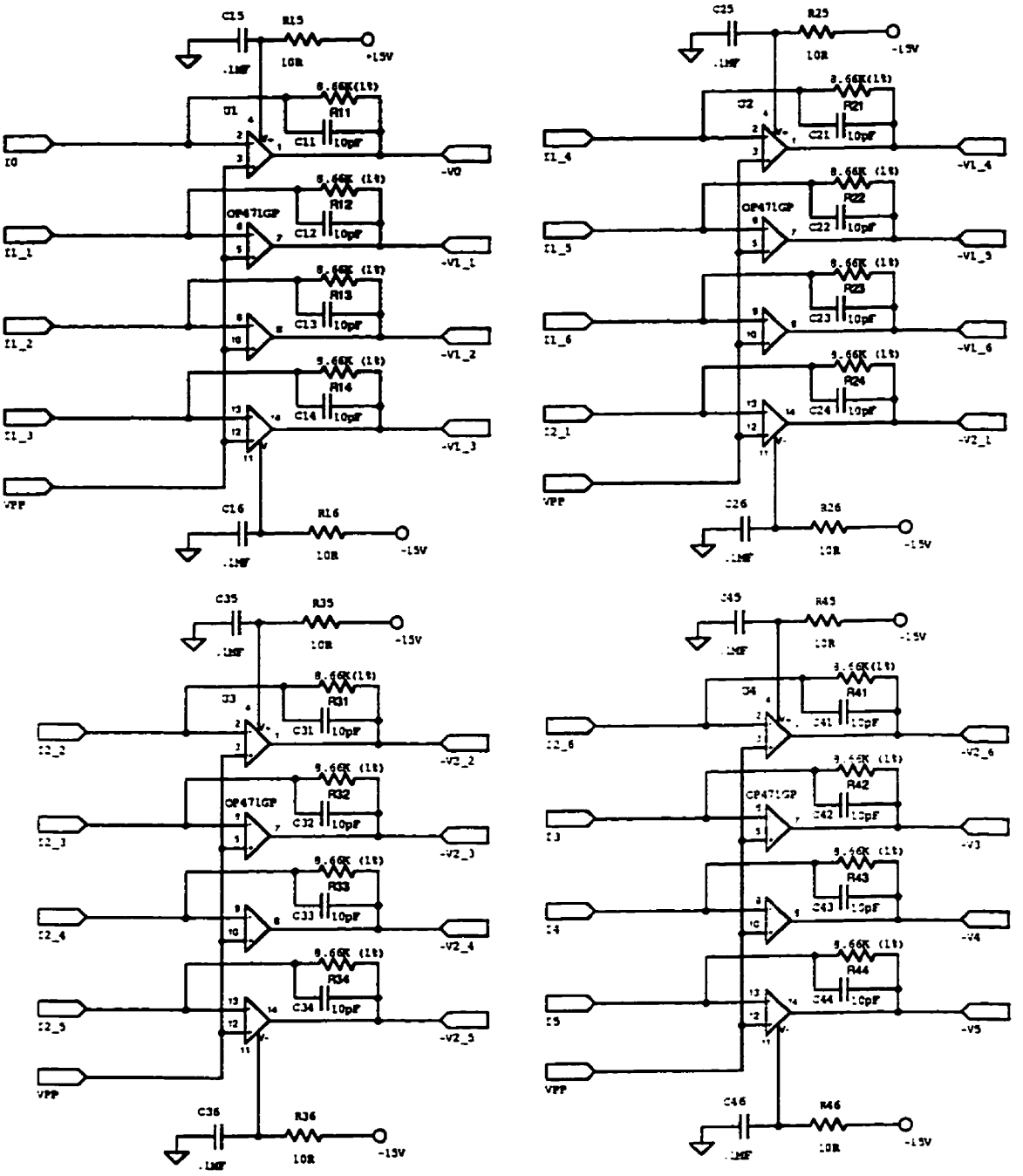


***Figure L.2** Plans des 16 premiers convertisseurs courant/tension pour les signaux analogiques provenant du capteur MAR.*

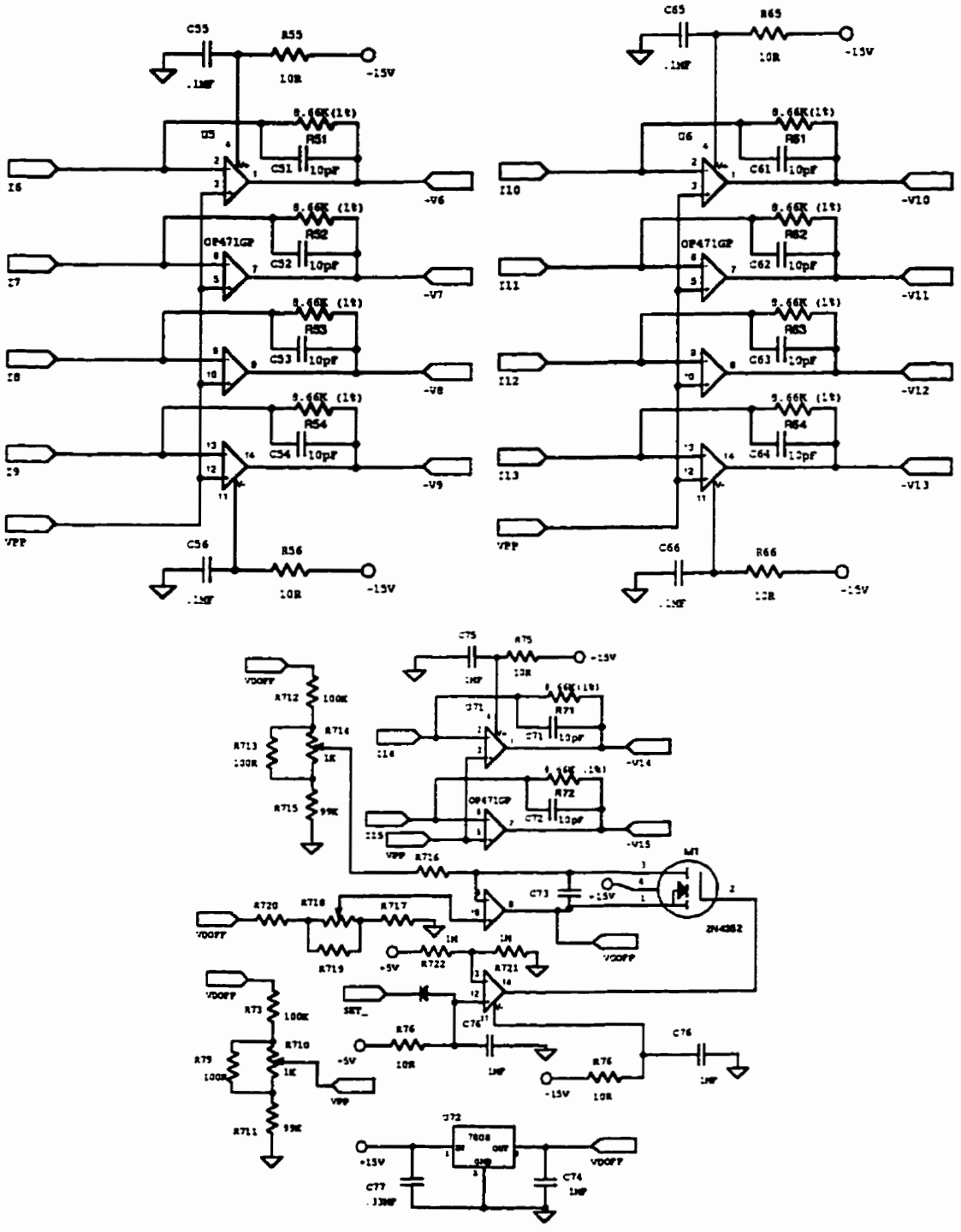


Figure L.3 *Plans des 10 autres convertisseurs courant/tension pour les signaux analogiques provenant du capteur MAR. Le PLAN 7 comprend également le circuit qui commande la compensation du courant de fuite.*

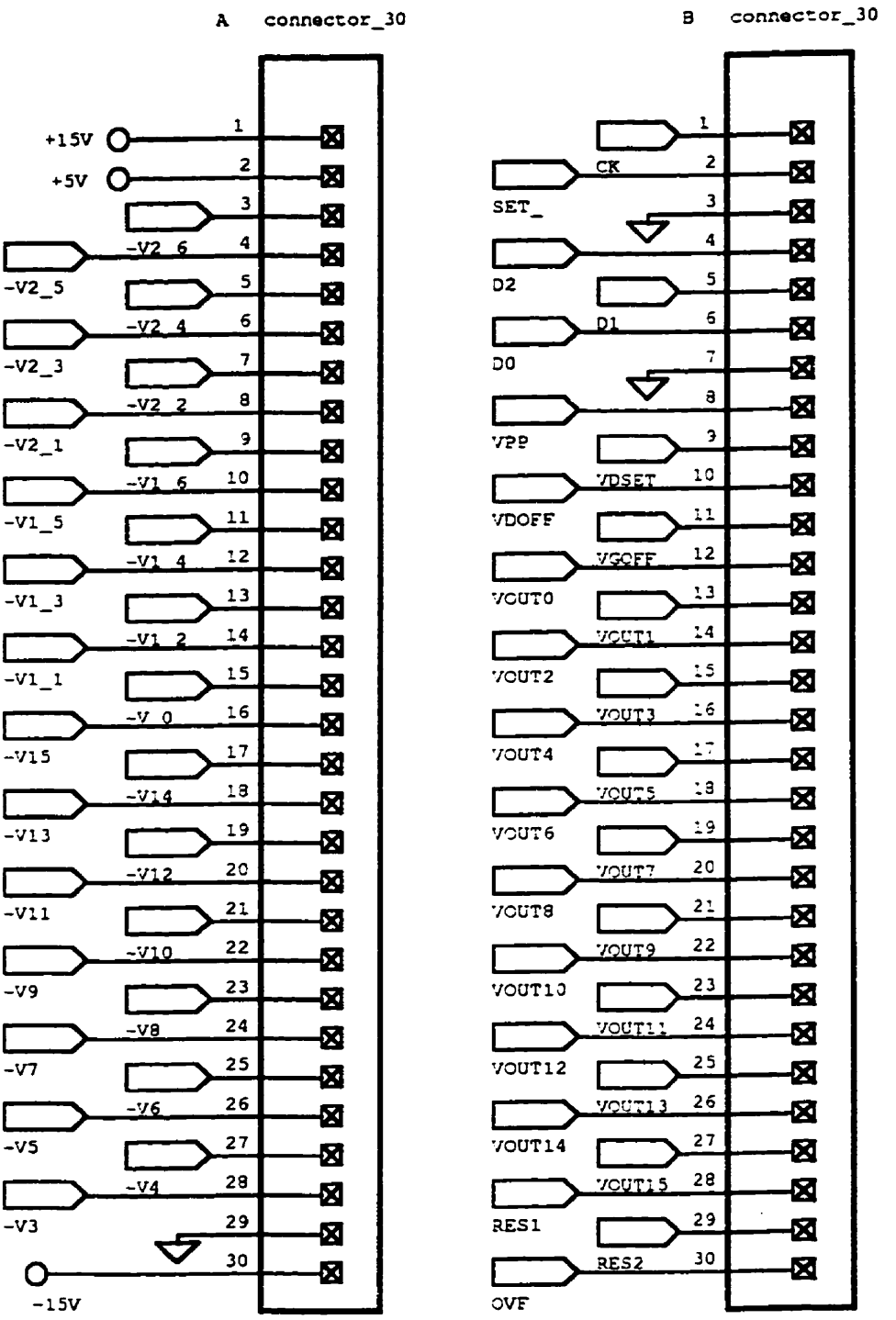
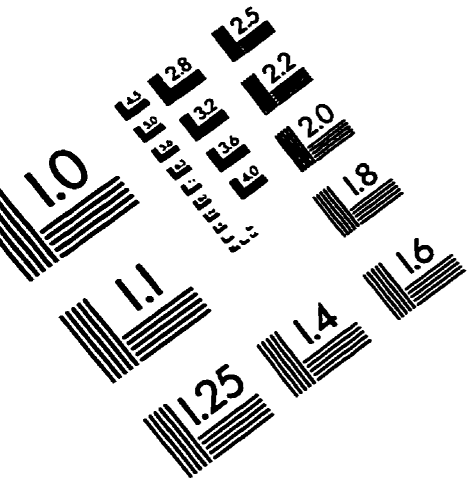
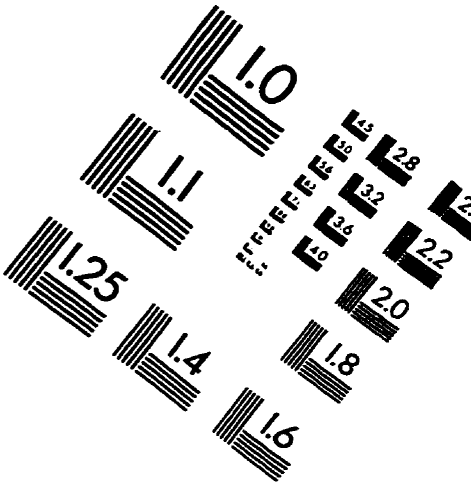
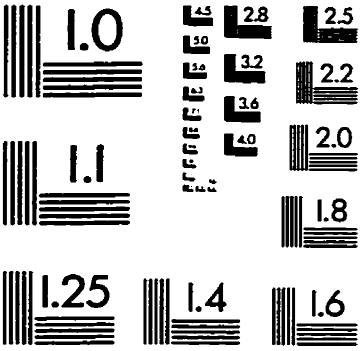


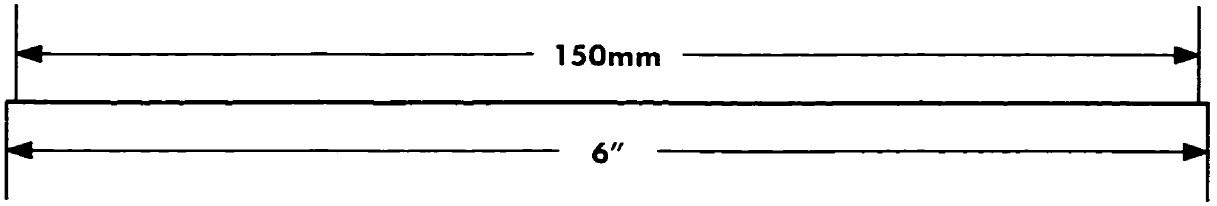
***Figure L.4** Définition du connecteur qui relie l'ensemble des circuit imprimés requis pour la caméra.*

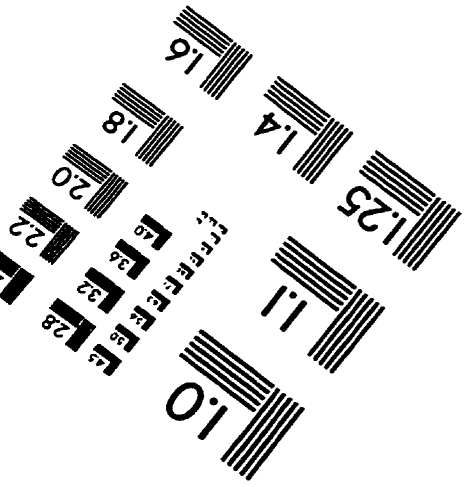
IMAGE EVALUATION TEST TARGET (QA-3)

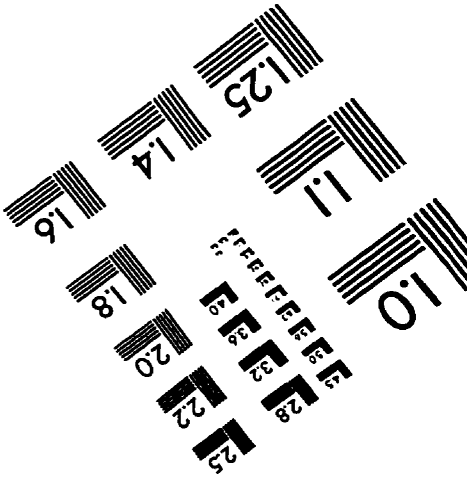












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989