# Étude d'une stratégie d'autotest intégré pour le compilateur de silicium SYCO

Kholdoun Torki

# THESE

présentée par

**Kholdoun TORKI**

pour obtenir le titre de **DOCTEUR**

**de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**
(arrêté ministériel du 23 novembre 1988)

(Spécialité : Microélectronique)

==========

# ETUDE D'UNE STRATEGIE D'AUTOTEST INTEGRE POUR LE COMPILATEUR DE SILICIUM SYCO

==========

Date de soutenance : 12 juillet 1990

| | | | |
|---|---|---|---|
| Composition du Jury: | M. | Pierre GENTIL | Président |
| | MM. | Mohamed BENAHMED | |
| | | Bernard COURTOIS | |
| | | Joan FIGUERAS | Rapporteur |
| | | Christian LANDRAULT | Rapporteur |
| | | Jean-Pierre MOREAU | |
| | | Mihaïl NICOLAIDIS | |

Thèse préparée au sein du Laboratoire TIM3/INPG.

# INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

46 avenue Felix Viallet
38031 GRENOBLE cedex

Tél. : 76.57.45.00

**Année universitaire 1989**

**Président de l'Institut :**
Monsieur Georges LESPINARD

## Professeurs des Universités

| | | | |
|---|---|---|---|
| BARIBAUD Michel | ENSERG | JAUSSAUD Pierre | ENSIEG |
| BARRAUD Alain | ENSIEG | JOST Rémy | ENSPG |
| BAUDELET Bernard | ENSPG | JOUBERT Jean-Claude | ENSPG |
| BEAUFILS Jean-Pierre | INPG | JOURDAIN Geneviève | ENSIEG |
| BLIMAN Samuel | ENSERG | LACOUME Jean-Louis | ENSIEG |
| BOIS Philippe | ENSHMG | LADET Pierre | ENSIEG |
| BONNETAIN Lucien | ENSEEG | LESIEUR Marcel | ENSHMG |
| BONNET Guy | ENSPG | LESPINARD Georges | ENSHMG |
| BRISSONNEAU Pierre | ENSIEG | LONGEQUEUE Jean-Pierre | ENSPG |
| BRUNET Yves | IUFA | LORET Benjamin | ENSHMG |
| CAILLERIE Denis | ENSHMG | LOUCHET François | ENSEEG |
| CAVAIGNAC Jean-François | ENSPG | LUCAZEAU Guy | ENSEEG |
| CHARTIER Germain | ENSPG | MASSE Philippe | ENSIEG |
| CHENEVIER Pierre | ENSERG | MASSELOT Christian | ENSIEG |
| CHERADAME Hervé | UFR PGP | MAZARE Guy | ENSIMAG |
| CHERUY Arlette | ENSIEG | MOHR Roger | ENSIMAG |
| CHOVET Alain | ENSERG | MOREAU René | ENSHMG |
| COHEN Joseph | ENSERG | MORET Roger | ENSIEG |
| COLINET Catherine | ENSEEG | MOSSIERE Jacques | ENSIMAG |
| CORNUT Bruno | ENSIEG | OBLED Charles | ENSHMG |
| COULOMB Jean-Louis | ENSIEG | OZIL Patrick | ENSEEG |
| COUMES André | ENSERG | PA ULEAU Yves | ENSEEG |
| CROWLEY James | ENSIMAG | PERRET Robert | ENSIEG |
| DARVE Félix | ENSHMG | PIAU Jean-Michel | ENSHMG |
| DELLA-DORA Jean | ENSIMAG | PIC Etienne | ENSERG |
| DEPEY Maurice | ENSERG | PLATEAU Brigitte | ENSIMAG |
| DEPORTES Jacques | ENSPG | POUPOT Christian | ENSERG |
| DEROO Daniel | ENSEEG | RAMEAU Jean-Jacques | ENSEEG |
| DESRE Pierre | ENSEEG | REINISCH Raymond | ENSPG |
| DOLMAZON Jean-Marc | ENSERG | RENAUD Maurice | UFR PGP |
| DURAND Francis | ENSEEG | ROBERT André | UFR PGP |
| DURAND Jean-Louis | ENSPG | ROBERT François | ENSIMAG |
| FAUTRELLE Yves | ENSHMG | SABONNADIERE Jean-Claude | ENSIEG |
| FOGGIA Albert | ENSIEG | SAUCIER Gabrièle | ENSIMAG |
| FONLUPTJean | ENSIMAG | SCHLENKER Claire | ENSPG |
| FOULARD Claude | ENSIEG | SCHLENKER Michel | ENSPG |
| GANDINI Alessandro | UFR PGP | SERMET Pierre | ENSERG |
| GAUBERT Claude | ENSPG | SILVY Jacques | UFR PGP |
| GENTIL Pierre | ENSERG | SIRIEYS Pierre | ENSHMG |
| GENTIL Sylviane | ENSIEG | SOHM Jean-Claude | ENSEEG |
| GREVEN Hélène | IUFA | SOLER Jean-Louis | ENSIMAG |
| GUEGUEN Claude | ENSIEG | SOUQUET Jean-Louis | ENSEEG |
| GUERIN Bernard | ENSERG | TROMPETTE Philippe | ENSHMG |
| GUYOT Pierre | ENSEEG | VINCENT Henri | ENSPG |
| IVANES Marcel | ENSIEG | ZADWORNY François | ENSERG |

## Personnes ayant obtenu le diplôme d'HABILITATION A DIRIGER DES RECHERCHES

BECKER Monique
BINDER Zdenek
CHASSERY Jean-Marc
CHOLLET Jean-Pierre
COEY John
COLINET Catherine
COMMAULT Christian
CORNUEJOLS Gérard
COULOMB Jean- Louis
COURNIL M.
DALARD Francis
DANES Florin
DEROO Daniel
DIARD Jean-Paul
DION Jean-Michel
DUGARD Luc
DURAND Madeleine
DURAND Robert
GALERIE Alain
GAUTHIER Jean-Paul
GENTIL Sylviane
GHIBAUDO Gérard
HAMAR Sylvaine
HAMAR Roger
LACHENAL D.
LADET Pierre
LATOMBE Claudine
LE HUY H.
LE GORREC Bernard
MADAR Roland
MEUNIER G.
MULLER Jean
NGUYEN TRONG Bernadette
NIEZ J.J.
PASTUREL Alain
PLA Fernand
ROGNON J.P.
ROUGER Jean
TCHUENTE Maurice
VINCENT Henri
YAVARI A.R.

## Chercheurs du C.N.R.S

### DIRECTEURS DE RECHERCHE CLASSE 0

| | |
|---|---|
| LANDEAU | Ioan |
| NAYROLLES | Bernard |

## Directeurs de recherche 1ère Classe

ANSARA Ibrahim
CARRE René
FRUCHART Robert
HOPFINGER Emile
JORRAND Philippe
KRAKOWIAK Sacha
LEPROVOST Christian
VACHAUD Georges
VERJUS Jean-Pierre

## Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ARMAND Michel
AUDIER Marc
BERNARD Claude
BINDER Gilbert
BONNET Roland
BORNARD Guy
CAILLET Marcel
CALMET Jacques
CHATILLON Chritiant
CLERMONT Jean-Robert
COURTOIS Bernard
DAVID René
DION Jean-Michel
DRIOLE Jean
DURAND Robert
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
GARNIER Marcel
GUELIN Pierre
JOUD Jean-Charles
KAMARINOS Georges
KLEITZ Michel
KOFMAN Walter
LEJEUNE Gérard
MADAR Roland
MERMET Jean
MICHEL Jean-Marie
MEUNIER Jacques
PEUZIN Jean-Claude
PIAU Monique
RENOUARD Dominique
SENATEUR Jean-Pierre
SIFAKIS Joseph
SIMON Jean-Paul
SUERY Michel
TEODOSIU Christian
VAUCLIN Michel
VENNEREAU Pierre
WACK Bernard
YONNET Jean-Paul

## Personnalités agrées à titre permanent à diriger des travaux de recherche
**(décision du conseil scientifique)**

**E.N.S.E.E.G**
HAMMOU Abdelkader
MARTIN-GARIN Régina
SARRAZIN Pierre
SIMON Jean-Paul

**E.N.S.E.R.G**
BOREL Joseph

**E.N.S.I.E.G**
DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

**E.N.S.H.M.G**
ROWE Alain

**E.N.S.I.M.A.G**
COURTIN Jacques

**C.E.N.G**
CADET Jean
COEURE Philippe
DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIEB Maurice
VINCENDON Marc

**Laboratoires extérieurs :**

**C.N.E.T**
DEVINE Rodericq
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

# Situation particulière

## PROFESSEURS D'UNIVERSITE

### DETACHEMENT

| | | | | |
|---|---|---|---|---|
| ENSIMAG | LATOMBE | J..Claude | Détachement | 21/10/1989 |
| ENSHMG | PIERRARD | J.Marie | Détachement | 30/04/1989 |
| ENSIMAG | VEILLON | Gérard | Détachement | 30/09/1990 |
| ENSIMAG | VERJUS | J.Pierre | Détachement | 30/09/1989 |
| ENSPG | BLOCH | Daniel | Recteur à c/ | 21/12/1988 |

### SURNOMBRE

| | | | |
|---|---|---|---|
| INPG | CHIAVERINA | Jean | 30/09/1989 |
| ENSHMG | BOUVARD | Maurice | 30/09/1991 |
| ENSEEG | PARIAUD | J.Charles | 30/09/1991 |

# UNIVERSITE Joseph FOURIER (GRENOBLE I)

**Président de l'Université :**
**M. NEMOZ Alain**

**Année Universitaire 1988 - 1989**

## MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

### PROFESSEURS DE 1ère Classe

| | |
|---|---|
| ADIBA Michel | Informatique |
| ANTOINE Pierre | Géologie I.R.I.G.M. |
| ARNAUD Paul | Chimie Organique |
| ARVIEU Robert | Physique Nucléaire I.S.N. |
| AUBERT Guy | Physique C.N.R.S |
| AURIAULT Jean-Louis | Mécanique |
| AYANT Yves | Physique Approfondie |
| BARBIER Marie-Jeanne | Electrochimie |
| BARJON Robert | Physique Nucléaire ISN |
| BARNOUD Fernand | Biochimie Macromoléculaire Végétale |
| BARRA Jean-René | Statistiques-Mathématiques Appliquées |
| BECKER Pierre | Physique |
| BEGUIN Claude | Chimie Organique |
| BELORISKY Elie | Physique |
| BENZAKEN Claude | Mathématiques Pures |
| BERARD Pierre | Mathématiques Pures |
| BERNARD Alain | Mathématiques Pures |
| BERTRANDIAS Françoise | Mathématiques Pures |
| BERTRANDIAS Jean-Paul | Mathématiques Pures |
| BILLET Jean | Géographie |
| BOELHER Jean-Paul | Mécanique |
| BRAVARD Yves | Géographie |
| CARLIER Georges | Biologie Végétale |
| CASTAING Bernard | Physique |
| CAUQUIS Georges | Chimie Organique |
| CHARDON Michel | Géographie |
| CHIBON Pierre | Biologie Animale |
| COHEN ADDAD Jean-Pierre | Physique |
| COLIN DE VERDIERE Yves | Mathématiques Pures |
| CYROT Michel | Physique du Solide |
| DEBELMAS Jacques | Géologie Générale |
| DEGRANGE Charles | Zoologie |
| DEMAILLY Jean-Pierre | Mathématiques Pures |
| DENEUVILLE Alain | Physique |
| DEPORTES Charles | Chimie Minérale |
| DOLIQUE Jean-Michel | Physique des Plasmas |
| DOUCE Roland | Physiologie Végétale |
| DUCROS Pierre | Cristallographie |
| FINKE Gerde | Informatique |
| GAGNAIRE Didier | Chimie Physique |
| GAUTRON René | Chimie |
| GENIES Eugène | Chimie |
| GERMAIN Jean-Pierre | Mécanique, |
| GIDON Maurice | Géologie |
| GUITTON Jacques | Chimie |
| HICTER Pierre | Chimie |
| IDELMAN Simon | Physiologie Animale |
| JANIN Bernard | Géographie |
| JOLY Jean René | Mathématiques Pures |

| | |
|---|---|
| JOSELEAU Jean Paul | Biochimie |
| KAHANE André, détaché | Physique |
| KAHANE Josette | Physique |
| KRAKOWIAK Sacha | Mathématiques Appliquées |
| LAJZEROWICZ Jeanine | Physique |
| LAJZEROWICZ Joseph | Physique |
| LAURENT Pierre-Jean | Mathématiques Appliquées |
| LEBRETON Alain | Mathématiques Appliquées |
| DE LEIRIS Joël | Biologie |
| LHOMME Jean | Chimie |
| LLIBOUTRY Louis | Géophysique |
| LOISEAUX Jean-Marie | Sciences Nucléaires I.S.N. |
| LONGEQUEUE Nicole | Physique |
| LUNA Domingo | Mathématiques Pures |
| MACHE Régis | Physiologie Végétale |
| MASCLE Georges | Géologie |
| MAYNARD Roger | Physique du Solide |
| OMONT Alain | Astrophysique |
| OZENDA Paul | Botanique (Biologie Végétale) |
| PANNETIER Jean | Chimie |
| PAYAN Jean-Jacques | Mathématiques Pures |
| PEBAY-PEYROULA Jean-Claude | Physique |
| PERRIER Guy | Géophysique |
| PIERRE Jean Louis | Chimie Organique |
| RENARD Michel | Thermodynamique |
| RIEDTMANN Christine | Mathématiques |
| RINAUDO Marguerite | Chimie CERMAV |
| ROSSI André | Biologie |
| SAXOD Raymond | Biologie Animale |
| SENGEL Philippe | Biologie Animale |
| SERGERAERT Francis | Mathématiques Pures |
| SOUCHIER Bernard | Biologie |
| SOUTIF Michel | Physique |
| STUTZ Pierre | Mécanique |
| TRILLING Laurent | Mathématiques Appliquées |
| VAN CUTSEM Bernard | Mathématiques Appliquées |
| VIALON Pierre | Géologie |

**PROFESSEURS de 2ème Classe**

| | |
|---|---|
| ARMAND Gilbert | Géographie |
| ATTANE Pierre | Mécanique |
| BARET Paul | Chimie |
| BERTIN José | Mathématiques |
| BLANCHI J.Pierre | STAPS |
| BLOCK Marc | Biologie |
| BLUM Jacques | Mathématiques Appliquées |
| BOITET Christian | Mathématiques Appliquées |
| BORNAREL Jean | Physique |
| BORRIONE Dominique | Automatique informatique |
| BOUVET Jean | Biologie |
| BROSSARD Jean | Mathématiques |
| BRUANDET J.François | Physique |
| BRUGAL Gérard | Biologie |
| BRUN Gilbert | Biologie |
| CASTAING Bernard | Physique |
| CERFF Rudiger | Biologie |
| CHIARAMELLA Yves | Mathématiques Appliquées |
| CHOLLET Jean Pierre | Mécanique |
| COLOMBEAU Jean François | Mathématiques (ENSL) |
| COURT Jean | Chimie |
| CUNIN Pierre Yves | Informatique |
| DAVID Jean | Géographie |

| | |
|---|---|
| DHOUAILLY Danielle | Biologie |
| DUFRESNOY Alain | Mathématiques Pures |
| GASPARD François | Physique |
| GIDON Maurice | Géologie |
| GIGNOUX Claude | Sciences Nucléaires |
| GILLARD Roland | Mathématiques Pures |
| GIORNI Alain | Sciences Nucléaires |
| GONZALEZ SPRINBERG Gérardo | Mathématiques Pures |
| GUIGO Maryse | Géographie |
| GUMUCHAIN Hervé | Géographie |
| HACQUES Gérard | Mathématiques Appliquées |
| HERBIN Jacky | Géographie |
| HERAULT Jeanny | Physique |
| HERINO Roland | Physique |
| JARDON Pierre | Chimie |
| KERCKHOVE Claude | Géologie |
| MANDARON Paul | Biologie |
| MARTINEZ Francis | Mathématiques Appliquées |
| MOREL Alain | Géographie |
| NEMOZ Alain | Thermodymamique CNRS - CRTBT |
| NGUYEN HUY Xuong | Informatique |
| OUDET Bruno | Mathématiques Appliquées |
| PAUTOU Guy | Biologie |
| PECHER Arnaud | Géologie |
| PELMONT Jean | Biochimie |
| PELLETIER Guy | Astrophysique |
| PERRIN Claude | Sciences Nucléaires I.S.N. |
| PIBOULE Michel | Géologie |
| RAYNAUD Hervé | Mathématiques Appliquées |
| REGNARD Jean René | Physique |
| RICHARD Jean-Marc | Physique |
| RIEDTMANN Christine | Mathématiques Pures |
| ROBERT Danielle | Chimie |
| ROBERT Gilles | Mathématiques Pures |
| ROBERT Jean-Bernard | Chimie Physique |
| SARROT-REYNAULD Jean | Géologie |
| SAYETAT Françoise | Physique |
| SERVE Denis | Chimie |
| STOECKEL Frédéric | Physique |
| SCHOLL Pierre-Claude | Mathématiques Appliquées |
| SUBRA Robert | Chimie |
| VALLADE Marcel | Physique |
| VIDAL Michel | Chimie Organique |
| VINCENT Gilbert | Physique |
| VIVIAN Robert | Géographie |
| VOTTERO Philippe | Chimie |

## MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

### PROFESSEURS de 1ère Classe

| | |
|---|---|
| BUISSON Roger | Physique IUT 1 |
| CHEHIKIAN Alain | E.E.A. I.U.T.1 |
| DODU Jacques | Mécanique Appliquée IUT 1 |
| NEGRE Robert | Génie Civil IUT 1 |
| NOUGARET Marcel | Automatique IUT 1 |
| PERARD Jacques | EEA. IUT 1 |

### PROFESSEURS de 2ème classe

| | |
|---|---|
| BEE Marc | Physique IUT 1 |
| BOUTHINON Michel | EEA. IUT 1 |
| CHAMBON René | Génie Mécanique IUT 1 |
| CHENAVAS Jean | Physique IUT 1 |

| | |
|---|---|
| CHILO Jean | Physique IUT 1 |
| CHOUTEAU Gérard | Physique IUT 1 |
| CONTE René | Physique IUT 1 |
| FOSTER Panayotis | Chimie IUT 1 |
| GOSSE Jean-Pierre | EEA.IUT 1 |
| GROS Yves | Physique IUT 1 |
| HAMAR Roger | Chimie IUT 1 |
| KUHN Gérard, (Détaché) | Physique IUT 1 |
| LEVIEL Jean Louis | Physique IUT 1 |
| MAZUER Jean | Physique IUT 1 |
| MICHOULIER Jean | Physique IUT 1 |
| MONLLOR Christian | EEA.IUT 1 |
| PERRAUD Robert | Chimie IUT 1 |
| PIERRE Gérard | Chimie IUT 1 |
| TERRIEZ Jean-Michel | Génie Mécanique IUT 1 |
| TOUZAIN Philippe | Chimie IUT 1 |
| TURGEMAN Sylvain | Génie civil |
| VINCENDON Marc | Chimie IUT 1 |
| ZIGONE Michel | Physique IUT 1 |

**PROFESSEURS DE PHARMACIE**

| | | |
|---|---|---|
| AGNIUS-DELORD Claudine | Physique | Faculté La Tronche |
| ALARY Josette | Chimie Analytique | Faculté La Tronche |
| BERIEL Hélène | Physiologie et Pharmacologie | Faculté La Tronche |
| CUSSAC Max | Chimie Therapeutique | Faculté La Tronche |
| DEMENGE Pierre | Pharmacodynamie | Faculté La Tronche |
| FAVIER Alain | Biochimie | C.H.R.G. |
| JEANNIN Charles | Pharmacie Galénique | Faculté Meylan |
| LATURAZE Jean | Biochimie | Faculté La Tronche |
| LUU DUC Cuong | Chimie Générale | Faculté La Tronche |
| MARIOTTE Anne-Marie | Pharmacognosie | Faculté La Tronche |
| MARZIN Daniel | Toxicologie | Faculté Meylan |
| RENAUDET Jacqueline | Bactériologie | Faculté La Tronche |
| ROCHAT Jacques | Hygiéne et Hydrologie | Faculté La Tronche |
| SEIGLE-MURANDI Françoise | Botanique et Cryptogamie | Faculté Meylan |
| VERAIN Alice | Pharmacie Galénique | Faculté Meylan |

**MEMBRES DU CORPS ENSEIGNANT DE MEDECINE**

**PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE**

| | | |
|---|---|---|
| AMBLARD Pierre | Dermatologie | C.H.R.G. |
| AMBROISE-THOMAS Pierre | Parasitologie | C.H.R.G. |
| BEAUDOING André | Pédiatrie-Puericulture | C.H.R.G. |
| BEZEZ Henri | Orthopédie-Traumatologie | Hopital SUD |
| BONNET Jean-Louis | Ophtalmologie | C.H.R.G. |
| BOUCHET Yves | Anatomie | Faculté La Merci |
| | Chirurgie Générale et Digestive | C.H.R.G. |
| BUTEL Jean | Orthopédie-Traumatologie | C.H.R.G. |
| CHAMBAZ Edmond | Biochimie | C.H.R.G. |
| CHAMPETIER Jean | Anatomie-Topographique et Appliquée | C.H.R.G. |
| CHARACHON Robert | O.R.L. | C.H.R.G. |
| COLOMB Maurice | Immunologie | Hopital sud |
| COUDERC Pierre | Anatomie-Pathologique | C.H.R.G. |
| DELORMAS Pierre | Pneumophtisiologie | C.H.R.G. |
| DENIS Bernard | Cardiologie | C.H.R.G. |
| GAVEND Michel | Pharmacologie | Faculté La Merci |

| | | |
|---|---|---|
| HOLLARD Daniel | Hématologie | C.H.R.G. |
| LATREILLE René | Chirurgie Thoracique et Cardiovasculaire | C.H.R.G. |
| LE NOC Pierre | Bactériologie-Virologie | C.H.R.G. |
| MALINAS Yves | Gynécologie et Obstétrique | C.H.R.G. |
| MALLION Jean-Michel | Médecine du Travail | C.H.R.G. |
| MICOUD Max | Clinique Médicale et Maladies Infectieuses | C.H.R.G. |
| MOURIQUAND Claude | Histologie | Faculté La Merci |
| PARAMELLE Bernard | Pneumologie | C.H.R.G. |
| PERRET Jean | Neurologie | C.H.R.G. |
| RACHAIL Michel | Hépato-Gastro-Entérologie | C.H.R.G. |
| DE ROUGEMONT Jacques | Neurochirurgie | C.H.R.G. |
| SARRAZIN Roger | Clinique Chirurgicale | C.H.R.G. |
| STIEGLITZ Paul | Anestésiologie | C.H.R.G. |
| TANCHE Maurice | Physiologie | Faculté La Merci |
| VIGNAIS Pierre | Biochimie | Faculté La Merci |

**PROFESSEURS 2ème CLASSE**

| | | |
|---|---|---|
| BACHELOT Yvan | Endocrinologie | C.H.R.G. |
| BARGE Michel | Neurochirurgie | C.H.R.G. |
| BENABID Alim Louis | Biophysique | Faculté La Merci |
| BENSA Jean-Claude | Immunologie | Hopital Sud |
| BERNARD Pierre | Gynécologie-Obstétrique | C.H.R.G. |
| BESSARD Germain | Pharmacologie | ABIDJAN |
| BOLLA Michel | Radiothérapie | C.H.R.G. |
| BOST Michel | Pédiatrie | C.H.R.G. |
| BOUCHARLAT Jacques | Psychiatrie Adultes | Hopital Sud |
| BRAMBILLA Christian | Pneumologie | C.H.R.G. |
| CHIROSSEL Jean-Paul | Anatomie-Neurochirurgie | C.H.R.G. |
| COMET Michel | Biophysique | Faculté La Merci |
| CONTAMIN Charles | Chirurgie Thoracique et Cardiovasculaire | C.H.R.G. |
| CORDONNIER Daniel | Néphrologie | C.H.R.G. |
| COULOMB Max | Radiologie | C.H.R.G. |
| CROUZET Guy | Radiologie | C.H.R.G. |
| DEBRU Jean-Luc | Médecine Interne et Toxicologie | C.H.R.G. |
| DEMONGEOT Jacques | Biostatistiques et Informatique Médicale | Faculté La Merci |
| DUPRE Alain | Chirurgie Générale | C.H.R.G. |
| DYON Jean-François | Chirurgie Infantile | C.H.R.G. |
| ETERRADOSSI Jacqueline | Physiologie | Faculté La Merci |
| FAURE Claude | Anatomie et Organogénèse | C.H.R.G. |
| FAURE Gilbert | Urologie | C.H.R.G. |
| FOURNET Jacques | Hépato-Gastro-Entérologie | C.H.R.G. |
| FRANCO Alain | Médecine Interne | C.H.R.G. |
| GIRARDET Pierre | Anesthésiologie | C.H.R.G. |
| GUIDICELLI Henri | Chirurgie Générale et Vasculaire | C.H.R.G. |
| GUIGNIER Michel | Thérapeutique et Réanimation Médicale | C.H.R.G. |
| HADJIAN Arthur | Biochimie | Faculté La Merci |
| HALIMI Serge | Endocrinologie et Maladies Métaboliques | C.H.R.G. |
| HOSTEIN Jean | Hépato-Gastro-Entérologie | C.H.R.G. |
| HUGONOT Robert | Médecine Interne | C.H.R.G. |
| JALBERT Pierre | Histologie-Cytogénétique | C.H.R.G. |
| JUNIEN-LAVILLAUROY Claude | O.R.L. | C.H.R.G. |
| KOLODIE Lucien | Hématologie Biologique | C.H.R.G. |
| LETOUBLON Christian | Chirurgie Générale | C.H.R.G. |
| MACHECOURT Jacques | Cardiologie et Maladies Vasculaires | C.H.R.G. |
| MAGNIN Robert | Hygiène | C.H.R.G. |
| MASSOT Christian | Médecine Interne | C.H.R.G. |

| | | |
|---|---|---|
| MOUILLON Michel | Ophtalmologie | C.H.R.G. |
| PELLAT Jacques | Neurologie | C.H.R.G. |
| PHELIP Xavier | Rhumatologie | C.H.R.G. |
| RACINET Claude | Gynécologie-Obstétrique | Hopital Sud |
| RAMBAUD Pierre | Pédiatrie | C.H.R.G. |
| RAPHAEL Bernard | Stomatologie | C.H.R.G. |
| SCHAERER René | Cancérologie | C.H.R.G. |
| SEIGNEURIN Jean-Marie | Bactériologie-Virologie | Faculté La Merci |
| SELE Bernard | Cytogénétique | Faculté La Merci |
| SOTTO Jean-Jacques | Hématologie | C.H.R.G. |
| STOEBNER Pierre | Anatomie Pathologique | C.H.R.G. |
| VROUSOS Constantin | Radiothérapie | C.H.R.G. |

*à Myriam,*

*à mes parents*

*Il y a des pierres qui sont au toucher comme de l'huile ou du savon,*
*d'autres comme des feuilles, d'autres comme du sable*
*et chacune a son caractère propre et prie le Om à sa manière, ...*

*Hermann HESSE (Siddhartha)*

Cette thèse a été soutenue par la CEE, dans le cadre du projet N°888 AIDA.

## Avant Propos

*Je tiens à remercier,*

*Monsieur Pierre GENTIL, Directeur du Centre Interuniversitaire de Micro-Electronique et Professeur à l'E.N.S.E.R.G., qui me fait l'honneur de présider le jury de cette thèse.*

*Monsieur Bernard COURTOIS, Directeur de recherche au C.N.R.S. et Directeur-adjoint du laboratoire TIM3, qui m'a acceuilli dans l'Equipe d'Architecture de Ordinateurs, et qui a dirigé avec compétence mes travaux.*

*Monsieur Joan FIGUERAS, Professeur à l'Universitat Politecnica de Catalunya et Responsable du Departament d'Enginyeria Electronica, pour avoir accepté d'être rapporteur de ce travail et membre de ce jury.*

*Monsieur Christian LANDRAULT, Directeur de recherche au C.N.R.S., pour avoir accepté d'être rapporteur de ce travail et membre de ce jury, et pour ses nombreux et précieux conseils.*

*Monsieur Jean-Pierre MOREAU, Chef du département "aide à la conception de circuits intégrés" de SGS-THOMSON, pour l'intérêt qu'il porte à mon travail et pour avoir accepté d'être membre du jury de cette thèse.*

*Monsieur Mohamed BENAHMED, Professeur et Directeur de l'Ecole Nationale des Sciences de l'Informatique de Tunis, pour l'intérêt qu'il porte à mon travail et pour avoir accepté d'être membre du jury de cette thèse.*

*Mon Collègue Mihaïl NICOLAIDIS, Chargé de recherche au C.N.R.S., pour sa participation au jury de cette thèse. Sa collaboration permanente et fructueuse et son aide ont largement contribué à l'évolution de mes travaux.*

*Monsieur Ahmed Amin JERRAYA, Chargé de recherche au C.N.R.S., pour l'intérêt constant qu'il a porté à mon travail, et pour ses nombreux conseils.*

*Monsieur Alain GUYOT, Maître de conférence à l'E.N.S.I.M.A.G., pour m'avoir permi de collaborer avec lui, et pour ses précieuses idées et conseils.*

*Mon père Béchir TORKI, Professeur à l'Université de Constantine et Responsable de l'équipe d'énergétique, pour son encouragement et son soutien. C'est grâce à lui qu'une partie de mon séjour a été possible.*

*J'ai vu défiler, tout au long de mon séjour dans l'équipe, un nombre considérable de personnes, les uns sont restés, d'autres sont partis. Je ne pourrais tous les citer. Qu'ils trouvent, ici, toute ma gratitude et ma reconnaissance, pour leur collaboration en des moments divers et à différents titres.*

**Résumé**

Bien que les techniques d'autotest intégré soient en perpétuel développement sous forme de théories et de schémas de conception, leur réalisation concrète et leur implémentation posent des problèmes cruciaux.

Une stratégie d'autotest intégré est proposée dans cette thèse pour des circuits générés par compilation de silicium.

Le schéma UBIST d'unification du test en-ligne et hors-ligne assure la plupart des tests nécessaires durant la vie d'un circuit intégré (test de fin de fabrication, test de maintenance, test en-ligne, ...).

A la base du schéma UBIST se trouve le schéma "Self-Checking" (test en-ligne, pour lequel le circuit est composé de blocs fonctionnels "Strongly Fault Secure" (SFS) et de contrôleurs "Strongly Code Disjoint" (SCD) ). Le but à atteindre par de tels circuits est couramment appelé le "Totally Self-Checking Goal", qui consiste à détecter la première erreur survenant aux sorties du bloc fonctionnel, sous forme d'indication d'erreur sur les sorties du contrôleur.

Autour de ce schéma Self-Checking est implémentée une structure de test, du type BILBO, assurant des phases de test hors-ligne, qui a pour objectif d'augmenter le taux de couverture des pannes multiples et de renforcer les propriétés SFS et SCD pour certains blocs fonctionnels et contrôleurs. L'unification des tests en-ligne et hors-ligne permet de tirer les avantages de chacun de ces tests, permettant une implémentation efficace d'autotest intégré.

Une méthodologie de conception pour implémenter ce schéma UBIST est proposée pour des parties contrôle hiérarchiques à base de PLAs et des parties opératives parallèles en structure bit-slice (du type de celle du MC 68000). Ce sont les architectures cibles utilisées par le compilateur de silicium SYCO (développé au sein de l'équipe d'architecture des ordinateurs du laboratoire TIM3/IMAG). Une solution topologique efficace est proposée pour ces schémas UBIST.

L'un des points essentiels de cette thèse est qu'elle s'appuie sur des réalisations d'outils de CAO spécifiques au test, compatibles avec les spécifications architecturales et structurelles utilisées par SYCO. C'est une phase importante de validation du concept de "compilateur de circuits autotestables".

**Mots-clés :**

Circuits autotestables, compilation de silicium, schéma UBIST, BILBO, PLA, partie contrôle hiérarchique, partie opérative parallèle, contrôleurs de code détecteur d'erreurs.

**Abstract**

Despite their perpetual development, built-in self-test techniques still require efficient realisation and optimal implementation. However, this frequently involves serious problems.

We propose in this thesis an efficient built-in self-test strategy for a silicon compiler.

The UBIST scheme for unification of on-line and off-line tests, ensures most of the tests needed during the life of a circuit (manufacturing test, maintenance test, concurrent error detection, ...).

The basic structure of the UBIST scheme is the self-checking circuit (on-line test, the circuit is composed of Strongly Fault Secure functional blocks and Strongly Code Disjoint checkers). The goal to be reached by these circuits is often called the "Totally Self-Checking goal", i. e. the first error on the functional block's outputs must be detected at the checker's outputs by an error indication. Around this self-checking structure is implemented a BILBO-like test structure, ensuring off-line phase tests, which increases multiple fault coverage, strengthening the SFS and SCD properties for some functional blocks and checkers. The unification of on-line and off-line tests takes advantage of each of these tests, giving an efficient built-in self-test implementation.

A methodology for implementing UBIST scheme is proposed for hierarchical control sections built around PLAs, and parallel bit-sliced datapaths (i. e. a MC 68000-like datapath). These architectures are the target architecture used by the SYCO silicon compiler (developed in the Computer Architecture Group of the IMAG/TIM3 laboratory). An efficient topological solution is proposed for these UBIST schemes. Finally, one of the major points of the thesis, is that it proposes test specific CAD tools, compatible with the architectural and structural specifications used by SYCO.

It is an important step for validating the concept of "Silicon Compilation of Testable Circuits".

**Keywords**

Testable circuits, silicon compilation, UBIST scheme, BILBO, PLA, hierarchical control section, parallel datapath, checkers, error dectecting codes.

## Avertissement

Des remarques générales, nous ont conduit à adopter certaines dispositions concernant la terminologie utilisée dans cette thèse.

L'adoption systématique de termes anglais est non adaptée et inadéquate. Pourtant leur utilisation aurait été plus précise, puisque ces termes sont connus et précis dans leur contexte.

L'équivalent du mot "Self-Checking", par exemple, dans le domaine de l'autotestabilité en-ligne, a été dans la littérature tantôt "autocontrôlable", tantôt "autotestable".

De même l'utilisation du terme contrôle pour désigner l'équivalent anglais "checking" pour le test, et "control" pour le séquencement d'un microprocesseur, peut entraîner des confusions.

Une normalisation des termes serait la bienvenue !

Nous ne prétendons pas ici en proposer, ni en imposer une. Néanmoins, nous utiliserons pour cette thèse la terminologie suivante :

Self-Checking (SC) : Auto-Contrôlable (AC)

Totally Self-Checking (TSC) : Totalement Auto-Contrôlable (TAC)

Self-Testing (ST) : Auto-Testable (AT)

Fault Secure (FS) : Sûr en Présence de Défauts (SPD)

Strongly Fault Secure (SFS) : Fortement Sûr en Présence de Défauts (FSPD)

Code Disjoint (CD) : à Code Disjoint (CD)

Strongly Code Disjoint (SCD) : Fortement à Code Disjoint (FCD)

Self-Exercising : Auto-Activable

Les définitions de ces propriétés se trouvent pour la plupart au premier chapitre de cette thèse.

D'autre part, certaines abréviations et autres mots tels que, BIST, s-open, UBIST, BILBO, PLA, ROM, ... ont été gardés volontairement tels qu'ils sont, vu leur popularité.

Par ailleurs, nous avons utilisé les termes dérivés du verbe "coder", pour tout ce qui concerne les codes détecteurs d'erreurs ; et les termes dérivés du verbe "encoder" pour tout ce qui concerne le code binaire classique, (le verbe décoder représenterait l'action inverse du verbe encoder).

Finalement, nous remarquerons que certaines parties de cette thèse (figures ou paragraphes) se retrouvent de manière redondante. Ceci est venu de manière directe lors de la rédaction, et contribue fortement à la clarté des propos tenus localement. Le lecteur se trouve libéré de se reporter à telle ou telle partie de la thèse. Ce qui ne veut pas dire que les différentes parties de cette thèse peuvent être toujours lues indépendamment l'une de l'autre.

# SOMMAIRE

# INTRODUCTION

L'unification des processus de conception de circuits intégrés et des méthodologies de test est l'étape ultime permettant de concevoir et de générer rapidement, et à moindre coût des circuits fiables et viables.

Le processus de conception d'un circuit intégré, partant d'une spécification de haut niveau jusqu'au niveau layout, est laborieux. Il a donc été nécessaire de diviser ce processus en plusieurs étapes de conception, pour lesquelles des outils de CAO ont été développés permettant le passage d'une description comportementale de haut niveau à un ou plusieurs niveaux intermédiaires, pour aboutir au niveau final qu'est le layout du circuit. Ce morcellement du processus de conception a porté ses fruits. En effet, on dispose actuellement de puissants outils de CAO, qui permettent le passage d'une description à une autre.

Parallèlement à celà, on a pu assister à l'évolution des méthodologies de conception et à la définition de langages de spécification de haut niveau. Ceci a entraîné le développement actuel des compilateurs de silicium.

Un compilateur de silicium idéal permet le passage automatique d'une description de haut niveau d'un circuit à son layout. Le concepteur n'a donc plus à sa charge les rudes besognes que cette traduction peut entraîner.

Actuellement, les compilateurs de silicium n'ont pas encore complètement atteint cet objectif. En effet, un des problèmes cruciaux posé à la compilation de silicium est celui de la vérification du résultat de compilation et/ou des outils de compilation.

D'autre part, les concepteurs cherchent à inclure, très tôt dans le processus de conception, des facilités de test, leur permettant de valider totalement ou partiellement leur produit, et leur permettant d'effectuer différents types de tests durant la vie du circuit. Le type de test adopté répondra aux différents besoins requis (par exemple: test de fin de fabrication, test de maintenance, test en-ligne, etc...).

Le schéma unifiant les tests intégrés en-ligne et hors-ligne (nommé UBIST) [NIC 88] répond à ces exigences. Les méthodologies de conception qui suivent ce schéma adoptent la même démarche qu'une conception classique.

Il n'en reste pas moins que les spécifications de haut niveau décrivant l'autotest intégré n'existent pas actuellement, puisqu'aucun langage de description n'est adapté pour lui trouver une solution souple et efficace.

Un tel langage doit être spécialisé pour décrire toute la méthodologie et la fonctionnalité de la technique UBIST indépendamment des fonctionnalités normales du circuit.

Néanmoins, si l'on se place au niveau architectural ou structurel, les spécifications utilisées par les compilateurs de silicium peuvent être utilisées pour décrire le schéma UBIST et le processus de génération automatique de circuits UBIST devient possible.

C'est dans cet esprit que cette thèse se veut en même temps, initiatrice à l'autotest intégré et validatrice d'une stratégie de testabilité pour le cas particulier d'architectures adoptées par un compilateur de silicium comportemental (i. e. le compilateur de silicium SYCO [JER 89]).

Greffer des outils de CAO de testabilité à un compilateur de silicium représente une phase de faisabilité importante d'unification d'outils de CAO, conduisant à la réalisation de compilateurs de circuits autotestables.

Le premier chapitre de cette thèse est une introduction au test intégré en-ligne. On adopte le schéma d'autocontrôle basé sur le codage et le contrôle des sorties des blocs fonctionnels. Le but à atteindre par une telle structure est appelé le but de l'autocontrôlabilité totale, qui consiste à détecter la première manifestation d'erreur dûe à une panne survenant au sein du circuit.

Les hypothèses de pannes considérées sont basées sur des pannes physiques réelles, elles sont proposées dans [COU 81] et [GAL 80].

Des techniques de base permettant l'analyse de la manifestation des pannes en erreurs sont rappelées. Ces notions ont servi à établir des règles de conception pour provoquer un type d'erreur précis lors de l'occurrence d'une panne simple. Des techniques de codage des bits d'information sont alors présentées. Le choix d'un code se fait donc en fonction du type d'erreur considéré.

Le deuxième chapitre présente une analyse sur les modèles de pannes utilisés dans la littérature et leur application au cas de structures régulières tels que les PLAs.

On propose ensuite des implémentations de contrôleurs. Les contrôleurs sont des organes clés d'un schéma d'autocontrôlabilité. Leur implémentation pose des problèmes cruciaux. Nous proposons des solutions efficaces pour leur implémentation et nous montrons leur introduction dans un schéma UBIST d'unification du test en-ligne et hors-ligne. Des notions de test hors-ligne sont rappelées, notamment la technique BILBO. Le schéma UBIST est présenté, il a été établi dans [NIC 88], et il est pour les circuits autocontrôlables ce qu'est le schéma BIST pour les circuits conventionnels.

Le troisième chapitre présente la stratégie globale adoptée pour introduire le schéma UBIST au compilateur de silicium SYCO (développé au sein de l'équipe d'architecture des ordinateurs du laboratoire TIM3 de l'IMAG).

On présentera ensuite d'autres stratégies développées ailleurs [FUN 85], [FUN 86], [SAB 86], [CAT 89], [BEE 89],... nous permettant de situer notre méthodologie dans le domaine de la compilation de circuits autotestables.

Les critères de comparaisons entre ces différentes méthodologies, sont liées:

- au type d'architecture (hiérarchique, parallèle, anarchique, ...),
- au niveau de description (comportemental, structurel, architectural, ...),
- au type de test adopté (intégré, externe, en-ligne, hors-ligne, ...),
- à la méthode de test (autotest, BIST, Scan-Path, ...),
- au niveau de panne considéré (porte logique, interrupteurs, transistors),
- à la couverture de panne atteinte (totale, partielle ou statistique).

La méthodologie de testabilité établie pour le compilateur de silicium SYCO se résume de la façon suivante:

- On part d'une description comportementale (utilisation d'une description algorithmique du circuit),
- on utilise une architecture cible pour le résultat de la compilation (partie contrôle hiérarchique et partie opérative parallèle),
- on prend en compte des hypothèses de pannes au niveau physique (i. e. niveau transistor), et pour lesquelles on veut adopter une technique garantissant une couverture de panne totale,
- on établit le schéma de test intégré UBIST adéquate unifiant le test en-ligne et hors-ligne (schéma d'autocontrôle et schéma BIST de type

BILBO) pour les différentes structures de cette architecture.

Les contraintes fixées sont, comme on peut le voir, fortes par rapport à celles utilisées par les autres méthodologies de test inclues aux compilateurs de silicium [FUN 85], [SAB 86], [CAT 89], [BEE 89],... On a donc voulu valider dans cette thèse une méthodologie performante, et initier les utilisateurs et les concepteurs de compilateurs de silicium au développement et à l'introduction d'outils de CAO de testabilité dans un compilateur de silicium.

Le quatrième chapitre propose deux outils de CAO développés dans le cadre de cette thèse.

Le premier, nommé PROTECT, génère automatiquement le codage des sorties de PLAs ou de ROMs [TOR 88a]. Une méthodologie particulière et originale est proposée, elle est à l'origine des performances obtenues pour cet outil.

Le deuxième outil est un simulateur spécialisé pour le calcul de la signature pour un test exhaustif ou pseudo-aléatoire d'un PLA ou d'une ROM. Cet outil utilise une technique de calcul, conduisant à un calcul quasi-vectoriel et entraînant des temps de calcul très raisonnables.

De tels outils sont une initiative qui doit encourager les concepteurs et les spécialistes du test à en développer d'autres si l'on veut promouvoir le test intégré et son introduction comme élément à part entière dans une chaîne d'outils de CAO.

Le cinquième chapitre traite l'implémentation UBIST de parties contrôle hiérarchiques à base de PLAs.

L'architecture de ces parties contrôle est l'architecture cible adoptée par le compilateur de silicium SYCO.

Les blocs fonctionnels sont rendus "fortement sûrs en présence de défauts" par l'utilisation de codage des sorties de PLAs, et la conception duale des différents circuits à logique anarchique (variables de contrôle, bus de contrôle et de comptes rendus d'exécution, et blocs de synchronisation).

Une solution topologique pour introduire efficacement les contrôleurs et modifier les registres d'entrée et de sortie sous forme de registres BILBO, est proposée, [TOR 88b].

L'augmentation en surface du schéma UBIST par rapport à la partie contrôle originale a été estimée à 53% pour un exemple généré par SYCO (i. e. le SYCO 6502 [REI 86]). Cette augmentation peut sembler excessive, mais si les objectifs requis sont une bonne couverture de panne et une détectabilité en-ligne, cette augmentation reste plus que tolérable. En effet, le schéma UBIST de parties contrôle, que nous proposons, assure 100% de couverture des pannes simples et une grande partie des pannes multiples.

Une implémentation UBIST de parties contrôle microprogrammées du type de celle du MC 68000 est proposée dans [NIC 90] où l'augmentation en surface représente moins de 30%.

L'augmentation en surface pour introduire le schéma UBIST dans une partie contrôle dépend de l'architecture de celle-ci.

Le sixième chapitre introduit un schéma d'autocontrôle de parties opératives du type de celle du MC 68000. La conception "fortement sûre en présence de défauts" des différents éléments de parties opératives est proposée.

La régularité de la partie opérative nous a permis d'établir une stratégie globale et locale d'une telle conception. Les opérateurs UA et UAL sont

conçus de façon à inclure leur duals, leur duplication est ainsi évitée.

Le septième chapitre propose un schéma BIST d'UAs et d'UALs. Ce schéma est basé sur les concepts de C-testabilité.

Des générateurs de vecteurs de test sont proposés, la taille des séquences de test qu'ils génèrent est indépendante du nombre de bits des UAs et UALs. L'augmentation de surface dûe à ces générateurs décroît de manière significative lorsque la partie opérative utilise des UAs et UALs à grand nombre de bits.

Ce schéma BIST associé non pas à une partie opérative conventionnelle, mais à une partie opérative autocontrôlable telle que celle présentée au sixième chapitre, réalisera le schéma UBIST de parties opératives. Un schéma général et une évaluation des performances sont données.

# CHAPITRE I

## Le test intégré en ligne

**Introduction :**

Dans ce chapitre, un rappel des principales définitions et règles de conception des systèmes intégrés autocontrôlables et des méthodes de détection des erreurs, sont exposés.

On part tout d'abord d'hypothèses de pannes analytiques bien définies, [COU 81]. Des définitions seront ensuite rappelées, elles sont dues à ANDERSON, SMITH, NICOLAIDIS, [AND 71], [SMI 78], [NIC 84a],.... Des règles de conception en NMOS de circuits "fortement sûrs en présence de défauts" (FSPD) sont exposées dans [NIC 84], ces règles existent pour des erreurs simples, unidirectionnelles ou multiples. On rappellera l'essentiel des règles pour les erreurs simples et unidirectionnelles, ce sont elles qui nous intéressent dans cette étude.

La conception d'un circuit autotestable en ligne consiste en un ajout de matériel pour que le circuit en fonctionnement normal soit capable de détecter toute occurrence de panne.

Suivant le type d'erreur, on adoptera un type de codage des bits à contrôler, on citera les techniques utilisées pour les différents types d'erreurs et on présentera les cinq techniques de codage les plus utilisées : le code de parité, le code de Berger, le code de Berger modifié, le code m-parmi-n, et le code double-rail.

**I. 1 Hypothèses de pannes :**

Une classification d'hypothèses de pannes a été proposée dans [COU 81], elle concerne des circuits conçus en technologie NMOS grille Si. Cette classification a été effectuée en prenant en compte le taux d'occurence des différentes défaillances, en étudiant les mécanismes de défaillance et leur fréquence. Dans [CRO 78] un exemple concret a été étudié où figuraient les pourcentages de ces défaillances, certes non significatifs quantitativement, mais de valeur qualitative permettant une classification.

La classification des hypothèses de pannes analytiques établie par B. Courtois [COU 81] est donnée en table I.1.

| CLASSE 0 | CLASSE I | CLASSE II |
|---|---|---|
| Un panne simple:<br>• 1contact, 1précontact.<br>• 1 MOS s-on<br>ou s-open<br>• 1 Alu coupé.<br>• 1 poly coupé.<br>• 1 diffusion coupée:<br>grille flottante -><br>MOS s-open | Classe 0, plus:<br>• 1 court-circuit entre Alu et Alu le plus proche géographiquement.<br>• Même chose pour diffusion. | Classe I, plus:<br>• Court-circuit entre Alus quelconques.<br>• Même chose pour diffusion.<br>• Pannes multiples. |

**Table I.1: Classe d'hypothèses de pannes (NMOS) [COU 81].**

Dans le cas de la technologie CMOS la même classification d'hypothèses de pannes peut être considérée, les mécanismes de défaillance étant les mêmes qu'en NMOS, bien que leurs conséquences soient différentes.

Pour une technologie CMOS à plusieurs couches d'interconnexions la classe 0 peut être redéfinie en considérant une panne physique simple, tel que:

- Un contact défaillant (un contact, précontact, via, ...).
- Un MOS défaillant collé s-on, s-open (MOS P, ou MOS N).
- Une équipotentielle coupée (diffusion, polySi, Alu1 ou Alu2).

On considère une technologie CMOS à 2 niveaux d'aluminium.

La classe I est redéfinie en CMOS en prenant comme hypothèses possibles : 1 court-circuit entre 2 diffusions, entre 2 équipotentielles Alu 1 ou 2 équipotentielles Alu 2. Ces courts-circuits sont considérés pour des équipotentielles immédiatement proches l'une de l'autre et de même matériau.

La classe $I^+$ admet des courts-circuits entre matériaux de n'importe quel type, qui sont au même niveau, ou à des niveaux différents, et la notion de proximité géographique importe peu; ce qui importe c'est que la panne

physique reste simple.

La classe II+ représente la classe I+ étendue à des pannes simples et multiples.

| CLASSE 0 | CLASSE I | CLASSE I + | CLASSE II + |
|---|---|---|---|
| Une panne simple:<br>• 1 contact interniveaux.<br>• 1 MOS N ou P s-on ou s-open.<br>• 1 équipotentielle coupée : diffusion, poly, Alu1, Alu2.<br><br>Grille flottante -> MOS s-open | Classe 0 plus:<br>• 1 court-circuit entre 1 diffusion et une autre diff la plus proche géographiquement<br><br>• Idem pour Alu1.<br>• Idem pour Alu2.<br>• Idem pour court-circuits entre Alu1 et Alu2. | Classe I plus : Court-circuit quelconque de la classe I entraînant des pannes simples. | Classe I+plus : pannes multiples de la classe I+ |

**Table I.2 : Classes d'hypothèses de pannes (CMOS) [COU 81].**

La table I.2 illustre les classes 0, I, I+, II+, pour une technologie CMOS à 2 niveaux d'interconnexion.

Par la suite, seules les pannes de la classe I nous intéresseront, les pannes de la classe I+ et II+ étant peu probables et pouvant être évitées en respectant des règles de conception.

Les classes d'hypothèses de pannes ont amené à établir des classes de défauts [CRO 78].

L'analyse de la manifestation logique des pannes a fait apparaître trois classes d'erreurs suivant que:

- Une seule porte est affectée (erreur simple).
- Plusieurs portes sont affectées de la même façon (erreur unidirectionnelle).
- Plusieurs portes sont affectées différemment (erreur multiple).

Pour chacune de ces classes, l'effet de la panne peut être soit permanent (erreur permanente), soit transitoire (erreur transitoire).

La table I.3 donne les principales pannes rencontrés dans chacune des classes.

| Erreurs | Pannes permanentes | Pannes transitoires |
|---|---|---|
| simples | • court-circuit ou coupure au niveau du réseau de transistors de commande d'une porte.<br>• Transistor de précharge coupé.<br>• court-circuit franc entre 2 portes | • coupure d'une interconnexion entre porte.<br>• transistor défectueux par décalage de la tension de seuil. |
| unidirectionnelles | • coupure d'une ligne d'alimentation<br>• court-circuit entre plus de 2 portes. | • décalage de la tension de seuil de plusieurs transistors (par pollution ponctuelle ou dérive normale). |
| multiples | • court-circuit résistif entre 2 portes entraînant un niveau mal défini.<br>• dégradation importante de propagation | • dégradation des temps de propagation |

**Table I.3 : Classification des erreurs en fonction des pannes [CRO 78].**

On s'attachera particulièrement aux erreurs simples et unidirectionnelles, car les PLAs implémentés en logique NOR-NOR n'admettent pas d'erreurs multiples pour la classe I d'hypothèses de pannes.

De plus l'autodétection matérielle des erreurs (structure autocontrôlable) traite de la même façon les erreurs transitoires et permanentes, on ne fera donc pas de distinction entre les erreurs permanentes ou transitoires.

Par la suite, la classe I d'hypothèses de pannes sera considérée pour traiter la conception des blocs fonctionnels. On peut montrer que les règles établies pour la classe I (NMOS) peuvent être étendues à la classe I (CMOS), puis à la classe $I^+$ (CMOS); en effet les courts-circuits considérés sont très peu probables et peuvent être évités par le respect des règles de conception.

Dans [CRO 78], des règles de conception ont été établies pour les classes de pannes provoquant des erreurs (Table I.3), les types d'erreurs qui seront produits aux sorties d'un circuit fonctionnel dépendront de la propagation des erreurs depuis l'origine de leur apparition jusqu'à ces sorties, les propriétés de propagation d'erreurs établies dans [CRO 78] s'arrêtent au niveau des portes logiques. Dans [NIC 84a] ces propriétés ont été transposées au niveau du transistor, et les hypothèses de pannes (Table I.1) sont considérées à ce niveau. On retiendra aussi les notions de succession, succession inversante, parité d'inversion et degré de divergence.

Ces définitions sont détaillées dans [NIC 84a], on retiendra celles de la parité d'inversion et du degré de divergence.

**I. 2 Parité d'inversion et degré de divergence:**

**I. 2. 1 Parité d'inversion:**

***Définition:*** *La parité d'inversion $I_p(p)$ d'un chemin p, dans lequel la propriété d'inversion est définie pour tous les couples de 2 lignes successives, est le nombre binaire $I_P(p) \in \{0,1\}$, tel que $I_p(p)=n$ (modulo2), n étant le nombre de couples inversant de lignes successives du chemin p.*

**Remarques:**

• La propriété de parité d'inversion est importante pour la manifestation et la détection d'erreur, en effet il existe des couples de lignes pour lesquels cette propriété est indéfinie.

• Cette définition permet de considérer la parité d'inversion de chemins entre une ligne interne à une porte et une sortie primaire, entre une ligne d'alimentation et une ligne de signal etc... Ceci n'est devenu possible que parce que l'on se situe au niveau du transistor.

**Exemples:**

• Parité d'inversion: (niveau transistor)

Les couples inversant et non inversant sont définis (Figure I.1):

**i)** Sur la figure I.1a) le couple $(I_i,I_j)$ est inversant (le transistor est un MOS de signal): la parité d'inversion est égale à 1.

**ii)** Sur la figure I.1b) le couple $(I_i,I_j)$ est inversant (le transistor est un MOS interrupteur): la parité d'inversion est égale à 1.

**iii)** Sur la figure I.1c) le couple $(I_i,I_j)$ est non inversant (le transistor est un MOS interrupteur): la parité d'inversion est égale à 0.

**iv)** Sur la figure I.1d) le couple $(I_i,I_j)$ est indéfini (le transistor est un MOS interrupteur), la ligne $I_k$ peut prendre 2 valeurs (0 , 1), le couple $(I_i,I_j)$ est indéfini vis à vis de la propriété d'inversion: la parité d'inversion est indéfinie.

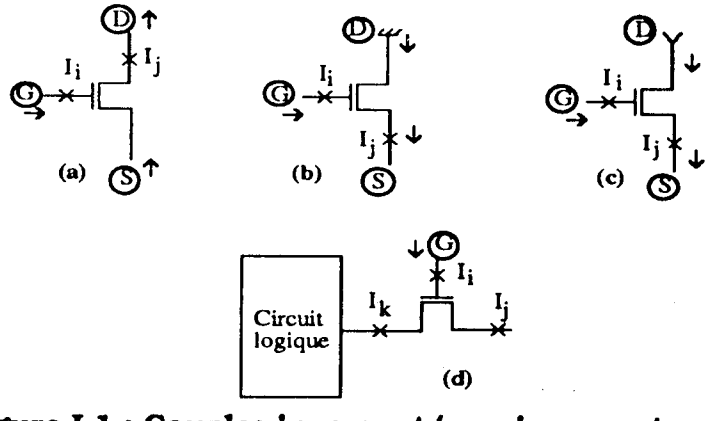



**Figure I.1 : Couples inversant/non inversant.**

• La parité d'inversion: (niveau logique)

C'est la parité d'inversion d'un chemin de succession de portes logiques: les portes logiques de type inverseurs, NAND, NOR, etc... ont une parité d'inversion égale à 1. Les portes logiques de type AND, OR, etc... ont une parité d'inversion égale à 0.

La figure I.2 donne un exemple où le chemin p a une parité d'inversion égale à 1 et le chemin p' a une parité d'inversion égale à 0.

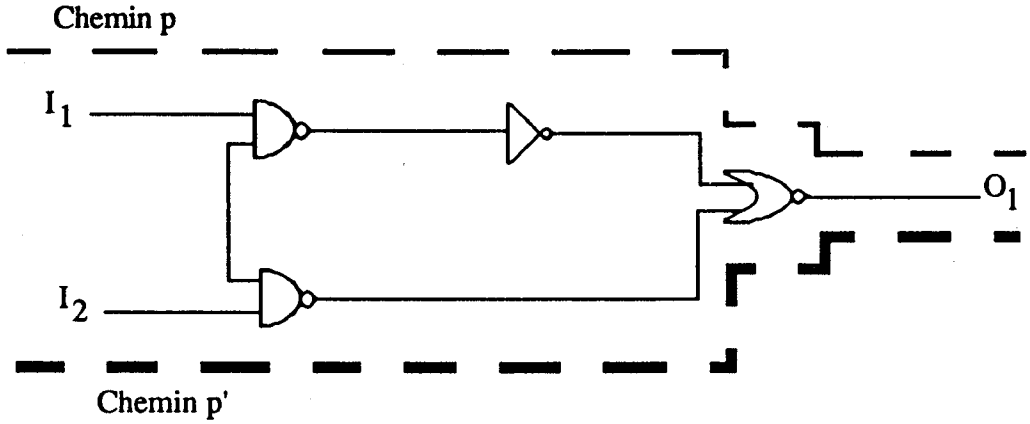



**Figure I.2 : Circuit pour lequel Ip(p)=1 et Ip(p')=0.**

**I. 2. 2 Degré de divergence:**

***Définition:*** *Le degré de divergence d'une ligne $L_{i,1}$ interne d'un bloc fonctionnel est le nombre des sorties primaires $L_{i,r}$ du bloc pour lesquelles il existe des chemins $<L_{i,1}$ , ..... $,L_{i,r}>$.*

Une panne au niveau d'une ligne de degré de divergence n peut provoquer des erreurs sur au plus n sorties du bloc fonctionnel, et créer une erreur multiple d'ordre n.

***Définition:*** *Le degré de divergence maximal d'un bloc fonctionnel, pour un ensemble de lignes internes $L_1$ , $L_2$ ....., $L_k$, est égal au maximum des degrés de divergence de ces lignes:*

$$D_{Max} = Sup[\ D(L_i),\ i=1,...,k\ ].$$

Sur le schéma de la figure I.3, on donne un exemple où le degré de divergence de la ligne $L_0$ est égal à 3: en effet cette ligne affecte trois sorties. La ligne $L_1$, n'affecte qu'une sortie, le degré de divergence de la ligne $L_1$ est égal à 1. Et le degré de divergence maximal du bloc fonctionnel est égal à 3.

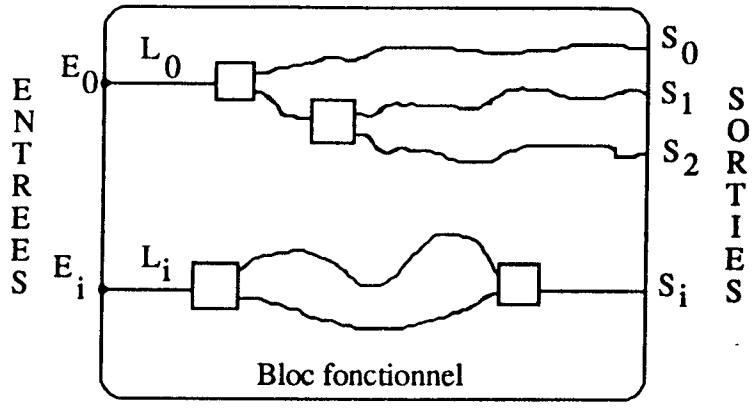



**Figure I.3 : Degré de divergence.**

**I. 3 Conventions:**

Ces conventions sont prises de [SMI 78]. On considère un circuit G en logique combinatoire possédant des entrées multiples et des sorties multiples. Ce circuit fonctionnel peut être formé de portes logiques et de transistors. Si ce circuit G a r entrées primaires, alors les $2^r$ vecteurs binaires de longueur r forment l'ensemble X des vecteurs d'entrées.

G admettant q sorties primaires, l'ensemble Y est formé des $2^q$ vecteurs de sorties de longueur q.

Le circuit G reçoit pendant le fonctionnement normal (avant l'occurrence d'une panne) un sous-ensemble A de vecteurs de X, ($A \subseteq X$), A est appelé le code d'entrée; et il produit un sous-ensemble B de vecteurs G(a,0) de Y, ($B \subseteq Y$), appelé code de sortie. En présence de panne f de l'ensemble de pannes F pris comme modèle, le bloc produit des sorties G(a,f).

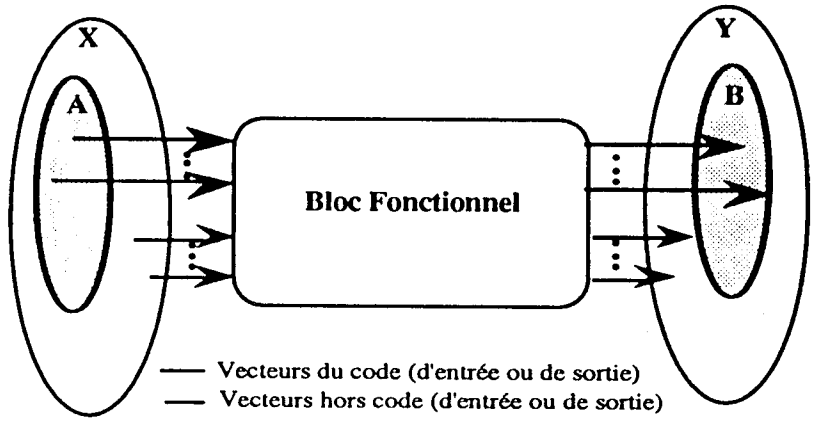



**Figure I.4 : Schéma d'un bloc fonctionnel.**

Pour simplifier ces notions on adopte la notation suivante:

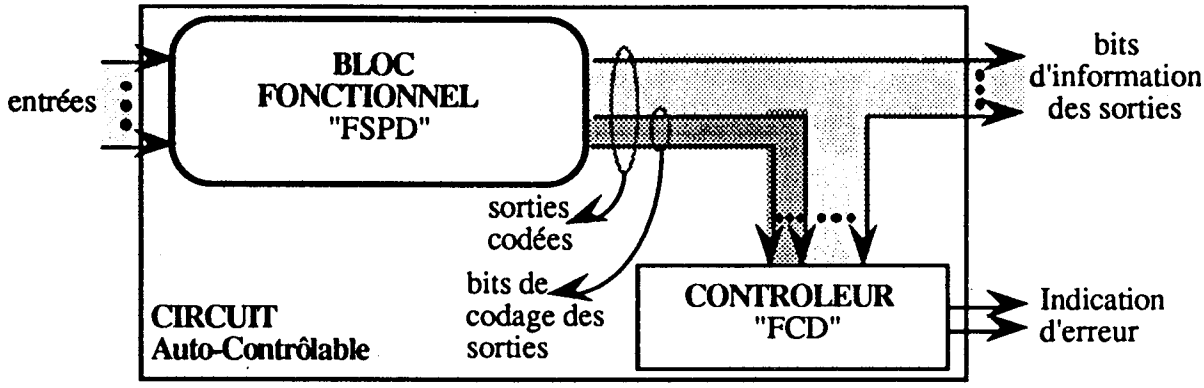
• En présence de la panne f, la sortie du circuit G qui reçoit un vecteur d'entrée x, est notée G(x,f).

• Avant l'occurrence des pannes cette sortie est notée G(x,∅).

**I. 4 Structure générale d'un système 'Auto-Contrôlable"(AC):**

La structure générale est donnée dans la figure I.5, le système "Auto-Contrôlable" (AC) est composé d'un bloc fonctionnel et d'un contrôleur.

Le bloc fonctionnel admet des entrées et génère des sorties codées. Ces sorties codées forment les entrées du contrôleur, celui-ci génère les indications d'erreur à ses sorties.




FSPD : Fortement Sûr en Présence de Défauts.

FCD: Fortement à Code Disjoint.

**Figure I.5 : Structure générale d'un système "Auto-Contrôlable" (AC) à code séparable.**

Le bloc fonctionnel transforme les valeurs de ses entrées en des valeurs codées sur ses sorties de manière à ce que la première manifestation d'erreur (sur les sorties du bloc), due à une panne au sein du circuit, soit immédiatement signalée par le contrôleur sous forme d'indication d'erreur. Un circuit composé d'un bloc fonctionnel "Fortement Sûr en Présence de Défauts" (FSPD) et d'un contrôleur "Fortement à Code Disjoint"(FCD), accomplira le but de "l'autocontrôlabilité totale" ("Totally Self-Checking Goal") (TSC Goal) pour une certaine classe d'hypothèses de pannes.

Les notions de "Totalement Auto-Contrôlable" (TAC), "But de l'Auto-Contrôlabilité Totale", "Fortement Sûr en Présence de Défauts" (FSPD), "Fortement à Code Disjoint" (FCD) seront définies dans le paragraphe suivant.

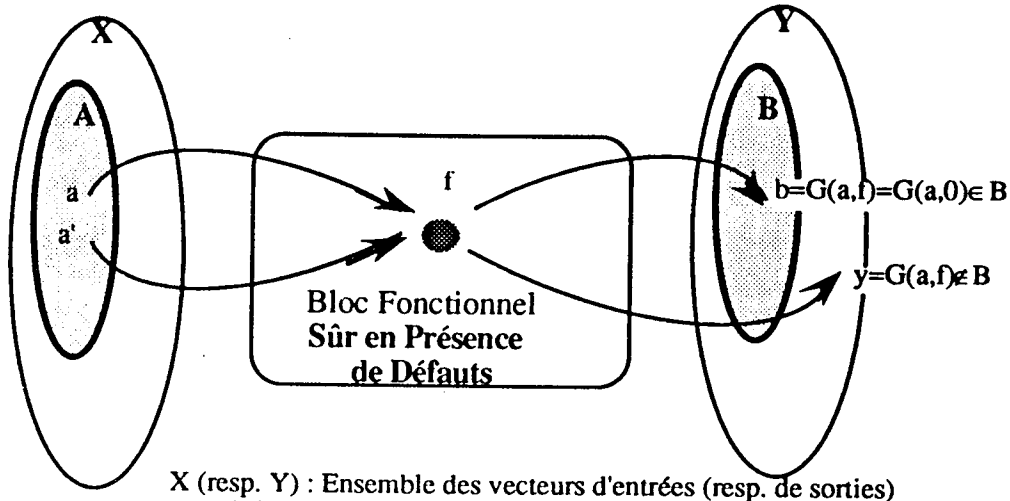
**I. 5 Définitions de base :**

Ces définitions sont dues à ANDERSON [AND 71]. On considère un bloc fonctionnel G, avec son code d'entrée A, son code de sortie B et un ensemble de pannes F.

***Définition 'Sûr en présence de défauts" :*** *Un circuit G est dit "Sûr en présence de défauts" (FS) pour un ensemble de pannes F si pour toute panne*

*f de F et si pour tout vecteur a du code d'entrée A, soit le vecteur de sortie est correct, soit il est en dehors du code de sortie.*

$\forall f \in F$, $\underline{\forall a \in A}$. *On a:* soit $G(a,f)=G(a,0)$.
soit $G(a,f) \notin B$.

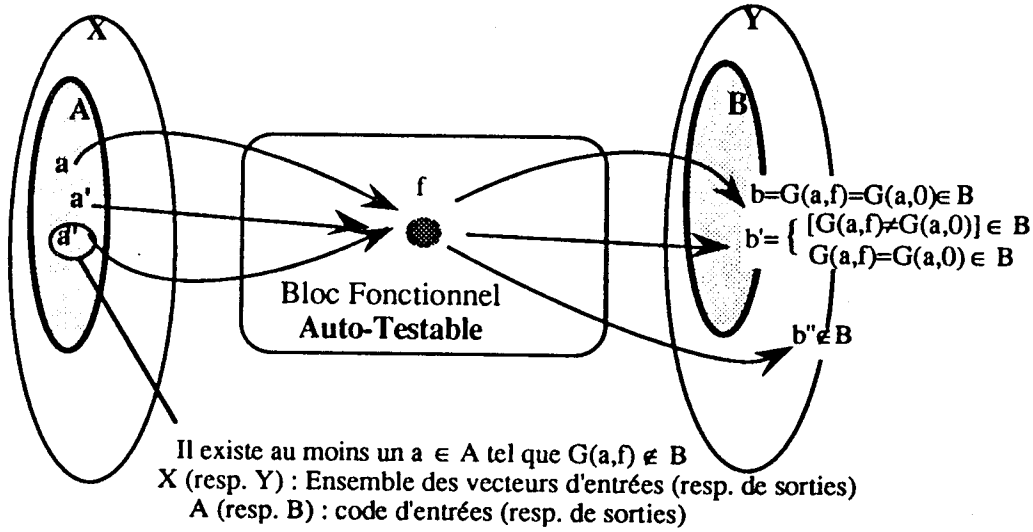



X (resp. Y) : Ensemble des vecteurs d'entrées (resp. de sorties)
A (resp. B) : code d'entrées (resp. de sorties)

**Figure I.6 : Bloc fonctionnel sûr en présence de défauts.**

***Définition 'Auto-Testable'*** *: Un circuit est "Auto-Testable" pour un ensemble de pannes F, si pour toute panne f de F, il existe au moins un vecteur a du code d'entrée pour lequel la sortie est en dehors du code de sortie.*

$\forall f \in F$, $\underline{\exists a \in A}$, *tel que* $G(a,f) \notin B$.




Il existe au moins un a ∈ A tel que G(a,f) ∉ B
X (resp. Y) : Ensemble des vecteurs d'entrées (resp. de sorties)
A (resp. B) : code d'entrées (resp. de sorties)

**Figure I.7 : Bloc fonctionnel AutoTestable.**

- Si G(a,f)=G(a,0), la panne n'intervient pas sur le fonctionnement du circuit.
- Si [G(a',f)≠G(a',0)]∈B, la panne modifie le fonctionnement, mais elle n'est pas détectée car la réponse du circuit appartient au code de sortie.

• Si G(a",f)∉ B, il y a au moins un vecteur du code d'entrée qui détecte la panne.
Pour un circuit autotestable qui reçoit tous les vecteurs de A, la panne est détectée.

***Définition 'Totalement AutoContrôlable' :*** *Un circuit G est "Totalement AutoContrôlable" pour un ensemble de pannes F, s'il est en même temps "autotestable" et "sûr en présence de défauts" pour ce même ensemble de pannes F.*

***Définition 'Fortement Sûr en Présence de Défauts' (FSPD) [SMI 78] [DAV 78] :*** *Un circuit G est dit "Fortement Sûr en Présence de Défauts" pour un ensemble de pannes F, si pour toute panne f de F, on a:*

*a) Soit le circuit est ""Totalement Autotestable".*

*b) Soit le circuit est "Sûr en Présence de Défauts"; et si une nouvelle panne f ' de F survient, pour la panne résultante on retombe dans le cas a) ou dans le cas b).*

Pour que le but de l'autocontrôlabilité totale soit accompli par un circuit totalement autocontrôlable, une hypothèse doit être assurée, concernant l'occurrence de pannes.

Les circuits "TAC" sont soumis à deux hypothèses fondamentales [SMI 78]:

**$H_0$:** Chaque panne peut être modélisée en une panne appartenant à F.

**$H_1$:** Les pannes apparaissent une par une, et entre l'occurrence de deux pannes quelconques de F, il s'écoule un temps suffisamment long pour que tous les vecteurs du code d'entrée soient appliqués aux entrées du circuit G.

On est amené alors à la propriété suivante:

***PROPRIETE 'But duTotalement AutoContrôlable :*** *Un circuit fonctionnel G, muni des deux hypothèses citées ci-dessus, accomplit le "But du Totalement AutoContrôlable" (but TAC) lorsque la première sortie erronée due au pannes de l'ensemble F se trouve en dehors du code de sortie.*

**Remarques:** **a)** L'hypothèse $H_1$ est peut être un peu plus forte que nécessaire, car souvent, un sous-ensemble convenable du code d'entrée peut tester toutes les pannes détectable

**b)** Cette hypothèse est formulée et spécifiée dans [NIC 84a] sous forme d'hypothèse "$H_2$", elle est nécessaire avec les définitions de blocs

fonctionnels "Fortement Sûrs en Présence de Défauts" (FSPD) et des contrôleurs "Fortement à Code Disjoint" (FCD), pour obtenir des circuits atteignant le "But du Totalement AutoContrôlable" (but du "TAC").

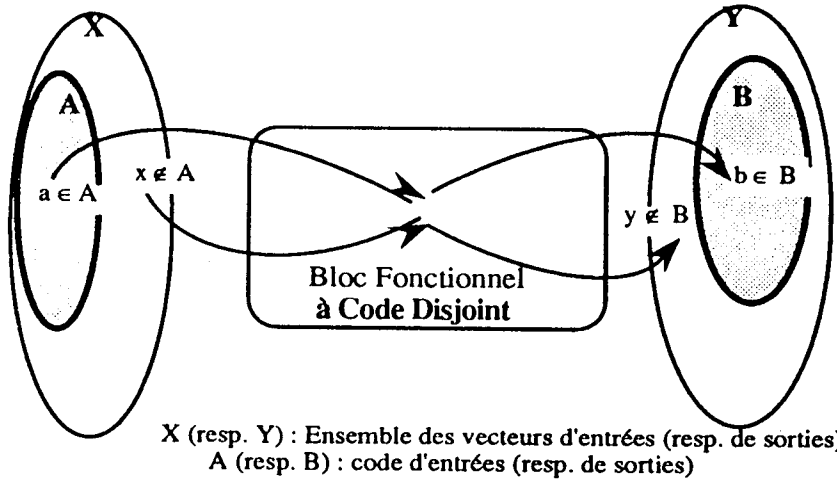
***HYPOTHESE H2:***

*(On rappelle que A est le code d'entrée du bloc fonctionnel; et B le code de sortie de celui-ci, B est ausst le code d'entrée du contrôleur).*

*Après l'occurrence d'une panne affectant le contrôleur, un laps de temps s'écoule suffisamment pour que tous les vecteurs d'entrée B soient appliqués au contrôleur avant qu'apparaisse une nouvelle panne dans le contrôleur ou dans le bloc fonctionnel. De même après l'occurrence d'une panne affectant le bloc fonctionnel, un laps de temps s'écoule suffisamment pour que tous les vecteurs d'entrée A soient appliqués au bloc fonctionnel avant qu'apparaisse une nouvelle panne dans le contrôleur ou dans le bloc fonctionnel.*

***Définition "à Code Disjoint" (CD):*** *Un circuit est à code disjoint s'il transpose un vecteur du code d'entrée en un vecteur du code de sortie, et un vecteur en dehors du code d'entrée en vecteur en dehors du code de sortie.*

$\forall\, a \in A \quad G(a,0) \in B\,; \qquad \forall\, a \notin A \quad G(a,0) \notin B.$




X (resp. Y) : Ensemble des vecteurs d'entrées (resp. de sorties)
A (resp. B) : code d'entrées (resp. de sorties)

**Figure I.8: Circuit à Code Disjoint.**

Un circuit à code disjoint est intéressant à utiliser comme contrôleur: connaissant son code d'entrée et son code de sortie, dès qu'un vecteur en dehors du code d'entrée lui est appliqué il le transpose en un vecteur en dehors du code de sortie. La sortie pourra servir ainsi comme indicateur d'erreur.

Cette propriété est nécessaire mais n'est pas suffisante, car le contrôleur doit aussi signaler ses propres erreurs: la solution la plus simple serait qu'il soit "Totalement AutoContrôlable".

Il est montré dans [NIC 84b] que la propriété "SPD" n'est pas nécessaire pour les contrôleurs et que les erreurs survenant dans le contrôleur peuvent être non détectables à condition que le contrôleur reste à code disjoint. La plus large classe de contrôleurs pouvant être associés à des blocs fonctionnels "FSPD" pour atteindre le "but du TAC", est celle des contrôleurs "Fortement à Code Disjoint" (FCD), [NIC 84b].

**Définition "Fortement à Code Disjoint" (FCD) [NIC 84b]:** *Un circuit G est "Fortement à Code Disjoint" (FCD) pour un ensemble de pannes F si:*

- *Avant l'occurrence d'une panne $f \in F$, G est à code disjoint.*
- *Après l'occurrence d'une panne $f \in F$ on a:*

*a) Soit G est AutoTestable.*

*b) Soit G transpose les vecteurs hors code d'entrée en vecteurs hors code de sortie.*

- *Et si un nouvelle panne $f \in F$ survient, on retombe dans le cas a) ou dans le cas b).*

**I. 6 Règles de conception :**

La détection des erreurs n'est assurée que lorsque la technique utilisée le permet. On adopte pour chaque type d'erreurs (simples, unidirectionnelles, multiples) une technique de codage et de contrôle.

Un ensemble de règles de conception permet de ne pas accroître l'importance de l'occurrence des pannes dans le circuit fonctionnel et dans les blocs de test (générateur de bits de codage et contrôleur). On arrive à limiter l'ensemble des pannes F, et à ne considérer que l'ensemble des pannes les plus probables (Classe d'hypothèses de panne I).

Des règles de conception de circuits FSPD ont été établies dans [NIC 86], pour permettre la détection des erreurs simples, unidirectionnelles, ou multiples. Le but de ces règles est d'arriver à concevoir des circuits FSPD pour la classe I d'hypothèses de pannes.

**I. 6. 1 Règles concernant les erreurs internes simples:**

Pour que le code de sortie détecte les erreurs simples d'un bloc fonctionnel FSPD il faut qu'il satisfasse les règles suivantes [NIC 84a], [CRO 78]:

***Règle R1:*** *Le degré de divergence maximal du bloc fonctionnel, pour l'ensemble de toutes les lignes du bloc est égal à 1.*

Ceci impose que les sorties soient générées par des chemins indépendants, c'est à dire n'ayant aucune ligne en commun.

Des problèmes de conception concernant l'alimentation par exemple surviennent, ces points ont été soulevés dans [CRO 78], des solutions sous forme de règles sont exposées dans [NIC 84a], [NIC 87], elles se résument en une règle générale:

***Règle R2':*** *Chaque ligne d'alimentation du bloc, est utilisée pour alimenter un groupe de portes liées (par des chemins) à une seule sortie primaire de ce même bloc.*

La règle R2' est illustrée par la figure I.9 ci-dessous, et elle ne peut être respectée que si la règle R1 l'est.

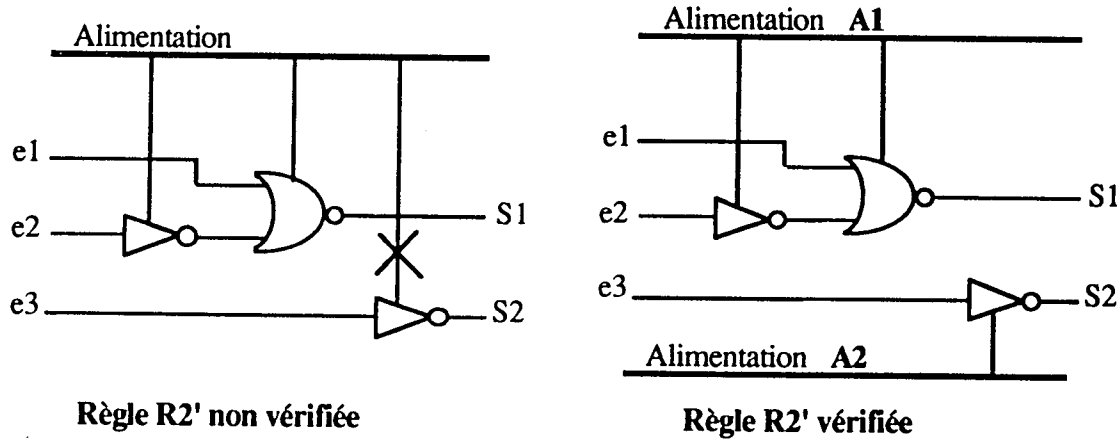



**Figure I.9 : La règle R2'**

Il n'est pas toujours facile de respecter cette règle et de dupliquer ainsi l'alimentation, on aura une augmentation de la surface du circuit, on dégrade aussi ses performances.

La règle R2" permet de remplacer la règle R2' dans certains cas, par exemple lorsque la logique utilisée est non régulière et complexe.

***Règle R2":***

*Chaque fois qu'une ligne d'alimentation ne vérifie pas la règle R2', cette ligne devra servir à alimenter les portes d'un contrôleur (en l'occurrence un contrôleur double-rail).*

La règle R2" est vérifiée, en effet chaque fois qu'il y a une panne qui survient dans la ligne d'alimentation, le contrôleur la détecte et la signale.

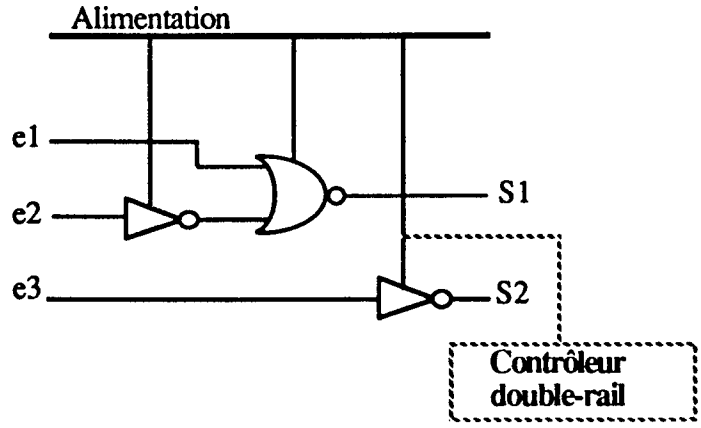



**Figure I.10 : La Règle R2"**

**I. 6. 2 Règles concernant les erreurs internes unidirectionnelles:**

Les erreurs unidirectionnelles sont des erreurs multiples qui peuvent affecter plusieurs sorties mais toutes de la même manière. Ici on aura besoin de la notion de parité d'inversion.

En technologie MOS les portes de base sont des portes NOR, NAND, ou des inverseurs, donc des portes inverseuses.

La règle R4 a été proposée dans [CRO 78] pour assurer la propagation des erreurs simples dans un circuit en erreurs unidirectionnelles à ses sorties primaires.

***Règle R4:*** *Tous les chemins entre une ligne divergente et les sorties primaires d'un bloc doivent avoir la même parité d'inversion.*

La détection de la coupure d'une ligne d'alimentation qui induit une erreur unidirectionnelle interne peut être assurée si l'on s'impose des règles pour l'alimentation des sous-blocs d'un bloc fonctionnel.

***Règle R6':*** *Chaque ligne d'alimentation d'un bloc est utilisée pour alimenter des groupes de portes dont les sorties sont liées aux sorties primaires du bloc par l'intermédiaire de chemins qui ont la même parité d'inversion.*

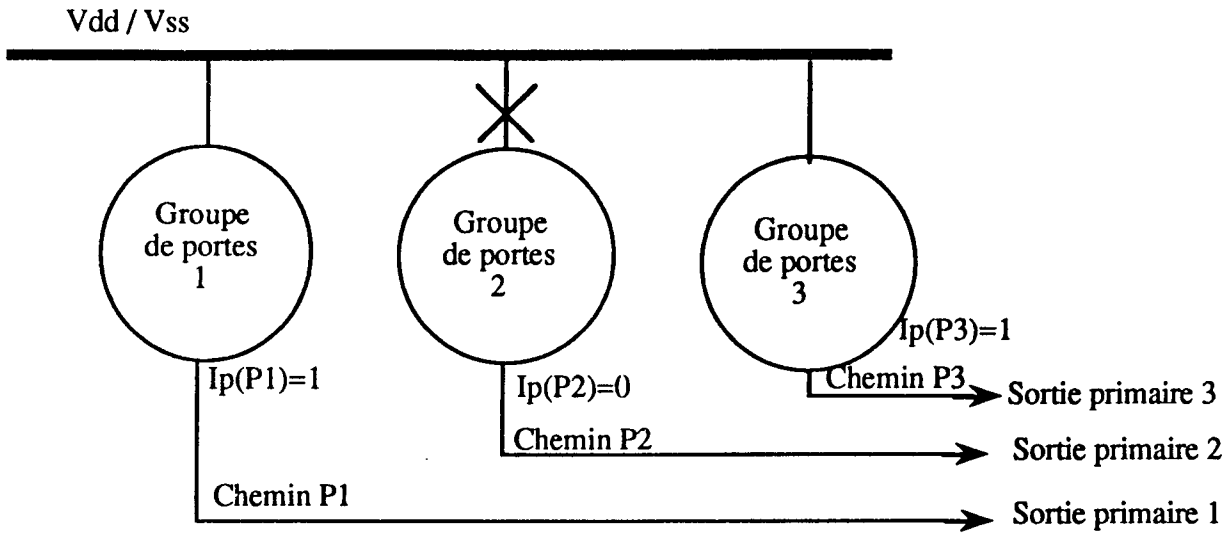



**Figure I.11 : Ligne d'alimentation respectant la règle R6'.**

Si un circuit G est conçu de façon à ce que son code détecte des erreurs unidirectionnelles et s'il respecte les règles R5 et R6' il sera possible de le concevoir "FSPD" pour les erreurs unidirectionnelles, et ce, en respectant d'autres règles détaillées dans [NIC 85b].

Par la suite on aura besoin de définir des PLAs FSPD pour un code détectant les erreurs unidirectionnelles.

Dans la fin du chapitre II figure le schéma d'un PLA: Les lignes d'alimentation de la première matrice alimentent les monômes,(la parité d'inversion des chemins entre les monômes et les sorties primaires est égale à "1"), les lignes d'alimentation de la deuxième matrice alimentent les sorties du PLA ( Les parités d'inversion sont égale à "0")

Il est montré dans [MAK 82], [NIC83], que le code de sortie du PLA doit être un code non ordonné. Un code non ordonné séparable optimal serait le code de Berger. On génère les bits de codage de Berger à des sorties supplémentaires du PLA. Aucune modification de la structure ordinaire des PLAs n'est effectuée, sauf le rajout de lignes des sorties de codage et éventuellement certaines lignes de monômes supplémentaires.

Dans le chapitre IV sera étudié en détail le codage des PLAs.

### I. 7 Codes détecteurs d'erreurs:

Le codage consiste en une redondance de données utilisant des bits de contrôle, appelés aussi bits de codage, avec chaque mot de donnée, procurant ainsi la possibilité de détecter (voire de corriger) les erreurs.

L'exemple de code le plus simple et le plus facile à mettre en oeuvre est la concaténation d'un bit de parité à un mot d'information.

La redondance d'un code est définie par le rapport du nombre de bits de

codage au nombre de bits de l'information réelle. Le meilleur code sera celui qui réalisera la détection de tout type d'erreurs, en nécessitant un minimum de bits supplémentaires.

Dans toute notre étude on adoptera le codage optimal assurant une détection totale du type d'erreur pouvant survenir sur le mot d'information.

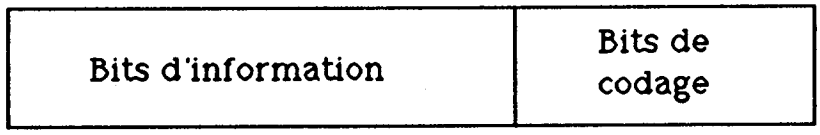
L'origine du type d'erreurs (simple, unidirectionnelle, multiple) est étroitement liée au type de panne considéré et au type d'implémentation du bloc affecté.

Les pannes considérées tout au long de cette étude seront les pannes de la classe I établies dans [COU81]. Et un ensemble de règles de conception a été établit pour permettre la manifestation de l'une de ces pannes en un type bien déterminé d'erreur. Il ne reste plus qu'à coder les mots susceptibles d'être affectés par ces erreurs en un code les détectant.

### I. 7. 1 Codes séparables:

Un code séparable est un code qui est formé de deux groupes de bits séparables:

- Un groupe de bits d'information.
- Un groupe de bits de codage.




### I. 7. 2 Codes non ordonnés:

On adopte un code non ordonné pour détecter les erreurs unidirectionnelles.

***Définition:*** *Un code non ordonné est un code où il n'existe pas deux vecteurs du code tels que l'un couvre l'autre.*

Un vecteur $a=[a_1 \cdots a_i \cdots a_n]$ couvre un vecteur $b=[b_1 \cdots b_i \cdots b_n]$ si $a_i$ prend la valeur "1" sur tous les bits où $b_i$ prend la valeur "1" ($a_i$ et $b_i$ étant les bits de ces vecteurs).

### I. 7. 3 Exemple de Codes: de parité, double-rail, de Berger:

**a) Code de parité:**

***Définition:*** *Un ensemble de configurations binaires de n variables $e_i$ forme un code de parité si la relation suivante est vérifiée:*

$e1 \oplus e2 \oplus \ldots\ldots\ldots \oplus en = a$

a=0 code à parité paire, a=1 code à parité impaire.

Pour un codage de parité des sorties d'un bloc fonctionnel, ($e_1$,$e_2$,.....,$e_n$) forment les bits d'information, a le bit de codage.

On génère "a" par un OU exclusif de tous les bits d'information.

Le code de parité est d'une mise en oeuvre facile et économique du fait de la simplicité des circuits de codage et de contrôle, c'est le code qui entraîne la redondance minimale: (un bit de contrôle). C'est aussi un code séparable.

**b) Codes double-rail :**

Le code double-rail consiste à générer en même temps que les bits d'information leur complémentaires. Un autre type de code double-rail consiste à dupliquer l'information.

(D'autres auteurs parlent de méthode de codage reflex ou duplex).

Ce code est facile à mettre en oeuvre et il est séparable, mais c'est le code à redondance maximale.

**c) Codes de Berger:**

Un code non ordonné séparable optimal serait le code de Berger [BER 61], c'est celui qui nécessite le moins de bits de contrôle pour un nombre donné de bits d'information [CRO 78]. On peut effectuer le codage de Berger de deux façons:

• 1er type de code de Berger: Soient $B_1$, $B_2$, ....., $B_n$ les bits d'information à coder. Les bits de contrôle $B_{n+1}$, ....., $B_{n+p}$ sont obtenus de la façon suivante: on compte le nombre de "1" dans les bits d'information, on lui associe sa représentation binaire qui est $\overline{B}_{n+1}$,....., $\overline{B}_{n+p}$ , il suffit alors d'inverser pour avoir $B_{n+1}$, ....., $B_{n+p}$.

• 2er type de code de Berger: Cette fois ci on compte le nombre de "0" dans les bits $B_1$, $B_2$, ....., $B_n$ , le nombre écrit en binaire correspond à $B_{n+1}$, ...., $B_{n+p}$.

Les deux types de code de Berger sont équivalents lorsque $n=2^p-1$. c'est à dire n=1,3,7,15,31,..... On dit alors que le code de Berger est à longueur maximale ($n=2^p-1$), il est complet puisque les p bits de contrôle prennent les $2^p$ configurations possibles.

Le code de Berger nécessite un nombre de bits de redondance important par rapport au code de parité, mais il présente l'avantage d'être non ordonné et séparable.

Une variante du code de Berger est le code de Berger modifié [MAK 82]. Pour un codage de Berger modifié, $p=\lceil \log_2(m-s+1) \rceil$ où m et s sont respectivement le nombre maximum et minimum du nombre de 1 des mots de sortie du PLA.

$B_{n+1}$, $B_{n+2}$, ...., $B_{n+p}$ est la représentation binaire de la différence entre le nombre de zéros du mot considéré $B_1$, $B_2$, ...., $B_n$ et le nombre minimum de zéros.

Les nombres de zéros maximum et minimum seront respectivement n-s et n-m, ce qui explique qu'on aurait besoin de $\lceil \log_2(((n-s)-(n-m)) + 1) \rceil = \lceil \log_2(m-s+1) \rceil$ bits de codage.

On remarque que l'on a un nombre de bits de codage inférieure à celui utilisé pour un codage de Berger classique. En effet, $m-s \leq n$, implique $\lceil \log_2(m-s+1) \rceil \leq \lceil \log_2(n+1) \rceil$.

En termes de nombre de bits de codage, le code de Berger modifié est performant sinon meilleur qu'un codage de Berger classique.

**d) Code m-parmi-n :**

Ce sont des codes optimaux parmi les codes non-séparables.

Les codes m-parmi-n sont composés de vecteurs où "m" bits ont la valeur "1", et "n-m" bits ont la valeur logique "0".

Le nombre de vecteurs du code correspond au nombre de combinaisons possibles, soit $C_n^m = (n!)/(m!(n-m)!)$; la valeur maximale est accomplie si $m= \lceil n/2 \rceil$, ou $m= \lfloor n/2 \rfloor$.

**e) Remarques :**

**i)** Le choix d'une technique de codage (et donc d'autotest) est conditionné par :

- Le type d'erreurs à détecter.
- La complexité de la mise en oeuvre des différentes techniques.

Le tableau ci-dessous représente la capacité de détection des différentes erreurs suivant le type de codage adopté, [CRO 78], [NIC 84].

| | Simples | Unidirec-tionnelles | Multiples |
|---|---|---|---|
| Code double-rail | * | * | * |
| Code de Berger | * | * | |
| Code de parité | * | | |

**ii)** Le nombre de bits d'information est aussi un critère de choix du type de codage. Si $n_i$ est le nombre de bits d'information, les courbes de la figure I.12 ci-dessous, montrent l'augmentation en pourcentage du nombre de bits de contrôle.

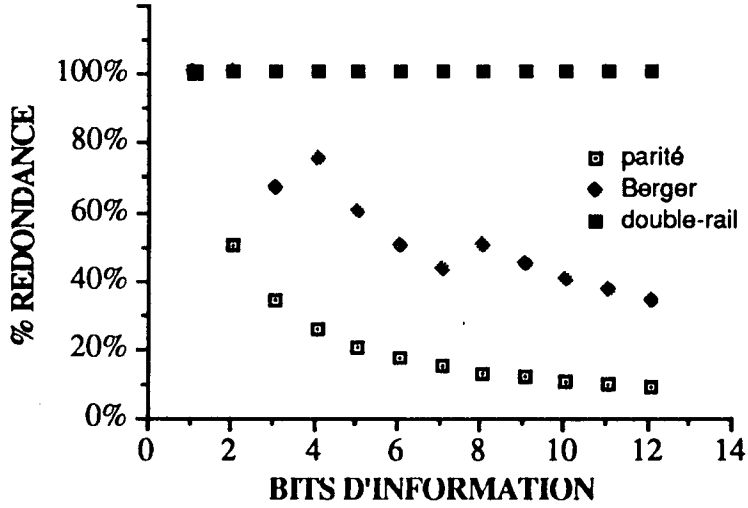



**Figure I.12 : Redondance des bits de codage.**

**iii)** Le choix du code à utiliser ne doit être effectué qu'en fonction de la longueur de l'information à coder et de la nécessité d'introduire ou non des circuits de transcodage.

## I. 8 Génération des bits de codage :

### I. 8. 1 Code de parité :

La génération du bit de parité est simple à obtenir, en effet il s'agit de faire des EXNORs de tous les bits d'information (figure I.13). Le codage nécessite un bit qui prendra la valeur "0" lorsque le nombre de "1" est pair dans les bits d'information et "1" quand il est impair.

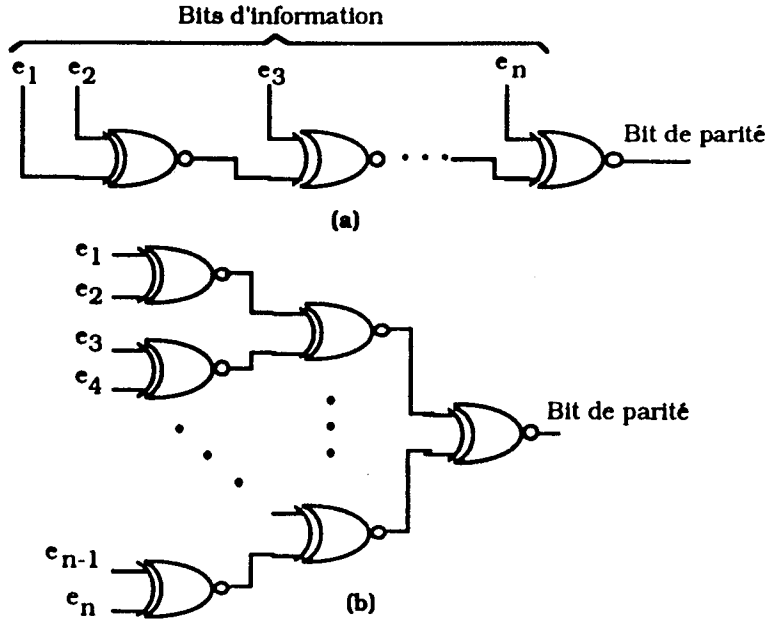



**Figure I.13 : Génération du bit de parité : a) cascade, b) arborescente.**

La concaténation des bits d'information avec le bit de parité, constitue le code de parité. Si n est le nombre de bits d'information le code de parité aura n+1 bits.

**I. 8. 2 Code de berger :**

Dans le cas de code de Berger à longueur maximale($n=2^p-1$), le générateur des bits de contrôle ne comporte que des modules "additionneurs 2 bits", le nombre de modules étant égal à (n-p). La concaténation des bits d'information et des bits de contrôle forme le code de Berger.

Un exemple de réalisation du circuit de comptage pour une information de 7 bits ($n=2^3-1$, $p=3$) est :

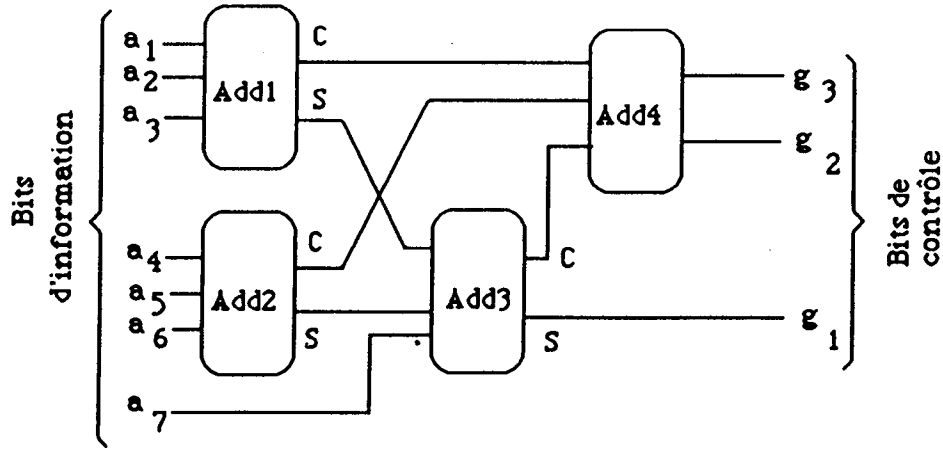



**Figure I.14 : Générateur des bits de contrôle du code de Berger.**

C étant la sortie "carry" pour l'additionneur, S sa sortie somme.

Dans le cas du code de Berger à longueur non maximale, le générateur comporte en plus des demi-additionneurs.

### I. 8. 3 Code double-rail :

Les contrôleurs pour ce code admettent en leur entrées un code double-rail (entrées et leur complémentaires). La réalisation cellulaire de ces contrôleurs permet de regrouper toutes les sorties de ces contrôleurs et d'obtenir à la fin un nombre limité (un ou deux bits) d'indication d'erreur (figure I.15), cette solution augmente considérablement la surface pour un nombre élevé de bits à contrôler.

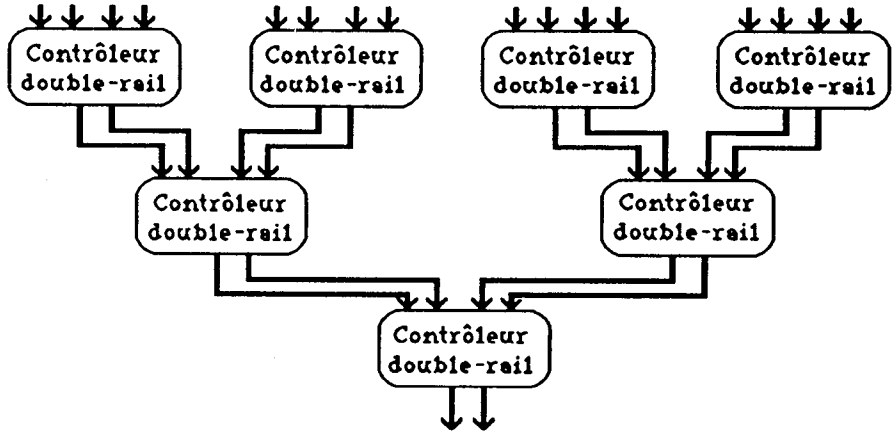



**Figure I.15: Réalisation cellulaire des contrôleurs.**

Plusieurs types de contrôleurs double-rail à sorties doubles existent. La figure I.16 montre le schéma logique d'un contrôleur double-rail, $X^1_1$ et $X^1_2$ représentent les deux bits d'information, leur codage double-rail sont les complémentaires de ces bits et sont représentés par $X^0_1$ et $X^0_2$.

Il est facile de voir que lorsque le code ($X^1_1$ $X^1_2$ $X^0_1$ $X^0_2$) est correct, alors ($f^0$ $f^1$)=(01) ou (10), et lorsque le code est incorrect ($f^0$ $f^1$)=(11) ou (00).

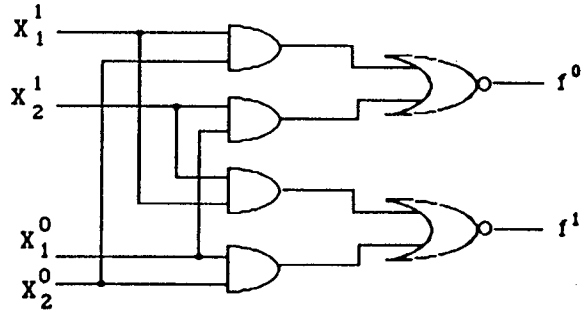



**Figure I.16: Contrôleur double-rail à sorties double.**

Une autre solution est exposée dans [NIC 88], elle permet d'effectuer le contrôle double-rail sur un seul niveau (figure I.17).

Ce type de contrôleur est à sortie simple S. S=1 pour des entrées du code double-rail et S=0 pour des entrées en dehors du code double-rail.

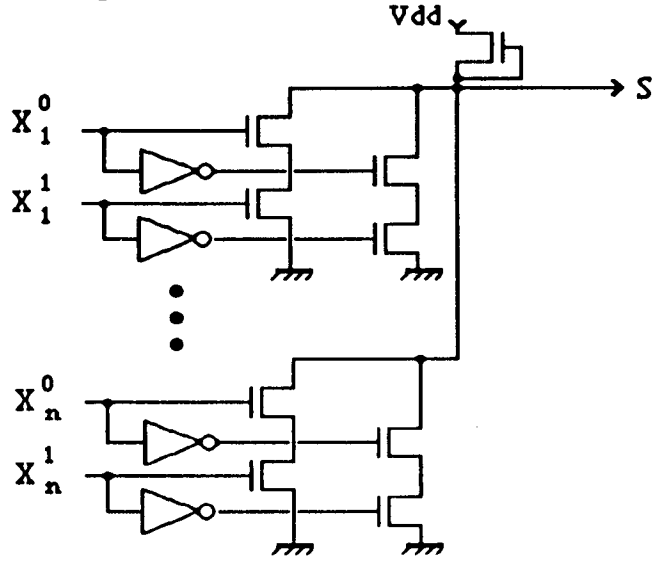



**Figure I.17: Contrôleur double-rail à sortie simple**

La taille d'une cellule de ce contrôleur (deux entrées $X^0_n$, $X^1_n$ et une sortie S) est voisine, sinon plus petite, que celle de deux registres tampon courants. Si les contrôleurs se situent à la sortie des registres, l'assemblage ne posera pas de problèmes de routage et de perte de surface superflue.

**Conclusion :**

Les notions exposées dans ce premier chapitre constituent un ensemble d'outils nécessaires pour le choix d'une technique particulière d'autotest. Le choix s'effectuera sont sa performance, et le but à atteindre est le "But de l'AutoContrôlabilité Totale".

La performance du schéma d'autotest adopté ne doit pas dégrader outre mesure les performances de vitesse et de surface du circuit, et ne doit pas introduire des modifications importantes de l'architecture globale et locale du circuit.

Il est sûr que l'introduction de contrôleurs et la conception SFS des blocs fonctionnels, va augmenter la surface du circuit, et dégrader dans certains cas sa vitesse d'exécution; des compromis sont à faire, et ils dépendent de plusieurs critères dont le type d'information à contrôler et sa longueur, la provenance et la destination de ces informations, ainsi que le type d'erreurs qui peut survenir sur ces informations.

Il n'est pas souvent facile de trouver une solution optimale, on doit cependant respecter l'architecture et la fonctionnalité du circuit tel qu'ils sont définis sans autotest.

Les méthodes d'autotest en-ligne (AutoContrôlabilité) ne sont effectives que si certaines contraintes sont respectées. Dans le cas de réalisation pratiques, le respect de ces contraintes devient difficile, voire impossible, à satisfaire. C'est pour ces raisons que dans les paragraphes II.4 et II.6, sont présentées des techniques conduisant à une conception en vue du test unifié UBIST. Elle consiste à intégrer le test en-ligne et le test hors-ligne dans le même circuit, et elle se fait à des coûts raisonnables, en assurant une très haute qualité pour la plupart des tests utilisés.

# CHAPITRE II

**Modèle de panne, contrôleurs, BILBO, et schéma UBIST**

**Introduction :**

Ce deuxième chapitre présente une analyse sur les modèles de pannes utilisés dans la littérature, et leur application au cas de structures régulières tels que les PLAs. Il est montré que les modèles classiques de collage logique sont insuffisants et inadaptés pour représenter toutes les pannes réelles physiques pouvant survenir dans un circuit.

On propose ensuite des implémentations de contrôleurs. Les contrôleurs sont des organes clés d'un schéma d'autocontrôlabilité. Leur implémentation pose des problèmes cruciaux. Nous proposons des solutions efficaces pour leur implémentation, et nous leur associons des générateurs de vecteurs de test. Ces contrôleurs assurent leur testabilité par une activation de phases de test hors-ligne. Il s'agit des contrôleurs auto-activables.

Nous montrons leur introduction dans un schéma UBIST d'unification du test en-ligne et hors-ligne. Des notions de test hors-ligne sont rappelées, notamment la technique BILBO, qui est facile à mettre en oeuvre.

Le schéma UBIST est présenté, il a été établi dans [NIC 88], et il est pour les circuits auto-contrôlable ce qu'est le schéma BIST pour les circuits conventionnels.

**II. 1 Modèles de pannes:**

Différentes études ont été faites pour modéliser les pannes physiques pouvant survenir dans une technologie donnée, [COU 81], [WAD 78], [GAL 80].

Une conclusion générale de ces études montre que les modèles classiques de collage logique sont insuffisants et inadaptés pour représenter toutes les pannes réelles physiques d'un circuit.

De plus, aucun résultat expérimental n'a été donné pour représenter, par des modèles logiques, les erreurs transitoires ou intermittentes. Ces erreurs sont par leur nature "rebelles" à un test hors-ligne. Il nous faut donc un modèle qui tient compte de ce genre de panne puisque le type de test qu'on utilisera aura une composante en ligne.

Pour des circuits VLSI, l'effet d'une panne simple sur les sorties du circuit dépend des détails d'implantation: si les lignes sont adjacentes, si elles se croisent etc... L'utilisation de structures régulières tels que les PLAs simplifie l'analyse des défaillances et des erreurs à leurs sorties.

**II. 1. 1 Pannes de faible niveau:**

Un transistor manquant a le même effet qu'un transistor qui ne réagit pas à la ligne qui le commande, il "interprète" cette ligne comme collée à zéro quand elle est à 1.

Dans [TAM 84], cette "interprétation" de la ligne est appelée panne du faible niveau de 0 ou du faible niveau de 1.

Un exemple de défaut physique en technologie NMOS pouvant induire ce type de panne serait un dosage incorrect de l'implantation ionique, provoquant un décalage du seuil dans un transistor de charge. La tension résultante est comprise entre les tensions respectives du 0 logique et du 1 logique.

Si la sortance de la porte alimentée par ce transistor est supérieure à 1, les portes alimentées par ce signal peuvent l'interpréter comme un 1 logique, tandis que d'autres l'interpréteront comme un 0 logique.

Si, en un instant donné, la ligne de sortie affectée est supposée être à 1 (resp. 0) logique, mais qu'elle serait interprétée par au moins une des portes comme 0 (resp. 1) logique, cette ligne a une panne de type faible niveau de 1 (resp faible niveau de 0).

En conséquence d'un défaut physique simple (du type classe I [COU 81]), une ligne peut manifester à la fois un faible niveau de 1 et un faible niveau de 0.

Si une ligne est collée à 1, tous les dispositifs recevant cette ligne en entrée l'interprètent toujours comme un 1 logique. Alors que pour une ligne à

panne de faible niveau de 0, il existe au moins un dispositif de ceux connectés à cette ligne l'interprétant toujours comme un 1 logique.

**II. 1. 2 Pannes de coupure :**

Un autre exemple possible de panne de faible niveau de 1 et de 0, est la coupure de lignes. Il a été montré que les coupures de lignes représentent un large pourcentage des défauts physiques survenant dans les circuits VLSI [COU 81] [GAL 80].

Une coupure de ligne donne naissance à deux segments. Un des segments véhicule l'information (e.g connecté à la sortie d'une porte ou chargé par un transistor de charge). L'autre segment est connecté uniquement à des grilles de transistors (ou des entrées de portes), ces transistors voient leur grille devenir "flottante", ils reçoivent une valeur incorrecte de la ligne (0 ou 1).

Une coupure simple sur une ligne peut provoquer un collage à 1 de cette ligne si tous les transistors la mettant à 0 (transistor de décharge) sont déconnectés du reste de la ligne, et un collage à 0 si tous les transistors la mettant à 1 (transistor de charge) sont déconnectés du reste de la ligne.

De plus si quelques transistors la mettant à 1 ou à 0 (charge ou décharge) sont déconnectés de la ligne, la ligne ne peut pas être collée à 0 ou à 1, mais prend la mauvaise valeur de certaines entrées qui commanderaient uniquement les transistors déconnectés (de charge ou de décharge).

Dans les circuits CMOS ou dans les circuits à logique dynamique la coupure d'une ligne (ou un transistor bloqué en permanence) peut rendre la sortie d'un circuit, supposé combinatoire, dépendante de la sortie précédente (mémorisé) plutôt qu'uniquement de l'entrée courante.

Ce type de défaut est appelé couramment stuck-open, il peut échapper à la détection, même si tous les vecteurs d'entrées possibles sont appliqués pour tester le circuit [WAD 78]. Il est montré dans ce cas là que des séquences de vecteurs bien définis sont nécessaires pour détecter cette panne, (voir annexe 1).

**II. 1. 3 Pannes de courts-circuits :**

Les courts-circuits entre lignes adjacentes ou croisées forcent ces lignes à une même valeur, qui peut être comprise entre le 0 logique et le 1 logique. Si deux lignes sont supposées véhiculer des valeurs logiques complémentaires, un court-circuit de ces lignes provoque une panne à faible niveau de 0 sur la ligne supposée être à 0, et une panne de faible niveau de 1 sur la ligne supposée être à 1.

Si le circuit est conçu de façon qu'un court-circuit force toujours une ligne à une valeur définie, cette valeur est déterminée par la ligne "dominante" attaquée par un gros dispositif (ampli, gros inverseur ou porte, ...).

**II. 2 Modèle de panne pour PLAs :**

Dans ce qui suit on discutera l'effet des pannes simples sur une structure NOR-NOR de PLA.

Le modèle élémentaire de défauts utilisé pour les PLAs est formé de 3 types [MAK 82], [OST 79], [WAN 79] :

- Collage d'une ligne d'entrée, ligne de monôme, ligne de sortie.
- Court-circuit entre deux lignes adjacentes ou croisées qui les force à la même valeur logique.
- Transistor MOS de croisement manquant ou parasite dans la matrice ET ou dans la matrice OU.

Les deux premiers types de pannes ont été présentés au (§I.1) et correspondent directement à des défauts physiques dans le circuit.

Le troisième type concerne les défauts physiques tels que leur effet serait équivalent à un transistor MOS de croisement manquant ou parasite. Par exemple, le résultat d'un collage à 0 (resp 1) d'une grille d'un transistor NMOS (resp PMOS) va se traduire par un transistor manquant.

Un transistor de croisement manquant dans la matrice ET est équivalent à une panne de faible niveau à 1 sur la ligne d'entrée correspondante. De même qu'un transistor manquant dans la matrice OU est équivalent à une panne de faible niveau à 1 sur la ligne de monôme correspondante. Il n'est donc pas nécessaire de considérer les pannes de transistor de croisement manquant.

De même, dans le modèle précédent, les trois types de panne ne considèrent pas les pannes de faible niveau et les coupures de lignes, ceux-ci ne sont pas équivalents aux collages logiques.

Comme il a été dit ci-dessus, les pannes de coupure de lignes représentent un large pourcentage des pannes de circuits VLSI. Il est donc clair que le modèle précédent à trois types de pannes ne reflète pas les défauts physiques possibles survenants dans des PLAs à MOS. Par contre, il est sûr que certains effets des coupures survenant dans des circuits à transistors MOS, peuvent ne pas survenir dans les PLAs pour des raisons de structure. Le modèle de panne qu'on serait amené à établir en analysant le

fonctionnement des PLAs après l'occurrence de pannes va être simplifié.

Une coupure de ligne d'entrée provoque un segment "flottant" de ligne, c'est équivalent à une panne de faible niveau à 0 ou à 1. De là, si les pannes de faible niveau à 0 ou à 1 sont considérées, il n'est donc pas nécessaire de considérer des pannes de coupure de ligne.

Les lignes de monômes et de sorties ont un transistor de charge ou de précharge indépendant des entrées du circuit. Chaque point sur une ligne (d'entrée ou monôme) coupée est soit connecté au transistor de charge, soit constamment déconnectée (figure II.1).

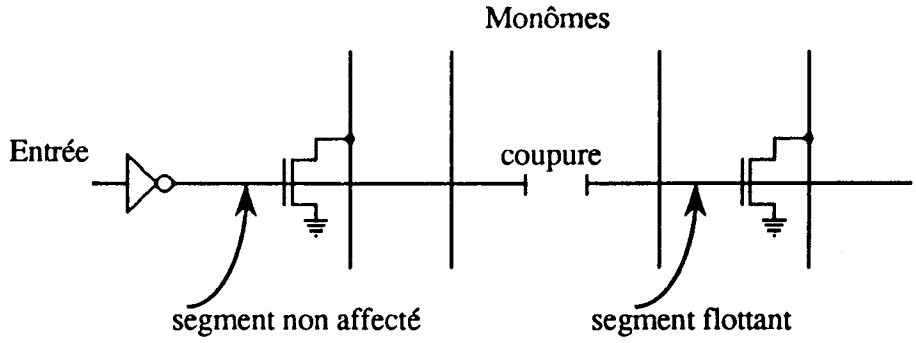



**Figure II.1: coupure de ligne d'entrée.**

Le segment de ligne connecté au transistor de charge est soit mis à la valeur logique 1, soit mis à zéro lorsqu'il est tiré par un transistor de croisement validé par un des vecteurs particuliers d'entrée.

L'autre segment de ligne déconnecté du transistor de charge, va être mis à zéro à la première entrée supposée le mettre à 0. Il restera collé à zéro tout le temps où le circuit n'aura pas été affecté par une autre panne, (survenant plus tard) pouvant le remettre à 1.

Ce type de coupure de ligne n'introduit aucun phénomène de mémorisation d'états, donc n'introduit pas des phénomènes de séquentialité dans le circuit.

Le modèle de pannes de PLAs proposé dans [TAM 84] inclue les pannes de faibles niveaux de 1 et de 0, et les coupures de ligne de monômes ou de sortie:

1) Faible niveau de 1 et/ou de 0 sur une ligne d'entrée, de monôme, ou de sortie.
2) Court-circuit de 2 lignes d'entrée, de monôme ou de sortie adjacentes.
3) Court-circuit de 2 lignes croisées (entrée/monôme) ou (monôme/sortie).
4) Transistor parasite dans la matrice ET ou OU.
5) Coupure d'une ligne de monôme ou de sortie.

Jusque là on a établit un modèle des pannes réelles qui peuvent survenir dans le PLA, sans se préoccuper des conséquences de ces pannes. Il nous faut analyser ces conséquences en les manifestant sous forme d'erreurs détectables.

Une propriété importante de la structure des PLAs (sans considérer les inverseurs d'entrée) est que n'importe quel chemin ayant comme point de départ une entrée et comme point d'arrivée une sortie, a toujours une inversion de parité égale à 1 (une inversion dans la matrice ET, une inversion dans le matrice OU, et une inversion des sorties).

Une panne affectant une ligne de ces chemins peut être localement modélisée en collage, celui-ci fait passer la ligne d'une valeur logique à l'autre, la propagation de cette erreur, jusqu'aux sorties, ne se fait que dans un seul sens (puisque la parité d'inversion est la même pour tous les chemins reliant le point de pannes aux sorties).

On considère les pannes permanentes simples. Toutes les pannes simples du modèle à 3 types de pannes sont des pannes permanentes, exceptés les coupures de lignes d'entrée. En effet, deux phénomènes distincts apparaissent lorsqu'on a une coupure de ligne d'entrée. Le premier concerne l'aspect temporaire du collage du segment isolé. Le second concerne la parité d'inversion des chemins passant par les lignes d'entrée et les lignes d'entrée inversées. Les inverseurs situés sur les entrées primaires du PLA modifient la parité d'inversion. Dans ce cas on est obligé de contrôler les entrées primaires du PLA. (Figure II.2).

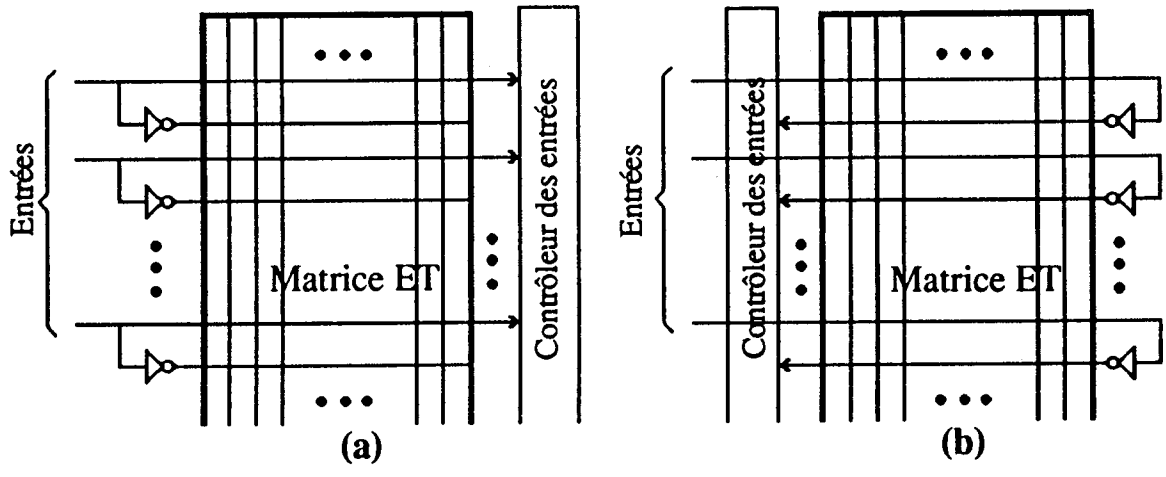



**Figure II.2 : Contrôle des entrées primaires de PLA.**

Le contrôle des entrées primaires peut s'effectuer soit juste avant les inverseurs (Figure II.2 a) et on peut profiter de contrôler les lignes d'entrée, soit après l'inversion (Figure II.2 b) et l'on profite pour contrôler les lignes d'entrée et leur complémentaires.

De toute façon, après avoir contrôlé les entrées primaires, le PLA démuni de ses inverseurs d'entrée produit des erreurs simples ou unidirectionnelles à ses sorties après l'occurrence d'une panne simple.

L'utilisation des règles de conception [NIC 84] [NIC 85] et [NIC 86], permet d'éviter les courts-circuits entre lignes d'entrée et monôme, et lignes de monôme et sortie. Ces lignes sont implantées à des niveaux différents (e.g. ligne d'entrée en polysilicium attaquant les grilles des transistors, et ligne de monôme en métal).

**II. 3 Conception des contrôleurs:**

Malgré les développements très avancés de la théorie des circuits auto-contrôlables, des problèmes cruciaux persistent quand à l'implémentation sur silicium de ces techniques.

Dans cette partie sont présentées différentes implémentations de contrôleurs des codes les plus souvent utilisés dans la conception de systèmes auto-contrôlables. On présentera les principaux contrôleurs classiques associés aux codes de parité, double-rail, de Berger, et Berger modifié.

Les contrôleurs sont des circuits actifs. Un circuit actif d'après [AND 71] est un circuit composé de lignes telles qu'elles prennent les valeurs '0' et '1' pour des entrées différentes appartenant au code. Cette propriété est une condition nécessaire pour la détection des erreurs (mais pas suffisante) permettant l'observabilité du circuit.

Contrairement aux circuits actifs, un circuit passif est défini comme ayant des lignes prenant des valeurs constantes pour les entrées du code et ne les changeant qu'en cas d'erreur.

Les circuits auto-testables ne peuvent pas avoir des lignes qui restent avec la valeur logique constante pendant le fonctionnement normal.

L'efficacité et la puissance des contrôleurs pour des codes détecteurs d'erreurs est souvent recherchée.

Les contraintes qui augmenteraient l'efficacité d'implémentation de contrôleurs et qui doivent être respectées sont :

a) Une conception dans une surface la plus petite possible.

b) Souvent le pas des lignes d'entrée du contrôleur doit être égal au pas des lignes de sortie du bloc fonctionnel, évitant ainsi un routage, et permettant un aboutement sans perte de surface.

c) Une structure régulière du contrôleur permettrait de le générer automatiquement et éviterait un routage interne superflu.

d) Un délai de calcul du contrôleur plus petit ou égal à celui du bloc fonctionnel. Dans ce cas, la vitesse d'exécution sera la même pour le circuit original et auto-contrôlable.

e) Le contrôleur doit être soit auto-testable, soit fortement à code disjoint pour un ensemble de vecteurs survenants lors du fonctionnement normal. Sinon la technique UBIST est adoptée et permettrait d'assurer l'une de ces propriétés en appliquant les vecteurs de test nécessaires lors d'une phase de test hors-ligne.

Les contrôleurs conventionnels pour le code double-rail [CAR 68] ou le code de Berger [MAR 78] ont une structure cellulaire et sont composés d'un grand nombre de petites portes. Leur implémentation implique un routage complexe et gourmand en surface.

Les contraintes a) et b) sont difficiles à respecter, surtout si une technologie CMOS est adoptée, puisqu'en FCMOS on a deux fois plus de transistors et de contacts.

Une logique CMOS dynamique pour l'implémentation de ces portes n'est pas efficace puisque chaque porte doit être préchargée et les sorties des portes doivent être inversées, sinon les signaux d'horloge doivent être retardés entre les différentes couches de portes.

Une autre technique de conception de ces contrôleurs est proposée par la suite. Ces contrôleurs occupent relativement peu de surface et ont un pas d'entrée assez petit et modulable avec les sorties du bloc fonctionnel. Ils peuvent par exemple être aboutés à l'une des matrices d'un PLA, puisqu'ils ont comme lui peu de transistors par pas d'entrée. De plus ils n'ont pas besoin de noeuds internes reliés à l'alimentation.

Finalement, ils ont l'avantage d'avoir une structure régulière permettant leur génération automatique.

Les contraintes a), b), et c) sont alors respectées par ces contrôleurs.

**II. 3. 1 Contrôleur double-rail :**

Une méthode directe pour la construction d'un contrôleur double-rail est de faire le OU-Exclusif de chaque paire de bits (bit et son complémentaire) et de faire un ET logique des résultats obtenus. Un '1' logique à la sortie de la porte ET indique que toutes les paires de bits sont complémentaires, les sorties des portes OU-Exclusif sont toutes égales à '1'.

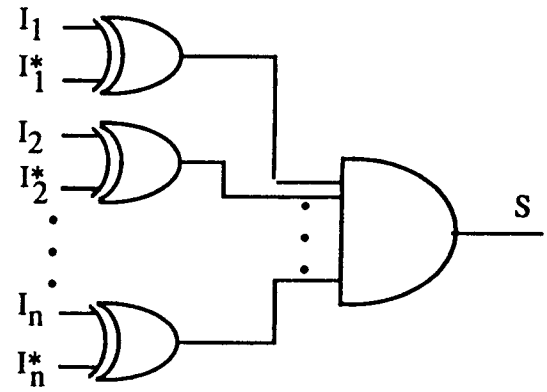



**Figure II.3: Schéma logique d'un contrôleur double-rail à sortie simple.**

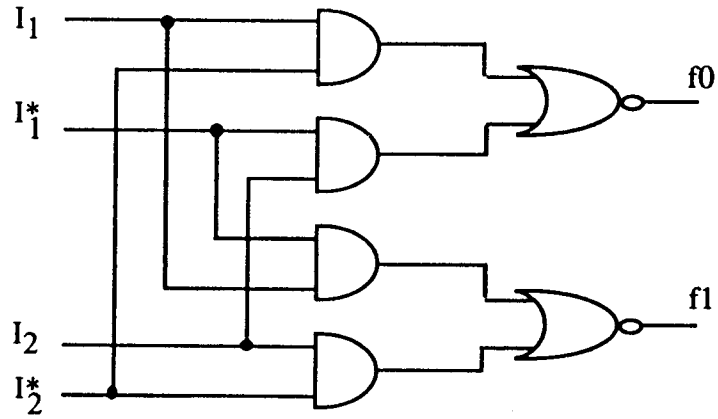



**Figure II.4 : Schéma logique d'un contrôleur double-rail à deux sorties.**

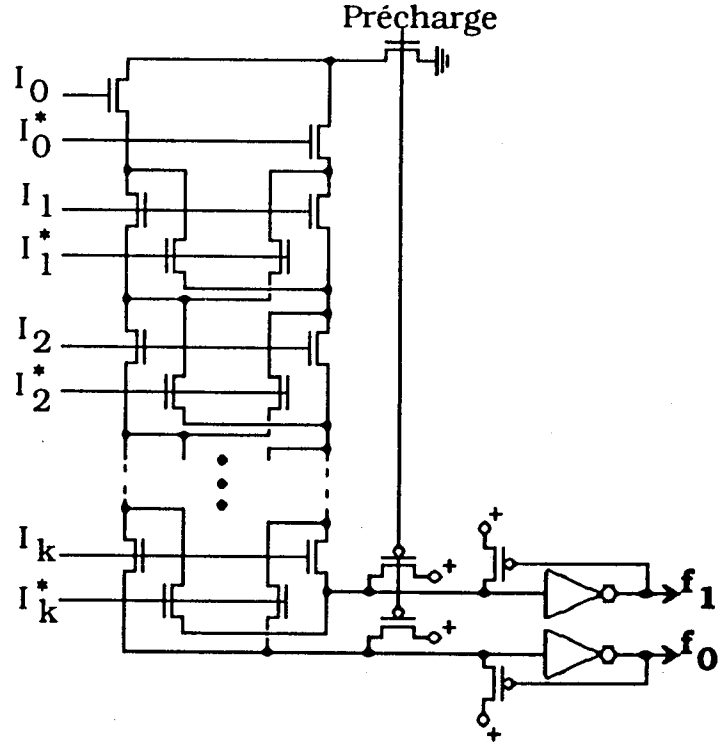



**Figure II.5 : Implémentation CVSL du contrôleur double-rail.**

La conception au niveau transistor du contrôleur double-rail est donnée en figure II.5. Cette structure utilise une implémentation CVSL de portes XOR donnée dans [WES 85].

Si le nombre d'entrées est assez grand, ce contrôleur doit être éclaté en plusieurs petits autres disposés en arborescence, ce qui respecterait la contrainte d).

Finalement, le contrôleur double-rail proposé est auto-testable quand un minimum de 4 mots du code d'entrée sont appliqués à ses entrées (voir annexe 3). La contrainte e) est alors vérifiée.

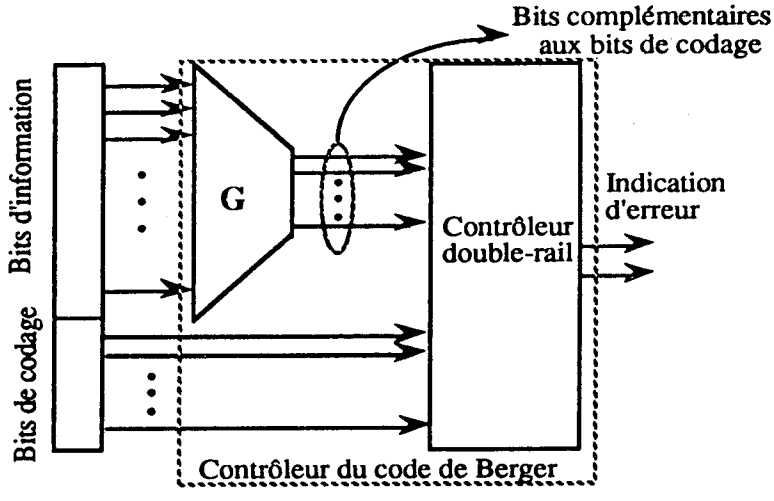
**II. 3. 2 Contrôleurs des codes de Berger:**

Les codes de Berger sont les codes séparables optimaux détectant les erreurs unidirectionnelles [BER 61].

Il est important de trouver des solutions d'implantation de contrôleurs pour ces types de codage.

Les critères de faible surface, grande vitesse de calcul, bonne insertion dans le circuit, propriété d'autotestabilité, sont à vérifier si l'on veut avoir une bonne structure autotestable d'un circuit ou une partie d'un circuit. De plus, une structure modulaire de ces contrôleurs permettrait leur génération automatique.

Les contrôleurs de Berger proposés dans la littérature [MAR 78], [PIE 85], sont basés essentiellement sur des structures de regénération des bits de codage de Berger ou de leur complément et de comparaison avec les bits de codage originaux. De sorte que le contrôleur de Berger est composé d'un bloc combinatoire générant les complémentaires aux bits de codage à partir des bits d'information, et un contrôleur double-rail comparant ces bits aux bits de codage originaux (Figure II.6).




G: Générateur des bits complémentaires aux bits de codage

**Figure II.6: Contrôleur de Berger à regénération du codage.**

Remarque: La structure du contrôleur de la Figure II.6 n'est pas particulière au code de Berger. Tout code séparable (code de parité, code à résidu, code arithmétique, ...) peut voir son contrôleur implémenté sous cette forme.

On proposera plus loin une structure d'implantation qui n'est pas basée sur la structure de la Figure II.6.

**II. 3. 2. 1 Implémentation des contrôleurs du code de Berger:**

Dans le cas du code de Berger à longueur maximale $n=2^P-1$ (n et p étant respectivement le nombre de bits d'information et le nombre de bits de codage), le contrôleur de code de Berger renferme un générateur G formé de modules additionneurs. Leur nombre est n-p [MAR 78].

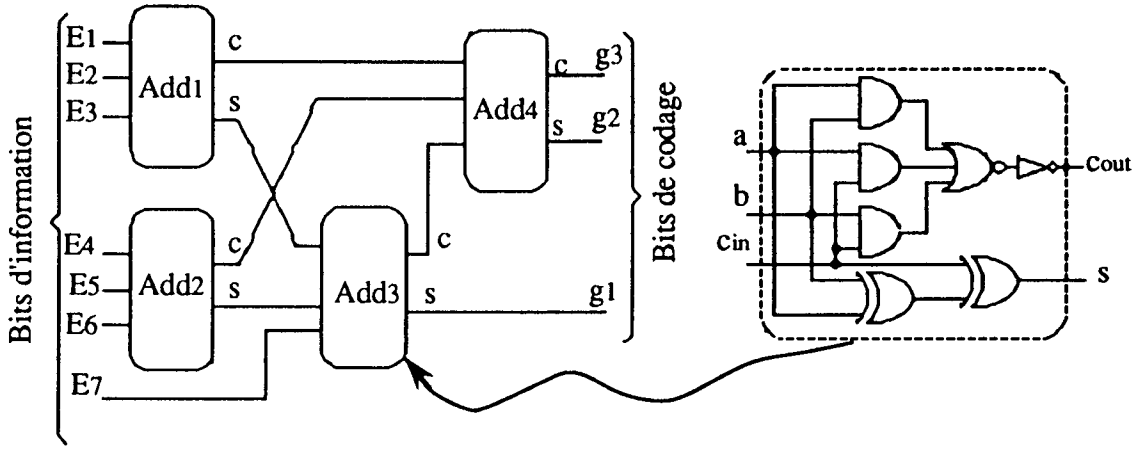



**Figure II.7: Générateur G de la Figure II.6 pour 7 bits d'information.**

Si le module additionneur est implémenté sous forme de porte complexe en pseudo-nMOS inspiré de [GLA 85] (Figure II.8), on a besoin de 14 transistors et deux inverseurs pour les deux sorties de la porte (somme et retenue sortante). Cela nous fait 18 transistors par module additionneur. Le nombre de transistors du générateur G serait 18(n-p).

Un exemple de réalisation du générateur est donné en figure II.7, pour 7 bits d'informations. Pour lequel on aura besoin de 18x(7-3)=72 transistors.

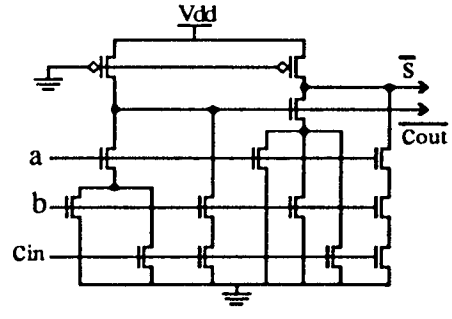



**Figure II.8: Porte complexe additionneur en pseudo-nMOS.**

Le contrôleur double-rail implémenté sous forme CVSL (statique ou dynamique) demanderait (4p-2)+9 transistors pour p paires de bits d'entrée.

Pour p=3 (cas de la figure II.7), le contrôleur double-rail aura 19 transistors. D'où 72+19=91 transistors sont nécessaires pour implémenter un contrôleur du code de Berger à 7 bits d'information et 3 bits de codage.

La solution telle qu'elle est présentée dans [CRO 78] demanderait 92 transistors pour le circuit G et 20 transistors pour le contrôleur double-rail, cela fait en tout 112 transistors pour le contrôleur du code de Berger à 7 bits d'information et 3 bits de codage. Ce qui représente 23% de différence avec le cas précédent.

Dans le cas de code de Berger à longueur non maximale $2^{P-1}-1<n<2^{P}-1$, le générateur sera formé de modules additionneur et demi-additionneur dont le nombre ne peut pas être évalué lorsqu'on utilise la procédure décrite dans [MAR 78].

Le contrôleur du code de Berger que l'on adoptera par la suite est implémenté suivant un autre principe. En effet, au lieu de regénérer les bits de codage ou leur compléments bit par bit, on propose d'assigner un mot I-parmi-(I+1) au mot d'information ayant le même mot de codage. En d'autres termes les mots ayant k zéros k={0, 1, 2 ...., I} ont tous le même mot de codage, on leur fait correspondre un mot I-parmi-(I+1).

k=0 111...10
k=1 111...01
.
.
.
k=I 011...11

Le circuit décodant une telle fonction est proposé en figure II.9, il correspond à une variante du circuit "Tally circuit" proposé dans [MEA 80]. (Il est supposé que les sorties sont préchargées à "1").

Une fois qu'une telle correspondance est effectuée on n'a qu'un bit à 0 et la position de ce bit nous renseigne sur le nombre de "0" existants dans le mot d'information. Il suffit de faire correspondre à ce bit la valeur du mot de codage implémenté en logique arborescente comme en figure II.10 b), ou les bits complémentés au mot de codage implémenté en logique NOR comme en figure II.10 a).

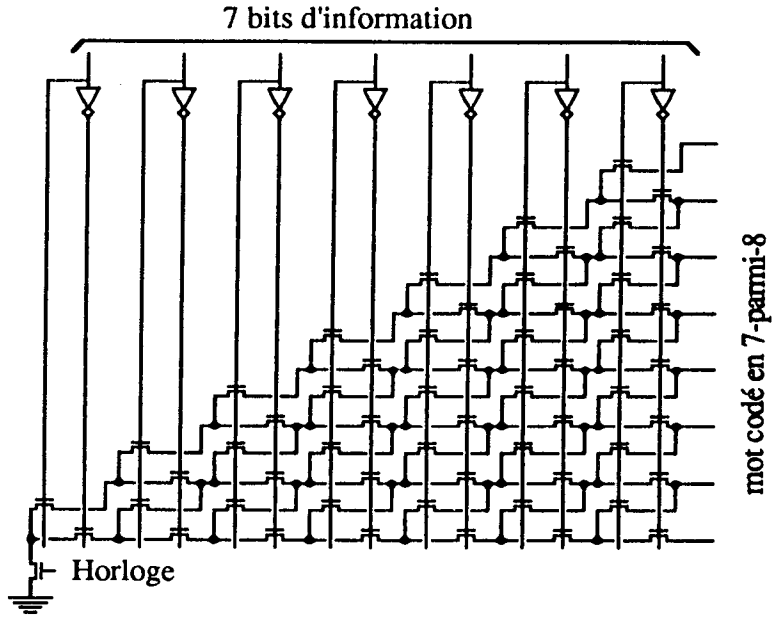



**Figure II.9 : Décodeur (variante du "Tally circuit" [MEA 80]).**

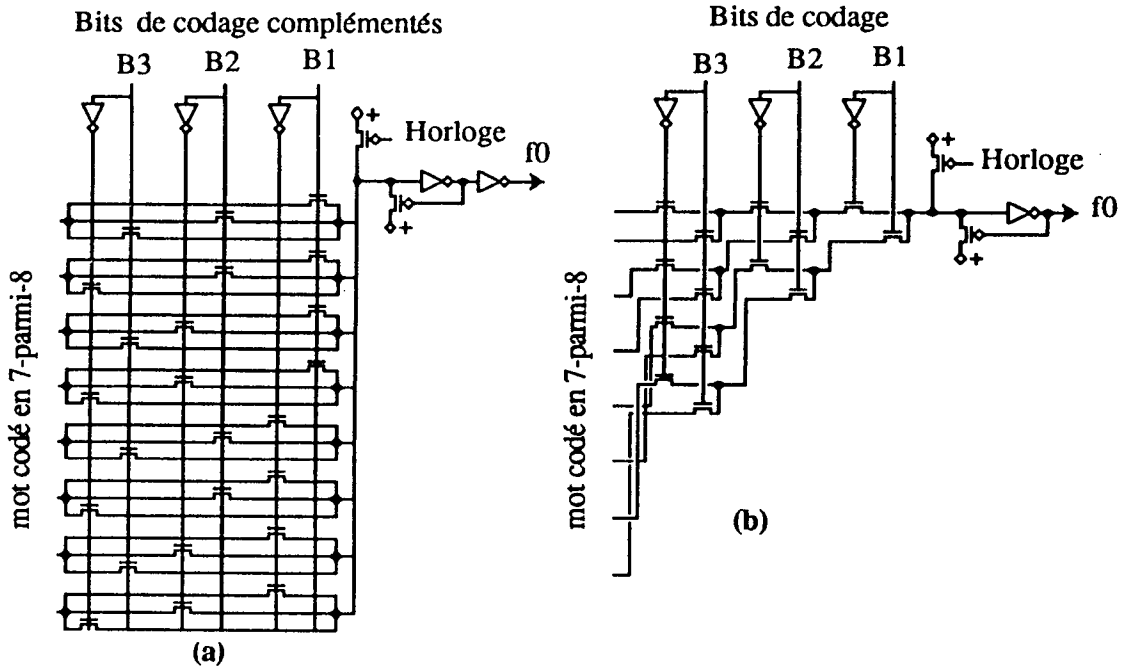



**Figure II.10 : Circuit de contrôle du mot de codage.**

La comparaison entre l'activation due au mot de codage de la ligne correspondante à la position du "0" nous donne une bonne indication d'erreur, (voir figure II.11).

Ce contrôleur a d'autres avantages outre sa simplicité et sa régularité, il nécessite moins de transistors que celui proposé dans [MAR 78].

Par contre ce contrôleur (figure II.11) ne vérifie ni la propriété d'autotestabilité ni la propriété "fortement à code disjoint" quand il reçoit les vecteurs du code d'entrée.

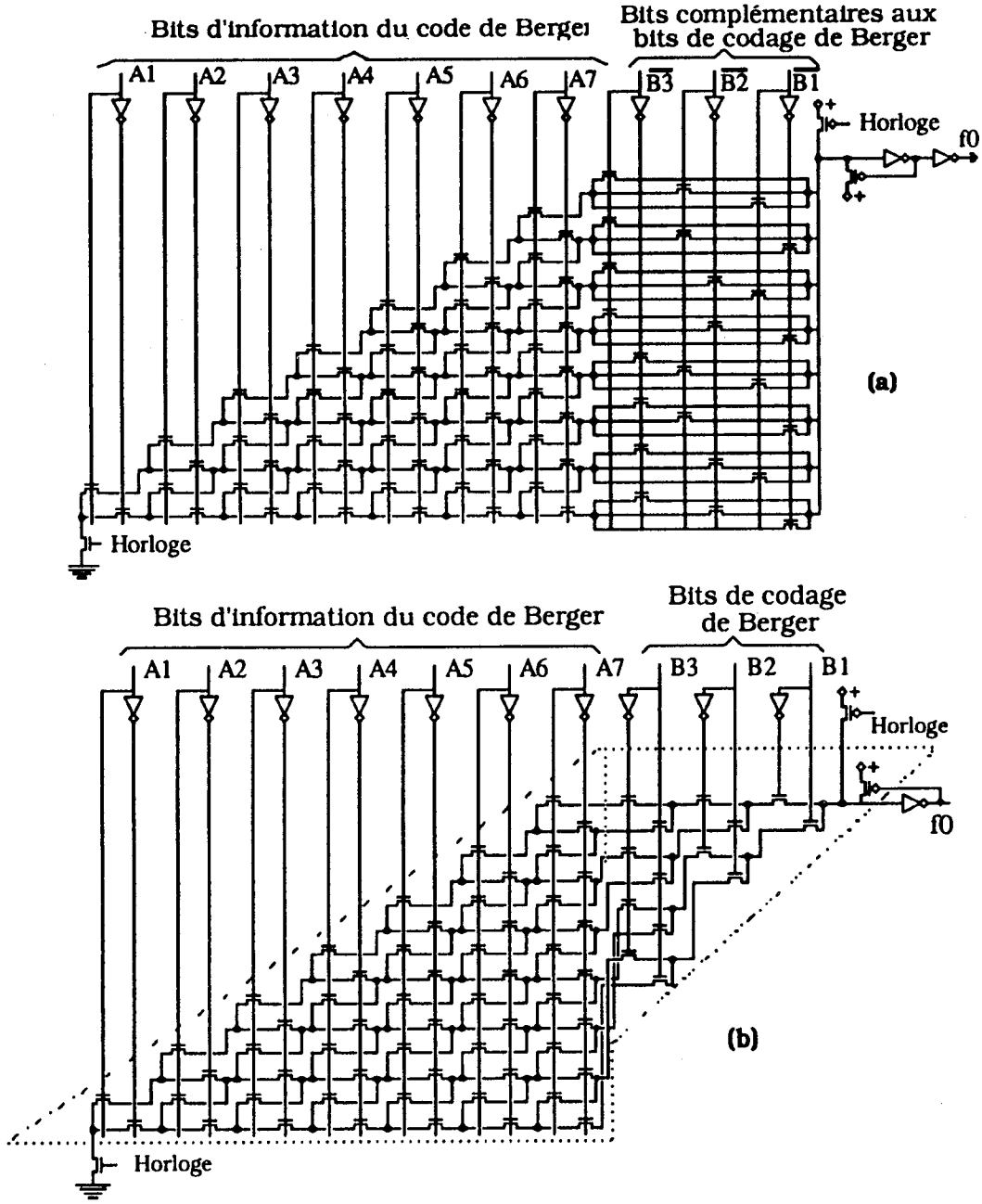



**Figure II.11 : Implémentations du contrôleur de Berger.**
**a) en utilisant une logique NOR.**
**b) en utilisant une logique arborescente.**

Pour vérifier certaines propriétés de testabilité (propriété FCD), on lui associe un générateur approprié de vecteurs de test. Ce schéma est proposé dans [NIC 88] et est présenté en figure II.16. Le générateur est composé d'un bloc générant les vecteurs de test, et d'un bloc vérifiant si le mot appliqué est un mot dans le code ou hors-code. Le contrôleur est activé durant des phases périodiques de test hors-ligne insérées dans le mode de fonctionnement normal.

Finalement, pour le contrôleur du code de Berger la structure proposée en figure II.11b) est adoptée pour respecter les contraintes a), b), et c).

Pour respecter la contrainte d) on limite le nombre des entrées du contrôleur (7 bits d'information et 3 bits de codage). Ceci veut dire que pour un grand nombre d'entrées, on aura à partitionner en groupes de 7 bits d'information, et on associera un contrôleur pour chaque groupe. Un contrôleur double-rail final compactera les réponses de ces contrôleurs.

**II. 4 Conception de contrôleurs auto-activables:**

Les problèmes concernant la testabilité de contrôleurs peuvent être résumés dans les points suivants:

a) Les implémentations de contrôleurs que l'on retrouve dans la littérature sont prévues pour recevoir un ensemble de mots leur permettant de vérifier la propriété TAC ou FCD. Souvent cette condition n'est pas vérifiée dans les circuits réels. De plus, on considère souvent des hypothèses de panne simplifiées pour ces contrôleurs (collage logique).

b) Dans un circuit, si les blocs fonctionnels sont FSPD et les contrôleurs sont FCD, il est nécessaire que durant le fonctionnement normal, les blocs fonctionnels et les contrôleurs reçoivent des ensembles prévus de mots d'entrées du code assurant la propriété TAC.

c) les contrôleurs et les blocs fonctionnels sont conçus pour détecter des pannes simples, mais durant la phase de fabrication, des pannes multiples peuvent survenir. Il est donc nécessaire qu'après fabrication, les circuits auto-contrôlables soient testés pour des pannes multiples. Ce n'est possible que si des techniques spéciales sont adoptées. Par exemple, dans un contrôleur, certaines pannes multiples ne peuvent être détectées que par des mots hors-code d'entrée. Or le contrôleur ne reçoit en fonctionnement normal que des mots du code d'entrée (sorties du bloc fonctionnel).

Pour résoudre ces problèmes, il a été proposé dans [NIC 88] un schéma de génération interne de séquences de test adapté aux circuits auto-contrôlables. Ce schéma UBIST (Unified BIST) est pour les circuits auto-contrôlables ce qu'est le schéma BIST pour les circuits ordinaires (e.g. BILBO [KOE 79]).

Le schéma UBIST (voir figure II.25) est basé sur le concept de contrôleurs Fortement à Code Disjoint (FCD). La propriété FCD pour ces contrôleurs n'est plus alors assurée par la définition donnée dans [NIC 84], mais par l'utilisation de génération interne de séquences de test.

Les contrôleurs FCD définis dans [NIC 84] représentent la plus large classe de contrôleurs permettant d'assurer le but TAC.

Pour un contrôleur, ce qui est important du point de vue sûreté de fonctionnement, est que soit le contrôleur est capable de transposer chaque mot hors-code d'entrée en un mot hors code de sortie ou que la panne dans le contrôleur produit une indication d'erreur.

Pour les contrôleurs auto-activables la propriété FCD est assurée par l'application aux contrôleurs de séquences de test générées dans le circuit durant des phases de test insérées dans le mode de fonctionnement normal.

Dans le cas où un bloc fonctionnel ne génère pas certains mots du code (ce qui peut être dû à la construction du bloc fonctionnel), ces mots du code seront appliqués au contrôleur durant une phase de test hors-ligne.

D'autre part, quand on utilise une génération de séquences de test on peut appliquer des mots d'entrée hors code, ce qui n'est pas possible dans les circuits auto-contrôlables ordinaires. Il est donc facile de vérifier si le contrôleur transpose bien les mots d'entrée hors code en mot de sortie hors code, et la propriété FCD est assurée pour une plus large classe d'hypothèses de panne.

La génération interne de séquences de test résout donc le problème donné en a) ainsi que ceux donnés en b) et c) concernant les contrôleurs.

Le schéma général de contrôleurs auto-activables est donné en figure II.12.

Pendant la phase de test, le générateur de séquences de test génère les vecteurs de test, qui peuvent être dans le code d'entrée comme être en dehors du code d'entrée, et sont appliqués au contrôleur. Le bloc CNCI est un bloc indicateur qui génère un signal S. S prend la valeur "1" si le mot appliqué au contrôleur est dans le code et S=0 si le mot est hors code. Ce signal est utilisé pour inverser ou non l'une des sorties du contrôleur, de façon à garder les deux sorties du contrôleur codées en double-rail, lorsque celui-ci est sans panne.

Si après l'occurrence d'une panne dans le contrôleur, celui-ci donne un mot de sortie hors code pour certains mots d'entrée dans le code, ou donne un mot de sortie dans le code pour certains mots d'entrée hors code, alors le mot de sortie f0 f1 ne sera pas codé en double-rail et la panne est détectée.

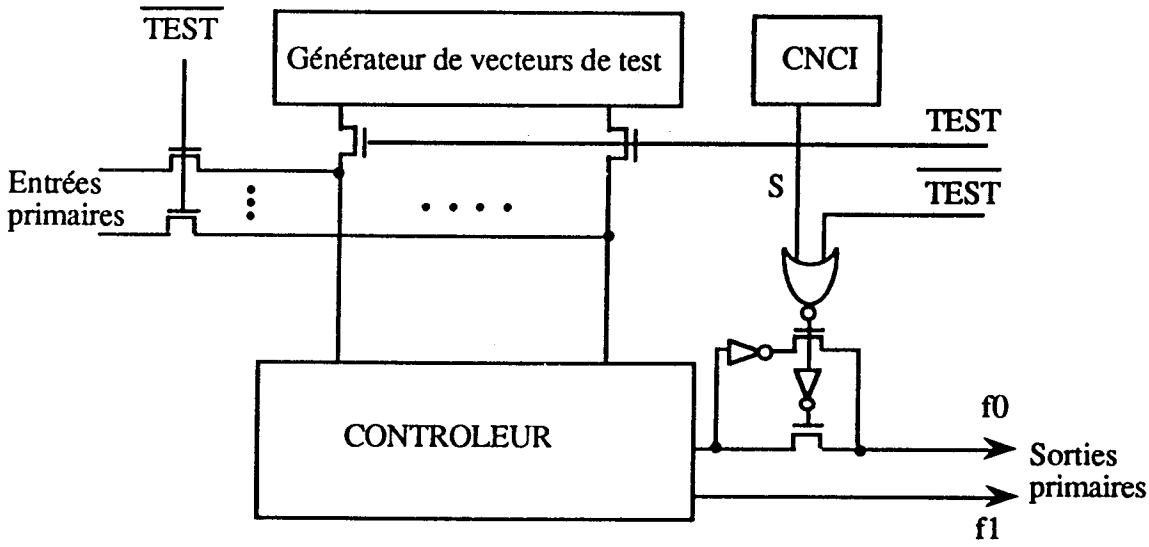



**Figure II.12: Schéma général d'un contrôleur auto-contrôlable.**

Durant le fonctionnement normal, les entrées primaires du contrôleur correspondent aux sorties du bloc fonctionnel, elles sont contrôlées et la sortie f0 du contrôleur n'est pas inversée.

En utilisant le schéma de contrôleurs auto-activables de la figure II.12, on remarque qu'il n'est pas nécessaire d'utiliser des contrôleurs à deux sorties, puisque les mots du code et hors code d'entrée sont appliqués au contrôleur. Si on utilise un contrôleur à une sortie, celle-ci prendra les valeurs "0" et "1", on pourra donc détecter s'il y a une panne ou pas, (il est sûr que l'utilisation de ce contrôleur dans un circuit auto-contrôlable ne pourra pas vérifier la propriété FCD : un collage de sa sortie à une indication de bon fonctionnement ne pourra pas signaler s'il y a erreur ou pas à ses entrées).

Par contre le contrôleur auto-activable global doit avoir deux sorties codées en double-rail f0 et f1 comme présenté en figure II.13. On suppose pour ce schéma que f1="1" pour les mots du code d'entrée et f1="0" pour les mots hors code.

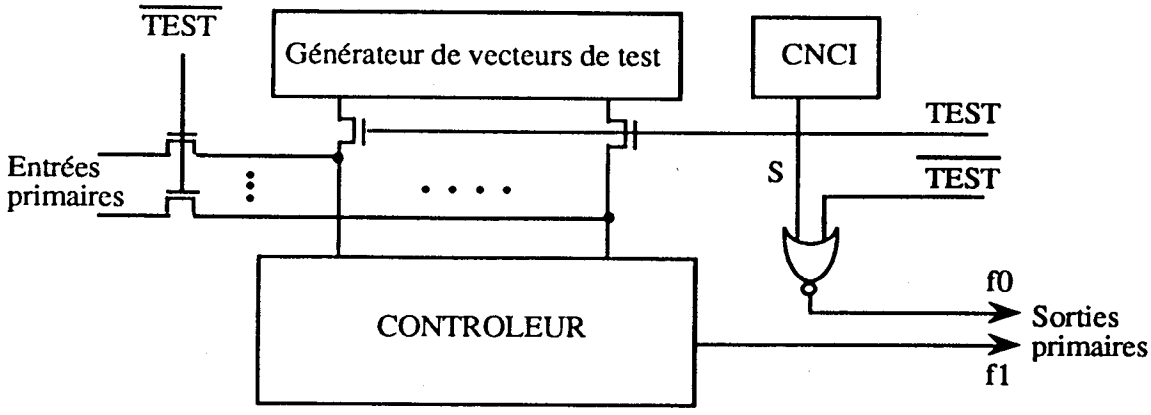



**Figure II.13: Contrôleur auto-activable à 2 sorties codées en double-rail**

Les pannes affectant un contrôleur auto-activable peuvent être détectées soit pendant le fonctionnement normal soit durant la phase de test, la propriété auto-testable et FCD pour de tels contrôleurs sont données comme suit:

**Définition d'auto-testabilité:** *Un contrôleur auto-activable est auto-testable pour un ensemble de pannes F, si pour chaque panne f de F : soit le contrôleur reçoit durant le fonctionnement normal un mot du code d'entrée qui produira un mot de sortie hors code, soit un mot hors code de sortie est produit aux sorties f0 f1 pendant la phase de test.*

**Définition FCD** : *Un contrôleur auto-activable est FCD si : Avant l'occurrence d'une panne il est à Code Disjoint, et pour chaque panne de F on a soit : a) le contrôleur auto-activable est auto-testable.*

*ou b) Le contrôleur transpose toujours un mot en dehors du code d'entrée appliqué à ses entrées primaires en un mot hors code de sortie à ses sorties primaires, et si une nouvelle panne de F survient, on a soit a) ou b) vrai.*

On notera que les contrôleurs auto-activables peuvent assurer la propriété FCD indépendamment des mots du code générés par le bloc fonctionnel et indépendamment du fonctionnement normal.

Pour avoir cette propriété, la propriété d'autotestabilité ne doit pas être basée que sur une détection de panne due aux phases de test.

Quand on applique, durant la phase de test, tous les mots hors code d'entrée, et si une panne survient dans le contrôleur, soit le contrôleur est capable de transposer tous ces mots hors code en mots hors code de sortie, soit la panne est détectée pendant la phase de test. Ce type de contrôleur assure la sûreté de fonctionnement du système pour toute panne combinatoire affectant le contrôleur.

Suite à ces remarques, le théorème suivant [NIC 88] formalise les propriétés d'un contrôleur auto-activable.

**Théorème II. 1:** *Si le générateur de vecteurs de test de la figure II.12 ou figure II.13 génère tous les mots hors code d'entrée, alors le contrôleur auto-activable est FCD :*

*a) pour toute panne affectant le bloc contrôleur, pour toute implémentation de celui-ci, et indépendamment du code de sortie du bloc fonctionnel et du mode de fonctionnement normal.*

*b) pour toute panne affectant le bloc CNCI.*

*c) pour toute panne affectant le générateur de vecteurs de test exceptées celles pour lesquelles:*

*i) L'ordre des mots du code/mots hors code générés par ce bloc n'est pas modifié (un mot du code est remplacé par un autre etc...).*

*ii) Certains mots hors code ne sont pas générés.*

Ce théorème permet d'implémenter les contrôleurs sous une forme intéressante et garantit un taux élevé de couverture de panne affectant celui-ci, indépendamment du type du bloc fonctionnel et du mode de fonctionnement normal.

D'autre part, les pannes affectant le générateur peuvent ne pas être toutes détectées, mais la couverture de panne augmente de manière significative lorsque les contrôleurs auto-activables sont utilisés dans un schéma UBIST où deux vérifications supplémentaires sont utilisées pour ces générateurs.

Des structures générales de contrôleurs auto-activables ont été proposées dans [NIC 88], on expose dans ce qui suit l'implémentation des contrôleurs auto-activables concernant le code de parité, le code de Berger.

**II. 4. 1 Contrôleur auto-activable pour le code de parité :**

On utilise le schéma de la figure II.14 pour générer tous les mots hors code d'entrée (suivant le théorème II. 1).

Le circuit de la figure II.14 est composé d'un LFSR à polynôme caractéristique primitif et générant $2^{n+1}-1$ vecteurs. Les cellules A1, A2, ...,An correspondent au bits d'information et la cellule P génère le bit de parité.

Le LFSR a n+1 cellules pour assurer la génération du mot hors code correspondant au 000...0.

Après chaque décalage la valeur de la cellule P est donnée par l'expression $P(t+1)=P(t) \oplus I_{in}(t) \oplus I_{out}(t)$. Evidemment la valeur de P change chaque fois que la parité de A1, A2, ...,An change. Et si l'entrée A0 est prise comme valeur de $I_{in}$ alors P va calculer la parité des cellules A0, A1, A2, ...,An. Le type de parité (parité paire ou impaire) est défini par l'initialisation du circuit : Si P est initialisé à la valeur impaire de la parité pour les valeurs initiales de A1, A2, ...,An, alors le circuit va générer les mots du code pour une parité impaire et un mot hors code pour une parité paire. De la même manière on peut générer des mots du code pour une parité paire et des mots hors code pour une parité impaire.

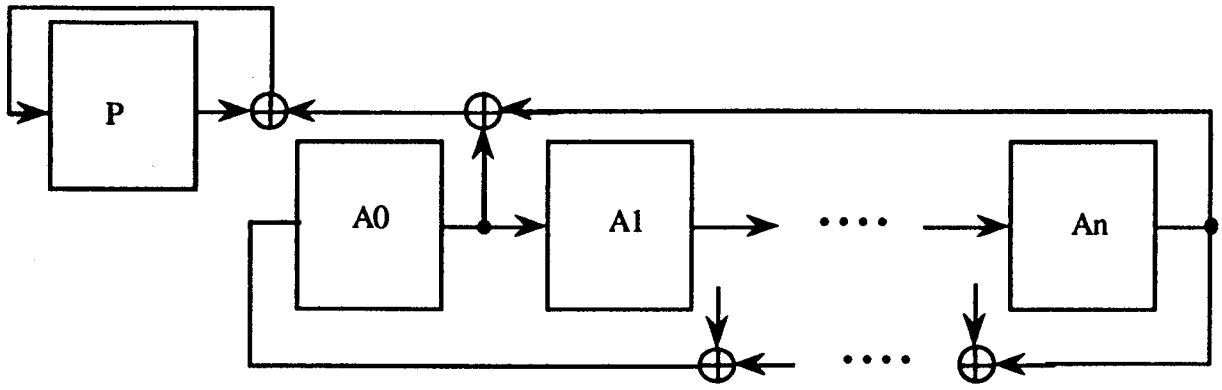



**Figure II.14 : Génération complète ($2^n$) de mots hors code de parité (ou mots du code)**

On assure ainsi la génération de tous les mots hors code d'entrée et la couverture de panne est donnée par le théorème II.1.

Durant la phase de test, on n'applique que les mots hors code au contrôleur. Le problème de distinguer les mots du code des mots hors code n'a pas lieu. Une des sorties du contrôleur est inversée pendant la phase de test.

Il est à noter que le schéma de prédiction de la parité pour le LFSR de la figure II.14 peut être utilisé pour une prédiction de parité dans le cas d'une analyse de signature série. La valeur de la cellule P devient : $P(t+1)=P(t) \oplus I_{in}(t) \oplus I_{out}(t)$, avec $I_{in}(t)$ l'entrée de la première cellule et $I_{out}(t)$ la sortie de la dernière cellule.

Dans le cas d'analyse de signature parallèle, un schéma similaire peut être utilisé où P devient $P(t+1)=P(t) \oplus I_{fd}(t) \oplus I_{out}(t) \oplus P_d(t)$, où $I_{out}$ est le même que précédemment, $I_{fd}$ est la valeur calculée par le rebouclage du LFSR et $P_d$ la parité de la donnée entrée en parallèle dans le LFSR.

Ces schémas combinés avec un contrôleur de parité donnent un analyseur de signature TAC.

### II. 4. 2 Contrôleur auto-activable pour le code de Berger :

Le code de Berger est un code systématique. Si un contrôleur de code de Berger reçoit tous les mots d'information d'entrée, alors le contrôleur est FCD pour toute panne affectant le générateur des bits de contrôle.

En figure II.15 est présenté le schéma d'un LFSR modifié générant les mots de code de Berger. Le LFSR génère les bits d'information et le compteur UP/DOWN a k cellules ($k=\lceil Log_2(k+1) \rceil$.

L'initialisation du circuit est donnée de telle façon que le LFSR contient un vecteur différent de 000...0 et le compteur contient les complémentaires des bits de codage pour ce vecteur.

L'horloge du compteur UP/DOWN est égale à $(A1_{in} \oplus An_{out}) \wedge \Phi$. Quand $A1_{in} \oplus An_{out} = 0$, le nombre de "1" n'est pas modifié dans le LFSR et le compteur ne compte pas (horloge=0). Quand $A1_{in} \oplus An_{out} = 1$, le nombre de "1" dans le LFSR a augmenté de 1 si $A1_{in} \oplus An_{out} = 0$ (le compteur est incrémenté), et si $A1_{in} \oplus An_{out} = 1$, le nombre de "1" dans le LFSR a diminué (le compteur est décrémenté). Le compteur UP/DOWN contient donc toujours les complémentaires aux bits de codage correspondants au mot d'information contenu dans le LFSR.

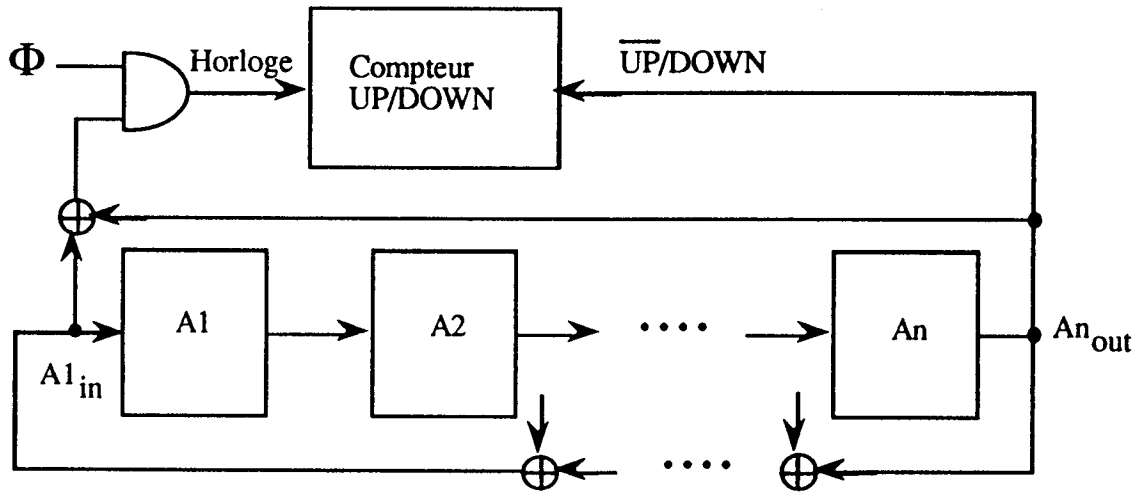



**Figure II.15 : Générateur LFSR avec codage de Berger.**

Dans le cas du générateur de la figure II.15, on n'a pas besoin d'avoir un signal indiquant si le mot est du code ou hors code, seuls les mots du code sont générés.

Un autre schéma (Figure II.16) peut être utilisé, où tous les mots du code et tous les mots hors code sont générés et un dispositif (CNCI) génère un signal pour distinguer si le mot est dans le code ou pas. Tout contrôleur de Berger associé à ce dispositif devient FCD d'après le théorème II.1.

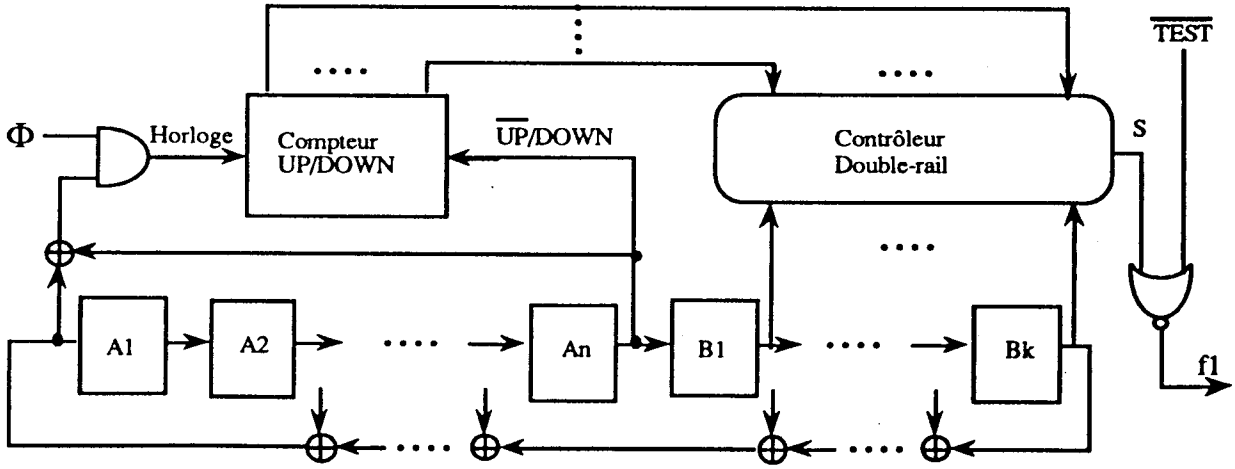



**Figure II.16 : Bloc générateur de mots du code et hors-code de Berger.**

Le dispositif CNCI est formé par un compteur UP/DOWN, un contrôleur double-rail à k-pair d'entrées et une porte NOR.

Lors de la génération des séquences dans le LFSR, la partie bit d'information et compteur UP/DOWN contient toujours un mot dans le code (c'est le schéma de la figure II.15), mais les bits de codage contenus dans B1, ...., Bk peuvent correspondre ou pas, car le LFSR génère des mots du code et des mots hors code. Un contrôleur double-rail est utilisé pour indiquer si les mots contenus dans B1, ...Bk sont les bits de codage de A1, ...An ou pas.

En prenant comme contrôleur de Berger le schéma de la figure II.11 b) f0=1 (resp. 0) lorsque le mot est dans le code (resp. hors code), et en utilisant un contrôleur double-rail à sortie simple S qui prend la valeur "1" (resp. "0") pour un mot dans le code (resp. hors code), on a résumé en table II.1 le fonctionnement du générateur de la figure II.16 :

• lors de la phase de test ($\overline{TEST}$=0), f1 prend les valeurs inverses de S, d'où (f0, f1) est toujours codé en double-rail.

• lors du fonctionnement normal ($\overline{TEST}$=1), f1 prend toujours la valeur 0 quelle que soit la valeur de S. Quand le mot est dans le code (f0, f1)=10 (codé en double-rail), et quand le mot est hors code (f0, f1)=00 (indiquant qu'il y a erreur).

| | $\overline{TEST}$ | S | f1 | f0 | |
|---|---|---|---|---|---|
| phase de test | 0 | 0 | 1 | 0 | hors-code |
| | 0 | 1 | 0 | 1 | code |
| fonct. normal | 1 | X | 0 | 0 | hors-code |
| | 1 | X | 0 | 1 | code |

**Table II.1 : Table de fonctionnement du générateur de la figure II.16.**

L'avantage du schéma de la figure II.16 est qu'on génère pendant la phase de test tous les vecteurs aux entrées du contrôleur de Berger.

La couverture de panne est de 100% pour :

• Toute panne combinatoire (simple ou multiple) affectant le contrôleur.

• Les pannes affectant le LFSR sont détectées, car elles modifient le nombre de 1 dans les cellules A1, ..., An, d'où le nombre de "1" contenu dans le compteur UP/DOWN ne correspond plus au nombre de "1" contenu dans les cellules A1, ..., An, et on détecte ces pannes.

• Les pannes affectant le compteur et le contrôleur double-rail.

Il est à noter finalement que le schéma de la figure II.16 peut être utilisé pour un contrôleur de Berger classique [MAR 78] (composé d'un générateur de bits de codage et d'un contrôleur double-rail à sortie simple), comme il peut être utilisé pour le contrôleur de Berger proposé en figure II.11, ou tout autre implémentation de ce contrôleur

**II. 5 Schéma de test à analyse de signature:**

Le test par analyse de signature consiste à appliquer une séquence de vecteurs de test en entrée d'un circuit, et plutôt que de vérifier toute la séquence de vecteurs de sortie, la signature de cette séquence est analysée. Il s'agit d'un compactage de la séquence des vecteurs de sorties.

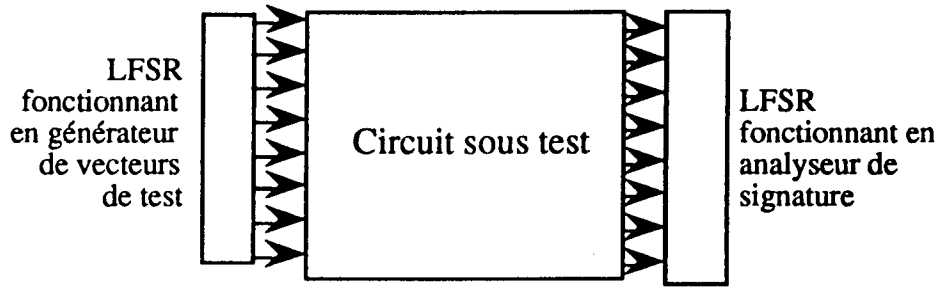



**Figure II. 17 : Schéma général de test par analyse de signature.**

La figure II. 17 donne un schéma de test par analyse de signature, où le circuit sous test reçoit une séquence de vecteurs générés par un LFSR, et les réponses du circuit sont envoyées à un autre LFSR effectuant une division polynômiale de ces réponses, le reste de la division représente la signature.

**II. 5. 1 Génération pseudo-aléatoire et exhaustive de vecteurs de test :**

Un générateur pseudo-aléatoire génère des séquences reproductibles qui ont des propriétés des séquences aléatoires. On le réalise à l'aide d'un LFSR. Les principes mathématiques à la base des méthodes de génération pseudo-aléatoire et de compaction exploitées dans les dispositifs de type LFSR peuvent être trouvés entre autres références dans [CLA 77].

On présentera à partir d'un dispositif matériel (LFSR) les principes de base.

Soit l'exemple de la figure II. 18.

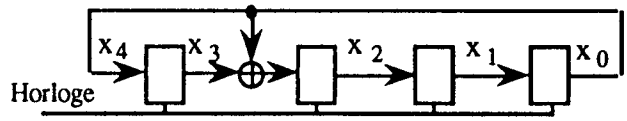



**Figure II.18 : LFSR à polynôme primitif pour 4 bits.**

L'horloge définit les instants 0, 1, 2, ...., t-1, t, t+1, ... et xi(t) indique la valeur de xi au temps t, on a:

$$x_0(t+1)=x_1(t)$$
$$x_1(t+1)=x_2(t)$$
$$x_2(t+1)=x_3(t)\oplus x_0(t)$$
$$x_3(t+1)=x_4(t)$$

$$X(t+1)=M.X(t) \qquad M=\begin{bmatrix}0&1&0&0\\0&0&1&0\\1&0&0&1\\1&0&0&0\end{bmatrix}$$

$M^{15} = I$, la période de la suite des valeurs $x_0$, $x_1$, $x_2$, $x_3$, $x_4$ est 15, ce qui correspond à la période maximale possible, la valeur (0, 0, 0, 0) étant une valeur "puits".

Les valeurs propres de M se retrouvent en calculant le déterminant de M-xI. Sachant que sur {0, 1}, les opérations + et - sont équivalentes, on trouve pour l'exemple ci-dessus $1+x+x^4$. Ce polynôme est appelé polynôme caractéristique du LFSR. On retrouve les coefficients de rebouclage du LFSR.

Dans le cas général on représente le LFSR à n étages du type standard comme en figure II. 19.

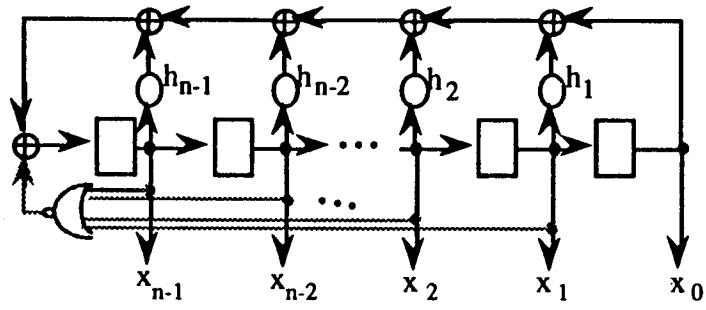



**Figure II.19 : LFSR standard.**

L'expression générale du polynôme caractéristique est :

$f(x) = 1 + h_1 x + h_2 x^2 + \ldots\ldots + h_{n-1} x^{n-1} + x^n$

Si $h_i = 1$, il y a connexion (et donc rebouclage), si $h_i = 0$ il n'y a pas de connexion.

Si le polynôme caractéristique est primitif (irréductible), alors le LFSR engendre une séquence maximale $2^n-1$, après $2^n-1$ coups d'horloges la séquence se répète.

Il est possible d'utiliser un LFSR pour engendrer une séquence série en Q0, ou en exploitant les sorties Q des différentes bascules D en parallèle. On réalise alors un générateur pseudo-aléatoire pouvant délivrer des vecteurs de n bits de large comme stimulis sur un dispositif à tester à n entrées.

Il faut toutefois remarquer que le vecteur "tout à zéro" (000...00) est manquant. Ceci sera gênant si ce vecteur est indispensable pour la détection de certaines pannes. En grisé est représenté sur les figures II. 19 et II.20 la circuiterie permettant de générer le vecteur (000 ... 00), le LFSR ainsi modifié devient capable de générer $2^n$ vecteurs.

En annexe 4, est donnée une liste de polynômes primitifs [WAN 86].

L'intérêt du LFSR est sa relative simplicité. En terme de surcoût en surface, on a intérêt à choisir pour une longueur donnée un polynôme primitif renfermant un nombre minimum de portes OU-exclusif.

Une autre implémentation du LFSR est possible en insérant les

rebouclages entre les bascules, il s'agit du LFSR modulaire représenté en figure II. 20, pour laquelle la même expression du polynôme caractéristique est valable.

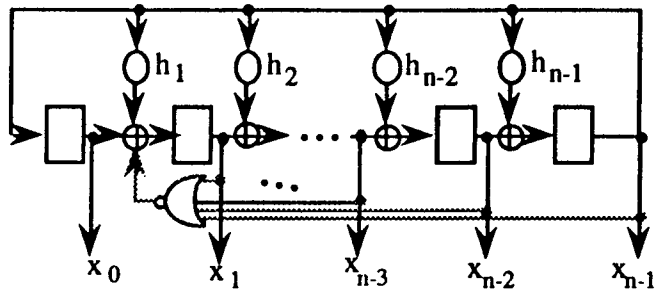



**Figure II. 20 : LFSR modulaire.**

**II. 5. 2 Analyse de signature :**

Un LFSR utilisé en analyseur de signature série (figure II. 21) ou parallèle (figure II. 22), effectue une division polynômiale de la réponse de la logique testée.

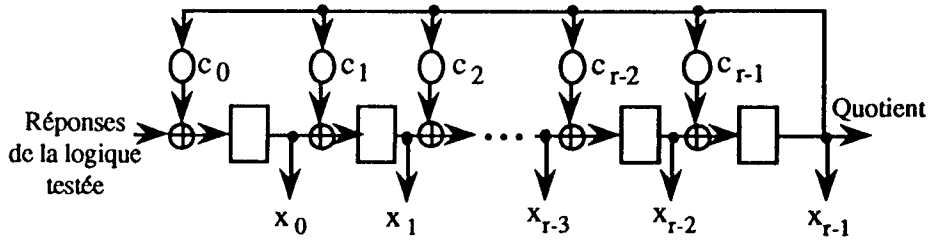



**Figure II. 21 : Analyseur de signature série.**

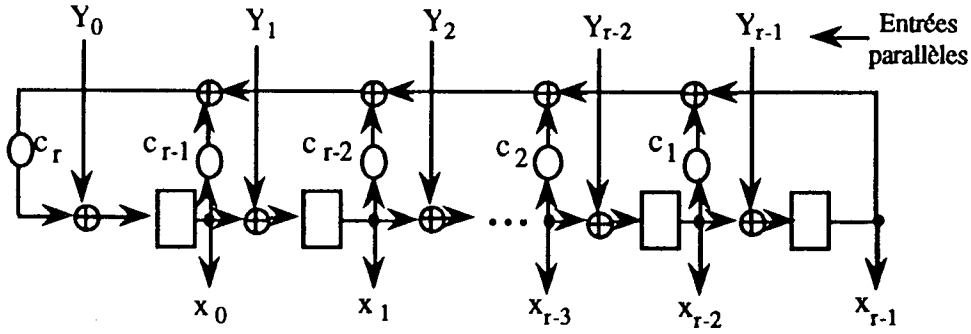



**Figure II. 22 : Analyseur de signature parallèle.**

A une séquence en sortie de la logique testée est associé un polynôme C(x) (diviseur). L'opération donne un quotient Q(x) et un reste R(x).

La division est fonction de la séquence traitée : le polynôme P(x), et du polynôme diviseur C(x).

La signature est constituée par le reste de la division, présentée en sorties des différentes bascules du registre.

Si un résultat correct est représenté par P(x), et si le résultat obtenu

avec un circuit donné est P'(x), on appelle polynôme d'erreur la différence entre P(x) et P'(x): E(x)= P(x) + P'(x).

E(x) est un polynôme du degré de P(x) dont les coefficients égaux à 1 correspondent aux bits incorrects de P'(x).

Si le circuit testé est correct, E(x)=0, mais certains circuits défectueux (E(x)≠0) peuvent donner la même signature que le circuit correct, c'est le problème de masquage ("aliasing") d'erreurs.

Le taux de masquage dans l'hypothèse d'une distribution uniforme d'erreurs a été étudié dans [SMI 80], [CAR 82], [DAV 80], il est de l'ordre de $2^{-n}$ où n est le degré du polynôme diviseur.

Des études plus récentes pour d'autres hypothèses de distribution d'erreurs donnent des taux de masquage d'erreurs similaires, [DAV 89], [WIL 86], [DAS 90].

**II. 5. 3 Technique BILBO :**

On a pu observer l'analogie entre le générateur de vecteur pseudo-aléatoire à LFSR et un analyseur de signature également réalisé par un LFSR. Dans [KOE 79] a été proposé l'idée de fondre en un seul ensemble le générateur de séquences et l'analyseur de signature selon le schéma appelé BILBO (Built-In Logic Block Observation).

Un exemple de registre BILBO à 8 bits pour un polynôme particulier est représenté en figure II. 23.

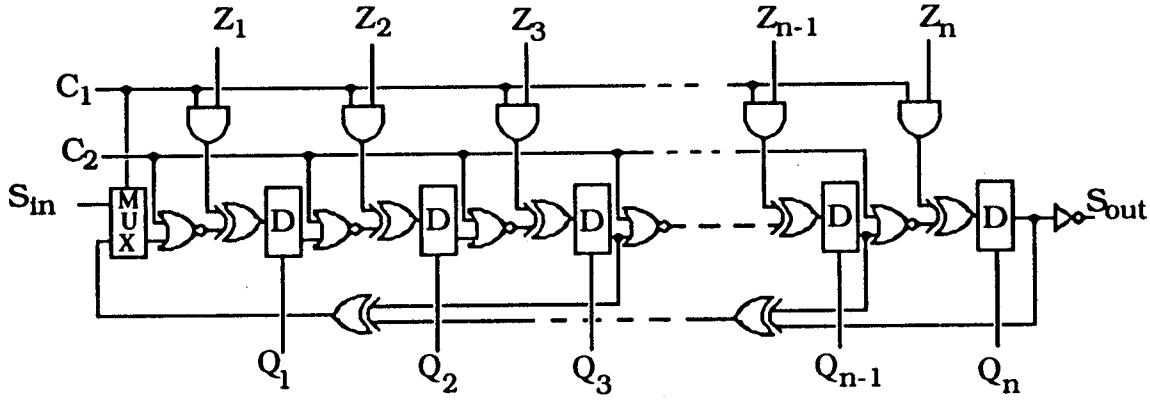



**Figure II. 23 : Schéma registre BILBO.**

Un tel registre fonctionne en différents modes selon les valeurs données aux commandes C1 et C2.

• Si $C_1$ $C_2$=11, le registre est un registre normal composé de 8 bits en parallèle (figure II. 24 a)).

• Si $C_1$ $C_2$=00, les bascules sont montées en série, le registre se transforme en registre à décalage (figure II. 24 b)).

• Si $C_1$ $C_2$=10, le registre est monté en diviseur polynômial (figure II.24c)), il peut alors servir soit de registre d'analyse de signature parallèle et les réponses à compacter sont injectées sur les entrées Z, soit en générateur de séquences pseudo-aléatoire parallèle et les entrées Z sont maintenues à 0.

• Si $C_1$ $C_2$=01, les bascules du registre sont remis à zéro.

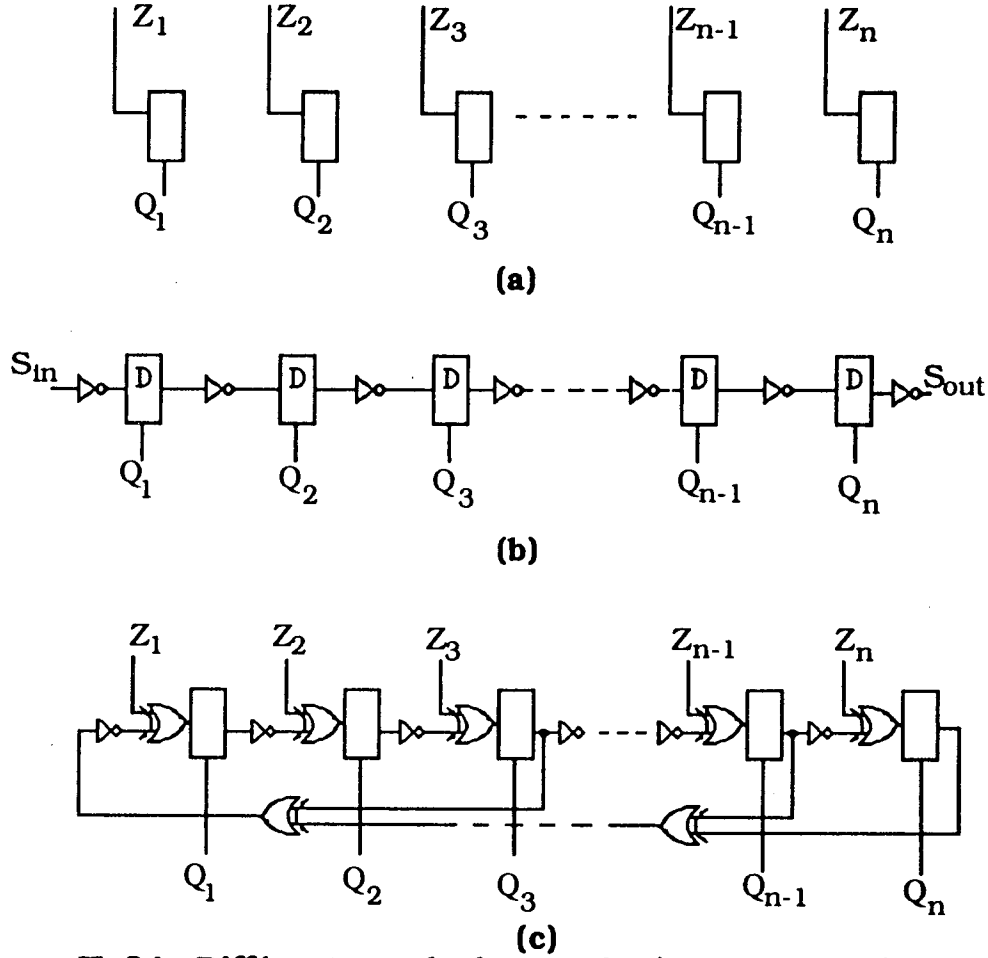



**Figure II. 24 : Différents mode du BILBO, a) $C_1C_2$=11 registre, b) $C_1C_2$=00 registre à décalage, c) $C_1C_2$=10 mode LFSR.**

**II. 6 Schéma d'unification du test intégré en ligne et hors-ligne : UBIST**

Si l'on voulait assurer d'une façon efficace tous les types de tests nécessaires pour les circuits intégrés, on serait amené à intégrer à la fois le test hors-ligne (technique BIST) et le test en-ligne (techniques d'autocontrôlabilité). Or les tests intégrés en-ligne et hors-ligne ont été étudiés in dépendamment l'un par rapport à l'autre. L'intégration de deux types de test nécessite le développement de nouvelles méthodes, afin d'exploiter au maximum les avantages que l'un pourrait apporter à l'autre.

Une structure systématique d'unification du test en ligne et hors-ligne est proposée dans [NIC 88], elle combine une structure à BILBO avec une

architecture globale d'un circuit auto-contrôlable.

Le schéma global est représenté en figure II. 25 .

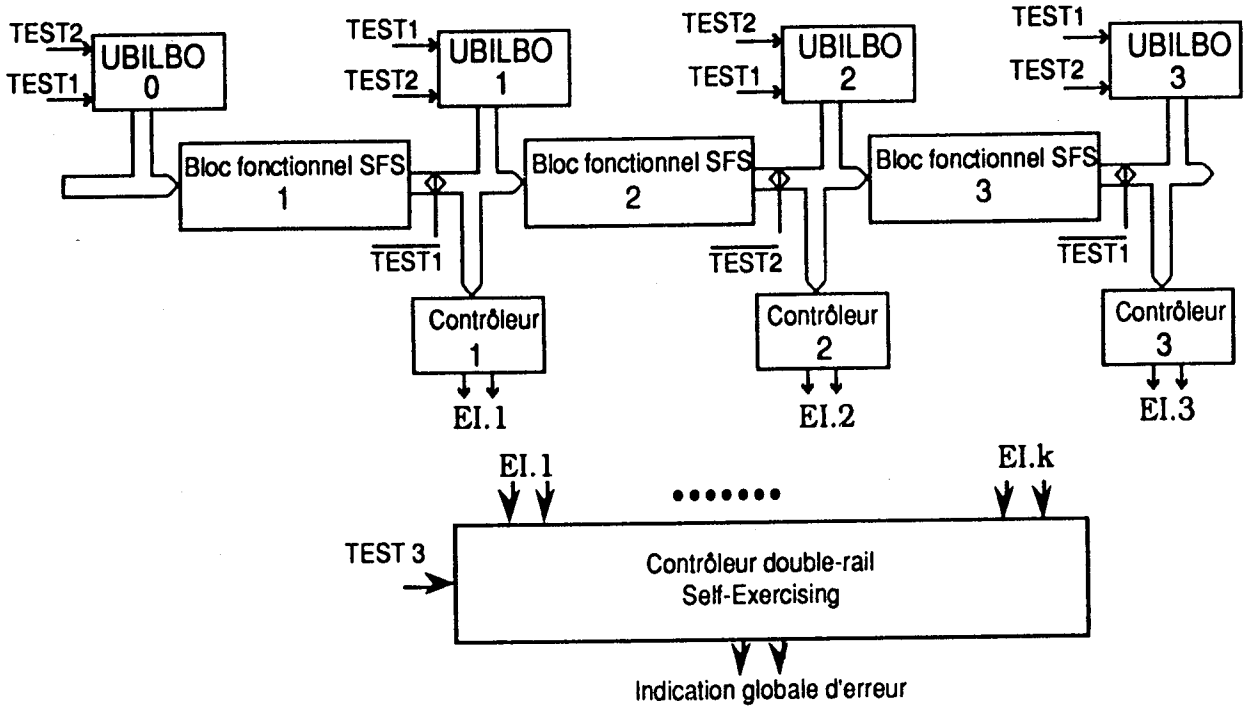



**Figure II. 25 : Schéma UBIST.**

Les UBILBOs sont des BILBOs modifiés de façon à être utilisés pour le test hors-ligne des blocs fonctionnels qui reçoivent $2^n$ vecteurs d'entrées (et sont vérifiés par le contrôleur aval) et le UBILBO suivant est utilisé en analyseur de signature compactant les réponses du bloc fonctionnel.

De façon simultanée, le contrôleur amont est également testé pour tout type de panne.

Ce schéma réalise des tests en parallèle en deux phases :

| **Mode** | **TEST 1** | **TEST 2** |
|---|---|---|
| Génération de vecteurs | 1 et 3 | 0 et 2 |
| Blocs fonctionnels sous test | 2 et 4 | 1 et 3 |
| contrôleurs sous test | 1 et 3 | 0 et 2 |
| Contrôleurs vérifiant | 2 et 4 | 1 et 3 |
| Analyse de signature | 2 et 4 | 1 et 3 |

Durant la phase de test TEST 1, les vecteurs de test générés par les UBILBOs paires (UBILBO 0, 2, 4, ...) sont appliqués sur les bloc fonctionnels impaires (bloc fontionnel 1, 3, 5, ...), et sur les contrôleurs paires (contrôleur 2, 4, ...). Les réponses des blocs fonctionnels impaires sont vérifiées par les contrôleurs impaires, et sont aussi compressées par les UBILBOs impaires sous forme de signature. A la fin de la phase de test

TEST 1, une sortie série des signatures dans les UBILBOs impaires est effectuée.

De la même manière, durant la phase de test TEST 2, les blocs fonctionnels paires et les contrôleurs impaires sont vérifiés.

Le bon fonctionnement du contrôleur global qui compacte les signaux d'indication d'erreur est assuré par une phase de test (TEST 3), le contrôleur global étant un contrôleur auto-activable.

Ce schéma UBIST est pour les circuits auto-contrôlables ce qu'est le schéma BIST pour les circuits conventionnels.

**Conclusion :**

Une discussion sur les modèles de pannes a été donnée, elle nous a fixé le modèle de pannes utilisé pour les PLAs.

Nous avons ensuite proposé des solutions d'implémentation des contrôleurs du code de Berger, et des contrôleurs double-rail. Ces schémas respectent des contraintes générales de voisinage et de performances de vitesse des blocs fonctionnels. Par contre, leur testabilité n'est pas systématiquement garantie. C'est pour cela qu'on leur a associé des générateurs spécifiques de vecteurs de test, qui les activent en des phases de test hors-ligne. Les contrôleurs ainsi construits sont appelés contrôleurs auto-activables. On leur assure par cette construction la couverture de toute panne pouvant les affecter.

D'autre part, le schéma BILBO permet d'activer un circuit en des phases de test hors-ligne. Il permet aussi la génération de séquences de test n'appartenant pas au code d'entrée du bloc fonctionnel, certaines de ces séquences sont capables de détecter des pannes multiples indétectables par le schéma d'auto-contrôlabilité.

L'unification du test en-ligne (schéma d'auto-contrôlabilité) et du test hors-ligne (schéma du type BILBO), est proposée dans [NIC 88]. Ce schéma, appelé UBIST, permet de tirer les avantages de chacun des tests.

Le schéma UBIST permet par ailleurs d'assurer, à des coûts raisonnables, la plupart des tests nécessaires aux circuits intégrés, (test de fin de fabrication, test de maintenance, test en ligne, ...).

# CHAPITRE III

**Stratégie d'autotest dans le compilateur de silicium SYCO**

**Introduction :**

Dans ce chapitre, une présentation des différents compilateurs de silicium avec test est donnée.

Nous présenterons, tout d'abord, les notions de base de la compilation de silicium, dans lesquelles nous introduirons le test comme élément à part entière pouvant agir à tous les niveaux du processus de compilation.

Nous présenterons alors un exemple concret qui est le compilateur de silicium SYCO développé dans l'équipe d'architecture des ordinateurs du laboratoire TIM3/IMAG, pour lequel on propose une stratégie UBIST de testabilité.

L'introduction de cette stratégie se fait au niveau architectural ou structurel sous forme de greffe d'outils de CAO de test transformant des descriptions architecturales ou structurelles en leur correspondantes testables.

Il n'est pas encore possible (voire nécessaire) de spécifier les mécanismes de test, tel que le schéma UBIST, dans un langage de description de haut niveau tel qu'une description comportementale. Les tentatives effectuées dans ce sens [FUN 85] n'ont pas atteint un degré suffisant de maturité pour spécifier complètement le test, et pour atteindre des résultats souples et performants.

**III. 1 Introduction aux compilateurs de silicium :**

Un compilateur de silicium est défini comme étant un système permettant de générer le dessin des masques d'un circuit intégré en partant d'une description de haut niveau de ce circuit [GAJ 87]. Cette description peut être un algorithme, une architecture, un dessin symbolique, etc...

On définit de façon plus large un compilateur de silicium comme étant un système capable de transformer la représentation d'un circuit ou une partie d'un circuit en une autre représentation de niveau d'abstraction inférieur.

Les niveaux d'abstraction correspondent aux étapes de la conception. La plupart des travaux effectués sur les niveaux d'abstraction [THO 83], [GAJ 87] distinguent les niveaux suivants:

• Niveau géométrique : C'est le dessin des masques du circuit. Cette description est suffisante pour la fabrication du circuit.

• Niveau électrique : On considère à ce niveau des objets de base ayant des propriétés électriques (transistors, résistances, capacité, ...).

• Niveau logique : Un circuit à ce niveau est décrit en un ensemble de portes logiques et d'éléments de mémorisation interconnectés. Ce niveau est resté jusqu'à nos jours un standard de description de circuits. Il correspond au niveau le plus bas qui soit commun aux méthodes de conception de circuits. Le formalisme mathématique (algèbre de Boole, ...) qu'il utilise est adopté par tous les concepteurs pour décrire des fonctions complexes de systèmes. Ce type de formalisme se prête bien aux méthodes formelles de synthèse, d'optimisation et de vérification [BRA 85].

• Niveau transfert de registre : Les langages à ce niveau manipulent des éléments de mémorisation et des opérateurs. Ce niveau est parfois désigné sous le nom de niveau micro-architecture. On utilise à ce niveau des descriptions comportementales et des descriptions structurelles.

• Niveau programme : Les objets manipulés à ce niveau sont aussi des éléments de mémorisation et des opérateurs, mais ceux-ci ne sont pas liés aux réalisations physiques. A ce niveau la description de circuits est du type comportemental.

La figure III.1 retrace le diagramme de GAJSKI (dit diagramme en "Y"). Ce diagramme représente les trois domaines de description: le domaine comportemental, le domaine structurel, et le domaine physique.

La spécification d'un circuit nécessite en général plusieurs représentations, chacune d'elles peut contenir des informations d'un ou de plusieurs domaines de description.

Un compilateur de silicium idéal peut être défini comme étant un système capable de transformer une description du niveau du cercle le plus

grand (niveau système de la figure III.1) en une description du niveau du cercle le plus petit (niveau circuit de la figure III.1).

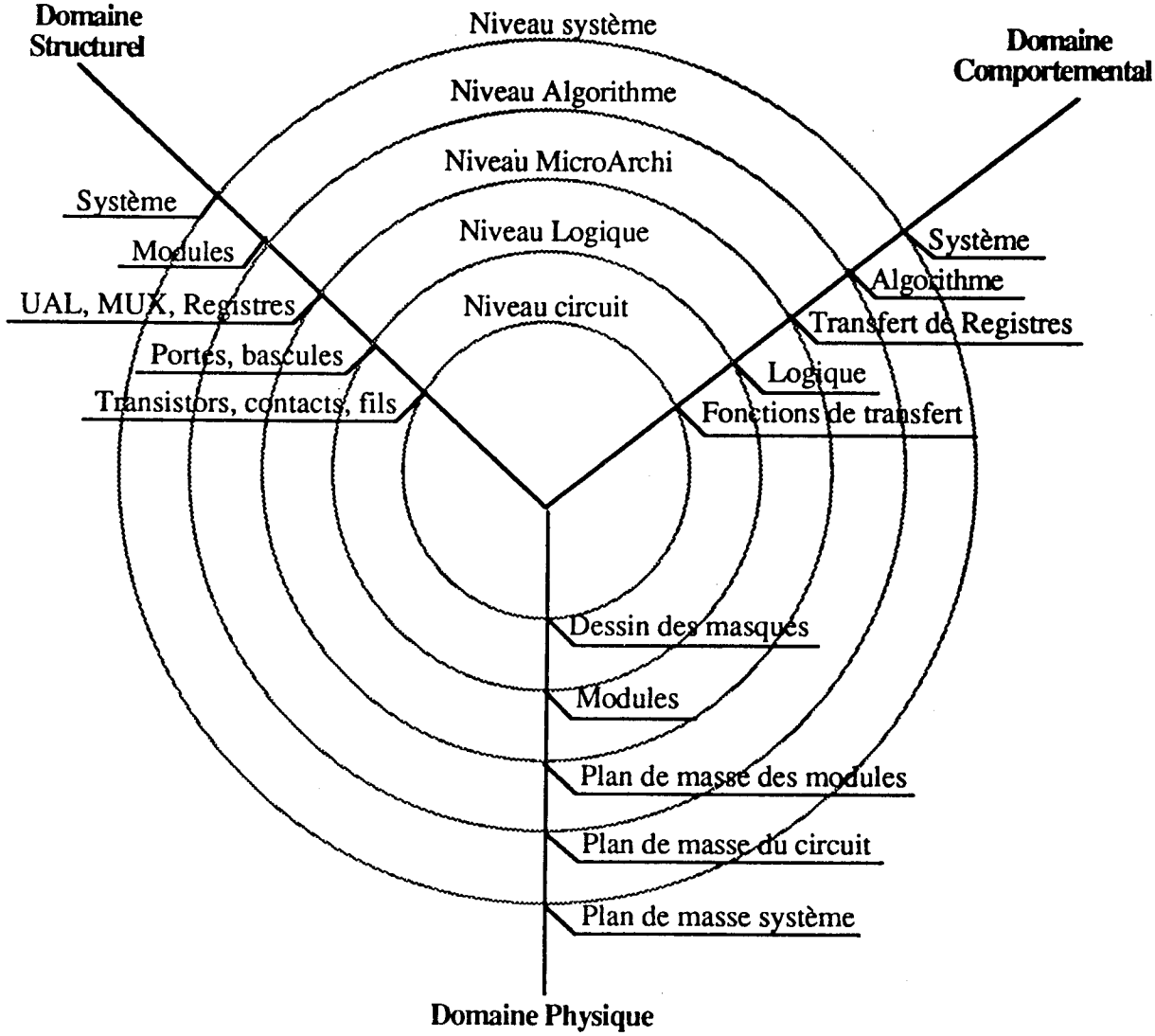



**Figure III. 1 : Diagramme en "Y" des domaines de description.**

On complète ce diagramme par une quatrième branche représentant la description du test du circuit. Le diagramme correspondant serait un diagramme en "X". La quatrième branche (dessinée en grisé en figure III.2), ne correspond pas à un domaine regroupant une hiérarchie de niveaux de représentation, mais à une hiérarchie de méthodes de testabilité agissant sur les différents niveaux.

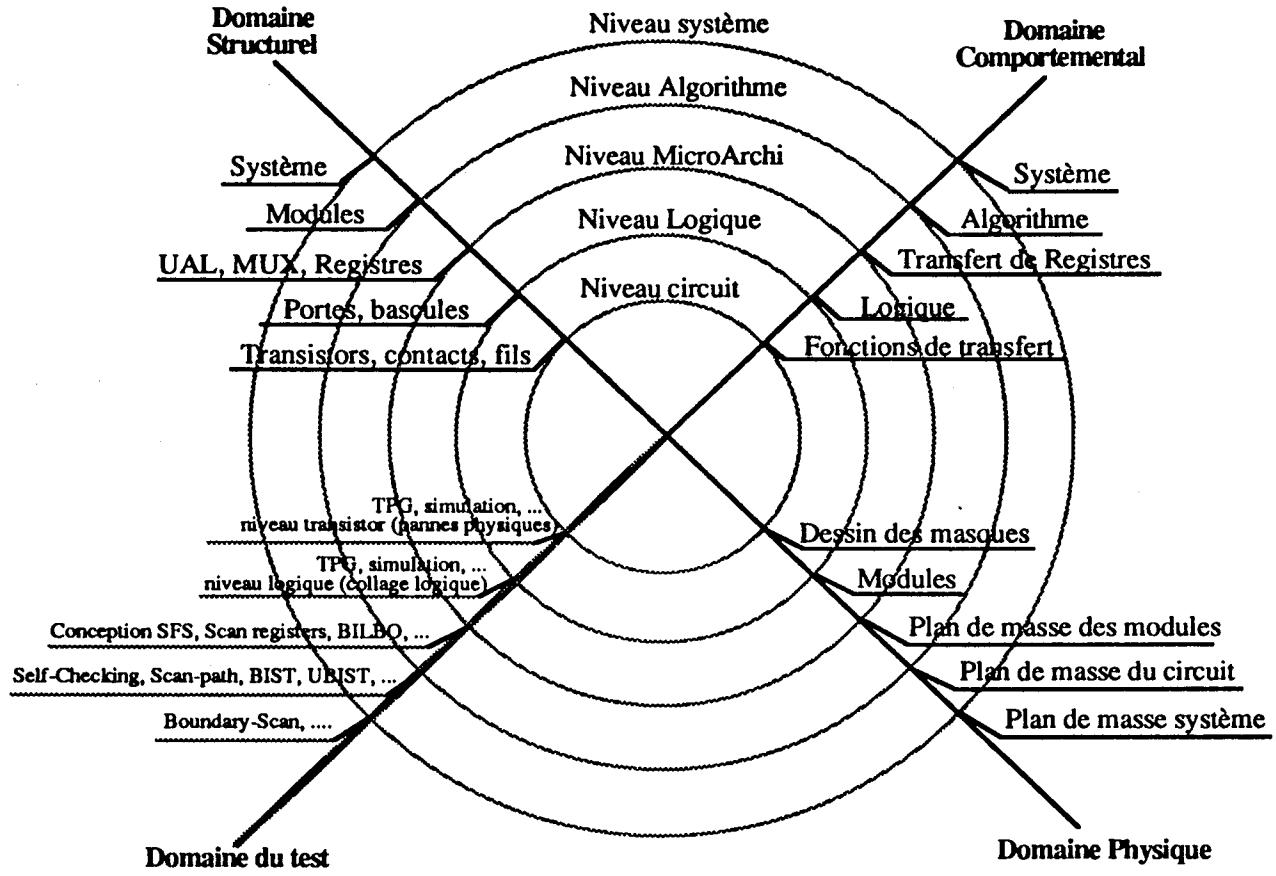



**Figure III. 2 : Diagramme en "X" des domaines de description.**

### III. 1. 1 Méthodologie de conception :

La conception d'un circuit intégré est régie par un processus mettant en oeuvre des outils de CAO travaillant sur plusieurs niveaux d'abstraction.

La figure III.4 montre un système de CAO travaillant sur plusieurs niveaux d'abstraction. Le système est organisé autour d'une pile d'étages. Chaque étage correspond à un niveau d'abstraction. Des outils de création, de validation, d'optimisation de la description, et de test existent à chaque niveau.

Les outils de compilation permettent le passage rapide et sans erreur entre les différents niveaux. Les outils de vérification permettent la vérification de la cohérence entre ces niveaux.

Les outils de test permettent de spécifier automatiquement des schémas de test pour les différentes représentations du circuit, (représentés en grisé en figure III. 3).

Le passage d'un niveau d'abstraction à un niveau inférieur peut se faire manuellement, assisté, ou automatique. Ces trois cheminements possibles définissent s'il s'agit d'un environnement pour la compilation de silicium, d'un assistant pour la compilation de silicium, ou d'un compilateur de silicium.

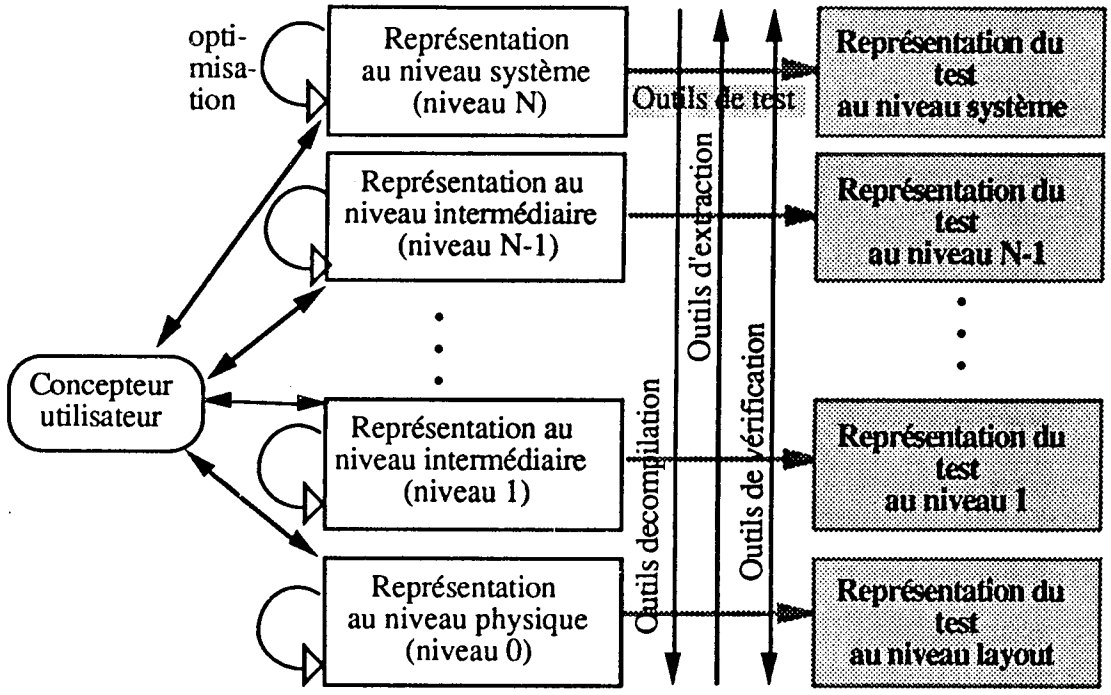



**Figure III. 3 : Environnement complet pour l'aide à la conception.**

D'autre part, trois niveaux d'abstraction sont utilisés comme spécification de circuits. Le niveau structurel permet de spécifier un circuit dans le cadre d'une famille technologique et indépendamment du processus de fabrication. Le niveau architectural permet de représenter le circuit indépendamment de la technologie. Le niveau comportemental permet de décrire la fonction réalisée par le circuit, indépendamment de son architecture. Un quatrième niveau, le niveau physique, permet de spécifier un circuit avec tous les détails nécessaires à sa fabrication.

Ces quatre niveaux de description définissent trois classes de compilateurs. Chaque niveau de représentation, autre que le niveau physique, définit une classe de compilateurs couvrant une étape du processus de conception.

La figure III. 4 donne un schéma de compilation de silicium complet utilisant les quatre niveaux, ce schéma comporte trois étapes :

- La compilation de comportement, qui produit l'architecture à partir de la représentation du comportement.
- La compilation d'architecture, qui produit la structure à partir de la représentation de l'architecture.
- La compilation de structure qui produit une représentation physique en partant de la structure.

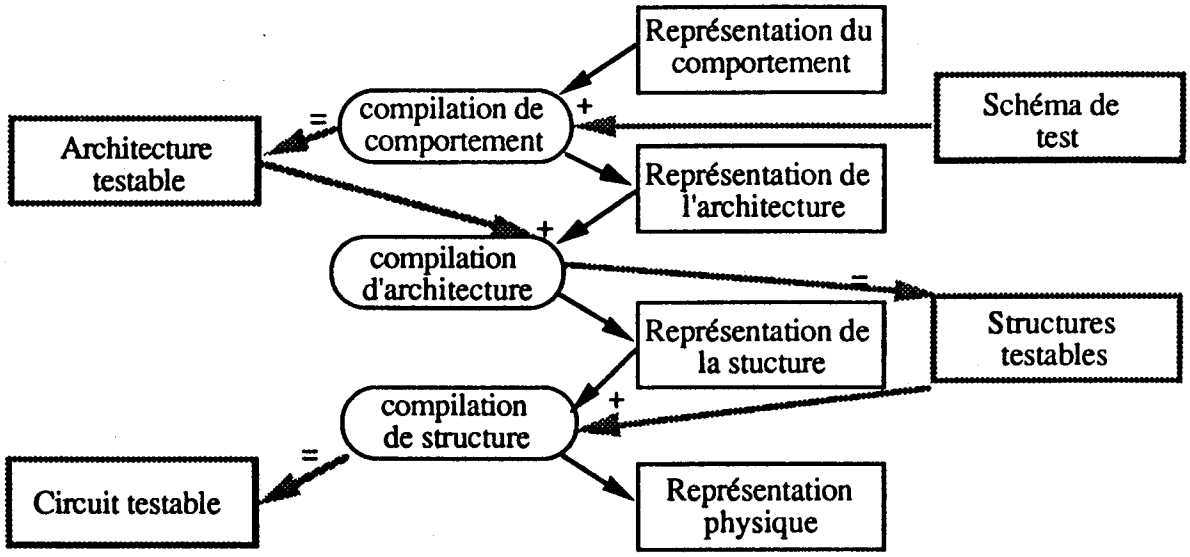



**Figure III. 4 : Etapes de compilation de silicium.**

En grisé a été rajouté les représentations du test à différents niveaux, et les processus nécessaires pour passer d'un niveau à un autre.

On trouvera une étude détaillée de ces types de compilateurs et leur classification dans [GAJ 88] et [JER 89]. On retiendra le fait que le compilateur de silicium SYCO [JER 88], [JER 89], développé au sein de l'équipe d'architecture des ordinateurs du laboratoire TIM3/IMAG, part d'une description du comportement et utilise une architecture cible permettant de générer des circuits ayant une partie contrôle hiérarchique et une partie opérative parallèle.

**III. 2 Les compilateurs de silicium avec test :**

La stratégie de test adoptée par certains compilateurs de silicium est basée essentiellement sur des schémas DFT, utilisant des techniques BIST ou scan-path.

Pour le compilateur SILC [FUN 85], [FUN 86] :

• Des techniques ADFT [AGR 84], transforment automatiquement une logique donnée en logique "scan série" dans un environnement "scan-cell". Le système renferme un générateur automatique de test utilisant un algorithme du type D-algorithme.

• Un système expert semi-automatique sélectionne une technique DFT de PLAs. Le choix se fait par rapport à la couverture de panne et au surcoût matériel souhaités.

H. S. Fung [FUN 85], propose des concepts de "testabilité par construction" dans SILC. Les propriétés de testabilité sont introduites tôt dans la phase de compilation. Les logiques de test sont synthétisées et

implémentées conjointement à la logique fonctionnelle normale. Leurs outils ADFT agissent après le parseur du langage d'entrée et avant l'extraction et la minimisation des structures.

Dans SILC, deux méthodes sont utilisées pour assurer le test:

• Une logique est rajoutée aux machines d'états finis, pour implémenter des techniques particulières de DFT.

• Des blocs de test sont désignés aux générateur de layout et des spécifications spéciales de test sont rajoutées aux blocs fonctionnels normaux.

Dans les machines d'états finis, des états sont rajoutés pour effectuer certaines opérations de contrôle du test.

Le chargement en série ou en parallèle dans les registres est décrit algorithmiquement au niveau de la structure de donnée, spécifiant le mode scan série ou parallèle. Il en est de même pour les registres BILBO.

Fung a soulevé quelques questions clés concernant l'introduction du test dans un compilateur de silicium:

• Insuffisance des outils automatiques industriels de test pour être inclus dans un environnement de compilation de silicium [FUN 86],

• Ces outils agissent essentiellement localement sur les structures, sans avoir une vue globale de testabilité d'un circuit.

• Dans un environnement de compilation de silicium, les décisions de testabilité doivent être établies tôt dans le processus de synthèse.

• Le comportement et les performances du circuit transformé doivent être revérifiées.

• Les techniques DFT doivent être adaptées à l'architecture du circuit.

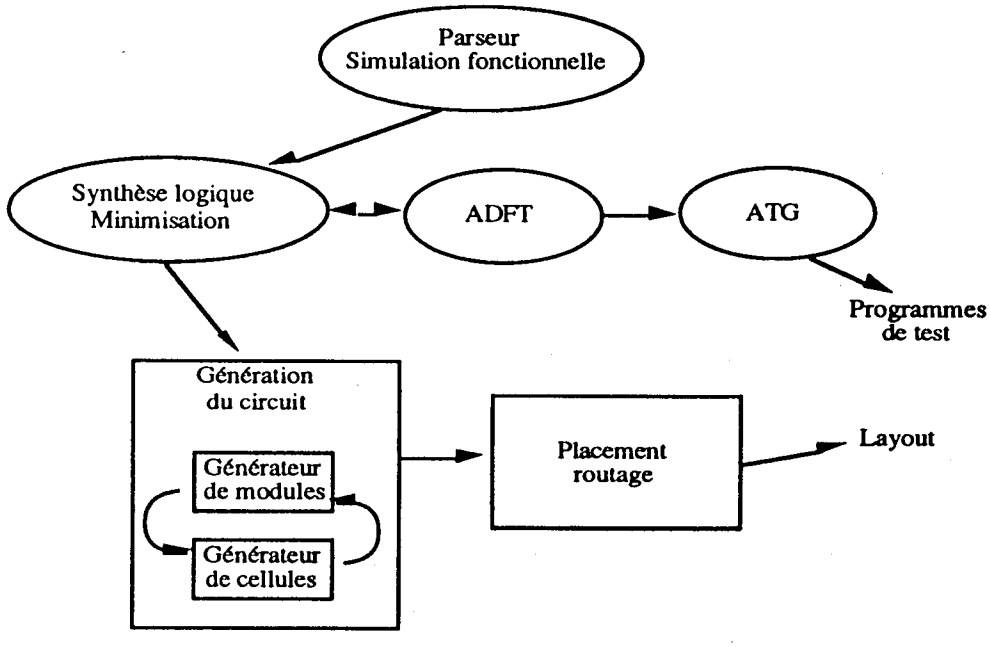



**Figure III. 5 : Stratégie du test dans le compilateur de silicium SILC.**

F. Beenker et al. proposent dans [BEE 89] une stratégie de test pour le compilateur de silicium PIRAMID [HUI 88].

PIRAMID est un compilateur de silicium destiné à générer des circuits du type multiprocesseur.

La méthodologie de test choisie pour ce compilateur, consiste aussi à développer des programmes de test pour faire du scan-path, (tous les registres sont configurés de façon à pouvoir être "scannables"), et les RAMs sont modifiées en un schéma BIST.

L'introduction des techniques de test se fait là aussi au niveau structurel, puisqu'il s'agit d'outils agissant très peu sur l'architecture du circuit (introduction de circuiterie pour le scan-path, sans modification des blocs fonctionnels).

D. G. SABO et al. propose dans [SAB 86], une stratégie similaire de test (scan-path et BILBO) au compilateur de silicium GENESIL.

**III. 3 Le test dans le compilateur de silicium SYCO :**

Un compilateur de silicium ne peut explorer tout l'espace des solutions en partant d'une description comportementale. Il faut lui limiter cet espace en adoptant des restrictions à tous les niveaux du processus de compilation, de la description comportementale à la description physique.

Dans le cas du compilateur SYCO, une restriction importante a été fixée, il s'agit de l'architecture cible imposée au modèle général de compilation. Il s'agit d'un découpage en niveaux d'interprétation, conduisant au scindement du circuit en tranches hiérarchisées. La tranche inférieure correspond au chemin de donnée (partie opérative), et toutes les autres correspondent à une hiérarchie de tranches de contrôle, formant la partie contrôle.

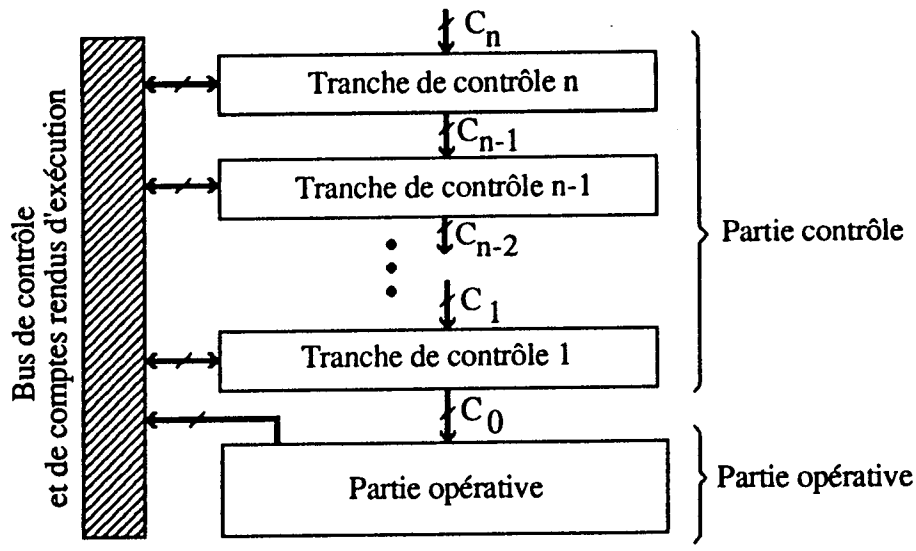



**Figure III. 6 : Architecture cible du compilateur de silicium SYCO.**

Le modèle utilisé pour la partie opérative est inspiré de celui de la partie opérative du MC 68000. Il s'agit d'une partie opérative parallèle organisée en tranches de bits (voir le chapitre VI).

Le modèle utilisé pour la partie contrôle est une hiérarchie de tranches de contrôle, chacune d'elles est organisée sous forme de PLA (voir chapitre V).

Les étages de contrôle communiquent à travers un bus de contrôle et utilisent un bus de comptes rendus d'exécution alimenté par la partie opérative (voir figure III.6).

Le dessin des masques des circuits est aussi composé d'une pile de tranches, constitué par une partie contrôle générée par CPC [MHA 87] et une partie opérative générée par APOLLON [JAM 85].

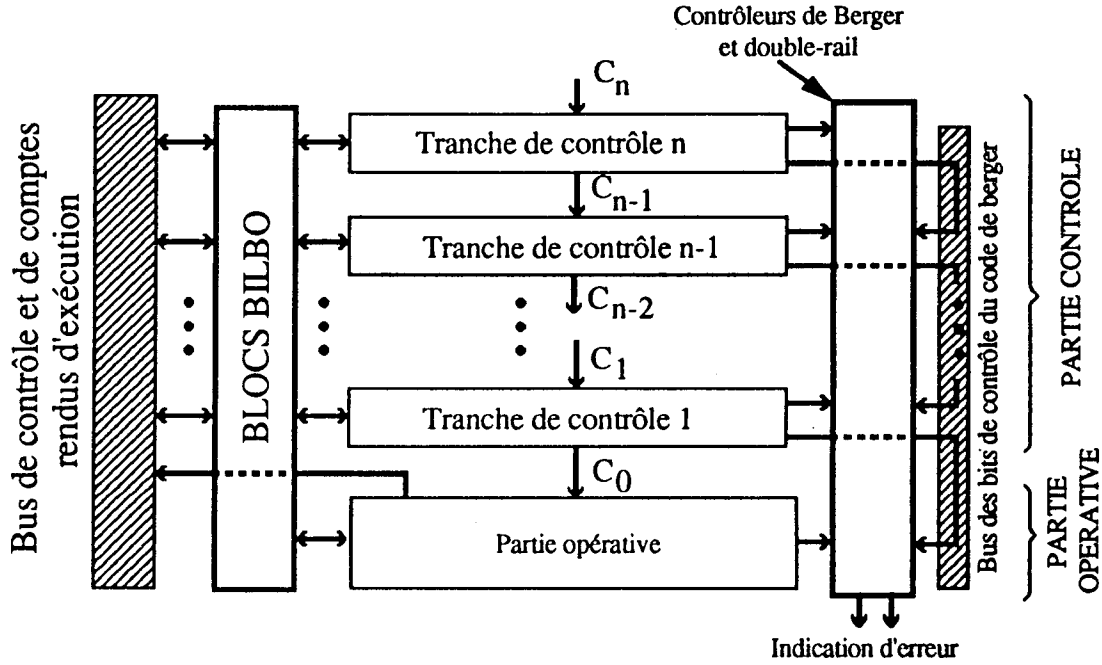



**Figure III. 7 : Architecture UBIST cible pour SYCO.**

Nous proposons aux chapitres V, VI et VII des schémas UBIST d'autotestabilité liées à l'architecture cible utilisée par SYCO. La version UBIST du compilateur de silicium SYCO, se situe au niveau des compilateurs d'architecture contrairement aux autres [FUN 85], [HUI 88], [SAB 86] qui se situeraient au niveau des compilateurs de structures.

Les implémentations du schéma UBIST soulèvent des problèmes architecturaux dont leur solutions sont laborieuses à automatiser dans le cas général. Par exemple, le schéma UBIST de partie contrôle du type de celui du MC 68000, a été proposé dans [NIC 90], l'automatisation du processus de sa génération est difficile à mettre en oeuvre par manque de régularité de ses structures.

**Conclusion :**

On a présenté dans ce chapitre, les notions de base de la compilation de silicium conduisant à une réflexion sur l'introduction d'une stratégie de testabilité.

L'introduction de méthodologies de test dans un compilateur de silicium dépend de plusieurs facteurs.

Quand l'architecture adoptée par le compilateur de silicium admet une ou plusieurs solutions de test, et si cette (ou ces) solution est automatisable, on pourra introduire les outils de CAO de test dans la chaîne d'outils de CAO utilisée par le compilateur. C'est déjà une étape importante franchie.

Si les outils de CAO de test agissent sur des descriptions comportementales et si le produit de compilation correspond à une solution optimale et performante (en comparaison à une solution obtenue par un concepteur expérimenté en test), on dira que l'on aura atteint le but de compilateur de circuits testables.

Malheureusement, on en est pas à ce stade actuellement. En effet, les outils de CAO de test développés actuellement, disponibles sur le marché, et présentés dans la littérature, agissent sur des spécifications architecturales ou structurelles (voire même physiques, tel que le ADFT pour la génération de vecteurs de test pour des modèles de pannes au niveau interrupteur).

Néanmoins, de la même façon qu'on parle de compilateur de silicium comportemental, architectural ou structurel, l'étude présentée dans les chapitres IV, V, VI et VII, pour la version du compilateur SYCO générant des circuits UBIST, propose un compilateur de silicium architectural.

# CHAPITRE IV

**Outils de génération de PLAs autotestables**

**Introduction :**

La régularité des structures du type PLA, ROM, favorise l'élaboration de techniques systématiques de test intégré.

En ce qui concerne le test intégré en-ligne de PLAs, nous présenterons le schéma de base de PLAs auto-contrôlables [MAK 82], et nous proposerons un outil (PROTECT) de génération automatique du codage des sorties d'un PLA donné, et les performances obtenues quand à l'augmentation de surface du PLA due au codage.

Pour un test hors-ligne de PLAs, la contrôlabilité est rendue difficile car chaque entrée contrôle deux lignes, le changement d'une entrée implique le changement de la ligne et de sa complémentaire. D'autre part, chaque vecteur d'entrée sélectionne en général plusieurs monômes à la fois. Pour ces raisons, la génération de vecteurs de test pour un PLA sans modifications est une tâche laborieuse.

En parcourant la littérature, on constate une diversité de schémas de test hors-ligne de PLAs basés sur le contrôle de chaque ligne d'entrée et de chaque monôme individuellement [FER 88]. On s'est intéressé aux PLAs autotestables utilisant une génération exhaustive ou pseudo-aléatoire et une compression des réponses sous forme de signature [DAE 81], [HAS 83]. Ce schéma est facile à mettre en oeuvre, il est suffisant lorsqu'il est utilisé comme complément au schéma d'auto-contrôlabilité.

Nous présenterons donc un outil spécialisé simulateur de signature permettant le calcul de la signature d'un PLA pour un test exhaustif ou pseudo-aléatoire, avec des temps de calcul faibles.

Ces outils représentent les outils clés de génération automatique de PLAs autotestables.

**IV. 1. Génération de PLAs FSPD :**

Dans les structures auto-contrôlables, un bloc FSPD doit générer un codage de ses sorties. En ce qui concerne les PLAs, le codage des sorties s'obtient en rajoutant des lignes de sorties correspondant à la fonction de codage. Chaque mot de sortie généré par le PLA sera concaténé à son codage pour former le mot de sortie codé.

Les différents codages intéressants utilisés dans la littérature pour coder les sorties de PLAs sont :

* Le code de Berger [MAK 82] [NIC 89].
* Le code m-parmi-n [WAN 79] [NIC 89].
* Le code de parité [NIC 86] [FUC 87] [NIC 89].
* Le code de Berger modifié [MAK 82].
* Le code double-rail [KHA 82] [NIC 86] [NIC89].

On rappelle que le code m-parmi-n est un code non séparable et le code de parité est un code qui n'est pas non ordonné.

Un PLA deviendra FSPD, si ses entrées sont contrôlées et si ses sorties sont codées en un code non ordonné et contrôlées (figure IV. 1).

Un autre schéma représenté en figure IV. 2 consiste à contrôler les entrées, coder et contrôler les monômes et les sorties en code de parité.

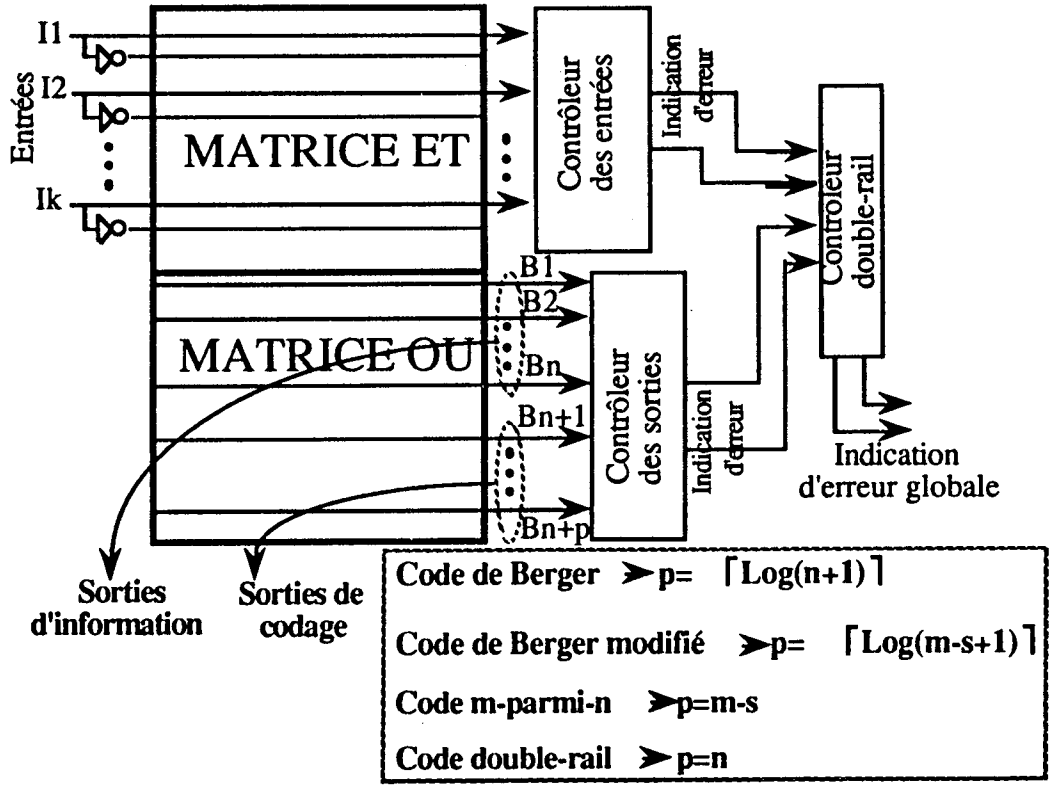



**Figure IV.1 : PLA FSPD avec code non ordonné.**

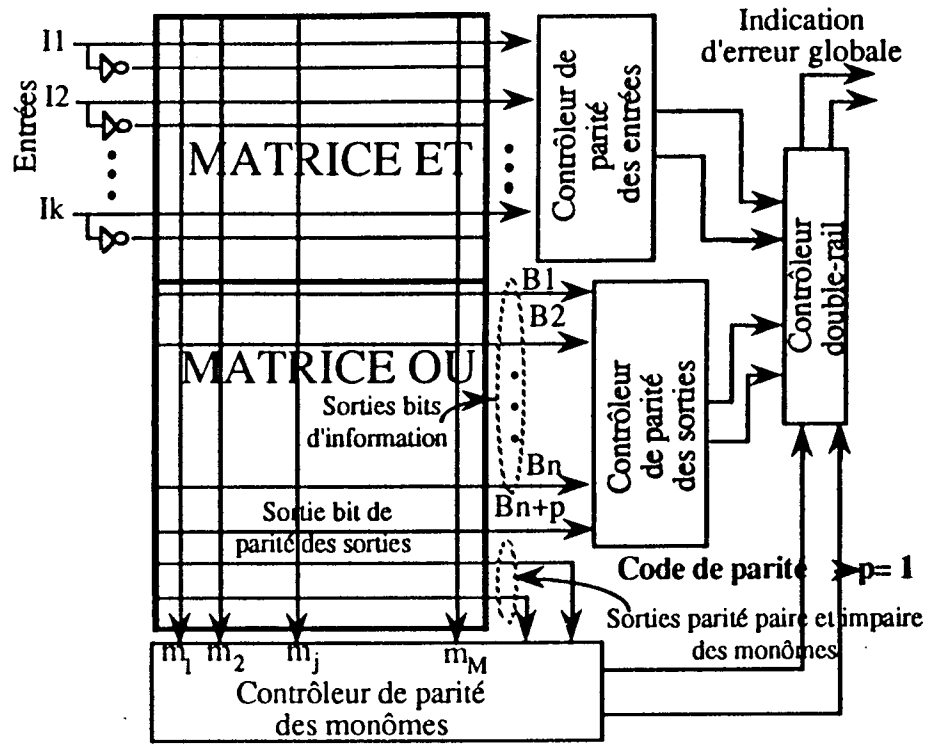



**Figure IV. 2 : PLA FSPD avec codage de parité**

On rappelle tout d'abord les définitions de concurrence et de non concurrence de PLA.

**Définition :** *Un PLA est dit non concurrent quand chaque vecteur du code d'entrée ne sélectionne qu'un monôme à la fois ( un monôme est considéré sélectionné s'il prend la valeur "1").*

**Définition :** *Un PLA est dit concurrent lorsqu'il existe au moins un vecteur du code d'entrée sélectionnant plus d'un monôme à la fois.*

Le problème de génération de bits de codage des sorties par le PLA lui-même, peut être abordé de deux manières :

1) La première est basée sur une description booléenne du PLA et du codage de ses sorties. Des équations booléennes du codage des sorties sont établies. Pour un codage double-rail des sorties, on doit complémenter les équations booléennes de chaque sortie normale du PLA. Pour un codage de Berger, on doit partir des équations booléennes décrivant les bits de codage par rapport aux bits d'information des sorties et retrouver les équations booléennes décrivant ces bits de codage par rapport aux entrées.

Cette méthode utilise un formalisme algorithmique des équations décrivant les sorties de codage, elle est onéreuse en temps de calcul.

2) La seconde méthode consiste à partir de la table de vérité du PLA. Il suffirait par exemple de complémenter les valeurs des sorties de la table pour un codage double-rail, ou de compter le nombre de zéros sur chaque mot de sortie et de joindre le résultat en représentation binaire pour un codage de Berger.

De toute façon, cette méthode est impraticable pour des PLAs de grande taille, et n'a aucun intérêt puisqu'on doit établir les tables de vérité exactes des PLAs. Par contre, on peut l'appliquer facilement à des spécifications non concurrentes de PLA, ou à des ROMs.

Si le PLA est déjà non concurrent, il n'y a pas de problème : chaque monôme représente un ensemble des vecteurs d'entrées pour lesquels on assigne un vecteur de sortie, il suffit donc de procéder à un codage comme décrit plus haut : Chaque mot de codage sera assigné à un vecteur de sortie, qui lui-même est assigné à un monôme, qui lui-même est assigné au même ensemble de vecteurs d'entrées. Et réciproquement, lors du fonctionnement normal, chaque vecteur d'entrée activera un monôme pour lequel on aura un vecteur de sortie avec le mot de codage correspondant.

Certaines descriptions de PLA sont obtenues au départ sous une forme non concurrente, par exemple si l'on décrit une machine à états (automate), on va procéder à un encodage non concurrent des différents états. De même qu'il existe dans la littérature des formes non concurrentes de PLAs (e.g. [WAN 79]).

Dans le cas où l'on aurait un PLA concurrent, son codage peut être effectué après l'avoir transformé en une forme non concurrente. Notre expérience sur un ensemble d'une trentaine de PLAs montre qu'il est toujours possible de générer la non concurrence. On propose d'adopter cette solution, c'est à dire développer un algorithme de génération d'une spécification non concurrente d'un PLA donné, coder les sorties du PLA non concurrent, et le minimiser pour le PLA codé optimisé.

Cependant on peut imaginer des situations de très forte concurrence où par exemple un monôme est activé en même temps que d'autres et pour lequel l'ensemble des vecteurs d'entrée concerné représente un grand ensemble du code d'entrée. Dans la figure IV.3 on remarque que le monôme N° 1 est toujours activé si l'un des autres monôme (2, 3, ....., 10) l'est. Ce cas correspond à un cas réel, il a été tiré d'une spécification du PLA de l'un des étages de la parties contrôle du 6502 généré par SYCO.

Dans ce cas le temps d'exécution de la non concurrence est exorbitant et il ne serait pas possible de coder ce PLA suivant cette démarche, ni suivant une démarche classique utilisant les équations booléennes, puisque

le PLA a un nombre élevé d'entrées et de sorties. Deux solutions peuvent être envisagées.

Soit on rajoute une entrée qui prendra la valeur "1" quand ce monôme est activé, et la valeur "0" pour les autres (2, 3, ....., 10).

Soit on rajoute une circuiterie qui déconnectera les autres monômes quand celui-ci est activé. On verra par la suite une solution efficace et spécifique à ce problème, et pour d'assez gros PLAs ne pouvant pas voir augmenter le nombre d'entrées, surtout lorsque celles-ci sont générées par des blocs fonctionnels voisins et non de l'extérieur (voir paragraphe IV.5).

| Entrées | Sorties | N° du monôme |
|---|---|---|
| 01101010101 00000000000000000000000000000000 01010101 | 000101010101 1 0110001 0000 | 1 |
| 01101010101 00000000100101010000000000000000 01010101 | 000000000000 1 0111010 0000 | 2 |
| 01101010101 00000000001010100000000001010101 01010101 | 000000000000 1 0111010 0000 | 3 |
| 01101010101 00000000001010100000000001010101 01010101 | 000000000000 1 0111010 0000 | 4 |
| 01101010101 00000000000110100000000000001000 01010101 | 000000000000 1 0111010 0000 | 5 |
| 01101010101 00000000000110010000000000000100 01010101 | 000000000000 1 0111010 0000 | 6 |
| 01101010101 00000000000101100000000000000010 01010101 | 000000000000 1 0111010 0000 | 7 |
| 01101010101 00000000000101010000000000000001 01010101 | 000000000000 1 0111010 0000 | 8 |
| 01101010101 00000000001001101000000000000000 01010101 | 000000000000 1 0111010 0000 | 9 |
| 01101010101 00000000001001010100000000000000 01010101 | 000000000000 1 0111010 0000 | 10 |

**Figure IV.3 : Très forte concurrence dans un PLA.**

**IV. 2. Procédure de génération la non concurrence :**

Pour un PLA qui ne renferme pas encore de propriétés d'auto-contrôlabilité (i. e. les entrées et sorties ne sont pas codées), on utilisera le terme code d'entrée comme l'ensemble des vecteurs d'entrées qui peuvent être appliqués au PLA lors de son fonctionnement normal. Evidemment, aucun autre vecteur en dehors de cet ensemble (noté par vecteur hors code d'entrée) ne peut être appliqué lorsque le PLA est un bloc imbriqué du circuit.

Considérons $m_i=m_{i1}\ m_{i2}\ .....\ m_{in}$ comme représentation binaire du monôme $m_i$ de la matrice ET du PLA.

où $m_{ik}$=0, 1 ou X, k$\in${1, 2, ....., n}

$m_{ik}$ prend la valeur 1 (respectivement 0) quand un transistor MOS peut mettre à zéro la ligne monôme mi par la $k^{ième}$ ligne d'entrée (respectivement par la $k^{ième}$ ligne complémentaire d'entrée).

mik est noté "X" (ou "-") quand le monôme $m_i$ ne peut être sélectionné ni par la $k^{ième}$ ligne d'entrée ni par sa ligne complémentaire. Aucun transistor MOS n'est implémenté à ces 2 croisements.

Il nous faut trouver une propriété nous permettant de distinguer les

monômes concurrents de PLA, on introduit la notion de distance de monômes, qui nous permettra de savoir si 2 monômes sont concurrents ou pas.

**Définition :** *La distance D entre deux monômes $m_i$ et $m_j$ est définie par*

$$D(m_i, m_j) = \sum_{k=1}^{n} (m_{ik} \oplus m_{jk})$$

*où $m_{ik} \oplus m_{jk}$ est définit par l'opération commutative suivante :*
*$0 \oplus 0=0$, $0 \oplus 1=1$, $0 \oplus X=0$, $1 \oplus 1=0$, $1 \oplus X=0$, $X \oplus X=0$.*

D est une pseudo-distance de Hamming. On a deux cas possibles pour D(mi,mj) :

Soit $D(m_i,m_j)>0$, dans ce cas $m_i$ et $m_j$ sont non concurrent,

Soit $D(m_i,m_j)=0$, dans ce cas $m_i$ et $m_j$ sont concurrent.

Dans ce dernier cas $D(m_i,m_j)=0$, on peut avoir deux cas de figure, soit les monômes $m_i$ et $m_j$ sont redondants donc égaux, soit distincts mais concurrents.

Le schéma de la Figure IV.4 montre la valeur de la distance et sa relation avec la concurrence.

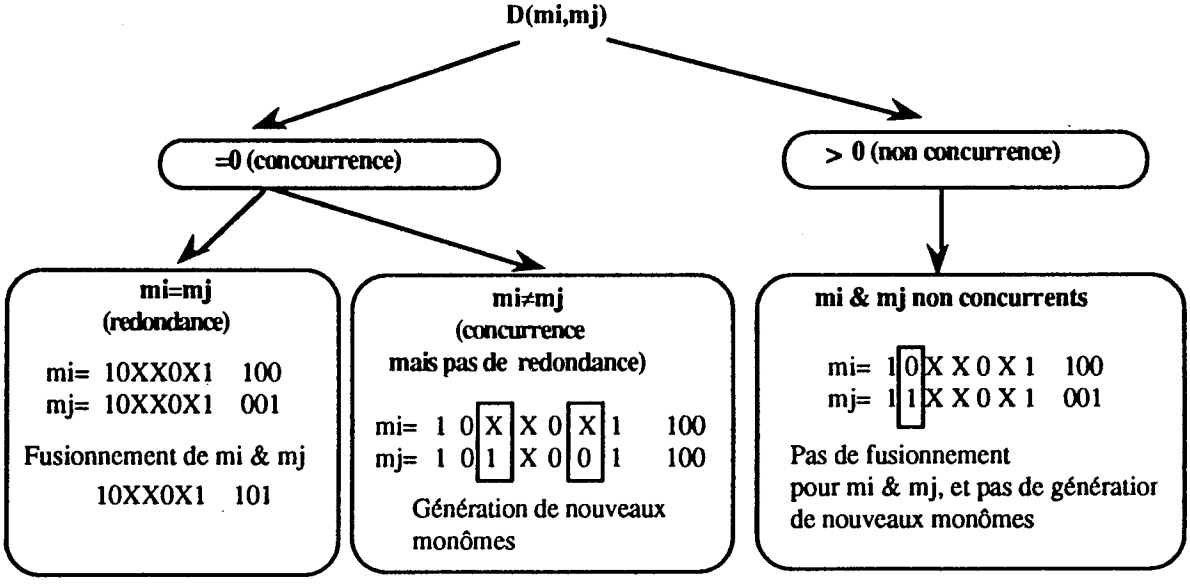



**Figure IV.4 : Valeur de la distance et concurrence.**

Dans le cas de redondance de deux monômes $m_i$ et $m_j$, les deux monômes sont égaux mais leur mots de sortie correspondants sont différents. Cette configuration est inutile, on supprime l'un des monômes (par exemple $m_j$) et on unifie les mots de sorties de façon à ce que tous les termes égaux à 1 dans la programmation de la matrice OU sont maintenus au monôme restant ($m_i$).

Dans le cas de la concurrence, il existe un ou plusieurs termes ignorés $m_{ik}$ pour lesquels les termes $m_{jk}$ correspondants prennent la valeur "0" ou "1" et tous les autres termes de $m_i$ et $m_j$ sont égaux.

Pour éliminer la concurrence, ce type de termes ignorés est éliminé en éclatant le monôme en question ($m_i$) en plusieurs autres tous non concurrents entre eux et non concurrents à l'autre monôme ($m_j$).

**IV. 2. 1. Représentation "ensembliste" de la concurrence dans un PLA :**

Soient $m_1$, $m_2$, ....,$m_i$, ....,$m_M$ les monômes d'un PLA, chacun d'eux peut être activé (seul ou avec d'autres) par un vecteur de l'ensemble A.

$A=e_1 \cup e_2 \cup e_3 \cup ... \cup e_N$ étant le code d'entrée du PLA, c'est un sous-ensemble de l'espace des vecteurs d'entrée X={ $2^n$ vecteurs d'entrée}.

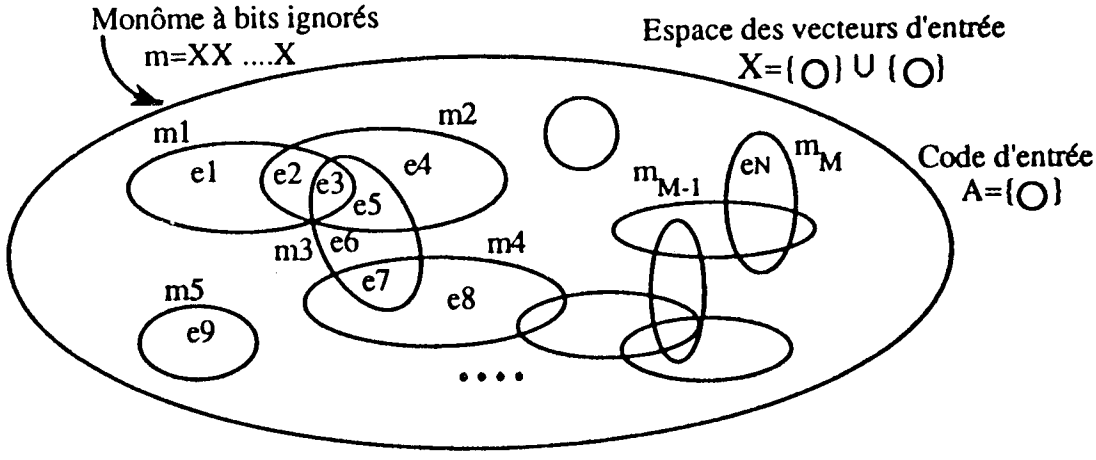



**Figure IV.5 : Représentation "ensembliste" d'un PLA concurrent.**

Dans l'exemple de la Figure IV.5, le monôme $m_1$ peut être activé par un vecteur d'entrée appartenant aux ensembles $e_1$, $e_2$ ou $e_3$.

$e_{m1} = e_1 \cup e_2 \cup e_3$ est l'ensemble des vecteurs du code d'entrée pouvant activer $m_1$.

Un vecteur appartenant à $e_1$ activera $m_1$ seul, alors qu'un vecteur appartenant à $e_2$ (resp. à $e_3$) activera systématiquement $m_1$ et $m_2$ (resp. $m_1$ $m_2$ et $m_3$).

Certains monômes sont non concurrents, c'est le cas de $m_5$, il est activé toujours seul par un des vecteurs de $e_9$.

L'idée d'éclater les monômes concurrents peut se voir de la manière suivante : Il faut arriver à représenter chacun des ensembles $e_1$, $e_2$, $e_3$, ....,$e_N$ par un ou plusieurs monômes tels que leurs intersections soient vides (figure IV.6).

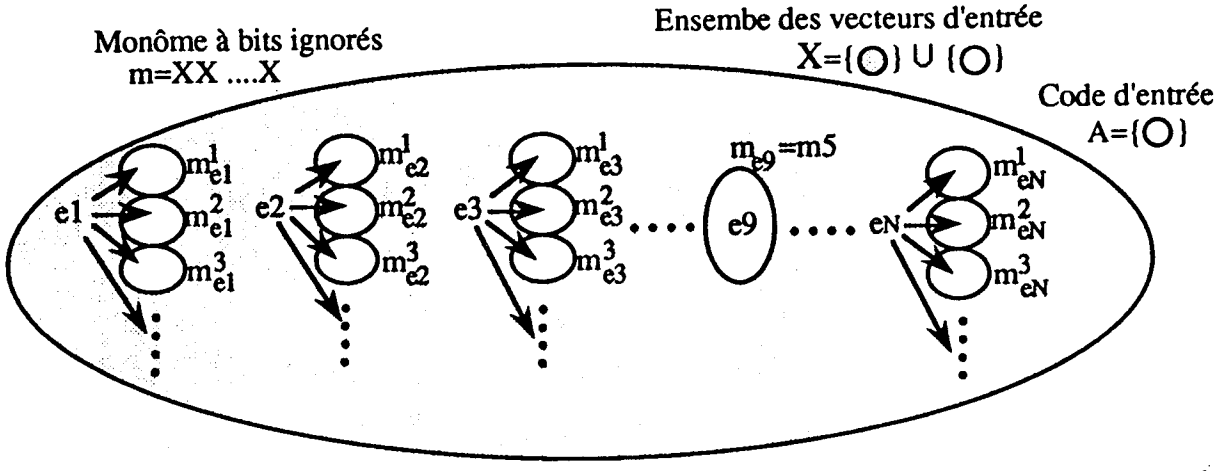



**Figure IV.6 : Représentation non concurrente du PLA de la Figure IV.5.**

### IV. 2. 2. Degré de concurrence :

On introduit la notion de degré de concurrence des monômes qui va nous permettre d'établir certaines propriétés.

**Définition :** *Le degré de concurrence "d" de deux monômes concurrents $m_i$ et $m_j$ est le nombre de bits ignorés "X" que $m_i$ et $m_j$ n'ont pas en commun.*

i. e. $d(m_i, m_j)= \Sigma_k(m_{ik} \otimes m_{jk})$ tel que $\otimes$ est l'opération commutative suivante : $0\otimes0=0$ , $1\otimes1=0$ , $0\otimes X=1$ , $1\otimes X=1$ , $X\otimes X=0$.

d(mi, mj)=1 ⇒ mi⊂mj ou mj⊂mi ⇔ (mi∩mj)=mi ou mj ⇔ (mi∪mj)=mj ou mi.
Exemple: mi=10X10 et mj=10XX0. On a d(mi,mj)=1 et mi⊂mj.

Dans la figure IV.5 un degré de concurrence égal à 1 entre $m_1$ et $m_2$ implique forcément que $m_1 \subset m_2$ ou $m_2 \subset m_1$, (l'inclusion est stricte). On a forcément $e_1=\{\emptyset\}$ ou $e_4 \cup e_5=\{\emptyset\}$. (La réciproque n'est pas toujours vraie).

Dès que le degré de concurrence augmente (d≥2), la relation d'inclusion n'est plus systématique.

| | | Exemples |
|---|---|---|
| d(mi, mj) ≥ 2 ⇒ | mi⊂mj **ou** mj⊂mi ⇔ (mi∩mj)=mi **ou** mj | mi= 10X10<br>mj= 1XXX0 |
| ou alors | mi⊄mj **et** mj⊄mi ⇔ (mi∩mj)⊂mi **et** mj | mi= 10XX0<br>mj= 1XX10 |

**Lemme IV.1** : *Si $d(m_i, m_j)=0$, alors mi et mj sont redondants.*

**Preuve** : $d(m_i, m_j)= \Sigma k\ (m_{ik} \otimes m_{jk})=0$ veut dire que pour chaque bit d'entrée $k\in\{1, 2, ..., n\}$, $(mik, mjk)\notin \{(0,X), (1,X), (X,0), (X,1)\}$. Puisque dans l'un de ces cas là $m_{ik} \otimes m_{jk}= (X \otimes(0 \text{ ou } 1))=1$.

Il ne reste plus que les combinaisons suivantes :

$(m_{ik}, m_{jk}) \in \{ (0,0), (1,1), (X,X) \}$, $\forall k\in\{1, 2, ..., n\}$

Ce cas correspond à $m_{ik}=m_{jk}$ $\forall k\in\{1, 2, ..., n\}$, cas de la redondance. **CQFD**

**Théorème IV.1** : *Si le degré de concurrence d entre deux monômes $m_i$ et $m_j$ est non nul (i.e. $d(m_i,m_j)=d>0$) alors d+1 monômes non concurrents sont nécessaires pour représenter $m_i$ et $m_j$.*

**Preuve** : La preuve se fera par récurrence : On montrera le théorème pour $d(m_i, m_j) =1$, et si pour $d(m_i, m_j)=d$ le théorème est vrai alors il est vrai pour $d(m_i, m_j) =d+1$.

• $d(m_i, m_j)=1 \Leftrightarrow \Sigma_k\ (m_{ik} \otimes m_{jk}) = 1 \Leftrightarrow \exists!\ k\in\{1, 2, ..., n\}$ tel que $(mik\otimes mjk) = 1 \Leftrightarrow (m_{ik}, mjk) \in \{ (0,X), (1,X), (X,0), (X,1)\}$

$\Leftrightarrow$ $(m_{ik},m_{jk}) = (0 \text{ ou } 1, X) \Rightarrow m_i \subset m_j$

ou $(m_{ik},m_{jk}) = (X, 0 \text{ ou } 1) \Rightarrow m_j \subset m_i$

Dans le cas où $m_i \subset m_j \Rightarrow m_j$ est éclaté en 2 monômes non concurrents $m^0{}_j$ et $m^1{}_j$, où $m^0{}_{jk}=0$ et $m^1{}_{jk}=1$.

On élimine $m_i$ en joignant sa programmation OU avec celle de son redondant $m^0{}_j$ (respectivement $m^1{}_j$). Ce dernier sera non concurrent au monôme restant $m^1{}_j$ (respectivement $m^0{}_j$).

Deux monômes non concurrents sont donc nécessaires pour représenter $m_i$ et $m_j$.

Le théorème est prouvé pour d=1 et $m_i \subset m_j$, la même preuve peut être faite pour l'autre cas ($m_j \subset m_i$).

• $d(m_i, m_j)=d+1 \Leftrightarrow \Sigma_k\ (m_{ik} \otimes m_{jk}) = d+1 \Leftrightarrow \exists\ (k_1, k_2, ..., k_{d+1}) \in \{1, 2, ..., n\}^{d+1}$ tel que $(m_{ik1} \otimes m_{jk1}=1)$, $(m_{ik2} \otimes m_{jk2}=1)$, ..., $(m_{ikd+1} \otimes m_{jkd+1}=1)$.

Considérons $m_{ik1}$ et $m_{jk1}$. On peut avoir soit $m_{ik1} \subset m_{jk1}$ ou $m_{jk1} \subset m_{ik1}$.

On lève la concurrence une fois :

Si $m_{ik1} \subset m_{jk1}$ alors mj est éclaté en 2 monômes $m_j{}^0$ et $m_j{}^1$ pour lesquels $m^0{}_{jk1}=0$ et $m^1{}_{jk1}=1$.

$m_i$, avec $m_{ik1}=0$ (resp. 1) va donc être non concurrent à $m_j{}^1$ (resp $m_j{}^0$) et concurrent à $m_j{}^0$ (resp $m_j{}^1$ ).

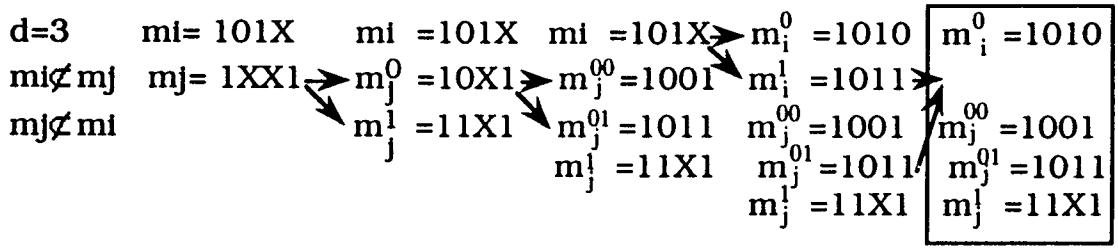
Prenons le cas où $m_i$ est concurrent à $m^0{}_j$ (le même raisonnement est

valable pour $m^1_j$). Le degré de concurrence entre $m_i$ et $m^0_j$ va s'écrire :
$d(m_i, m^0_j) = (m_{ik2} \otimes m^0_{jk2}) + (m_{ik3} \otimes m^0_{jk3}) + .... + (m_{ikd+1} \otimes m^0_{jk_{d+1}})$.

Les termes k qui restent sont $k_2, k_3, ... k_{d+1}$. On a d termes. Le théorème étant valable pour d, on doit avoir d monômes non concurrents représentants $m_i$ et $m^0_j$.

$m^1_j$ n'étant pas concurrent à $m^0_j$ et $m_i$, il en résulte qu'on a d+1 monômes pour représenter $m_i$ et $m_j$. **(CQFD)**

L'exemple qui suit illustre deux monômes concurrents avec d=3, tels qu'il n'y a pas de relation d'inclusion (i. e. $m_i \not\subset m_j$ **et** $m_j \not\subset m_i$). Il y a trois étapes d'éclatement et une étape d'élimination pour lesquels on obtient 4 monômes (d+1=4) non concurrents représentants $m_i$ et $m_j$.




**IV. 2. 3. Algorithme de génération de la non concurrence de PLAs :**

Pour plus de clarté, nous allons traiter un exemple avant de montrer l'algorithme d'éclatement de la concurrence utilisé.

Soient les deux monômes concurrents $m_i$ et $m_j$ suivants :

$m_i$= (1 0 X X 0 X 1) 100
$m_j$= (1 0 1 X 0 0 1) 001

Le monôme $m_i$ contient deux bits ignorés $m_{i3}=m_{i6}=X$, il doit être éclaté en remplaçant "X" par "0" et "1".

On procède en deux étapes de duplication puisque $d(m_i, m_j)=2$.

• Etape1 (duplication) :

$m_i$ donne : $m_i^0$ = (1 0 **0** X 0 X 1) 100
$m_i^1$ = (1 0 **1** X 0 X 1) 100
$m_j$ reste : $m_j$ = (1 0 1 X 0 0 1) 001

• Etape2 (duplication) :

$m_i^0$ reste : $m_i^0$ = (1 0 0 X 0 X 1) 100
$m_i^1$ donne : $m_i^{10}$ = (1 0 1 X 0 **0** 1) 100
$m_i^{11}$ = (1 0 1 X 0 **1** 1) 100
$m_j$ reste : $m_j$ = (1 0 1 X 0 0 1) 001

• Etape3 (élimination) :

$m_i^0$ reste : $m_i^0$ = (1 0 0 X 0 X 1) 100

$m_i^1$ donne : $m_i^{10}$ éliminé.

$m_i^{11}$ = (1 0 1 X 0 1 1) 100

$m_j$ donne : $m_j$ = (1 0 1 X 0 0 1) 101

**Figure IV.7 : Exemple de traitement.**

Etape1 : On a dupliqué $m_i$ en $m_i^0$ et $m_i^1$ pour lesquels $m_{i3}$ a été remplacé par "0" pour $m_{i3}^0$ et par "1" pour $m_{i3}^1$.

Etape2 : $m_i^1$ reste encore concurrent à $m_j$ car $m_{i6}^1$=X et $m_{j6}$=0, alors que les autres termes sont égaux. On duplique $m_i^1$ en $m_i^{10}$ et $m_i^{11}$ pour lesquels $m_{i6}^{10} = 1 = m_{i6}^{11}$.

Etape3 : On remarque que $m_i^{10}$ n'est autre que $m_j$, il y a redondance.

On élimine donc $m_i^{10}$ en joignant sa programmation OU (i. e. le vecteur de sortie correspondant) aux sorties correspondants à $m_j$

Après ces 3 étapes on obtient 3 monômes non concurrents représentants le deux monômes concurrents mi et mj.

L'algorithme de génération de la non concurrence qui va agir sur un fichier de description d'un PLA sera le suivant :

```
Tant Que i≠EOF Faire
|   Tant Que j≠EOF Faire
|   |  D(mi,mj)=Σk(mik⊕mjk)
|   |  Si D(mi,mj)>0 break;
|   |  |  Si D(mi,mj)=0
|   |  |  |  |  Pour k=0 à n Faire
|   |  |  |  |  |  Si (mik=X et mjk=0 ou 1) Alors
|   |  |  |  |  |  |(Dupliquer mi en m0i et m1i
|   |  |  |  |  |  |pour lequel m0ik=0 et m1ik=1)
|   |  |  |  |  |  |/* mi est remplacé par m0i et m1i est créé */
|   |  |  |  |  |  FinSi
|   |  |  |  |  |  Si ( mik =0 ou 1 et mjk =X) Alors
|   |  |  |  |  |  |(Dupliquer mj en m0j et m1j
|   |  |  |  |  |  |pour lequel m0jk=0 et m1jk =1)
|   |  |  |  |  |  |/* mj est remplacé par m0j et m1j est créé */
|   |  |  |  |  |  FinSi
|   |  |  |  |  FinPour
|   |  |  |  |  Si (mi=mj) Alors
|   |  |  |  |  |  (Eliminer mj en plaçant les sorties à "1" sur celles de mi)
|   |  |  |  |  FinSi
|   |  |  FinSi
|   |  FinSi
|   |  j=j+1;
|   FinTantQue
|   i=i+1;
FinTantQue
```

Cet algorithme a été écrit en langage C sous UNIX et a comme nom

CONCUR. Il procède par étapes. La première étape consiste à compter la distance D(mi, mj) entre chaque couple de monômes du PLA.

Quand la distance est positive, les deux monômes sont non concurrents.

Quand la distance est nulle, CONCUR éclate ces monômes en monômes non concurrents. Et si les monômes sont redondants on élimine l'un d'eux en plaçant la programmation OU de celui-ci dans l'autre. Quelques exemples d'exécution sont donné au paragraphe IV.4, où des temps d'exécution sont donnés dans le tableau IV.1.

Un petit exemple de PLA est montré en Figure IV.8, où le PLA d'origine a 9 monômes et après CONCUR, il en a 10 : il a fallu rajouter 1 monôme pour lever la concurrence.

CONCUR →

| | | | |
|---|---|---|---|
| .i 4 | | .i 4 | |
| .o 3 | | .o 3 | |
| .p 9 | | .p 10 | |
| 0011 | 001 | 0011 | 011 |
| 01-0 | 001 | 0100 | 101 |
| 101- | 100 | 0110 | 001 |
| -101 | 010 | 1010 | 100 |
| 001- | 010 | 1011 | 110 |
| 10-1 | 010 | 0101 | 111 |
| 100- | 010 | 1101 | 111 |
| 010- | 101 | 0010 | 010 |
| 1-01 | 101 | 1001 | 111 |
| .e | | 1000 | 010 |
| | | .e | |

**Figure IV.8 : Exemple d'éclatement d'un PLA.**

### IV. 3. Procédure de génération du codage des sorties de PLAs :

On a vu comment rendre un PLA non concurrent. On va décrire dans ce paragraphe les procédures de génération du codage des sorties. Les codages utilisés sont : code de Berger, code de Berger modifié, code double-rail, code m-parmi-n.

Soit ($B_1$, $B_2$, ..., $B_n$) les n sorties du PLA et ($B_{n+1}$, ..., $B_{n+p}$) les "p" sorties de codage du PLA. Soit ($Bj_1$, $Bj_2$, ..., $Bj_n$) la représentation binaire du mot de sortie du monôme mj, et ($Bj_{n+1}$, $Bj_{n+2}$, ..., $Bj_{n+p}$) la représentation binaire du mot de codage correspondant.

$Bj_k \in \{0, 1\}$ pour k=(1, 2, ..., n, n+1, ..., n+p).

### IV. 3. 1. Codage de Berger des sorties de PLA :

Pour un codage de Berger $p=\lceil \log_2(n+1) \rceil$ puisque $Bj_{n+1}$, $Bj_{n+2}$, ..., $Bj_{n+p}$ est la représentation binaire du nombre de zéros dans les bits d'information

$Bj_1$, $Bj_2$, ..., $Bj_n$.

La procédure de génération du code de Berger est :

Pour chaque monôme mj :

Etape 1 : Compter le nombre de croisements sans transistor MOS entre mj et chacune des sorties $Bj_1$, $Bj_2$, ..., $Bj_n$. Soit Nj le résultat. Alors $Bj_{n+1}$, $Bj_{n+2}$, ..., $Bj_{n+p}$ prendra la représentation binaire Nj comme résultat.

Etape 2 : Pour chaque sortie de codage $B_{n+k}$ ($k \in \{1, 2, ..., p\}$) :

Si $Bj_{n+k}=1$ alors un transistor MOS doit être implanté au croisement $B_{n+k}$ et $m_j$. Sinon $Bj_{n+k}=0$ et aucun transistor MOS n'est à implanter.

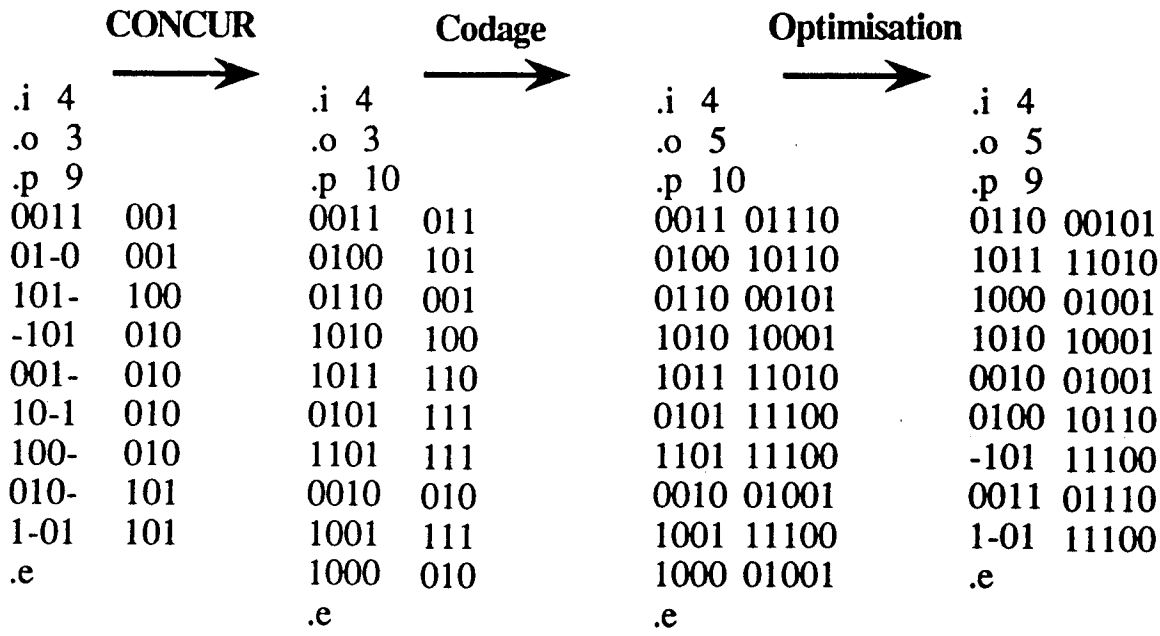



**Figure IV.9 : Exemple de codage par le code de Berger d'un PLA.**

**IV. 3. 2. Codage de Berger modifié des sorties de PLAs :**

Pour un codage de Berger modifié [MAK 82], $p= \lceil \log_2(m-s+1) \rceil$ où m et s sont respectivement le nombre maximum et minimum du nombre de 1 des mots de sortie du PLA.

$Bj_{n+1}$, $Bj_{n+2}$, ..., $Bj_{n+p}$ est la représentation binaire de la différence entre le nombre de zéros du mot et le nombre de zéros minimum.

Les nombres de zéros maximum et minimum seront respectivement n-s et n-m, ce qui explique qu'on aurait besoin de $\lceil \log_2((n-s)-(n-m)) + 1) \rceil = \lceil \log_2(m-s+1) \rceil$ bits de codage.

On remarque que l'on a un nombre de bits de codage inférieure à celui utilisé pour un codage de Berger classique. En effet, $m-s \leq n$, d'où $\lceil \log_2(m-s+1) \rceil \leq \lceil \log_2(n+1) \rceil$.

D'autre part $\lceil \log_2(m-s+1) \rceil < m-s$, où m-s est le nombre de bits de codage

à ajouter pour former un codage m-parmi-n.

En termes de nombre de bits de codage, le code de Berger modifié est performant sinon meilleure qu'un codage de Berger classique ou un codage m-parmi-n.

La procédure de génération du code de Berger modifié est :

Etape 1 : Pour chaque monôme mj, compter le nombre de croisements sans transistor MOS entre mj et chacune des sorties $B_1$, $B_2$, ..., $B_n$. Soit Nj le résultat. Etablir une liste des nombres Nj dans laquelle seront sélectionnés (n-s) et (n-m), (les nombres maximum et minimum de croisements sans transistors MOS).

Etape 2 : Pour chaque monôme mj faire Lj=Nj - (n-m).

$B^j_{n+1}$, $B^j_{n+2}$, ..., $B^j_{n+p}$ prendra la représentation binaire de Lj.

Etape 3 : La même que l'étape 2 de la procédure de génération du code de Berger.

**IV. 3. 3. Codage m-parmi-n des sorties de PLA :**

Un codage m-parmi-n des sorties demanderait p=m-s bits de codage. Les bits $B_{n+1}$, ..., $B_{n+p}$ vont servir à fixer à chaque fois le nombre de "1" (et par conséquent le nombre de "0") dans les mots de sortie.

Si "m" représente le nombre maximum de "1" que l'on peut avoir pour un mot d'information, si Nj est le nombre de zéros dans le mot d'information à coder, et par conséquent (n-Nj) le nombre de "1" dans ce mot, alors m-(n-Nj) va être le nombre de "1" à mettre dans le mot de codage.

L'assignement des bits de codage n'est pas unique, puisque ce qui importe c'est que dans le mot de codage on ajoute le nombre manquant de "1" dans n'importe quel ordre de façon à rendre tout le mot de sortie $B_1$, $B_2$, ..., $B_n$, $B_{n+1}$, ..., $B_{n+p}$ codé en m parmi n+p.

Un codage m-parmi-n peut se faire de plusieurs manières (contrairement au code de Berger ou au code de Berger modifié) tout en ayant la même redondance.

La procédure de génération du code m-parmi-n est :

Etape 1 : Pour chaque monôme mj, compter le nombre de croisements sans transistors MOS entre mj et chacune des sorties $B_1$, $B_2$, ..., $B_n$. Soit $N_j$ le résultat. Etablir une liste des nombres $N_j$ dans laquelle seront sélectionnés (n-s) et (n-m), les nombres maximum et minimum de croisements sans transistors MOS. On aura p=(m-s) bits de codage à rajouter.

Etape 2 : Pour chaque monôme mj faire $B^j_{n+1}$, ..., $B^j_{n+m-(n-Nj)}$=11 ... 1 et

$B^j_{n+m-(n-Nj)+1}$ ..... $B^j_{n+p}$= 00 ... 0.

Etape 3 : La même que l'étape 2 de la procédure de génération du code de Berger.

### IV. 3. 4. Codage double-rail des sorties de PLAs :

En ce qui concerne le code double-rail des sorties, on a une redondance maximum, en effet p est égal à n dans ce cas.

Chaque bit $B^j_{n+1}$, $B^j_{n+2}$, ..., $B^j_{n+p}$ d'un mot de sortie pour un monôme mj donné, représente le complémentaire correspondant dans le mot d'information de sortie $B^j_1$, $B^j_2$, ..., $B^j_n$.

i.e. ($B^j_{n+1}= \overline{B}^j_1$, $B^j_{n+2}= \overline{B}^j_2$, ..., $B^j_{n+p}= B^j_{2n}= \overline{B}^j_n$).

La procédure de génération du code double-rail est simple à implanter :

Pour chaque monôme mj et pour chaque sortie $B_{n+l}$ ($k \in \{1, 2, ..., p\}$) :

S'il n'y a pas de transistor MOS au croisement entre mj et $B^j_k$, mettre un transistor MOS au croisement entre mj et $B^j_{n+k}$. Sinon ne pas mettre de transistor MOS à ce croisement.

Cette procédure peut être effectuée de façon à avoir alternativement le bit d'information suivi de son code double-rail (complémentaire), de cette façon ces sorties attaqueront directement le contrôleur double-rail.

Dans le cas contraire elle peut être suivie par une procédure de réarrangement des sorties. (Figure IV.10).

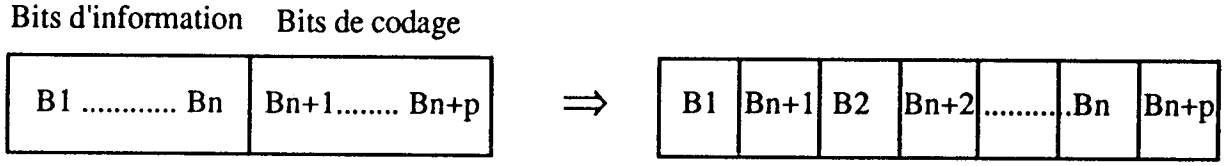



**Figure IV.10 : Sorties codées bit par bit en code double-rail.**

### IV. 4. Applications et résultats de PROTECT :

L'outil de génération du codage de PLA : PROTECT (a **PLA** **R**eorganizer in view of **O**n-line **TE**st using **C**oding **T**echniques) est un ensemble de procédures effectuant le codage de sorties de PLAs, son organisation est représenté en figure IV.10. Il a été appliqué à un ensemble standard de PLAs de Berkeley [PLA 87]. On a choisi cet ensemble de PLAs comme exemples car ils ont été traité récemment dans [SER 88] pour un codage de Berger par une méthode utilisant des outils classiques de synthèse logique. La comparaison de nos résultats avec ceux obtenus dans [SER 88] est donc possible.

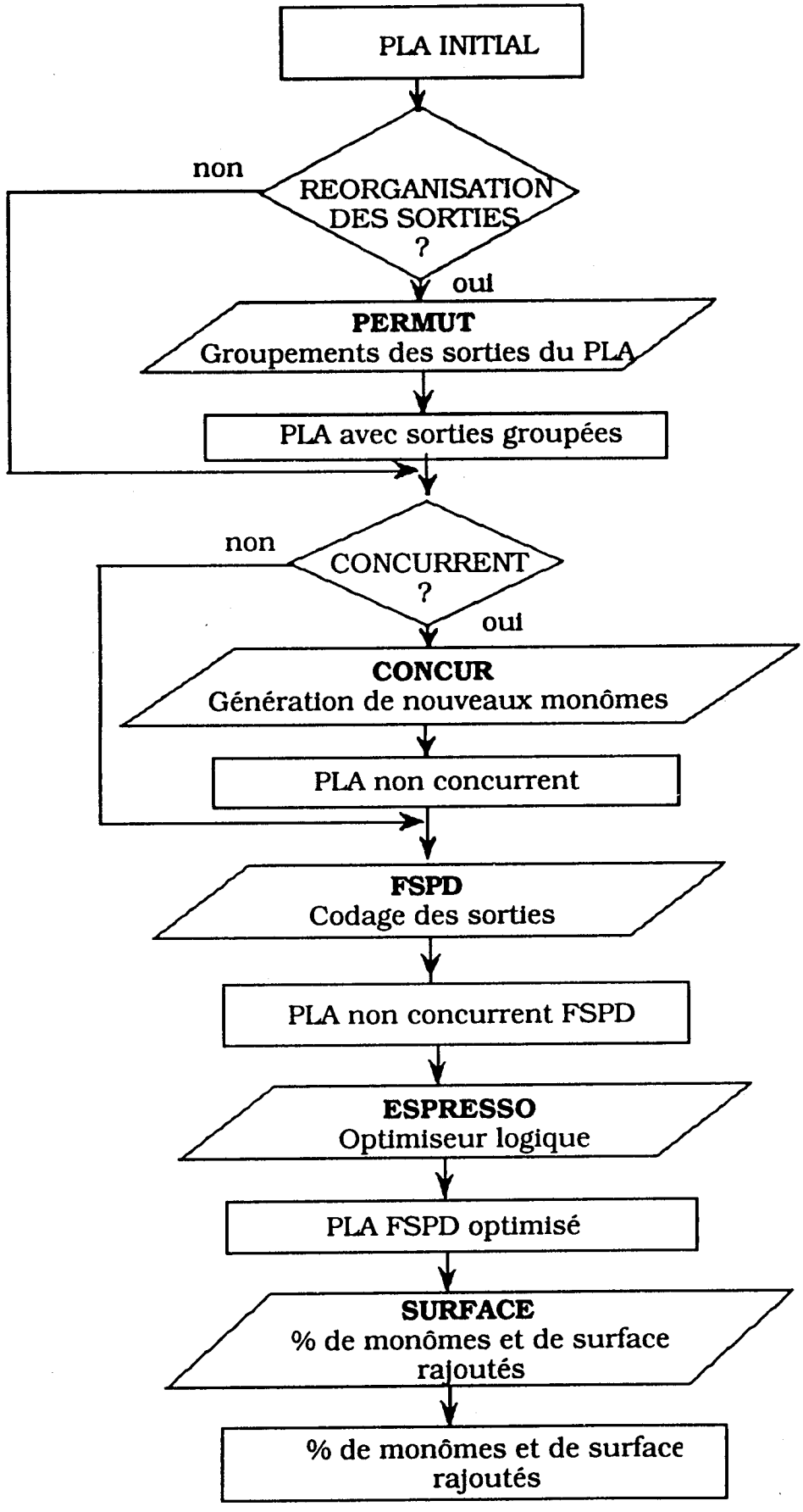



**Figure IV. 10 : Organisation de PROTECT.**

Dans le tableau IV.1, la première colonne donne le nom du PLA, les colonnes 2, 3 et 4 donnent respectivement le nombre d'entrées (i), de sorties (o) et de monômes (p) avant minimisation.

La colonne 5 donne le nombre de monômes (pmin) après minimisation avec l'optimiseur logique ESPRESSO.

La colonne 6 donne le nombre de monômes obtenus après codage de Berger et minimisation et le pourcentage de monômes rajoutés par rapport au PLA minimisé et non codé.

%pminBerger=[(pminBerger - pmin)/pmin] x 100%.

La colonne 7 donne la surface rajouté au PLA minimisé et non codé pour introduire son codage.

Enfin la colonne 8 donne le temps CPU nécessaire à PROTECT pour générer les PLAs codés et minimisés.

Les PLAs nommés "niv2", "niv1", et "niv0" sont les PLAs des trois tranches de contrôle de la partie contrôle du 60502 générée par SYCO, (voir chapitre V).

Dans les 2 dernières lignes du tableau IV.1, on a calculé les moyennes des pourcentages d'augmentation pour tous les PLAs (avant dernière ligne) et pour les PLAs sans tenir compte de "misex3". Ce dernier introduit 180% d'augmentation de monôme et 206% d'augmentation de surface, ce qui augmente considérablement la moyenne, puisque la moyenne d'augmentation des monômes est de 19.8% et devient 31.2% avec "misex3".

19% de moyenne d'augmentation de monômes pour PROTECT et pour l'ensemble des PLAs du tableau IV.2, représente moins de la moitié de la moyenne obtenue dans [SER 88] qui est de 42%, pour le même ensemble de PLAs, (voir Tableau IV.2). Ce qui est une performance.

D'autre part, les temps CPU de traitement sont inférieurs à la minute, c'est une performance de plus, obtenue grâce à une bonne programmation des algorithmes. Un cas particulier est le PLA "misex3", il a demandé un temps CPU de traitement de l'ordre de l'heure.

Les autres tableaux IV.3 IV.4, IV.5, représentent les traitements du même ensemble de PLAs respectivement, pour des codages de Berger modifié, m-parmi-n, et double-rail.

Dans le tableau IV.6 figurent les meilleurs résultats obtenus entre les différents codages non ordonnés (code de Berger, code m-parmi-n et code double-rail) obtenus dans les tableaux IV.1, IV.4, IV.5. La moyenne d'augmentation de surface de 27.6%, représente un pourcentage faible par rapport aux chiffres obtenus pour chaque codage séparément.

| | | | | | PROTECT | | |
|---|---|---|---|---|---|---|---|
| | | | | | **pmin (%)** | **% surface** | **Temps** |
| Nom | i | o | p | pmin | **Berger** | **Berger** | **CPU** |
| 9sym | 9 | 1 | 87 | 85 | **86** (1.17%) | 5% | 18".1 |
| bw | 5 | 28 | 87 | 22 | **22** (0%) | 13% | 2".9 |
| f2 | 4 | 4 | 12 | 8 | **10** (25%) | 56% | 0".4 |
| rd53 | 5 | 3 | 32 | 31 | **31** (0%) | 15% | 1".1 |
| rd73 | 7 | 3 | 141 | 127 | **127** (0%) | 11% | 7".3 |
| rd84 | 8 | 4 | 256 | 255 | **256** (0.4%) | 15% | 31".1 |
| sao2 | 10 | 4 | 58 | 58 | **58** (0%) | 12% | 21".2 |
| con1 | 7 | 2 | 9 | 9 | **18** (100%) | 125% | 0".9 |
| misex1 | 8 | 7 | 32 | 12 | **13** (8%) | 22% | 0".9 |
| misex3 | 14 | 14 | 1848 | 690 | **1932** (180%) | 206% | 57' 42".6 |
| 5xp1 | 7 | 10 | 75 | 64 | **115** (79%) | 109% | 34".2 |
| niv2 | 18 | 33 | 55 | 52 | **57** (9.6%) | 19% | 58".0 |
| niv1 | 23 | 28 | 166 | 146 | **165** (13%) | 20.6% | 2' 11".0 |
| niv0 | 39 | 32 | 170 | 133 | **161** (21%) | 27.6% | 14' 56".0 |
| MOYENNES | | | | | **31.2%** | **46.9%** | |
| Moyennes sans tenir compte de misex3 | | | | | **19.7%** | **34.7%** | |

**Tableau IV.1 : Résultats pour un codage en code de Berger.**

| | | | | | SERRA | **PROTECT** |
|---|---|---|---|---|---|---|
| | | | | | pmin (%) | **pmin (%)** |
| Nom | i | o | p | pmin | Berger | **Berger** |
| 9sym | 9 | 1 | 87 | 85 | 156 (83%) | **86** (1.17%) |
| bw | 5 | 28 | 87 | 22 | 16 (-27%) | **22** (0%) |
| f2 | 4 | 4 | 12 | 8 | 15 (87%) | **10** (25%) |
| rd53 | 5 | 3 | 32 | 31 | 32 (3%) | **31** (0%) |
| rd73 | 7 | 3 | 141 | 127 | 128 (0.8%) | **127** (0%) |
| rd84 | 8 | 4 | 256 | 255 | 256 (0.4%) | **256** (0.4%) |
| sao2 | 10 | 4 | 58 | 58 | 60 (3%) | **58** (0%) |
| con1 | 7 | 2 | 9 | 9 | 22 (144%) | **18** (100%) |
| misex1 | 8 | 7 | 32 | 12 | 18 (50%) | **13** (8%) |
| misex3 | 14 | 14 | 1848 | 690 | 1937 (181%) | **1932** (180%) |
| 5xp1 | 7 | 10 | 75 | 64 | 115 (79%) | **115** (79%) |
| MOYENNES | | | | | **55%** | **33%** |
| Moyennes sans tenir compte de misex3 | | | | | **42%** | **19%** |

**Tableau IV.2 : Comparaison des résultats pour un codage en code de Berger avec ceux obtenus dans [SER 88].**

| | | | | | | PROTECT | |
|---|---|---|---|---|---|---|---|
| | | | | | MBerger | pmin (%) | % surface |
| Nom | i | o | p | pmin | o | MBerger | MBerger |
| 9sym | 9 | 1 | 87 | 85 | 2 | **86** (1.17%) | 6.5% |
| bw | 5 | 28 | 87 | 22 | 31 | **22** (0%) | 8% |
| f2 | 4 | 4 | 12 | 8 | 5 | **10** (25%) | 35% |
| rd53 | 5 | 3 | 32 | 31 | 4 | **31** (0%) | 7% |
| rd73 | 7 | 3 | 141 | 127 | 5 | **127** (0%) | 11% |
| rd84 | 8 | 4 | 256 | 255 | 6 | **256** (0.4%) | 10% |
| sao2 | 10 | 4 | 58 | 58 | 6 | **58** (0%) | 8% |
| con1 | 7 | 2 | 9 | 9 | 3 | **18** (100%) | 112% |
| misex1 | 8 | 7 | 32 | 12 | 10 | **13** (8%) | 22% |
| misex3 | 14 | 14 | 1848 | 690 | 18 | **1933** (180%) | 206% |
| 5xp1 | 7 | 10 | 75 | 64 | 14 | **119** (85%) | 117% |
| niv2 | 18 | 33 | 55 | 52 | 37 | **56** (7.7%) | 13.9% |
| niv1 | 23 | 28 | 166 | 146 | 32 | **165** (13%) | 19% |
| niv0 | 39 | 32 | 170 | 133 | 35 | **161** (21%) | 24.3% |
| MOYENNES | | | | | | **31.4%** | **42.7%** |
| Moyennes sans tenir compte de misex3 | | | | | | **20%** | **30.3%** |

**Tableau IV. 3 : Résultats pour un codage en code de Berger modifié.**

| | | | | | | PROTECT | |
|---|---|---|---|---|---|---|---|
| | | | | | m-parmi-n | pmin (%) | % surface |
| Nom | i | o | p | pmin | o | m-parmi-n | m-parmi-n |
| 9sym | 9 | 1 | 87 | 85 | 2 | **86** (1.17%) | 6.5% |
| bw | 5 | 28 | 87 | 22 | 35 | **22** (0%) | 18% |
| f2 | 4 | 4 | 12 | 8 | 5 | **10** (25%) | 35% |
| rd53 | 5 | 3 | 32 | 31 | 4 | **31** (0%) | 7% |
| rd73 | 7 | 3 | 141 | 127 | 5 | **127** (0%) | 11% |
| rd84 | 8 | 4 | 256 | 255 | 7 | **256** (0.4%) | 15% |
| sao2 | 10 | 4 | 58 | 58 | 6 | **58** (0%) | 8% |
| con1 | 7 | 2 | 9 | 9 | 3 | **18** (100%) | 112% |
| misex1 | 8 | 7 | 32 | 12 | 11 | **13** (8%) | 27% |
| misex3 | 14 | 14 | 1848 | 690 | 23 | **1649** (138%) | 190% |
| 5xp1 | 7 | 10 | 75 | 64 | 18 | **112** (75%) | 133% |
| niv2 | 18 | 33 | 55 | 52 | 45 | **56** (7.7%) | 26.4% |
| niv1 | 23 | 28 | 166 | 146 | 38 | **165** (13%) | 28.2% |
| niv0 | 39 | 32 | 170 | 133 | 39 | **160** (20.3%) | 27.9% |
| MOYENNES | | | | | | **27.7%** | **46%** |
| Moyennes sans tenir compte de misex3 | | | | | | **19.3%** | **35%** |

**Tableau IV. 4 : Résultats pour un codage en code m-parmi-n.**

| | | | | | PROTECT | |
|---|---|---|---|---|---|---|
| | | | | | pmin (%) | % surface |
| Nom | i | o | p | pmin | 2-rail | 2-rail |
| 9sym | 9 | 1 | 87 | 85 | **86** (1.17%) | 6.5% |
| bw | 5 | 28 | 87 | 22 | **22** (0%) | 73% |
| f2 | 4 | 4 | 12 | 8 | **10** (25%) | 66% |
| rd53 | 5 | 3 | 32 | 31 | **31** (0%) | 23% |
| rd73 | 7 | 3 | 141 | 127 | **127** (0%) | 17% |
| rd84 | 8 | 4 | 256 | 255 | **256** (0.4%) | 20% |
| sao2 | 10 | 4 | 58 | 58 | **60** (3%) | 20% |
| con1 | 7 | 2 | 9 | 9 | **17** (89%) | 112% |
| misex1 | 8 | 7 | 32 | 12 | **14** (16%) | 52% |
| misex3 | 14 | 14 | 1848 | 690 | **1933** (180%) | 273% |
| 5xp1 | 7 | 10 | 75 | 64 | **74** (15%) | 63% |
| niv2 | 18 | 33 | 55 | 52 | **56** (7.7%) | 59.2% |
| niv1 | 23 | 28 | 166 | 146 | **164** (12.3%) | 54.8% |
| niv0 | 39 | 32 | 170 | 133 | **172** (29.3%) | 67% |
| MOYENNES | | | | | **27%** | **64.7%** |
| Moyennes sans tenir compte de misex3 | | | | | **15.3%** | **48.7%** |

**Tableau IV. 5 : Résultats pour un codage en code double-rail.**

| | | | | | PROTECT | |
|---|---|---|---|---|---|---|
| | | | | | CODAGE | % surface |
| Nom | i | o | p | pmin | MINIMAL | |
| 9sym | 9 | 1 | 87 | 85 | Berger | 6.5% |
| bw | 5 | 28 | 87 | 22 | Berger | 13% |
| f2 | 4 | 4 | 12 | 8 | m-parmi-n | 35% |
| rd53 | 5 | 3 | 32 | 31 | m-parmi-n | 7% |
| rd73 | 7 | 3 | 141 | 127 | Berger | 11% |
| rd84 | 8 | 4 | 256 | 255 | Berger | 15% |
| sao2 | 10 | 4 | 58 | 58 | m-parmi-n | 8% |
| con1 | 7 | 2 | 9 | 9 | m-parmi-n | 112% |
| misex1 | 8 | 7 | 32 | 12 | Berger | 22% |
| misex3 | 14 | 14 | 1848 | 690 | Berger | 206% |
| 5xp1 | 7 | 10 | 75 | 64 | Double-rail | 63% |
| niv2 | 18 | 33 | 55 | 52 | Berger | 19% |
| niv1 | 23 | 28 | 166 | 146 | Berger | 20.6% |
| niv0 | 39 | 32 | 170 | 133 | Berger | 27.6% |
| MOYENNES | | | | | | **40.4%** |
| Moyennes sans tenir compte de misex3 | | | | | | **27.6%** |

**Tableau IV. 6 : Résultats minimaux pour un codage non ordonné.**

**IV. 5 Evitement d'une concurrence particulière dans les PLAs :**

Une concurrence particulière existe dans les PLAs générés actuellement par SYCO. Elle peut tout aussi bien exister dans des PLAs fonctionnant en machine d'état fini où il y aurait eu fusionnement d'états.

On présente d'abord le type d'optimisation utilisé et le résultat de cette optimisation. Une méthode "matérielle" levant la concurrence sera proposée par la suite.

Cette méthode n'introduit pas d'ajout de lignes d'entrée, ni de lignes de sortie, ni de lignes de monôme.

Un tel évitement permet de coder les sorties du PLA par le code détecteur désiré sans avoir à rajouter de monômes.

**IV. 5. 1 Optimisation logique des PLAs de SYCO :**

Une optimisation logique est effectuée dans le compilateur de parties contrôle (CPC).

A partir de la description algorithmique des états le CPC génère la table de vérité du PLA.

La description algorithmique est un algorithme qui contient chaque état d'entrée, et chaque état est composé de différentes étapes avec son action correspondante :

<state1> (If cond1 action1;END;
If cond2 action2;END;)

L'optimisation logique effectuée par le CPC est locale à chaque état. Elle concerne 2 ou plusieurs étapes qui pourraient être fusionnées.

**Définition** : *Deux étapes peuvent être fusionnées si une des conditions suivantes est vérifiée :*

- *Ces étapes ont les mêmes conditions cond1=cond2 et les actions correspondantes sont compatibles (actions qui pourraient s'exécuter en parallèle).*
- *La condition d'une étape est incluse dans l'autre et les actions sont égales.*

Dans [MHA 87] est représenté un exemple où 2 étapes sont fusionnées.

<seq 32> (IF(a=0 and b=1) EXECUTE point1; ENDIF;
IF(a=0) EXECUTE point1; ENDIF;)

Ces 2 étapes sont fusionnées en :

<seq 32> (IF(a=0) EXECUTE point1;ENDIF;)

Lorsque se présente le cas suivant :

<seq 32> (IF(a=0 and b=1) ACTION1; ENDIF;
IF(a=0 and c=0) ACTION2; ENDIF;)

Il peut y avoir lors du fonctionnement normal 2 situations :

a) Si b=1 et c=0 sont vrais en même temps, alors les 2 étapes précédentes sont vraies et sont activées en même temps, alors la condition pour avoir une opération correcte serait d'avoir action1 compatible avec action2 (i. e. action1 et action2 exécutées en parallèle et activant des modules différents). Sinon on a un problème qui peut être détecté dynamiquement par simulation.

b) Si le cas b=1 et c=0 ne peut pas survenir pendant le fonctionnement normal, il n'y a pas de problème de concurrence, puisque les deux étapes ne peuvent pas être activées en même temps.

Dans le cas a) on peut avoir des actions (action1 et action2) compatibles, et donc exécutées en parallèle, et les étapes (a=0 et b=1) et (a=0 et c=0) sont le résultat d'une optimisation logique, comme par exemple la fusion d'étapes:

Si l'on considère les étapes sans fusion on a :

<seq 32> (IF(a=0 and b=1 and c=1) ACTION1; ENDIF;
IF(a=0 and b=0 and c=0) ACTION2; ENDIF;
IF(a=0 and b=0 and c=1) *pas d'opération (cas imprévu)*
IF(a=0 and b=1 and c=0) ACTION1; ACTION2; ENDIF;)

On peut voir que la quatrième étape est non nécessaire si action1 et action2 sont exécutées en parallèle et si on a besoin de action1 et action2.

Ceci veut dire que l'on décrit n'importe quel type d'étape avec les opérations désirées, et plusieurs autres cas sont possibles où plus d'une étape est activée en même temps.

**IV. 5. 2 Propriétés de la concurrence particulière dans les PLAs :**

On va représenter cette concurrence particulière dans la table de PLA plutôt qu'algorithmiquement.

On rappelle que chaque étape est représentée dans le PLA comme monôme. Chaque monôme est divisé en plusieurs champs. Dans l'exemple qui va suivre on a 3 champs, un pour les instructions, un pour les conditions et le troisième pour le séquencement interne.

Une concurrence particulière peut apparaître lorsqu'un monôme d'un état est sans condition alors que d'autres monômes du même état ont des conditions.

**Propriété IV. 1 :** *Tous les monômes à condition sont au moins adjacents entre eux (donc non concurrents entre eux) lorsque leur actions correspondantes ne sont pas compatibles.*

Si une simulation a été faite, cette propriété doit en principe être valable lors du fonctionnement normal.

Cette propriété n'est pas valable pour des couples de monômes contenant le ou les monômes sans condition.

Soit l'exemple typique suivant :

| Entrées | | | Sorties | | | | |
|---|---|---|---|---|---|---|---|
| Inst | Cond | seq | | | | | |
| 011010101010 | 00000000000000000000000000000000 | 01010101 | 000101010101 | 1 | 0110001 | 0000 | *monôme 1* |
| 011010101010 | 00000000100101010000000000000000 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 2* |
| 011010101010 | 00000000001010100000000001010101 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 3* |
| 011010101010 | 00000000001010100000000001010101 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 4* |
| 011010101010 | 00000000000110100000000000001000 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 5* |
| 011010101010 | 00000000000110010000000000000100 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 6* |
| 011010101010 | 00000000000101100000000000000010 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 7* |
| 011010101010 | 00000000000101010000000000000001 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 8* |
| 011010101010 | 00000000001001101000000000000000 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 9* |
| 011010101010 | 00000000001001010100000000000000 | 01010101 | 000000000000 | 1 | 0111010 | 0000 | *monôme 10* |

**Figure IV. 11 : Exemple type de concurrence particulière.**

Le monôme sans condition est représenté en première ligne. Tous les monômes suivants (2, 3, ....) ont une condition.

Chaque fois qu'un monôme à condition est activé, le monôme sans condition (monôme 1) l'est aussi, et les sorties sont compatibles puisqu'elles sont complémentaires, il y a un champ des sorties où elles prennent les mêmes valeurs et un autre champ où le monôme 1 impose ses valeurs.

**Propriété IV. 2 :** *Les monômes sans condition peuvent être activés seuls par un ensemble de vecteurs du code d'entrée. (ces vecteurs ont un champ de condition différent de celui des vecteurs activant les monômes à condition).*

**Propriété IV. 3 :** *Quand un des monômes à condition est activé par un vecteur du code d'entrée, le monôme sans condition l'est aussi. En plus des sorties activées par le monôme sans condition, d'autres sorties le sont par le monôme à condition.*

Ces 2 propriétés décrivant la concurrence particulière que l'on a observée. Et pour éviter cette concurrence, une solution radicale consiste à

ajouter d'autres monômes pour lever cette concurrence. Mais une question se pose concernant l'augmentation en monôme et en surface qu'une telle solution donnerait. Il n'y a pas de réponse correcte à cette question, si ce n'est que l'augmentation en monôme dépend de la longueur du champ de condition et du nombre de monômes à condition qui vérifient les 3 propriétés précédentes.

Néanmoins, on propose une solution qui traduirait une telle concurrence en une non concurrence, suivant la proposition suivante :

**Proposition IV.1 :**

*Pour rendre des monômes, vérifiant les 3 propriétés précédentes, non concurrents, on doit désactiver le monôme sans condition chaque fois qu'un monôme à condition est activé. Les sorties correspondantes doivent être associé à ce dernier. (i. e. on doit ajouter la même programmation des sorties qui existe sur le monôme sans condition aux monômes à condition).*

Cette proposition se résume comme suit :

- Déconnexion du monôme sans condition.
- Génération des sorties correspondantes.

1) Déconnexion du monômes sans condition :

On propose un rajout de logique qui déconnecterait la ligne de monôme sans condition chaque fois que la ligne de monôme à condition est activée. Deux solutions équivalentes peuvent effectuer cette opération. Elles sont représentées en figure IV.12 pour lesquelles la circuiterie rajoutée est représentée en gras.

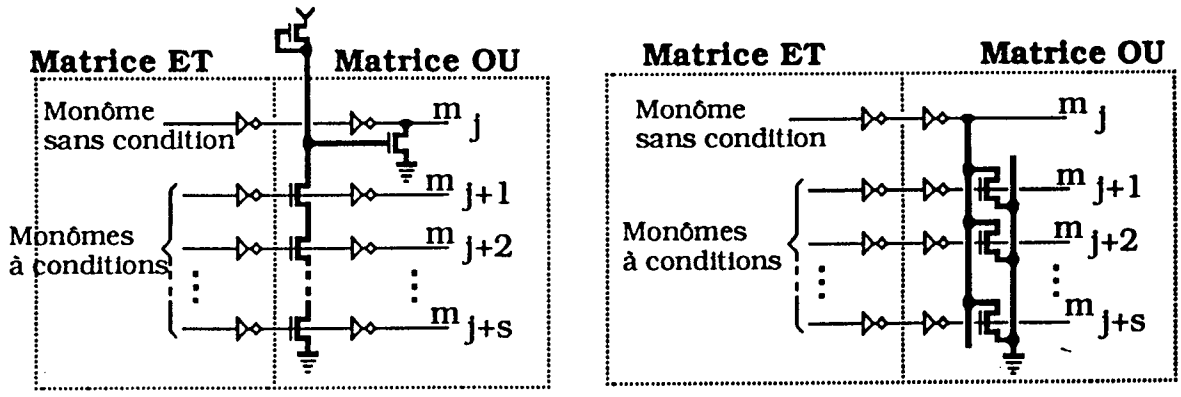



**Figure IV. 12 : Evitement dans une concurrence particulière dans un PLA.**

La deuxième solution représentée en figure IV. 12 b) est préférable car elle demande un ajout minimal de transistors et est plus régulière à implémenter.

Si l'un des monômes à condition est activé il prend la valeur "1" et mettrait à zéro le monôme sans condition.

2) Génération des sorties correspondantes :

La deuxième étape concerne la génération des sorties correctes correspondantes aux monômes à condition. La figure IV. 13 montre la table du PLA représenté en figure IV. 11 où l'on a rajouté à chaque monôme à condition la programmation manquante des sorties.

| Entrées | | | Sorties | | |
|---|---|---|---|---|---|
| Inst | Cond | seq | | | |
| 011010101010 | 00000000000000000000000000000000 | 01010101 | 000101010101 | 1 | 0110001 |
| | 0000 monôme 1 | | | | |
| 011010101010 | 00000000100101010000000000000000 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000001010100000000001010101 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000001010100000000001010101 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000000110100000000000001000 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000000110010000000000000100 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000000101100000000000000010 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000000101010000000000000001 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000001001101000000000000000 | 01010101 | 000101010101 | 1 0111011 | 0000 |
| 011010101010 | 00000000001001010100000000000000 | 01010101 | 000101010101 | 1 0111011 | 0000 |

**Figure IV. 13 : Génération des sorties correspondantes.**

Les sorties qui sont activées par le monôme sans condition doivent être générées par les monômes à condition.

L'activation d'une ligne de sortie est obtenue lorsqu'un MOS existe dans le croisement de cette sortie avec le monôme activé, et est représenté par un "1" dans la figure IV. 11 et figure IV. 13.

Soit $m_j$ le monôme sans condition et soit $B_i$ une ligne de sortie.

Soit $m_{j+1}$, $m_{j+2}$, ...., $m_{j+s}$ les monômes à condition.

Initialement, on a un encodage de la table avec une programmation des sorties générées par $m_j$ et par $m_{j+1}$, $m_{j+2}$, ...., $m_{j+s}$.

Pour rendre vraie la proposition IV.1 on procède de la manière suivante :

Chaque fois qu'il y a un transistor MOS entre $m_j$ et $B_i$ (un "1" dans la table de la figure IV. 11), mettre un transistor MOS ( un "1" dans la table de la figure IV. 13) au croisement entre $m_{j+1}$ et $B_i$, entre $m_{j+2}$ et $B_i$, ....., entre $m_{j+s}$ et $B_i$.

Cette procédure vérifie la seconde partie de la proposition IV.1.

On obtient alors une non concurrence et les procédures du paragraphe II.3, de génération des codage des sorties, sont alors applicables.

**IV. 5. 3 Propriété FSPD pour l'évitement de la concurrence :**

En figure IV. 12 sont proposés deux schémas d'évitement de la concurrence. Pour ces deux schémas on a un problème concernant la parité d'inversion de différents chemins reliant les monômes aux sorties.

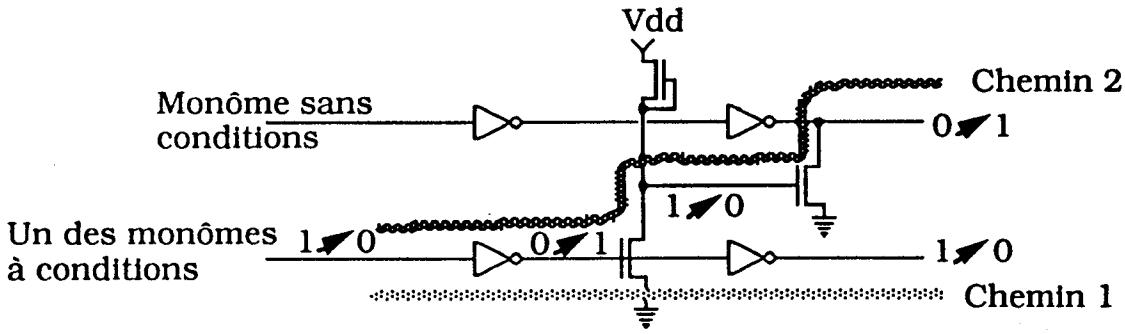



**Figure IV. 14 : Parité d'inversion du circuit d'évitement de concurrence.**

Sur la figure IV. 14, le chemin 1 a une parité d'inversion égal à "0", alors que pour la chemin 2 la parité d'inversion est "1".

La figure IV. 14 montre un exemple de collage à "0" d'un monôme à condition et ses conséquences jusqu'à la programmation OU.

Une telle panne simple induit une erreur double et non unidirectionnelle sur les lignes de monômes, et la propagation jusqu'aux sorties n'est pas unidirectionnelle, puisque la parité d'inversion est fixe (égale à "1") entre les monômes et les lignes de sortie.

Pour être FSPD, la circuiterie ajoutée doit être dupliquée, et tous les monômes à condition doivent aussi être dupliqués. Ces duplications introduisent généralement moins de surface qu'en effectuant un éclatement de ces monômes pour les rendre non concurrents.

**IV. 6 Analyse de signature de PLAs :**

L'implémentation de blocs BILBO aux entrées et sorties de structures à base de PLAs ou de ROMs (figure IV. 15) nécessite la prévision de la signature pour une séquence donnée et pour un PLA donné.

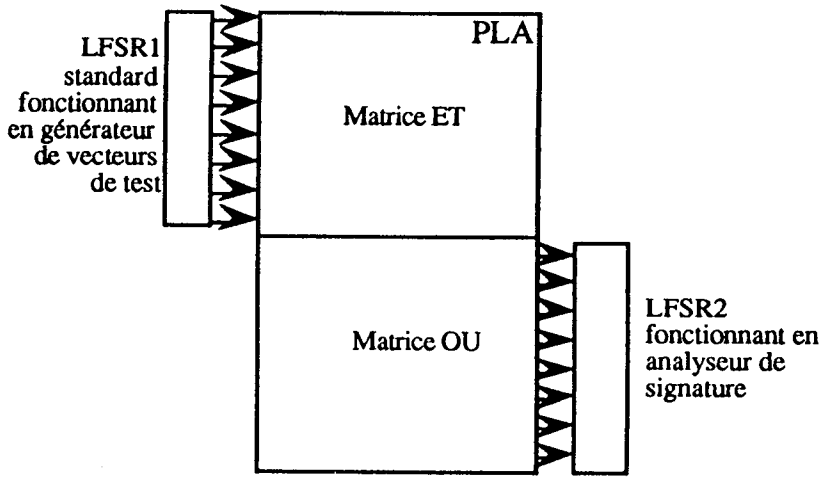



**Figure IV. 15 : Structure BIST de PLA pour un test exhaustif ou pseudo-aléatoire.**

Une façon de calculer la signature serait d'utiliser un simulateur logique (e. g. HILO, SILOS, ...). Mais le temps CPU utilisé pour ce calcul est déraisonnable pour des PLAs de grande taille.

On a donc cherché à concevoir à implanter dans la chaîne d'outils manipulant les tables de PLAs, un simulateur spécialisé calculant la signature.

Ce simulateur extrait automatiquement la nombre d'entrées et de sorties, et assigne à partir de ces informations les LFSRs à polynôme primitif correspondants (voir annexe 4) [WAN 86].

Il masque les bits ignorés du PLA (i. e. les "X" ou "-").

En lui spécifiant s'il s'agit d'un test exhaustif ou pas, il calcule les séquences de test aux entrées du PLA, les réponses à ses sorties et la signature en fin de séquence.

La procédure de calcul de signature est représentée par l'organigramme de la figure IV. 16.

Il est évident de voir qu'un vecteur V généré par l'LFSR1 et activant un monôme M du PLA vérifie la relation $(V \oplus M1) \wedge M2=0$ où M1 et M2 sont extraits de M :

- M1 est le même que M où les bits ignorés sont remplacées par "1".
- Dans M2 les bits ignorés sont remplacées par "0" et tous les autres bits par "1".

Les opérateurs $\oplus$ (OU exclusif) et $\wedge$ (ET) peuvent être effectuée sur une représentation entière de V, M1 et M2. Ce qui nous évite un calcul bit par bit entre V et M.

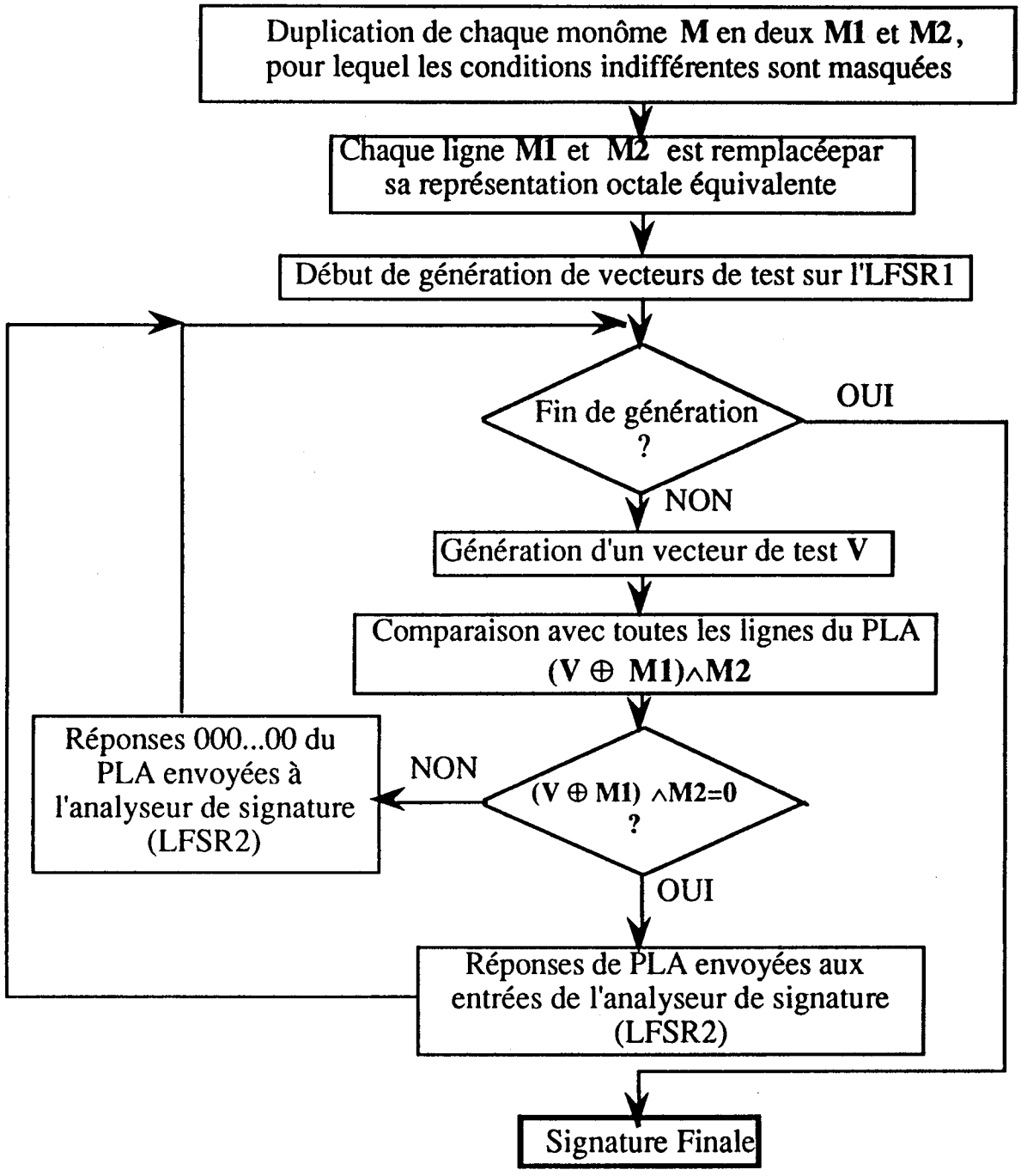



**Figure IV. 16 : Calcul de la signature d'un PLA pour un test exhaustif.**

La technique de "masquage" des bits ignorés est quelque peu connue, mais son application sous cette forme est originale et nous a permis de développer un outil performant et rapide.

Un test exhaustif d'exemples de PLAs justifie ces considérations. En effet, en tableau IV.5 est donné un ensemble de PLAs. i, o, p, pmin représentent respectivement le nombre d'entrées, de sorties, de monômes, de monômes après optimisation logique. Les 2 dernières colonnes représentent le nombre de vecteurs, pour une séquence maximale, envoyés

au PLA et le temps CPU utilisé pour le calcul de la signature pour cette séquence.

| Nom | i | o | p | pmin | Nombre de vecteurs($2^i$) | Temps CPU |
|---|---|---|---|---|---|---|
| 9sym | 9 | 1 | 87 | 85 | 512 | 1.14s |
| bw | 5 | 28 | 87 | 22 | 32 | 0.16s |
| f2 | 4 | 4 | 12 | 8 | 16 | 0.04s |
| rd53 | 5 | 3 | 32 | 31 | 32 | 0.06s |
| rd73 | 7 | 3 | 141 | 127 | 128 | 0.5s |
| rd84 | 8 | 4 | 256 | 255 | 256 | 1.78s |
| sao2 | 10 | 4 | 58 | 58 | 1024 | 1.66s |
| con1 | 7 | 2 | 9 | 9 | 128 | 0.08s |
| misex1 | 8 | 7 | 32 | 12 | 256 | 0.22s |
| misex3 | 14 | 14 | 1848 | 690 | 16 384 | 245s |
| 5xp1 | 7 | 10 | 75 | 64 | 128 | 0.3s |
| niv2 | 18 | 33 | 110 | 110 | 262 144 | 629s≈11mn |
| niv1 | 23 | 28 | 166 | 166 | 8 388 608 | ≈13Heures |

**Tableau IV.5 : Temps CPU de calcul de signature de PLAs pour un test exhaustif.**

**Conclusion :**

On a proposé dans ce chapitre des outils de génération de PLAs FSPD, et de calcul de leur signature.

Ces outils sont opérationnels et efficaces, ils constituent les outils clés agissant dans une chaîne d'outils de CAO, tel qu'un compilateur de silicium.

Une comparaison entre les résultats de PROTECT et ceux obtenus dans [SER 88] montre l'efficacité de PROTECT. En effet, deux fois moins d'augmentation de monômes a été nécessaire pour PROTECT, pour coder les PLAs en code de Berger, par rapport aux résultats obtenus dans [SER 88].

PROTECT transforme la description d'un PLA donné en une description non concurrente. Il est montré au théorème IV.1, que cette description non concurrente est optimale. Ceci permet une meilleure minimisation logique, par ESPRESSO, du PLA après codage des sorties.

L'outil de simulation de signature utilise une technique de masquage des bits ignorés permettant une description quasi-vectorielle des monômes, et permettant un calcul rapide de la signature d'un PLA pour un test exhaustif ou pseudo-aléatoire. Cet outil assigne automatiquement les polynômes diviseurs (stockés dans un fichier-bibliothèque), de l'LFSR d'entrée (générateur de vecteurs de test), et de l'LFSR de sortie (analyseur de signature).

# CHAPITRE V

**Implémentation UBIST de parties contrôle**

**Introduction :**

Dans ce chapitre, on propose une stratégie de conception en vue d'autotest pour des parties contrôle hiérarchiques à base de PLAs.

Une partie contrôle de ce type est composée d'un empilement de tranches de contrôle, chacune d'elles représente une machine d'états finis implémentée sous forme de PLA séquentiel.

Ce type d'architecture a été proposé dans [ANC 83] et correspond à l'architecture cible adoptée par le compilateur de silicium SYCO [JER 88], [JER 90].

Modifier cette architecture pour insérer des blocs et circuiteries de test intégré, est une tâche qui, arrivée à son terme doit être rendue systématique pour toute partie contrôle basée sur cette architecture.

Un schéma UBIST est proposé pour des parties contrôle hiérarchiques:

- Où les blocs fonctionnels ont été rendus FSPD par l'utilisation de codage des sorties de PLAs, et conception duale des différents circuits anarchiques (variables de contrôle, bus de contrôle et de comptes rendus, blocs de synchronisation).
- Des contrôleurs sont introduits de façon originale dans la structure topologique de la partie contrôle, évitant des routages superflus et des pertes de surface.
- Les registres d'entrée et de sortie sont modifiés sous forme de BILBOs assurant la génération de vecteurs de test et l'analyse de signature, effectuant des phases de test hors-ligne des PLAs.

**V. 1 Organisation et description de la partie contrôle :**

Une partie contrôle hiérarchique est constituée d'une pile de tranches de contrôle (figure V. 1). La tranche (i) exécute les instructions générées par la tranche (i+1). Elle découpe une instruction en "sous-instructions" qui seront exécutées par la tranche (i-1). Une instruction correspond à un cycle de contrôle.

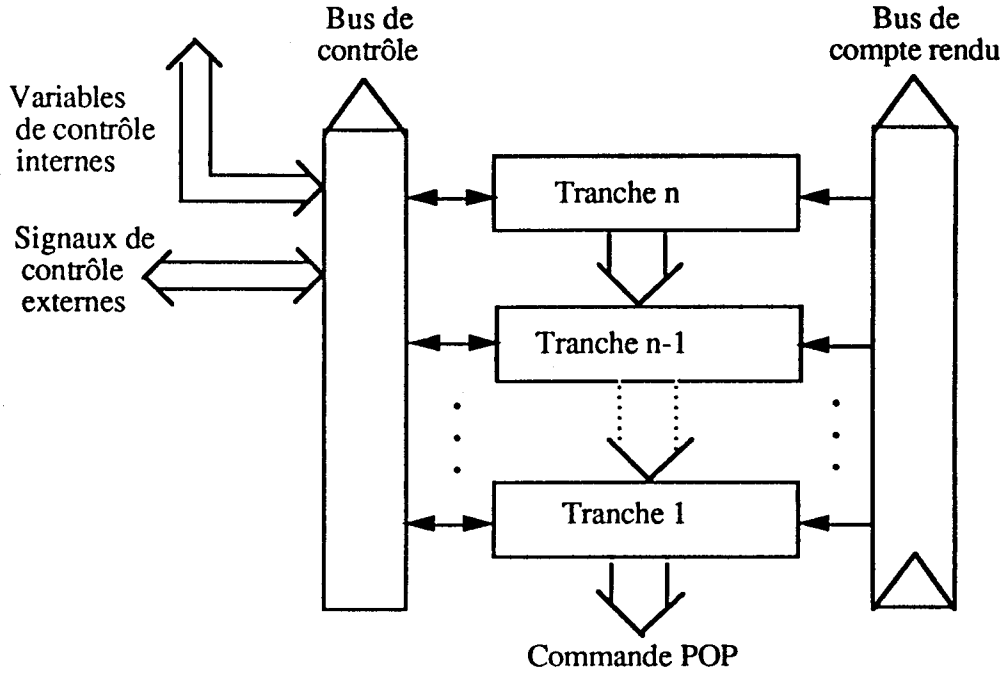



**Figure V. 1 : Organisation globale de la partie contrôle.**

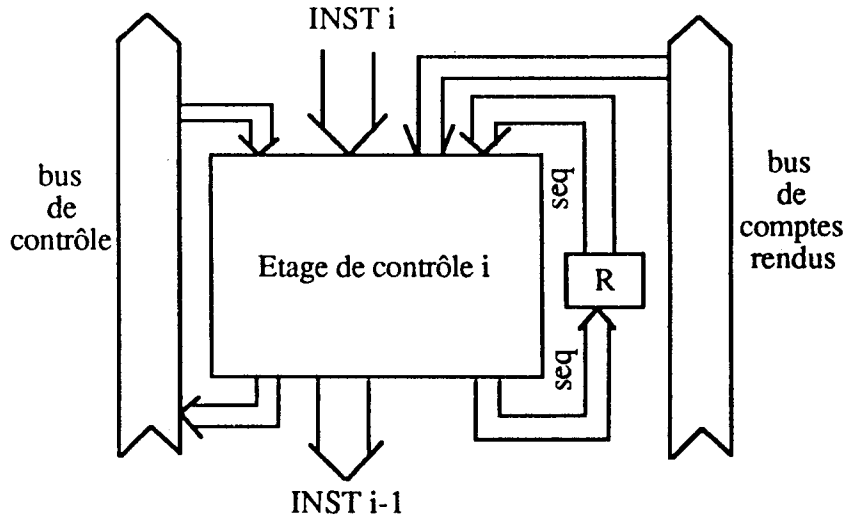



**Figure V. 2 : Organisation d'une tranche de contrôle.**

On utilise un cycle à 4 phases d'horloge (T1, T2, T3, T4) pour chaque tranche de contrôle. Le cycle est imposé par la partie opérative.

Pendant T1 on mémorise les entrées, T2 et T3 sont utilisées pour calculer les sorties qui sont mémorisées en T4.

Les tranches de contrôle communiquent à l'aide de deux bus, le bus de contrôle et le bus de compte rendu d'exécution.

Le bus de compte rendu d'exécution est utilisé comme entrée des tranches. Les signaux véhiculés par ce bus sont générés par la partie opérative et correspondent en entrée des tranches de contrôle aux conditions.

Le bus de contrôle peut être utilisé comme entrée ou sortie pour les tranches. Les valeurs véhiculées par ce bus correspondent aux variables de contrôle, aux signaux d'horloge, et aux signaux de synchronisation interne.

Les entrées et les sorties de chaque tranche sont:

Les entrées :

• Inst : sont les instructions générées par la tranche de contrôle supérieure. (Ce type d'entrée n'existe pas pour la tranche supérieure).

• Seq : sont les entrées de condition. Elles sont de différents types:

* comptes rendus d'exécution de la partie opérative.

* Variables de contrôle : (externes venant des plots de sortie, internes mémorisées dans les registres de contrôle, et variables internes de synchronisation générées par le bloc de synchronisation).

Les sorties :

• InstInf : sont les instructions envoyées à la tranche inférieure et correspondant pour celle-ci à ses entrées de type Inst. Pour la tranche activant la partie opérative, ses sorties correspondent aux signaux de commande de la partie opérative.

• Seq : sont les sorties de séquencement interne, elles vont aux entrées de même type de la tranche de contrôle.

• Variables de contrôle: elles sont de deux types : Celles mémorisées dans le champs de registre de variables de contrôle, et celles allant au bloc de synchronisation.

### V. 2 Détails de l'implémentation UBIST d'une tranche de contrôle:

Intégrer des blocs de test dans une topologie donnée pose un problème classique d'architecture et de conception tel que la non dégradation de la vitesse du circuit, la mutuelle insertion des blocs ne laissant pas de lacunes, une faible consommation. Une bonne couverture de pannes est à ajouter à toute ces contraintes, rendant encore plus délicat l'élaboration d'une architecture UBIST.

L'organisation générale d'une tranche de contrôle est montrée en figure V.2. Plusieurs solutions sont possibles pour l'organiser. Le choix d'une organisation donnée dépend de certaines exigences, tels que trouver une

bonne topologie pour pouvoir assembler la tranche de contrôle avec les autres blocs du circuit sans grande perte de surface, trouver des algorithmes efficaces pour sa génération automatique, permettre l'établissement de fonctions simples et rapides pour l'évaluation des performances afin d'offrir un choix entre plusieurs solutions.

Ces exigences font que certaines bonnes solutions connues sont difficilement automatisables. Par exemple, il est difficile de trouver un modèle topologique et donc d'établir des fonctions d'évaluation de performances [JER 89] pour le modèle de partie contrôle avec générateur de temps [OBR 82].

Par contre, les modèles basés sur les structures programmables (PLA, ROM) sont faciles à automatiser. La figure V. 3 montre l'organisation d'une tranche de contrôle à base d'un PLA.

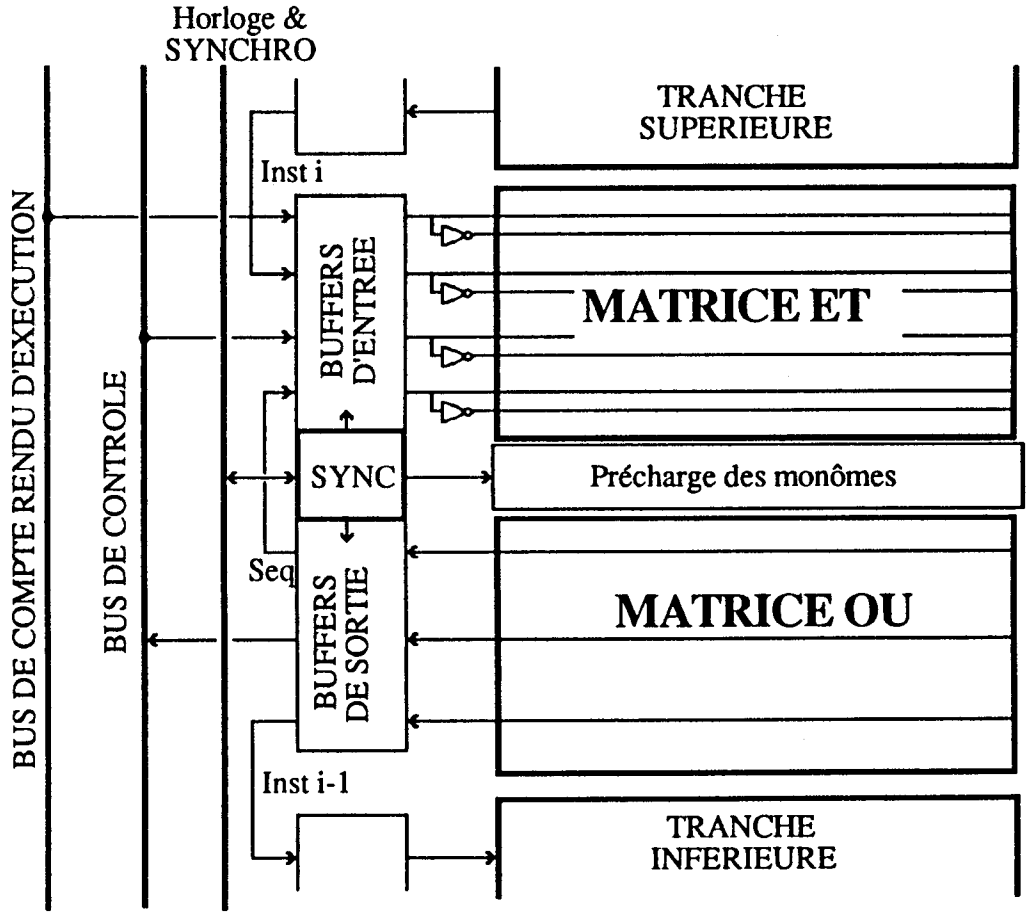



**Figure V. 3 : Organisation d'une tranche de contrôle à base d'un PLA.**

Cette figure montre aussi l'organisation topologique de la tranche de contrôle étudiée pour faciliter son assemblage avec ses voisines.

Nous allons discuter, dans ce type de tranche de contrôle, l'introduction des contrôleurs, des générateurs de vecteurs de test, analyseurs de signature

et des différents circuits annexes.

La contrainte b) (cf § II. 3) est respectée pour un contrôleur de code de Berger du type présenté en Figure V. 5, en effet le pas d'entrée du contrôleur renferme deux transistors MOS de taille minimale, ce qui est le cas pour la matrice ET d'un PLA. Ce qui reste à vérifier est que les entrées de codage du contrôleur ne sont pas à connecter aux entrées du PLA, mais aux bits de codage des sorties du PLA précédent (voir Figure V. 6).

Ce qui nous amène à envisager la structure de la Figure V.5 plutôt que celle de la figure V. 4.

En effet, la figure V.5 représente une alternative d'implantation du contrôleur de code de Berger pour 7 bits d'information et 3 bits de codage, pour laquelle on peut avoir superposition de plusieurs contrôleurs agissant de façon à obtenir une continuité de bits d'information, a l'inverse du contrôleur de la figure V. 4 où il y aurait alternance de 7 bits d'information et 3 bits de codage.

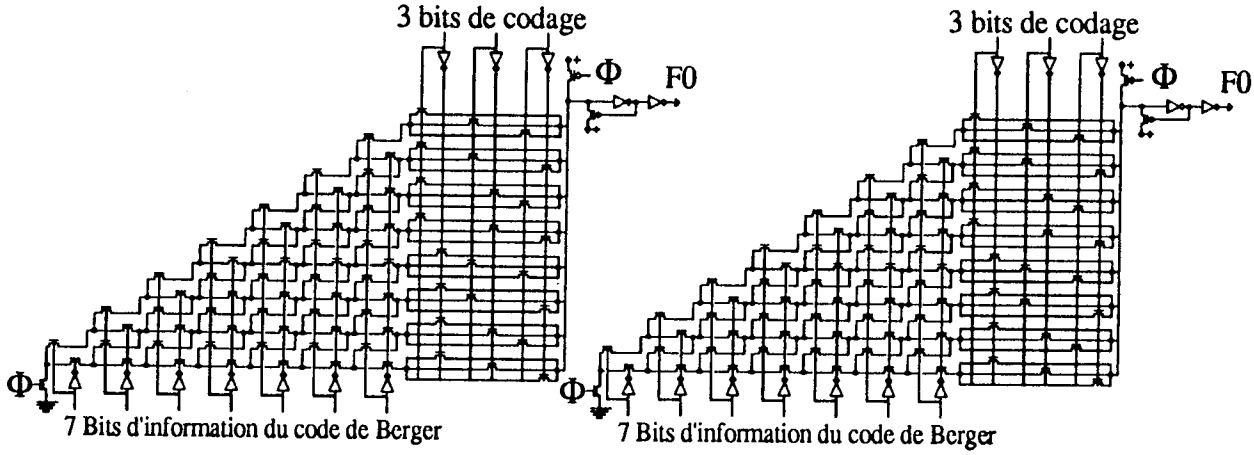



**Figure V. 4 : Contrôleur de Berger de la figure II. 11a).**

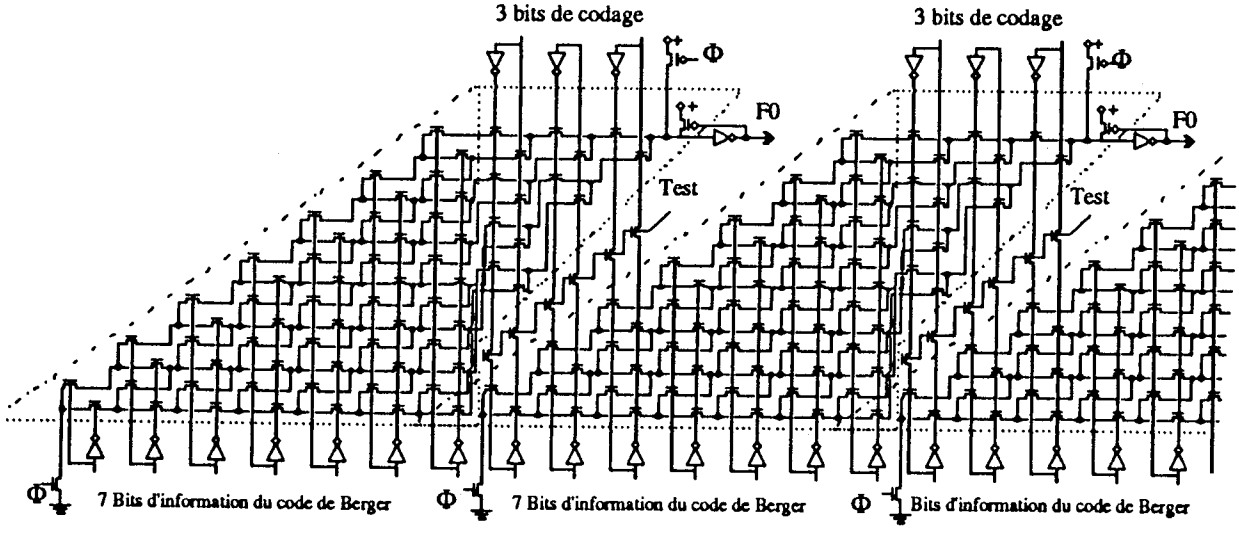
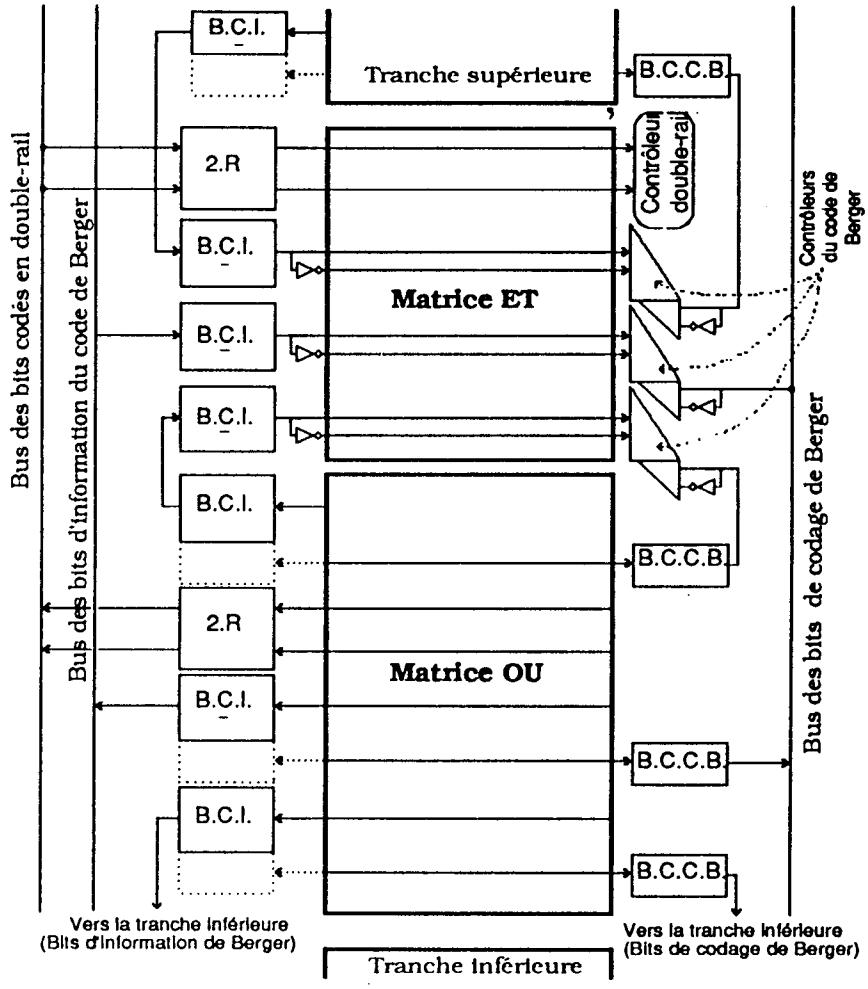



**Figure V. 5 : Une alternative d'implantation des contrôleurs du code de Berger.**

Pouvant ainsi superposer plusieurs contrôleurs il ne reste plus qu'à les assembler contre la matrice ET du PLA sans avoir besoin de routage. Il y a une intime insertion entre les contrôleurs et la matrice ET du PLA.




B.C.C.B. = Registre des bits de codage de Berger
B.C.I.B. = Registre des bits d'information du code de Berger
2.R = Registre des bits codés en double-rail

**Figure V. 6 : Schéma UBIST d'une tranche de contrôle.**

Le PLAi+1 supérieure génère les entrées du type Inst du PLAi courant et leur codage. Ces entrées vont traverser la matrice ET du PLAi courant et vont servir comme bits d'information au contrôleur qui a déjà reçu les bits de codage.

Le contrôleur va non seulement vérifier le codage des sorties Inst du PLAi+1, mais va aussi contrôler les lignes d'entrées et leur complémentaires du PLAi après la traversée de la matrice ET. Ce qui est très avantageux vu que d'après le chapitre II (PLA auto-contrôlable) les lignes d'entrée doivent être contrôlées au moins juste avant les inverseurs.

Les remarques qui suivent détaillent la topologie donnée en figure V.6:

• Si des sorties de PLAs ne peuvent pas être regroupées en ensembles ayant la même destination alors elles sont codées en code double-rail. Par exemple une sortie qui activera une commande du bloc de synchronisation, ne sera pas regroupée avec les commandes de type Inst, elle doit être codée en code double-rail.

Réciproquement, les groupes de sorties ayant un bloc de destination commun seront codés en code de Berger.

• Comme présenté en figure II.2 et IV.1, les entrées primaires des PLAs doivent être contrôlées au moins juste avant les inverseurs. Pour éviter d'avoir recours à deux contrôleurs par PLA, l'un pour les entrées, l'autre pour les sorties, les sorties des PLAs sont contrôlées aux points de leur destination (e.g. après leur traversée de la matrice ET de PLA destination). Le même contrôleur sert donc au contrôle des sorties du PLA source et des entrées du PLA destination.

• En figure V. 6, on remarque aussi que les entrées de PLAs codées en code double-rail alimentent les PLAs sans utiliser des inverseurs d'entrée. Dans ce cas le contrôle des entrées n'est pas nécessaire. Cependant, comme la largeur des tranches de contrôle a été augmentée par l'utilisation de contrôleurs de code de Berger, supprimer ces contrôleur double-rail n'introduira pas de gain de surface. Ces contrôleurs sont donc maintenus.

• Pour la version originale des tranches de contrôle, on associe à chaque entrée une cellule de registre. Par contre pour la version UBIST on a besoin d'avoir une cellule de registre par ligne d'entrée pour les entrées codées en code double-rail.

Pour résoudre ce problème, on propose de disposer les deux cellules de registre sur la largeur d'une seule, comme représenté en figure V. 7 a).

Il est à noter que durant le mode de fonctionnement normal les lignes de bit B et B* sont issues de deux cellules de registre indépendantes. Durant la phase de test hors-ligne (Test=1), quand les registres d'entrée sont configurés en mode LFSR, les deux cellules de registre sont connectées et utilisées comme bascules maître-esclave formant une cellule du LFSR. Durant ce mode les valeurs des lignes d'entrée B et de leur complémentaires B* sont issues de la même cellule du LFSR, ce qui a pour conséquence que B et B* sont toujours complémentaires et ne prennent jamais de valeurs invalides où ils seraient égaux.

• Les cellules des registres de sorties de bits de codage de Berger (les blocs BCCB de la figure V. 6) sont implémentées en bascules maître-esclave comme représenté en figure V. 7 b), de façon à ce que les entrées

d'information et de codage soient appliquées simultanément au contrôleur du code de Berger (durant la phase d'horloge de validation des entrées sur la matrice ET du PLA destination).

Durant la phase de test hors-ligne, ces cellules maître-esclave sont utilisées comme cellules maître-esclave de base du compteur UP/DOWN.

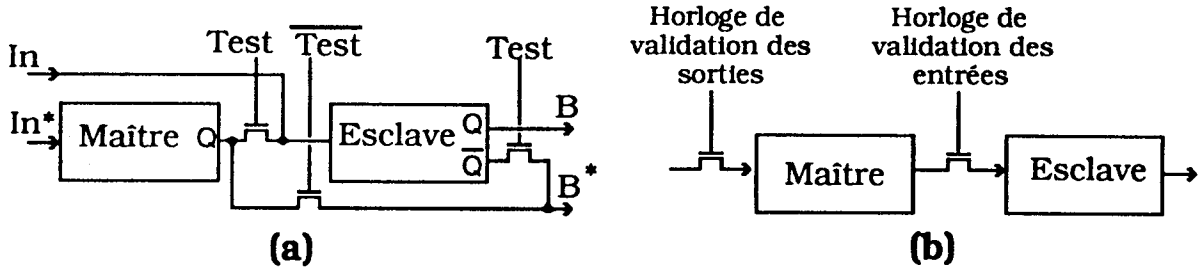



**Figure V. 7 : a) cellules de registre d'entrées codées en code double-rail (2.R), b) cellules de registre de sorties de codage de Berger (B.C.C.B.)**

• Durant le mode de test hors-ligne, les registres d'entrée de chaque PLA sont configurés en LFSR générant les séquences de test de PLAs. On doit donc rajouter une logique (portes XOR, rebouclage, logique de sélection des modes, ...) au cellules de registre.

Pour ce même mode, les registres le sortie sont connectés en analyseurs de signature.

Les cellules maître ne sont pas utilisées en mode de fonctionnement normal (excepté pour les entrées codées en code double-rail).

Il est à noter que les cellules de registre représentées en pointillés sur la figure V. 6 ne sont pas utilisées en mode de fonctionnement normal, elles correspondent aux cellules d'analyseur de signature des sorties de bits de codage de Berger.

• En ce qui concerne les contrôleurs, durant le mode de test hors-ligne, les vecteurs de test générés par les LFSRs sont aussi appliqués aux contrôleurs.

Un schéma particulier est proposé en figure V. 8 pour les contrôleurs du code de Berger.

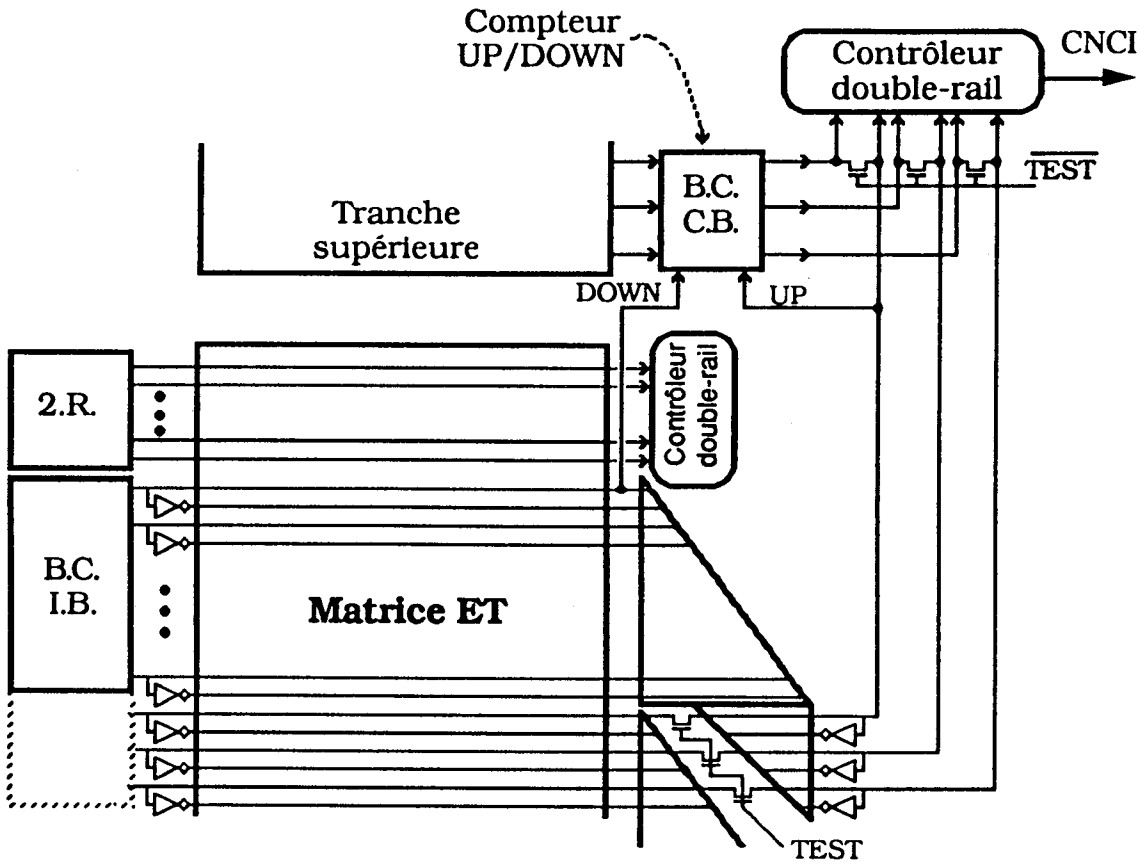



**Figure V. 8 : Génération de vecteurs de test pour les contrôleurs du code de Berger.**

Durant le mode de test hors ligne (Test=1) les entrées de codage du contrôleur sont connectées aux lignes d'entrée du PLA (et déconnectées des sorties correspondantes aux registre BCCB). Ainsi toutes les entrées du contrôleur (bits d'information et bits de codage) reçoivent les vecteurs de test générés par les LFSRs. Pendant ce temps les cellules maître-esclave des registres BCCB sont connectées en compteur UP/DOWN (en utilisant une logique additionnelle).

Les bits de contrôle du compteur et du LFSR sont comparés par un contrôleur double-rail indiquant si le mot appliqué au contrôleur est un mot du code ou hors-code (signal CNCI).

Ce schéma est valable pour tous les contrôleurs du code de Berger de toutes les tranches excepté le dernier contrôleur (figure V. 9). Pour celui-ci les entrées de codage ne correspondent pas à des lignes d'entrées du PLA. On connecte donc ces entrées aux cellules supérieures du LFSR comme indiqué en figure V. 9.

Cette implémentation utilise les mêmes fils de connexion des signaux des/aux registres BCCB aux/des contrôleurs du code de Berger durant les modes de fonctionnement normal et de test hors-ligne.

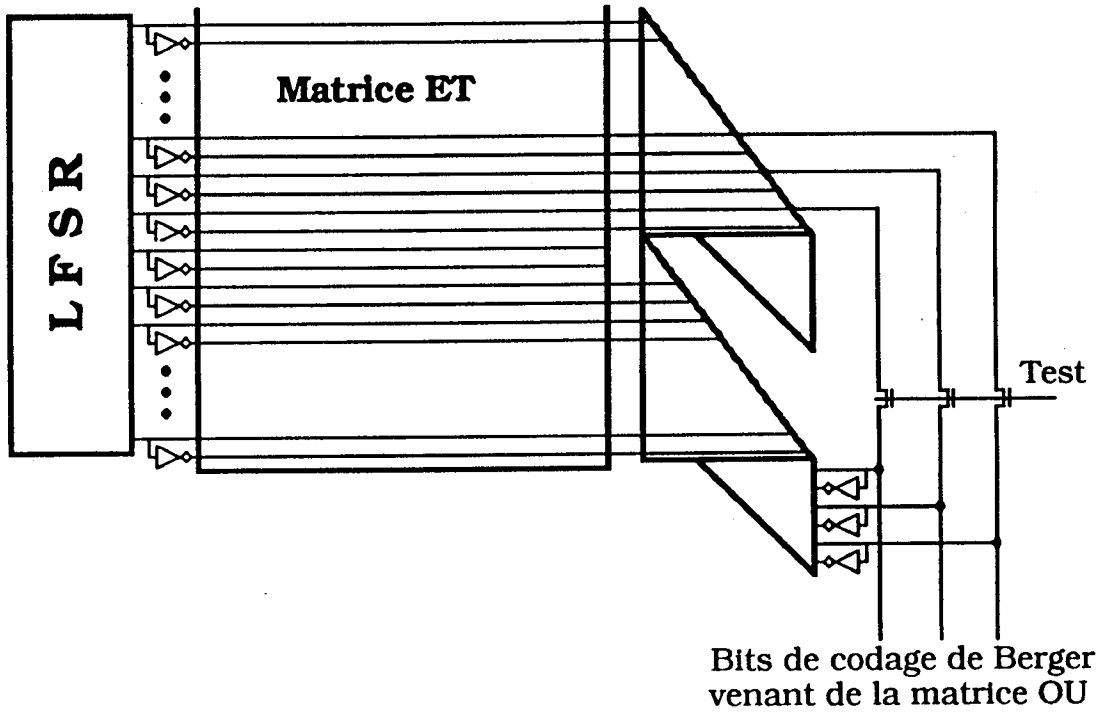



**Figure V. 9 : Entrées de test pour le dernier contrôleur d'une tranche.**

Elle utilise aussi des blocs existant (registres BCCB) pour le compteur UP/DOWN et permet d'insérer une logique additionnelle (contrôleurs double-rail et "switches") dans les lacunes existantes entre les registres BCCB, ce qui introduit une augmentation de surface insignifiante et n'altère pas la largeur des tranches de contrôle.

**V. 3 Couverture de pannes du schéma UBIST de parties contrôle :**

Le test hors-ligne est assuré par des LFSRs placés sur les entrées de PLAs générant les séquences de test et par des LFSRs placés sur les sorties effectuant une analyse de signature parallèle des réponses.

L'utilisation de contrôleurs de sorties de PLAs nous amène deux contraintes :

La première est l'augmentation du temps de test, puisque les sorties de chaque PLA doivent être transférées aux registres d'entrée du PLA de l'étage inférieur pour être vérifiées. La seconde est l'augmentation de surface due au fait que les sorties de PLAs doivent être aussi codées pour des vecteurs ne survenant pas lors du fonctionnement normal (e.g. ajout de monômes supplémentaires).

C'est pour ces raisons que l'on a plutôt choisi de vérifier les réponses de PLAs uniquement par analyse de signature. De plus, les analyseur de signature ont un nombre de cellules égal au nombre de sorties codées des PLAs, ce nombre est généralement grand (e. g. 54, 45, et 51 cellules pour les trois

tranches de contrôle UBIST du 6502 généré par SYCO [REI 86], [JER 88]). Ce qui a pour conséquence que le taux de masquage dû à l'analyse de signature est négligeable, il en résulte donc que le taux de couverture de panne ne dépendra que du type de séquences de test appliqué au PLA.

Il est à noter que les PLAs sont implémentés en logique CMOS dynamique à 4 phases d'horloge (voir annexe 2), une telle logique peut être activée par un ensemble de test couvrant des pannes combinatoires si cet ensemble est appliqué périodiquement au circuit.

Les séquences de test suivantes peuvent être générée par les LFSRs, tenant compte des caractéristiques des PLAs :

• Séquences exhaustives de test : Elle assure 100% de couverture de n'importe quelle panne combinatoire.

• Séquences pseudo-exhaustives de test : Le circuit est virtuellement partitionné en sous-blocs. Chaque sous-bloc inclut une sortie et la partie de circuit la générant. Une séquence pseudo-exhaustive couvrirait toutes les pannes combinatoires affectant chaque sous-bloc.

• Séquences pseudo-aléatoires de test : Elle est adoptée quand les deux précédentes méthodes sont inapplicables. La couverture de pannes est à calculer pour chaque PLA, elle est généralement élevée puisque la longueur des séquences est de l'ordre de $2^{20}$ vecteurs de test.

On peut utiliser un outil de simulation de pannes pour calculer le taux de couverture de pannes pour les séquences de test pseudo-aléatoire. Si le taux de couverture n'est pas satisfaisant, on peut tout aussi bien changer le polynôme diviseur du LFSR, que changer la longueur de test de façon à obtenir un taux élevé et acceptable de couverture de pannes.

Si le taux malgré ces modifications reste non satisfaisant, on utilisera un outil de génération de séquences de test calculant quelques vecteurs supplémentaires à appliquer aux plots d'entrée du circuit et permettant la détection des pannes indétectables par le test pseudo-aléatoire. Ceci peut être obtenu par un simulateur de panne rapide et spécifique aux PLAs et un outil de génération de vecteurs de test comme PLATYPUS [WES 85]. Cette séquence étant courte, elle peut tout à fait être appliquée aux plots d'entrée du circuit. Ceci dit, il est improbable qu'une séquence pseudo-aléatoire de test d'une longueur de $2^{20}$ vecteurs de test ait un taux faible de couverture de pannes.

En ce qui concerne le taux de couverture de panne du test hors-ligne des

contrôleurs du code de Berger, il est de 100% dans le cas où les PLAs sont testés par des séquences exhaustives ou pseudo-exhaustives (les contrôleurs ont 10 entrées et reçoivent tous les mots d'entrée du code et hors-code).

Ce qui est encore vrai quand les PLAs sont testés par une séquence pseudo-aléatoire : La probabilité pour qu'un contrôleur ayant 10 entrées ne reçoive pas toutes les combinaisons possibles $2^{10}$ vecteurs d'entrée quand $2^{20}$ vecteurs d'entrée sont appliqués sur le PLA, est plus petite que $3.10^{-400}$.

Pour les contrôleurs double-rail, la couverture de pannes est similaire à celle d'un contrôleur double-rail conventionnel donné dans [CAR 68].

En particulier pour un contrôleur à k paires d'entrées, il existe $4^{k-2}$ ensembles de 4 vecteurs, chacun assurant un taux de 100% de couverture des pannes simples et une large couverture des pannes multiples (similairement à la couverture des pannes multiples assurée pour un contrôleur double-rail cellulaire donné dans [NAN 88]).

Il est évident que la probabilité que le contrôleur double-rail ne reçoive aucun des ensembles de vecteurs du code d'entrée, est négligeable.

Pendant le fonctionnement normal et pour le test en ligne, les PLAs doivent être "fortement sûrs en présence de défauts" ou "totalement auto-contrôlables", et les contrôleurs doivent être "fortement à code disjoint" ou "auto-testables". D'autre part les PLAs et les contrôleurs doivent être suffisamment activés pour éviter les pannes latentes.

Par construction les PLAs sont FSPD pour toute panne simple, les contrôleurs double-rail sont auto-testables pour toute panne simple et les contrôleurs du code de Berger sont FCD pour toute panne simple et multiple par leur construction auto-activable.

Le test hors-ligne est activé périodiquement avec une période plus petite que le MTBF du circuit, permettant ainsi d'éviter les pannes latentes. Le test en ligne permet alors de détecter au moins toutes les pannes simples.

### V. 4 Contrôle des instructions opératives :

Le PLA du niveau d'interprétation le plus haut PLAn ne reçoit pas le type d'entrée Inst, il n'y a donc pas à les contrôler.

Le niveau d'interprétation le plus bas PLA0 génère des sorties du type Inst destinées à actionner la partie opérative. Elles peuvent être encodées, cela veut dire qu'il existe une interface électrique et logique qui, à partir d'une instruction Inst, génère les commandes des éléments de la POP (Partie OPérative). Ou alors non encodées, c'est à dire que la POP reçoit directement tous les fils de commande de ses éléments, l'interface se réduit alors à des amplis validés par les phases d'horloge.

Dans les deux cas il est possible d'implanter ces sorties de façon latérales (voir Figure V.10) (parallèle aux monômes) en un niveau supérieure de métal (non utilisé). De cette façon ces sorties seront face à la partie opérative. L'assemblage d'interface sera facile si l'on connaît de plus les positions exactes des amplis de commande de la POP.

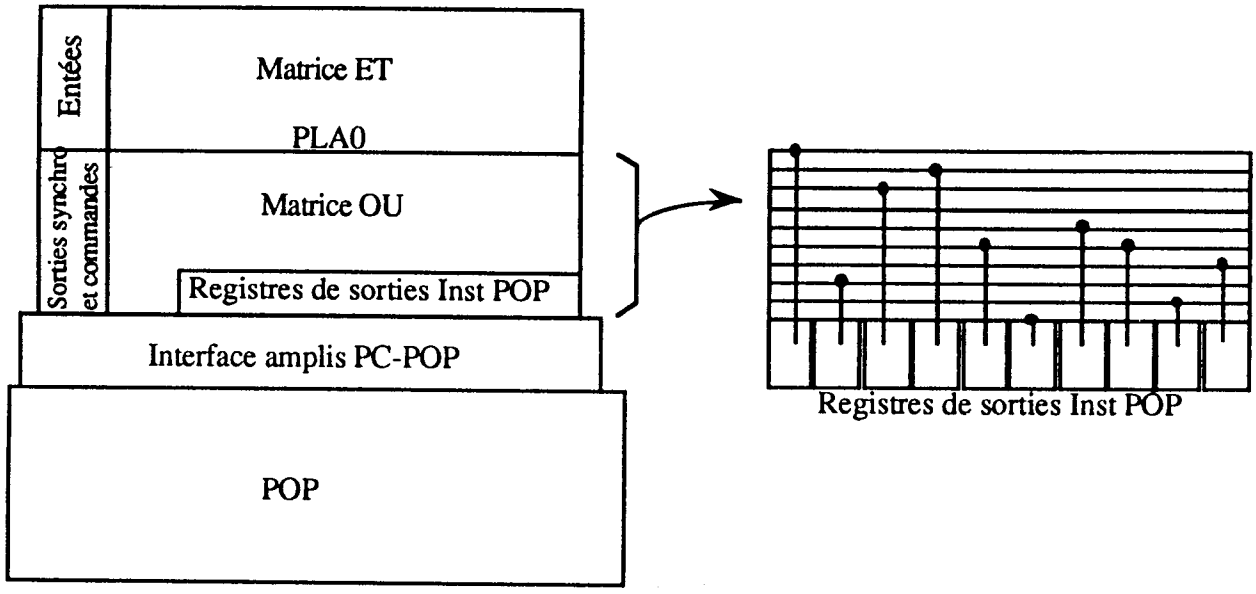



**Figure V. 10 : Sorties latérales parallèles aux monômes.**

De point de vue du test, il serait préférable que les instructions opératives soient non encodées. On verra par la suite une solution pour celles qui sont encodées.

En effet, si les Inst POP (Instructions opératives) ne sont pas encodées elles vont être amplifiées et transmises directement à la POP, le contrôle se fera après la traversée de la POP.

Si les Inst ne sont pas encodées, cela veut dire qu'on dispose d'un nombre important de bits de commande, le codage sera fait en code de Berger, les contrôleurs de code de Berger seront disposés en bas de la POP (Figure V. 11).

La hauteur de la POP va augmenter, et le circuit va être encore allongé.

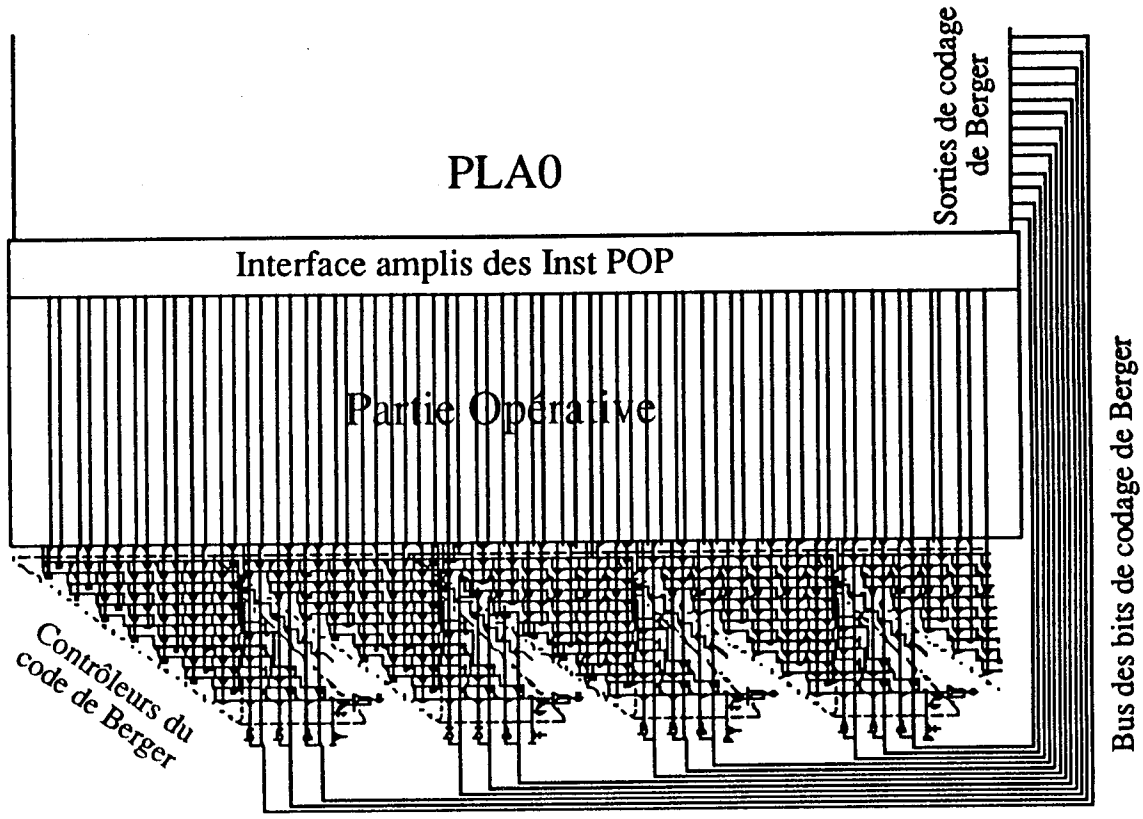



**Figure V. 11: Contrôle par code de Berger des Inst POP.**

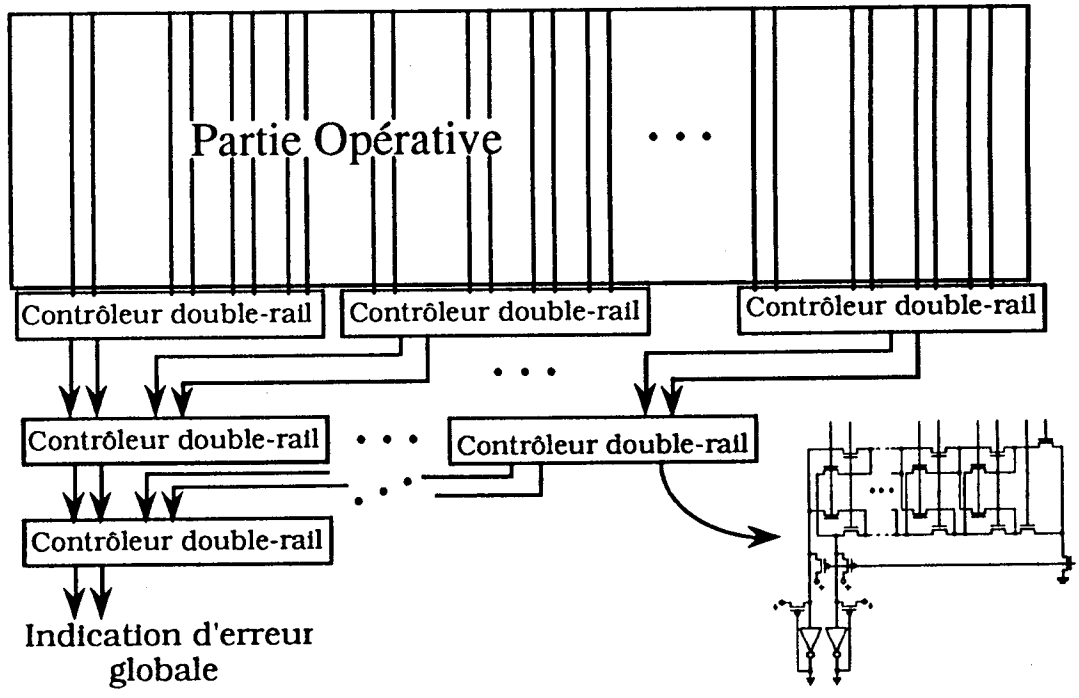



**Figure V. 12 : Contrôle double-rail des Inst POP.**

Une autre solution consisterait à coder en double-rail toutes ces Inst POP, la matrice OU du PLA0 va pratiquement doubler de hauteur et de surface. D'un autre côté ces instructions codées en double-rail vont servir à

actionner les différents éléments de la POP et leur duals. Elles doivent être contrôlées après traversée de la POP par des contrôleurs double-rail, ceux-ci étant moins large que les contrôleurs de Berger, n'augmenteront pas considérablement la hauteur de la POP (Figure V. 12).

Les deux solutions ci-dessus ont des avantages et des inconvénients complémentaires.

**V. 4. 1 Inst POP codées en code de Berger:**

Inconvénients:

• Les contrôleurs de Berger étant relativement gros, on a une augmentation notable de la hauteur de circuit.

• Les Insts POP doivent être de toutes façon codées en double-rail pour activer les éléments de la POP et leur dual, et pour attaquer les contrôleurs de code de Berger. Elles doivent donc être inversées à l'interface PC-POP.

•Les bits de codage de Berger ne pourront pas systématiquement traverser la POP, ils vont donc la contourner, ceci va entraîner une perte de surface due à leur routage. Sur la Figure V. 11 est représenté le bus des bits de codage de Berger.

Avantages:

• Faible augmentation de la matrice OU du PLA0, entraînant un faible ralentissement. C'est un avantage considérable puisque l'étage 0 est généralement le plus gros, il ne doit pas être "gonflé" outre mesure.

Le cycle de calcul des Inst POP sera celui du cycle de base de la POP (transfert de registre à registre).

• On utilisera un seul générateur pour activer tous les contrôleurs, il activera en parallèle tous les contrôleurs. (Noter que le codage se fait pour des groupes de 7 bits d'information et 3 bits de codage. Le générateur aura un LFSR de 10 bits, un compteur/décompteur de 3 bits et un contrôleur double-rail de 3 paires de bits d'entrée).

**V. 4. 2 Inst POP codées en code double-rail:**

Inconvénients:

• Forte augmentation de la matrice OU du PLA0, (elle est pratiquement doublée), entraînant un ralentissement notable du calcul des Inst POP.

Il n'est pas garanti que le cycle de calcul des Inst POP soit inférieur à celui de la POP. Le ralentissement entraînera des temps d'attente de la POP.

• Dans le cas où on aurait un nombre important de bits, le contrôleur

double-rail sera arborescent. Ceci pourrait augmenter notablement la hauteur du circuit.

Avantages:

• Les Inst POP codées en double-rail pourront activer les éléments de la POP et leur duals après une amplification à l'interface PC-POP.

• En outre, elles traverseront systématiquement la POP pour être contrôlées.

• Le contrôleur double-rail n'a pas besoin d'être activé en des phases de test hors-ligne : en effet il existe $4^{n-2}$ ensembles de 4 vecteurs (pour un contrôleur double-rail à n paires d'entrées). Chacun d'eux assurant 100% de couverture des pannes simples et une large couverture des pannes multiples (voir annexe 3). Il est évident que la probabilité pour que le contrôleur ne reçoive aucun de ces ensembles est négligeable.

**V. 6 Test des variables de contrôle et de synchronisation :**

Chaque tranche de contrôle a son propre bloc de synchronisation, lequel communique avec le PLA et les autres blocs.

Ces blocs sont implémentés en logique non régulière. Pour contrôler les entrées et les sorties de ces blocs, on doit concevoir leurs duals, lesquels recevront les entrées complémentaires et généreront les sorties complémentaires. Un contrôleur double-rail est utilisé pour contrôler les signaux d'entrée et de sortie avec leurs complémentaires.

Les blocs de synchronisation communiquent entre eux par des signaux codés en code double-rail, ces signaux n'ont pas besoin d'être contrôlés, seuls les signaux communiquant avec les différentes tranches de contrôle sont contrôlés.

Du fait de la propriété FSPD des blocs de synchronisation, l'assemblage de ces blocs forme un bloc global FSPD.

Tandis que les sorties qui sont les commandes des registres tampons des PLAs, doivent être contrôlées après la traversée de ces registres.

Soit le bloc de synchronisation SBi de la tranche de contrôle i, il admet à ses entrées :

a) Les signaux générés par le PLA courant de la tranche i, ces signaux sont codés en code double-rail. (ResetWait, ResetWait*, SetWork, SetWork*, ResetWaitSup, ResetWaitSup*).

b) Les signaux générés par le PLA de la tranche supérieure i-1, (SetWork, SetWork*).

c) Les signaux générés par le bloc de synchronisation SBi+1 de la tranche inférieure i+1. (ResetWaitInf, ResetWaitInf*, WorkInf, WorkInf*).

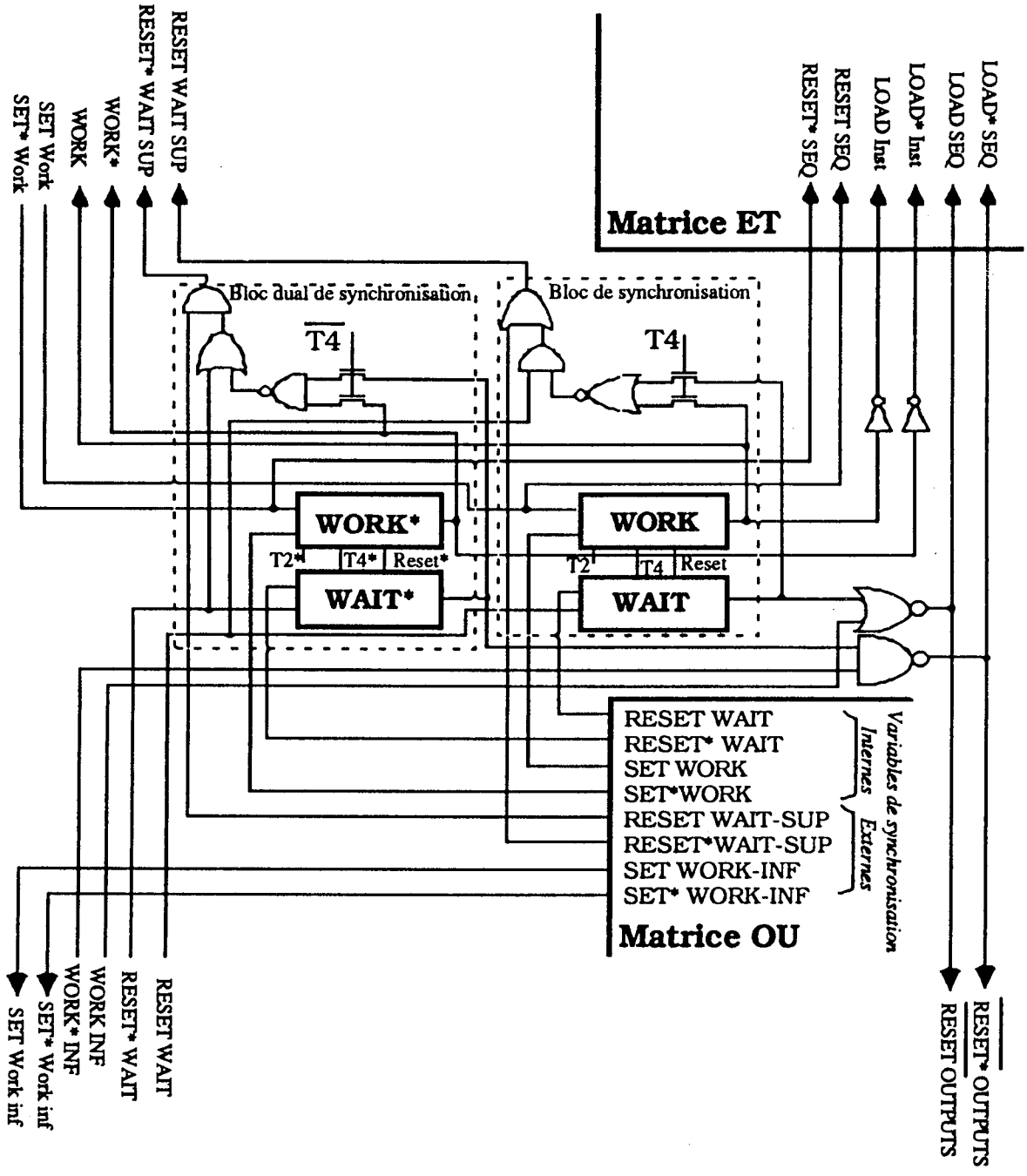



**Figure V. 13 : Bloc de synchronisation FSPD.**

Les signaux donnés en a) et b) générés par les PLAs (PLAi et PLAi+1) doivent être contrôlés. Il le sont de toute façon avec le schéma de contrôle des sorties des PLAs.

Les signaux donnés en c) et qui sont codés en code double-rail n'ont pas besoin d'être contrôlés puisqu'ils activent un bloc de synchronisation FSPD. Une erreur sur ces signaux sera propagée en erreur aux sorties du bloc global.

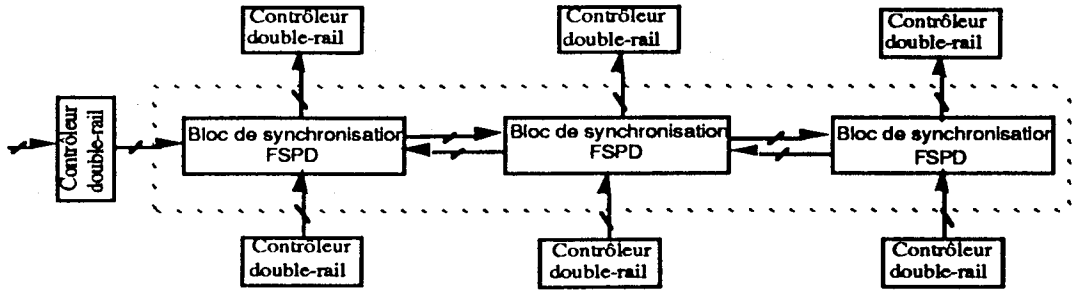



**Figure V. 14 : Bloc global de synchronisation auto-contrôlable.**

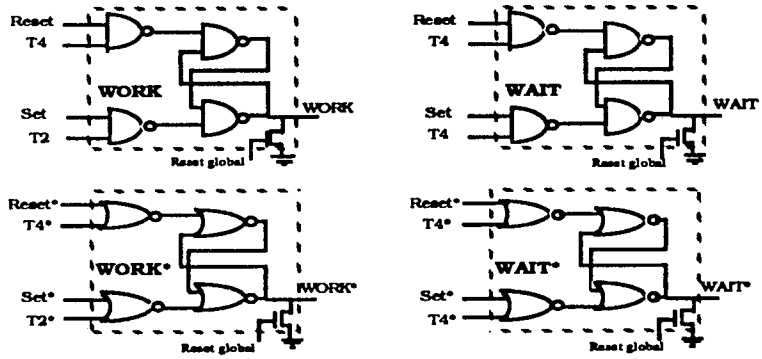



**Figure V. 15 : Variables de synchronisation.**

Les plots d'entrées sont connectés au bus après traversé d'amplificateur. Chaque tranche de contrôle a accès aux signaux véhiculés par le bus et les reçoit sur ses registres d'entrée (figure V. 16). Les signaux de contrôle venant de l'extérieur doivent être codés en code double-rail. Pour ne pas doubler le nombre de plots, ce codage double-rail se fera juste après chaque plot en introduisant un inverseur (voir figure V. 16).

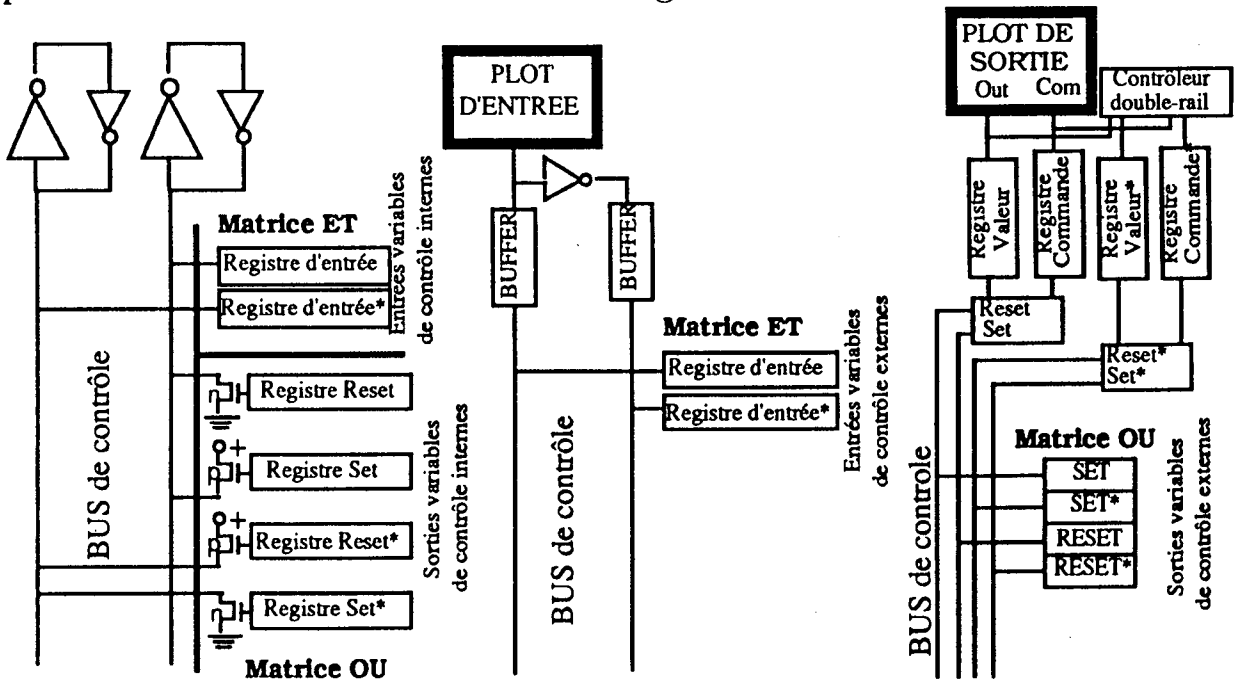



**Figure V. 16 : Variables internes et externes de contrôle.**

Les plots de sorties ne sont pas dupliqués. Chaque tranche de contrôle génère deux signaux (Set, Reset) pour chaque signal de sortie. A partir des signaux Set et Reset, sont générés deux autres signaux qui sont la valeur de la sortie de contrôle et la commande de validation de celle-ci.

Tous les signaux sont codés en double-rail et toute la logique de génération de ces signaux est dupliquée sous forme duale. Un contrôleur double-rail vérifie ces signaux juste avant les plots de sortie.

**Conclusion :**

Dans ce chapitre a été proposé un schéma de génération de parties contrôles hiérarchiques à test intégré UBIST.

On a présenté les structures détaillées des tranches de contrôle et l'on a discuté des outils à utiliser autour du compilateur SYCO, pour générer automatiquement de telles structures.

Les outils de codage de PLAs, de calcul de signature sont opérationnels et permettent la génération de PLAs FSPD associés un schéma BIST.

L'outil de génération automatique de contrôleurs de Berger et double-rail (voir annexes 6 et 7) génère ces structures avec leurs connecteurs remontés permettant l'utilisation de n'importe quel outil de placement/routage pour les introduire dans le circuit.

Le codage des PLAs introduit un rajout de surface représentant 25% de moyenne.

Pour estimer le rajout de surface de toute une partie contrôle, on doit considérer l'augmentation de la hauteur (due aux bits de codage des sorties de PLAs) et l'augmentation en largeur (due aux monômes rajoutés au PLAs, à l'augmentation de lignes de bus, au rajout de contrôleurs et la modification des registres d'entrée et de sortie).

L'augmentation de surface dépend de la partie contrôle considérée. Elle diminue quand on considère des parties contrôle de plus grande taille.

Pour la partie contrôle du 6502 généré par SYCO [REI 86], [JER 88], l'augmentation de surface a été estimée à 53%. Ce pourcentage diminue considérablement lorsqu'on considère le circuit global (incluant les plots).

# CHAPITRE VI

**Parties opératives auto-contrôlables**

**Introduction:**

La stratégie UBIST adoptée pour concevoir et générer automatiquement des parties opératives est élaborée en se basant sur une architecture cible inspirée de l'architecture de la partie opérative du MC 68000, puisque celle-ci est puissante et flexible de plusieurs points de vue, tels que :

- Structure duale des bus permettant des transferts rapides.
- Structure topologique en tranches de bits (bit slice), permettant leur génération automatique.
- Les bus sont segmentés, ce qui permet d'exécuter des opérations en parallèle si nécessaire.

Il existe plusieurs styles de conception de parties opératives, [THO 83], suivant plusieurs critères divers. Deux styles de base peuvent être retenus pour concevoir des parties opératives dépendant de leur architecture :

**a) Le style distribué :** Il est caractérisé par l'utilisation de nombreux éléments (non structurés) de mémorisation éparpillés et d'opérateurs interconnectés de façon irrégulière, et un faible taux d'optimisation d'utilisation des ressources.

Les composants de base sont les registres, les opérateurs monofonctions, les connexions directes, et les multiplexeurs.

Les opérateurs et les chemins de données sont souvent dupliqués au lieu d'être combinés et regroupés autour d'une structure à bus.

Ce style permet d'obtenir des circuits rapides puisqu'aucune limite matérielle n'est imposée. Cependant les circuits générés de cette façon ne sont pas très denses à cause de l'utilisation de nombreux opérateurs et multiplexeurs.

Le type d'implantation d'une telle structure est, soit des cellules de bibliothèques précaractérisées LSI ou VLSI, et la partie opérative est intégrée dans un même circuit, soit des composants TTL, SSI ou MSI, et la parties opérative est constituée d'une carte de composants.

**b) Le style centralisé :** est caractérisé par l'utilisation d'un faible nombre d'unités de traitement multifonctions (unité arithmétique, unité arithmétique et logique, par exemple) et d'éléments d'interconnexion. Les chemins de donnés non parcourus simultanément sont regroupés pour former des bus. Les registres et les opérateurs sont donc reliés entre eux par des bus.

La tendance actuelle est de concevoir des parties opératives de style centralisé à structure en tranches de bit (bit slice), comme représenté en

figure VI.1. La partie opérative est constituée d'un bloc rectangulaire résultant de l'empilement de tranches. Chaque tranche correspond à un bit de la partie opérative.

Pour que cette technique donne de bons résultats, il faut que toutes les cellules d'une tranche aient la même hauteur. Cela permet d'obtenir un dessin des masques dense puisqu'il n'y a pas de trous dans la tranche.

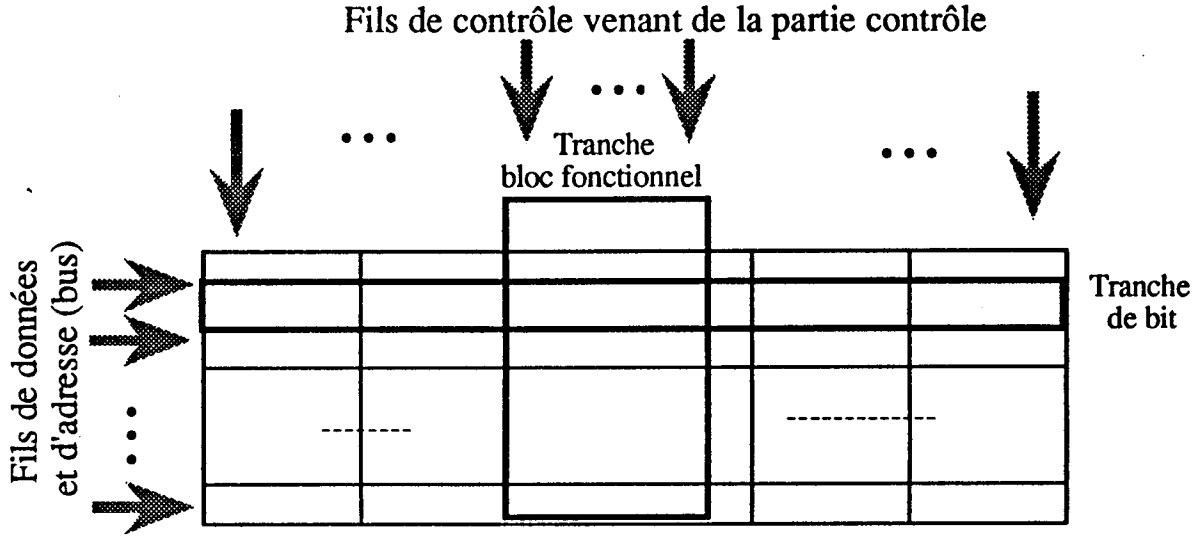



**Figure VI. 1 : Structure en bit slice d'une partie opérative.**

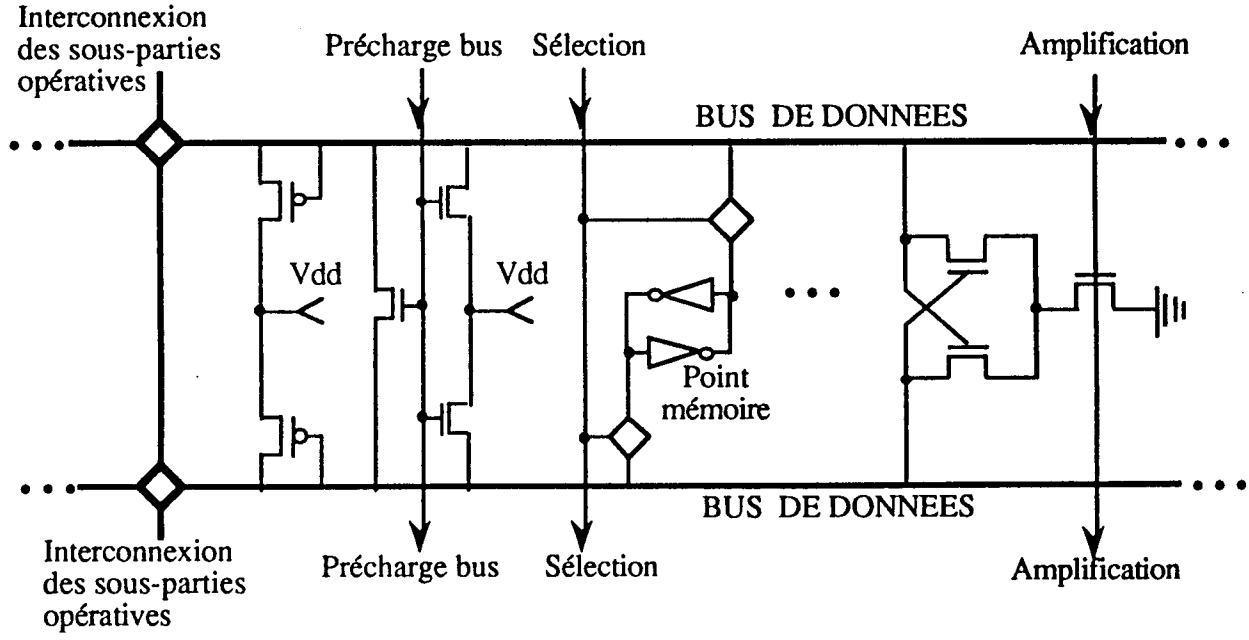



**Figure VI. 2 : Structure duale du bus.**

• Le chemin de données est divisé en plusieurs sous-chemins de données (e.g. un pour le calcul des données, et un pour le calcul des adresses), on peut ainsi exécuter en parallèle plusieurs traitements associés aux

instructions. On utilise de simples interrupteurs "switch" pour partager les variables. La vitesse est ainsi améliorée par l'utilisation de bus globaux pour chaque sous-chemin de donnée, permettant le transfert des données en parallèle.

• On utilise une structure duale des bus. Cette structure nécessite deux lignes par bit, une pour la donnée et l'autre pour la donnée complémentée.

Les deux lignes sont préchargées durant la première phase d'horloge. Pendant la seconde phase, une cellule de registre est connectée aux deux lignes et fait basculer l'une d'entre elles obtenant ainsi la valeur désirée sur le bus. Un amplificateur réduit le temps de lecture. A la troisième phase d'horloge on peut soit faire une écriture sur un autre point mémoire ou registre, soit envoyer la donnée sur un opérateur.

L'utilisation de structure duale de bus (figure VI. 2), en dépit du coût double en ligne de bus, permet d'accroître la vitesse de transfert dans le chemin de données et permet de simplifier les cellules de mémorisation.

• On utilise de simples points mémoire à double accès, plutôt que des bascules maître-esclave à structure duale.

• Une seule ligne de sélection par registre est nécessaire et sert aussi bien aux opérations de lecture que d'écriture.

• Une structure topologique efficace. Les blocs fonctionnels de la parties opérative sont parcourus par une intersection orthogonale de lignes à des niveaux d'implantation différents (métal inférieur et métal supérieur par exemple). Le premier type de ligne est formé du flux de données. Il traverse, parallèlement au bus et aux tranches de bits, les blocs fonctionnels (registres, UAs, UALs, registre à décalage, etc). Alors que le second type de ligne est formé des lignes de contrôle et la propagation des données.

Les blocs fonctionnels d'une tranche de bit ont la même hauteur de façon à avoir une tranche bien remplie, et que l'assemblage des tranches, en les superposant les unes sur les autres, donne un résultat dense sans perte de surface.

Cette structure est représentée en figure VI. 1, où les lignes de bus et alimentation sont parallèles à la largeur d'une tranche de bit, alors que les lignes de contrôle et les lignes de propagations de retenues lui sont perpendiculaires.

Cette disposition permet, en utilisant deux couches de conducteurs isolés, d'occuper une surface minimale. On dessine les cellules de façon à ce que les fils de connexions soient placés au-dessus des transistors. Ainsi on ne perd pas de place pour connecter les cellules entre elles. Il ne doit pas y avoir de problème de routage, puisque d'une part les connexions entre les éléments de la partie opérative sont assurées par deux bus parallèles qui traversent toutes les cellules, et d'autre part les cellules doivent être dessinées de telle façon que toutes les autres connexions entre cellules (par exemple les connexions horizontales entre les cellules d'un même élément fonctionnel et les connexions verticales entre des cellules de tranches voisines) se fassent par aboutement.

C'est la raison principale pour laquelle un tel modèle est choisi: Il n'y a aucune place perdue à cause des connexions. Toutes les connexions entre les éléments se font par l'intermédiaire de bus qui sont situés au-dessus de la logique ; toute les autres connexions se font par aboutement.

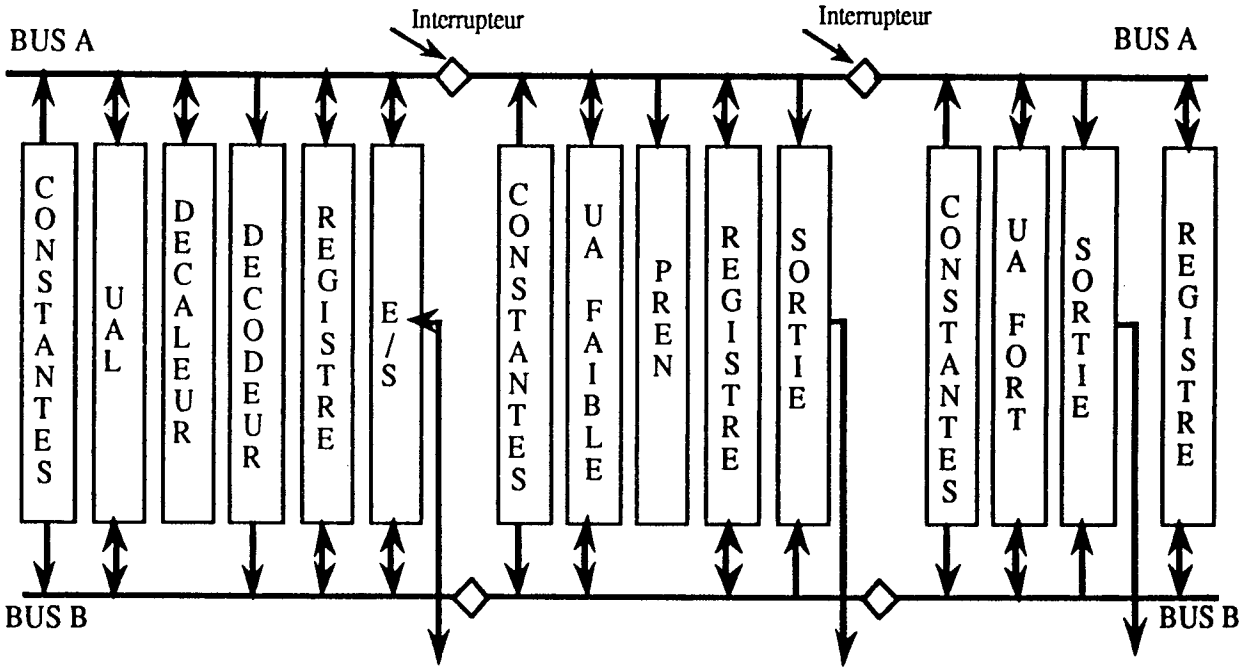



**Figure VI. 3 : Structure de la partie opérative du MC 68000.**

La figure VI.3 montre l'architecture de la partie opérative du microprocesseur MC 68000.

La partie opérative est divisée en trois sous-parties opératives. Cette partition permet d'exécuter trois opérations en parallèle. Tandis que les interrupteurs (switches) permettent de communiquer les données aux trois sous-parties opératives en utilisant le même bus chaque fois que cela est nécessaire.

Chaque sous-partie opérative a une taille de 16 bits. La partie gauche contient le bloc des constantes, l'unité arithmétique et logique de 16 bits, le registre à décalage, un décodeur transformant un mot binaire de 4 bits en un mot de 16 bits encodé en 1-parmi-16.

La partie centrale est composée d'un bloc de constantes d'une unité arithmétique de 16 bits (UA bits de poids faible), un encodeur de priorité (PREN), un registre et un buffer de sortie. La partie droite contient un bloc des constantes, une unité arithmétique de 16 bits (UA bits de poids fort), un registre et un buffer de sortie.

La structure des parties opératives permet un accès facile aux différents blocs par le bus. Ces différents blocs peuvent donc avoir le même générateur de vecteurs de test et le même analyseur de signature. Un simple analyseur de signature est utilisé pour compacter les réponses des différents blocs de la partie opérative.

**VI. 1 Règles assurant la propriété "fortement sûr en présence de défauts :**

Le modèle de pannes considéré pour assurer la propriété FSPD est celui discuté et présenté dans la première partie. On considère toujours les pannes réelles au niveau transistor, à savoir : défaut de contact, transistor MOS stuck-on ou stuck-open, coupure de ligne, courts-circuits entre ligne voisines (deux lignes de métal, ou de polysilicium).

Cet ensemble de pannes représente la classe I d'hypothèses de pannes [COU 81].

**VI. 1. 1 Propriété FSPD pour des erreurs simples :**

Dans le cas où les sorties d'un bloc sont codées en code de parité pour détecter 100% des erreurs simple survenant dans les mots de sorties, il a été établit des règles de conception assurant la propriété FSPD [NIC 85b], [NIC 86], [NIC 87].

La règle R1 (voir paragraphe I.6) assure la propriété Fault Secure pour toutes les pannes de la classe I exceptées les coupures de lignes d'alimentation.

Les pannes affectant les lignes d'alimentation ne sont généralement pas abordées dans la littérature [NIC 84a], [NIC 85b]. Pour tenir compte de coupures de lignes d'alimentation, une seconde règle, la règle R2 (voir paragraphe I.6) doit être vérifiée, évitant que de telles coupures entraînent des erreurs multiples aux sorties d'un bloc.

Les règles R1 et R2, vérifient la propriété Fault Secure. Il existe néanmoins des types de pannes pouvant empêcher le circuit d'être FSPD. Les courts-circuits indétectables, entre deux lignes connectées à deux sorties différentes, peuvent empêcher le circuit d'être FSPD. Toutes les autres pannes ne peuvent jamais empêcher le circuit d'être FSPD.

**VI. 1. 2 Propriété FSPD pour des erreurs unidirectionnelles :**

Dans le cas où les sorties sont codées en code non ordonné, les règles de conception assurant la propriété FSPD sont données dans [NIC 86]. La règle R4 (voir paragraphe I.6) assure la propriété sûr en présence de défauts pour toutes les pannes de la classe I, excepté les coupures de ligne d'alimentation.

Si l'on doit tenir compte des coupures de lignes d'alimentation, une seconde règle est nécessaire, il s'agit de la règle R6 (voir paragraphe I.6).

La propriété Fault Secure ayant été vérifiée par les règles R4 et R6, il pourrait exister des pannes pouvant empêcher le circuit d'être FSPD.

Les courts-circuits indétectables, entre deux lignes connectées aux sorties du circuit, à travers des chemins ayant différentes parité d'inversion, peuvent empêcher le circuit d'être FSPD. Toute les autres pannes ne peuvent jamais empêcher le circuit d'être FSPD.

Les règles R2 et R6 sont difficiles à respecter dans certains cas, on pourrait utiliser à ce moment là les règles R2' et R6' :

*Règles R2' et R6' : Si les règles R2 et R6 ne sont pas vérifiées pour certaines lignes d'alimentation, alors ces lignes doivent être utilisées à leur extrémités pour alimenter des portes logiques pour lesquelles les sorties sont codées en code double-rail.*

Si une coupure survient sur une ligne d'alimentation, alors cette dernière règle permet de propager l'erreur aux sorties de portes codées en double-rail en un collage de ces sorties à la même valeur logique, la panne de coupure est alors immédiatement détectée.

**VI. 2 Stratégie d'implantation de parties opératives auto-contrôlables :**

L'étude qui va suivre porte sur l'implantation de parties opératives auto-contrôlables. On va considérer les blocs les plus utilisés dans une partie opérative: bus avec sa logique annexe (précharge, amplification, ...), les registres, les unités arithmétiques, les unités arithmétiques et logiques, registres à décalage, ... Ces blocs peuvent être divisés en deux groupes :

a) Les blocs qui n'ont pas d'interactions entre tranches de bits (par exemple les bus, la logique de précharge, les amplificateurs, et les registres).

b) Les blocs pour lesquels il y a une propagation de signaux d'une tranche de bit à une autre (par exemple les unités arithmétiques, les unités arithmétiques et logiques, pour lesquels on a une propagation de retenue à travers les tranches de bit, et les registres à décalage).

**VI. 2. 1 Registres, bus et leurs circuits annexes :**

Concernant les blocs de type a), il est évident de vérifier qu'exceptées les lignes de contrôle, toutes les autres lignes pour chaque bloc respectent la règle R1. La règle R2 est vérifiée par les lignes d'alimentation puisque chacune d'elle alimente une seule tranche de bit (une seule sortie pourrait être affectée si une coupure survient sur une ligne d'alimentation).

Tenant compte de ces considérations, un codage par parité peut être utilisé pour contrôler des blocs de type a), pourvu que les lignes de contrôle soient contrôlées après leur traversée de la partie opérative (voir figure VI.4.a)).

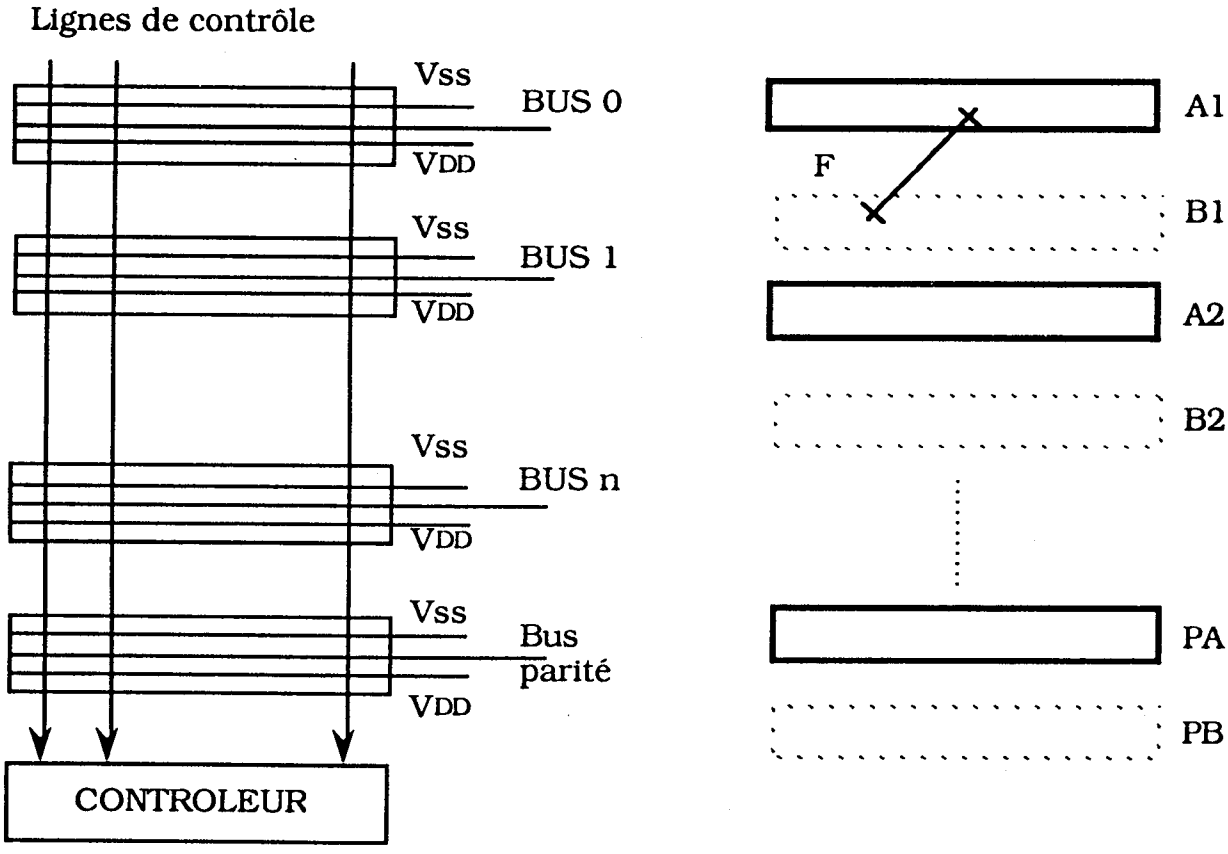



**Figure VI. 4 : a) Contrôle des lignes de contrôle d'une partie opérative, b) Schéma évitant les courts-circuits indésirables.**

Les signaux des lignes de contrôle sont prévus pour être générées par une partie contrôle auto-contrôlable, donc codées en un code détecteur d'erreur (code de parité par exemple).

Ce schéma assure la propriété Fault Secure pour des pannes simples. Pour assurer la propriété FSPD, il est nécessaire d'éviter des courts-circuits indétectables entre deux tranches de bits, comme il a été signalé précédemment.

Toutes les valeurs binaires possibles étant transférées sur le bus et les registres, la détectabilité de ce type de court-circuit est garanti par le contrôle des bus.

Néanmoins, si l'on veut assurer cette condition indépendamment des valeurs transférées dans la partie opérative, deux bits de parité peuvent être utilisés, tel que des tranches adjacentes soient contrôlées par des bits de parité différents (voir figure VI. 4 b)).

**VI. 2. 2 Unités arithmétiques et unités arithmétiques et logique :**

Ces blocs appartiennent au groupe b). La propagation de la retenue d'une tranche de bit à une autre ne permet pas la vérification de la règle R1 pour les UAs et UALs. L'utilisation de code de parité ne permettrait pas de les contrôler.

Une solution possible serait la duplication (i.e. implémentation de deux blocs identiques et comparaison), mais cette solution a l'inconvénient de ne pas permettre la détection d'erreurs de conception, puisque celles-ci affectent les deux blocs dupliqués.

On pourrait les détecter par un test après fabrication (Debug de circuits par exemple). Cependant, des pannes intermittentes dues à une mauvaise qualité de conception peuvent échapper à un test d'après fabrication, et peuvent se manifester plus tard durant la vie du circuit et engendrer, au niveau système, des erreurs indétectables.

Il est beaucoup plus intéressant dans ce cas là d'implémenter des blocs duals. Cette implémentation permettra de couvrir une grande partie des erreurs de conception au niveau électrique, ainsi qu'une grande partie des pannes de mode commun survenant pendant la durée de vie des circuits.

**VI. 2. 2. 1 Unité Arithmétique FSPD:**

En figure VI. 5 a) est montrée l'unité arithmétique utilisée dans le microprocesseur MC 68000, et sa duale en figure VI. 5 b). On a obtenu le circuit dual en échangeant réciproquement la logique OR en logique AND.

Un cas particulier est la logique nécessaire pour la propagation de retenue, on l'obtient en utilisant le même signal pour actionner T1 et T1' et les signaux complémentaires pour actionner T2 et T2'. Un transistor supplémentaire T3' a été rajouté pour éviter la propagation de signaux des tranches supérieurs aux tranches inférieures.

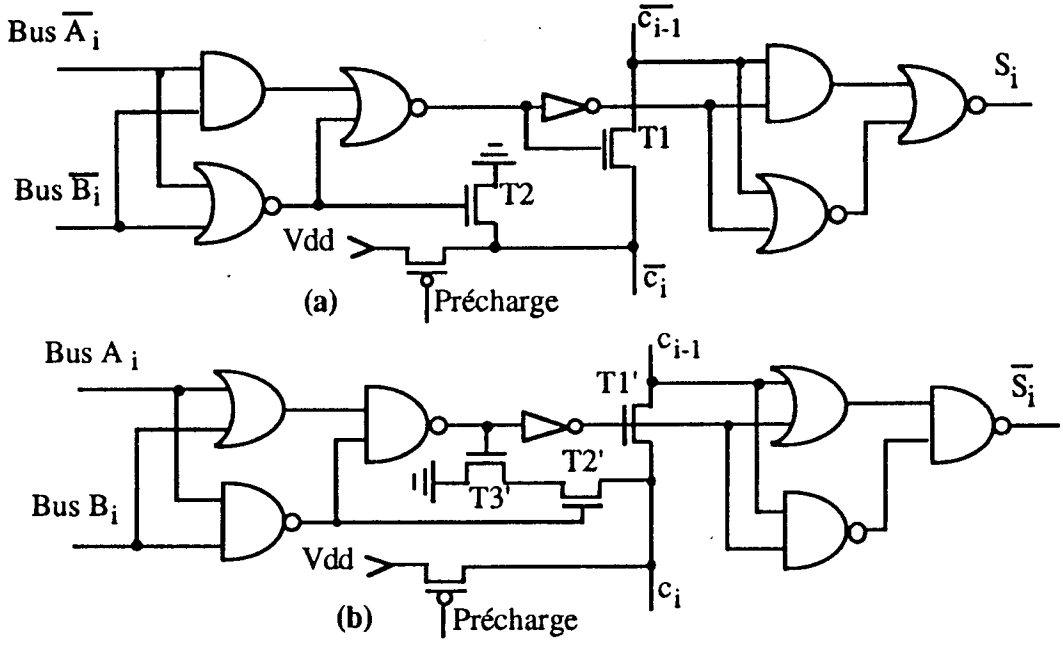



**Figure VI. 5 : Unité arithmétique, a) originale, b) duale.**

La figure VI. 6 donne le schéma de l'unité arithmétique et logique originale et l'unité arithmétique et logique duale.

Les quatre commandes $\overline{COR}$, $\overline{CAND}$, $\overline{CXOR}$, SL servent à commander les opérations arithmétiques et logiques et le décalage à gauche.

Il est à remarquer que le circuit de génération de $\overline{c}_i$ est un circuit MOS série, le fonctionnement correct de ce circuit implique que l'état d'un étage ne peut pas influencer l'état de l'étage précédent. Ceci est assuré par les valeurs possibles des commandes $\overline{COR}$, $\overline{CAND}$, $\overline{CXOR}$, SL qui garantissent que l'on n'a jamais $p_i=1$, $G_i=1$ et $\overline{c}_{i-1}=1$.

Que ce soit pour le schéma de l'UA de la figure VI.5, ou l'UAL de la figure VI.6, la regénération du signal sur les lignes $C_i$ et $\overline{c}_i$, se fait en intercalant des amplificateurs toutes les n tranches de bits, [MEAD 80], [WES 85].

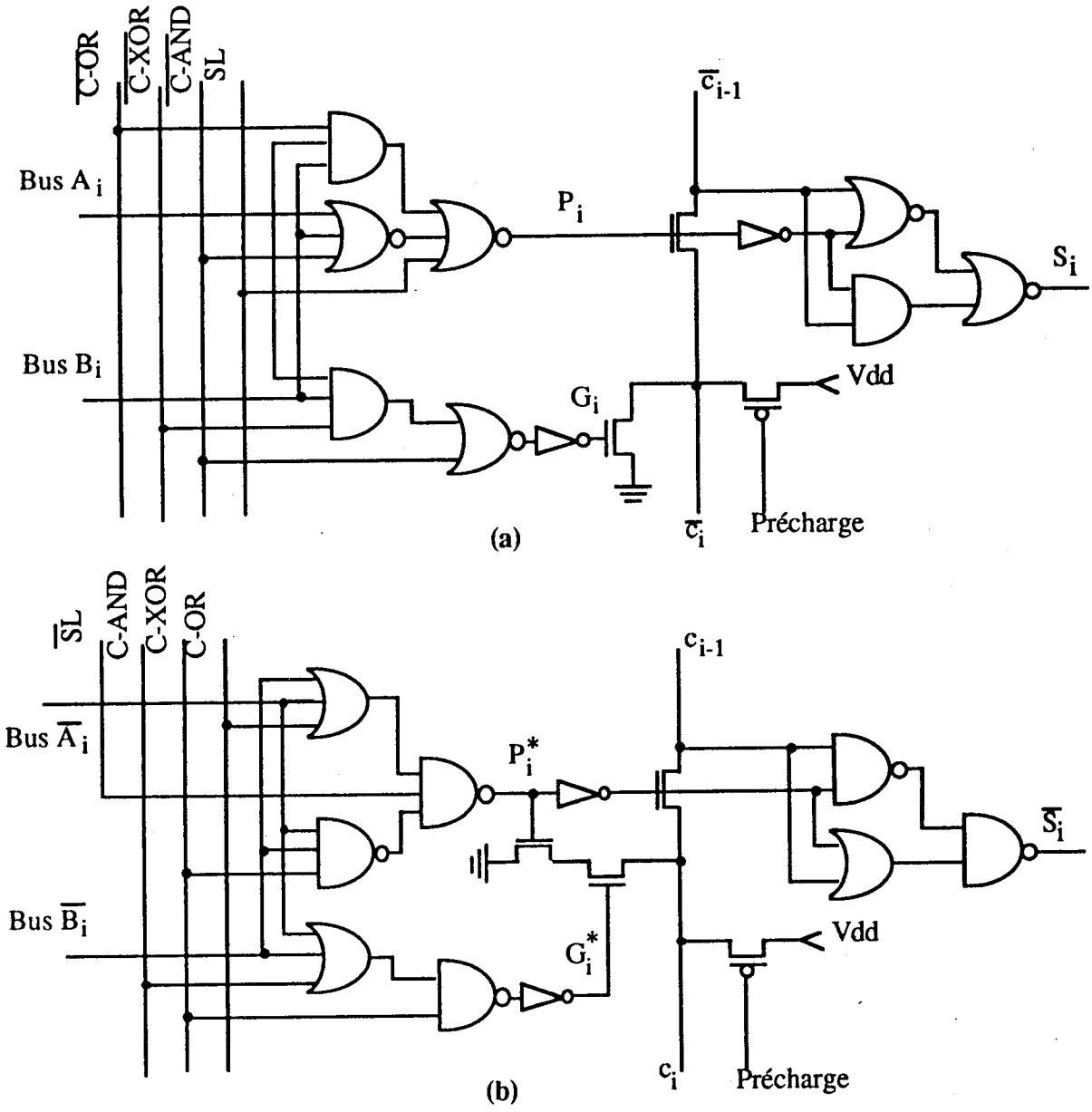



**Figure VI. 6 : Unité arithmétique et logique, a) originale, b) duale.**

Les solutions de duplications duales ont été proposées dans [NIC 84] et [NIC 85a]. Une autre solution permettrait de réduire l'augmentation de surface, ce serait la conception d'une unité arithmétique respectant les règle R5 et R6'. L'unité arithmétique FSPD est donnée en figure VI. 8.

Elle admet des entrées codées en double-rail, qui sont délivrées de toutes façons par la structure duale de bus, et génère ses sorties codées en code double-rail (détectant toutes les erreurs unidirectionnelles). Dans la conception de cette UA, on utilise une propagation de retenue duale proposée en figure VI. 5 et figure VI. 6.

On utilise des structures différentielles de portes XOR (figure VI. 7).

Tenant compte des points suivants :

• La parité d'inversion de n'importe quelle ligne entre les entrées et les sorties de la XOR différentielle est "1" (un seul couple inversant (grille, source) d'un MOS pour chaque chemin).

• Le couple (grille, source) des transistors T1 et T1' est toujours inversant (ceci est dû à la précharge de la retenue pour chaque tranche de bit). Ces transistors transfèrent toujours le niveau "0".

On vérifie que l'unité arithmétique basée sur la structure de la figure VI.8 vérifie la règle R5.

La règle R6' est aussi vérifiée par le fait que chaque ligne d'alimentation alimente les portes différentielles XOR délivrant une sortie codée en double-rail ($S_i$, $\overline{S}_i$).

Finalement on vérifie simplement que n'importe quelle panne simple est détectable par au moins un des 8 vecteurs binaires correspondants aux signaux $A_i$, $B_i$, $C_{i-1}$.

L'unité arithmétique est alors TAC.

L'unité arithmétique peut être implémentée en technologie NMOS en utilisant les portes XOR de la figure VI. 7 a). En technologie CMOS statique (porte XOR à la figure VI. 7 c)). Ou en logique NORA en utilisant la porte XOR de la figure VI. 7 d), pour générer les signaux Pi et $\overline{P}i$ et utiliser la porte XOR de la figure VI. 7 a) pour générer les sorties Si et $\overline{S}i$.

Le nombre de transistors pour une implantation d'une UA originale CMOS statique de la figure VI.5 a) est 25, alors que pour la figure VI. 8 il est de 42. L'augmentation en nombre de transistors est de 68%.

Pour une implantation en logique NORA l'augmentation est de 56% et pour une logique domino elle est de 54%.

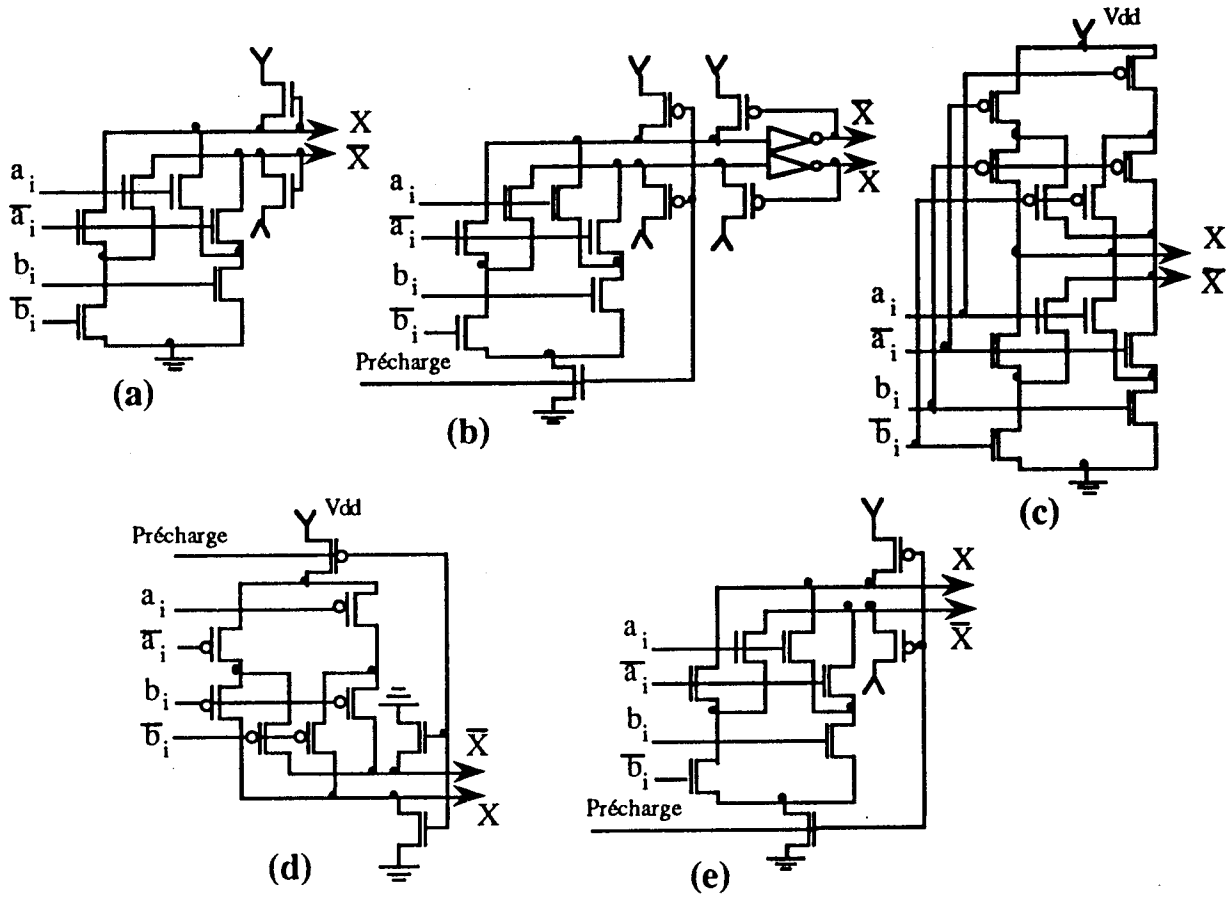



**Figure VI. 7 : Portes XOR différentielles, a) NMOS, b) Domino, c) CMOS statique, d) CMOS dynamique (réseau p), e) CMOS dynamique (réseau n)**

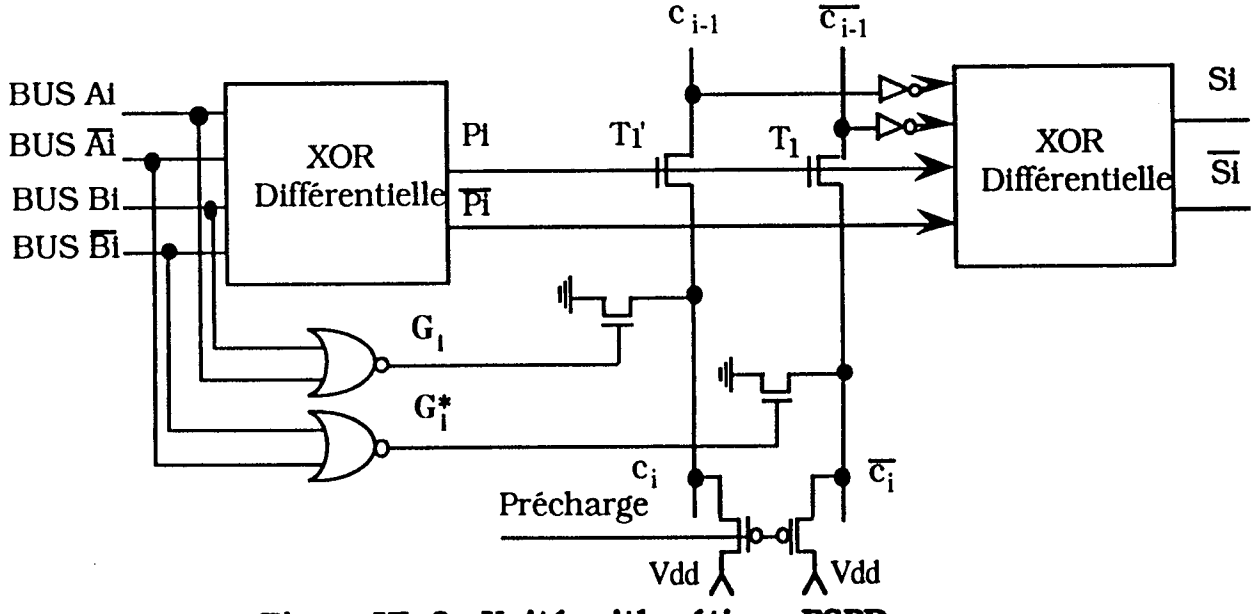



**Figure VI. 8 : Unité arithmétique FSPD.**

### VI. 2. 2. 2 Unité arithmétique et logique FSPD :

On va présenter dans la suite une unité arithmétique et logique FSPD

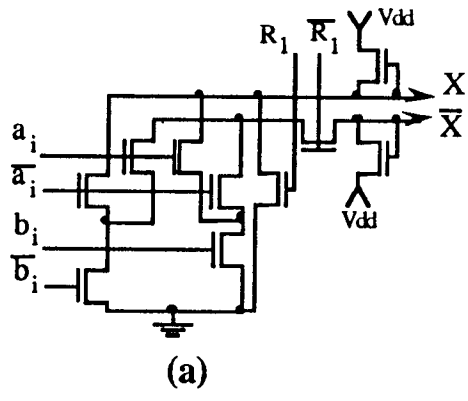
destinée à être couplée à un générateur de vecteurs de test, assurant une implémentation UBIST de l'UAL, (voir chapitre VII).

Il est important dans ce cas de minimiser le nombre de lignes de contrôle de l'UAL, pour obtenir des générateurs de vecteurs de test peu complexes, et des séquences de test pas trop longues.

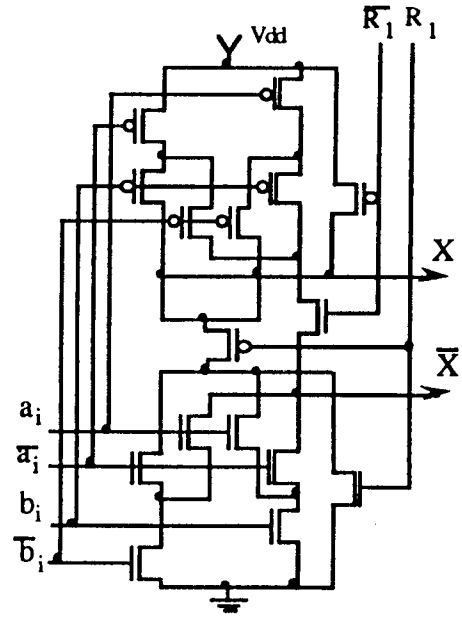
On propose une UAL FSPD qui peut effectuer les 10 opérations logiques non-triviales sur deux bits, l'addition, la soustraction, le OU exclusif, et le décalage à gauche (celui-ci est exécuté par l'UAL dans certains systèmes).

Quatre signaux de contrôle sont nécessaires pour les 14 opérations citées plus haut, ils représentent le nombre minimum de signaux de contrôle nécessaires pour coder les 14 opérations.

Dans un premier temps on donne une UAL FSPD qui effectue l'addition, le décalage à gauche et la fonction OU exclusif. Pour effectuer ces opérations on modifie le XOR différentiel de la figure VI. 7, pour mettre les sorties Pi et $\overline{Pi}$ dans l'état 0, 1 lorsque R1 est activé. La porte XOR différentielle modifiée est donnée en figure VI. 9, et elle est utilisée en figure VI. 10, obtenant une unité arithmétique et logique FSPD.




(a)




(b)

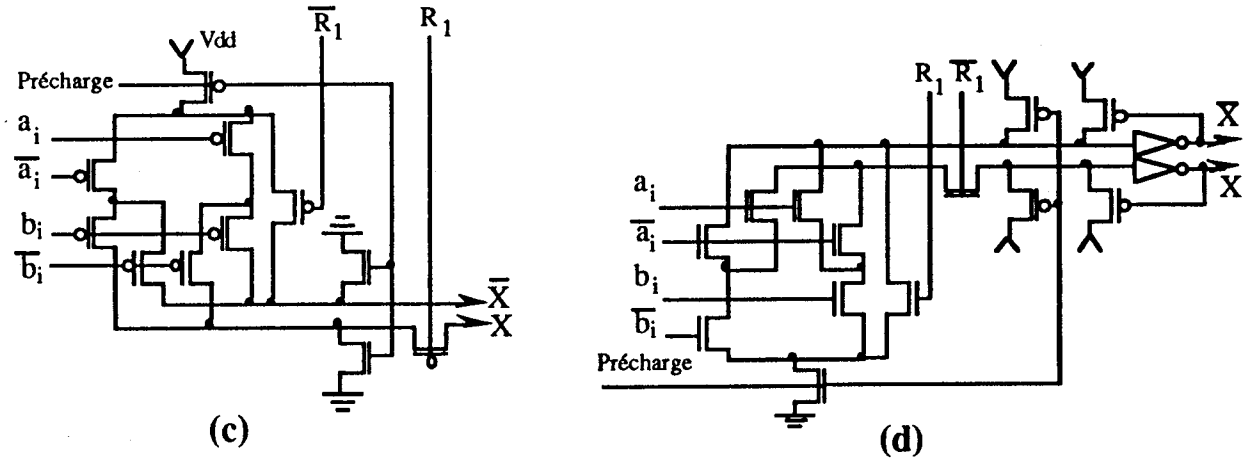



**Figure VI. 9 : Portes XOR différentielles modifiées, a) NMOS, b) CMOS statique, c) CMOS dynamique (réseau p), d) Domino.**

Quand (R1,R2)=(0, 0), l'UAL fonctionne en UA et effectue l'addition. Quand (R1, R2)=(1, 0), les signaux de propagation sont désactivés ($P_i$=0, $\overline{P}_i$=1) et bloquent la propagation de la retenue. En même temps les signaux de génération retiennent leur valeurs sur les bus A, $\overline{A}$ (i.e. $G_i=a_i$, $G_i^*=a_i$). La retenue devient $c_i=a_i$, $\overline{c}_i=\overline{a}_i$ et la sortie $S_i$ devient $S_{i+1}=ci\oplus Pi=ai\oplus 0=ai$ ($\overline{S}_{i+1}=\overline{c}i\oplus Pi=\overline{a}i\oplus 0=\overline{a}i$). Il en résulte que (R1, R2)=(1, 0) est une opération de décalage à gauche.

Finalement lorsque (R1, R2)= (0, 1), toutes les retenues sont forcées à "0" (i.e. ci=0, $\overline{c}i$=1) et les sorties prennent les valeurs $S_i=P_i\oplus c_{i-1}=P_i=a_i\oplus b_i$ ($\overline{S}_i=P_i\oplus\overline{c}_{i-1}=a_i\oplus b_i$). Il en résulte que (R1, R2)=(0, 1) correspond à l'opération de OU exclusif (XOR).

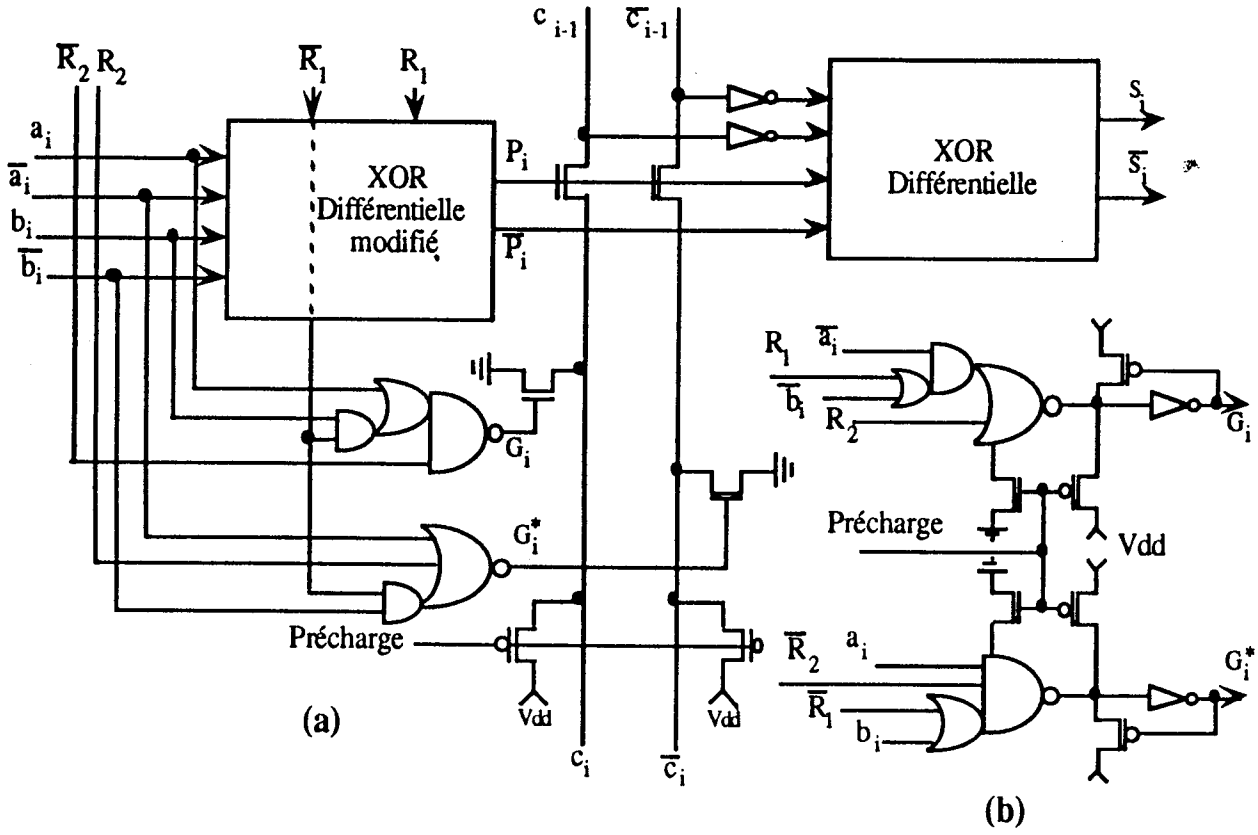



**Figure VI. 10 : a) UAL FSPD effectuant les opérations addition, décalage à gauche et XOR, b) portes nécessaires pour une implantation Domino.**

La propriété FSPD pour cette UAL est vérifiée comme pour l'UA de la figure VI. 8, puisqu'on peut facilement vérifier que la parité d'inversion entre n'importe quelle entrée ai, $\overline{ai}$, bi, $\overline{bi}$, R1, $\overline{R1}$, R2, $\overline{R2}$ et n'importe quelle sortie Sj, $\overline{Sj}$ de l'UAL est égale à "0".

Cette UAL peut être implémentée:

- En NMOS en utilisant la XOR différentielle de la figure VI. 7 a) et la XOR différentielle modifiée de la figure VI. 9 a).
- En CMOS statique en utilisant les blocs de la figure VI.7c) et figure VII.9b).
- En implémentation NORA on utilisera la XOR différentielle de la figure VI.7e), les portes générant Gi et Gi* sont implémentées en logique dynamique en figure VI.10 b), et on utilisera la XOR différentielle modifiée de la figure VI. 9 c).
- Finalement en implémentation Domino, on utilisera la XOR différentielle de la figure VI. 7 b), les portes générant Gi et Gi* sont implémentées en logique dynamique en figure VI.10 b), et la XOR différentielle modifiée est celle de la figure VI. 9 d).

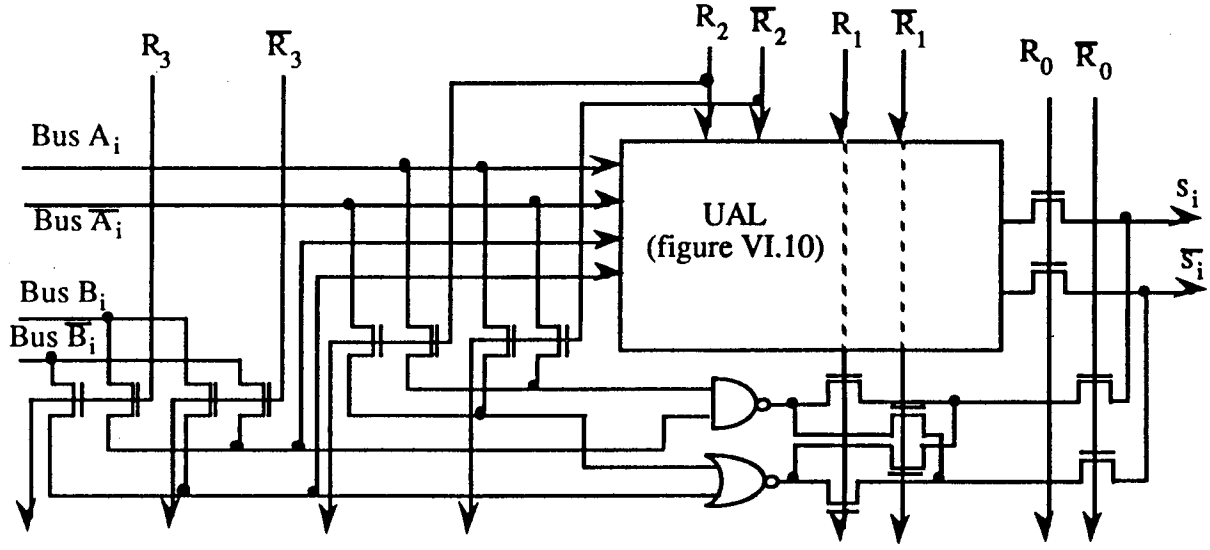



**Figure VI.10bis : UAL FSPD à encodage optimal des signaux de contrôle.**

La tranche d'UAL de la figure VI. 10 peut être utilisée en figure VI.10 bis pour concevoir l'UAL FSPD avec un encodage minimal des signaux de contrôle. Cette UAL effectue 14 opérations décrites en table VI.1 (i.e. addition, soustraction, décalage à gauche et les 10 opérations logiques non-triviales de deux variables), 4 signaux de contrôle (R0, R1, R2, R3) sont nécessaires. On a un encodage optimal des signaux de contrôle.

Par comparaison, l'UAL de la figure VI. 6 utilise 4 signaux de contrôle pour effectuer l'addition, le décalage à gauche, les opérations AND, OR et XOR. Un signal de décalage est utilisé dans le MC 68000 pour effectuer la soustraction, et si l'on veut implémenter les autres opérations logiques, il faut prévoir d'autres signaux de contrôle.

Ce codage optimal nous permet de minimiser le temps de génération des vecteurs de test et de diminuer la complexité et taille du générateur de vecteur de test.

On va calculer l'augmentation du nombre de transistors de l'UAL FSPD de la figure VI. 10bis par rapport à une UAL sans propriétés d'autocontrôlabilité (figure VI.11) effectuant le même ensemble d'opérations avec un encodage optimal des signaux de contrôle.

| R2,R3 \ R0,R1 | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 00 | $\bar{A}\wedge\bar{B}$ | $\bar{A}\vee\bar{B}$ | A-B | shift A |
| 01 | $\bar{A}\wedge B$ | $\bar{A}\vee B$ | A+B | shift A |
| 10 | $A\wedge\bar{B}$ | $A\vee\bar{B}$ | $\overline{A\oplus B}$ | 0 |
| 11 | $A\wedge B$ | $A\vee B$ | $A\oplus B$ | 0 |

**Table VI. 1 : Table de vérité pour l'UAL de la figure VI. 11.**

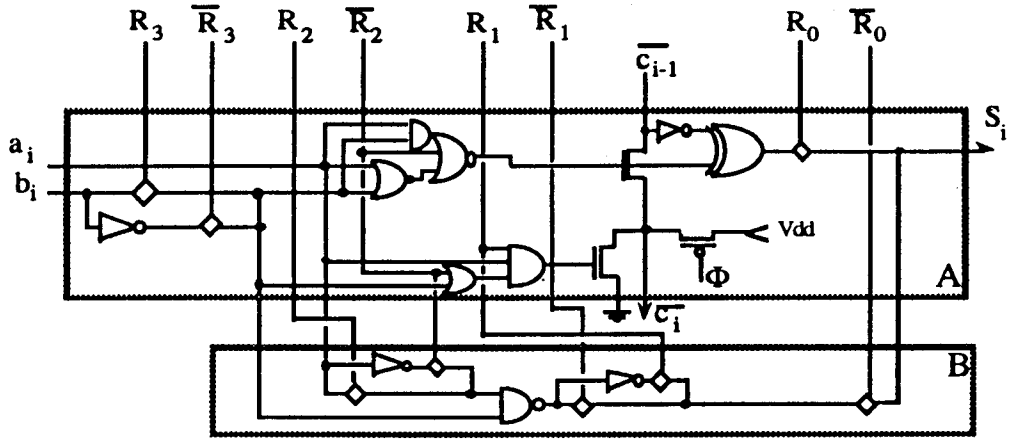



**Figure VI. 11 : UAL avec nombre optimal de signaux de contrôle.**

Le nombre de transistors MOS nécessaires pour implémenter chaque tranche d'UAL est de 54 en figure VI. 11 et est de 78 pour l'UAL FSPD de la figure VI. 10bis. L'augmentation du nombre de transistor est donc de 44%.

Pour une implémentation NORA l'augmentation est de 37% et pour une implémentation Domino l'augmentation est de 31%. Cependant l'implémentation Domino n'est pas intéressante pour des portes ayant peu d'entrées (c'est le cas pour l'UAL) puisqu'elle demande plus de transistors qu'une implémentation statique avec les mêmes performances.

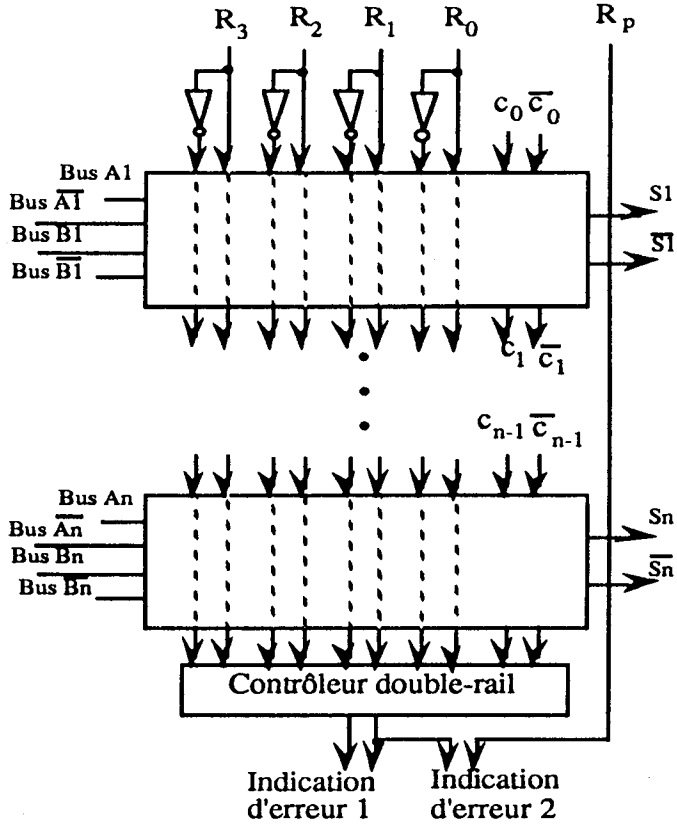



**Figure VI. 12 : L'UAL FSPD globale.**

Finalement, l'UAL FSPD globale est présentée en figure VI. 12, pour laquelle les signaux de contrôle sont protégés d'un bit de parité ($R_p$) et contrôlés par un contrôleur double-rail, comme on verra par la suite en paragraphe VI. 2. 4.

**VI. 2. 3 Les décodeurs :**

Dans une partie opérative on a parfois besoin d'un circuit décodeur traduisant un mot binaire de k bits en un mot codé en 1-parmi-$2^k$. Ce circuit est largement utilisé dans le MC 68000, dans lequel les instructions contiennent des opérations de manipulation de bits.

Le bit à manipuler est décrit par un mot binaire de k bits et le mot encodé en 1-parmi-$2^k$ est utilisé pour masquer tous les autres bits.

La figure VI.13 montre un tel décodeur, il est constitué d'une partie registre et d'une partie décodeur.

Les bits A(0), A(1), A(2), A(3) sont stockés dans le registre. Les bits D(0), D(1), D(2), D(3) sont décodés par la partie décodeur qui donne à sa sortie le numéro du bit à manipuler en code 1-parmi-16.

Les sorties du décodeur sont directes et complémentaires pour avoir accès au bus à structure duale.

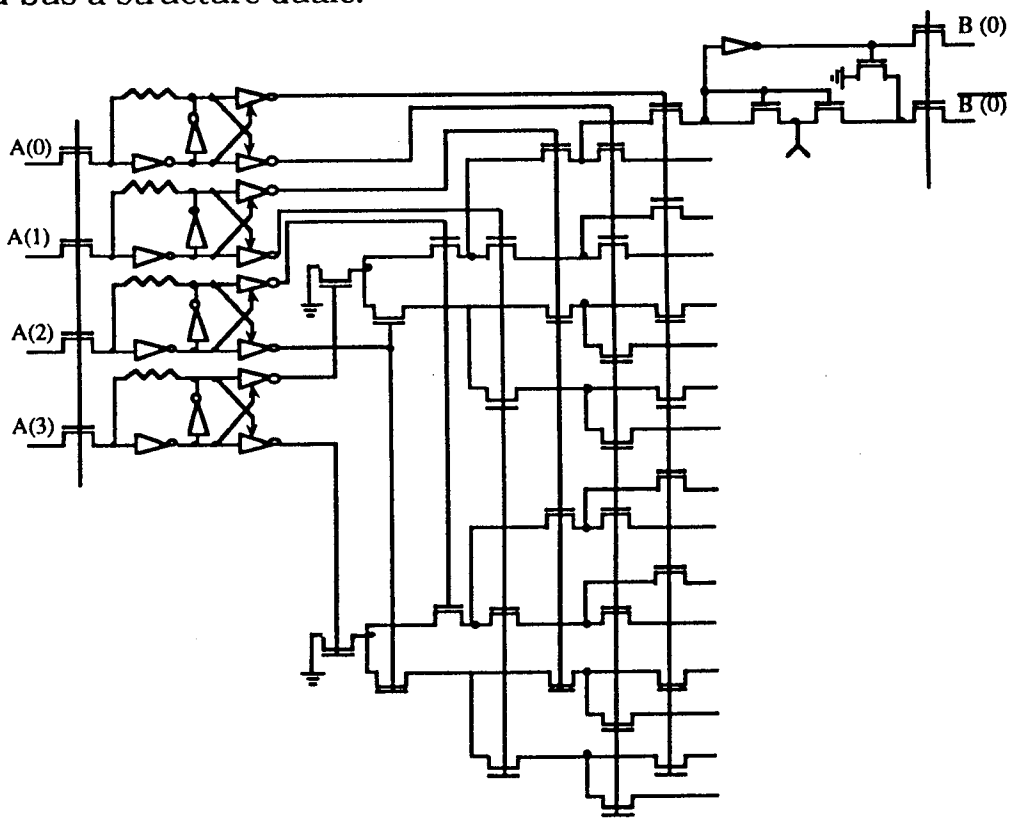



**Figure VI. 13 : Circuit décodeur.**

Une implémentation totalement auto-contrôlable du décodeur de la figure VI.13 est présentée en figure VI. 14. Les entrées de l'arbre du décodeur sont contrôlées par un contrôleur double-rail.

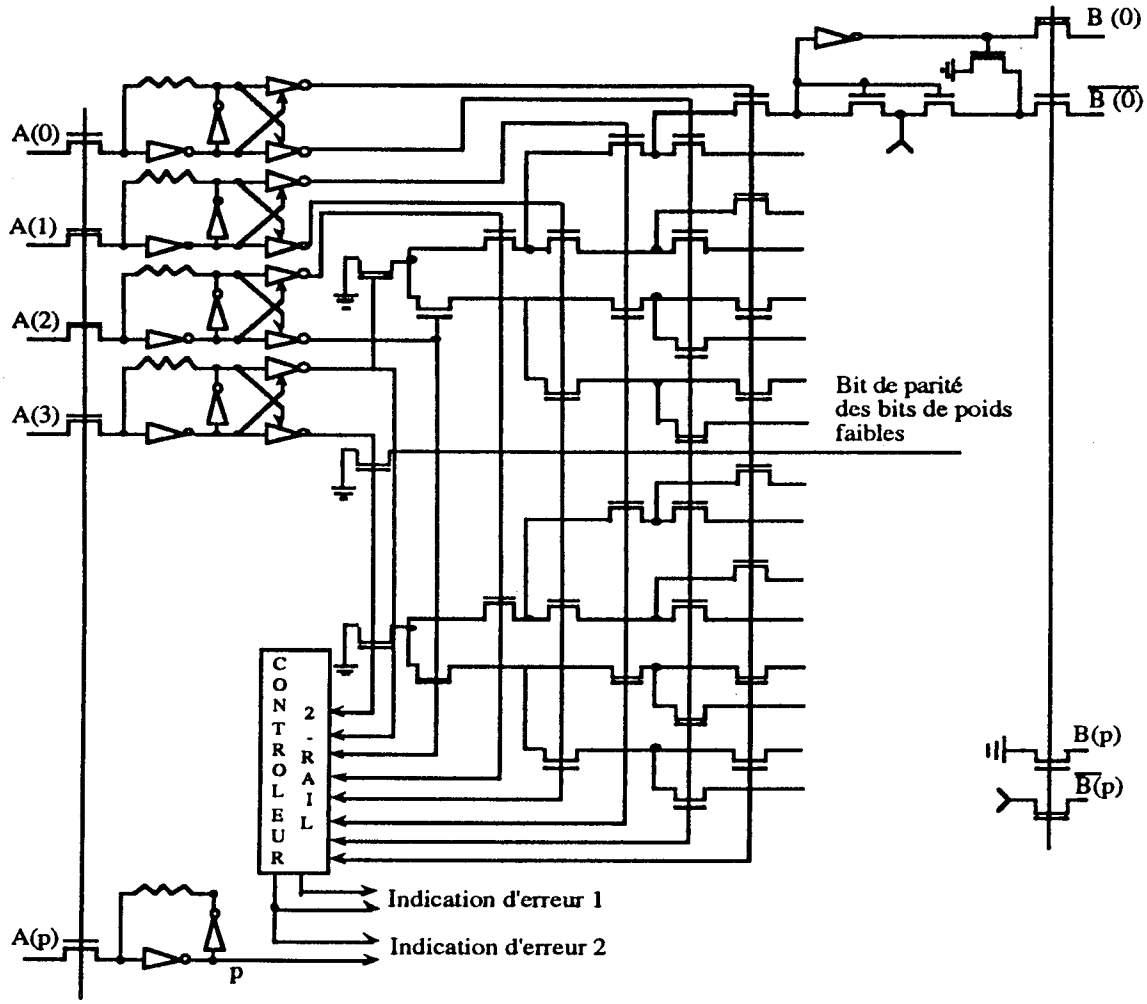



**Figure VI. 14 : Circuit décodeur totalement auto-contrôlable.**

Le circuit est sûr en présence de défauts puisqu'une panne simple (s-on, s-open, ou court-circuit) modifie forcément la valeur d'une ligne de l'arbre. D'autre part, à chaque instant il n'y a qu'un seul chemin sensibilisé entre n'importe quelle ligne de l'arbre et ses sorties. Il en découle que toute panne simple produit une erreur simple dans une ligne interne au circuit, qui va se propager à une seule sortie. L'erreur aux sorties est détectée par un codage par parité.

De plus, on peut vérifier que n'importe quelle panne s-on, s-open, collages et courts-circuits est détectable, donc le circuit est aussi auto-testable.

Le circuit étant sûr en présence de défauts et auto-testable, il est donc totalement auto-contrôlable

**VI. 2. 4 Les contrôleurs double-rail et de parité :**

Le contrôle des sorties des UALs et UA demande l'utilisation de contrôleur double-rail.

D'autre part, les résultats de ces blocs sont injectés dans les bus, qui sont codés et contrôlés par un code de parité.

Pour réduire l'augmentation de circuiterie due à ces contrôleurs, on va utiliser une propriété avantageuse des contrôleurs double-rail : Le contrôleur double-rail est un générateur différentiel de la parité, il admet un ensemble d'entrées codées en double-rail et génère une paire de sortie codée en double-rail qui est le calcul de la parité des entrées.

En figure VI. 15 est représenté le contrôle des sorties d'UAs et de UALs et la génération de la parité.

La même technique peut aussi être utilisée pour des blocs dupliqués tels que le bloc encodeur de priorité.

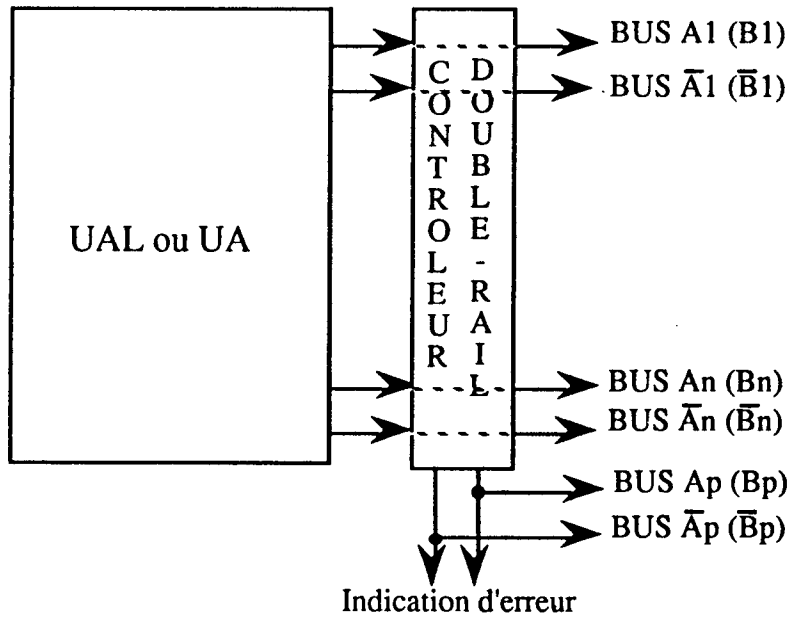



**Figure VI.15 : Contrôle double-rail et génération de la parité de sorties de UAs et de UALs.**

La structure duale des bus dans le MC 68000 demande l'utilisation de deux contrôleurs pour chaque bus, un contrôle des lignes directes du bus et un second pour les lignes complémentaires du bus. Cependant, la propriété des contrôleurs double-rail signalée plus haut permet de contrôler chaque bus en utilisant un simple contrôleur double-rail comme indiqué en figure VI.16.

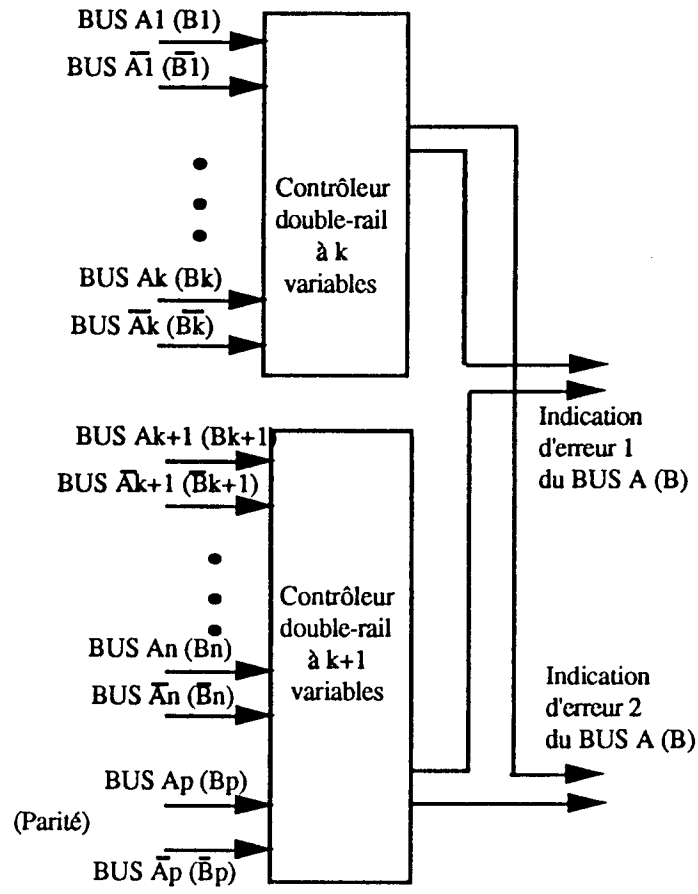



**Figure VI. 16 : Schéma de contrôle des bus.**

L'indication d'erreur 1 et l'indication d'erreur 2, ne prennent pas les valeurs des 4 vecteurs nécessaires (voir annexe 3) pour activer le contrôleur double-rail qui les reçoit (i.e. les vecteurs 0110 et 1001 n'apparaîssent jamais lorsqu'il s'agit d'anticipation de la parité paire dans le bus $A_p$ ($B_p$). Pour cette raison, pour les comprimer, on ne peut pas les envoyer sur la même cellule de contrôleur double-rail (celle-ci ne sera pas suffisamment activée).

Ces signaux sont plutôt à comprimer et à injecter avec d'autres sorties de contrôleurs, dans un contrôleur double-rail final délivrant l'indication globale d'erreur du système. Dans la figure VI. 17, on a associé ces signaux avec d'autres signaux : **Indication d'erreur 1 du BUS A** avec Indication d'erreur 1 du BUS B, et **Indication d'erreur 2 du BUS A** avec Indication d'erreur de l'UAL.

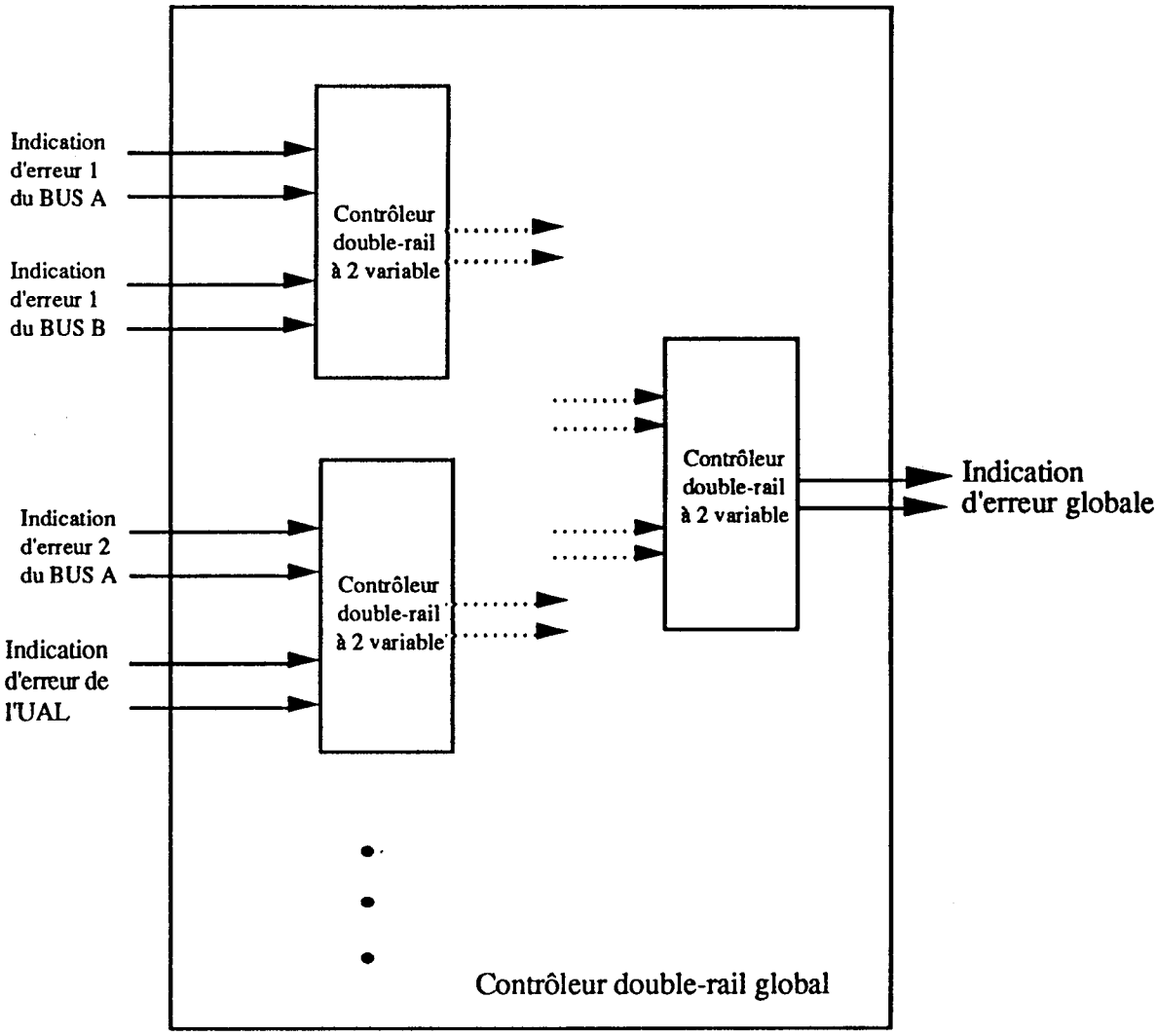



**Figure VI. 17 : Regroupement des indications d'erreur.**

Le contrôleur double-rail à utiliser pour ces structures est représenté en figure VI. 18. Plusieurs de ces cellules peuvent être cascadées pour tester un grand nombre de signaux codés en double-rail, (voir annexe 3).

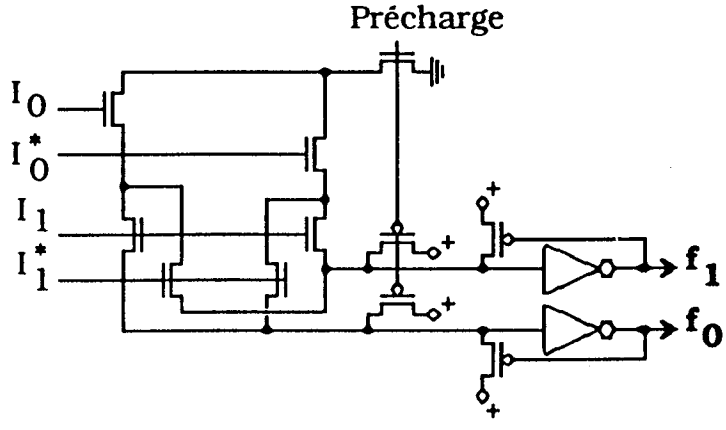



**Figure VI. 18 : Contrôleur double-rail CVSL à 2-paires d'entrées.**

**Conclusion :**

Dans ce chapitre, des méthodes de conception, des différents composants FSPD d'une partie opérative, ont été présentées.

Ces composants FSPD (UA, UAL, décodeurs, registres, ...) sont utilisés pour concevoir une partie opérative auto-contrôlable.

On a pu facilement concevoir ces différents blocs en utilisant des règles générales sans avoir besoin d'examiner en détail les pannes de la classe I pouvant survenir.

La régularité de la partie opérative nous a permis d'établir une stratégie globale et locale de conception en vue d'auto-contrôlabilité. Les structures conçues en structure "bit-slice" sont contrôlées par un code de parité.

Les opérateurs UA, UALs sont conçus de façon qu'ils incluent leurs duals. Leur duplication est ainsi évitée.

Si la couverture des pannes simples est garantie à 100% par une telle conception, dans le chapitre suivant (chapitre VII), des schémas de générateurs intégrés de vecteurs de test sont proposés permettant une plus large couverture de panne.

# CHAPITRE VII

**Implémentation UBIST de parties opératives**

**Introduction :**

Dans cette partie on va présenter des techniques BIST pour des unités arithmétiques (UAs) et unités arithmétiques et logiques (UALs). Ce schéma associé aux UAs et UALs auto-contrôlables présentées au chapitre précédent, constitue le schéma UBIST de celles-ci. Au paragraphe VII.4 sera présenté le schéma synoptique UBIST d'une partie opérative.

On considère les UAs et UALs implémentées sous différentes formes : en NMOS, Domino, CMOS statique ou dynamique.

La technique BIST utilisée ici, est basée sur les concepts de la C-testabilité [FRI 73], [SRI 81]. Le nombre de séquences de test et la taille des générateurs de séquences de test sont donc indépendants du nombre de bits que peut avoir les UAs ou UALs.

Dans [THO 87], une technique de test pseudo-aléatoire est proposée pour des UAs et UALs. Cette technique utilise des séquences longues de test et ne garantit pas toujours 100% de couverture de panne.

La technique BIST adaptée aux structures des UAs et UALs peut être courte et déterministe puisque ces structures sont itératives.

Récemment dans [CER 88], a été proposée une technique BIST pour des UALs implémentées en logique NORA. Mais cette technique ne peut pas être utilisée si l'on considère une UAL implémentée sous une autre forme, ou si l'on utilise la même conception au niveau logique mais que l'implémentation est faite en CMOS statique.

**VII. 1 Implémentation BIST pour les UAs et les UALs :**

La technique BIST d'UAs et d'UALs que l'on expose par la suite [NIC 89b], [NIC 89c] a l'avantage que la complexité des générateurs et la taille des séquences de test qu'ils génèrent sont indépendants du nombre de bits des UAs et UALs à tester. Cette technique est en outre applicable à des structures autocontrôlables, des simulations logiques de cette technique appliquée à une UAL du type de la figure VI.11 ont été effectuées et sont présentées en annexe 9.

D'autre part, il est possible de concevoir ces générateurs pour différentes implémentations d'UAs et d'UALs et différents taux de couverture de panne.

Le schéma BIST proposé est composé d'un LFSR associé à une logique pour générer les séquences de test et utilise un analyseur de signature pour compacter les réponses.

Comme il a été signalé précédemment, la taille du générateur de séquences de test est indépendant du nombre de bits des UAs et UALs, ce qui implique que l'augmentation de surface due au générateur et à l'analyseur

de signature décroît de manière significative lorsque la partie opérative utilise des UAs et UALs à grand nombre de bits.

**VII. 2 Générateur de séquences optimales de test pour UAs :**

Les unités arithmétiques ont une structure itérative unidimentionnelle et unilatérale (voir figure VII. 1). Cette structure convient parfaitement à la C-testabilité.

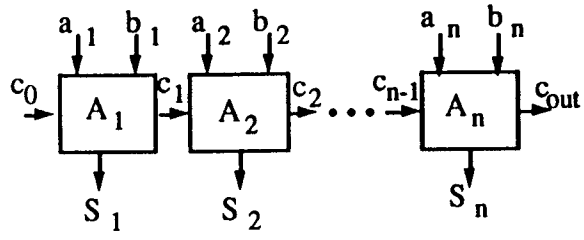



**Figure VII. 1 : Structure itérative des UAs.**

On dit qu'une logique à structure itérative est C-testable, si elle peut être testée par un nombre constant de vecteurs de test indépendamment de la taille de la structure [FRI 73].

Si l'on veut tester des pannes survenant sur une cellule à la fois de la structure de la figure VII. 1, on doit appliquer aux entrées $c_{i-1}$, $a_i$, $b_i$ de chaque tranche $A_i$ une séquence de test telle que chaque panne peut se signaler et s'observer en erreur aux sorties $S_i$ et $c_i$ de cette même cellule.

Pour les pannes détectées en observant $c_i$, il faut aussi s'assurer que les erreurs induites sont propagées jusqu'à certaines sorties $S_i$. Cette dernière condition est toujours valable pour les unités arithmétiques puisque $S_{i+1}= a_{i+1} \oplus b_{i+1} \oplus c_i$, il en découle qu'une erreur sur $c_i$ provoque une erreur sur $S_{i+1}$.

On peut donc assurer un test de haute qualité pour des UAs de la figure VII. 1 en appliquant une séquence de test, telle que chaque cellule $A_i$ est activée exhaustivement. Une telle séquence de test couvrira toutes les pannes combinatoires affectant une cellule du circuit.

Pour avoir un taux de couverture de panne plus élevé encore, on doit générer une séquence de test activant exhaustivement tout couple de cellules consécutive (cellules $A_i$, $A_{i+1}$). De telles séquences de test peuvent détecter toute panne combinatoire affectant une ou deux cellules consécutives. Ceci est nécessaire pour assurer la couverture des pannes doubles, et plus particulièrement la couverture de certaines pannes affectant des lignes communes à des cellules consécutives.

On peut à ce rythme augmenter le taux de couverture de panne en considérant des séquences de test activant exhaustivement k cellules

consécutives $k \in \{3, 4, ...., n\}$.

On trouve une telle séquence en appliquant une séquence de test exhaustif au premier groupe de k cellules $A_1, A_2, ..., A_k$. Pour tester le second groupe $A_2, A_3, ..., A_{k+1}$ de k cellules il suffit de combiner les séquences appliquées sur $c_1, a_2, b_2, a_3, b_3, ..., a_k, b_k$ à certaines séquences appliquées sur $a_{k+1}, b_{k+1}$. Un tel générateur de ces séquences de test va être gros.

Pour réduire la taille du générateur et la durée de test, on doit optimiser la longueur de la séquence de test. La longueur de cette séquence est au moins égale à $2^{2k+1}$, puisque chaque groupe de k cellules consécutives a 2k+1 entrées $(c_{i-1}, a_i, b_i, a_{i+1}, b_{i+1}, ..., a_{i+k-1}, b_{i+k-1})$.

Le LFSR va avoir une longueur minimale de 2k+1 et 2k+1 sorties qu'on va appeler $X_0, X_1, X_2, ..., X_{2k}$.

On doit, d'un autre côté, avoir une simple correspondance entre l'ensemble des sorties du LFSR et les entrées de l'UA : $<X_0, X_1, X_2, ..., X_{2k}> — F \rightarrow <c_0, a_1, b_1, a_2, b_2, ..., a_n, b_n>$ pour pouvoir simplifier la logique additionnelle nécessaire.

La correspondance $< X_0, X_1, X_2, ..., X_{2k}> — F_{i,k} \rightarrow < c_{i-1}, a_i, b_i, a_{i+1}, b_{i+1}, ..., a_{i+k-1}, b_{i+k-1} >$ doit par ailleurs être une relation bijective, pour chaque $i \in \{1, ..., k\}$.

Les ensembles d'entrées et de sorties de $F_{i,k}$ ayant le même nombre de vecteurs $(2^{2k+1})$ implique que si $X_0, X_1, X_2, ..., X_{2k}$ prend toutes les valeurs binaires possibles alors les groupes de cellules $A_i, A_{i+1}, ..., A_{i+k-1}$ sont activées exhaustivement.

Il n'est pas possible de trouver de manière systématique une relation de correspondance optimale, sauf si l'on en trouve une qui vérifie la propriété de bijection et qui soit implémentable avec une logique simple.

Tenant compte des propriétés arithmétiques des retenues, il a été proposé dans [NIC 89b] un théorème dans lequel la relation F peut être implémentée de manière simple et assez proche de la solution optimale.

**Théorème VII.1 [NIC 89b] :** Soit $x_0, x_1, x_2, ..., x_{2k}$ les variables binaires, et soit $x_1'=x_0 \oplus x_1, x_2'=x_0 \oplus x_2, ..., x_{2k}' = x_0 \oplus x2k$ , et soit $<x_0, x_1, x_2, ..., x_{2k}>$ ---- $F_{i,k} \rightarrow < c_0, a_1, b_1, ..., a_n, b_n >$ la relation définie par assignement cyclique des valeurs $x_1, x_2, ..., x_{2k-1}, x_{2k}, x_1', x_2', ..., x_{2k-1}', x_{2k}'$ aux entrées $a_1, b_1, ..., a_k, b_k, a_{k+1}, b_{k+1}, ..., a_{2k}, b_{2k}, ... a_n, b_n$ et la valeur $x_0$ à la retenue $c_0$. La relation est comme suit : $c_0=x_0, a_1=x_1, b_1=x_2, ..., a_k=x_{2k-1}, b_k=x_{2k}, a_{k+1}=x_1', b_{k+1}=x_2', ..., a_{2k}=x_{2k-1}', b_{2k}=x_{2k}', a_{2k+1}=x_1, b_{2k+1}=x_2,...$ Alors pour chaque i la relation $<x_0, x_1, x_2, ..., x_{2k}> --- F_{i,k} \rightarrow < c_{i-1}, a_i, b_i, ..., a_{i+k-1}, b_{i+k-1}>$ est bijective.

La preuve de ce théorème est donnée dans [NIC 89b].

Les séquences de test décrites par ce théorème, et permettant de tester exhaustivement n'importe quel groupe de k cellules consécutives des unités arithmétiques, est appelé test universel d'ordre k des unités arithmétiques.

On implémente les générateurs du théorème VII.1 et pour chaque cellule de k cellules consécutives de l'UAL, en utilisant un LFSR de 2k+1 bascules D et 2k portes XOR (voir figure VII. 2). Le LFSR a un polynôme caractéristique primitif et inclut une porte NOR au rebouclage permettant de générer les $2^{2k+1}$ combinaisons [McC 86].

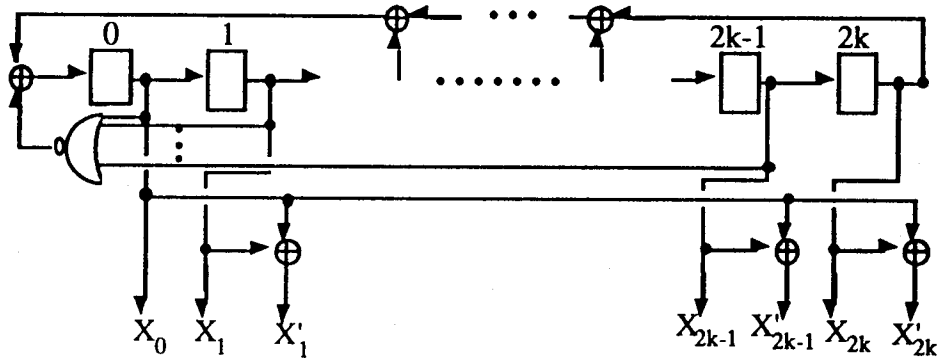



**Figure VII. 2 : Générateur de vecteur de test pour les UAs.**

Le théorème VII. 1 nous permet de concevoir plusieurs générateurs de vecteurs de test pour différentes couvertures de pannes suivant différentes valeurs de k. Ces générateurs permettent de couvrir toutes les pannes pouvant affecter r cellules consécutives avec $r \leq k$.

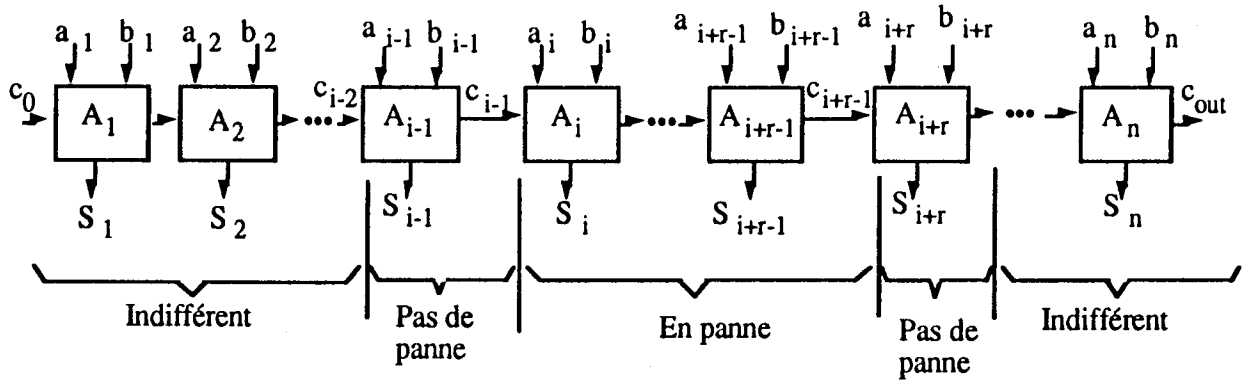
Mais si ces pannes sont combinées avec d'autres pannes affectant d'autres cellules on peut avoir masquage d'erreurs, et l'on ne pourra rien détecter. Il est montré en proposition VII. 1 que le masquage d'erreur ne peut pas survenir si les r blocs consécutifs ont des cellules voisines immédiates sans panne.

**Proposition VII.1** : *Si les entrées $c_{i-1}$, $a_i$, $b_i$, $a_{i+1}$, $b_{i+1}$, ..., $a_{i+k-1}$, $b_{i+k-1}$ de k cellules consécutives $A_i$, $A_{i+1}$, ..., $A_{i+k-1}$ d'une unité arithmétique sont activées exhaustivement, alors on détecte toutes les pannes multiples combinatoires, tel qu'il y a moins de r cellules consécutives en panne, avec $1 \leq r \leq k$ pour lesquelles les cellules voisines $A_{i-1}$ et $A_{i+r}$ soient sans pannes. De manière similaire, si $i+r-1=n$ ($A_{i+r-1}$ est la dernière cellule), on n'a besoin d'avoir que la cellule $A_{i-1}$ sans pannes.*

**Preuve** : Considérons en figure VII. 3, les cellules $A_i$, $A_{i+1}$, ..., $A_{i+r-1}$ en panne, les cellules $A_{i-1}$ and $A_{i+r}$ sans pannes, et les autres restantes $A_1$, $A_2$, ..., $A_{i-2}$, $A_{i+r+1}$, $A_{i+r+2}$, ..., $A_n$ peuvent avoir certaines d'entre elles en panne et éventuellement toutes. Dans cette situation, pour la valeur de $c_{i-2}$ on peut avoir 2 situations :

• Pour certaines séquences, $c_{i-2}$ prend des valeurs erronées, comme l'on a supposé que la cellule $A_{i-1}$ est sans panne, et puisque $s_{i-1} = a_{i-1} \oplus b_{i-1} \oplus c_{i-2}$, alors les erreurs sur $c_{i-2}$ sont propagées à la sortie $s_{i-1}$. Il y a donc détection d'erreurs.

• Les valeurs de $c_{i-2}$ sont correctes pour toutes les séquences de test et comme la cellule $A_{i-1}$ est sans panne alors les valeurs de $c_{i-1}$ sont correctes aussi. On assure donc que toutes les entrées $c_{i-1}$, $a_i$, $b_i$, $a_{i+1}$, $b_{i+1}$, ..., $a_{i+r-1}$, $b_{i+r-1}$ sont activées exhaustivement. Donc la panne est détectée en observant les sorties $s_i$, $s_{i+1}$, ..., $s_{i+r-1}$ , $c_{i+r-1}$. Comme la cellule $A_{i+r}$ est sans panne alors $s_{i+r} = a_{i+r} \oplus b_{i+r} \oplus c_{i+r-1}$ , la panne est détectée en observant les lignes de sortie $s_i$, $s_{i+1}$, ..., $s_{i+r-1}$ , $s_{i+r}$. **(CQFD)**.




Indifférent signifie que les cellules concernées peuvent inclure n'importe quelle type de pannes ou ensemble de pannes, la couverture de panne n'est pas affectée.

**Figure VII. 3 : Testabilité des pannes multiples.**

En se basant sur cette proposition on peut calculer la couverture de pannes multiple.

Pour k=2 et une multiplicité de panne égale à 5 par exemple, les 5 pannes peuvent être distribuées de différentes manières dans les n cellules de l'UA. Si elles affectent 5 cellules différentes il peut y avoir $C^5_n$ distributions possibles. Si 2 pannes affectent une même cellule et 3 autres affectent 3 cellules différentes, il peut y avoir $C^4_n$ distributions différentes des 4 cellules en panne et pour chacune de ces distributions il y a 4 façons différentes pour distribuer ces pannes : (2111), (1211), (1121) et (1112). On aura donc 4 $C^4_n$ distributions. De cette manière on obtient globalement $C^5_n+4C^4_n+6C^3_n+4C^2_n+C^1_n$ façons de distribuer les 5 pannes.

Dans le cas où 5 pannes affectent 5 cellules différentes, on peut avoir (n-4) distributions pour lesquelles la détection n'est pas garantie, elles correspondent au cas où les 5 pannes affectent 5 cellules consécutives.

Si des cellules en pannes sont séparées par d'autres (par exemple, 3-2, 4-1, 3-1-1 etc...) la proposition1 assure la détection de panne.

De la même manière, si 5 pannes affectent 4 cellules différentes on trouve 4(n-3) pannes dont la détection n'est pas garantie par la proposition1.

Finalement on trouve que la limite inférieure du taux de couverture de panne s'écrit:

$$1 - \frac{(n-4) + 4(n-3) + 6(n-2)}{C^5_n+4C^4_n+6C^3_n+4C^2_n+C^1_n}$$

La figure VII. 4 donne les générateurs qui peuvent être utilisés pour les cas k=1, 2 ou 3.

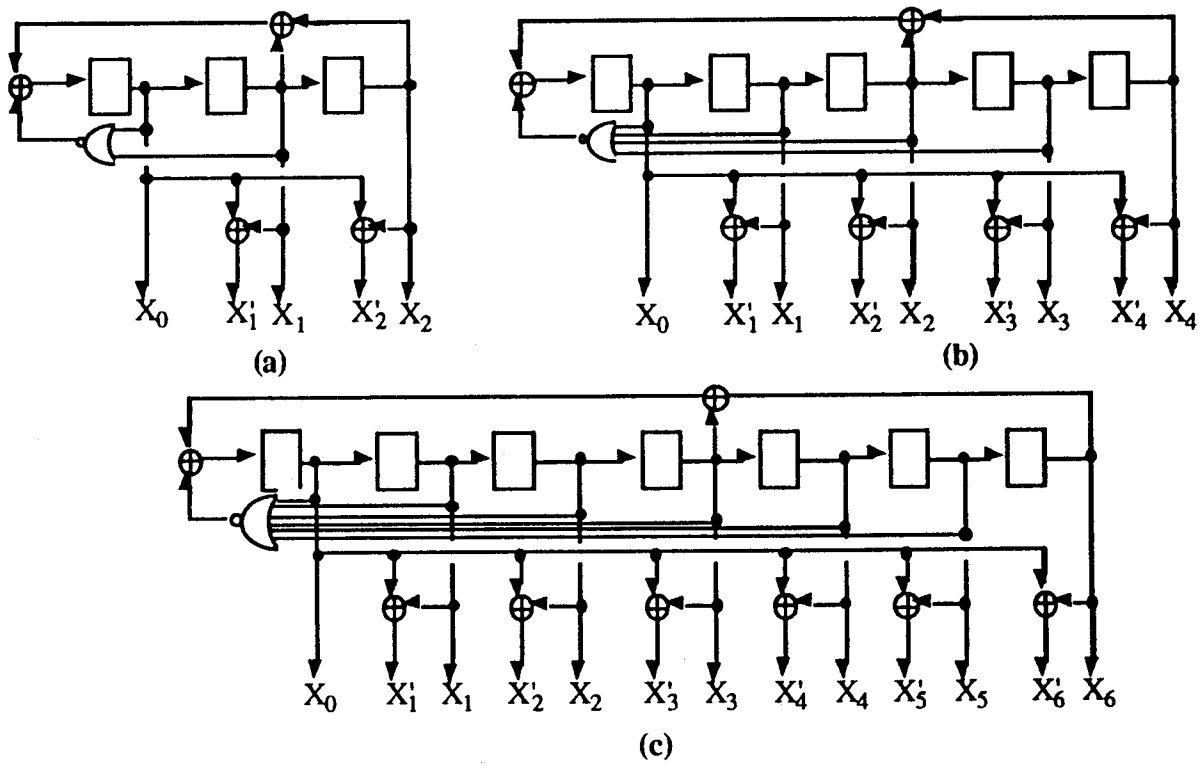



**Figure VII. 4 : Générateurs de vecteurs de test pour a) k=1, b) k=2, c) k=3.**

| | m=1 | m=2 | m=3 | m=4 | m=5 |
|---|---|---|---|---|---|
| n=4 | 100% | 70 | 60 | 54,3 | 50 |
| n=8 | 100% | 80,6 | 83,4 | 83,6 | 82,8 |
| n=16 | 100% | 85,8 | 94,2 | 95,3 | 95,8 |
| n=32 | 100% | 94,1 | 98,5 | 98,8 | 99,2 |

k=1

| | m=1 ou 2 | m=3 | m=4 | m=5 |
|---|---|---|---|---|
| n=4 | 100% | 90 | 80 | 71,4 |
| n=8 | 100% | 95 | 93 | 92,4 |
| n=16 | 100% | 98,2 | 98,5 | 99 |
| n=32 | 100% | 99,3 | 99,8 | 99,9 |

k=2

| | m=1, 2 ou 3 | m=4 | m=5 |
|---|---|---|---|
| n=4 | 100% | 97 | 92,8 |
| n=8 | 100% | 98,5 | 96,9 |
| n=16 | 100% | 99,7 | 99,5 |
| n=32 | 100% | 99,9 | 99,96 |

k=3

m : multiplicité des pannes.

n : nombre de cellules de l'unité arithmétique.

**Table VII. 1 : Couverture des pannes multiples.**

Le taux de couverture de panne atteint par ces générateurs est donné en table VII.1. Cette table donne les limites inférieures des taux de couverture de panne pour les multiplicités 1, 2, 3, 4 et 5 quand on utilise un test universel d'UA d'ordre k=1, 2 et 3. Les taux de couverture réels sont évidemment plus élevés que ces valeurs, puisque la plupart des pannes non garanties d'être détectées par la proposition VII.1 le sont. Ceci peut être expliqué de la même façon qu'une séquence de test prévue pour détecter des pannes simples est capable de détecter une grande partie des pannes multiples.

### VII. 3 Générateurs de séquences optimales de test pour les UALs :

Les UALs ont une structure similaire aux UAs. Les tranches d'UALs sont plus complexes et admettent des signaux d'entrée de contrôle supplémentaires.

On doit modifier les générateurs proposés pour les UAs de façon à tester exhaustivement les lignes de contrôle.

Le générateur de test va être de plus en plus complexe et les séquences de plus en plus longues, si l'on dispose de beaucoup de lignes de contrôle. C'est pour cette raison que les UALs qu'on va considérer doivent avoir un encodage optimal des signaux de contrôle, et par conséquent un nombre optimal de lignes de contrôle.

On peut implanter une UAL avec au plus 4 lignes de contrôle, pouvant effectuer les 10 opérations logiques non triviales de 2 variables, l'addition, la soustraction et le décalage à gauche. La figure VII. 6 montre une telle UAL et la table VII. 2 donne sa table de vérité.

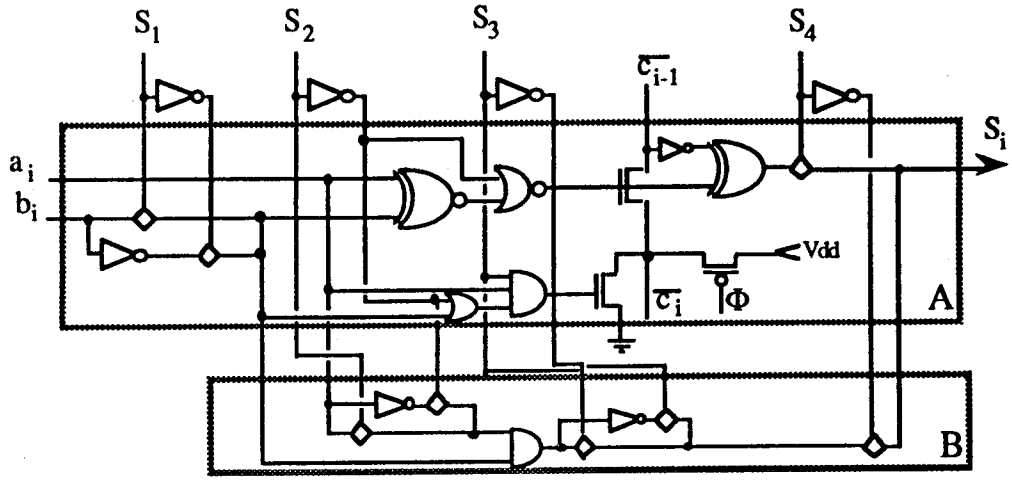



**Figure VII. 6 : UAL avec encodage optimal des signaux de contrôle.**

| S1,S2 \ S3,S4 | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 00 | $\overline{A} \wedge \overline{B}$ | 0 | $A \vee B$ | shift A |
| 01 | $A \wedge \overline{B}$ | $\overline{A \oplus B}$ | $\overline{A} \vee B$ | A-B |
| 10 | $\overline{A} \wedge B$ | 0 | $A \vee \overline{B}$ | shift A |
| 11 | $A \wedge B$ | $A \oplus B$ | $\overline{A} \vee \overline{B}$ | A+B |

**Table VII. 2 : Table de vérité de l'UAL de la figure VII. 6.**

Pour concevoir les générateurs pour les UALs, on reprend ceux qui ont été proposés pour les UAs pour lesquels on rajoute au LFSR un nombre de bascules D égal au nombre de signaux de contrôle, on génère ainsi une séquence exhaustive de ces signaux.

En figure VII. 7 est proposée un générateur de vecteurs de test composé d'une partie générant les données d'entrée de l'UAL, celle-ci est similaire au générateur d'UA et de 4 bascules D supplémentaires générant exhaustivement les signaux de contrôle pour une UAL à 4 signaux de contrôle.

Pour les générateurs des figures VII.4a) VII.4b) et VII.4c) cette modification se traduit par l'utilisation d'un LFSR ayant respectivement 7, 9, et 11 bascule D.

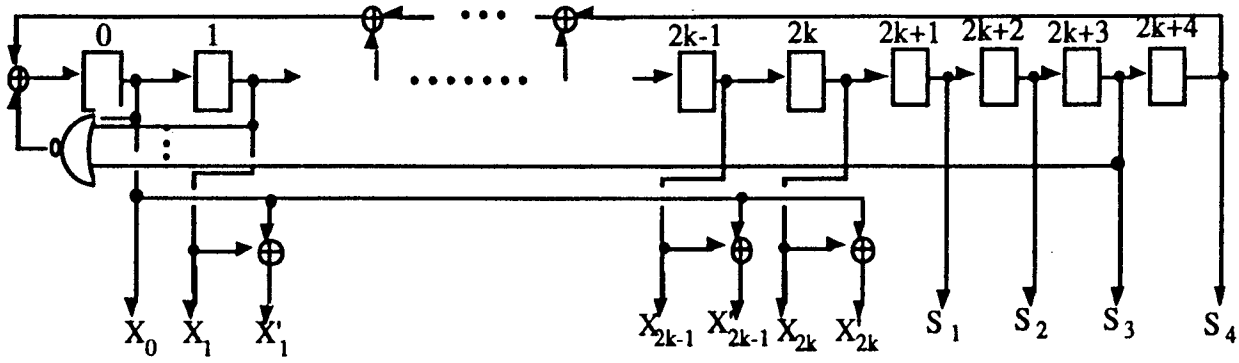



**Figure VII. 7 : Générateur de vecteurs de test pour des UALs à 4 signaux de contrôle.**

• Durant le décalage à gauche, le signal de propagation $p_i$ de chaque tranche d'UAL est désactivé ($p_i$=0) pour bloquer la propagation de retenue. Ce signal intervient sur le calcul des sorties (i.e. $s_i = p_i \oplus c_{i-1}$) d'où $s_i = c_{i-1}$. D'autre part la retenue sortante de chaque tranche prend la valeur d'une entrée de la tranche (par exemple on peut appliquer sur l'entrée $a_i$ la donnée à décaler). D'où pour une tranche donnée $A_i$ on a $c_{i-1} = a_{i-1}$, on obtient $s_i = a_{i-1}$, ce qui correspond à un décalage à gauche.

• Durant chaque opération logique, les signaux de contrôle forcent toutes les retenue à une valeur constante. Les sorties de chaque tranche ne dépend donc que de ses entrées.

On remarque que les tranches d'UAL ne peuvent pas être activée exhaustivement. Par exemple un vecteur encodant une opération logique sur les lignes de contrôle ne peut être combiné qu'avec la valeur 0 ou avec la valeur 1 appliquée sur la retenue entrante de chaque tranche.

Comme on va le voir dans la suite cette contrainte ne réduit pas le taux de couverture de panne.

**Proposition VII. 2** : *Si r ($r \leq k$) cellules consécutives ($A_i$, $A_{i+1}$, ..., $A_{i+r-1}$) d'une UAL sont affectées par un ensemble de pannes détectables et si les tranches $A_{i-1}$ et $A_{i+r}$ sont sans pannes, alors le générateur de vecteur de test de la figure VII. 7 permet la détection des pannes dans l'UAL pour n'importe quelle distribution de pannes.*

La preuve de cette proposition est donnée dans [NIC 89b].

Si l'on veut tester les pannes de type s-open dans l'UAL, on doit utiliser comme pour l'UA, deux fois plus de bascules D pour le LFSR. Pour une UAL à 4 signaux de contrôle, le LFSR doit avoir 14 bascules D dans le cas k=1,

18 bascules D pour k=2, et 22 pour k=3. On remarque que pour k=2 la durée du test est grande mais acceptable, par contre pour k=3 la durée de test devient critique.

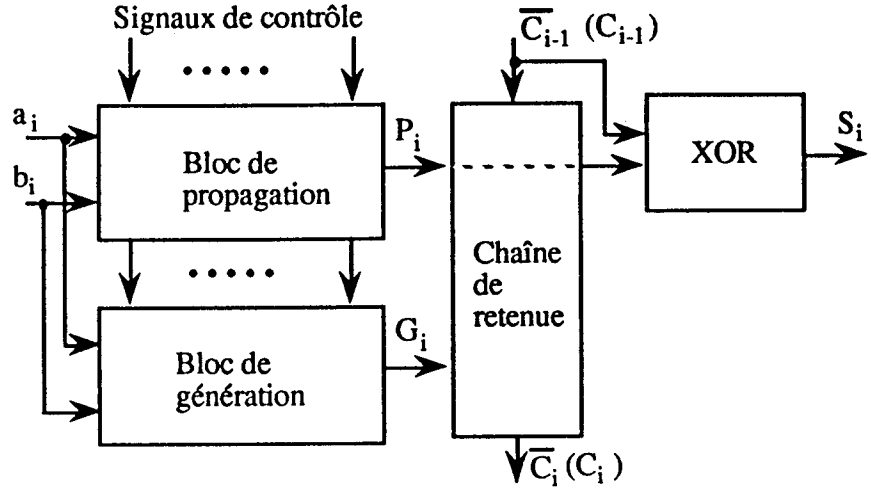



**Figure VII. 8 : Structure générale d'une UAL.**

Les unités arithmétiques et logiques sont généralement implémentées suivant la structure de la figure VII. 8.

Les blocs de propagation et de génération sont souvent implémentés en logique CMOS dynamique (logique Domino par exemple), alors que la porte XOR est souvent implémentée en logique CMOS statique et rarement en CMOS dynamique.

On peut réduire la durée d'application du test par la proposition VII.3.

**Proposition VII. 3** : *Si les pannes de collage des sorties de portes des blocs de propagation et de génération de la figure VII. 8, sont détectables en effectuant l'opération d'addition, et si ces blocs sont implantés en logique CMOS dynamique (logique Domino ou NORA) alors les pannes s-open dans l'UAL sont détectables soit par une séquence de test, soit en effectuant l'opération d'addition sur des couples d'entrées.*

La preuve de cette proposition est donnée dans [NIC 89b].

Si une panne s-open survient sur le transistor Q4, la sortie de la porte de la figure VII. 9a) est constamment déconnectée de la masse Vss.

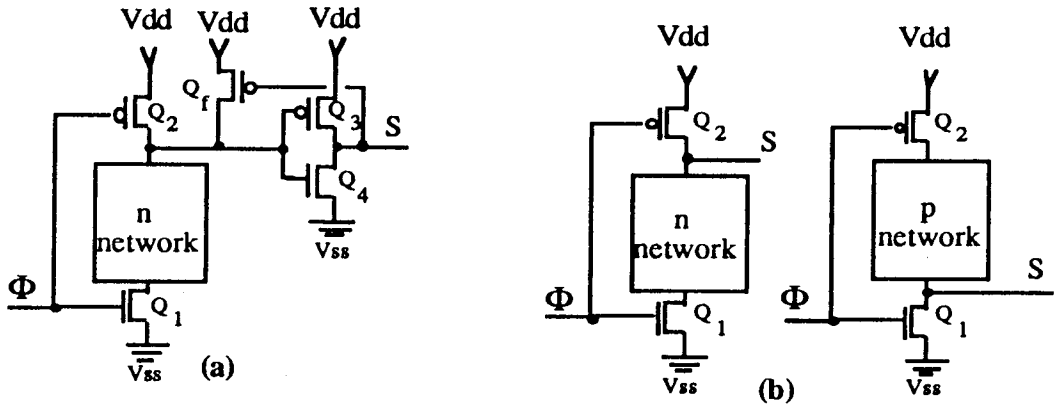



**Figure VII. 9 : Portes à logique dynamique.**

Un vecteur d'initialisation pour cette panne peut survenir lors de l'application des vecteurs de test, la sortie S pour ce vecteur se met à l'état "1". Le vecteur de test peut alors survenir après cette initialisation. Néanmoins, dû aux courants de fuite la sortie peut se décharger et l'initialisation disparaît, il est donc nécessaire que le vecteur de test soit appliqué immédiatement après l'application du vecteur d'initialisation.

La séquence de test de la proposition VII.3 peut être appliqué en 2 phases comme suit :

a) Effectuer les opérations d'addition en appliquant tous les couples d'entrées appartenant à la séquence de test qui couvre les pannes combinatoires (les séquences de test universel pour les UAs définies au paragraphe VII. 2).

b) Effectuer chaque opération logique et chaque opération arithmétique en appliquant toutes les données d'entrées appartenant à la séquences de test couvrant les pannes combinatoires (les séquences du test universel pour les UAs définies au paragraphe VII. 2).

La figure VII.10a) montre le circuit générant des séquences du type a) et b) pour des données d'entrées appartenant à un test universel d'UA d'ordre k=1. Le LFSR est initialisé à 000...00 et le flip-flop F à 0. La séquence a) est appliquée durant la première phase de test. Durant cette phase la sortie Q du flip-flop F est à 0 et force les signaux de contrôle S1, S2, S3 et S4 encodant une opération d'addition (0011 en figure VII.10).

Pour cette phase les cellules du LFSR génère tous les couples de donnée appartenant au test universel d'UAs d'ordre k=1.

A la fin de cette phase, les valeurs des cellules du LFSR sont 00...01 (si le LFSR comporte une porte NOR pour la génération de $2^n$ vecteurs et s'il est initialisé dans l'état 000...00, alors le dernier état du LFSR est 000...01 quelleque soit la longueur n du LFSR). D'où, à la fin de cette phase la bascule F est mise à 1 et les signaux de contrôle S1, S2, S3, et S4 prennent les

valeurs complémentaires des valeurs générées par leur cellule correspondante du LFSR. Pendant cette deuxième phase de test, on va donc générer la séquence b).

La même technique est utilisée en figure VII.10b) pour un test universel d'UAs d'ordre k=2. Le LFSR a 10 bascules D pour générer tous les couples de donnée de la séquence a).

En général, dans le cas de séquences de test d'ordre k (k=2, 3, ...), le LFSR va avoir k+2 bascules D.

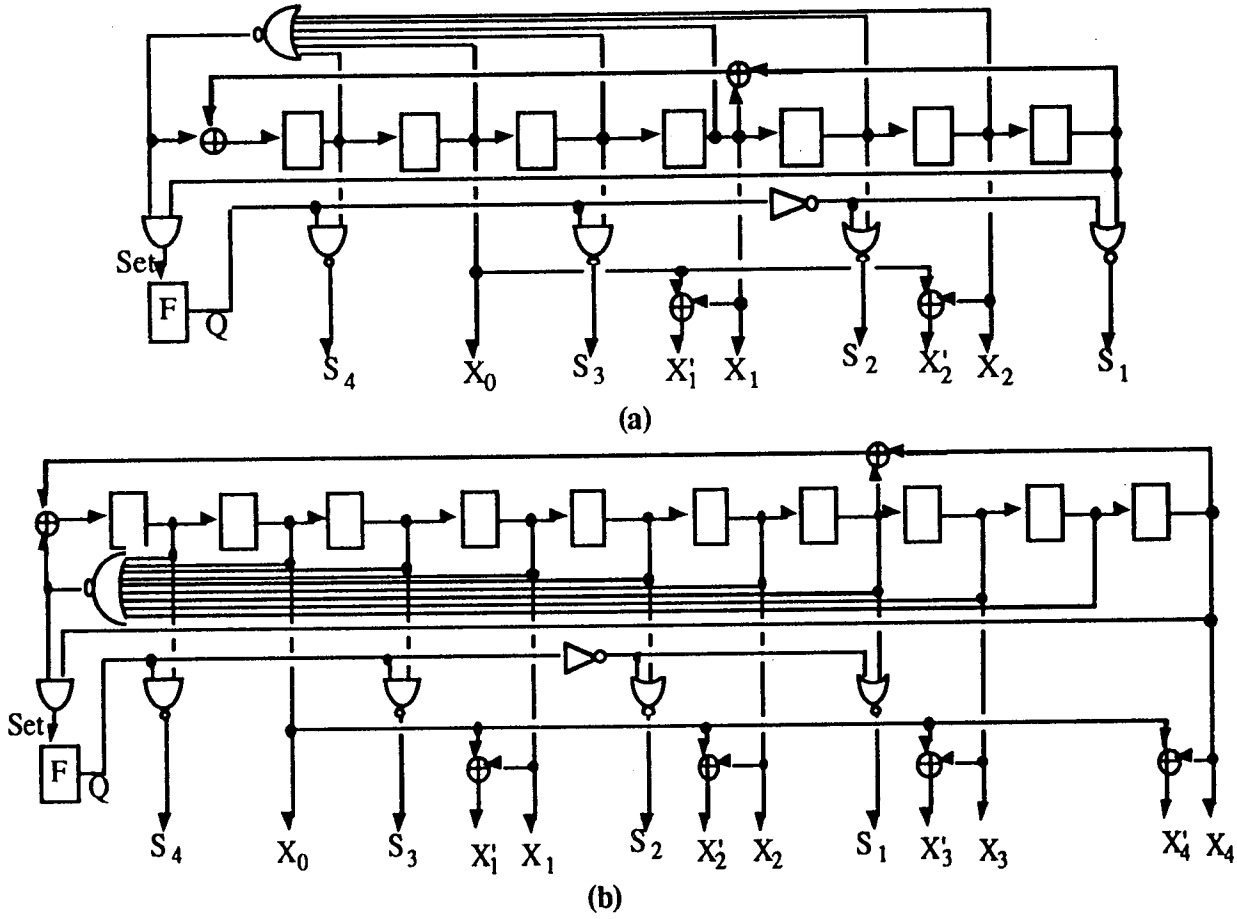



**Figure VII . 10 : Générateur de vecteurs de test pour des UALs en CMOS dynamique à 4 signaux de contrôle et pour un test universel d'ordre, a) k=1, b) k=2.**

Ces générateurs sont valables quand l'UAL vérifie les hypothèses de la proposition VII.3. Les pannes de collage des sorties de portes des blocs de propagation et de génération, sont détectées en effectuant l'opération d'addition.

Ceci est vrai pour la plupart des UALs, en effet les blocs de propagation et de génération sont souvent implémentés en utilisant une logique complexe. Dans ce cas l'hypothèse requise est vérifiée automatiquement puisque l'opération d'addition ne peut s'effectuer correctement si les

sorties Pi et Gi de ces blocs sont collées à 0 ou à 1.

Par exemple dans [MEA 80] est proposée une implémentation en porte complexe NMOS pour les blocs de propagation et de génération. Cette porte est implémentée dans [CER 88] en logique NORA et est utilisée pour une UAL en CMOS. La figure VII. 11a) représente cette porte en logique Domino.

Dans d'autres cas, les blocs de propagation et de génération sont implémentés en utilisant plusieurs portes. La plupart du temps l'opération d'addition ne peut être effectuée correctement si les sorties des blocs imbriqués sont collées à la valeur 0 ou à 1. Un tel cas est représenté en figure VII.11b) où l'UAL considérée est celle utilisée dans le micro-processeur MC68000.

Le bloc de génération est une porte complexe AND-NOR, et le bloc de propagation est composé d'une porte complexe AND-NOR et d'une porte primitive NOR. La sortie de la porte NOR imbriquée vérifie l'hypothèse de la proposition P3.

L'UAL de la figure VII.11b) effectue l'addition quand S1, S2, S3 et S4 prennent la valeur 0, 0, 1, 1. Les générateurs de la figure VII.10 peuvent être utilisés pour tester une implémentation en CMOS de cette UAL si les blocs de propagation et de génération sont implémentés en logique CMOS dynamique (Domino, NORA, ...).

L'UAL de la figure VII.11a) a 8 signaux de contrôle $p_0$, $p_1$, $p_2$, $p_3$, $g_0$, g1, $g_2$ et $g_3$, et effectue une opération d'addition quand ces signaux prennent la valeur 01100001. Le générateur de vecteurs de test de la figure VII.12 peut alors être utilisé pour générer les séquences a) et b) décrites précédemment et correspond à un test universel d'UA d'ordre k=1. La longueur de la séquence de test (séquence a) et b)) est de $2^{12}$ vecteurs de test. En comparaison avec celle donnée dans [CER 88], cela correspond à deux fois moins de vecteurs de test, donc deux fois moins de durée de test. De plus l'augmentation de surface utilisée pour la vérification des réponses de l'UAL est plus petite (on n'utilise pas de filtres et des portes XOR).

La testabilité dans notre cas est meilleure. Dans [CER 88] les pannes s-open dans les portes CMOS dynamique sont testées en générant 2 vecteurs, le premier correspond à l'initialisation, le second est un vecteur de test.

Pour notre technique, on peut augmenter le taux de couverture des pannes multiples en utilisant un test universel d'UA d'ordre k supérieure (k=2, 3, ...). Pour k=2, on aura besoin pour tester la même UAL d'un LFSR à 13 bascules D et la longueur du test sera $2^{13}$ vecteurs. Pour k=3, on a 15 bascules D pour le LFSR et la longueur du test est de $2^{16}$ vecteurs.

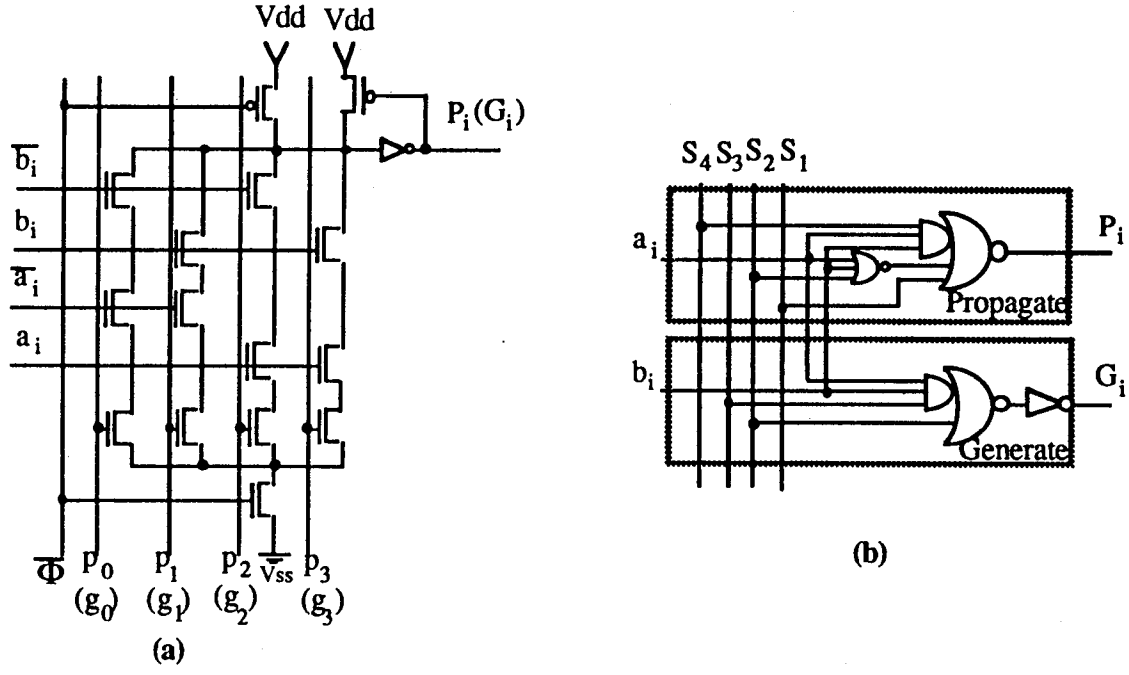



**Figure VII. 11 : Blocs de propagation et de génération, a) version CMOS de l'UAL proposée dans [MEA 80], b) UAL du 68000.**

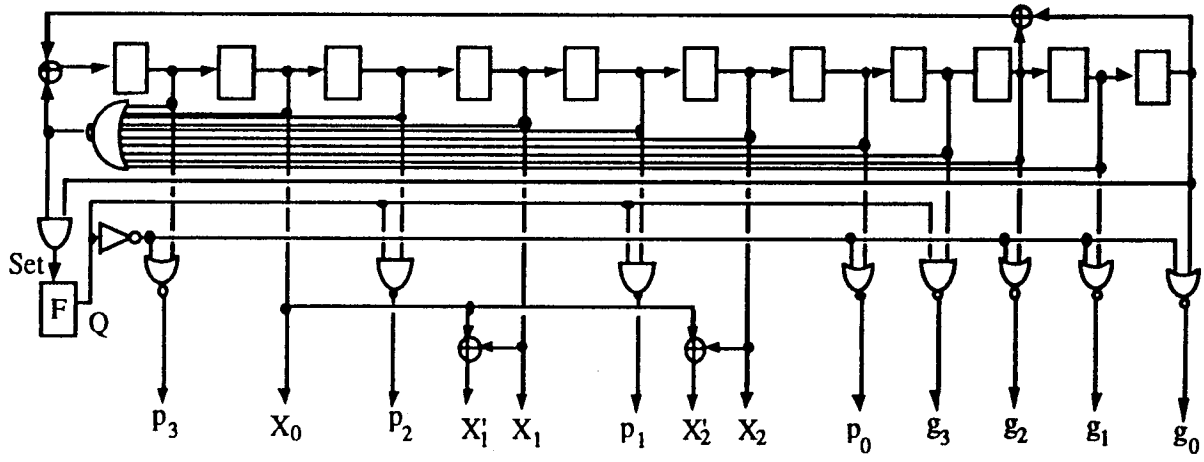



**Figure VII. 12 : Générateur pour l'UAL de la figure VII.11a).**

D'autre part, pour l'UAL utilisant un encodage optimal de ses signaux de contrôle (UAL de la figure VII.6), la technique est valable pour le bloc A de l'UAL (ce bloc effectue l'addition, la soustraction, les opérations XOR et $\overline{XOR}$, et le décalage à gauche).

Pour tester les pannes du type s-open du bloc B, on doit générer deux autres séquences (c et d) qui sont validées par les valeurs 01 et 11 des bascules F1 et F2 (voir figure 13).

La porte ET du bloc B peut être implémentée aussi bien en logique CMOS statique que dynamique. Les séquences c et d testent aussi bien les pannes de s-open de cette porte que celle des inverseurs et des transistors de passage. Les séquences a et b décrites précédemment sont validées par

les valeurs 00 et 10 des bascules F1 et F2.

Lors de l'initialisation du générateur, les bascules F1 et F2 sont mises à zéro en même temps que le LFSR.

On trouvera en annexe 9 les résultats d'une simulation logique du générateur de la figure VII.13 activant une UAL à 2 tranches de bits du type de la figure VII.6.

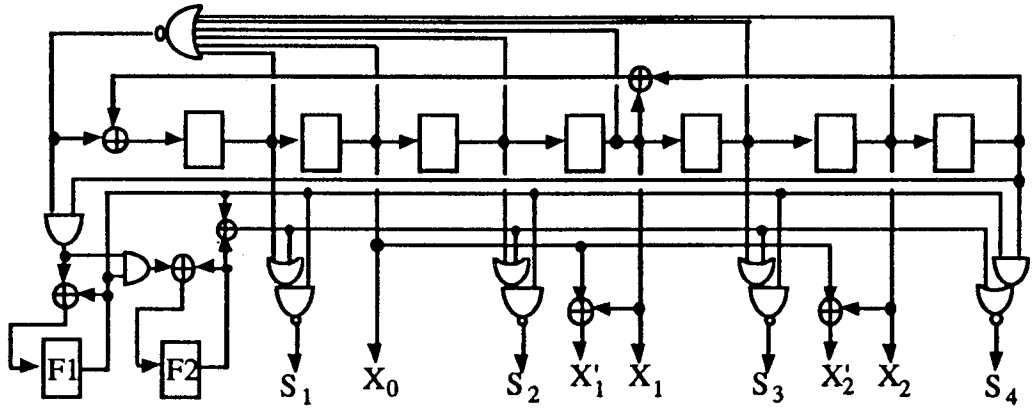



**Figure VII.13 : Générateur pour l'UAL de la figure VII.6**

**VII. 4 Schéma UBIST de parties opératives :**

Au chapitre VI a été proposé le schéma d'autocontrôlabilité d'une partie opérative du type de celle du MC 68000. La structure de parties opératives de ce type permet une accessibilité des différents blocs par l'intermédiaire des bus.

Les différents blocs peuvent alors utiliser le même générateur de vecteurs de test et le même analyseur de signature. Un seul analyseur de signature est utilisé pour compacter les réponses des blocs de la partie opérative.

Un des générateurs de vecteurs de test présentés en figure VII.10 peut être utilisé pour tester l'unité arithmétique et logique en deux phases (les séquences a et b décrites en paragraphe VII. 3).

Le même générateur va tester l'unité arithmétique de bits de poids faible et l'unité arithmétique de bits de poids fort. Dans ce cas seules les sorties ($X_0$, $X_1$, $X'_1$, $X_2$, $X'_2$) du générateur de la figure VII.10.a, ou les sorties ($X_0$, $X_1$, $X'_1$, $X_2$, $X'_2$ , $X_3$, $X'_3$, $X_4$, $X'_4$) du générateur de la figure 10. b, sont connectées aux unités arithmétiques.

Chaque unité arithmétique est testée en une phase et reçoit tous les couples de vecteurs appartenant au test universel d'ordre 1 (resp. d'ordre 2).

Le test du registre à décalage demande d'effectuer les différentes opérations de décalage à droite et à gauche et de rotation à droite et à

gauche, par l'application des données suivantes : 000...0, 111...1, 0101...01 et 1010...10. Ces données peuvent être stockées dans les blocs de constantes (les constantes 000...0 et 111...1 existent déjà dans la partie opérative originale). Le test du décodeur peut être fait en testant exhaustivement les bits $a_1$, $a_2$, $a_3$, et $a_4$ du bus A. On effectue ce test en connectant 4 sorties du générateur de vecteurs de test à ces bits là.

Le test de l'encodeur de priorité (PREN) peut être fait en appliquant la séquence 111...1, 011...11, 001...11, ...., 000...01, 000...00. Cette séquence peut être générée en chargeant la constante 111...11 dans le registre à décalage et en exécutant un certain nombre de décalage à droite avec une valeur 0 comme bit d'entrée série du registre.

Finalement, les registres peuvent être testés en envoyant les donnée des constantes 000...00, 111...11, 0101...01 et 1010...10. On peut utiliser les algorithmes proposés dans [MAR 82] pour assurer la détection de couplage de pannes. Dans ce cas l'adressage est assuré par un LFSR UP/DOWN [NIC 85] à 4 cellules.

En conclusion, la surface rajoutée pour une implémentation BIST d'une partie opérative est déterminée par le rajout de : un analyseur de signature (de 16 bits pour le MC 68000), un générateur de ceux proposés en figure VII.10, un LFSR UP/DOWN à 4 cellules et des constantes supplémentaires (i. e. 0101...01 et 1010...10). Les constantes 000...00 et 111...11 existent déjà dans la partie opérative originale.

Cette implémentation est très efficace comparée à la complexité de la partie opérative.

La figure VII.14 représente un schéma synoptique d'une implémentation UBIST de la partie opérative. Les blocs représentés en gris clair correspondent aux différents contrôleurs introduits à la partie opérative autocontrôlable. Les blocs représentés en gris foncé correspondent au générateur de vecteurs de test et à l'analyseur de signature, ils sont connectés au bus double-rail A et au bus double-rail B. Ces blocs représentent l'augmentation essentielle en surface de la partie opérative autocontrôlable.

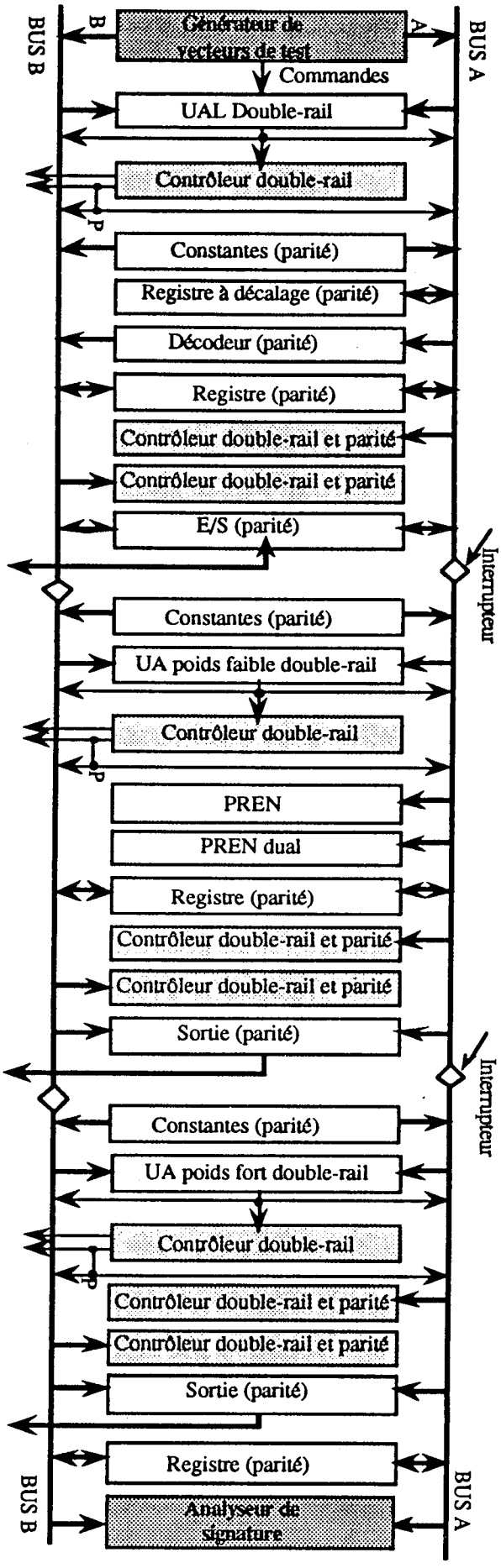



**Figure VII.14 : Schéma synoptique d'une partie opérative UBIST.**

**Conclusion :**

Dans ce chapitre nous avons proposé un schéma UBIST de parties opératives.

L'élaboration de ce schéma s'est faite par une étude préliminaire de test hors-ligne d'unités arithmétiques et d'unité arithmétiques et logiques.

Des générateurs de vecteurs de test utilisant des concepts de C-testabilité sont proposés dans [NIC 89b], [NIC 89c]. Ces générateurs génèrent des séquences de test indépendamment du nombre de bits des UAs et des UALs. Ils ont l'avantage d'être applicables à des UAs et UALs autocontrôlables.

D'autre part, l'augmentation en surface, due à ces générateurs, est indépendante du nombre de bits des UAs et des UALs. Cette augmentation décroît donc de manière significative, lorsqu'on considère des parties opératives utilisant des UAs et des UALs de grande taille.

Ce schéma BIST associé au schéma d'autocontrôlabilité (proposé au chapitre précédent), constitue le schéma UBIST de parties opératives de type "bit-slice". Un schéma synoptique d'une telle implémentation UBIST est proposé.

# CONCLUSIONS

## CONCLUSIONS

L'étude présentée dans cette thèse s'est voulue unificatrice de deux domaines de conception de systèmes: La compilation de silicium et le test.

L'avenir du test sera l'automatisation de certaines de ses techniques, il rejoindra donc les chaînes d'outils de CAO, à différents niveaux d'abstraction, donc la compilation de silicium.

La compilation de silicium doit garantir que son produit est fiable et "viable", elle inclura donc nécessairement des techniques, à tout niveau, de testabilité et de sécurité.

Nous avons voulu montrer certains aspects de ces deux domaines et leurs intérêts communs afin qu'en émane une stratégie de compilation de circuits autotestables.

Ailleurs, des tentatives ont été menées [FUN85], [FUN 86], [BEE 89], [SAB 86] ..., pour greffer des outils automatiques de testabilité industriellement éprouvés, à un compilateur de silicium. Cela ne s'est pas traduit par un échec, mais le produit résultant reste quand même à performances moyennes par rapport à celles de ses outils originels.

Une stratégie visant à avoir une architecture cible, des performances surface/vitesse, une très bonne couverture de panne, une complexité du test et un temps de fabrication raisonnables, est à établir si les circuits intégrés numériques ou analogiques veulent un jour envahir les marchés de systèmes de haute sécurité où leur réalisation se fait encore très souvent sous forme de systèmes à composants discrets et/ou électromagnétiques.

Le schéma de test de circuits intégrés qu'on a adopté répond à des critères stricts de testabilité et de sécurité.

En effet, les hypothèses de pannes considérées sont les hypothèses de pannes physiques réelles (au niveau transistor).

Quand une de ces pannes survient, on s'est fixé comme objectif de détecter immédiatement les erreurs produites.

Les techniques de test intégré en-ligne, tels que les circuits auto-contrôlables, garantissent l'accomplissement d'un tel objectif.

D'autre part, des techniques de test intégré hors-ligne (schémas BIST du type BILBO par exemple), activent le circuit en des phases de test en dehors du fonctionnement normal. Le circuit reçoit d'autres séquences de vecteurs d'entrée que celles survenant pendant son fonctionnement normal. Certaines de ces séquences sont capables de détecter des pannes multiples indétectables par le schéma d'autocontrôlabilité.

L'unification des tests intégrés en ligne et hors-ligne (appelée UBIST [NIC 88]) a permis de tirer les avantages de chacun de ces tests. En effet, on est capable, grâce à ce schéma, d'effectuer la plupart des tests nécessaires au circuits intégrés (test de fin de fabrication, test de maintenance, test en ligne, etc...)

Nous avons établi le schéma UBIST de parties contrôle hiérarchisées à base de PLAs. Une solution topologique efficace à ce schéma est proposée, pour laquelle une augmentation en surface de 53% a été obtenue pour un exemple de partie contrôle généré par SYCO [REI 86], [JER 89].

L'augmentation de surface due à l'introduction du schéma UBIST dans une partie contrôle dépend de l'architecture adoptée pour celle-ci. En effet, une augmentation de surface inférieure à 30% a été estimée pour

l'introduction d'un schéma UBIST à une partie contrôle microprogrammée du type de celle du MC 68000 [NIC 90].

Par ailleurs, nous avons établit un schéma UBIST pour des parties opératives parallèles organisées en tranches de bit, du type de celle implémentée dans le MC 68000.

Une conception FSPD des différents éléments de la partie opérative est proposée, et l'introduction de contrôleurs double-rail et de parité a aboutit à son implémentation auto-contrôlable. Pour ce schéma, on a associé un schéma BIST qui teste les différents éléments de la partie opérative.

L'association du schéma BIST au schéma d'auto-contrôlabilité constitue le schéma UBIST de la partie opérative.

Les solutions proposées et développées montrent la faisabilité du schéma UBIST pour des cas réels d'architecture.

D'autre part, ces architectures ont été choisies pour leur régularité, qui permettrait leur génération par le compilateur SYCO. On a donc tracé la stratégie d'introduction du schéma UBIST dans le compilateur SYCO.

L'un des points essentiels de cette thèse est qu'elle s'appuie sur des réalisations de certains outils CAO de test, compatibles avec les spécifications architecturales et structurelles utilisées par SYCO, conduisant à la transformation de SYCO en un compilateur de circuits autotestables.

Même si une réalisation totale n'a pu être obtenue, cette étude nous a permis d'avoir une vue plus claire et précise des problèmes posés par une telle réalisation.

On peut remarquer que les outils de CAO de testabilité à introduire dans SYCO, agissent sur des spécifications architecturales en ce qui concerne les parties contrôle, et structurelles pour les parties opératives.

En effet, en ce qui concerne les parties contrôle générés par SYCO, on agit sur une partie de leur fonctionnalité en modifiant les PLAs, en augmentant les bus et le champs de registres de variables de contrôle, en modifiant les blocs de synchronisation etc...

Par contre, pour les parties opératives les modifications sont plutôt structurelles, on utilise une bibliothèque de cellules autotestables, on introduit des contrôleurs double-rail et de parité, le générateur de vecteurs de test, l'analyseur de signature etc...

Une architecture cible simple favorise le développement d'une stratégie de compilation en vue d'un schéma UBIST.

La complexité de l'architecture cible déterminera le niveau d'abstraction sur lequel le compilateur de silicium et les outils de CAO de test, vont agir pour générer le schéma UBIST pour cette architecture.

# REFERENCES

**[ABO 84]** ABOULHAMID M., CERNY E., "Built-In Testing of One-Dimensional Unilateral Iterative Arrays", IEEE Trans. Computers , Vol. C-33, Juin 1984, pp. 560-564.

**[AGR 84]** AGRAWAL V. D. & al. "Automation in design for testability", Proc. Custum Integerated Circuits Conf., Rochester, New York, May 1984, pp. 159-163.

**[AMB 86]** AMBLER A. P., "Economically viable automatic insertion of self-test features for costom VLSI" Intl. Test Conf. 86, pp. 232-243, 1986.

**[ANC 83]** ANCEAU F. "CAPRI : A design methodology and a silicon compiler for VLSI circuits specified by algorithms" 3ième Caltech Conference on VLSI, Mars 1983.

**[AND 71]** ANDERSON D. A. "Design of self-checking digital networks using coding techniques" Urbana, CSL Univ. of Illinois, Sept. 1971 (rapport 527).

**[BAR 79]** BARANOV S. "Synthèse des automates microprogrammés", traduit du russe par Boris PAVLOV, Edition MIR, Moscou 1979.

**[BEE 89]** BEENKER F. & all. "A testability strategy for silicon compilers" Intl. Test Conf., pp. 660-669, 1989.

**[BER 61]** BERGER J. M. "A note on error detection codes for asymetric binary channels", Inform. Contr., vol. 4, pp. 68-73, Mars 1961.

**[BLA 81]** BLANQUART J. P. "Validation du matériel : Le test hors-ligne", Notes thechniques LASS-ASF 81.T.17, 1981.

**[BRA 82]** BRAYTON R., HACHTEL G.D., HEMACHANDRA L., NEWTON A.R. and SANGIVANNI-VINCENTELLI A.L. "A comparison of logic minimization strategies using ESPRESSO. An APL program package for partitioned logic minimization" Proc. Int. Symp. on Cir. and Syst. Rome 1982.

**[CAR 68]** CARTER W.C., SCHNEIDER P.R. "Design of dynamically checkers", in Pro. 4th Congress IFIP, vol.2, Edinburgh, Scotland, Août 1968, pp. 878-883.

**[CAR 82]** CARTER W.C., "Signature testing with guaranteed bounds for fault coverage", Proc 1982 IEEE Int'l Test Conf., pp. 775-782, Cherry Hill, Pensylvanie, Nov. 1982.

**[CAT 89]** CATTHOOR F. & all., "A testability strategy for multiprocessor architecture" IEEE Design & Test of Comp., Vol 6, N° 2, Avril 1989.

**[CER 88]** CERNY E., ABOULHAMID M., BOIS G., CLOUTIER G., "Built-in Self-Test of a CMOS ALU" IEEE Design & Test of Computers, Août 1988.

**[CHA 82]** CHANDRAMOULI R. "Design VLSI chips for testability", Electronics Test, Nov 1982.

**[CHE 85]** CHEN C. Y., FUCHS W. K., ABRAHAM J. A., "Efficient concurrent error detection in PLAs and ROMs", ICCD'85, Port Chester, New York, Oct. 1985.

**[CLA 77]** CLAVIER J. & al. "Théorie et technique de la transmission de données" Masson 1977.

**[COU 81]** COURTOIS B., "Test et LSI", Thèse de Doctorat d'Etat ès Sciences USMG-INPG, Juin 1981.

**[COU 81a]** B. COURTOIS, "Failure mecanism, fault hypotheses, and analytical testing of LSI.NMOS (HMOS) circuits", VLSI 81, Univ. of Edinburg, Août 1981.

**[CRO 78]** CROUZET Y., "Conception de circuits à large échelle d'intégration totalement autotestables" Thèse de Docteur-Ingénieur Toulouse, Nov. 1978.

**[CRO 80]** CROUZET Y. & LANDRAULT C. "Design of self-checking MOS LSI circuits, application to a four-bit microprocessor" IEEE Trans. on Comp., pp. 532-537 Juin 1980.

**[DAE 81]** DAEHN W., MUCHA J., " A hardware approach to self-testing of large programmable logic arrays", IEEE Trans. on Cir. and syst., C-30, pp. 829-833, Nov. 1981.

**[DAR 89]** DARLAY F., "Contribution au test des circuits intégrés CMOS : Etude du test des pannes stuck-on et stuck-open" Thèse de Doct. INPG, Nov. 1989.

**[DAS 90]** DAS A.K. SAHA D., CHOWDHURY A.R., MISRA S. & CHAUDHURI P.P. "Signature analysers based on additive cellular automata" Proc. FTCS 20, New Castel, juin 1990.

**[DAV 78]** DAVID R. THEVENOD-FOSSE P. "Design of totally self-checking asynchronous modular circuits", Journal of Design Auto. and Fault-Tolerant Comp. Vol2, Oct. 1978.

**[DAV 80]** DAVID R., "Testing by feedback shift register", IEEE Trans. on Comp. Vol. C-29, N°. 7, pp. 668-673, Juillet 1980.

**[DAV 84]** DAVID R., "Signature analysis of multi-output circuits" Proc. FTCS 14, Kissimmee, juin 1984.

**[DON 82]** DONG H. "Modified Berger codes for detection of unidirectional errors" FTCS-12, Sta. Monica, Juin 1982.

**[ELS 59]** ELSPAS B. "Theory of autonomous linear sequential networks" IRE Trans. on Circuit Theory, Vol. CT6, Mars 1959, p 45.

**[FER 88]** FERNANDES A. O., "Le test des PLAs optimisés topologiquement", Thèse de Doct. INPG, Sept. 1988.

**[FRI 71]** FRIEDMAN A.D. & MEMON P. R. "Faut detection in digital circuits" Prentice Hall Inc. 1971.

**[FRI 73]** FRIEDMAN A.D., "Easily testable iterative systems", IEEE Trans. Comput. , vol C-22, pp 1061-1064, Déc. 1973.

**[FUC 87]** FUCHS W. K., CHIEN Ch. Y., and ABRAHAM J. A., "Concurrent error detection in highly structured logic arrays" IEEE J. Solid-State Circuits, vol. SC-22, No. 4, Août 1987.

**[FUJ 80]** FUJIWARA H., KINOSHITA K., OSAKI H. , "Universal test sets for programmable logic arrays" Proc 10th Int. CONF. Fault-Tolerant Computing.

**[FUJ 80]** FUJIWARA H. "Logic testing and design for testability", Computer systems, MIT Press, 1985.

**[FUN 85]** FUNG H. S., "A testable-by-construction strategy for the SILC silicon compiler", ICCD'85, Port Chester, New York, Oct. 1985.

**[FUN 86]** FUNG H. S., HIRSCHHORN S., "An automatic DFT system for the SILC silicon compiler", IEEE Design and Test of Comp., Fév. 1986.

**[GAL 80]** GALLIAY J., CROUZET Y. & VERGNIAULT M., "Physical versus logical fault models MOS LSI circuits : impact on their testability IEEE Trans. on Comp. Vol C-29, N° 6 Juin 1980.

**[GER 87]** GERONIMI J. P. & JERRAYA A. A., "Architecture des parties contrôles générées par SYCO : GENTOPO", Rapport interne TIM3, Oct. 1987.

**[GER 88]** GERONIMI J. P. "Génération automatique des parties contrôles" Rapport interne TIM3, 1988.

**[GER 88a]** GERONIMI J. P. "Compilation de parties opératives" Rapport de DEA informatique ENSIMAG, 1988.

**[GUY 88]** C. G. GUY & al., "An automatic concurrent error detecting tool for PLA generation", Microelectron. Reliab., vol. 28, No. 3, 1988, pp. 395-405.

**[HAL 84]** HALBERT M.P, BOSE S.M. "Design approach for a VLSI self-checking MIL-STD-1750 microprocessor" FTCS-14, Kissemmee,1984.

**[HAS 83]** HASSAN S. Z. & McCLUSKEY E. J. "Testing PLAs using multiple parallel analyzers" FTCS-13, pp.422-425, Juin 1983.

**[HUI 88]** HUISKENS J. & al., "Design of DSP systems using the PIRAMID library and design tools", MCNC Logic Synthesis Workshop, Grenoble, 1988.

**[JAM 85]** JAMIER R. & JERRAYA A.A. "APOLLON: a datapath compiler" ICCD'85, Port Chester, New York, Oct. 1985.

**[JAM 86]** JAMIER R., "Génération automatique de parties opératives de circuits VLSI de type microprocesseur" Thèse de Doct. Ing. INPG, Nov. 1986.

**[JAN 85]** JANSCH SCHREIBER I. E., "Conception de contrôleurs autotestables pour des hypothèses de pannes analytiques" Thèse Doct. Ing. INPG, Janv. 1985.

**[JER 86]** JERRAYA A.A. & al. "Principles of the SYCO compiler", 23rd Design Automation Conference, Juin 1986.

**[JER 88]** JERRAYA A.A., MHAYA N., GERONIMI J.P. & COURTOIS B. "SYCO: A Silicon Compiler For VLSI ASICs Specified By Algorithms" Computer-Aided Engineering Journal IEE, Juin 1988.

**[JER 89]** JERRAYA A.A., "Contribution à la compilation de silicium et au compilateur SYCO" Thèse de Doct. ès Sciences (Informatique) UJF-INPG, Déc. 1989.

**[JHA 89]** JHA N. K. "Fault detection in CVS parity trees: Application to SSC CVS parity and two-rail checkers" FTCS-19, Juin 1989, Chicago, USA.

**[JOH 79]** JOHANNNSEN D., "Bristles blocs, a silicon compiler", 16th DAC, 1979.

**[KHA 82]** KHAKBAZ J. & McCLUSKEY E. J. "Concurrent error detection and testing for large PLA's", IEEE Trans. Electr. Devices, vol. ED-29, pp 756-764, Avril 1982.

**[KOE 79]** B. KOENMANN, J. MUCHA, and G. ZWIHOFF, "Built-in logic block observation techniques", Proc 1979 IEEE Int'l Test Conf., pp. 37-41, Cherry-Hill, New Jersey, Oct. 1979.

**[MAK 82]** MAK G.P., ABRAHAM J.A., DAVINDSON E.S. "The design of PLAs with concurrent error detection" FTCS-12 Santa-Monica, Juin 1982.

**[MAR 78]** MAROUF M.A., FRIEDMAN A.D. "Design of self-checking checkers for Berger code" FTCS-8, Toulouse, Juin 1978.

**[MAR 82]** MARINESCU M., "Simple and efficient algorithms for functional RAM testing", IEEE International Test Conference, Novembre 1982.

**[MAR 84]** MARCHAL P., NICOLAIDIS M., COURTOIS B., "Microarchitecture of the MC 68000 and evaluation of a self-checking version", in Microarchitecture of VLSI Computers, Martinus Nijhoff Publishers, 1985.

**[McC 86]** Mc CLUSKEY E. J., "Logic design principles: with emphasis on testable semicustom circuits", Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[MEA 80] MEAD C., and CONWAY L. "Introduction To VLSI Systems" Addison-Wesley Publ. Co., 1980.

[MHA 87] MHAYA N. & JERRAYA A. A., "CPC : The SYCO control section compiler" ESSIRC'87.

[NAI 78] NAIR R., THATTE S.M., ABRAHAM J.A., "Efficient Algorithms for Testing Semiconductor Random-Access Memories", IEEE Transactions on computers, Vol. C-27, pp. 572-576, Juin 1978.

[NAN 88] NANYA T., MOURAD S., McCLUSKEY E.J. "Multiple stuck-at fault testability of self-testing checkers", FTCS-18, Tokyo, Juin 1988.

[NIC 83] NICOLAIDIS M., COURTOIS B., "Design of self-checking systems based on analytical fault hypotheses", IMAG Report RR353, 1983.

[NIC 84a] NICOLAIDIS M., "Conception de circuits intégrés autotestables pour des hypothèses de pannes analytiques", Thèse Docteur-Ingénieur, INPG 1984.

[NIC 84b] NICOLAIDIS M., JANSCH I., COURTOIS B. "Strongly Code Disjoint checkers", Proc. of FTCS-14, Kissemmee, Juin 20-22 1984. Aussi dans IEEE Trans. on Comp., Juin 1988.

[NIC 85] NICOLAIDIS M., "Evaluation of a Self-Checking version of the MC68000 Microprocessor", Proc. 15th Fault Tolerant Computing Symposium, Ann Arbor, USA, Juin 1985.

[NIC 85a] NICOLAIDIS M., "An efficient Built-in Self-Test scheme for functional test of RAMS" Proc. 15th Fault Tolerant Computing Symposium, Ann Arbor, USA, Juin 1985.

[NIC 85b] NICOLAIDIS M., COURTOIS B., "Layout rules for the design of self-checking circuits". VLSI Conference, Août 85, Tokyo.

[NIC 86] NICOLAIDIS M., COURTOIS B. "Design of Self-Checking Circuits Using Unidirectional Error Detecting Codes". In Proc. of the 16th FTCS Vienna, Juillet 1986, Austria.

[NIC 87] NICOLAIDIS M., "Shorts in Self-Checking circuits", Proc. Int. Test Conf., Washington, D.C., Sept. 1987, pp. 408-417.

[NIC 88] NICOLAIDIS M. "A Unified Built-In Self-Test scheme: UBIST", FTCS-18, Tokyo, Juin 1988, paru aussi dans IEEE Trans. on CAD, vol. 8, N°3, pp 203-218, Mars 1989.

[NIC 89a] NICOLAIDIS M., COURTOIS B. "Self-Checking Logic Arrays" Microprocessors & Microsystems, Butterworth Scientific Ldt. England, Mai 1989.

[NIC 89b] NICOLAIDIS M., "Test pattern generators for arithmetic units and arithmetic and logic units", Rapport interne TIM3, Janvier 1990.

[NIC 89c] NICOLAIDIS M., TORKI K., "Implementation of testing properties within a datapath compiler", Rapport interne TIM3, Novembre 1989.

[NIC 90] NICOLAIDIS M., "Efficient UBIST implementation for microprocessor sequencing parts", Proc. Int. Test Conf., Washington D.C., USA, Sept. 1990.

[OBR 82] OBREBSKA M. "Etude comparative de différentes méthodes de conception des parties contrôle des microprocesseurs" Thèse de Docteur-Ingénieur, INPG, Juin 1982.

[OKL 84] OKLOBDZIJA V., KOVIJANIC P., "On Testability of CMOS-Domino Logic", Proc. Fault Tolerant Computing Symp., Juin 1984, pp. 50-55.

**[OSS 86]** OSSEIRAN A. & all., "Design of a self-checking microprocessor for real time applications" Proc. IFAC, 5th Conf. of Security in Transportation Systems, Vienna, Austria, Juil. 1986.

**[PIE 85]** PIESTRAK S. "Design methods of totally self-checking checkers for m-out-of-n codes",Proc. 13th FTCS, Milan, Italy, Juin 1983, pp. 162-168.

**[PLA 87]** "FSM benchmarks" Microelectronics Center of North Carolina, 1987.

**[REI 86]** REIS R., JERRAYA A. A. and JAMIER R. "Design of the SYCO6502 using SYCO silicon compiler", ICCD'86, Port Chester, New york, Oct. 1986.

**[SAB 86]** SABO D. G., JOHANNSEN D., YAU R., "GENESIL silicon compilation" IEEE Custom Integrated Circ. Conf., 1986.

**[SAY 85]** SAYERS I. L., RUSSEL G., & KINNIMENT D. J. "Concurrent checking techniques-A DFT alternative" in Developments in Integrated Circuit Testing, Academic Press, 1987, pp. 359-389.

**[SEG 79]** PEREZ SEGOVIA T., "Etude de la conception et de la minimisation des PLAs, dessin des PLAs du NOM 400", Rapport interne TIM3, Sept. 1979.

**[SER 88]** SERRA M. "Some experiments on the overhead for concurrent checking" 3rd Tech. Workshop, Halifax, Canada, Oct. 1988.

**[SMI 77]** SMITH J.E. & METZE G. "The design of totally self-checking combinatorial circuits", Proc. 7th FTCS, Los Angeles, USA, Juin 1977.

**[SMI 78]** SMITH J.E. & METZE G. "Strongly Fault Secure logic networks", IEEE Trans. on Comp. Vol. C-27 N°6, Juin 1978.

**[SMI 80]** SMITH J.E "Measure of the effectiveness of fault signature analysis", IEEE Trans. on Comp. Vol. C-29, N° 6, pp. 442-451, Juin 1980.

**[SMI 84]** SMITH J.E., "On separable unordered codes", IEEE Trans. Computers, Vol. C-33, N°8, Août 1984, pp. 741-743.

**[SRI 81]** SRIDHAR T., HAYES J.P., "Design of Easily Testable Bit-Sliced Systems", IEEE Trans. Computers, Vol. C-30, Nov. 1981, pp. 842-854.

**[SUK 81]** SUK D.S., REDDY S.M., "A march test for functional faults in semiconductor random access memories", IEEE transaction on computers, Déc. 1981.

**[TAO 86]** TAO D. L., LALA P.K., and HARTMANN C. R. P. "A Concurrent testing strategy for PLAs" in Proc. 1986 Int. Test Conf., Washington, D. C., Sept. 1986, pp. 705-709.

**[TAM 84]** TAMIR Y. & SEQUIN C. H., "Design and application of self testing comparators implemented with MOS PLAs" IEEE Trans. on Comp. Vol. C33, Juin 1984.

**[THO 77]** THOMAS D. E. "Automating technology relative logic synthesis and module selection", IEEE Trans. on CAD, Vol 2, N° 2, Avril 1983.

**[THO 87]** THOREL P., DAVID R., PULOU J., RAIWARD J.L., "Design for Random Testability", International Test Conference, Washington D.C., Sept. 1987, USA.

**[TOR 86]** TORKI K. "Etude de l'architecture de parties contrôle autotestables pour le compilateur de silicium SYCO (Compilation de circuits autotestables)" Rapport de DEA, USTMG-INPG, Juin 1986.

**[TOR 87]** TORKI K. & NICOLAIDIS M. "Unification of the design of Self-Checking and BIST control section organized around unfolded and/or folded PLAs", Rapport interne, INPG/TIM3, Juin 1987.

**[TOR 88a]** TORKI K. "PROTECT : a PLA coding tool for on-line built-in test" Rapport interne, INPG/TIM3, Déc. 1988.

**[TOR 88b]** TORKI K., NICOLAIDIS M., JERRAYA A. A., COURTOIS B. "UBIST version of the SYCO's control section compiler", IEEE 1988 International Conference on Computer Design, ICCD'88, Rye Brook, New York, Oct. 1988.

**[TOR 89]** TORKI K. "Implementation of testing properties within a control section compiler: Experimental software" Rapport interne, INPG/TIM3, Juin 1989.

**[TOR 91]** TORKI K., NICOLAIDIS M., FERNANDES A. O., "A Self-Checking PLA Automatic Generator Tool Based On Unordered Codes Encoding", A paraître à EDAC91, Amsterdam, Pays-Bas, Février 1991.

**[WAD 78]** WADSACK R.L., "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits", The Bell Sys. Tech. Jour., Vol. 57, Mai-Juin 1978, pp. 1449-1474.

**[WAN 86]** WANG L.T. & E.J. McCLUSKEY "Circuits for Pseudo-Exhaustive test pattern generation", ITC-86, Washington, Sept. 1986.

**[WAN 79]** WANG S.L. & AVIZIENIS A. "The design of Totally Self-Checking circuits using programmable logic arrays", FTCS-9, Madison, Juin 1979.

**[WAT 62]** WATSON E.J., "Primitive polynomials (Mod-2)", Mathematics of computation, pp. 368-369, 1962.

**[WEI 85]** WEI R.S. and SANGIOVANNI-VINCENTELLI A., "PLATYPUS: A PLA Test Pattern Generation Tool", Proc. of the 22nd Design Automation Conference DAC, 1985 IEEE, Juin 23-26 1985, Las Vegas, Nevada.

**[WES 85]** WESTE N.H.E. and ESHRAGHIAN K., "Principles Of CMOS VLSI Design: A system perspective" Addison-Wesley Publ. Co., 1985.

**[WIL 86]** WILLIAMS T. W., DAEHN N. W., GRUETZNER M., STARKE C. W., "Comparison of aliasing errors for primitive and non-primitive polynomials", Proc. ITC, 1986, pp. 282-288.

# ANNEXES

**Annexe1 : Le test des circuits Self-Checking à logique CMOS dynamique.**

Les pannes latentes provoquant des accumulations de pannes dans les circuits Self-Checking peuvent rendre non garantie la propriété Fault-Secure.

Pour éviter ces pannes il est supposé que les pannes surviennent une par une, et entre l'occurrence de 2 pannes, il s'écoule un temps suffisamment long pour que le circuit reçoive pendant le fonctionnement normal tous les vecteurs du code d'entrée (hypothèse H2).

Cette hypothèse est simplifiée lorsqu'on considère un ensemble particulier de pannes (e. g. collage), auquel cas le circuit doit recevoir des vecteurs de test assurant la couverture de panne de collage. Aucune restriction quand à l'ordre d'application des vecteurs de test n'est considérée dans ce cas.

Par contre, la détection de certaines pannes affectant des circuits CMOS demande l'application de séquences de couple de vecteurs.

On doit examiner si des restrictions sont à considérer ou pas, concernant les séquences de test appliquées à des circuits Self-Checking implémentés en logique CMOS à précharge.

En figure X a) est présentée la structure d'une porte CMOS à précharge [WES 85]. De telles portes ont une structure très différente d'une logique FCMOS ou CMOS hybride (figure X b)). En effet, la testabilité de telles portes est différente. Un MOS stuck-open, par exemple, dans le réseau n de la figure X b) demanderait l'application de 2 vecteurs de test (t1, t2) pour être détecté [WAD 78] [DAR 89]. Quand le vecteur t2 détectant la panne s-open est appliqué, la sortie de la porte est en état de haute impédance (dû à la panne s-open) conservant ainsi sa valeur initiale. Si le vecteur t1 connecte la sortie de la porte au Vss (ce qui est possible puisque le réseau n peut connecter la sortie au Vss à travers des chemins ne passant pas par le MOS s-open), la sortie de la porte est maintenue à la valeur 0 lorsque t2 est appliqué, ce qui correspond à une réponse correcte. Ce qu'il faut faire alors, c'est initialiser la sortie de la porte à 1 par un vecteur t1.

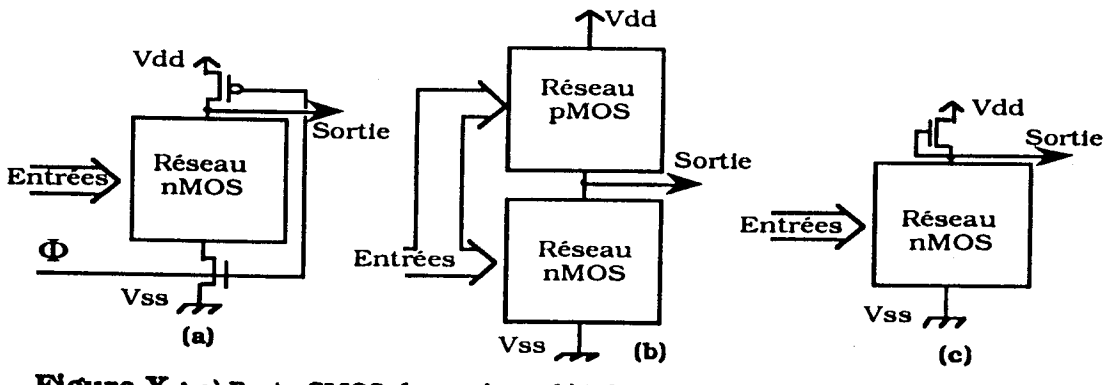



**Figure X : a) Porte CMOS dynamique, b) FCMOS et CMOS hybride, c) nMOS .**

Par opposition, pour une porte implémentée sous forme domino la sortie de la porte est systématiquement préchargée à 0 par t1, durant la phase de précharge, pour que t2 détecte la panne s-open indépendamment de la séquence précédente.

Une autre différence concerne la testabilité de pannes s-on. Dans le cas du réseau n de la figure X b), une panne s-on peut donner des niveaux intermédiaires, puisque quand le vecteur de test est appliqué, la porte se transforme en un diviseur de tension connectant simultanément la sortie de la porte au Vdd et au Vss à travers respectivement le réseau p et le réseau n. Ces pannes ne sont donc pas détectables par des techniques de test à collage logique.

Par contre, quand le vecteur de test est appliqué, une panne s-on dans le réseau n de la figure X a) donnera en sortie le niveau logique 0, puisque durant la phase d'évaluation le transistor MOS p est déconnecté.

Par conséquent, la porte de la figure X a), manifeste les mêmes comportements de testabilité des pannes de s-on et s-open de transistors MOS dans le réseau n, qu'une implémentation nMOS de la figure X c).

Un cas particulier est celui du transistor MOS p de précharge. Pour détecter les pannes de s-open de ce transistor, on doit appliquer sur les entrées de la porte, un vecteur d'entrée pour lequel la sortie prend la valeur "1". Quand ce vecteur est appliqué, la sortie de la porte est en état de haute impédance (le comportement est similaire au cas d'une implémentation nMOS pour laquelle la connexion avec le Vdd est coupée).

Il résulte que les circuits Self-Checking implémentés en logique CMOS à précharge (et même pour une logique nMOS) demande à être exercisé régulièrement par 2 séquences de test. Mais la situation est différente de celle des logiques FCMOS et CMOS hybride.

Dans de telles portes, pour un transistor p s-open (resp. transistor n s-open), plusieurs chemins peuvent exister connectant la sortie de la porte au Vdd (resp. Vss) et ne passant pas par le MOS s-open de façon à ce que juste avant l'application du vecteur de test détectant la panne s-open, la sortie de la porte peut être initialisée à la valeur correcte.

D'autre part, dans le cas d'une porte dynamique, la sortie de la porte est déconnectée indéfiniment du Vdd. Il en résulte que les hypothèses standard utilisées pour des circuits Self-Checking à logique nMOS sont aussi valables et suffisants à des circuits Self-Checking à logique CMOS à précharge.

En fait, il est garanti que quelque temps après l'occurrence d'une panne s-open, un vecteur pour lequel la sortie de la porte prend la valeur 0 survient (sinon le circuit Self-Checking n'aurait pas été exercisé pour un collage à 1 de la sortie). Après un tel vecteur, la sortie reste à 0 jusqu'à l'apparition d'un

vecteur détectant le collage à 0 de la sortie.

Ce vecteur détectera donc la panne s-open et son occurrence est garantie si une génération standard de vecteurs de test est effectuée. Les hypothèses standards concernant la génération de vecteurs de test sont donc suffisantes pour exerciser des circuits Self-Checking implémentés en logique CMOS à précharge (comme pour une logique nMOS).

Un autre cas concerne les pannes s-on affectant un transistor de précharge p ou n. Ces pannes induisent des valeurs de sortie à un niveau indéterminé, leur détection n'est pas garantie.

Si l'on veut détecter ces pannes, il faut que leur occurrence produise des valeurs logiques à la sortie de la porte.

Si la sortie d'une porte prend la valeur logique 0 (resp. 1) quand les deux réseaux n et p conduisent en même temps, la porte est dite n-dominante (resp. p-dominante) [OKL 84].

Ceci devient possible en calculant les tailles des transistors, comme dans le cas d'une logique pseudo-nMOS (figure Y).

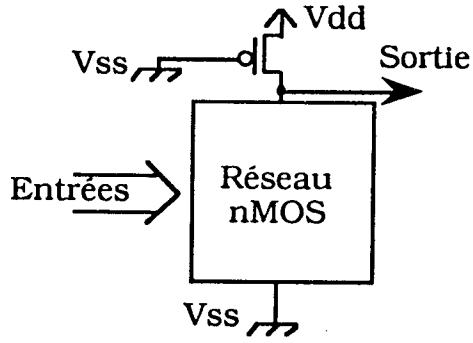



**Figure Y : Porte pseudo-nMOS.**

Si l'on considère que les portes sont p-dominantes. Il suffit d'augmenter la largeur de canal du transistor de précharge p de la porte en figure X a). Alors qu'une conception n-dominante de cette porte demanderait à augmenter la largeur de canal des transistors n.

Une conception p-dominante est préférable à une n-dominante, en effet pour une porte n-dominante on doit tailler plusieurs transistors n, et la détectabilité d'un s-on du transistor p de précharge dépendrait du vecteur d'entrée durant la phase de précharge.

Finalement, les courts-circuits entre sorties de portes dynamiques sont discutées dans [NIC 87], ils ont les mêmes conséquences que ceux affectant les portes nMOS.

**Annexe 2 : PLAs implémentés en logique CMOS dynamique :**

Dans [MAK 82] sont présentés des implémentations nMOS de PLAs ayant leurs sorties codées en codes non ordonnés. Ces PLAs sont SFS vis à vis de pannes simples de collage, de transistor MOS de croisement manquant ou parasite, et de courts-circuits.

D'autre part dans [NIC 87], sont présentées des règles de conception de PLAs assurant la propriété SFS pour les pannes ci-dessus, plus des pannes s-on et s-open. Ces règles concernent une technologie nMOS, mais leur extension à une technologie CMOS à précharge est directe.

Toute panne simple, de l'ensemble décrit ci-dessus, survenant dans un PLA produit une erreur simple sur une ligne d'entrée, ou une ligne de monôme, ou une ligne de sortie. Dans un PLA, tous les chemins reliant une ligne (ligne d'entrée, de monôme, ou de sortie) et une ligne de sortie a la même parité d'inversion. Les erreurs simples précédentes sont propagées en erreurs unidirectionnelles aux sorties du PLA.

Une panne de collage du signal de précharge dans la matrice ET (resp. matrice OU) produit des erreurs multiples sur les lignes de monôme (rep. les lignes de sortie).

On peut avoir les quatre situations suivantes :

• Collage à 0 du signal de précharge dans la matrice ET :

Cette panne met à 1 toutes les lignes de monôme, toutes les erreurs sont du type $0\rightarrow1$. Ces erreurs unidirectionnelles sur les lignes de monôme sont propagées aux sorties du PLA en erreurs unidirectionnelles du type $1\rightarrow0$. Ceci est dû au fait que la parité d'inversion entre une ligne de monôme et une ligne de sortie est impaire.

• Collage à 1 du signal de précharge dans la matrice OU :

Cette panne empêche la précharge des lignes de monôme du PLA, il en résulte que toutes les lignes de monôme ont une erreur du type $1\rightarrow0$. Il en résulte, comme précédemment, que les erreurs sont propagées aux sorties du PLA en erreurs unidirectionnelles du type $0\rightarrow1$.

• Collage à 0 du signal de précharge dans la matrice ET :

Cette panne engendre des erreurs unidirectionnelles du type $0\rightarrow1$ à toutes les lignes de sortie du PLA.

• Collage à 1 du signal de précharge dans la matrice OU :

Cette panne engendre des erreurs unidirectionnelles du type $1\rightarrow0$ à toutes les lignes de sortie du PLA.

Le PLA est ainsi muni de la propriété Fault-Secure. En annexe 1 on a étudié la détectabilité de pannes dans les portes CMOS dynamiques et on a vu qu'elle était la même que pour une logique nMOS. La propriété SFS est donc assurée pour un PLA CMOS dynamique de la même façon qu'elle l'est pour un PLA nMOS.

**Annexe 3 : Testabilité de contrôleurs double-rail CVSL dynamiques :**

L'étude présentée en annexe 1 peut être étendue à une logique CMOS domino représentée en figure Z [WES 85].

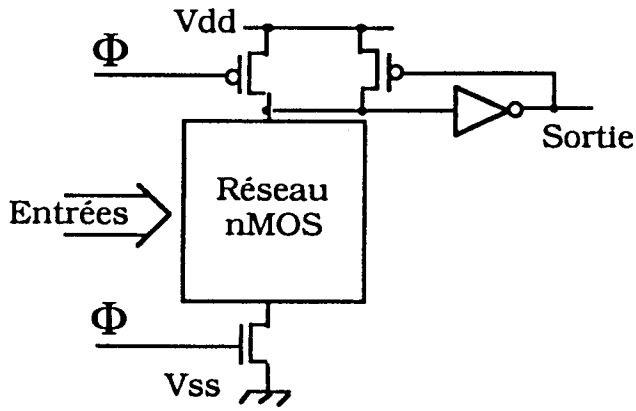



**Figure Z : Logique CMOS domino.**

En comparaison aux portes CMOS dynamiques (figure X a)) on a 3 transistors MOS supplémentaires dans une logique domino.

Un transistor p de rebouclage et un inverseur CMOS avec un transistor n et un transistor p. Comme en annexe 1 on considère que la porte est p-dominante.

L'inverseur peut tout aussi bien être p-dominant ou n-dominant, mais une implémentation sous forme n-dominante est plus efficace, elle sera donc adoptée.

Sous cette condition les pannes s-on des 3 MOS supplémentaires sont testables par un test à modèle de collage logique (pannes s-on du transistor p de rebouclage et le transistor n de l'inverseur) ou ne modifie pas les valeurs des sorties (pannes s-on du transistor p de l'inverseur).

Une panne s-open sur le transistor p de rebouclage ne modifie pas les valeurs de sortie.

Dans le cas de panne de s-open du transistor p de l'inverseur, la sortie est déconnectée en permanence du Vdd, et une panne s-open du transistor n de l'inverseur déconnecterait en permanence la sortie du Vss.

A travers les mêmes arguments présentés en annexe 1 concernant la testabilité des pannes s-open du MOS p de précharge, les hypothèses standard d'application de vecteurs de test à des circuits Self-Checking assurent la testabilité tantôt d'un collage à 1, tantôt d'un collage à 0 de la sortie. Ce qui signifie que la sortie change à certains moments de 1 à 0. Ce qui permet la détection de la panne s-open du transistor n de l'inverseur qui empêche une transition de la sortie de 1 à 0. De la même façon, une panne s-open du transistor p de l'inverseur est détectable.

En conclusion, la porte de la figure Z est suffisamment exercisée si elle

reçoit régulièrement un ensemble de vecteurs de test exercisant suffisamment la même porte implémentée en logique nMOS (i. e. la porte nMOS correspondante a le même réseau de décharge que le réseau n de la porte de la figure Z).

Il est facile de voir que ces résultats peuvent directement être appliqués au contrôleur double-rail de la figure W.

Plusieurs séquences de test existent ayant chacune 4 vecteurs de test et pouvant chacune exerciser l'implémentation nMOS du contrôleur. Elles peuvent être utilisées pour exerciser une implémentation CVSL de ce contrôleur.

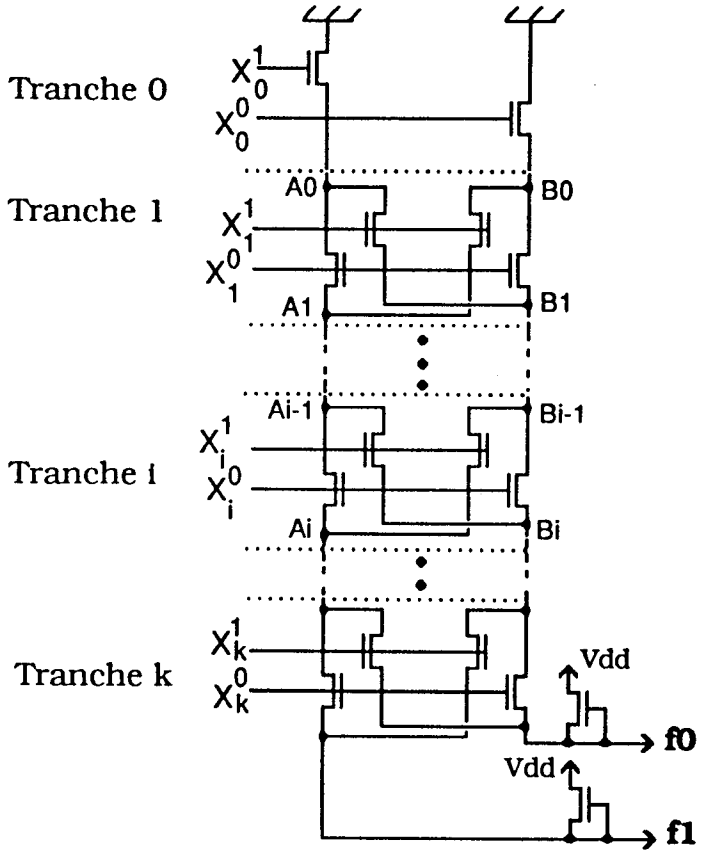



**Figure W : Implémentation CVSL du contrôleur double-rail à k variables.**

Remarque 1 : Si en figure W, les deux noeuds Ai et Bi de la tranche i sont connectés au (resp. déconnectés du) Vss, les sorties f0 f1 prennent toutes les deux la valeur 0 (resp. 1).

De cette remarque, on déduit que si une panne provoque la connexion ou la déconnexion du Vss des deux noeuds de sortie d'une tranche i, la panne est détectée.

L'état d'un noeud connecté au Vss sera noté 0 et l'état d'un noeud déconnecté du Vss sera noté H.

Remarque 2 : Les pannes simples de collage, s-on, et s-open dans la

tranche 0 de la figure W sont détectées si l'on applique aux entrées $X^0_0$ $X^1_0$ les vecteurs (1,0) et (0,1).

Remarque 3 : Les pannes simples de collage, s-on, et s-open dans la tranche i sont détectées si l'on applique sur les entrées ($X^0_0$ $X^1_0$), ($X^0_1$ $X^1_1$), ..., ($X^0_i$ $X^1_i$) 4 vecteurs (V1, V2, V3, V4) provoquant sur $X^0_i$, $X^1_i$, $A_{i-1}$, $B_{i-1}$ 4 états donnés en table I.

| | Ai-1 | Bi-1 | $Xi^0$ | $Xi^1$ | Ai | Bi |
|---|---|---|---|---|---|---|
| V1 | 0 | H | 0 | 1 | H | 0 |
| V2 | 0 | H | 1 | 0 | 0 | H |
| V3 | H | 0 | 0 | 1 | 0 | H |
| V4 | H | 0 | 1 | 0 | H | 0 |

**Table I : Les 4 états permettant le test de la tranche i.**

La preuve de cette remarque est évidente.

Il est facile de voir qu'il existe un seul ensemble de 4 vecteurs de test appliqués sur $X^0_0$, $X^1_0$, $X^0_1$, $X^1_1$ générant les 4 états de la table I et testant ainsi la tranche 1. Ces mots de code sont {0101, 0110, 1001, 1010}.

Si on considère les sorties $A_i$, $B_i$ de la tranche i de la table I, on trouve forcément que pour tester la tranche i+1 il y a 4 façons de combiner les vecteurs $V_1$, $V_2$, $V_3$, $V_4$ avec des mots du code $X^0_{i+1}$ $X^1_{i+1}$=01 et $X^0_{i+1}$ $X^1_{i+1}$=10. Ces combinaisons sont ($V_1$ 01, $V_2$ 01, $V_3$ 10, $V_4$ 10), ( $V_1$ 01, $V_2$ 10, $V_3$ 01, $V_4$ 10), ( $V_1$ 10, $V_2$ 01, $V_3$ 10, $V_4$ 01) et ( $V_1$ 10, $V_2$ 10, $V_3$ 01, $V_4$ 01). Ces combinaisons donnent en Ai, Bi, $X^0_{i+1}$, $X^1_{i+1}$ les 4 ensembles de 4 vecteurs suivants: (H001, 0H01, 0H10, H010), (H001, 0H10, 0H01, H010), (H010,0H01, 0H10, H001), (H010, 0H10, 0H01, H001). Chacun de ces ensembles assure la testabilité de la tranche i+1. De chacun des ensembles de vecteurs de test de la tranche i on obtient 4 ensembles de vecteurs de test testant la tranche i+1. Comme l'on a un seul ensemble de vecteurs de test de 4 mots du code testant les tranches 0 et 1, pour un contrôleur à n variables on aura $4^{n-2}$ différents ensembles de test de 4 mots du code, chacun de ces ensembles assure le test du contrôleur.

La figure R montre un contrôleur double-rail à 2 couches cascadant plusieurs petits contrôleurs. Si on considère un groupe composé de $\lambda$ ensembles de vecteurs de test, chacun de ces ensembles contenant 4 mots du code du contrôleur double-rail de la première couche. On a exactement $(4^{n_1}-2)x(4^{n_2}-2)x \ldots x(4^{n_\lambda}-2)$ de tels groupes.

Par une analyse similaire au cas d'une réalisation sur une couche, on peut montrer que les 4 ensembles de vecteurs du code d'un tel groupe peut être

combiné de $4^{2\lambda-2}$ différentes façons pour donner des ensembles pour le contrôleur double-rail à $\lambda$ variables. Par conséquent on a $(4^{n_1}- 2) \times (4^{n_2}- 2) \times .... \times (4^{n_\lambda}-2) \times 4^{2\lambda-2} = 4^{(n_1+n_2+....+n_\lambda-2)}$ ensembles de test de 4 mots du code, chacun assurant le test de tout le contrôleur.

Ce résultat peut être étendu à tout nombre de couches, de façon à ce que pour une réalisation d'un contrôleur double-rail à k variables on a $4^{k-2}$ ensembles de 4 mots du code de test. On conclu par le fait que le contrôleur est testable par un grand nombre d'ensembles de test de 4 mots du code impliquant une très bonne testabilité.

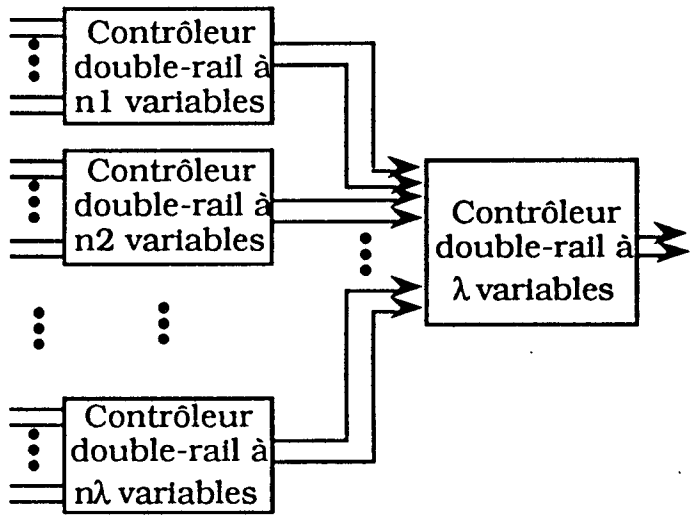
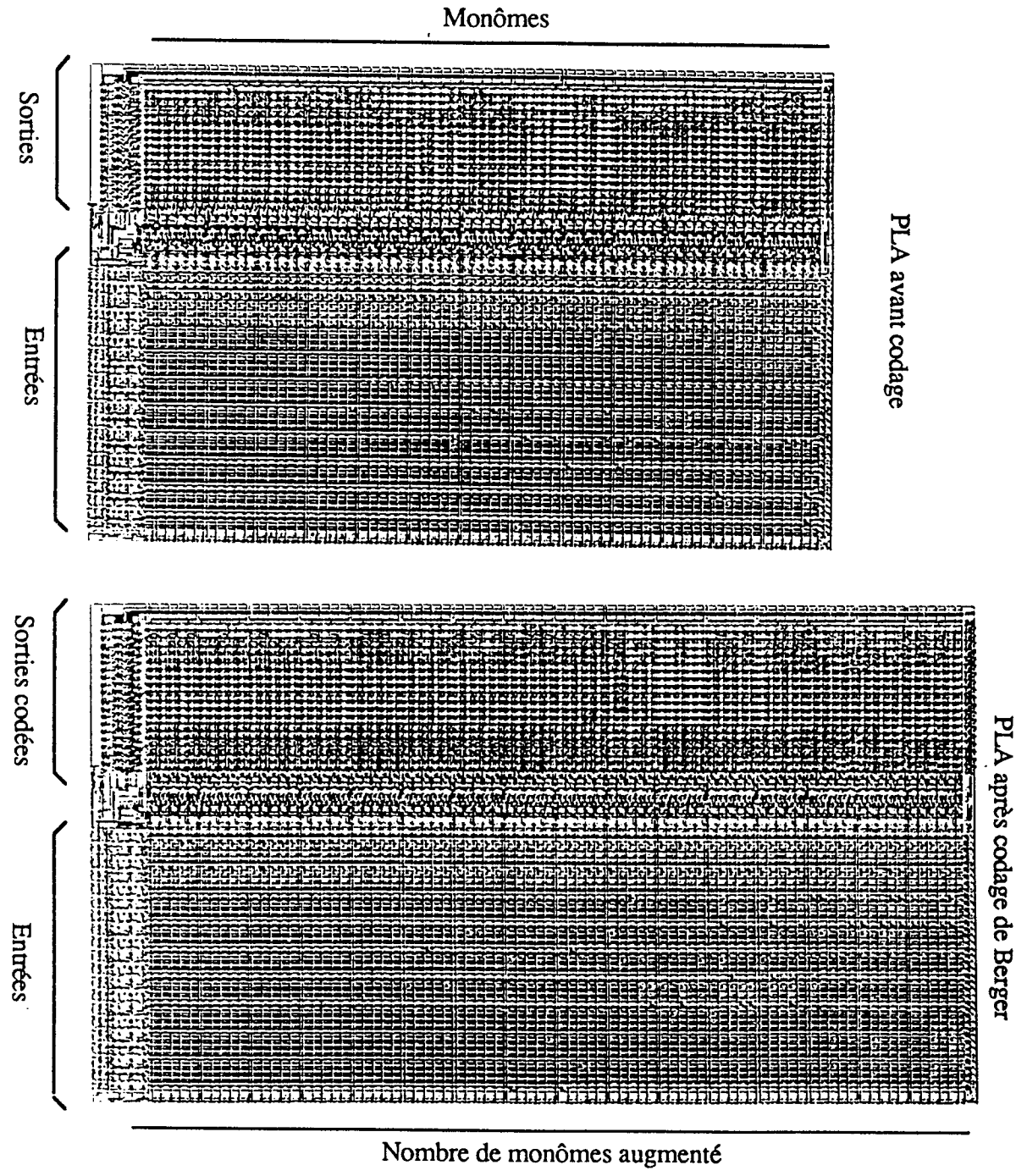



**Figure W : Contrôleurs double-rail en cascade.**

## Annexe 4 : Liste des polynômes primitifs de degré n≤100

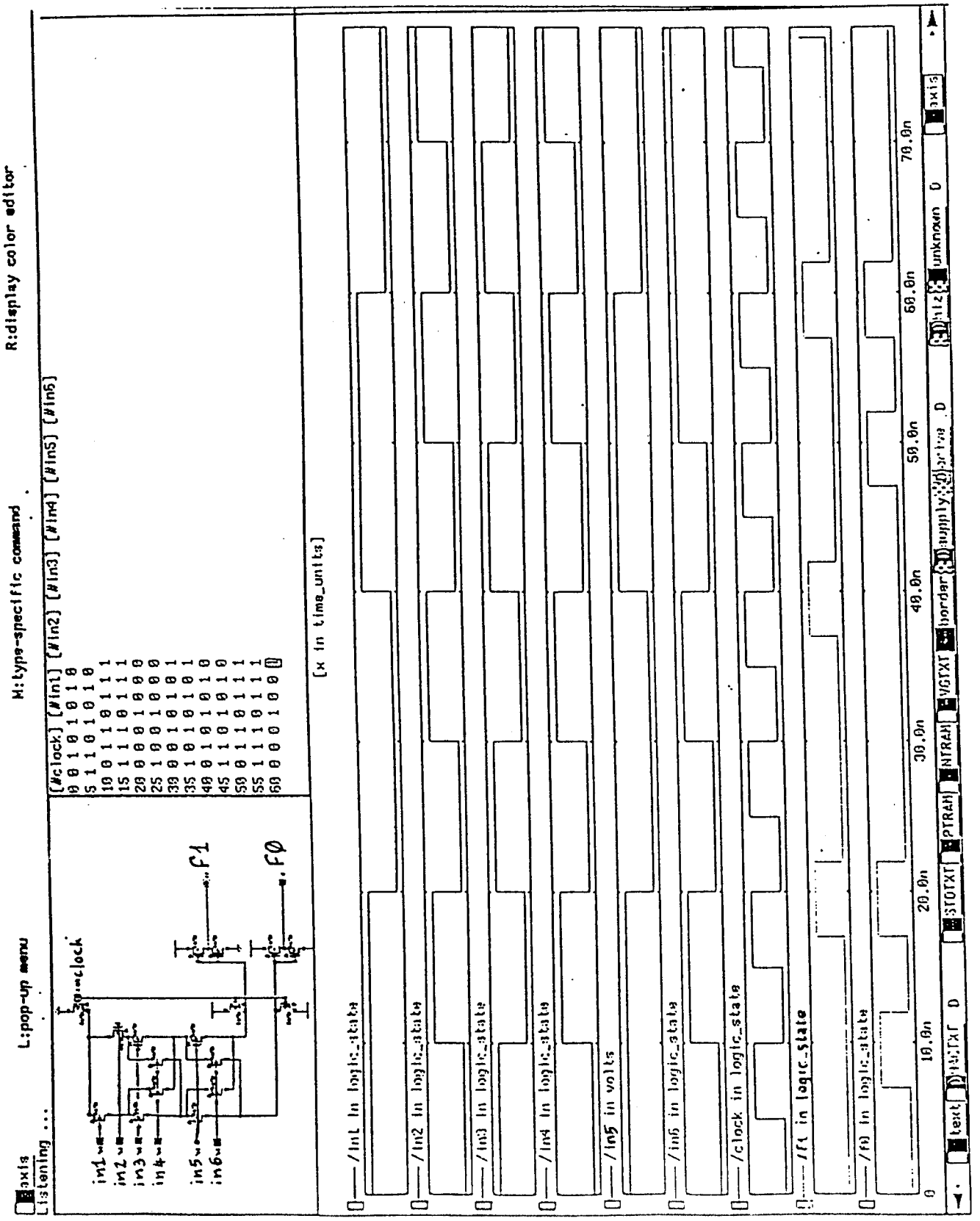
| n | f(x)=p(x) | n | f(x)=p(x) | n | f(x)=p(x) |
|---|---|---|---|---|---|
| 1 | $1+x$ | 35 | $1+x^2+x^{35}$ | 69 | $1+x^2+x^5+x^6+x^{69}$ |
| 2 | $1+x+x^2$ | 36 | $1+x+x^2+x^4+x^5+x^6+x^{36}$ | 70 | $1+x+x^3+x^5+x^{70}$ |
| 3 | $1+x+x^3$ | 37 | $1+x+x^2+x^3+x^4+x^5+x^{37}$ | 71 | $1+x+x^3+x^5+x^{71}$ |
| 4 | $1+x+x^4$ | 38 | $1+x+x^5+x^6+x^{38}$ | 72 | $1+x+x^2+x^3+x^4+x^6+x^{72}$ |
| 5 | $1+x^2+x^5$ | 39 | $1+x^4+x^{39}$ | 73 | $1+x^2+x^3+x^4+x^{73}$ |
| 6 | $1+x+x^6$ | 40 | $1+x^3+x^4+x^5+x^{40}$ | 74 | $1+x^3+x^4+x^7+x^{74}$ |
| 7 | $1+x^3+x^7$ | 41 | $1+x^3+x^{41}$ | 75 | $1+x+x^3+x^6+x^{75}$ |
| 8 | $1+x+x^2+x^7+x^8$ | 42 | $1+x+x^2+x^3+x^4+x^5+x^{42}$ | 76 | $1+x^2+x^4+x^5+x^{76}$ |
| 9 | $1+x^4+x^9$ | 43 | $1+x^3+x^4+x^6+x^{43}$ | 77 | $1+x^2+x^5+x^6+x^{77}$ |
| 10 | $1+x^3+x^{10}$ | 44 | $1+x^2+x^5+x^6+x^{44}$ | 78 | $1+x+x^2+x^7+x^{78}$ |
| 11 | $1+x^2+x^{11}$ | 45 | $1+x+x^3+x^4+x^{45}$ | 79 | $1+x^2+x^3+x^4+x^{79}$ |
| 12 | $1+x+x^5+x^8+x^{12}$ | 46 | $1+x+x^2+x^3+x^5+x^8+x^{46}$ | 80 | $1+x+x^2+x^3+x^5+x^7+x^{80}$ |
| 13 | $1+x+x^2+x^{12}+x^{13}$ | 47 | $1+x^5+x^{47}$ | 81 | $1+x^4+x^{81}$ |
| 14 | $1+x^2+x^3+x^{13}+x^{14}$ | 48 | $1+x+x^2+x^4+x^5+x^7+x^{48}$ | 82 | $1+x+x^4+x^6+x^7+x^8+x^{82}$ |
| 15 | $1+x+x^{15}$ | 49 | $1+x^4+x^5+x^6+x^{49}$ | 83 | $1+x^2+x^4+x^7+x^{83}$ |
| 16 | $1+x+x^7+x^{10}+x^{16}$ | 50 | $1+x^2+x^3+x^4+x^{50}$ | 84 | $1+x+x^3+x^5+x^7+x^8+x^{84}$ |
| 17 | $1+x^3+x^{17}$ | 51 | $1+x+x^3+x^6+x^{51}$ | 85 | $1+x+x^2+x^8+x^{85}$ |
| 18 | $1+x^7+x^{18}$ | 52 | $1+x^3+x^{52}$ | 86 | $1+x^2+x^5+x^6+x^{86}$ |
| 19 | $1+x+x^4+x^{16}+x^{19}$ | 53 | $1+x+x^2+x^6+x^{53}$ | 87 | $1+x+x^5+x^7+x^{87}$ |
| 20 | $1+x^3+x^{20}$ | 54 | $1+x^2+x^3+x^4+x^5+x^6+x^{54}$ | 88 | $1+x+x^3+x^4+x^5+x^8+x^{88}$ |
| 21 | $1+x^2+x^{21}$ | 55 | $1+x+x^2+x^6+x^{55}$ | 89 | $1+x^3+x^5+x^6+x^{89}$ |
| 22 | $1+x+x^{22}$ | 56 | $1+x^2+x^4+x^7+x^{56}$ | 90 | $1+x^2+x^3+x^5+x^{90}$ |
| 23 | $1+x^5+x^{23}$ | 57 | $1+x^2+x^3+x^5+x^{57}$ | 91 | $1+x^2+x^3+x^5+x^6+x^7+x^{91}$ |
| 24 | $1+x^{20}+x^{21}+x^{23}+x^{24}$ | 58 | $1+x+x^5+x^6+x^{58}$ | 92 | $1+x^2+x^5+x^6+x^{92}$ |
| 25 | $1+x^3+x^{25}$ | 59 | $1+x+x^3+x^4+x^5+x^6+x^{59}$ | 93 | $1+x^2+x^{93}$ |
| 26 | $1+x+x^2+x^6+x^{26}$ | 60 | $1+x+x^{60}$ | 94 | $1+x+x^5+x^6+x^{94}$ |
| 27 | $1+x+x^2+x^5+x^{27}$ | 61 | $1+x+x^2+x^5+x^{61}$ | 95 | $1+x+x^2+x^4+x^5+x^6+x^{95}$ |
| 28 | $1+x^3+x^{28}$ | 62 | $1+x^3+x^5+x^6+x^{62}$ | 96 | $1+x^2+x^3+x^4+x^6+x^7+x^{96}$ |
| 29 | $1+x^2+x^{29}$ | 63 | $1+x+x^{63}$ | 97 | $1+x^6+x^{97}$ |
| 30 | $1+x+x^2+x^{23}+x^{30}$ | 64 | $1+x+x^3+x^4+x^{64}$ | 98 | $1+x+x^2+x^3+x^4+x^7+x^{98}$ |
| 31 | $1+x^3+x^{31}$ | 65 | $1+x+x^3+x^4+x^{65}$ | 99 | $1+x^4+x^5+x^7+x^{99}$ |
| 32 | $1+x+x^2+x^{22}+x^{32}$ | 66 | $1+x^2+x^3+x^5+x^6+x^8+x^{66}$ | 100 | $1+x^2+x^7+x^8+x^{100}$ |
| 33 | $1+x^{13}+x^{33}$ | 67 | $1+x+x^2+x^5+x^{67}$ | 107 | $1+x+x^2+x^3+x^5+x^7+x^{107}$ |
| 34 | $1+x+x^2+x^{27}+x^{34}$ | 68 | $1+x+x^5+x^7+x^{68}$ | 127 | $1+x+x^{127}$ |

**Annexe 5 : Layout de PLA avant et après un codage de Berger.**

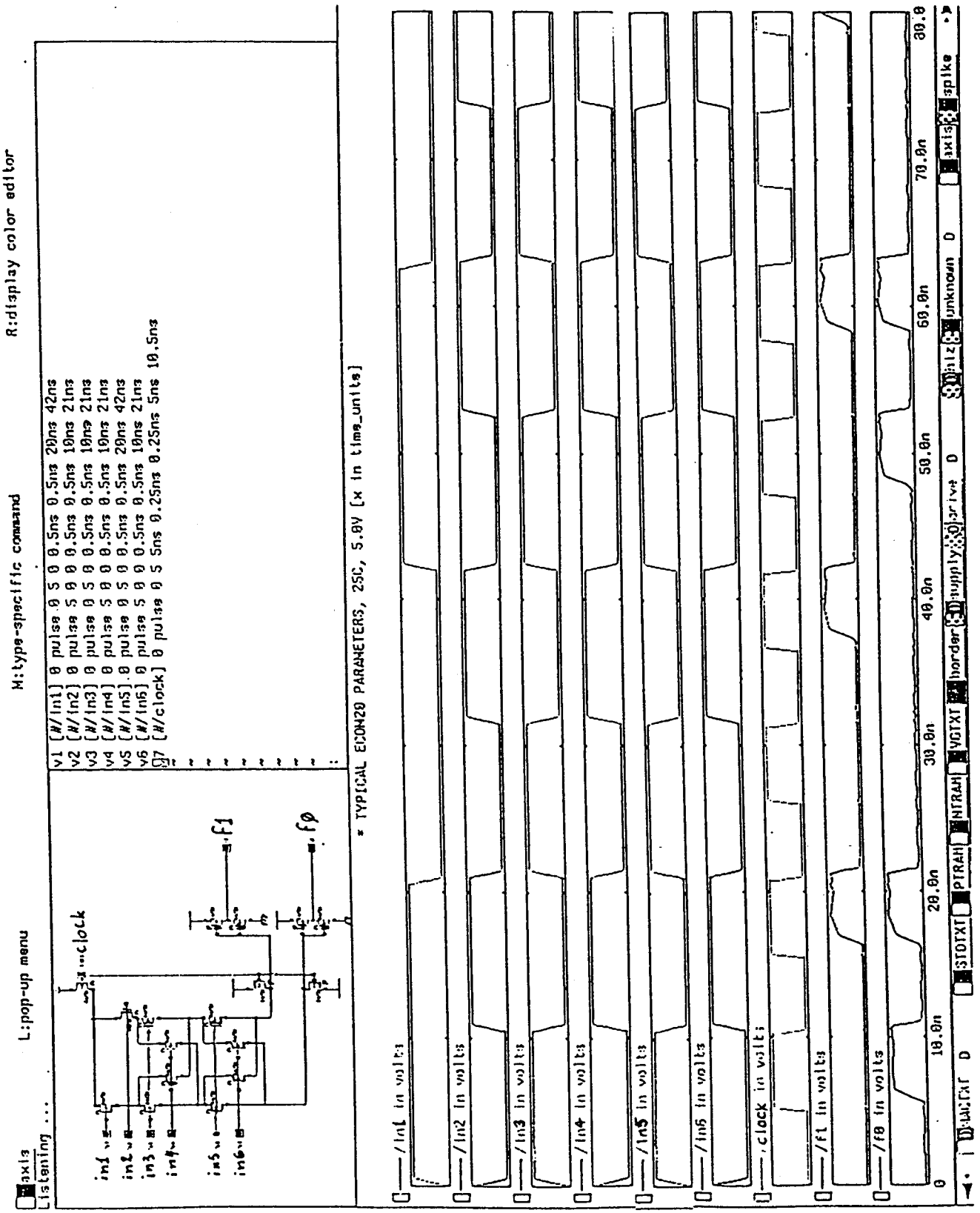


**Annexe 6 : Contrôleur double-rail : Simulation logique et électrique, et la génération automatique de son layout.**

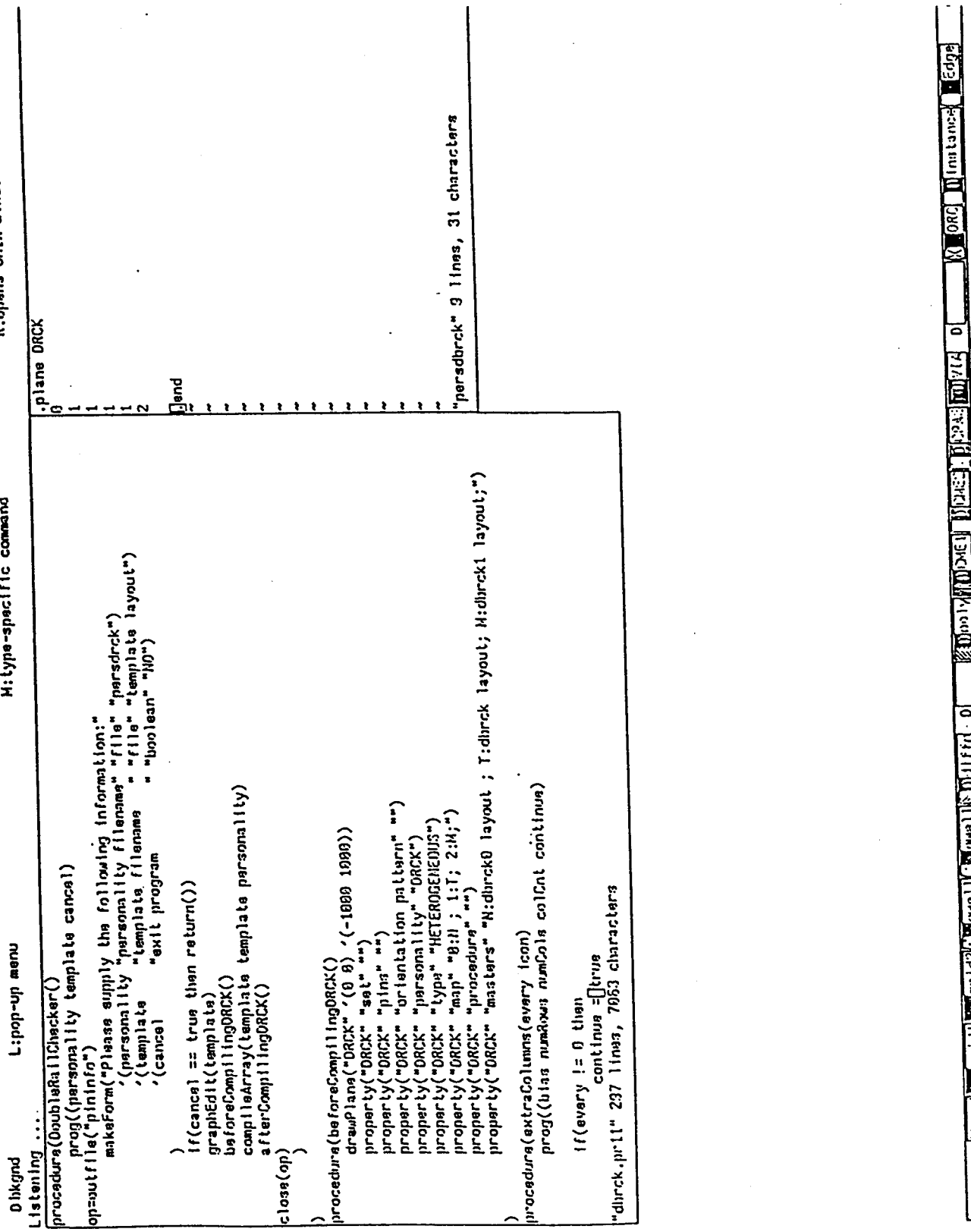
Simulation logique (SILOS) d'un contrôleur double-rail à 3 paires d'entrées.



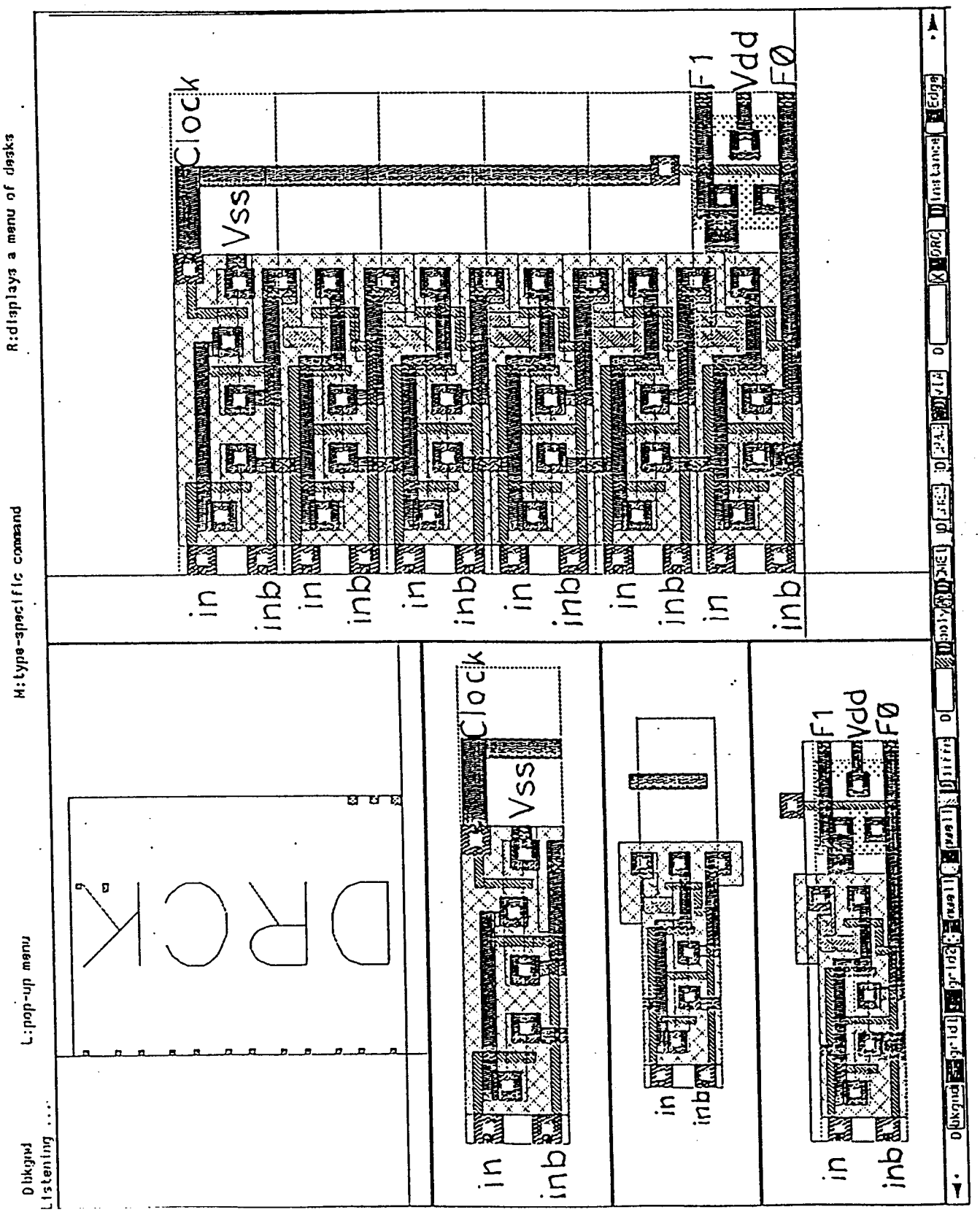
Simulation électrique (SPICE) d'un contrôleur double-rail à 3 paires d'entrées



Spécifications et fichier de personnalisation, pour la génération automatique d'un contrôleur double-rail à 6 paires d'entrées.

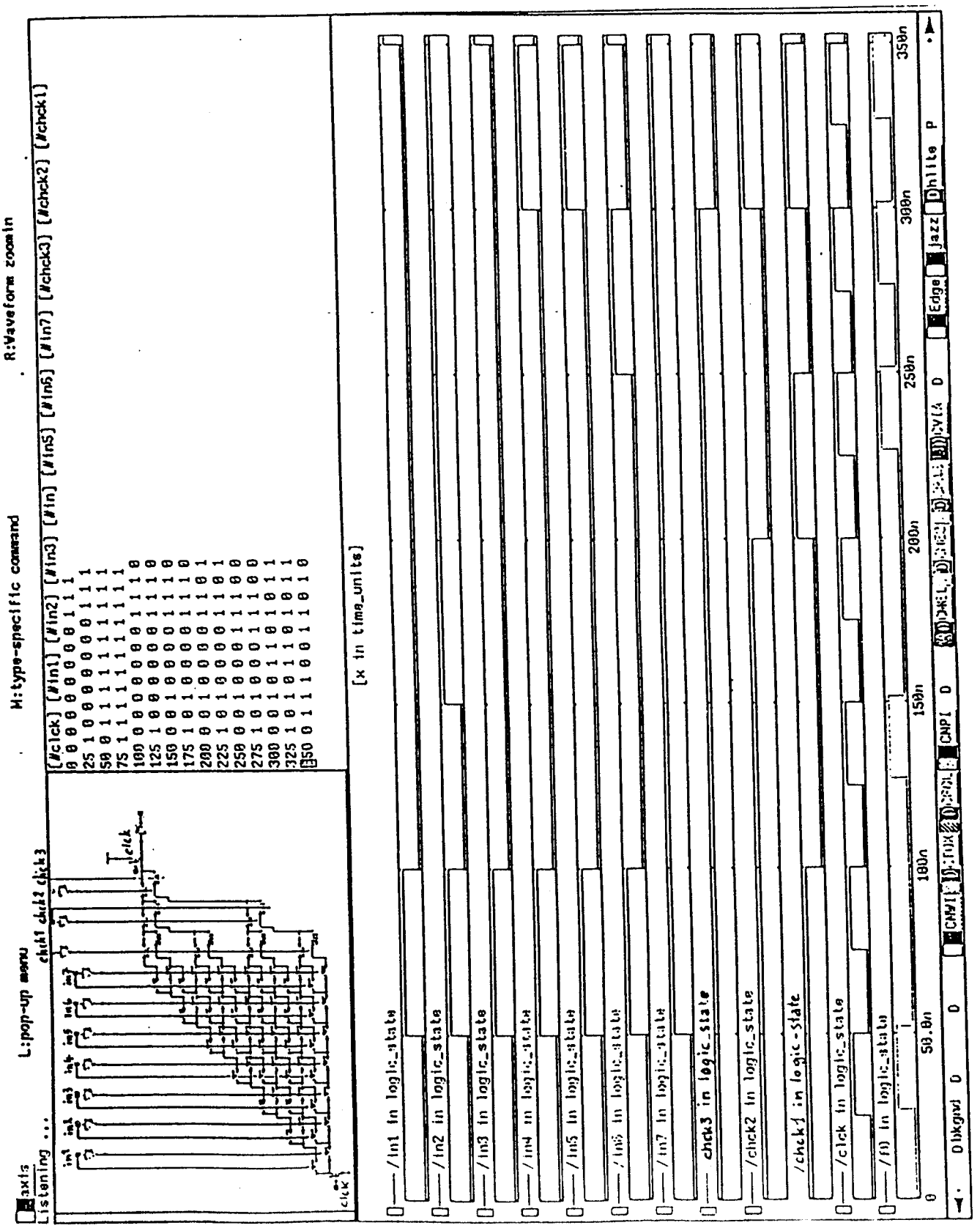


Cellules de base pour générer un contrôleur double-rail CVSL dynamique, et layout d'un contrôleur à 6 paires d'entrées.

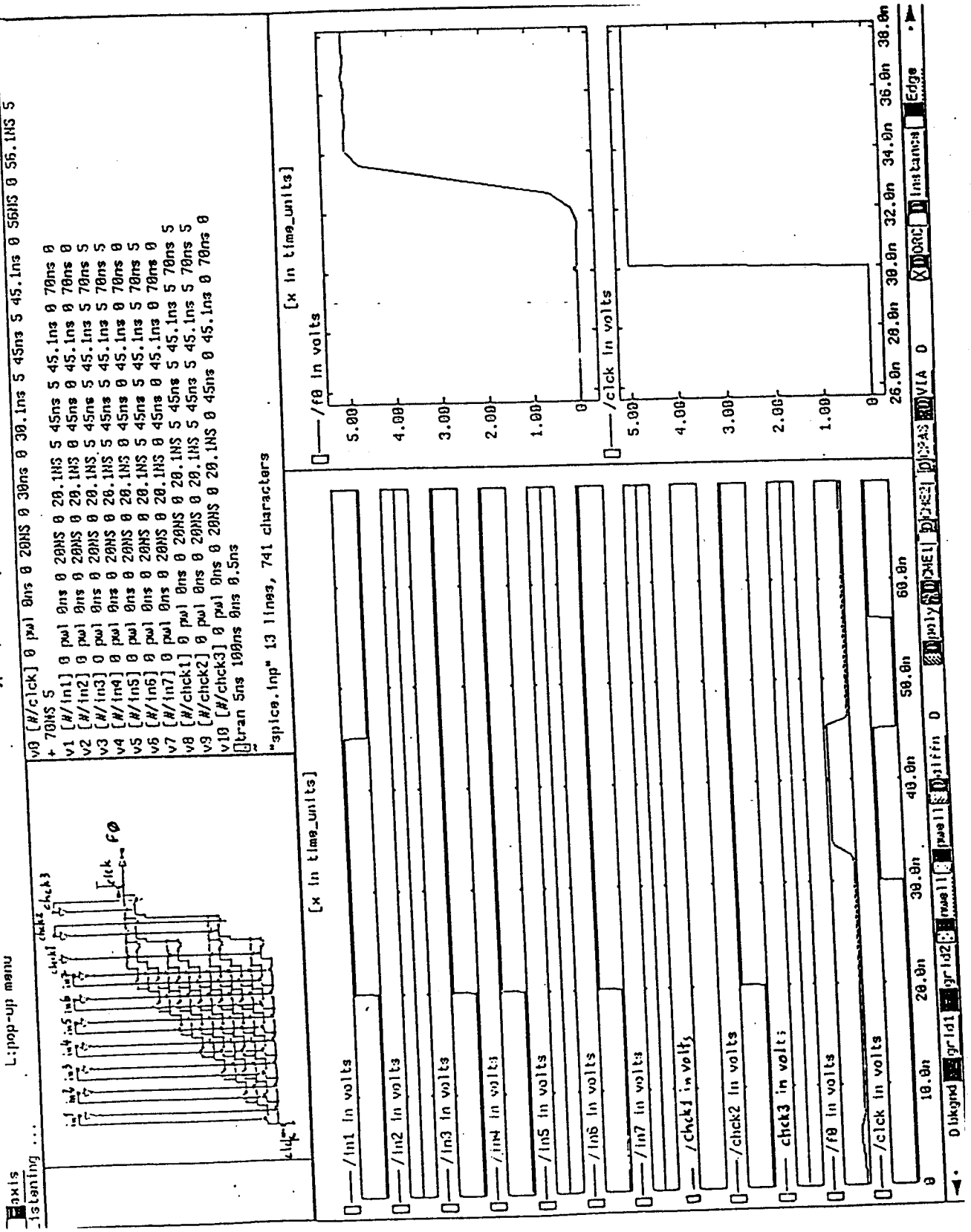


**Annexe 7 : Contrôleur du code de Berger : Simulation logique et électrique, et la génération automatique de son layout.**

Simulation logique (SILOS) d'un contrôleur du code de Berger à 7 bits d'information et 3 bits de codage.

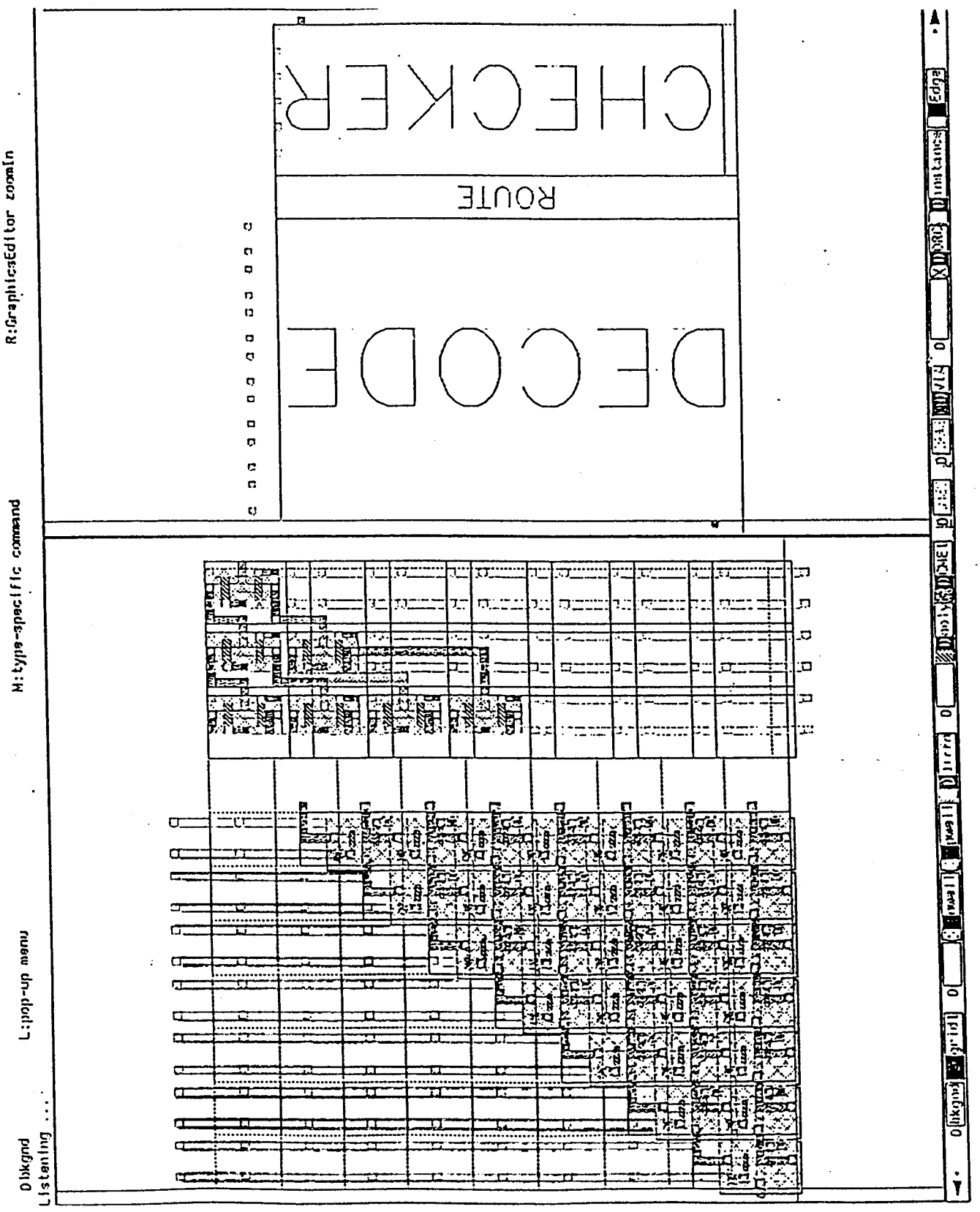


Simulation électrique (SPICE) d'un contrôleur du code de Berger à 7 bits d'information et 3 bits de codage.



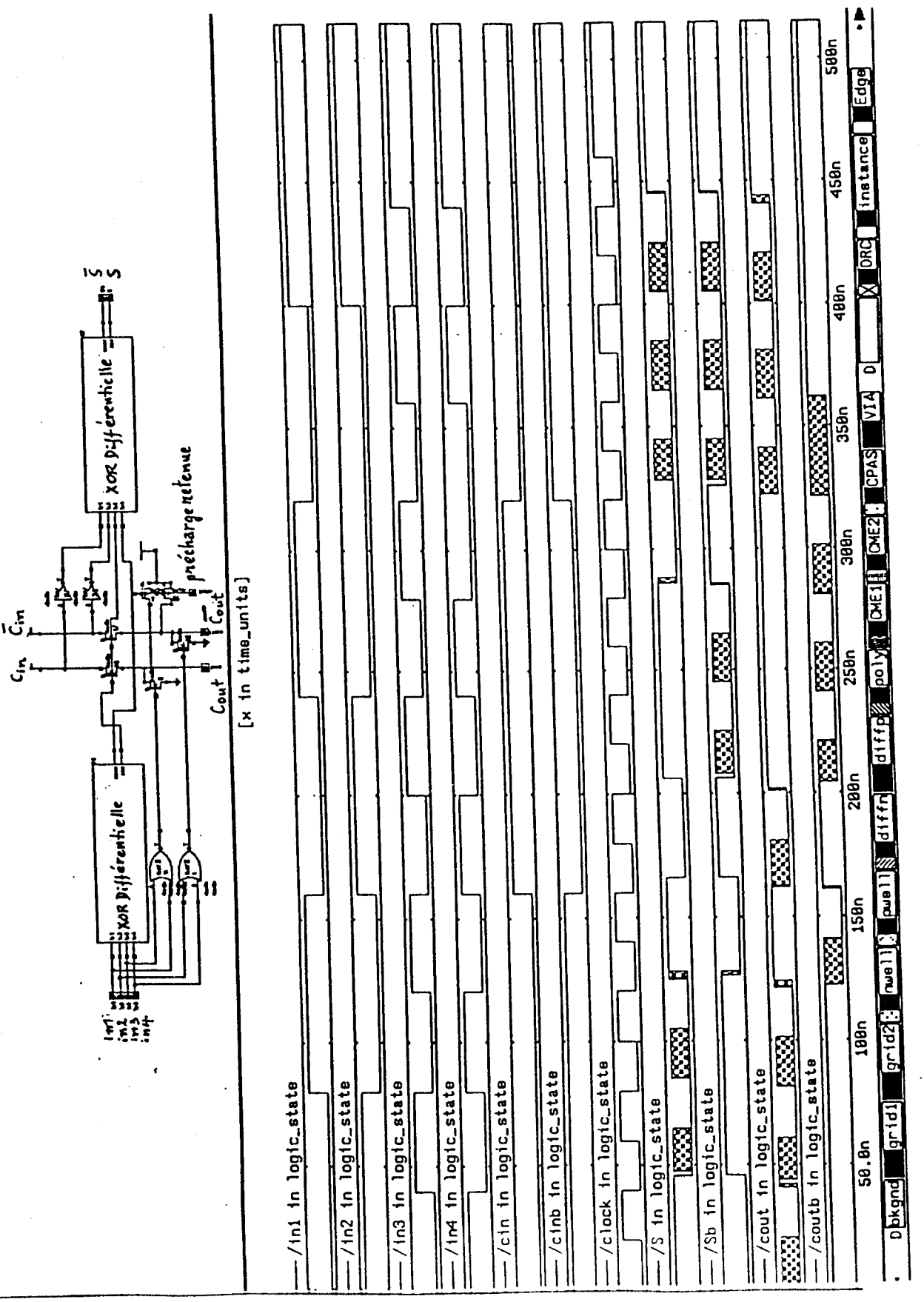
Spécifications et fichier de personnalisation, pour la génération automatique d'un contrôleur du code de Berger à 7 bits d'information et 3 bits de codage.

Layout généré automatiquement, d'un contrôleur du code de Berger à 7 bits d'information et 3 bits de codage.

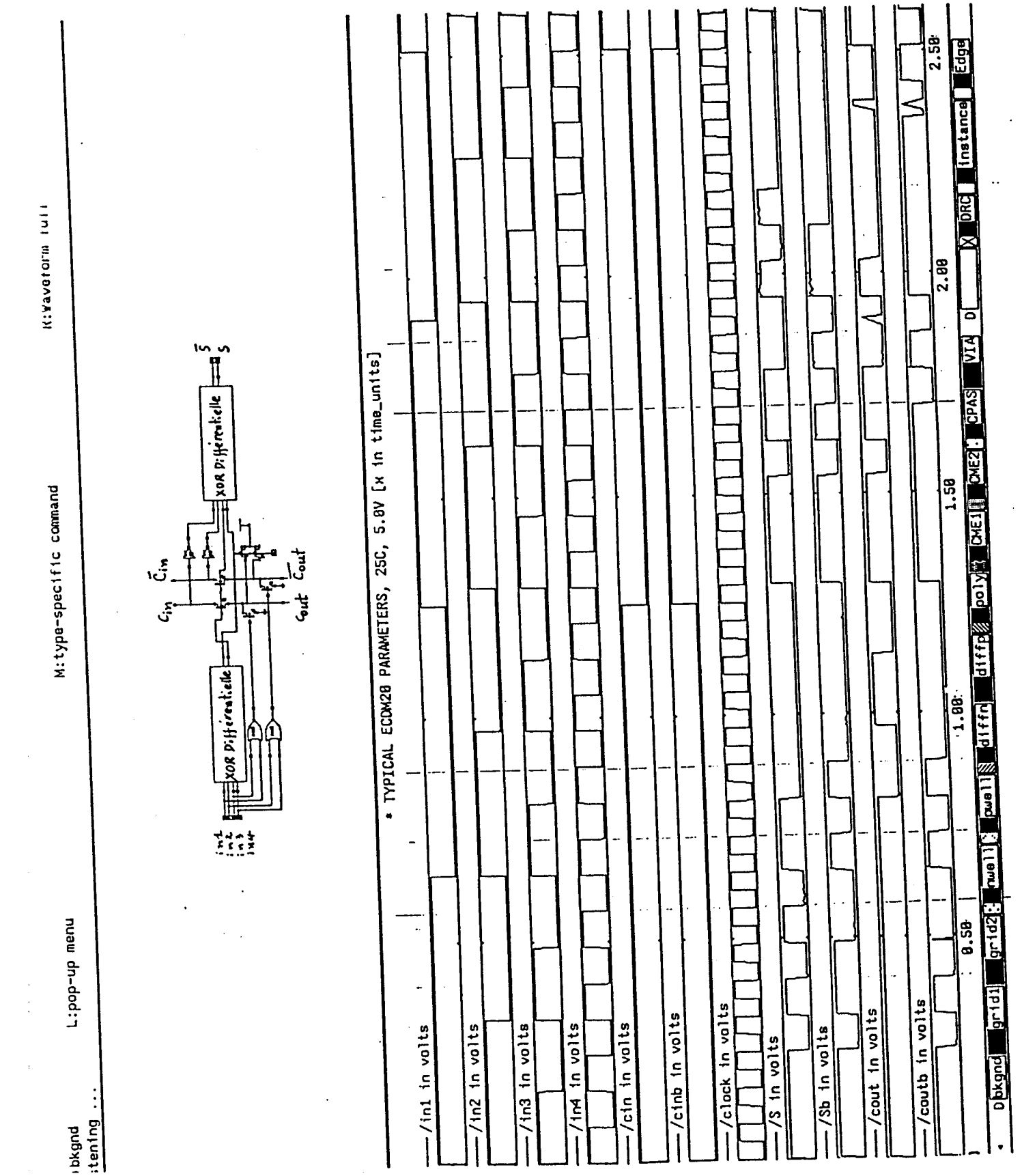


**Annexe 8 : UA et UAL autocontrôlables : Simulation logique et électrique, et la génération automatique de leur layout.**

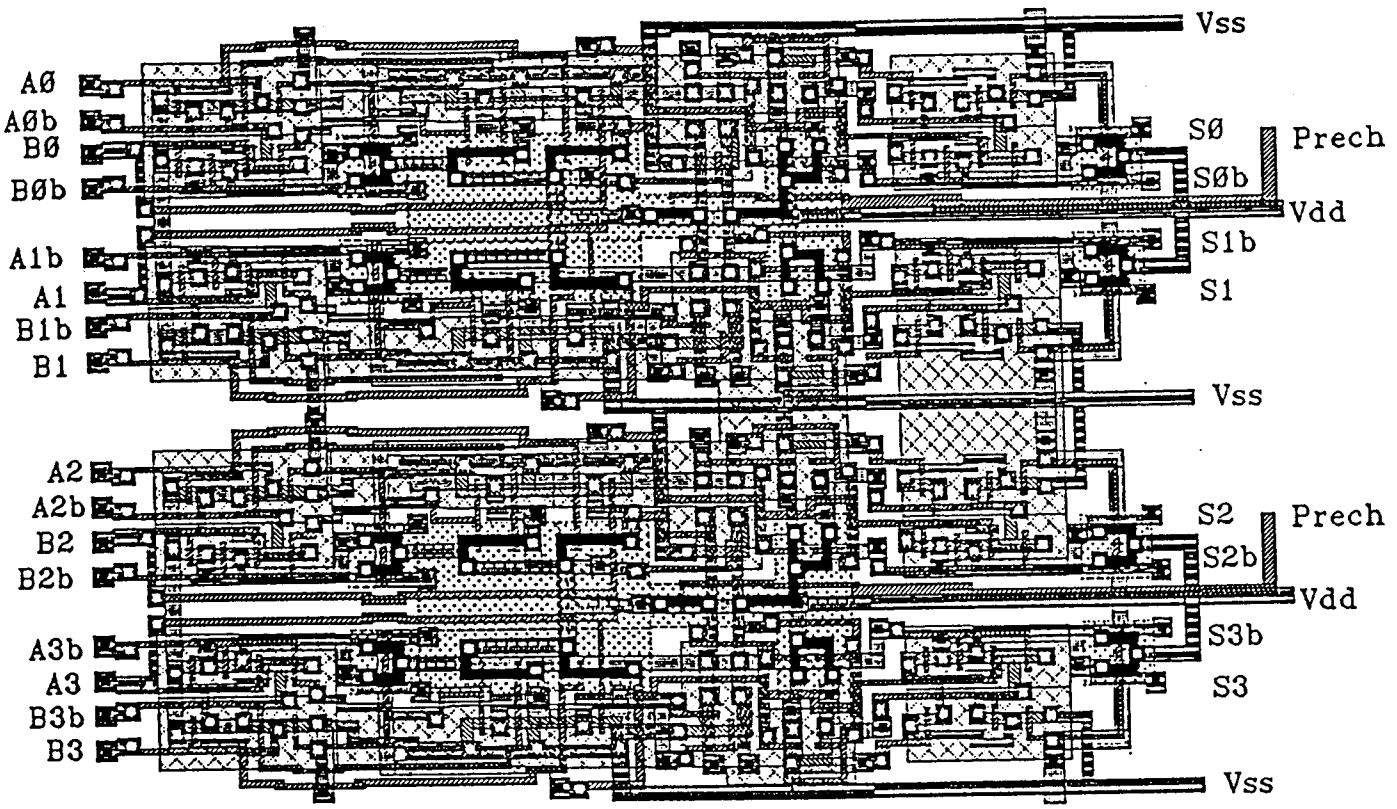
Simulation logique (SILOS) d'une tranche de bit d'une unité arithmétique autocontrôlable de la figure VI. 8.

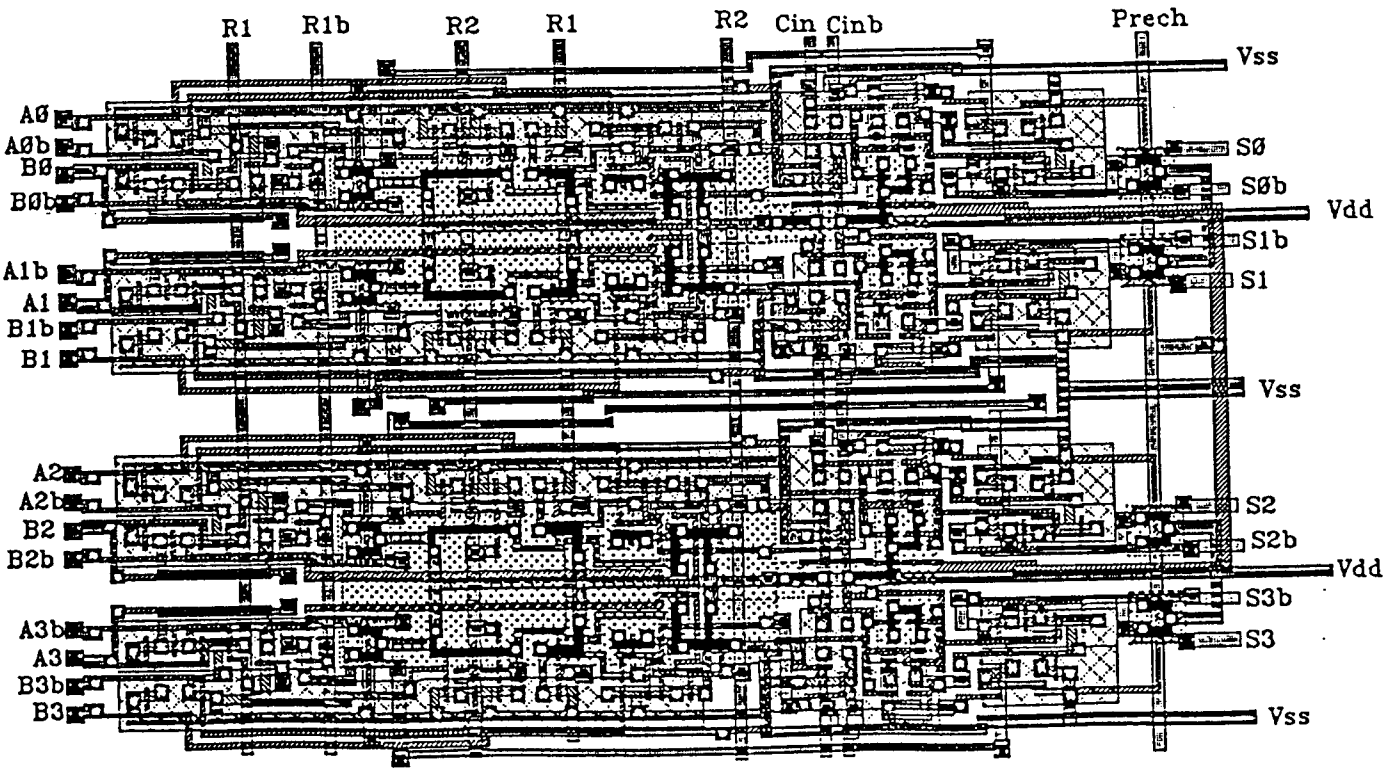


Simulation électrique (SPICE) d'une tranche de bit d'une unité arithmétique autocontrôlable de la figure VI. 8.



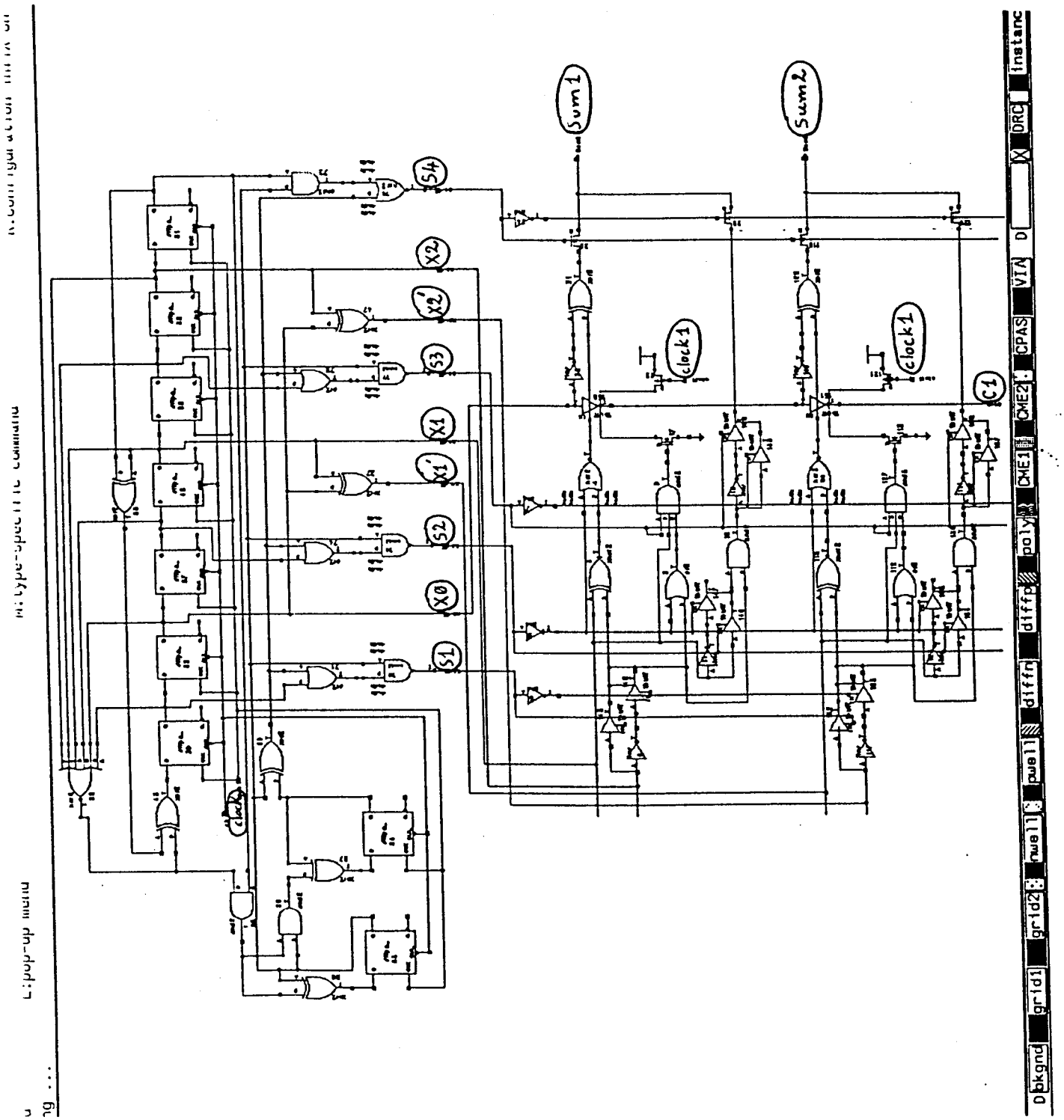
Layout d'une unité arithmétique de la figur VI.8 (resp. unité arithmétique et logique de la figure VI.10a) autocontrôlable de 4 bits.



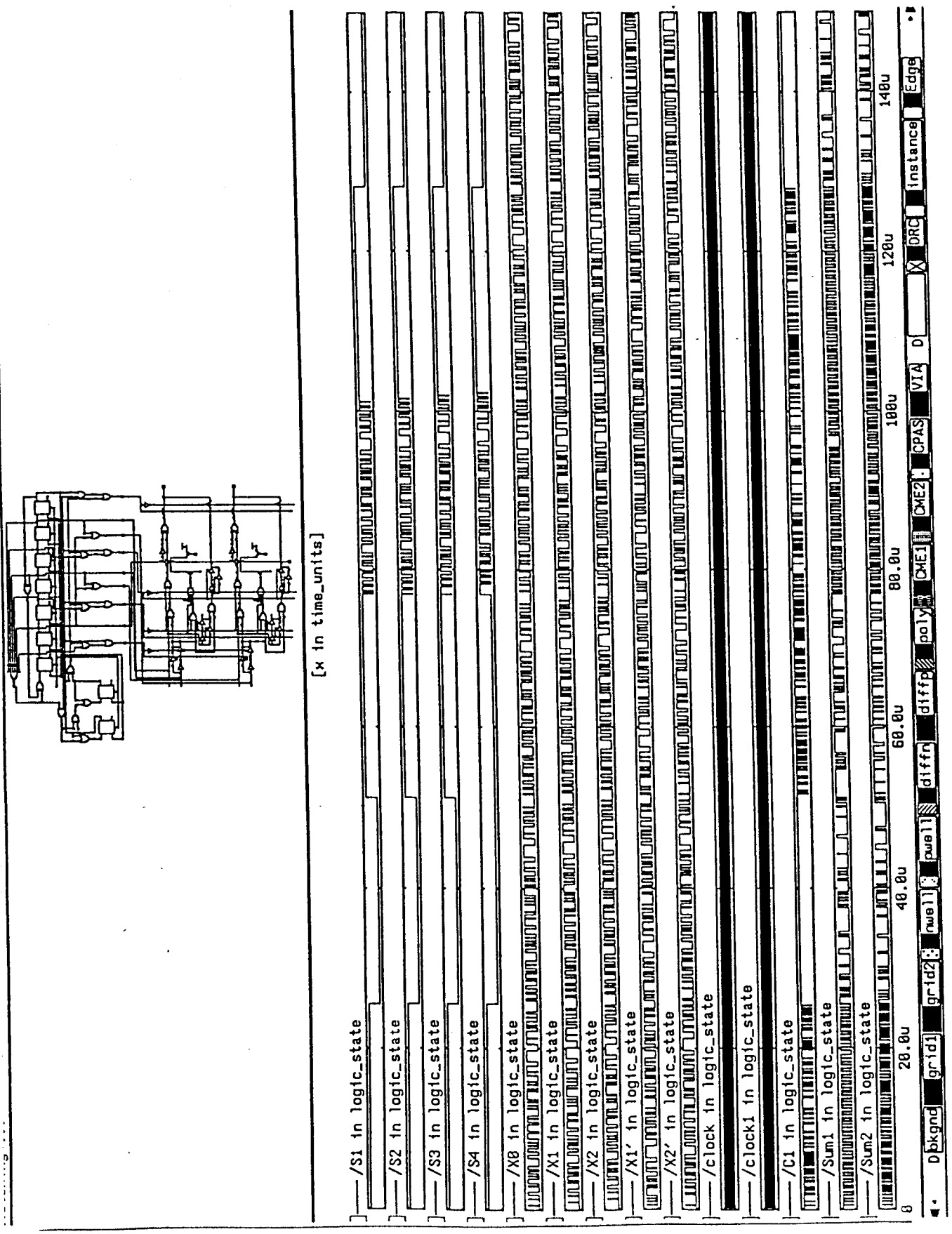



**Annexe 9 : Simulation logique de génération complète de vecteurs de test pour une UAL CMOS à 2 tranches de bits.**

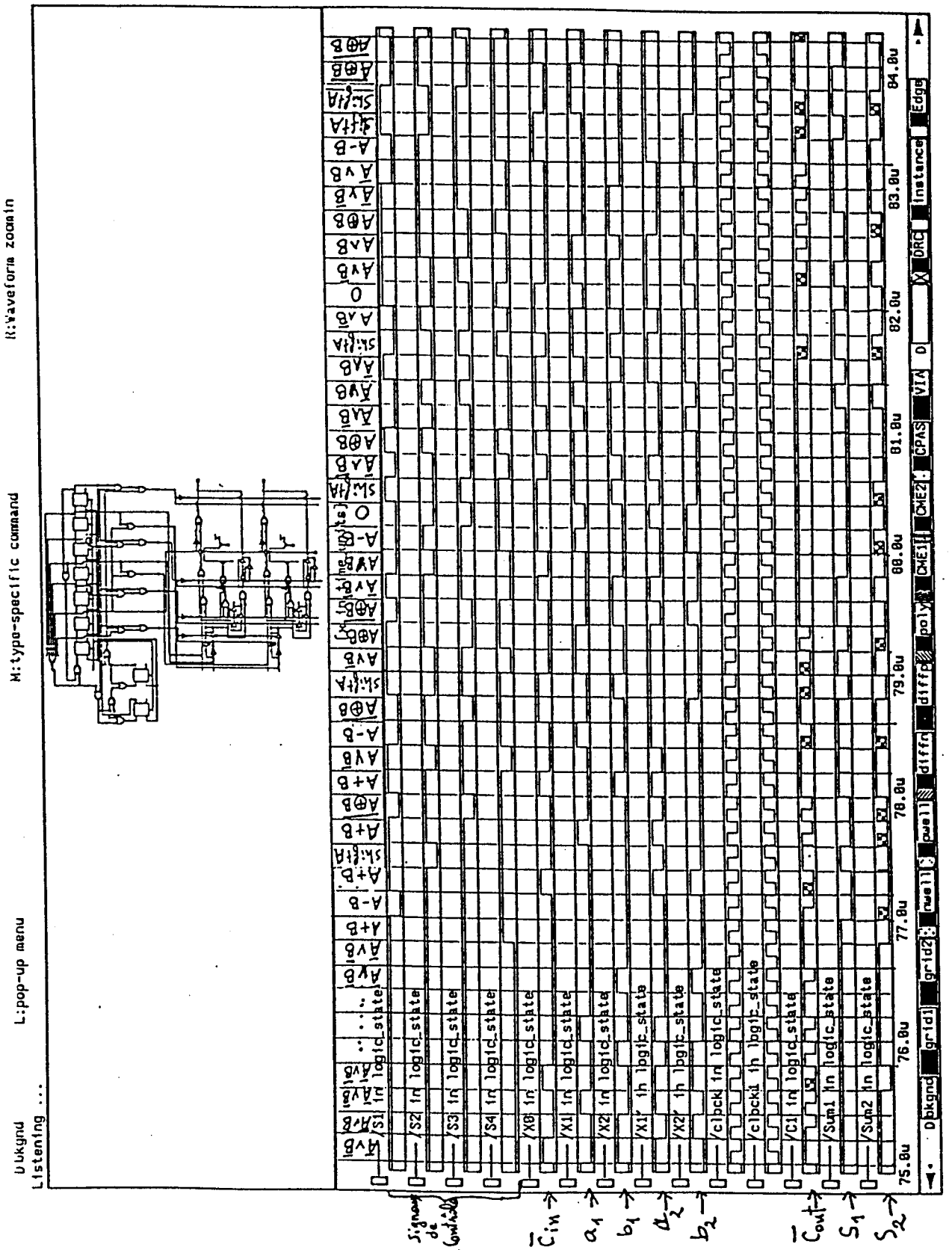
"Schématic" du générateur de la figure VII.13, activant une UAL à 2 tranches de bit conçue suivant le schéma de la figure VII.6.



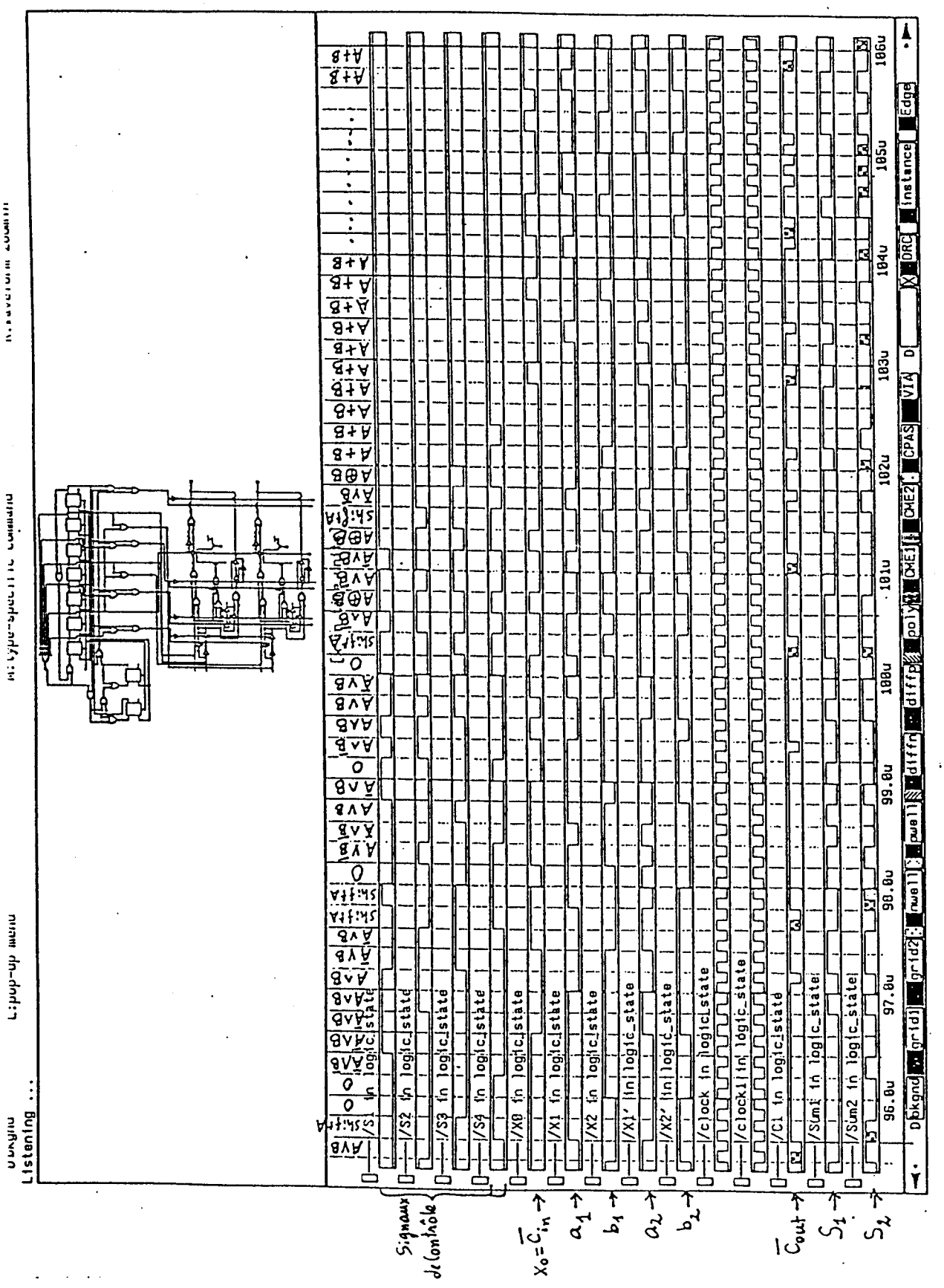
Vue globale de la simulation, de 0 à 140 µs.



Vue en détail et commentaire sur le type d'opération de 0 à 84 µs.



Vue en détail et commentaire sur le type d'opération de 84 à 106 µs.



INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

INPG

DÉPARTEMENT DES ÉTUDES DOCTORALES

# AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales

VU les Rapports de présentation de :

- Monsieur Joan FIGUERAS
- Monsieur Christian LANDRAULT

**Monsieur Kholdoun TORKI**

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE, spécialité

"Microélectronique"

Fait à Grenoble, le 4 Juillet 1990

Pour le Président de l'I.N.P.G.
et par délégation
Le Vice-Président

C. GAUBERT

INPG

Résumé

Bien que les techniques d'autotest intégré soient en perpétuel développement sous forme de théories et de schémas de conception, leur réalisation concrète et leur implémentation posent des problèmes cruciaux.

Une stratégie d'autotest intégré est proposée dans cette thèse pour des circuits générés par compilation de silicium.

Le schéma UBIST d'unification du test en-ligne et hors-ligne assure la plupart des tests nécessaires durant la vie d'un circuit intégré (test de fin de fabrication, test de maintenance, test en-ligne, ...).

A la base du schéma UBIST se trouve le schéma "Self-Checking" (test en-ligne, pour lequel le circuit est composé de blocs fonctionnels "Strongly Fault Secure" (SFS) et de contrôleurs "Strongly Code Disjoint" (SCD) ). Le but à atteindre par de tels circuits est couramment appelé le "Totally Self-Checking Goal", qui consiste à détecter la première erreur survenant aux sorties du bloc fonctionnel, sous forme d'indication d'erreur sur les sorties du contrôleur.

Autour de ce schéma Self-Checking est implémentée une structure de test, du type BILBO, assurant des phases de test hors-ligne, qui a pour objectif d'augmenter le taux de couverture des pannes multiples et de renforcer les propriétés SFS et SCD pour certains blocs fonctionnels et contrôleurs. L'unification des tests en-ligne et hors-ligne permet de tirer les avantages de chacun de ces tests, permettant une implémentation efficace d'autotest intégré.

Une méthodologie de conception pour implémenter ce schéma UBIST est proposée pour des parties contrôle hiérarchiques à base de PLAs et des parties opératives parallèles en structure bit-slice (du type de celle du MC 68000). Ce sont les architectures cibles utilisées par le compilateur de silicium SYCO (développé au sein de l'équipe d'architecture des ordinateurs du laboratoire TIM3/IMAG). Une solution topologique efficace est proposée pour ces schémas UBIST.

L'un des points essentiels de cette thèse est qu'elle s'appuie sur des réalisations d'outils de CAO spécifiques au test, compatibles avec les spécifications architecturales et structurelles utilisées par SYCO. C'est une phase importante de validation du concept de "compilateur de circuits autotestables".

**Mots-clés :**

Circuits autotestables, compilation de silicium, schéma UBIST, BILBO, PLA, partie contrôle hiérarchique, partie opérative parallèle, contrôleurs de code détecteur d'erreurs.