Méthodes de synthèse optimisée pour compilateurs de silicium

Franck Poirot

THESE

présentée par

Franck POIROT

pour obtenir le titre de **DOCTEUR**

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

Spécialité : **Micro-électronique**

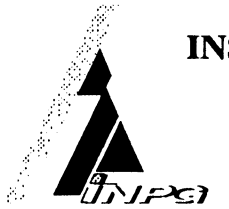
Méthodes de Synthèse Optimisée pour Compilateurs de Silicium

Date de soutenance : 3 Juillet 1990

Composition du jury :

Jacques Mossière	Président
Roland Gerber	Rapporteurs
Christian Landrault	
Gabrièle Saucier	Examinateurs
Jacques-Olivier Piednoir	

Thèse préparée au sein du laboratoire Conception de Systèmes Intégrés



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

GRENOBLE 1
UNIVERSITE
JOSEPH FOURIER
SCIENCES. TECHNOLOGIE. MEDECINE

Président de l'Université :
M. NEMOZ Alain

Année universitaire 1988-1990

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GÉOGRAPHIE

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 I.S.N.
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 Gerd	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 Jean-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	Thermodynamique 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. IUT 1
DODU Jacques	Mécanique Appliquée IUT 1
NEGRE Robert	Génie Civil IUT 1
NOUGARET Marcel	Automatique IUT 1
PERARD Jacques	E.E.A. IUT 1

PROFESSEURS de 2ème Classe

BEE Marc	Physique IUT 1
BOUTHINON Michel	E.E.A. 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	E.E.A. 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	E.E.A. 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 Thérapeutique	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	Hygiène 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 Exceptionnelle et 1ère Classe

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatrie-Puériculture	C.H.R.G.
BEREZ Henri	Orthopédie-Traumatologie	Hôpital 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	Hôpital 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	Anesthésiologie	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	Hôpital 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	Hôpital 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	Hôpital 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.

<< Tous les événements sont enchaînés dans le meilleur des mondes possibles : car enfin si vous n'aviez pas été chassé du beau château à grands coups de pied dans le derrière pour l'amour de mademoiselle Cunégonde, si vous n'aviez pas été mis à l'Inquisition, si vous n'aviez pas couru l'Amérique à pied, si vous n'aviez pas donné un bon coup d'épée au baron, si vous n'aviez pas perdu tous vos moutons du bon pays Eldorado, vous ne mangeriez pas ici des cédrats confits et des pistaches. -Cela est bien dit, répondit Candide, mais il faut cultiver notre jardin. >>

Candide ou l'Optimisme, chapitre 30, Voltaire.

Remerciements

Je tiens à remercier :

Madame Gabrièle Saucier, Professeur à l'ENSIMAG
qui m'a accueilli dans son équipe de recherche et qui m'a encadrée tout au long de l'élaboration de cette thèse.

Monsieur Jacques Mossières, directeur de l'ENSIMAG,
de me faire l'honneur de présider le jury de cette thèse.

Monsieur Roland Gerber, responsable de la division "Conception de Circuits Intégrés" au Centre Norbert Segart (CNET) à Grenoble,
et Monsieur Christian Landrault, directeur de recherche au CNRS,
d'avoir accepté d'être rapporteur de cette thèse et membre du jury.

Monsieur Jacques-Olivier Piednoir, responsable du groupe "Recherche et Développement" de la société VLSI Technology,
de m'avoir accueilli dans son groupe et d'avoir accepté d'être membre du jury.

La société VLSI Technology et tout particulièrement Monsieur Paul McLellan, directeur de son centre de recherche européen,
ainsi que le service CIFRE dépendant du ministère de la recherche et de la technologie,
de m'avoir permis de mener à bien cette thèse dans d'excellentes conditions.

Que tous mes collègues de la société VLSI Technology et du laboratoire CSI de l'Institut National Polytechnique trouvent ici l'expression de ma profonde sympathie pour les remarques constructives dont ils m'ont fait profiter et pour leur soutien.

Résumé

La synthèse logique joue un rôle fondamental dans les compilateurs de silicium. Alors que l'état de l'art de la synthèse deux couches est très avancé, celui de la synthèse multi-couches reste encore un sujet très ouvert. L'objet de cette thèse est de présenter des méthodes originales de synthèse de contrôleurs et de systèmes combinatoires pour une implémentation multi-couches à base de cellules de bibliothèque.

Le premier chapître définit le concept de compilation de silicium et introduit l'une de ses composantes, la synthèse logique. L'importance du marché de la synthèse logique y est clairement définie ainsi que ses implications dans la conception actuelle de circuits intégrés.

Le deuxième chapitre concerne la synthèse de contrôleurs. Le problème du codage des machines d'états fini est traité en détail et une méthode basée sur la théorie d'immersion de cubes intersectants dans un hypercube booléen est proposée.

Le troisième chapitre est consacré à la synthèse de circuits combinatoires et une méthode d'optimisation temporelle de tels dispositifs est développée.

Ces travaux ont été implémentés dans un environnement industriel.

Mots-Clefs : CAO, Compilateurs de silicium, Synthèse logique, Synthèse de contrôleurs, Codage des états, Hypercube booléen, Synthèse multi-couches, Optimisation de circuits.

Introduction

L'évolution de la technologie des circuits intégrés ainsi que l'augmentation de la puissance de calcul des systèmes informatiques ont contribué au développement de logiciels puissants d'aide à la conception de ces circuits. Ces outils permettent d'augmenter la productivité des concepteurs en les déchargeant de tâches longues et fastidieuses tout en garantissant un produit final de bonne qualité et exempt de défauts.

Actuellement, la complexité des circuits intégrés est telle (le million de transistors pour le microprocesseur INTEL 80860), qu'il n'est pas envisageable de concevoir leur réalisation sans faire appel à une méthode de conception utilisant de façon massive des outils automatiques. De plus, la compétitivité dans le domaine des circuits à la demande ou ASICs (Application Specific Integrated Circuits) ainsi que les nouvelles méthodes de réalisation rapide de circuits intégrés (prédiffusés ou précaractérisés) demandent des outils fonctionnant de façon rapide et à moindre coût.

C'est dans le cadre de la compilation de silicium et plus particulièrement de la synthèse logique, c'est à dire de la synthèse de fonctions booléennes et d'automates d'états fini que s'est effectuée cette thèse.
Une attention toute particulière a été portée à l'impact des méthodes de synthèse sur les caractéristiques physiques des circuits finaux. En effet, les outils de synthèse logique généralement proposés dans le commerce, ont pour objectif de produire des circuits optimisés suivant des critères fondés sur leur niveau de description structurelle (les circuits obtenus sont optimisés en terme de portes logiques équivalentes nécessaires à leur réalisation). Ici, nous avons recherché et développé des méthodes dont les critères d'optimisation sont influencés par des critères de plus bas niveau liés à l'implémentation finale du circuit.

Le premier chapitre de cette thèse définit le concept de compilation de silicium et introduit l'une de ses composantes, la synthèse logique. L'importance du marché de la synthèse logique y est clairement définie ainsi que ses implications dans la conception actuelle de circuits intégrés.

Le deuxième chapitre concerne la synthèse de contrôleurs. La conception architecturale d'un circuit de type processeur y est exposée ainsi que la méthodologie permettant de traiter de tels circuits en partant d'une description de haut niveau. Le problème du codage des machines d'états fini est traité en détail et une méthode basée sur la théorie d'immersion de cubes intersectants dans un hypercube est proposée.

La troisième partie est consacrée à la synthèse automatique de circuits combinatoires. La problématique et les divers outils permettant de la résoudre efficacement y sont détaillés. Ensuite, une méthode d'optimisation temporelle de circuits combinatoires est développée. Cette méthode est basée sur des modèles précis de propagation temporelle aboutissant à des décompositions locales optimales de portes logiques.

Les résultats de ces travaux ont été implémentés en MAINSAIL [MAI86] sur SUN dans le logiciel de synthèse logique de la société VLSI TECHNOLOGY.

I. Introduction à la Synthèse Logique

Les compilateurs de silicium :

Depuis quelques années, l'évolution dans la conception de circuits intégrés a amené les concepteurs à automatiser de plus en plus leurs tâches pour pouvoir réaliser des circuits complexes en un temps raisonnable. C'est de ce besoin d'outils automatiques qu'est apparue la notion de "compilateurs de silicium".

Afin de définir ce concept, nous allons introduire le contexte dans lequel se situe la conception d'un circuit.

La définition du circuit à réaliser contenue dans le cahier des charges est généralement exprimée dans un langage de haut niveau. Ce langage peut adresser plusieurs niveaux d'abstraction : niveau logique, niveau transfert de registres ou niveau algorithmique.

- Le niveau *logique* exprime le comportement du circuit sous forme d'un ensemble d'équations booléennes, indépendantes de toute architecture interne,
- au niveau *transfert de registre*, le fonctionnement d'un circuit est décrit en terme d'opérations et de transfert entre registres,
- le niveau *algorithmique* exprime le comportement du circuit par un algorithme en terme de variables ; il correspond au niveau le plus élevé.

L'ensemble de ces niveaux est couvert par des langages de spécification de haut niveau tels que VHDL [VHD87], VERILOG ou CADOC [CRA85].

A partir de cette description de haut niveau, l'objectif est dans un premier temps de définir une architecture composée de blocs de haut niveau permettant de satisfaire les fonctionnalités initiales du circuit, puis, dans un deuxième temps, de traduire chacun des blocs définis dans cette architecture à l'aide d'une interconnexion de ressources élémentaires disponibles dans la structure cible choisie.

C'est pour traduire toutes ces étapes consistant en la transformation d'une spécification de haut niveau en un circuit intégré, qu'est apparue la notion de compilation de silicium. Ce terme générique fut tout d'abord introduit par D. JOHANNSEN [JOH79] pour définir les "assembleurs" de silicium. Les assembleurs de silicium sont des outils de génération automatique de dessins de circuits décrits comme un assemblage hiérarchique de motifs élémentaires (PLA, RAM, cellules de bibliothèque...). Ensuite, P. WALLICH [WAL83] étendit cette définition à un programme ou un ensemble de programmes permettant de générer l'implémentation physique d'un circuit (dessin des masques de fabrication), à partir d'une description de haut niveau.

Le processus de la compilation de silicium s'effectue selon plusieurs étapes qui sont représentées à l'aide de la carte en Y de GAJSKI et KUHN [GAJ83]. Cette carte permet de décrire un compilateur de silicium selon ses trois domaines d'application : le domaine comportemental ou fonctionnel, le domaine structurel et le domaine physique (figure I-1).

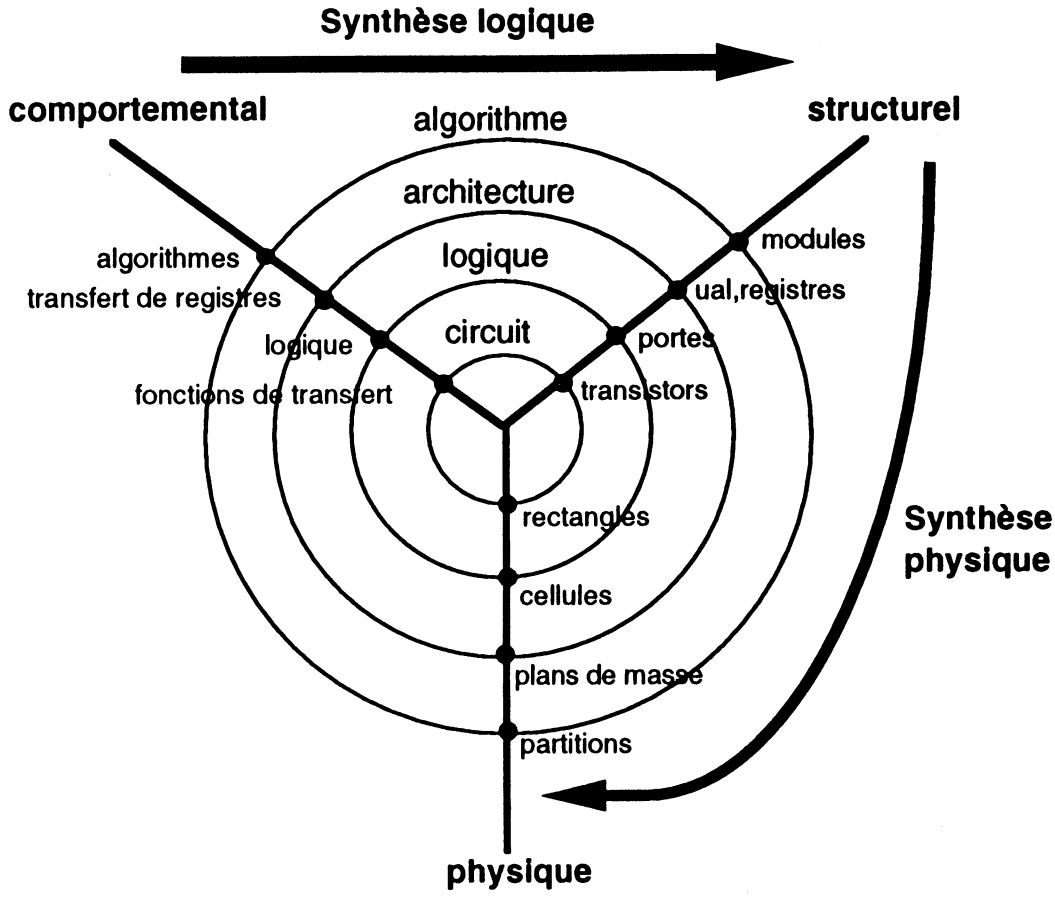


figure I-1 : carte en Y d'un compilateur de silicium

Le domaine comportemental considère la fonctionnalité du circuit, c'est à dire son comportement en terme d'entrée / sortie. Il est indépendant de la technologie.

Le domaine structurel précise la structure du circuit comme une interconnexion de blocs logiques considérés comme des entités de base. Plusieurs représentations étant possibles, on recherche celle correspondant au mieux aux contraintes du cahier des charges (vitesse, surface, testabilité, puissance, coût...).

Le domaine physique relie la structure précédente à l'implémentation physique sur silicium (dessin des masques).

Un compilateur idéal [WER82] part d'une description fonctionnelle ou algorithmique, génère un représentation structurelle, pour obtenir finalement la description physique ou géométrique du circuit (figure I-2).

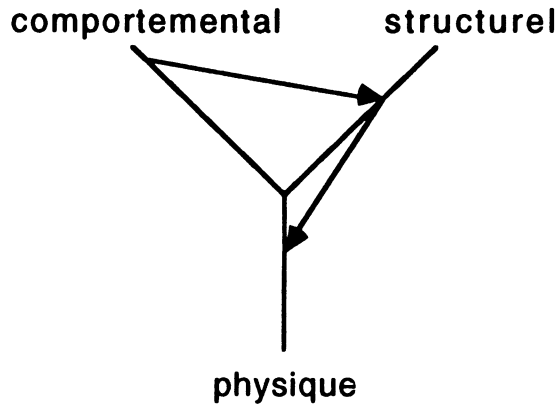


figure I-2 : compilateur idéal

Nous allons maintenant définir plus en détail l'un des aspects de la compilation de silicium, la synthèse logique, domaine dans lequel s'est effectuée cette thèse.

Synthèse logique :

Les outils de synthèse logique permettent de partir d'une spécification de haut niveau (algorithmique, transfert de registres ou logique) et de générer la description d'un circuit au niveau logique en terme de portes de base.

Cette synthèse s'effectue généralement par étapes successives (figure I-3). Par exemple, à partir d'une description algorithmique une architecture en terme d'interconnexion de blocs logiques est proposée selon des critères

du cahier des charges (1). Les blocs qui ne correspondent pas à une structure directement implémentable (2), seront synthétisés pour générer une nouvelle description exprimée en terme de blocs de plus bas niveau (3) compatibles avec la structure d'accueil choisie (éléments de bibliothèques de cellules précaractérisés...). Cette forme structurelle est alors traduite en une description géométrique du circuit (4).

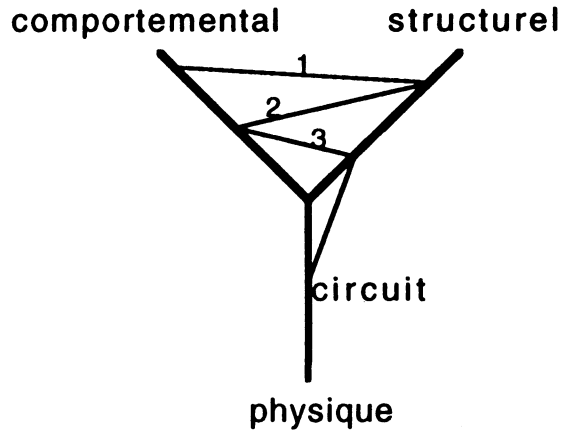


figure I-3 : Représentation du synthétiseur de logique

Ainsi, les outils de synthèse logique peuvent se diviser en deux grands ensembles. Le premier concerne ceux qui produisent une description architecturale d'un circuit à partir d'une description de haut niveau (algorithme ou transfert de registres) ; c'est la synthèse de haut niveau. Le second génère une description structurelle directement assimilable par les outils de synthèse physique, c'est la synthèse de bas niveau.

L'intérêt des outils de synthèse s'est soudainement accru ces dix dernières années.

Au début des années 1980, IBM CORP. fut l'une des premières compagnies à développer son propre outil de synthèse à usage interne, le LOGIC SYNTHESIS SYSTEM (ou LSS). Le LSS fut premièrement utilisé sur une

centaine de circuits du système 3090. Il s'est avéré qu'il a généré automatiquement des circuits dont les caractéristiques de surface et de vitesse étaient similaires (dans un rapport de 3 à 5 %) à celles obtenues par un concepteur expérimenté.
Plus tard, SUN MICROSYSTEMS INC. utilisa un outil de synthèse pour concevoir les processeurs de ses stations de travail. Un ingénieur de la compagnie a estimé qu'environ 58000 portes sur les 67000 du circuit intégré avaient été réalisées par un outil de synthèse automatique. De plus, le gain en transistors obtenu par cet outil a permis à SUN de réaliser ses accélérateurs graphiques GX en cellules prédiffusées, permettant ainsi un gain en temps et en argent.

Actuellement, les outils de synthèse et les techniques d'optimisation sont souvent utilisés pour la conception de microprocesseurs. Par exemple, INTEL CORP. a investi avec succès dans un outil de synthèse pour la réalisation des processeurs I860 et 80486. Dans les deux cas, leur utilisation a permis une réduction notable des délais de conception.

Actuellement, DATAQUEST [DAT88] affirme que le marché de la synthèse logique représente le plus dynamique des marchés de la CAO. Avec un revenu d'à peine 5 millions de dollars en 1988, les prévisions de DATAQUEST estiment que ce chiffre va être porté à 115 millions de dollars en 1992. L'estimation de cette évolution est résumée par la figure I-4.

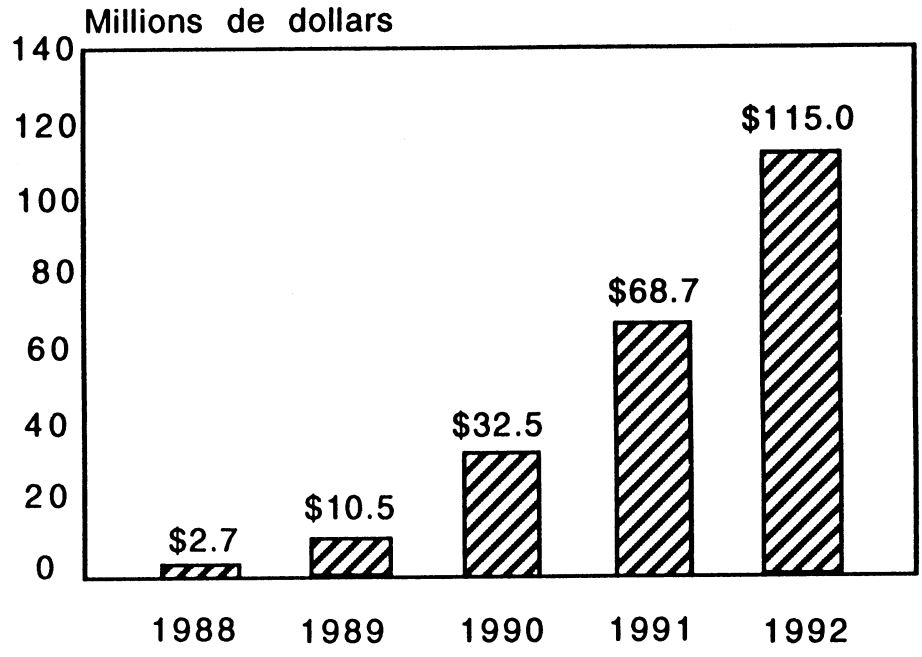


figure I-4 : marché de la synthèse logique

Les bénéfices à réaliser dans ce domaine ont donc suscité une grande effervescence et de nouvelles sociétés ont été créées pour se spécialiser dans ce domaine (Synopsis, TRIMETER, APTOR...). D'autres ont favorisé la recherche et augmenté leur potentiel de chercheurs (PHILIPS, VLSI TECHNOLOGY...).

La figure I-5, proposée par le journal IEEE-SPECTRUM, retrace quelques unes des plus importantes sociétés de ce domaine ainsi que les fonctionnalités des outils qu'elles proposent.

Compagnie	Langage d'entrée	Format de sortie	Langage supportés	Systèmes informatiques supportés	Fondeurs
Mentor Graphics	finite st. mach; hdl; netlist; schematique	netlist; schematique	VHDL	Apollo 3000, 4000, 10000	Texas Ins.
Silc. Techn.	Finite-state machine;netlist hdl	netlist schematique test vect.	vhdl SilcSyn hdl	Apollo 3000 4000 SUN 3,4,sparc	NCR
Synopsis	fsm;netlist hdl;schemat. high level	netlist schem.	verilog; vhdl	Apollo 4000, 10000 SUN; DEC 3100	fujitsu;harris lsi logic; motorola...
VLSI Techn.	fsm;bubble; hdl;netlist	layout;netl. schem.; vect. test	ELLA;Verilog; VHDL;	Apollo 3-4000 DEC 3100 VAX HP9000 SUN	Harris,Hitachi VLSI Techn

figure I-5 : principaux outils de synthèse logique commercialisés

Sur la figure précédente, un des aspects de la synthèse logique est résumé, celui de l'interface-utilisateur. L'interface-utilisateur (langages supportés, systèmes informatiques, fondeurs,...) est un paramètre souvent déterminant dans l'évaluation d'un outil de synthèse puisqu'il est lié à la facilité d'utilisation du logiciel. D'un autre côté, il y a la partie algorithmique de la synthèse qui, elle, doit répondre aux exigences de qualité requises par le concepteur.

C'est dans ce deuxième aspect de la synthèse que s'est aiguillée cette thèse.

II. Synthèse de Contrôleurs

A. Méthodologie

Introduction

La très haute intégration actuelle (million de transistors) des circuits sur silicium requiert une démarche de conception structurée [SAU79], [TRI81]. L'utilisation d'une telle approche a pour but de diminuer la probabilité d'erreur de conception et de rendre le circuit plus facilement vérifiable en fin de conception, en fin de fabrication, et d'assurer un temps de conception très réduit.

La stratégie utilisée est en général une démarche de conception descendante largement inspirée des méthodes utilisées en informatique (logiciel et matériel) et en automatique. Le principe consiste à passer d'une spécification de haut niveau à une spécification de niveau inférieur explicitant une ou plusieurs fonctions en terme de primitives plus simples jusqu'à arriver aux primitives de base.

On peut distinguer en général, la conception architecturale, la conception logique, la conception topologique et électrique.

La conception architecturale d'un circuit intégré spécifique composé d'une partie contrôle et d'une partie opérative ou chemin de donnée se fera en trois étapes :

- choix de l'algorithme de calcul,
- choix du chemin de donnée,
- choix de la structure de contrôle.

Après avoir décrit les fonctions du circuit sous forme d'un algorithme de calcul, le concepteur définit la structure du chemin de donnée à savoir les éléments de mémorisation, les opérateurs qui sont les primitives fonctionnelles de base et les structures de communication qui permettront de réaliser cet algorithme.
Ensuite, l'algorithme de calcul est décrit ou "interprété" en terme de primitives de ce chemin de donnée.
En pratique, cette approche conduit à décomposer le circuit en associant une partie opérative et une partie contrôle (figure IIA-1).

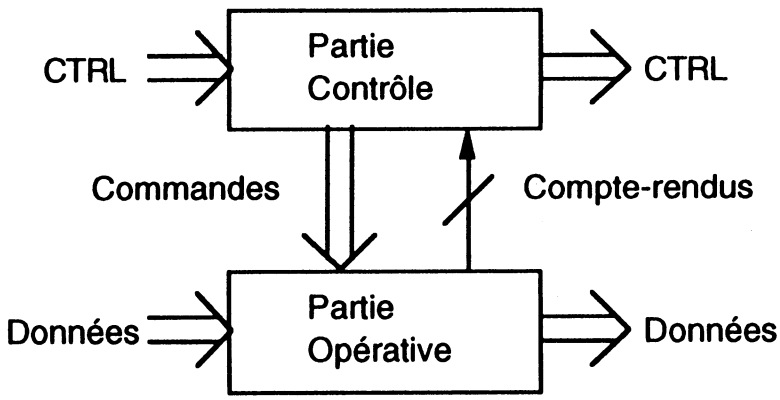


figure IIA-1 : architecture de circuits

La partie opérative est constituée d'un ensemble de registres et d'opérateurs interconnectés. Les variables de l'algorithme stockées dans des registres internes de la partie opérative (PO) sont modifiées par l'activation d'opérateurs. Les ordres d'activation de ces opérateurs ainsi que le chargement des registres sont envoyés par la partie contrôle. La partie opérative renvoie à la partie contrôle des comptes-rendus d'exécution.

La partie contrôle gère le séquencement des opérations dans la partie opérative. Elle peut être représentée par un automate dont les sorties

(associées aux états ou aux transitions) correspondent à des ordres d'activation des opérateurs. Les entrées de la partie contrôle sont les comptes-rendus de la partie opérative, l'horloge utilisée comme référence temporelle ainsi que les signaux venant d'unités externes (commande de mise en route, d' arrêt ou d'attente) susceptible d'influer sur le séquencement des opérations au sein du circuit.

Dans la suite de l'étude, nous nous intéresserons plus particulièrement aux problèmes posés par la conception, l'optimisation et la réalisation de circuits tels que les parties contrôle de circuits.
Leur fonctionnement est généralement représenté par un organigramme de contrôle interprété analogue à celui de la figure IIA-2.

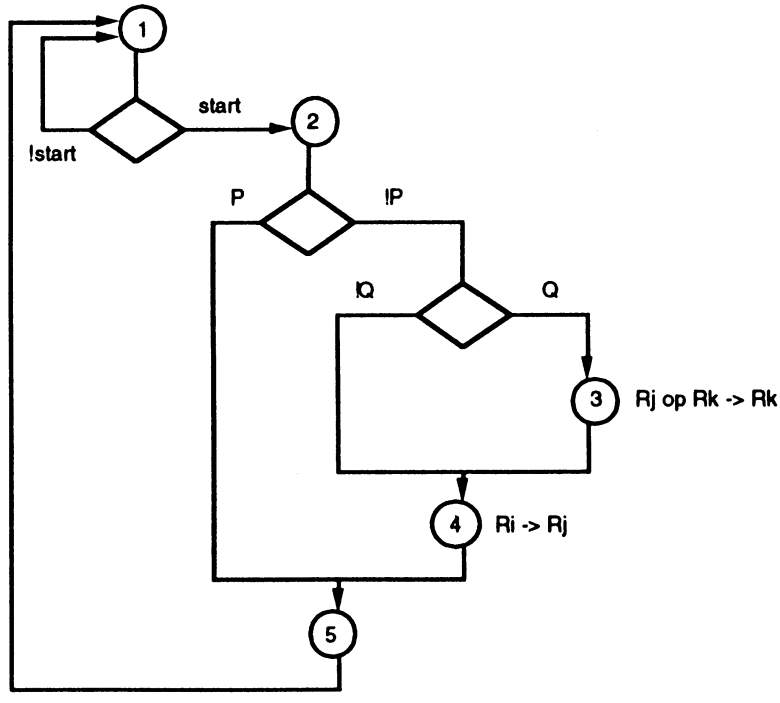


figure IIA-2 : organigramme de contrôle

Sur les noeuds de cet organigramme sont représentés les micro-opérations et les transferts de registre. Des tests ou prédicats sur des variables d'entrée conditionnent les transitions.
Cet organigramme peut être transformé sous forme de graphe de contrôle (figure IIA-3).

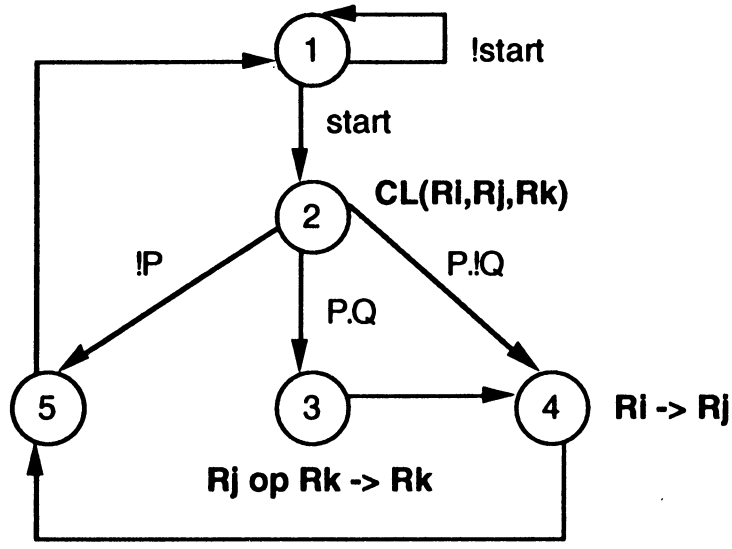


figure IIA-3 : graphe de contrôle

Cette transformation s'effectue par compaction des tests d'entrée qui deviennent des prédicats sur les transitions. Une transition entre deux états correspond aux chemins existants entre les états correspondants de l'organigramme. A un chemin est associé le produit des prédicats portés par les arcs de l'organigramme le constituant. A un ensemble de chemins est associé une somme de prédicats.

Aux noeuds sont associés les commandes émises pour déclencher les transferts élémentaires. Ainsi, Ri -> Rj nécessite l'émission de deux commandes : O_{Ri} (lecture du registre Ri) et I_{Rj} (écriture dans Rj). dont le

synchronisme n'est pas spécifié ; ceci permet de définir une machine d'états finis (figure IIA-4).

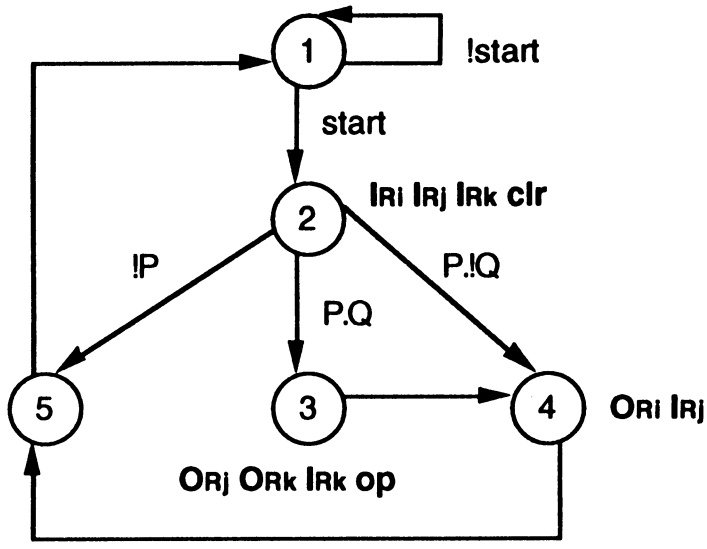


figure IIA-4 : machine d'états finis

Le graphe de contrôle obtenu définit une machine d'états finis.

1. Modélisation d'une machine d'états finis

Un circuit séquentiel est défini mathématiquement comme une machine ou un automate d'états finis (figure IIA-5).

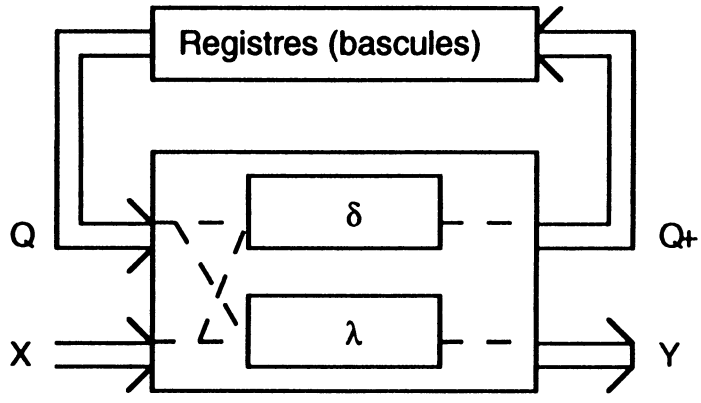


figure IIA-5 : machine d'états finis

L'implémentation physique de machines d'états finis peut être décomposée en 2 parties principales : une partie combinatoire et une partie séquentielle utilisant des éléments de mémorisation tels que les bascules. Pour des raisons pratiques, nous n'utiliserons que des bascules D synchronisées par une seule horloge comme éléments de mémorisation.
La partie combinatoire quant à elle sera divisée en un circuit combinatoire réalisant l'implémentation des fonctions destinées à calculer la valeur des états suivants et en un circuit combinatoire réalisant le calcul de la valeur des sorties.

Définitions mathématiques :

Une machine d'états finis est formellement définie par cinq grandeurs mathématiques $\{I,O,S,\delta,\lambda\}$ représentant :

$I = \{i_1, i_2, ..., i_{|i|}\}$: l'ensemble des variables d' entrées primaires,

$O = \{o1, o2, ..., z_{|o|}\}$: l'ensemble des sorties primaires,

$S = \{s1, s2, ..., s_{|s|}\}$: l'ensemble des états internes.

Les états suivants des états courants S sont notés S^+.
Les fonctions d'états suivants et de sorties du contrôleur sont données par les relations :

$\lambda : (S * I) \text{ ----> } Y$

$\delta : (S * I) \text{ ----> } S+$

Ainsi, l'utilisation de ces fonctions permet de décrire l'automate à l'instant t de la manière suivante :

$O(t) = \lambda(S(t) , I(t))$

$S(t+1) = \delta(S(t) , I(t))$

L'étude des machines séquentielles a débuté par les travaux de D.A. HUFFMAN [HUF54] et E.F. MOORE [MOO56]. HUFFMAN publia la première étude systématique de ces machines alors que MOORE donna une formulation plus abstraite et commença l'étude formelle de ces machines. Dans l'approche de MOORE, les sorties de ces machines dépendent de l'état de la machine. Ceci fut généralisé par G.H. MEALY [MEA55] qui différencia deux types de machines.

1.1. Machine de MOORE

Elle permet de décrire un contrôleur dont les sorties ne dépendent que de l'état courant (figure IIA-6).

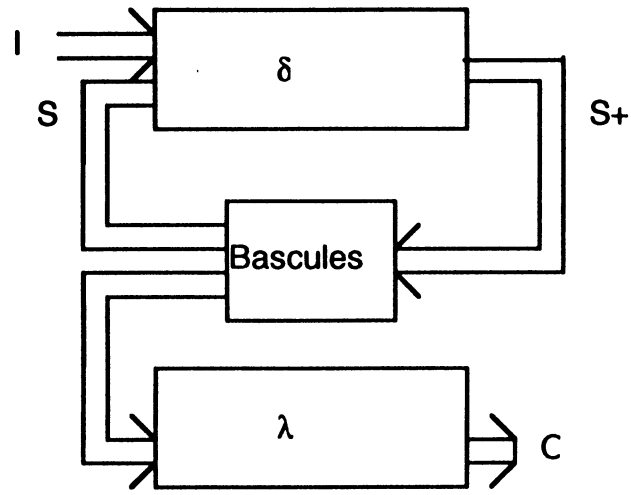


figure IIA-6 : automate de MOORE

Son fonctionnement à l'instant t est régi par les équations suivantes :

$$O(t) = \lambda\ (S(t))$$

$$S(t+1) = \delta\ (S(t)\ ,\ I(t))$$

1.2. Machine de MEALY

Cette machine est plus générale que le modèle précédent puisque ses sorties dépendent à la fois de l'état courant et des entrées (figure IIA-7).

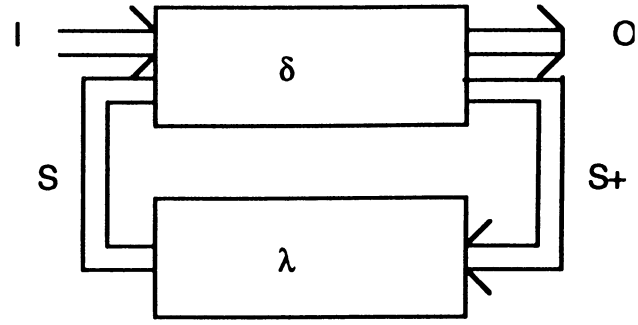


figure IIA-7 : machine de MEALY

Son fonctionnement à l'instant t est le suivant :

$$O(t) = \lambda\ (S(t)\ ,\ I(t))$$

$$S(t+1) = \delta\ (S(t)\ ,\ I(t))$$

2. Représentation d'une machine d'états finis

Deux principales représentations fonctionnelles de la machine d'états finis sont généralement utilisées.

Graphe de contrôleur :

Comme montré précédemment, ce graphe est obtenu à partir de l'organigramme de contrôle défini par le concepteur. Elle décrit le contrôleur sous forme de graphe de transitions G(V,E,W(E)) (figure IIA-8) où :

- V représente l'ensemble des sommets du graphe,
- E l'ensemble des arcs reliant ces sommets entre-eux,
- un arc (vi,vj) relie les 2 sommets vi et vj s'il existe une transition entre vi et vj ; W(E) représente le prédicat d'entrée de la transition.

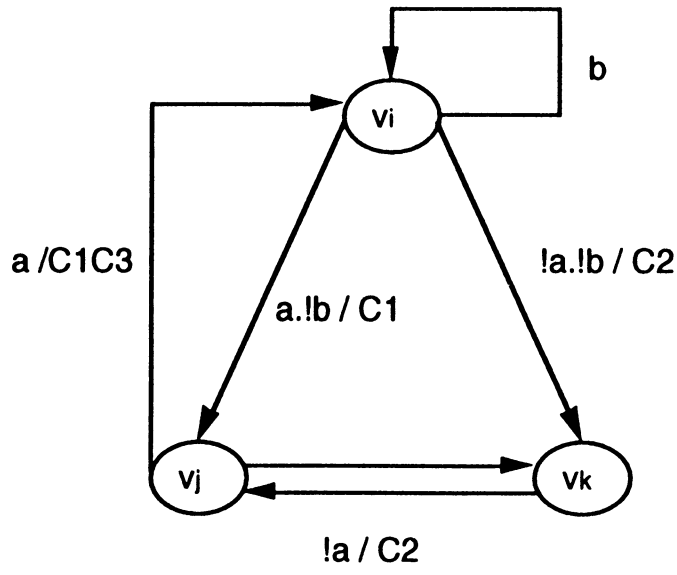


figure IIA-8 : graphe de transitions

Table de transitions :

Cette représentation consiste en une table de transitions des états T(I,S,O) où I définit l'ensemble des entrées primaires, S l'ensemble des états et O l'ensemble des sorties primaires (figure IIA-9).

I	S	S'	O
-1	vi	vi	000
10	vi	vj	100
00	vi	vk	010
1-	vj	vi	101
0-	vj	vk	010
--	vk	vj	000

figure IIA-9 : table de transition des états

Dans cette représentation fonctionnelle seuls les états n'ont pas encore une forme booléenne ; ils ont une représentation symbolique. Les vecteurs d'entrée et de commande ont une forme booléenne utilisant le "1" logique, le "0" logique et la valeur non définie "ø" dans le cas où la variable correspondante est non définie. Par exemple, dans le cas d'un système à trois variables (a, b et c), le monôme "a.b'" sera représenté par "10-".

3. Conformité de la description

La première étape de l'outil de synthèse est destinée à vérifier si la spécification du contrôleur ne contient pas d'ambiguïté.
Trois types de vérification sont initialement effectuées. Deux d'entre elles sont dites essentielles car elles conditionnent la fonctionnalité du contrôleur : évolution parallèle à partir d'un état et blocage sur un état.
La troisième est donnée à titre d'avertissement au concepteur car, dans certains cas, elle peut avoir été intentionnellement définie : absence d'état "RESET" dans la spécification initiale ou impossibilité d'accéder à un état à partir du "RESET".

3.1 Evolution parallèle

Une première vérification formelle est réalisée à partir de la spécification initiale. Elle consiste à vérifier que pour une configuration d'entrée donnée chaque état possède un unique état suivant.

Dans la figure IIA-10, la condition (a=1 et b=0) entraîne une évolution indéterminée du contrôleur à partir de l'état "s1".

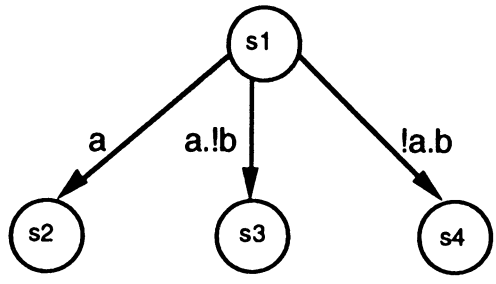


figure IIA-10 : évolution parallèle

Dans notre outil de synthèse, l'ensemble des évolutions pour chaque état du graphe est analysé et une exclusion des prédicats de sortie est créée si le concepteur ne l'a pas spécifié lui même.

Considérons l'exemple partiel de la figure IIA-11 ou à partir d'un état s_0 on peut atteindre les états s_k ($k \in \{1..n\}$) avec les conditions respectives i_k

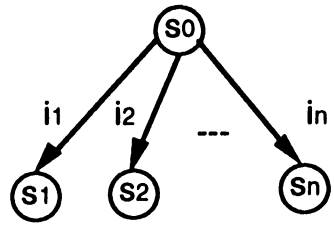


figure IIA-11 : exemple partiel (généralisation)

Les évolutions parallèles sont éliminées en restreignant chaque condition à son intersection avec le complément de la somme des conditions précédentes de telle sorte qu'elles soient mutuellement exclusives. Cela confère une priorité lors de la définition du graphe. Les conditions i_k sont modifiées en i'_k de la manière suivante :

$i'_1 = i_1$

$i'_2 = i_2 \,.\, !i'_1$ (où $!i'_1$ représente le complément de i'_1)

$i'_3 = i_3 \,.\, !(i'_1 + i'_2)$

(...)

$i'_n = i_n \,.\, !(\Sigma[k=1..n-1]\ i'_k)$

3.3 Blocage sur un état

Cette vérification peut s'effectuer en même temps que la précédente. Son but est de vérifier que quel que soit l'état dans lequel se trouve la machine à un instant donné, toute condition d'entrée permet d'évoluer vers un nouvel état (qui peut être le même en cas de bouclage). Cela revient à vérifier que la somme de l'ensemble des conditions en aval d'un état forme une tautologie.

L'exemple de la figure IIA-12 illustre une situation de blocage puisque la condition d'entrée (a=1 et b=1) ne permet pas d'évoluer de l'état "s1" vers un autre état. Il y a risque de blocage sur l'état "s1".

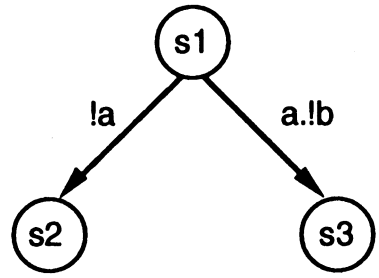
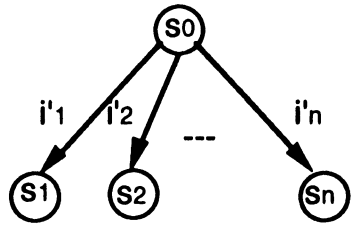


figure IIA-12 : blocage sur un état

Dans le cas ou la somme des conditions en aval d'un état ne forme pas une tautologie (Σi_k <> 1), un arc supplémentaire est alors rajouté permettant à la machine de rester sur ce même état ; il y a création d'un bouclage sur cet état.

Ainsi, si le graphe partiel initial a la forme suivante (figure IIA-13) :



$\Sigma i'_k <> 1$

figure IIA-13 : blocage sur état (généralisation)

La condition associée à cet arc (i_{n+1}) représente le complément à l'ensemble des conditions en aval i_k permettant ainsi d'obtenir une tautologie de telle sorte qu'il existe toujours une transition activable quel que soit le vecteur d'entrée (figure IIA-14) : $i_{n+1} = !(\Sigma i'_k)$

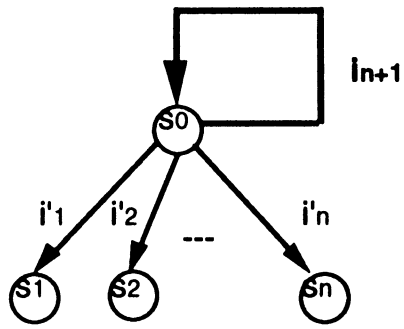


figure IIA-14 : bouclage sur état

3.3 Etat "RESET"

Lorsqu'un état "RESET" est spécifié par le concepteur, il est vérifié que tout état de l'automate est accessible à partir de cet état "RESET".

Néanmoins, la présence de l'état "RESET" n'est pas indispensable au bon fonctionnement de l'automate car un état "RESET" implicite peut être atteint grâce à des séquences de synchronisation.

4. Codage des états

Le codage des contrôleurs consiste à affecter de manière injective un code binaire à chaque état. De ce code dépendront la complexité des équations des variables internes et des sorties et la surface de la cible finale. L'optimisation du codage consiste à choisir un code menant ultérieurement à des équations simples des variables internes et des sorties (cette complexité s' évalue en terme de littéraux présents dans ces équations) et plus objectivement à un circuit final optimisé en surface et performance. Pour ce dernier point, une dépendance par rapport à la cible est évidente.

Deux types de codage sont généralement utilisés :

a) Codage un parmi n :

Dans cette méthode le code courant est représenté par une variable n valuée où n représente le nombre total d'états dans le graphe. Une variable n valuée V^n est caractérisée par un ensemble de n variables booléennes simples v_1, v_2, ..., v_n physiquement représentées par n points de mémorisation. Elle prend n valeurs possibles de 1 à n de la manière suivante :

$$V^n = |i| <=> v_i = 1 \text{ et } \forall\ j<>i\ v_j = 0$$

b) Codage compact :

Dans cette approche, les n états du contrôleur sont caractérisés par des codes binaires représentés par p variables d'états Y1, Y2,..., Yp
où $p \geq \max(\log(n))$.

D'une manière générale, le codage un parmi n permet d'obtenir des circuits plus rapides car il y a moins de logique de décodage. Par contre, il a le désavantage d' être très couteux en surface de silicium.

5. Génération des équations

Lorsqu'un code binaire unique est affecté à chaque état du contrôleur, il est alors possible de générer les équations de commande et de séquencement.

Si le code de l'état si est note c(Si) et si les variables d'états correspondant aux n bits du code sont notées y1, y2,..,yn, alors le monôme de caractérisation associé à l'état Si est défini comme suit :

$$m(Si) = \Pi_{[Yi(S)=1]} Yi \; . \; \Pi_{[Yj(S)=0]} Y'j$$

où Yi(S) représente la valeur du ième bit du code de l'état S.

Exemple :

Le monôme de caractérisation associé à l'état S dont le code binaire sur 4 bits est c(S)=0111 est égal à $y'_1y_2y_3y_4$

a) Equations de commande :

Leurs formulations différent selon que l'on synthétise un contrôleur de type MOORE ou MEALY.

Contrôleur de MOORE :

A chaque état "si" est associé un ensemble de commandes Zi. Chaque équation de commande zk se calcule de la manière suivante :

$$zk = \Sigma[i/zk \; \varepsilon \; Zi] \; c(si)$$

Contrôleur de MEALY :

Pour ce type de contrôleur, les commandes Zi sont associées aux transitions entre les états (si,sj) conditionnés par cij.

Chaque équation de commande zk s'obtient par :

$$z_k = \Sigma_{[i/zk \ \varepsilon \ Zi]} \ c(si) \ . \ cij$$

b) Equations de séquencement :

Les équations de séquencement ou des variables d'états dépendent du type de bascules utilisées pour la mémorisation de l'état. Les bascules sont des circuits de mémorisation élémentaires qui permettent de stocker une information binaire de un bit ; elles ont donc deux états : 0 ou 1. Une bascule a deux types d'entrée :

- des entrées qui autorisent la modification de l'état ; à partir de ces entrées est calculée la fonction d'activation,
- des entrées qui indiquent la modification de l'état à effectuer (fonction d' évaluation).

Les équations de séquencement pour les bascules D sont données.

Bascules D :

Ces bascules ont une entrée d'évolution D ; l'entrée D est recopiée dans la bascule si celle-ci est active (figure IIA-15).

D	Q	Q+
1	0	1
0	1	0
1	1	1
0	0	0

figure IIA-15 : table d'évolution d'une bascule D

Q (resp. Q+) représente l'état de la bascule avant (resp. après) échantillonnage de ses entrées.

L'équation de la variable interne Yk se calcule de la manière suivante :

$$D\_Yk = \Sigma\ [(si,sj)\ \varepsilon\ Ak(0->1)\ U\ Ak(1->1)]\ c(si)\ .\ cij$$

6. Synthèse de contrôleurs

La synthèse de contrôleurs se fait selon l'organigramme suivant (figure IIA-16) :

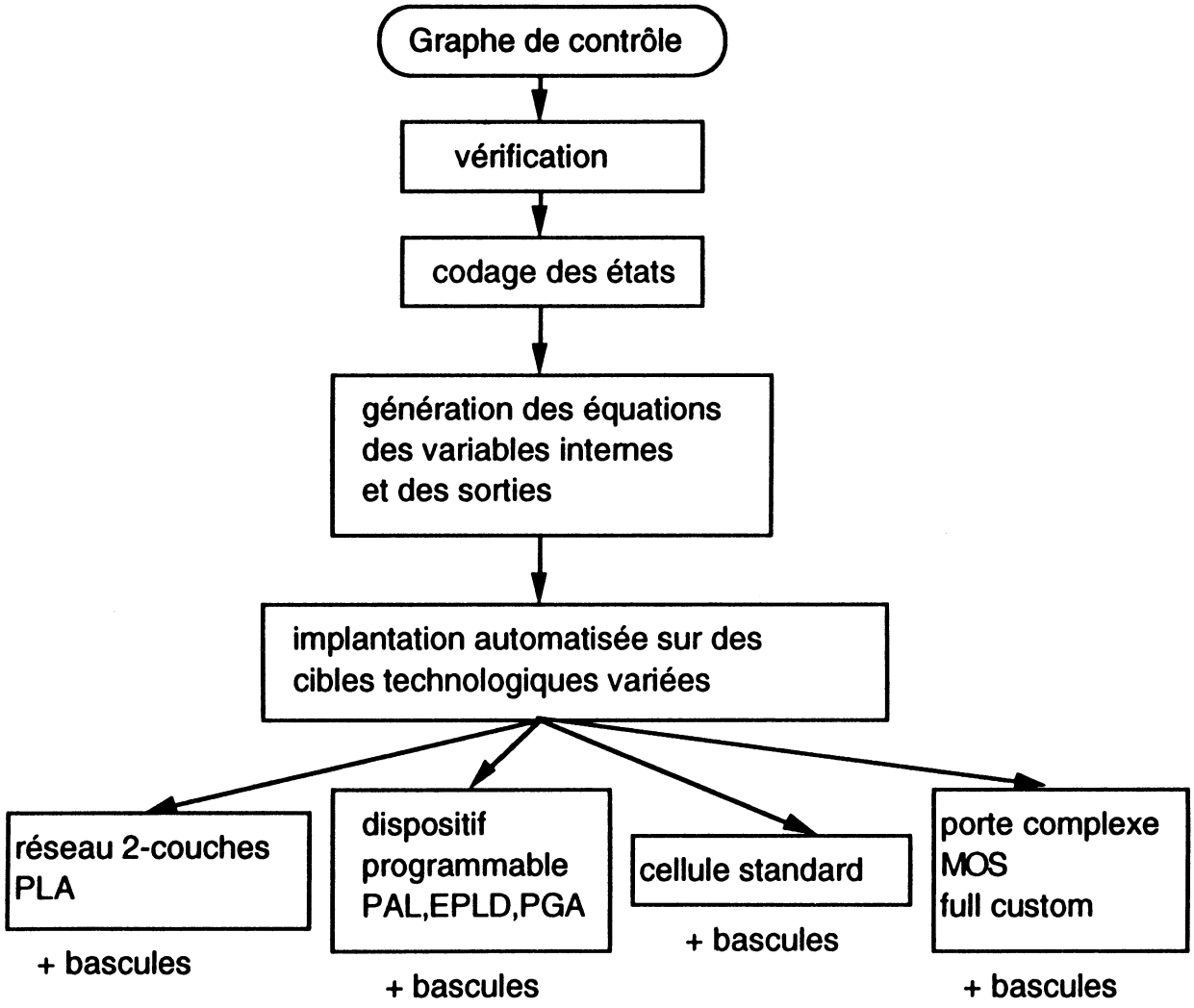


figure IIA-16 : synthèse de contrôleurs

Dans la partie suivante, un algorithme de codage des états sera présenté. Cette étape est fondamentale lors de la synthèse des contrôleurs car elle influe de façon importante sur la qualité du circuit final.

B. Codage des Etats

Introduction

Les contrôleurs ou automates de contrôle de circuits spécifiques intégrés sur silicium utilisés dans l'industrie deviennent de plus en plus complexes. Les méthodes de synthèse automatisée permettant d'obtenir ces contrôleurs sont un point névralgique des outils CAO car elles permettent de réduire de près de 40% la surface d'implémentation et d'optimiser les performances [POI89].

Le codage des états du contrôleur est l'étape clef de cette synthèse. Il consiste à affecter un code binaire à chacun des états. Dès ce codage effectué, les équations de sortie et des variables internes sont générées automatiquement en tenant compte du type d'éléments de mémorisation désiré (bascules D, RS, JK...).
Ces équations sont ensuite optimisées suivant la cible technologique. Pour un PLA, on pourra utiliser un outil de minimisation de type ESPRESSO [BRA84] ou ASYL [SIC88] afin de réduire le nombre de monômes nécessaires à la réalisation des fonctions logiques.

Dans certains cas, l'architecture PLA ne satisfait pas les contraintes de temps ou de surface imposées par le concepteur. Une architecture multi-niveau s'avère nécessaire. Les équations booléennes sont alors factorisées et exprimées en terme de primitives de cellules de bibliothèque ; cette dernière phase s'appelle un "mapping" technologique. Le terme de "mapping" étant très utilisé, nous nous sommes permis de l'employer tel quel tout au long de cette thèse.
De nombreux algorithmes de factorisation ont été proposés [BRA84], [BRA87], [BAR86] ...

L'efficacité des algorithmes de factorisation est évaluée soit en terme de gain de littéraux, soit en terme de gain en surface pour une technologie donnée.

En particulier, dans le cas d'une implémentation en logique 2 couches (PLA) cette complexité se réduit au nombre final de monômes utilisés pour exprimer les fonctions.
Dans le cas d'une implémentation en logique multi-couches, cette complexité sera évaluée après placement et routage en terme de surface et de vitesse (cycle maximal de l'horloge).

La phase de codage est donc très importante car elle va permettre de préparer les minimisations ultérieures des variables internes et des commandes. Elle est basée sur des méthodes prédictive de minimisation et de factorisation.

Le problème du codage est traité différemment selon le type d'éléments de mémorisation ou bascule utilisé. L'algorithme implémenté est volontairement restreint à une synthèse utilisant des bascules D puisque c'est le type le plus fréquemment utilisé par les concepteurs de circuits.

1. Etat de l'art

La littérature est abondante et relativement ancienne dans le domaine de la recherche sur le codage des états des contrôleurs.

Dans les années 60-70, deux raisons principales ont motivé cette recherche. La première était d'éviter les courses critiques dans les circuits séquentiels asynchrones. Les méthodes de codage proposées dans [UNG63], [ARM62], [LIU63] et [SAU68] assuraient la non-existence des courses critiques. La deuxième raison motivant ces travaux était la simplification ultérieure des équations des variables internes et des sorties [HAR66].

Un point crucial fut clairement soulevé, à savoir le compromis inévitable entre le nombre de variables internes et la simplicité des équations logiques [SAU72]. Du point de vue théorique, deux approches peuvent être identifiées.
Une approche est fondée sur la théorie des partitions. A chaque variable interne est associée une partition de l'ensemble des états : les états pour lesquels cette variable est égale à 0 et les états pour lesquels cette variable est égale à 1 ; ceci correspond à une bissection de l'hypercube. Cette approche a conduit à la théorie des partitions bien connue de HARTMANIS [HAR66]. A toute partition de l'ensemble des états présents de l'automate correspond pour chaque entrée une partition de l'ensemble des états successeurs (paire de partitions). L'etude de ces paires de partitions a permis d' appréhender la complexité des variables internes et des sorties. Ceci a conduit à des méthodes de codage effectuées progressivement, variable par variable, en choisissant chaque fois la bissection du cube. Ceci est appelé codage par colonne ou ("column encoding").

En parallèle, une autre approche a été proposée, fondée sur la théorie du treillis de BOOLE [SAU72]. Un hypercube, N-cube ou treillis de BOOLE est isomorphe au treillis de l'ensemble des parties de N éléments structure par la relation classique d'inclusion. Affecter un code binaire à un état consiste à assigner à cet état un sous-ensemble de ces N éléments. Le codage est alors ramené à un placement progressif des sommets dans le N-cube. Cette approche appelée encore codage par ligne ("row-encoding") a permis de meilleures optimisations du codage en nombre de variables internes et en simplicité des équations.

Avec les progrès des compilateurs de silicium, toutes ces approches furent formulées à nouveau afin de produire un outil efficace de codage d'états en tenant compte des nouvelles techniques de minimisation pour les différentes cibles technologiques.

En ce qui concerne les PLAs, DE MICHELI [DEM83], [DEM85] a proposé une méthode de codage et minimisation symbolique. Cette méthode qui s'appuie sur les techniques classiques de minimisation deux couches, consiste à affecter un codage un parmi n aux états et à minimiser le PLA ainsi obtenu. Cette minimisation met en évidence des ensembles d'états (groupe d'adjacence) qui sont ensuite affectés à une même face de l'hypercube.
La phase d'immersion utilise l'approche de codage par colonnes. Les noeuds appartenant à une même situation sont placés sur une même face définie par des valeurs fixes d'une ou plusieurs variables internes : cela correspond à une bissection de l'hypercube.
Cette méthode fut étendue par DE MICHELI [DEM86] au codage des entrées. Cependant, cette méthode a le désavantage de conduire à un nombre très important de variables internes pour les exemples complexes, ce qui

implique un temps de calcul important, voire une explosion en complexité pour minimiser cette représentation deux couches.

En ce qui concerne une cible multi-couches, on cherche à produire une expression factorisée facilitant le "mapping" technologique ultérieur.
Ce problème fut traité dans Mustang [DEV87] ou le placement des états ayant même successeur dans un proche voisinage de l'hypercube entraîne des facteurs communs entre les monômes correspondants à ces états.
Néanmoins, la phase de reconnaissance de situations reste très sommaire. Elle consiste soit à regrouper les états émettant la même commande ou ayant le même état suivant (*fanout oriented*), soit à regrouper les états suivants dont les transitions en amont sont libellées par un même littéral d'entrée (*fanin oriented*). Il a été montré d'autre part, que le critère de minimisation de distance peut conduire à minimiser le nombre de littéraux des expressions finales mais pas à réduire effectivement la surface globale du circuit. Le résultat global est en effet pénalisé par un mauvais contrôle de la connectique introduite au cours de la factorisation [SAU90].

Parallèlement, d'autres architectures nécessitent de nouvelles heuristiques de codage. AMANN [AMA87] notamment a ajusté ses heuristiques pour des architectures où les éléments de mémorisation sont montés en compteurs.

Les méthodes classiques de codage des états d'un contrôleur comportent généralement trois étapes essentielles :

- une première étape de reconnaissance de situations dans le graphe de contrôle établit des contraintes sur les codes de certains groupes d'états,

- une deuxième étape utilise une méthode de placement des noeuds dans l'hypercube satisfaisant au mieux ces contraintes,
- finalement, une dernière étape affecte un code binaire aux états du graphe.

Nous allons donc successivement considérer ces 3 étapes pour la méthode que nous proposons.

2. Reconnaissance de situations et groupes d'adjacence

La première étape de l'algorithme de codage recherche des situations dans le graphe de contrôle pouvant amener des simplifications ultérieures.
Un ensemble de règles basées sur l'identification de ces situations permet d'obtenir un jeu de contraintes sur les codes des états. Le respect de ces contraintes permettra de générer des équations logiques de séquencement et de commandes mieux adaptées aux opérations de minimisation ultérieures.
Ces situations ou règles sont basées sur la prédiction des minimisations futures des équations logiques. Leur exploitation permettra en priorité de fusionner des monômes des équations logiques, puis, de créer des facteurs communs à l'intérieur d'une fonction et entre fonctions et ainsi de minimiser l'implémentation finale après une synthèse multi-niveaux.

A partir de la description du contrôleur donnée par le concepteur, la première étape consiste à rechercher les situations clefs induisant des contraintes entre les codes des états et à générer un ensemble de groupes d'adjacence. Un groupe d'adjacence regroupe les états impliqués dans ces situations. On montrera que le fait d'affecter une face à ces états ou à les placer à proche distance dans l'hypercube simplifie les équations.

2.1 Règles et groupes d'adjacence

Quatre règles de base ont été identifiées : état à entrance multiple, états ou transition émettant les mêmes commandes, états à sortance multiple et transitions étiquetées par des entrées identiques.

Les équations logiques ultérieurement générées seront représentées par Yi (i ε {1..|s|}) pour les équations de séquencement et par Oj (j ε {1..|o|} pour les commandes.

a) Règle 1 : Etats à entrance multiple

Cette situation fréquente dans les graphes de contrôle a été largement décrite dans la littérature. Des branchements génèrent plusieurs chemins qui convergent de nouveau vers des points clefs du graphe d'états. Les arcs qui convergent vers un même état sont généralement étiquetés par une même condition (figure IIB-1).

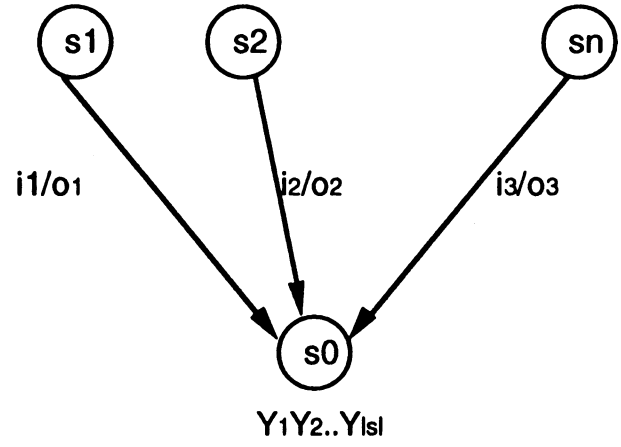


figure IIB-1 : entrance multiple

Les monômes obtenus pour ces situations sont les suivants : pour chaque variable interne égale à 1 dans le code de l'état jonction, la somme des code des états prédécesseurs est générée. Dans le cas de l'exemple de la figure IIB-1, l'équation de chaque variable interne égale à 1 est :

$$Yi = i_1.c(s_1) + i_2.c(s_2) + .. + i_n.c(s_n)$$

Dans le cas où les conditions d'entrée sont identiques (i), l'expression devient :

$$Yi = i*(c(s_1) + .c(s_2) + .. + c(s_n))$$

Les états en amont constituent un groupe d'adjacence. La meilleure solution pour des minimisations futures sera de les mettre sur une même face du cube. En effet, dans ce cas et ce cas seulement, plusieurs monômes seront fusionnés en un seul ; ce monôme correspond à la composition de la partie invariante des codes des états amonts et de la condition d'entrée i.

Exemple :

Considérons un groupe d'adjacence de 4 états {s1,s2,s3,s4} et les codes suivants :

c(s1) = 01000 ; c(s2) = 01001 ; c(s3) = 01010 ; c(s4) = 01011

Dans ce cas et pour une condition de transition "a'b", le monôme résultant pour chaque variable interne égale à 1 dans le code de l'état suivant sera :

$\sim Yi = a'b\ Y'_1.Y_2.Y'_3$

où le prime représente la forme complémentée de la variable.

Dans le cas où les prédicats ne sont pas identiques, alors la mise sur une même face du cube des états amonts favorisera la création d'une expression commune à tous les monômes qui est égale à la partie invariante du code de ces états.

b) Règle 2 : Transitions émettant les mêmes commandes

Les équations des commandes sont égales au produit de la condition portée sur la transition par le code de l'état en amont.

Ainsi, lorsque une commande o_k est émise lors de n transitions ayant comme origine les états s_1, s_2,..,s_n, son équation est égale à :

$$o_k = i_1.c(s_1) + i_2.c(s_2) + .. + i_n.c(s_n)$$

Les états émettant les mêmes commandes constituent un groupe d'adjacence. Les effets sur les équations de sorties seront identiques à ceux de la règle précédente mais cette fois le gain obtenu ne concerne qu'une seule fonction (o_k) au lieu de l'ensemble des variables internes comme précédemment.

c) Règle 3 : Etats à sortance multiple

Cette règle concerne les équations d'états suivant du contrôleur. Par construction lors du pré-traitement initial du contrôleur, l'ensemble des transitions en aval d'un état sont exclusives et forment une tautologie. Supposons qu'a partir d'un état on accéde à 2^k états par 2^k monômes (figure IIB-2). Ces monômes définissent une partition des états successeurs. Deux états sont mis dans le même bloc si la variable apparait sous la même forme dans les deux monômes menant à cet état. L'exemple de la figure IIB-2 définit les partitions {(s_1,s_2),(s_3,s_4)} pour la première variable et {(s_1,s_3),(s_2,s_4)} pour la deuxième. Ces partitions définissent des relations d'adjacence entre les états du 2-cube.

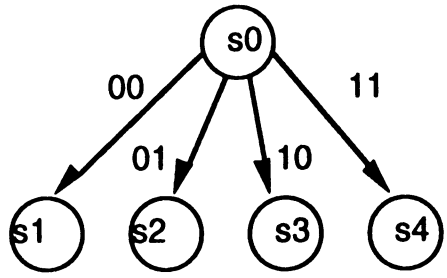


figure IIB-2 : exemple de sortance multiple

Cette situation permet de déclarer $\{s1,s2,s3,s4\}$ comme un groupe d'adjacence avec contraintes : ces états doivent être affectés sur un 2-cube défini par les partitions précédentes. Cela signifie que $\{(s1,s2),(s3,s4)\}$ et $\{(s1,s3),(s2,s4)\}$ définissent des relations d'adjacence qui doivent être respectées lors de la construction du 2-cube associé aux états successeurs. Ceci conduit à la génération d'un monôme $\{c(s0)\}$ pour toutes les variables internes invariantes des états successeurs et d'un monôme $\{c(s0)*\sim i_k\}$ pour les autres variables internes ($\sim i_k$ représente soit i_k soit son complément).

Dans le cas ou il n'est pas possible de faire des partitions selon les entrées (par exemple si les conditions sont {01-}, {000}, {001} et {1--}), un groupe d'adjacence non contraint est créé regroupant les états en aval de l'état source. Le respect de cette adjacence conduit à la génération du monôme $\{c(s_0)\}$ pour les variable internes invariantes des états successeurs.

d) Règle 4 : Transitions étiquetées par des entrées identiques

Cette situation se réfère à des travaux antérieurs sur les paires de partition [HAR66]. Une première étape consiste à chercher toutes les transitions étiquetées par un vecteur d'entrée identique. Dans l'exemple de

la figure IIB-3, quatre transitions étiquetées par la même transition i sont représentés :

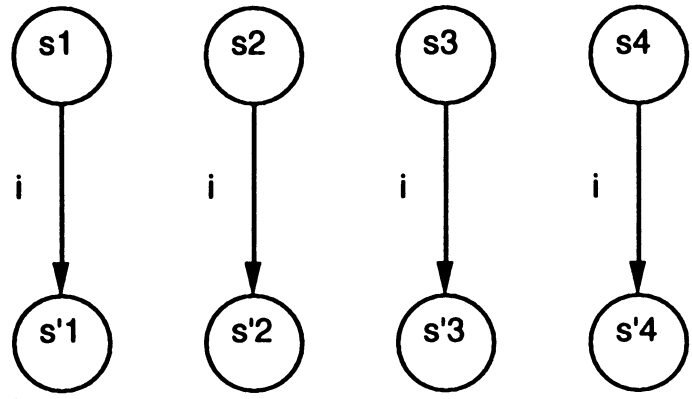


figure IIB-3 : transitions avec même vecteur d'entrée

Supposons que les quatre états origines soient affectés à un 2-cube défini par les deux partitions $\{(s1,s2),(s3,s4)\}$ et $\{(s1,s4).(s2,s3)\}$. Ces partitions des états amonts conduisent par application de l'entrée i aux partitions des états successeurs $\{(s'1,s'2),(s'3,s'4)\}$ et $\{(s'1,s'4).(s'2,s'3)\}$. Si nous codons les états destination en respectant les adjacences définies par les partitions $\{(s'1,s'2),(s'3,s'4)\}$ et $\{(s'1,s'4).(s'2,s'3)\}$, nous obtenons un ensemble de monômes spécialement intéressants pour la synthèse multi-couche.

De façon plus précise, pour les variables invariantes des codes des états successeurs, il sera généré un monôme égal à la composition du prédicat i et de la partie invariante des codes des états initiaux. Les autres variables produisent un monôme égal au produit de la partie invariante du code par la condition d'entrée et par la partie du code respectant la partition. Cette propriété est une extension de la propriété de dépendance réduite [HAR66] due au fait que les variables internes respectent les paires de partitions associées à un vecteur d'entrée i.

Une implémentation simple de l'algorithme consiste à chercher des groupes de 2^k états allant vers 2^k états successeurs et de les affecter à deux k-cubes comme défini ci-dessus. Il reste alors à ajouter les groupes permettant de respecter la partition selon les entrées.
Un cas particulier très courant de ce type de situation est celui d'ensemble de boucles étiquetées par un même vecteur d'entrée.

Remarque : représentation en machine
Les relations d'adjacence engendrées par ces règles sont représentées par une matrice d'adjacence $\mathcal{A}$ où chaque ligne représente un groupe d'adjacence et chaque colonne un état du contrôleur. Cette représentation, très compacte, permet d'avoir, d'une part, un encombrement minimal de la mémoire et, d'autre part, une manipulation performante des groupes d'adjacence.

Un exemple de matrice d'adjacence est donné en figure IIB-4 :

$\mathcal{A}$	s1	s2	s3	s4	s5
g1	1	1	0	0	1
g2	0	1	1	0	1
g3	1	0	0	0	1

figure IIB-4 : matrice d'adjacence

Le groupe d'adjacence g1 est donc composé de 3 états {s1,s2,s5}. Les manipulations entre les différents groupes se traitent aisément par des opérations binaires. Par exemple, l'intersection entre g1 et g2 s'obtient par une opération ET qui donne un nouveau groupe composé de 2 états g4 = {s2,s5}.

Ainsi, chacune de ces règles conduit à une matrice partielle d'adjacence. Cette matrice est traitée localement afin d'éliminer les groupes redondants et d'affecter à chacun des groupes un gain représentant son pouvoir de minimisation. Cette matrice partielle est alors ajoutée à la matrice d'adjacence globale.

2.2 Intérêt des contraintes cubiques

Nous allons justifier l'avantage de notre méthode de traitement des contraintes cubiques consistant à placer les états d'un groupe d'adjacence sur la même face d'un cube par rapport à une méthode tendant à minimiser les distances entre ces états [POI90]. Nous allons énoncer succinctement le principe de cette deuxième approche.

En effet, certaines techniques de codage développées récemment sont basées sur la minimisation de distance entre états d'un même groupe d'adjacence [DEV87], [LIN89]. Elles procèdent comme suit.
Tout d'abord, un graphe d'attraction est créé. A chaque situation est affecté un poids. Si deux états appartiennent au groupe d'adjacence associé à cette situation, l'arc les reliant symbolisant une attraction est pondéré par le poids de la situation. Ces poids s'ajoutent si les 2 états sont impliqués dans plusieurs situations. Deux états reliés par un arc de poids fort se trouvent attirés pendant l'immersion ce qui se traduit par une minimisation de la distance de HAMMING entre ces états.
Le codage optimal des états respectant la minimisation des distances contenues dans le graphe étant un problème NP-complet, une heuristique appelée "wedge clustering", est utilisée. Tout d'abord, le noeud v0 dont le somme des poids entrant est maximale est sélectionné ; un code lui est

affecté s'il n'a pas été précédemment codé. Ensuite, un code est attribué aux noeuds (v1,...,vn) reliés à v0 de telle sorte que la somme des distances de HAMMING entre ces états et v0, pondérée par le poids de l'attraction soit minimale :

$$\mathbf{MIN}\ (\Sigma_{[i=1..n]}\ w_{i0} \cdot \mathbf{H}\ (v0,vi))$$

où w_{i0} et H(v0,vi) représentent respectivement le poids de l'attraction et la distance de HAMMING entre v0 et vi.
Enfin, le noeud v0 ainsi que les arcs entrants sont enlevés du graphe d'attraction et l'opération d'affectation du code est réitérée jusqu'à ce que tous les noeuds soit codés.
Il apparait que la limitation majeure de cette heuristique est de ne pas considérer les attractions entre les états vij (i,j ε {1,..,n}) lorsque le codage s'effectue en fonction de v0. Ainsi, il est peu probable par cette méthode d'obtenir la solution optimale qui est de réduire le nombre de monômes à un seul ou d'obtenir le plus gros facteur commun puisqu'elle est tributaire de l'ordre de traitement des états comme nous l'avons illustré dans l'exemple suivant.

Considérons un contrôleur de 6 états qui sera codé par un codage compact utilisant 3 bits.
Pour cela, supposons que deux groupes d'adjacence aient été détectés G1= {A,B,C,D} et G2= {B,C,E,F} avec un gain de w chacun.

La figure IIB-5a représente le graphe d'attraction obtenu :

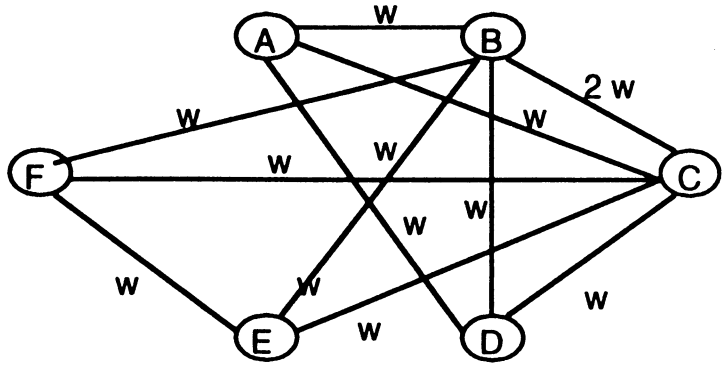


figure IIB-5a : graphe d'attirance entre états

La première étape de l'immersion consiste à rechercher le noeud dont la somme des arcs entrant est maximale ; les noeuds B et C sont candidats avec une somme des poids égale à 6w. Choisissons par exemple B et affectons lui le code 000. Le sous graphe représentant le noeud B et les noeuds qui sont attirés par lui est le suivant (figure IIB-5b) :

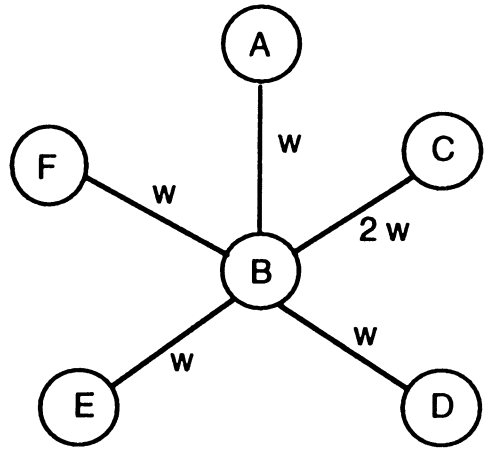


figure IIB-5b : sous graphe d'attraction du noeud B

La relation guidant l'immersion des états attirés par B est la suivante :

$$\mathbf{MIN\ (w.H(A,B) + 2w.H(B,C) + w(B,D) + w.H(B,E) + w.H(B,F))}$$

Un code adjacent à B est affecté à C afin d'optimiser la relation précédente, soit le code 100 par exemple.
C'est à cette étape que l'on remarque que les noeuds A,D,E et F sont attirés par B avec le même poids et donc l'heuristique va traiter dans un ordre quelconque ces états. Si le code 010 est attribué à A, 001 à D, 110 à E et 101 à F, la relation précédente est respectée mais on réalise que les groupes G1 et G2 ne forment pas des contraintes cubiques. Par contre, notre méthode aurait placé les états de G1 et ceux de G2 sur un cube en respectant l'arète commune (B,C).

Par cette approche globale, il est clair que les sommets d'un groupe d'adjacence ne seront pratiquement jamais placés sur une même face.

Nous allons maintenant montrer que ceci a une influence néfaste sur la surface et les performances du circuit. En d'autres termes, nous allons prouver que seul le placement sur une même face optimise de façon sure la surface et les performances.

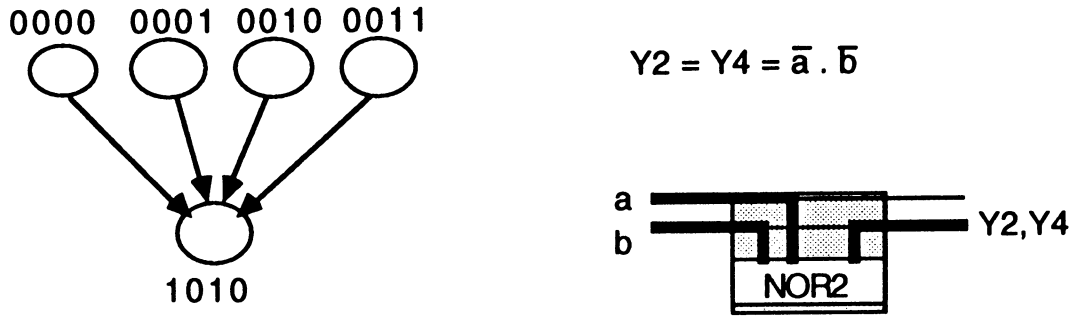
Afin d'illustrer cette comparaison, nous considérons l'exemple fréquemment rencontré dans un graphe de contrôle, d'un ensemble d'arcs qui convergent vers un même état par des conditions toujours vraies. Un groupe d'adjacence des états sources est formé lors de la reconnaissance de situations.

Cette situation produit un ensemble de monômes correspondant à la somme des codes des états source, pour chaque variable égale à 1 dans le code de l'état jonction.
Pour ces illustrations, une méthode classique de *"mapping"* technologique est utilisée à partir de cellules précaractérisées de la bibliothèque VSC010 de technologie 2 microns [VSa88].

Les figures suivantes illustrent l'influence de différentes méthodes de traitement des contraintes sur l'implémentation finale du circuit après placement et routage.
Nous avons mis en relief certains paramètres caractéristiques des circuits afin d'évaluer l'incidence du traitement des groupes d'adjacence :

- nombre de monômes et de littéraux dans les expressions factorisées,
- nombre de transistors et de cellules précaractérisées du circuit,
- somme des distances de HAMMING des codes des états pris 2 à 2,
- nombre de canaux de routage,
- performance du circuit (en ns) pour une capacité de sortie de 1pF,
- surface totale (avec routage) du circuit en $K\mu^2$.

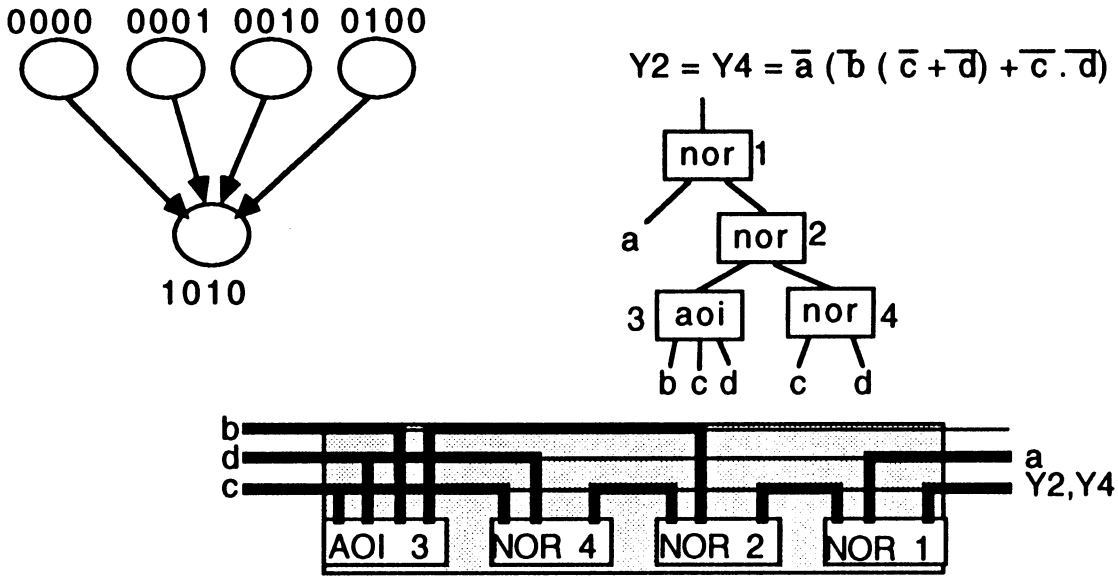
La première figure (figure IIB-6) montre la solution optimale de traitement des contraintes consistant à mettre sur la même face d'un cube les états appartenant à un même groupe d'adjacence.



monomes	littéraux	transistors	cellules	Σdistance	routage	vitesse	surface
1	2	4	1	8	2	0.88	1.41

figure IIB-6 : contrainte cubique

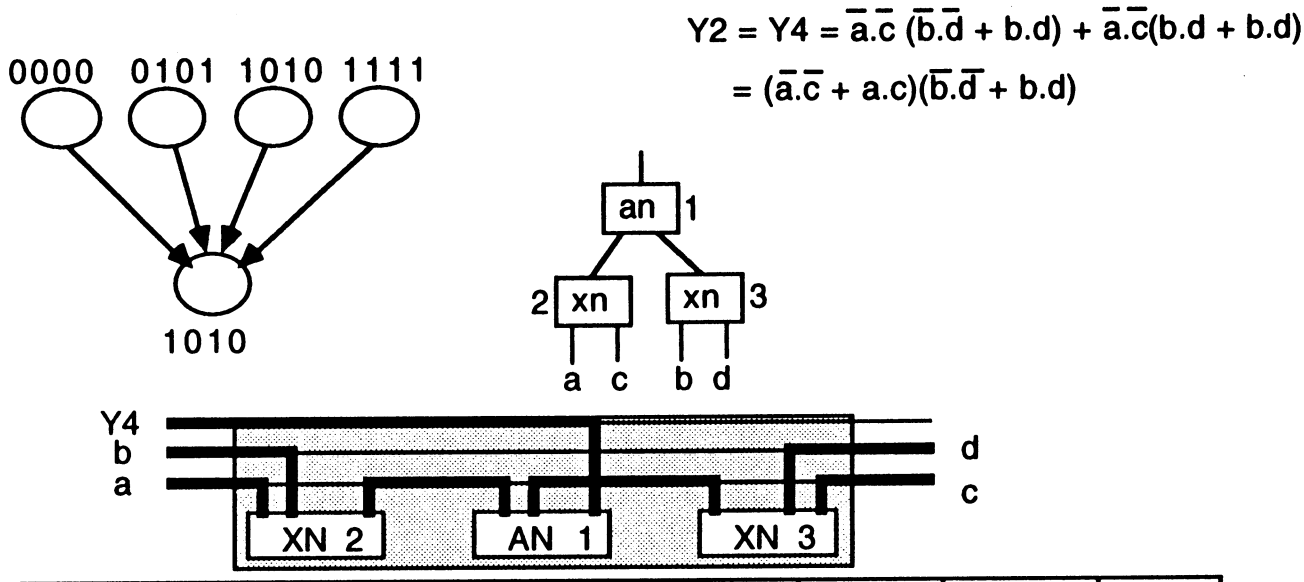
La deuxième figure (figure IIA-7) montre l'influence de la méthode de traitement des distances minimisant les distances.



monomes	littéraux	transistors	cellules	Σdistance	routage	vitesse	surface
3	6	18	4	9	3	15.81	6.79

figure IIB-7 : minimisation des distances

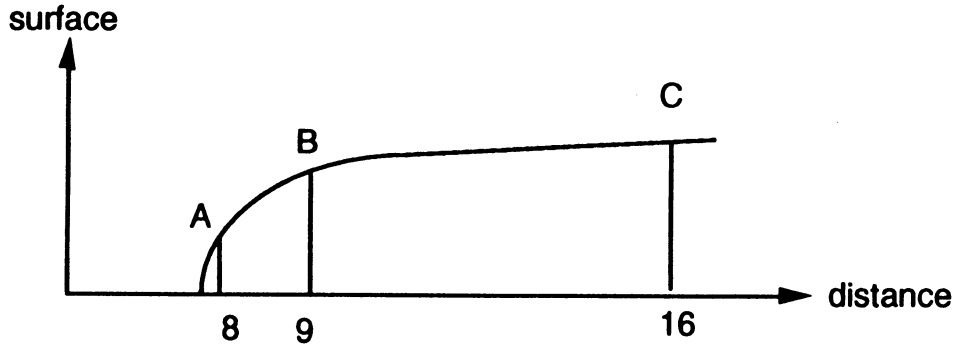
Finalement, un compromis entre un codage aléatoire et un pseudo pire cas a été établi à titre de comparaison (figure IIB-8).



monomes	littéraux	transistors	cellules	Σdistance	routage	vitesse	surface
4	8	22	3	16	9	9.2	7.31

figure IIB-8 : pire cas

Cet exemple montre une incontestable supériorité de la méthode de codage basée sur les techniques d'immersion dans l'hypercube booléen. La courbe de la figure IIB-9 exprime la corrélation entre la surface finale et la distance entre les états d'un même groupe.



A. contrainte cubique
B. minimisation des distances
C. pire cas

figure IIB-9 : influence de la distance sur la surface

Il est également intéressant de noter que la méthode de traitement des contraintes basée sur la réduction de distance entre états n'affecte pas de manière significative la surface globale du circuit. Cela se justifie par l'excès de routage nécessaire qui compense en partie la surface des cellules épargnées.

Le calcul des performances pour chacun de ces circuits montre également la supériorité de notre méthode de codage. Il apparait que la méthode qui minimise les distances donne les plus mauvaises performances. Ceci provient principalement du nombre élevé de niveaux de logique résultant de la création de petites sous-expressions communes à certains monômes. Ainsi, les signaux ont donc plus de logique et de routage à traverser.

2.3 Gain associé à un groupe d'adjacence

Les groupes d'adjacence étant en très grand nombre et leur immersion en tant que face souvent incompatible dans un cube de taille fixée, la solution

optimale consistant à en immerger un nombre maximal ne peut être recherchée. On adopte donc une stratégie de type "greedy" en classant les contraintes. La complexité finale du circuit est dépendante de l'ordre de traitement des groupes d'adjacence. Il est donc important de considérer ces groupes selon un ordre efficace permettant ainsi leur utilisation optimale.

Afin de déterminer cet ordre, un gain est associé à chaque groupe. Il permet d' établir une corrélation entre le respect d'une adjacence et la complexité finale du circuit en s'appuyant sur une prédiction des heuristiques de minimisation et de décomposition.

Il est égal à la somme d'un gain local lié aux règles d'adjacence et d'un gain global prenant en compte les intersections entre les groupes d'adjacence.

Les notations suivantes seront adoptées pour le calcul des gains.

Notations :

log (n) : représente le logarithme en base 2 d'un nombre entier n.

min (n) : représente l'entier de borne minimale du réel n

max(n) : représente l'entier de borne supérieure du réel n.

Le contrôleur a les caractéristiques suivantes :

- I est l'ensemble des entrées primaires de cardinal |i|,
- O est l'ensemble des commandes de cardinal |o|,
- S est l'ensemble des états de cardinal |S|.

A ces |S| états sera affecté le codage compact minimal générant |s| variables internes avec |s| = max (log|S|).

Les équations de commandes (resp. variables internes) sont notées Oi (resp. Yi), et le code binaire d'un état "si" est noté c(si).

a) Gain local associé aux règles :

Le gain local représente le nombre d' occurrences de variables gagnées en plaçant un groupe d'adjacence sur un cube. Plus précisément, une borne supérieure de ce nombre est estimée en calculant la différence entre un pire et un meilleur cas dans une approche multi-niveau.
Son calcul dépend de la règle considérée et du nombre d'états composant le groupe d'adjacence ; une fonction gain a donc été développée pour chaque règle.

Etat à entrance multiple :

Lorsque les prédicats portés sur les arcs sont identiques alors une expression $Y_j = i * \Sigma_{[p=1..n]}\ c(s_p)$ est générée pour chaque variable interne égale à 1 dans l'état jonction. Ainsi, quel que soit le code affecté aux "s_p", les variables d'entrée i seront factorisées ; elles ne rentrent donc pas en compte pour le calcul du gain.
Dans le pire cas, le nombre de variables apparaissant dans les p monômes est égal à p*|s|.
Dans le meilleur cas, (|s| - k) variables sont générées pour chaque variable interne (où k = max(log(n)). Le nombre de variables internes égales à 1 n'étant pas encore connu, un coefficient moyen de pondération de 1/2 est utilisé ce qui nous donne |s|/2 variables internes actives. Le gain de cette règle s'exprime donc de la manière suivante :

$$\textbf{gainR1-1} = |s|/2 * [(p - 1)*|s| + k]$$

Si le prédicat est différent, alors la mise des états sur une même face permettra de générer une expression commune à l'ensemble des p monômes égal à la partie commune invariante du code soit |s| - k, pour les |s|/2 variables à 1.

$$\textbf{gainR1-2} = p*(|s|-k)*|s|/2$$

Transitions émettant les mêmes commandes :

Le raisonnement est le même que précédemment si ce n'est que les commandes ne sont considérées qu'individuellement et non regroupées comme les variables internes. Le gain associé est donc :

$$\textbf{gainR2-1} = (p - 1) * |s| + k$$

Et lorsque les variables sont différentes :

$$\textbf{gainR2-2} = p*(|s|-k)$$

Etats à sortance multiple

Le calcul du gain est plus compliqué dans ce cas car il faut considérer le gain associé aux variables internes invariantes et celui de celles formant le k-cube. Il n'est pas tenu compte des variables composant le vecteur état courant ainsi que celles de la partie commune du prédicat puisqu'elles seront factorisées de toute façon.

Dans le premier cas, ces |s|-k variables internes invariantes sont formées de k variables dans le pire cas pour p/2 monômes en utilisant le même facteur de modulation. Dans le meilleur cas, il y à fusion des monômes.
La première partie du gain est donc :

$$\textbf{gainR3-1} = \textbf{p*k*(|s|-k)/2}$$

Ce gain correspond à celui de cette règle lorsqu'il n'est pas possible de faire de partitions selon les entrées.

La seconde partie du gain concerne les k-variables internes restantes. Le pire cas est le même que précédemment mais pour k variables soit p*k*k/2 et dans le meilleur cas, seules |k variables sont conservées puisque les fonctions recopient les entrées du fait de la partition.
Le gain est donc :

$$\textbf{gainR3-2} = \textbf{p*k*k/2-k} = \textbf{k*(p*k/2 - 1)}$$

Le gain associé à cette règle est la somme de ces deux gains si les partitions selon les entrées sont respectées.

Finalement, le gain sera donné par :

- s'il est possible de faire des partitions selon les entrées :

$$\textbf{gainR3} = \textbf{k*(p*|s|/2 - 1)}$$

- sinon :

$$\textbf{gainR3} = \textbf{k*(p*k/2 - 1)}$$

Transitions étiquetées par des entrées identiques

Le raisonnement est le même que précédemment si ce n'est que l'on induit des partitions selon les variables des états courants :

$$\mathbf{gainR4 = k*(p*|s|/2 - 1)}$$

b) Gain global :

Un gain global prenant en compte intersections et inclusions entre groupes d'adjacence est ajouté au gain local de chaque groupe.
Si par exemple, deux groupes d'adjacence {s1,s2,s3,s4,s5} et {s1,s3,s6,s7} sont obtenus, le gain associé à ces deux groupes est augmenté de celui de leur intersection {s1,s3}.

2.4 Pré-traitement des groupes d'adjacence

L'étape précédente de reconnaissance de situations dans la table de transitions a conduit à la génération d'un ensemble de groupes d'adjacence représenté par une matrice d'adjacence. Ces différents groupes induisant des adjacences entre états vont être analysés. Il en résultera une nouvelle matrice d'adjacence ordonnée selon un gain où chaque groupe représentera un k-cube à 2^k éléments, et dans laquelle les occurrences multiples sont supprimées.

a) Fusion des groupes d'adjacence identiques :

La première étape consiste à réduire la matrice d'adjacence en regroupant les groupes identiques. La complexité de cette procédure est de l'ordre de

$\Theta(n^2)$ où n représente le nombre de groupes. Il apparait alors l'intérêt de notre structure de donnée permettant une comparaison optimisée groupe à groupe grâce à des opérations binaires rapides sur les mots binaires représentant les groupes.
Lorsque deux groupes sont identiques, les gains s'ajoutent et le deuxième est éliminé de la matrice.

b) Intersections entre groupes :

La difficulté essentielle de la phase d'immersion est de prendre en compte globalement les groupes d'adjacence afin de satisfaire le maximum d'adjacences induites par ces groupes. Ces groupes d'adjacence étant en général intersectants, il convient de tenir compte en immergeant l'une de la partie intersectante avec l'autre.

Si par exemple, les groupes $G1=\{s1,s2,s3,s4\}$ et $G2=\{s1,s3,s4,s5\}$ sont obtenus, leur immersion mutuelle est conditionnée par le respect de leur intersection $G1 \cap G2=\{s1,s3\}$. Dans la figure IIB-10, ceci se traduit par la création d'une arète commune $(s1,s3)$.

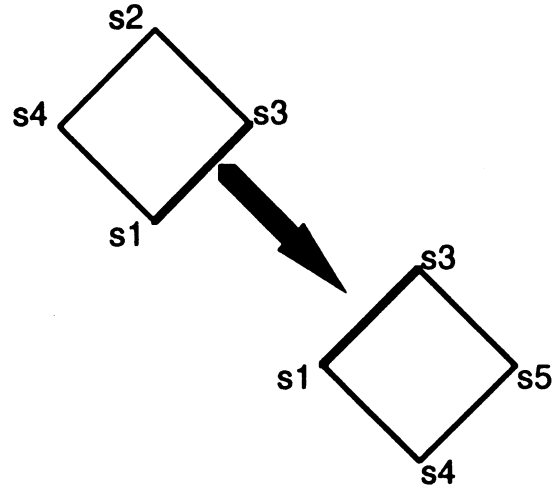


figure IIB-10 : respect de l'intersection de groupes d'adjacence

Il apparait donc que les groupes d'adjacence G1 et G2 peuvent être placés sur un 2-cube à condition que leur intersection soit une arète. Ainsi, une manière simple de prendre en considération cette intersection est de créer un nouveau groupe d'adjacence G3={s1,s3} qui sera pris en compte lors de l'immersion de G1 et G2.

Cette phase de pré-traitement consiste donc à générer les intersections des groupes d'adjacence pris deux à deux. Ces nouveaux groupes, affectés de leur gain calculé comme précédemment, sont ajoutés à la matrice d'adjacence.
Afin d'éviter l'explosion en mémoire due à de trop nombreuses intersections, un nombre limite de groupes d'adjacence est fixé comme paramètre (limit_groupe) ; sa valeur est déterminée selon les caractéristiques de l'ordinateur utilisé.

2.5 Groupes d'adjacence étendus

Comme nous l'avons exposé précédemment, notre méthode de codage est fondée sur l'immersion de sous-cube dans l'hypercube booléen.
Les groupes d'adjacence comportent de 2 à |S| états. Il faut donc modifier chacun de ces groupes afin qu'ils contiennent 2**k états exactement.

Pour cela, deux types de transformations sont possibles :

- extension au sous-cube supérieur,
- décomposition en plusieurs sous-cubes de degrés inférieurs.

a) Méthodes de réalisation de sous-cubes :

Extension au sous-cube de degré supérieur :

L'extension d'un groupe est réalisée par l'ajout d'états supplémentaires. Les codes de ces états ne doivent pas être utilisés ultérieurement pour coder les états du graphe. Ils correspondent aux 2**max(log|S|)-|S| codes non utilisés qui formeront l'ensemble des états indéfinis. Ces états non définis seront utilisés lors de la minimisation.

Définition II-1 :

Les états indéfinis sont des états fictifs n'intervenant pas dans la fonctionnalité du contrôleur et dont les codes appartiennent à la couverture a ϕ de la représentation deux couches. L'ensemble des états indéfinis est noté "dc_state".

Définition II-2 :

Si une machine d'états fini composée de |S| états est codée sur n bits, alors il existe 2^n - |S| états non définis.

Décomposition en sous-cubes de degrés inférieurs :

Un groupe d'adjacence de dimension n est partitionné en un ensemble de sous groupes de dimension inférieure. Chaque sous groupe généré est de dimension maximale en fonction du nombre d'états restants. Ainsi, le premier est de dimension p1 = 2**min(log(n)), le deuxième est de dimension p2 = 2**min(log(n-p1)) ou d'une manière générale, la dimension "pi" d'un sous groupe Gi est :

$$pi = 2^{**} \min (\log(n - \Sigma[j=1..(i-1)] \; pj))$$

Si n est impair alors un dernier groupe est formé associant l'état restant avec le précédent.

Par exemple, G = {s1,s2,s3,s4,s5,s6,s7} est décomposé en G1 = {s1,s2,s3,s4} et G2 = {s5,s6}. L'état restant est associé au précédent pour former un nouveau groupe d'adjacence G3 = {s6,s7}.

Néanmoins, afin d'éviter une partition aléatoire des sous groupes il est tenu compte des autres groupes de la matrice d'adjacence de 2**k états qui sont inclus dans G.

b) Heuristique utilisée :

La meilleure transformation possible est d'étendre les groupes d'adjacence au sous-cube supérieur puisque cela permettra un maximum de minimisations ultérieures. Malheureusement, pour une dimension du cube fixée, le nombre d'états non définis est limité et tous les groupes ne pourront être étendus. Une heuristique doit donc être utilisée afin de compléter judicieusement seulement certains groupes d'adjacence dans le but d'optimiser le gain global. Cette heuristique est la suivante :

Pour chaque groupe ne formant pas un cube, un nouveau gain est calculé. Il est égal à la différence entre le gain du groupe s'il est étendu par des éléments non définis (c'est le gain maximum qui a été calculé préalablement) et celui obtenu si le groupe est décomposé en sous-groupes.

Par exemple, considérons le groupe G = {s1,s2,s3,s4,s5,s6} et utilisons la fonction gain simplifiée suivante pour illustrer le problème, à savoir :

gain = n**2, où n représente le nombre d'états.

Si la contrainte est étendue à un 3-cube le gain est alors de 6**2 soit 36. Si cette contrainte est décomposée en deux contraintes G1={s1,s2,s3,s4} et G2={s5,s6}, le gain est alors de 4**2 + 2**2 soit 20.
La différence de ces 2 nombres correspond au gain dû à l'extension du groupe G à un 3-cube et est égale à 16 dans ce cas.

A ce groupe est également associé un autre paramètre important qui est le nombre d'états à rajouter pour le compléter au cube supérieur.

Ainsi, à chaque groupe Gi sont associés 2 paramètres :

- ci représentant le nombre d'états à rajouter, c'est à dire le **coût**,
- gi représente le **gain**.

Le problème consiste à trouver une partition des groupes telle que la somme des gains associés à chaque groupe la composant soit maximum, sans toutefois que le nombre d'états indéfinis nécessaires ne soit supérieur au nombre permis (ldc_statel).
Il se formalise de la manière suivante :

$\Sigma c_i \leq$ ldc_statel

Σg_i maximale

Ce type de problème s'apparente au problème combinatoire classique connu sous le nom de "KNAPSACK problem" [GAR79].
Il peut être résolu par programmation dynamique [DAN57], [LAW76].
La méthode optimale permettant sa résolution est la suivante.

Une table V(s) de (ldc_statel+1) éléments est construite (o..ldc_statel). V(s) correspond au gain maximal obtenu par une partition de G pour un nombre total de s états non définis utilisés.
Cette table est construite par étape. Après la ième étape, la table V(s) contient la valeur maximale obtenue pour chacun des coûts c_i.
Le pseudo-code pour une étape i s'exprime très simplement (figure IIB-11) :

```
for j:= c(i) upto ldc_statel do

  V(j) .max (V[j-c(i)] + g(i));
```

figure IIB-11 : pseudo-code du "KNAPSACK problem"

Après N étapes, le gain maximal est tout simplement le plus grand élément de V(s).

La complexité de cet algorithme est de Θ (N*ldc_statel). Malheureusement, cet algorithme à temps polynomial intéressant devient NP-complet lorsque la contrainte (ldc_statel dans notre cas) devient très grande [LAW76].

Dans le cas où cette contrainte ne peut être bornée par un nombre fixe N, une heuristique consiste à ordonner les groupes selon le rapport gain/coût. Les groupes sont alors complétés selon un ordre décroissant de ce nouveau facteur.
Il a été démontré [GAR79] que le résultat obtenu par cette heuristique n'est jamais inférieur à la moitié du gain optimal.

3. Immersion des groupes d'adjacence

3.1 Théorie des hypercubes booléens

L'algorithme d'immersion est basé sur une immersion progressive des groupes d'adjacence dans l'hypercube booléen. Un hypercube de dimension N ou N-cube est considéré comme un treillis de BOOLE isomorphe au treillis de l'ensemble des parties de N éléments (figure IIB-12).

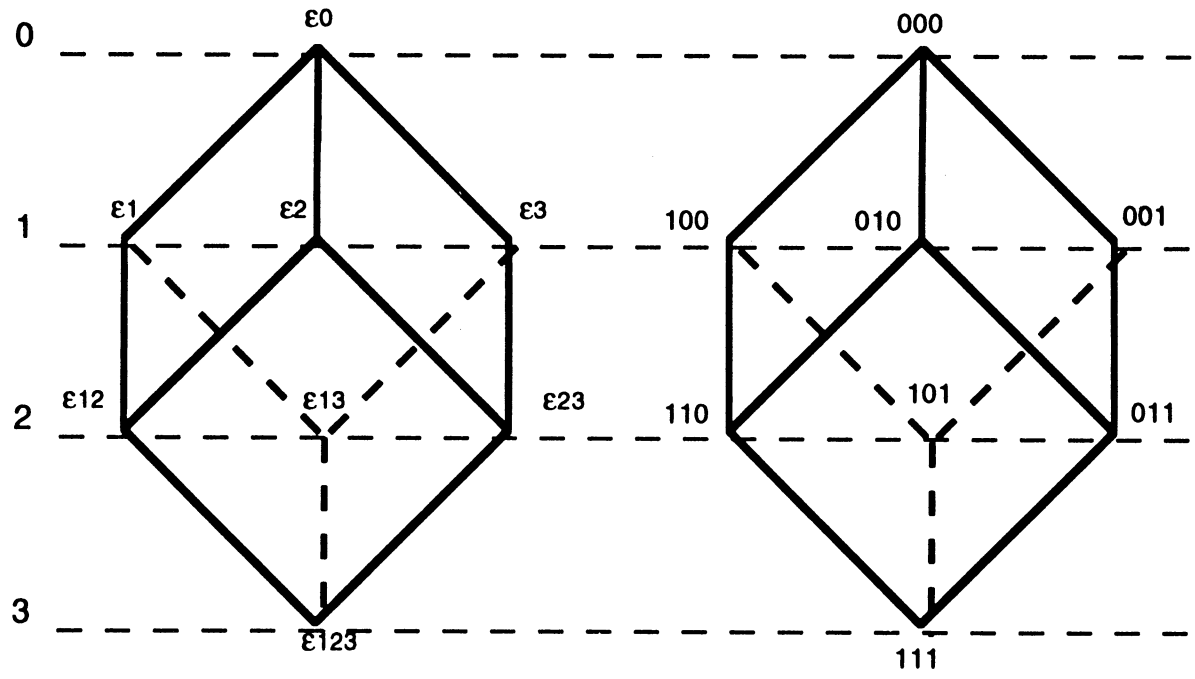


figure IIB-12 : deux types de représentation d'un 3-cube par un treillis de BOOLE

Si la dimension de l'hypercube est N, alors N+1 couches sont nécessaires [0..N] pour le représenter sous forme d'un treillis. On appelle coordonnées

relatives, le sous ensemble de N définissant un noeud dans un treillis de BOOLE.

Définition II-3 :

Une couche d'indice p contient C_N^p noeuds notés ε_i.

Définition II-4 :

Un cube est défini par les coordonnées relatives de ses extrémités ε_{min} et ε_{max}. Il est note $H=\{\varepsilon_{min},\varepsilon_{max}\}$.

Ainsi, un 3-cube est défini par $H=\{\varepsilon_0,\varepsilon_{123}\}$.

Définition II-5 :

Les éléments composant les coordonnées relatives sont appelés les indices caractéristiques.

Définition II-6 :

L'ensemble caractéristique d'un noeud indique les composantes pour lesquelles les codes d'un noeud diffèrent de celui de l'origine.

Définition II-7 :

Un hypercube $H'=\{\varepsilon'_{min},\varepsilon'_{max}\}$ est inclus dans $H=\{\varepsilon_{min},\varepsilon_{max}\}$ si les deux conditions sont respectées : $\varepsilon'_{min} >= \varepsilon_{min}$ et $\varepsilon'_{max} <= \varepsilon_{max}$.

Définition II-8 :

Le cardinal des coordonnées relatives ε_i est le nombre d'indices caractéristiques, il est noté $|\varepsilon_i|$.

Par définition : $|\varepsilon_0| = 0$.

Définition II-9 :

La dimension d'un hypercube $H=\{\varepsilon_{min},\varepsilon_{max}\}$ est égale à la différence entre le cardinal des coordonnées relatives de ε_{max} et des coordonnées relatives de ε_{min}. Elle est notée $|H| = |\varepsilon_{max}| - |\varepsilon_{min}|$.

3.2 Méthode d'immersion

a) Algorithme :

L'algorithme d'immersion fonctionne de la manière suivante.

Les groupes d'adjacence construits lors de l'étape précédente sont traités dans un ordre de gain décroissant. A chaque groupe Gi de 2^k éléments est associé un cube H_{Gi} de dimension k ; cette affectation est notée Gi : $H_{Gi}=\{^{\varepsilon t}_{min},^{\varepsilon t}_{max}\}$.

Les coordonnées de H_{Gi} sont déterminées en fonction des noeuds disponibles et des états de Gi déjà placés dans l'hypercube.

Par exemple, considérons le 2-cube $H_{G1}=\{\varepsilon_{min},\varepsilon_{max}\}$ associé au groupe G1={s1,s2,s3,s4} devant être immergé dans l'hypercube $H=\{\varepsilon_0,\varepsilon_{123}\}$ et supposons que les états S1 et S5 soient déjà placés dans ce cube (figure IIB-13).

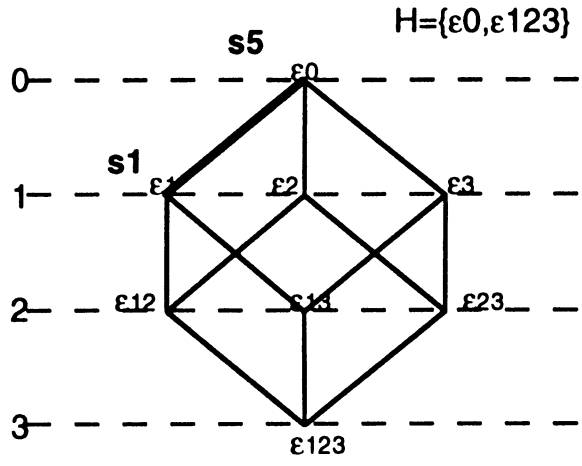


figure IIB-13 : hypercube partiellement contraint

Du fait de la contrainte induite par l'état s1, la position de H_{G1} est $\{\varepsilon_1,\varepsilon_{123}\}$. Si l'état s1 est fixe dans H_{G1}, $s1(\varepsilon_1)$, par contre les autres états peuvent être indifféremment placés sur les noeuds $(\varepsilon_{12},\varepsilon_{13},\varepsilon_{123})$ (figure IIB-14).

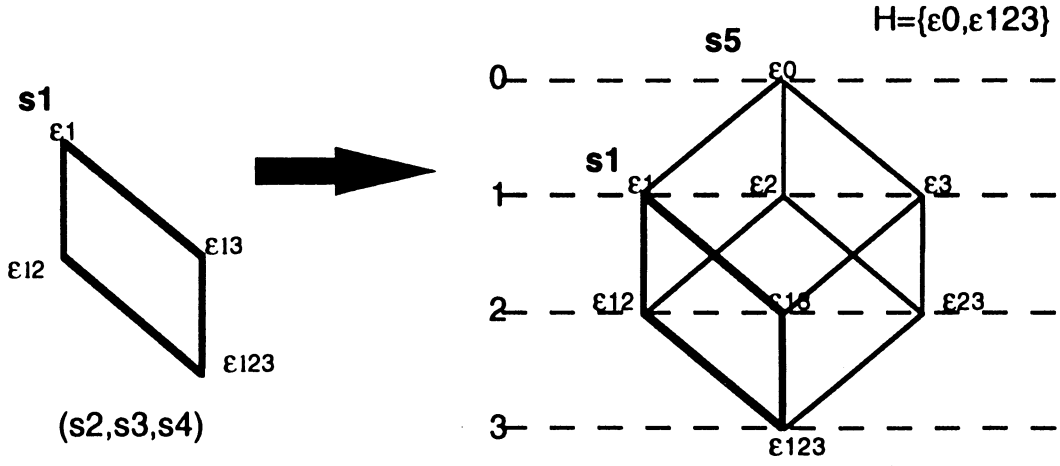


figure IIB-14 : immersion du 2-cube dans l'hypercube

Cette indécision peut être levée grâce à des informations contenues dans les autres groupes d'adjacence.

En effet, d'autres groupes d'adjacence de gain inférieur et inclus dans G1, peuvent générer de nouvelles contraintes. Ces groupes ont été produits soit initialement sur le graphe de contrôle, soit par intersection de groupes d'adjacence.

L'algorithme va donc rechercher s'il existe un groupe G2 strictement inclus dans G1. Le cube H_{G2} résultant sera alors immergé dans H_{G1} de la même manière que précédemment.

Supposons que G2 soit formé des deux états S2 et S3 : G2={s2,s3} inclus dans H_{G1}. Les deux affectations possibles ($\{\varepsilon_{12},\varepsilon_{123}\}$ et $\{\varepsilon_{13},\varepsilon_{123}\}$) étant symétriques, nous choisissons indifféremment la première pour H_{G2} (figure IIB-15).

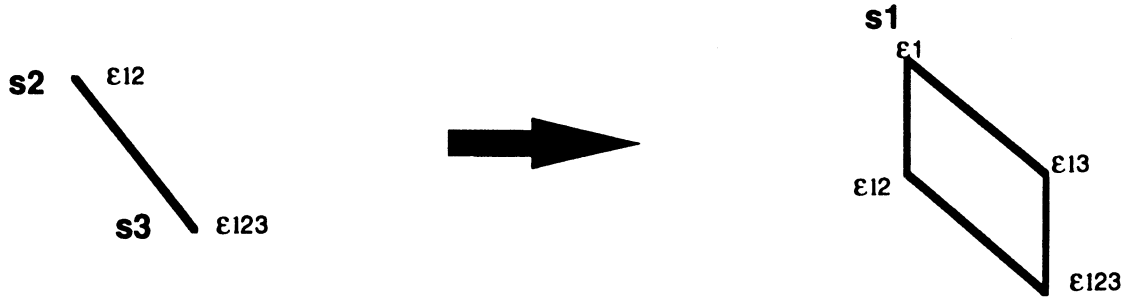
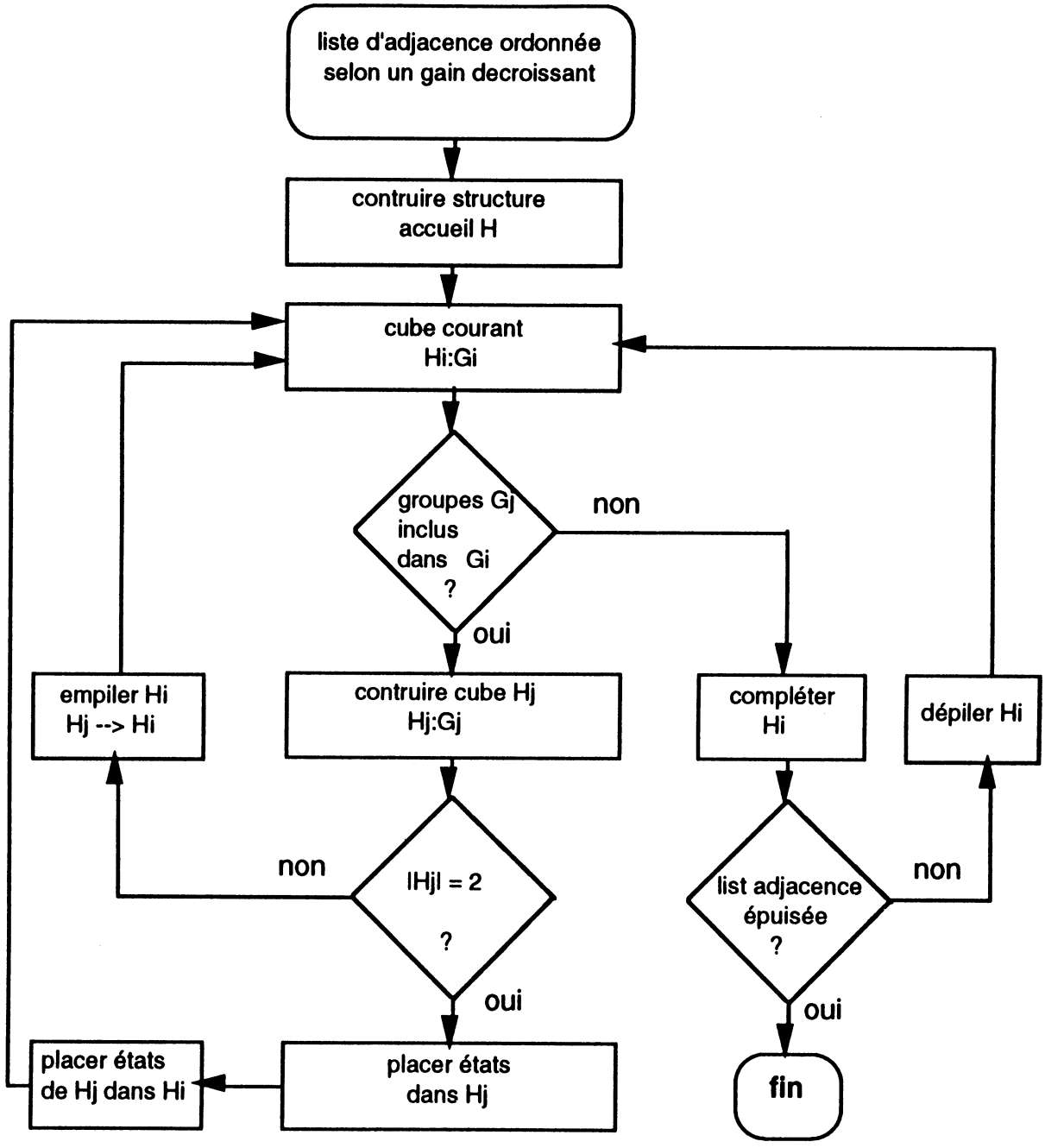


figure IIB-15 : affectation dans le 2-cube

Ceci met en évidence l'intérêt d'un algorithme récursif pour traiter efficacement ce problème. Il permet de construire progressivement le cube associé à un groupe d'adjacence en respectant au cours de sa construction les groupes inclus.
De plus, l'algorithme récursif permet de ramener ce problème à un degré de complexité moindre, puisque à tout instant il suffit de procéder à l'immersion d'un cube Hj dans un autre cube Hi de degré supérieur.

L'organigramme de l'algorithme d'immersion est représenté sur la figure IIB-16a. L'organigramme d'un algorithme récursif n'étant pas facilement représentable, son pseudo-code est aussi donné sur la figure IIB-16b afin d'en faciliter sa compréhension.



Légende :

Gi : groupe d'adjacence dont le cube associé est Hi

Gj : groupe d'adjacence inclus dans Gi et dont le cube associé est Hj

figure IIB-16a : organigramme de l'algorithme d'immersion

```
procedure embed (Hi , Gi , i);
begin

  # break condition if there are only 2 states in Gi
  # it is not possible to find another group stricly
  # included
  if |Gi| = 2 then begin
    placeStates(Hi , Gi);
    return;
  end;

  # recursive part - find included groups
  for j:= i+1 upto maxList do begin
    if cubeAllComplete (Hi) then done;
    if included (Gi , Gj) AND NOT alreadyEmbedded (Gj)
    then begin
      if Hij:= findHypercube (Hi , Gj) then begin
        embed (Hij , Gj , j);
      end;
    end;
  end;

  # complete Hi with remaining states of Gi
  placeStates (Hi , Gi);
  return;

end;
```

figure IIB-16b : pseudo-code de l'algorithme d'immersion

b) Illustration de l'algorithme d'immersion :

Un exemple de construction progressive d'un 3-cube par l'algorithme récursif d'immersion est développé ci-dessous :

Considérons les groupes d'adjacences classés par gain décroissant suivants :

G1={s1,s2,s3,s4,s5,s6,s7,s8}
G2={s1,s2,s3,s4}
G3={s1,s4,s5,s6}
G4={s3,s4}
G5={s1,s6}

Comme nous l'avons expliqué préalablement, les intersections entre groupes ont été préalablement traitées afin de générer de nouveaux groupes d'adjacence.

Ainsi, l'intersection entre G2 et G3 a permis de générer un nouveau groupe :

G6={s1,s4}

Le cube $H=\{\varepsilon_0,\varepsilon_{123}\}$ est affecté au groupe G1 (figure IIB-17a).

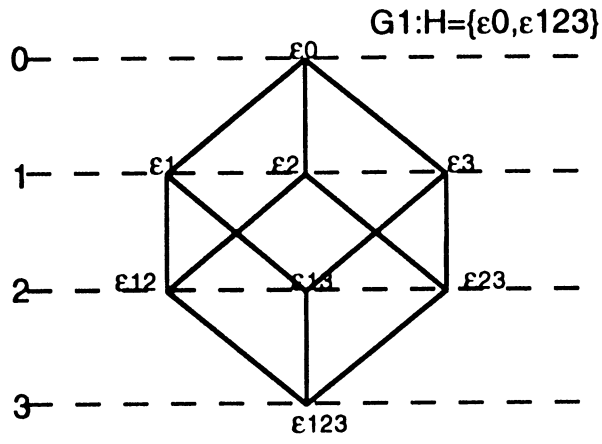


figure IIB-17a

On recherche dans l'ensemble des groupes strictement inclus dans G1. Le groupe G2 est le premier de la liste et il lui est affecté H1={ε_0,ε_{12}} (figure IIB-17b) :

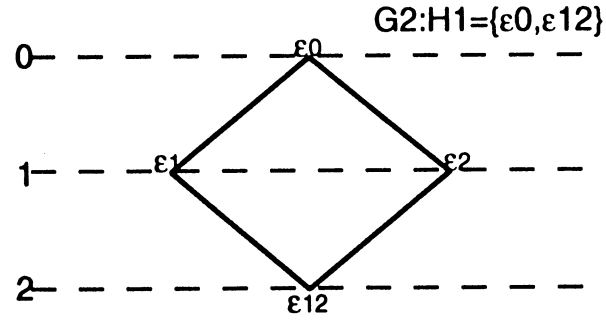


figure IIB-17b

G4 est inclus dans G2 et le cube H11={ε_0,ε_1} est construit (la notation Hi..jk indique le degré de récursivité dans la phase d'immersion ; dans ce cas, on immerge le cube associé au k^{ième} groupe dans le j^{ième}). Ce groupe ne contenant que 2 éléments, aucun autre ne pourra y être inclus. Ses états sont donc placés sur le cube H11 : s3(ε_0), s4(ε_1) comme l'illustre la figure IIB-17c.

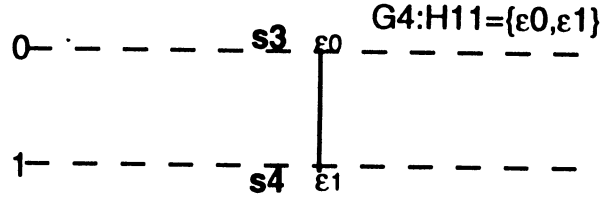


figure IIB-17c

La pile de stockage des données de cette procédure récursive "remonte" d'un niveau et les états de H11 sont reportés dans H1 (figure IIB-17d).

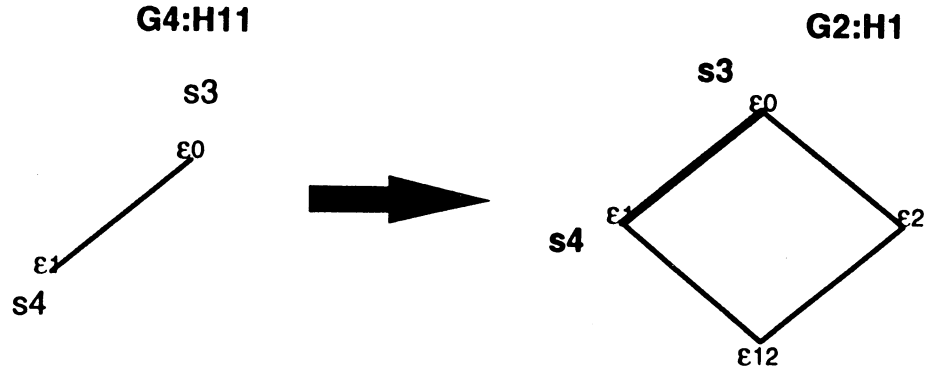


figure IIB-17d

G6 qui est inclus dans G2 est alors traité et le cube H12={e1,e12} est construit. L'etat s4 étant déjà placé, l'affectation de s1 est donc immédiate s1(e12) (figure IIB-17e).

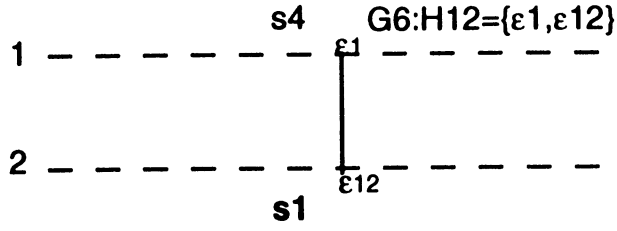


figure IIB-17e

Ces informations sont reportées dans H1. L'état s2 qui est le seul état non placé est alors affecté au noeud restant s2(e2) (figure IIB-17f).

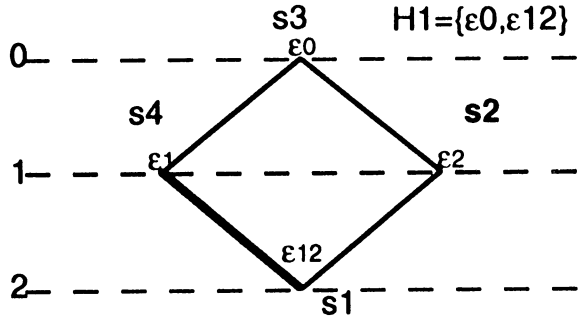


figure IIB-17f

Cela permet de remplir partiellement H (figure IIB-17g).

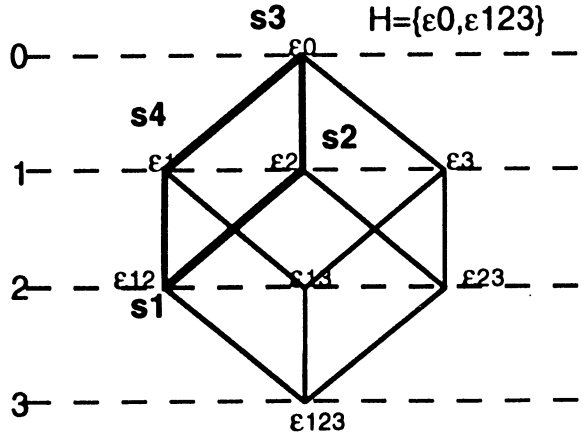


figure IIB-17g

Le groupe G3 est alors traité de la même manière et les différentes étapes sont détaillées par les figures IIB-17h,i,j.

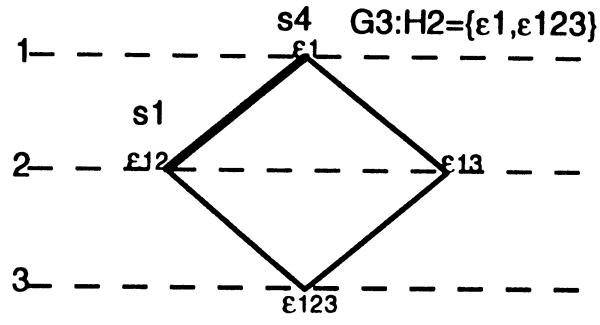


figure IIB-17h

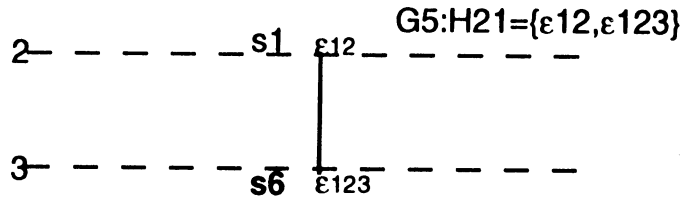


figure IIB-17i

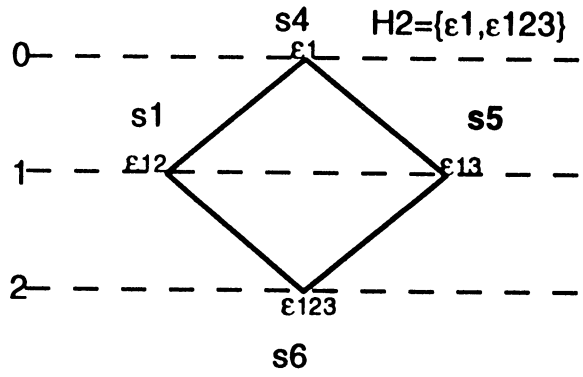


figure IIB-17j

Les états s7 et s8 n'ont pas de contraintes spécifiées par un groupe d'adjacence, ils sont donc affectés arbitrairement aux noeuds restants (ε3) et (ε23).

Finalement, le 3-cube respectant l'ensemble de ces contraintes est complètement défini (figure IIB-17k).

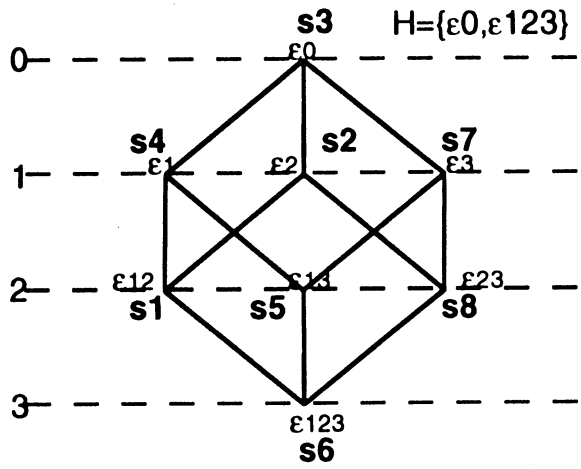


figure IIB-17k

3.3. Affectation des codes aux états

La dernière étape de l'algorithme consiste à affecter un code binaire unique à chacun des états du contrôleur en respectant les relations d'adjacence entre états contenues dans l'hypercube booléen.

a) Détermination de l'état code 0 :

L'affectation du code 0 à un état (tous les bits du code de l'état valent 0) a des conséquences intéressantes sur les équations de séquencement. En effet, pour une synthèse en bascules D, cet état ne génèrera pas de

monômes pour les variables internes puisqu'il n'y aura pas de 1 dans son code.

Par exemple, si l'état s de la figure IIB-18 est code 0, alors les monômes i1, i2, i3 et i4 n'interviendront pas dans les équations de séquencement.

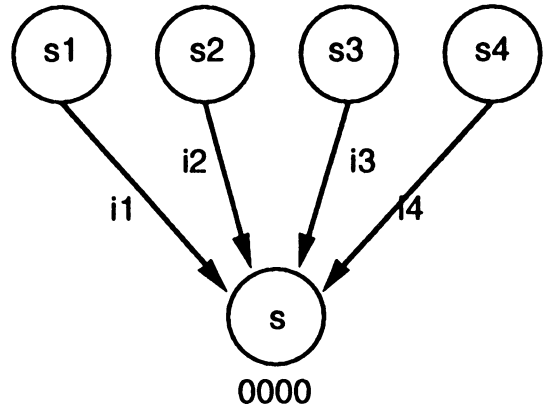


figure IIB-18 : code 0

Si, de plus, il n'y a pas de commandes portées sur les arcs qui convergent vers cet état alors le monôme en question n'interviendra dans aucune équation.

Ainsi, pour optimisation optimale cet état sera celui ayant le maximum d'arcs entrant dans le graphe de contrôle.

b) Code définitif :

L'algorithme attribuant le code définitif aux états du contrôleur est alors très simple. Il suffit d'affecter un code à un des états. Dans l'exemple précédent, si S5 a comme code 000, le sommet S3 aura comme code 101 car le premier et le troisième bit de S3 et S5 différent des codes des autres sommets.

Si pour l'exemple précédent, il a été déterminé que c'est le code de l'état s5 qui doit être nul, alors l'ensemble des premiers et derniers bits du code de tous les états doit obligatoirement être complémenté (figure IIB-19).

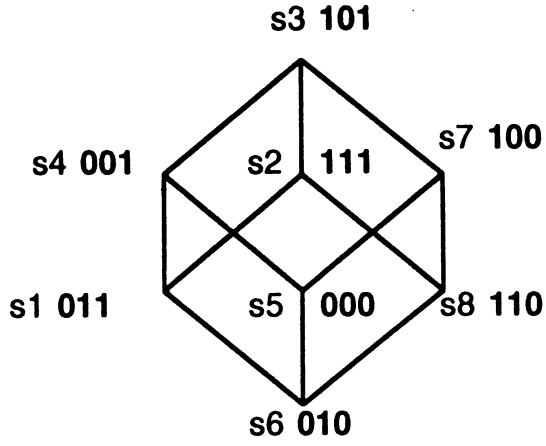


figure IIB-19 : rotation de l'hypercube

La dernière étape consiste à immerger les états qui ne sont pas encore dans l'hypercube. Ces états correspondent aux états qui ne font partie d'aucune contrainte, ou dont la contrainte à laquelle ils appartiennent n'a pas été satisfaite.

Cette immersion se fait de préférence sur les noeuds ayant le moins de 1 possible afin de réduire aux mieux les futures équations de séquencement.

4. Résultats expérimentaux

Cet algorithme a été implémenté en MAINSAIL [MAI86] sur SUN 3/60 dans l'outil de synthèse logique de VLSI TECHNOLOGY. Ses performances ont été mesurées sur un ensemble d'exemples standards des outils de synthèse logique provenant de l'*International Workshop on Logic Synthesis* [MCN89].

Les caractéristiques des contrôleurs testés sont données dans le tableau IIB-1.

	entrées	sorties	états
bbara	5	2	10
bbtas	3	2	6
keyb	8	2	19
mc	4	5	4
modulo12	2	1	12
planet	8	19	48
s1	9	6	20
s1a	9	6	20
shiftreg	2	1	8
tav	5	4	4
tbk	7	3	32

tableau IIB-1 : caractéristiques des contrôleurs

Une bonne évaluation de l'efficacité des algorithmes de codage n'est pas facile à effectuer car, lors de la synthèse de contrôleurs, de nombreuses étapes sont réalisées pour parvenir à l'implémentation finale du circuit. Il est donc très difficile de faire une évaluation de la seule phase de codage.

Une manière raisonnable d'estimer les performances de l'algorithme et d' établir des comparaisons avec d'autres outils consiste à comparer les résultats obtenus par l'application de la méthode avec une méthode de codage aléatoire en laissant les autres étapes de la synthèse strictement identiques. De plus, cette comparaison peut être effectuée à différentes étapes de la synthèse (nombre de monômes, nombre de littéraux dans les expressions factorisées, nombre de cellules précaractérisés, surface, vitesse...).
C'est de cette manière que nous avons procédé, et les valeurs utilisées comme base de comparaison pour le codage aléatoire ont été obtenues en réalisant la moyenne des résultats de cinq codages aléatoires.

L'evaluation de la méthode de codage a été réalisée en fonction de deux cibles : implémentation deux couches ou PLA et implémentation multi-niveaux à base des cellules précaractérisées disponibles dans la bibliothèque en technologie 1µ de VLSI TECHNOLOGY [VSc90].

Le deuxième tableau donne les résultats de l'application de notre méthode pour une implémentation sur PLA.

	aleat.	asyl
bbara	30	27
bbtas	14	13
keyb	111	51
mc	9	8
modulo12	15	13
planet	106	98
s1	101	80
s1a	96	75
shiftreg	12	10
tav	11	11
tbk	271	157
TOTAL	776	543
RATIO	**1.43**	

tableau IIB-2 : implémentation sur PLA

Ces résultats expriment le nombre de monômes nécessaires pour les expressions de séquencement et de commandes après l'étape de minimisation. Ils montrent que cette méthode de codage apporte un gain moyen de 43% par rapport à une méthode de codage aléatoire. Ce résultat est très significatif car le gain en monômes est directement corrélé au gain en surface lorsque l'implémentation physique considérée est le PLA.

Le troisième tableau permet d'estimer l'efficacité de cette méthode par rapport aux autres algorithmes connus de codage dans le cas d'une synthèse multi-niveaux. Quelques publications ont déjà fait l'objet de comparaisons des résultats obtenus par l'application de divers algorithmes

(KISS, MUSTANG_P et MUSTANG_N [DEV87]). Malheureusement, les résultats sont en terme de littéraux gagnés sur les expressions après factorisation, et, comme nous l'avons expliqué précédemment, ce type d'évaluation reflète mal l'implémentation physique finale.

Ici, les résultats de notre méthode sont évalués en terme de portes équivalentes ; une porte équivalente représente le nombre de transistors d'une porte NAND à deux entrées (quatre transistors en CMOS). Ce nombre est associé à toute cellule de la bibliothèque de VLSI Technology. Ce critère a été choisi car il reflète fidèlement la surface réelle de la cellule.

	alea-1	asyl	alea-2	musP	musN	kiss
bbara	82	62	121	67	84	129
bbtas	23	23	31	30	22	32
keyb	264	169	369	330	230	286
mc	16	12	142	21	21	27
modulo12	45	32	53	37	27	50
planet	601	564	592	637	558	591
s1	357	290	420	309	248	376
s1a	280	206	347	289	164	287
shiftreg	21	14	32	16	8	8
tav	16	16	21	21	21	21
tbk	874	515	367	309	306	381
TOTAL	2579	1904	2383	2066	1689	2188
RATIOS		**1.35**		**1.15**	**1.41**	**1.09**

tableau IIB-3 : implémentation en cellules précaractérisées

- alea-1 représente le résultat après codage aléatoire dans l'outil de synthèse de VLSI TECHNOLOGY ; il exprime le nombre de portes équivalentes.
- alea-2 représente le nombre de littéraux obtenus dans l'environnement MIS [BRA87].

Le rapport alea-1/asyl représente un gain de 35% de portes équivalentes pour notre méthode. Il apparait que KISS est mal adapté à une synthèse multi-niveaux puisqu'il ne permet qu'un gain de alea-2/KISS=9%. Mustang N donne un bon gain en littéraux (alea-2/musN=41%) mais ce critère ne

laisse pas précisément entrevoir le gain réel en surface globale (cellules et connectique) après placement et routage [SAU90].

Enfin, nous avons évalué dans le tableau 4, l'impact de notre méthode sur la surface finale du circuit après placement et routage. Le nombre de portes équivalentes (P), la surface totale après placement et routage (S) ainsi que le facteur de routage (R) sont explicités pour chacune de ces méthodes.

	Aléatoire			ASYL			RATIOS		
	Pa	Sa	Ra	Pv	Sv	Rv	P	S	R
bbara	82	463	1.33	62	414	1.30	1.32	1.12	1.02
keyb	264	2069	1.88	169	1158	1.45	1.56	1.79	1.30
mc	16	110	0.70	12	83	0.60	1.33	1.33	1.17
modulo12	45	235	0.92	32	176	0.74	1.41	1.34	1.24
planet	601	4252	2.58	564	3786	2.48	1.07	1.12	1.04
s1	357	2310	2.10	290	2013	2.05	1.23	1.15	1.02
s1a	280	1835	2.09	206	1270	1.82	1.36	1.45	1.15
tbk	874	13020	1.81	515	10526	1.62	1.70	1.24	1.12
					MOYENNE		1:37	1.32	1.13

tableau IIB-4 : influence du codage sur la surface des circuits

Ces résultats démontrent très clairement l'efficacité de cette méthode sur la surface finale du circuit. Ils mettent en évidence que la minimisation du nombre de portes équivalentes (37%) est étroitement associée à la minimisation de surface finale (32%). Cela signifie que notre algorithme

n'induit pas de factorisation dont le gain en portes aurait été contre-balancé par un accroissement du routage.

Conclusion

Les résultats obtenus dans un environnement industriel confirment l'importance du codage des états lors de la synthèse de contrôleurs. La méthode que nous avons présentée a l'avantage d'être adaptée à plusieurs cibles (PLA ou logique multi-niveaux) grâce à un algorithme original et efficace d'immersion de cubes dans un hypercube booléen.

De plus, cette méthode a également permis de réduire le routage en conjonction avec la minimisation du nombre de portes nécessaires à une implémentation du circuit en cellules précaractérisées.

III. Synthèse de fonctions booléennes

A. Etapes de la synthèse

Introduction

La synthèse automatique de circuits combinatoires permet d'obtenir la description structurelle d'un circuit intégré à partir d'une spécification fonctionnelle de haut niveau. Cette phase de synthèse s'effectue en plusieurs étapes dont les plus importantes sont : la minimisation des équations booléennes, la décomposition, le "mapping" technologique et finalement l'optimisation finale (figure IIIA-1).

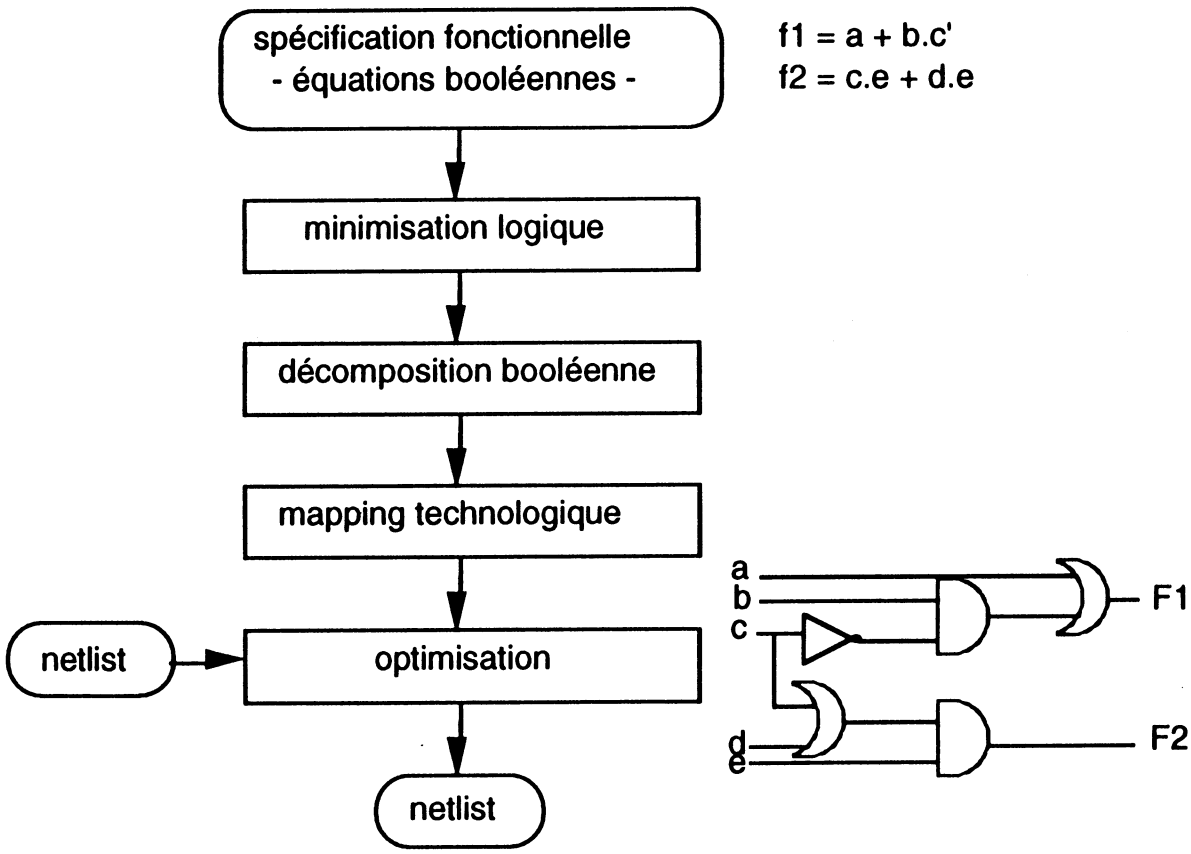


figure IIIA-1 : étapes de la synthèse combinatoire

L'étape de minimisation consiste à simplifier les équations logiques préalablement définies en une somme de monômes en minimisant le nombre de monômes et de littéraux.

La phase de décomposition détermine les éléments logiques susceptibles d'être partagés entre une ou plusieurs fonctions en vue de simplifier la réalisation de l'ensemble des fonctions.

Ensuite, le "mapping" technologique réalise l'implémentation physique des fonctions logiques par un réseau de cellules de bibliothèque.

Enfin, une étape d'optimisation est effectuée si le résultat en temps ou surface obtenu à l'étape précédente n'est pas jugé satisfaisant.

1. Définition de base de l'algèbre de BOOLE

Georges BOOLE, mathématicien anglais du dix neuvième siècle, a été le premier à utiliser une algèbre à deux variables pour l'étude de la logique mathématique (applicable au raisonnement).
L'extension du champ d'application de ce type d'algèbre aux machines électroniques a permis d'en définir une terminologie précise dont nous allons rappeler les définitions mathématiques de base. Pour une étude plus profonde de ce domaine, le lecteur pourra se référer aux travaux du professeur J. KUNTZMANN [KUN68].

Soit B={0,1}, l'espace booléen.

Définition III-1 :

On appelle *variable booléenne simple*, toute quantité x susceptible de prendre seulement les deux valeurs logiques 0 ou 1, $x \in B$.

Définition III-2 :

Un *littéral* représente une variable définie sous forme normale (x) ou complémentée (x').

Définition III-3 :

Un *monôme* est le produit de n littéraux ; n représente le degré du monôme. Il est aussi appelé cube du fait de son interprétation géométrique dans l'espace booléen B^n.

Définitions III-4 :

On appelle *fonction booléenne générale* à n variables la fonction f définie par :

$f : B^n \dashrightarrow B^m$

On appelle *fonction booléenne simple* à n variables, la fonction f définie par :

$f : B^n \dashrightarrow B$

Une fonction booléenne est dite *incomplète* si elle est définie par :

$f : B^n \dashrightarrow Y^p$ ou $Y=\{0,1,\emptyset\}$.

$\emptyset$ représente la valeur non définie appelée aussi "don't care" dans la terminologie anglo-saxone, ce qui signifie que la fonction peut prendre indifféremment la valeur logique 0 ou 1.

Une fonction booléenne est exprimée en fonction de ses variables et des opérations booléennes classiques (+,.).

Définition III-5 :

On définit la relation d'ordre partiel sur les fonctions booléennes :

$f \leq g \Leftrightarrow f \cdot g = f \Leftrightarrow f + g = g$

Définition III-6 :

Une fonction booléenne incomplète de f peut être définie par trois ensembles disjoints :

- *C1(f)* représente la *couverture à 1* d'une fonction, c'est à dire l'ensemble de monômes pour lesquels le résultat de la fonction est vrai (f = 1).
- *C0(f)* (resp. $C\phi(f)$) représente la *couverture à 0* (resp. *couverture a* ϕ) c'est à dire l'ensemble des monômes pour lesquels f = 0 (resp. f = ϕ).

Définition III-7 :

Un monôme est dit *canonique* s'il contient toutes les variables de la fonction.

Définition III-8 :

L'expression d'une fonction booléenne est *polynomiale* si elle est exprimée sous forme d'une somme de monômes.

Définition III-9 :

Un monôme *premier* d'une fonction f est un monôme qui n'est pas totalement contenu dans un autre monôme de f.

Définition III-10 :

L'expression d'une fonction booléenne polynomiale est *première* si elle composée uniquement de monômes premiers.

Définition III-11 :

Un monôme premier est *essentiel* s'il contient un monôme canonique qui n'est inclus dans aucun autre monôme premier.

Définition III-12 :

Une somme de monômes premiers est dite *irrédondante* si l'élimination d'un de ses termes ne la rend plus logiquement équivalente à la représentation initiale.

2. Minimisation logique

La minimisation logique a eu comme premier objectif d'exprimer les fonctions booléennes sous forme de bases irrédondantes de monômes premiers. Cette minimisation effectuée manuellement sur tableau de KARNAUGH [KAR53] a été peu à peu automatisée. Les premiers algorithmes proposés par QUINE McCLUSKEY [MCC65], s'appuient sur des représentations canoniques de la fonction et recherchent les solutions optimisées. Ces techniques étaient explosives pour des exemples de taille importante. La théorie des consensus proposée par KUNTZMANN [KUN68] et TISON [TIS67] a permis de se débarrasser de ce point de départ d'expressions booléennes sous forme canoniques en proposant des méthodes de minimisation partant de n'importe qu'elle expression déjà partiellement minimisée.

Plus tard, la recherche en synthèse booléenne a essentiellement évolué au gré des cibles technologiques. Les dispositifs du type réseaux programmables (PLA) ont introduit un nouveau concept, celui d'optimisation globale. En effet, le but recherché dans ces cibles est l'obtention d'un ensemble de monômes de cardinal minimal permettant d'exprimer un ensemble de fonctions dépendant des mêmes variables. Ainsi sont nées les théories de l'optimisation globale consistant à rechercher des bases irrédondantes globales à l'aide de monômes premiers globaux. Les logiciels correspondants se sont avérés très efficaces (MINI [HON74], PRESTO [BRO81], ESPRESSO-II [BRA84], ASYL [SIC88]...).

Le problème actuel des outils de minimisation sur PLA est d'utiliser des heuristiques permettant de traiter des circuits réels dont la taille est importante tout en garantissant un résultat proche de l'optimal. Ceci n'est

pas simple car la complexité de la phase de détermination des monômes premiers globaux est de $\Theta (3^n/n)$, où n représente le nombre de variables [BRA84]. De plus, la recherche de toutes les bases globales irrédondantes est un problème NP-complet [GAR79].

Les deux logiciels les plus connus, ESPRESSO-II et ASYL, sont donc fondés sur une heuristique très utilisée en optimisation. Elle s'apparente aux techniques de recuit simulé. On part d'un ensemble de monômes exprimant les fonctions et on cherche à en extraire un ensemble minimal de monômes premiers. On tombe ainsi dans un minimum local. On part ensuite dans une nouvelle phase de génération de monômes ou des transformations de monômes permettent d'atteindre un nouveau minimum.

ASYL et ESPRESSO-II différent dans le principe de génération de nouveaux monômes. Dans ESPRESSO-II, les nouveaux monômes sont obtenus par agrandissement de monômes existants. Dans ASYL, ces monômes sont obtenus par intersection de monômes. C'est pour cette raison que ASYL part de gros monômes obtenus par minimisation locale. Les intersections sont générées et agrandies dans la direction des monômes originels.

Une comparaison des deux méthodes est représentée par l'organigramme de la figure IIIA-2.

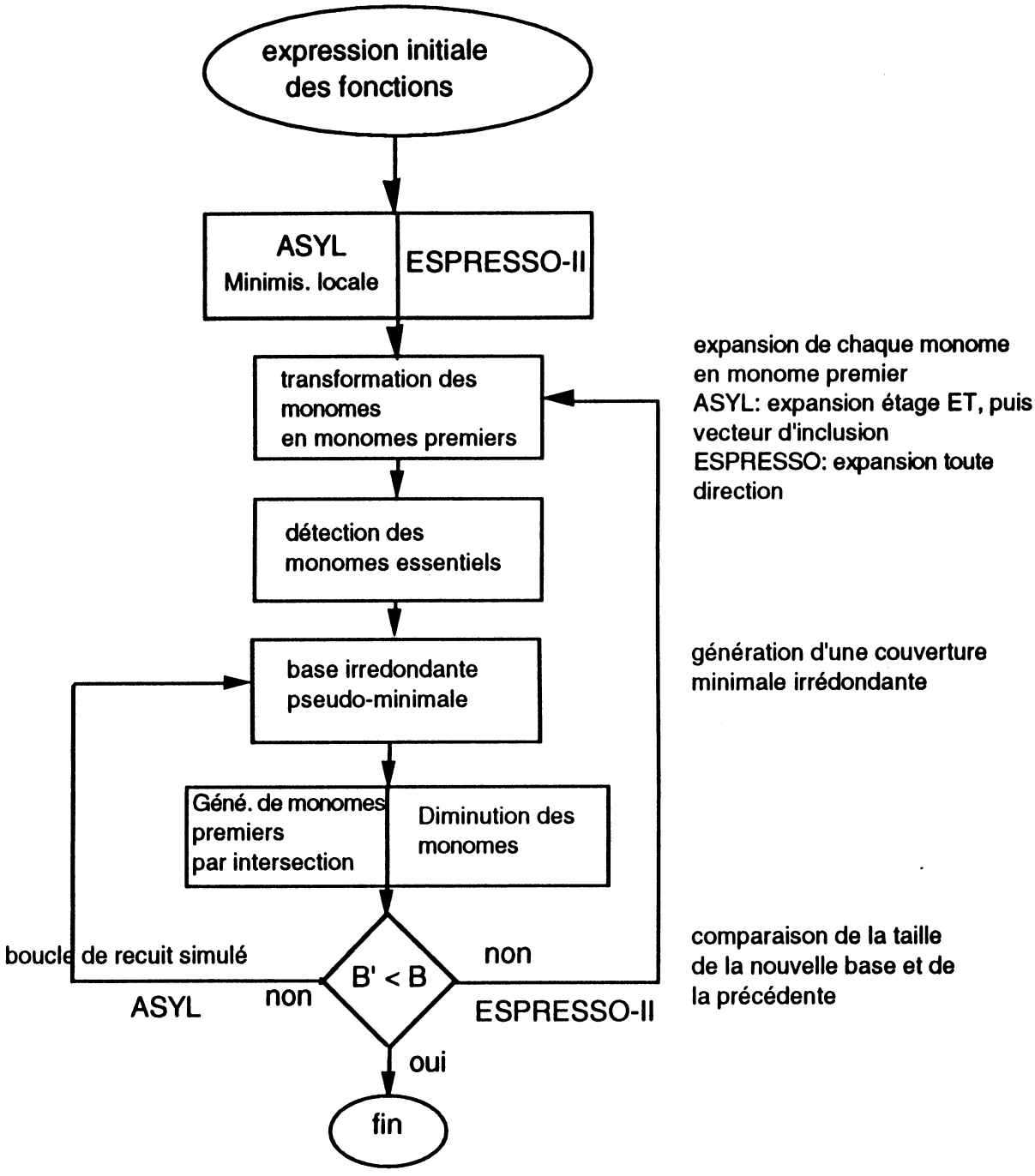


figure IIIA-2 : comparaison ESPRESSO-II et ASYL

3. Décomposition logique

Une autre cible technologique concernant les réalisations de circuits à base de cellules précaractérisées a conduit à une évolution significative des méthodes de synthèse logique. Ces nouvelles techniques s'intéressent à la synthèse multicouches de plusieurs fonctions booléennes sur cellules précaractérisées. Elle est réalisée en une étape de décomposition suivie du "mapping" technologique.

L'étape de décomposition vise à rechercher des expressions booléennes à nombre minimal de variables. Pour ceci, les fonctions booléennes sont factorisées en recherchant de facteurs communs entre une ou plusieurs fonctions. Ces facteurs communs ou expressions communes deviennent des sous-fonctions qui ne sont implémentées qu'une seule fois.

La figure IIIA-3 montre l'implémentation de deux fonctions f1 et f2 sur une structure 2-couches et une structure multicouche pour laquelle a été extrait les sous fonctions q, r, s, t.

Afin de facilité la compréhension de cette transformation, nous avons représentés sur la partie gauche de la figure, la forme du circuit après "mapping" technologique.

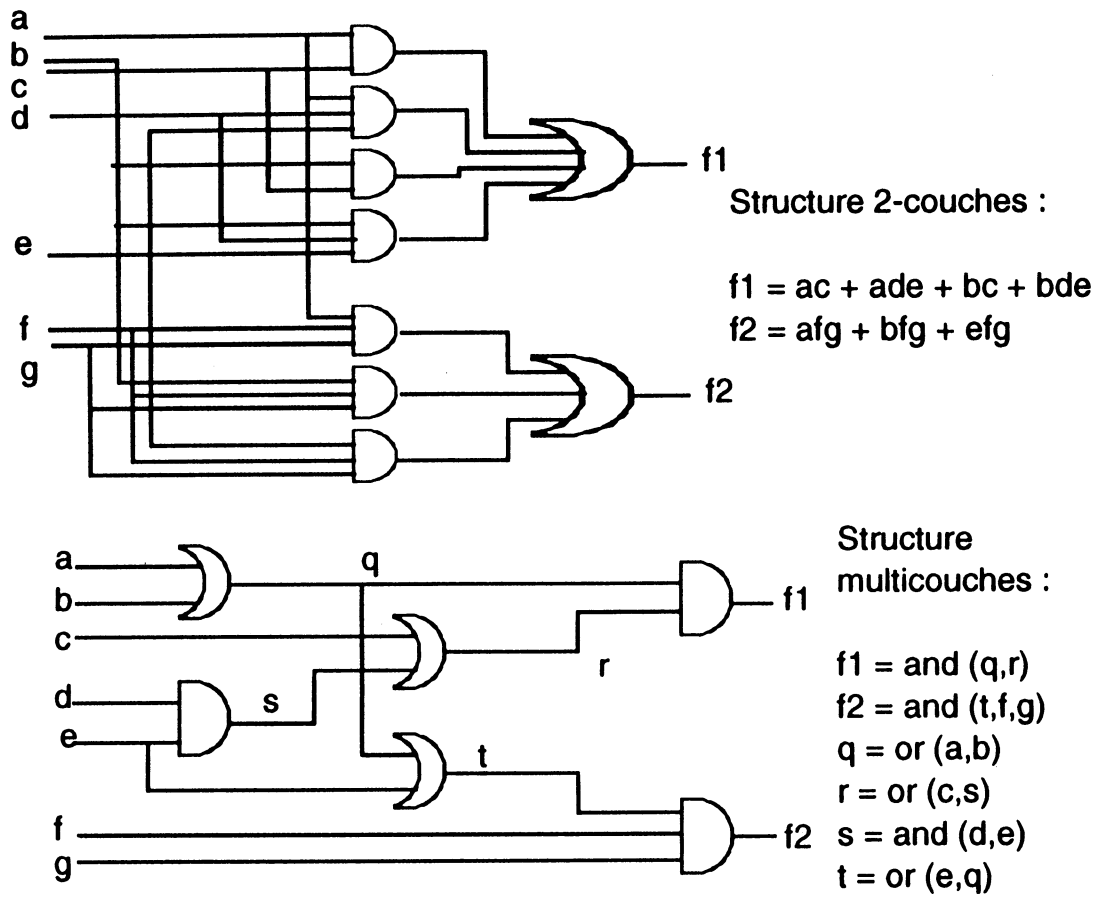


figure IIIA-3 : décomposition de la logique combinatoire

On remarque que la représentation deux-couches nécessite 26 entrées de portes et 9 portes logiques alors que la représentation multi-couches minimise ce coût à 13 entrées de portes et 6 portes logiques.

La décomposition logique est effectuée en trois étapes successives : la génération de candidats diviseurs, la sélection du meilleur d'entre-eux et la substitution de celui-ci dans l'ensemble des fonctions auxquelles il appartient.

D'abord, la *génération* crée un ensemble de candidats diviseurs. Ce sont des sous-expressions qui apparaissent plusieurs fois dans les expressions logiques et dont l'implémentation unique en réduit la complexité.

Puis, la *sélection* détermine le candidat le plus intéressant qui induit le plus grand gain. Ce gain dépend du critère d'optimisation requis. Il peut exprimer soit une estimation de surface gagnée par ce choix, soit une estimation de vitesse.

Enfin, une *substitution* par division (algébrique ou booléenne) du candidat sélectionné est effectuée dans l'ensemble des fonctions auxquelles il appartient.

Ces trois étapes sont réitérées successivement jusqu'à épuisement du nombre de candidats.

3.1 génération des candidats diviseurs

Cette étape produit un ensemble de candidats diviseurs pour toutes les fonctions logiques. Le problème majeur est de générer un nombre raisonnable de candidats assurant toutefois une implémentation physique ultérieure optimale.

Une méthode efficace d'extraction de sous expressions intéressantes a été introduite par BRAYTON [BRA82], [BRA84] ; elle est basée sur l'extraction de monômes communs de plusieurs fonctions et de noyaux de fonctions.

Définition III-13 :

Le produit algébrique f.g existe et est unique si f et g dépendent d'ensembles de variables disjoints. Il est défini par :

$f.g = \Sigma_{[i=1..n,\ j=1..m]}\ mi.mj$

avec $f = \Sigma_{[i=1..n]}\ mi$ et $g = \Sigma_{[j=1..m]}\ mj$

Par exemple, le produit algébrique de $f=ab + d'$ par $g = c + e'$ est :

f.g = abc + abe' + cd' + d'e'

Définition III-14 :

g est un diviseur algébrique de f si :

f = g.h + r

où h est la plus grande expression possible non nulle, c'est à dire qu'il n'existe pas de ¬h tel que h < ¬h + r

et où r est une expression polynomiale.

Le quotient et le reste de cette division sont uniques.

Ainsi la division algébrique de $f = abd + cd + e$ par $g = ab + c$

donne comme quotient h = f/g = d et comme reste r = e

Définition III-15 :

Une expression f est libre si aucun monôme ne peut la diviser globalement. En d'autres termes, il n'existe pas de monôme g tel que f puisse s'écrire f=gh, où h est une expression polynomiale.

Par exemple, ab+c est libre mais ab+ac ne l'est pas.

Définition III-16 :

L'ensemble des noyaux d'une expression f est défini par :

$$K(f) = \{f/c \text{ où } c \text{ est une expression libre}\}$$

Définition III-17 :

Le monôme c permettant d'obtenir le noyau k = f/c est appelé le conoyau de k.

Par exemple, un des noyaux de la fonction f définie par :

$$f = a.b + b.c + d$$

est k = a + c et son conoyau associé est b.

Remarque :

Les noyaux d'une fonction booléenne résultent d'une division récursive des cofacteurs d'une expression de la fonction. Le cofacteur algébrique d'une fonction, par le littéral x représente le quotient de f/x sans considérer le reste de la division.
Par exemple, le cofacteur de f = ab + ad + bc par le littéral "a"
est f / a = b + d.

Le théorème suivant montre une propriété importante des noyaux.

Théorème :

f et g ont un diviseur algébrique commun d, où d est une expression polynomiale, si et seulement si :
$d = k_f \cap k_g$ avec $k_f \varepsilon K(f)$ et $k_g \varepsilon K(g)$.

Ce théorème (démontré dans [BRA82]) montre que deux fonctions ont une expression polynomiale en commun si et seulement si l'intersection entre un noyau de f et un noyau de g contient plus d'un monôme.
Cela met en évidence une méthode permettant de déterminer des sommes de monômes communes à plusieurs fonctions. L'algorithme généralement utilisé recherche dans un premier temps l'ensemble des noyaux de chaque fonction et procède à l'intersection de ces ensembles.

3.2 Sélection du candidat diviseur :

Cette étape permet de choisir dans l'ensemble des candidats déterminés lors de la phase de génération celui qui impliquera un gain maximal de logique. Deux types de sélection sont actuellement utilisés : une première méthode choisit le candidat permettant d'obtenir un gain local maximal (algorithme "greedy") sans vérifier sa compatibilité avec les autres, et une deuxième méthode considère globalement l'ensemble des candidats pour tenir compte de l'impact créé par le choix de l'un d'entre-eux sur l'ensemble des candidats restants. Cette deuxième méthode aboutit à la génération d'un ensemble de sous-expressions mutuellement compatibles. Deux expressions étant compatibles si elles ne dépendent pas des mêmes monômes de la fonction initiale.

La première méthode est plus simple mais n'aboutit pas à un résultat optimal. Elle consiste à affecter un gain à chacun des candidats et à sélectionner celui de plus grand gain.
Ce gain peut s'exprimer de plusieurs manières. Pour les outils indépendants de la technologie, il représente le nombre de littéraux gagnés dans les expressions grâce à cette substitution, cette estimation ne

reflète que peu la réalité physique finale. Pour les outils dépendants d'une technologie, il peut être exprimé en comparant les implémentations physiques locales avant et après transformation. Il peut ainsi être mesuré en terme de surface ou en terme de vitesse selon les spécifications du concepteur.

La deuxième méthode est basée sur le principe de la couverture de rectangles énoncée dans [BRA87]. Elle consiste à construire une matrice M où chaque ligne correspond à un unique conoyau d'un noyau et où chaque colonne à un unique monôme d'un noyau de la fonction. Ainsi, un monôme de l'expression initiale de la fonction est une case de cette matrice.
Par exemple, considérons la fonction : f = ac + ade + bc + bde
Les noyaux et conoyaux de f sont (figure IIIA-4) :

noyaux	conoyaux
c + de	a , b
a + b	de , c

figure IIIA-4 : noyaux et conoyaux de f

La matrice M est donc (figure IIIA-5) :

	a 1	b 2	c 3	de 4
a 1			1	1
b 2			1	1
c 3	1	1		
de 4	1	1		

figure IIIA-5 : matrice de couverture de rectangles

Un rectangle de cette matrice booléenne est défini par un sous-ensemble de lignes et de colonnes (L,C) tel que Bij = 1, $\forall$ i ε L et $\forall$ j ε C. Dans notre exemple, ({3,4},{1,2}) est un rectangle.

On définit un rectangle premier comme un rectangle non totalement couvert par un autre. Dans l'exemple précédent, ({3,4},{1,2}) est un rectangle premier, par contre ({3},{1,2}) ne l'est pas.
L'approche consistant à ne traiter que les rectangles premiers est fondamentale car elle permet de ramener un problème NP-complet à un ordre de complexité raisonnable. Dans l'exemple, deux rectangles premiers sont déterminés R1=({3,4},{1,2}) et R2=({1,2},{3,4}).

A chacun des monômes de l'expression de f est alors associé un indice qui est reporté dans la matrice (figure IIIA-6). La raison de ceci est due au fait que certains éléments de la matrice correspondent au même monôme. Par exemple, M31 et M13 correspondent au monôme "ac".

$$f = ac + ade + bc + bde = m1 + m2 + m3 + m4$$

	a	b	c	de
a			m 1	m 2
b			m 3	m 4
c	m 1	m 3		
de	m 2	m 4		

figure IIIA-6

Un gain est alors affecté à chacun des rectangles premiers. Il correspond au gain de littéraux dans l'expression factorisée, pondéré par le nombre de points (c'est à dire de monômes) composant la rectangle dans M.
Le rectangle de plus grand gain est sélectionné. Il donne ainsi un ensemble de candidats compatibles entre-eux (R1 et R2 sont indifférents dans

l'exemple). Les indices des monômes auxquels ils appartiennent seront enlevés de M et cette opération est réitérée jusqu'à épuisement de rectangles premiers.

Par cette méthode, l'expression factorisée de f est :

$f = (a + b)(c + de)$.

Bien que la formulation de cette méthode semble intéressante, il s'avère que la complexité actuelle des circuits ne permet par toujours de l'appliquer de manière optimale. Par exemple, une expression qui comporte 100 monômes pour 10 variables peut facilement générer un million de rectangles. Dans ce cas, l'algorithme de synthèse peut faire un choix de la méthode en fonction de la complexité des expressions ou autoriser l'utilisateur à utiliser l'une ou l'autre de ces méthodes ce qui lui procure un bon compromis temps de calcul / qualité des résultats.

3.3 Substitution algébrique et booléenne

La procédure de substitution consiste à remplacer la sous expression précédemment sélectionnée dans l'ensemble des fonctions logiques. Cela revient à diviser les fonctions par cette sous expression.

Si f est une fonction et d la sous-expression alors f s'exprime par :

$f = qd + r$ où q est le quotient, r le reste et $qd \neq 0$.

Dans la définition précédente, toute manipulation logique génèrant une expression booléenne égale à la fonction initiale est permise. Par exemple, la fonction f=ab+c peut aussi s'exprimer par f=(a+c)(b+c). Cette division s'est opérée grâce à l'application de règles booléennes (cc=c ; c+ac=c). Par contre, la transformation de f=ab+ac+d en f=a(b+c)+d n'est qu'algébrique.

Ces deux types de transformation sont utilisables. La première, appelée division booléenne, donne des résultats toujours meilleurs que la seconde (la division algébrique) mais requiert de plus grands temps de calcul.
Ces méthodes sont souvent incluses dans les outils de synthèse classiques et sont appelées en fonction du compromis temps de calcul / qualité des résultats requis.

a) division algébrique :

La division algébrique est simple à implémenter et de complexité faible (définition XY). Elle extrait une somme de monômes dans la fonction booléenne originale. sans utiliser les règles propres à l'algèbre de BOOLE.
Par exemple, la division de f = ab + ac + bc + cd par d = a + b
donne le quotient q = c et le reste ab + cd.
Donc, f = c(a+b) + ab + cd

Cette méthode donne des résultats satisfaisants pour un faible temps de calcul.

b) division booléenne :

La division booléenne fait appel à des transformations élémentaires de l'algèbre de BOOLE (aa' = 0 ; aa = a ; a + a' = 1 ; a + ab =a). Elle utilise généralement des procédures propres à la minimisation (calcul de complément, preuve de tautologie, réduction).

Une des méthodes utilisées pour effectuer une division booléenne a été proposée par R.K. BRAYTON [BRA84]. Elle est basée sur le principe qu'une substitution booléenne utilise des conditions Φ-booléennes. La condition Φ-booléenne associée à t=g(x) correspond à Φ = t'g(x) + tg'(x). Ainsi, si on minimise la fonction f en respectant cette condition Φ-booléenne, la

variable interne q (sous sa forme normale ou complémentée) sera automatiquement substituée dans f.

Le problème de la division booléenne est donc réduit à une minimisation avec respect de la condition Φ-booléenne.

L'exemple suivant illustre les différentes étapes de cette méthode.

Exemple :

Effectuons la division booléenne de f = a + bc par q = a + b

Par division algébrique, aucune substitution n'est possible.

La condition Φ-booléenne associée à l'expression du diviseur q est donc :

Φ = q'.(a + b) + q.a'.b'.= q'a + q'b + qa'b'

Effectuons alors la minimisation de f par Φ. Pour cela, nous utilisons le tableau de KARNAUGH de la figure IIIA-7.

a b c q	00	01	11	10
00		ϕ	ϕ	ϕ
01	ϕ	1	1	1
11	ϕ	1	1	1
10		ϕ	ϕ	ϕ

figure IIIA-7 : minimisation de f par utilisation de Φ

On remarque notamment que le monôme "a" de f a été réduit à "aq" par l'utilisation du Φ-booléen.

Nous obtenons ainsi l'expression de f en fonction de q suivante :

f = qa + qc = (a + b)(a + c)

4. "Mapping" technologique

Les méthodes de *"mapping"* consistent à décomposer le graphe représentant l'ensemble des fonctions factorisées en sous graphes représentant les motifs de base correspondant aux éléments de bibliothèque.

Deux types de techniques ont été développées.

La première utilise une heuristique qui décompose le graphe du réseau booléen initial en une forêt d'arbres. La racine de chacun de ces arbres représente soit une sortie primaire, soit la sortie d'une porte dont la sortance est supérieure à 1. Ces arbres sont alors couverts par des arbres élémentaires correspondant aux éléments de bibliothèque par programmation dynamique [AHO76] ; c'est la méthode du *Tree Matching* (DAGON [KEU87], MIS[DET87]).

La seconde technique est une méthode à base de règles (système expert) inspirée par les techniques d'intelligence artificielle. Ces dernières méthodes procédent par une couverture progressive du graphe à l'aide des sous-graphes autorisés, jusqu'à obtenir une couverture totale. Le choix des sous-graphes est guidé par des considérations technologiques (SOCRATES [GRE86], DIRMAP [LEG88], ASYL [SAK90]).

Exemple :

Soient les fonctions booléennes f1 et f2 dont les formes après minimisation et décomposition sont les suivantes :

$f1 = ab + q$

$f2 = qd'$

$q = b'c$

Après "mapping" technologique, une représentation possible de ces fonctions est la suivante (figure IIIA-8).

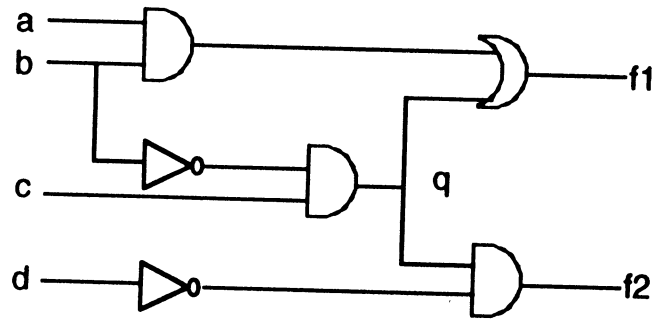


figure IIIA-8 : "mapping" technologique

Il est possible de réaliser la même fonction f1 et f2 à l'aide de représentations différentes en utilisant d'autres types de cellules disponibles dans la bibliothèque. Ces autres représentations auront des caractéristiques de vitesse ou de surface différentes.

Parallèlement à la représentation schématique à l'aide d'icônes, les fonctions logiques sont aussi toujours décrites sous la forme d'une liste de cellules et d'interconnexion entre elles. Ce format standard, appelé *"netlist"* dans la terminologie anglo-saxone, peut ainsi être utilisé dans différents compilateurs (figure IIIA-9).

```
B netlist
X 1 a I;
X 2 b I;
X 3 d I;
X 4 c I;
X 5 f1 o;
X 6 f2 o;
G 7 VDD;
G 8 VSS;
G 9 BULK;
M MDE an02d1 *g I A1 A2 Z I ;
M MDE In01d2 *g I I ZN I ;
M MDE or02d1 *g I A1 A2 Z I ;
I MDE an02d1 *g * 1 U1 I 1 2 12;
I MDE In01d2 *g * 2 U6 I 3 11;
I MDE In01d2 *g * 3 U7 I 2 10;
I MDE or02d1 *g * 4 U4 I 12 13 5;
I MDE an02d1 *g * 5 U3 I 13 11 6;
I MDE an02d1 *g * 6 U2 I 10 4 13;
E netlist
```

figure IIIA-9 : description textuelle d'une "netlist"

5. Optimisation de circuits :

Cette dernière étape consiste à optimiser le circuit obtenu après *"mapping"* technologique en fonction des spécifications définies par le concepteur. Cette optimisation est orientée soit en surface, soit en performances ou soit en terme de compromis vitesse/surface. D'autres paramètres peuvent guider cette optimisation tels que la consommation du circuit ou sa testabilité. Elle utilise les caractéristiques de la bibliothèque de cellules (équation de temps, capacités, surface, puissance) afin d'obtenir un nouveau circuit dont les caractéristiques correspondent au mieux à celles fixées par le concepteur.

Dans de nombreux systèmes MIS [BRA87], TECHMAP [HAC88]..., cette étape est rattachée au "mapping" technologique où la transformation des expressions factorisées en une interconnexion de cellules est guidée par les critères d'optimisation. Cependant, il s'avère que ce type de procédé est souvent très couteux en temps de calcul et prend mal en compte certains types de portes n'ayant pas une structure d'arbres (portes exclusives, multiplexeurs...). De plus, une telle approche reste très éloignée de la réalité physique du dispositif. Ainsi, afin de mieux prendre en compte les spécificités de la bibliothèque, nous avons séparés cette étape d'optimisation du "mapping". En outre, cette phase d'optimisation peut ainsi être également utilisée comme un outil indépendant.

Cette étape étant stratégique dans un outil de synthèse puisqu'elle permet de valider les spécifications de conception, nous avons étudié en détail, puis déterminé des techniques qui permettent d'en optimiser les résultats.

B. Optimisation temporelle de circuits combinatoires

Introduction

Parmi toutes les phases invoquées lors des opérations de synthèse automatique des circuits, la phase d'optimisation est certainement l'une des plus importantes. C'est pendant cette étape d'optimisation que l'on cherchera à optimiser le circuit en terme de surface ou en terme de performance afin de respecter au mieux les contraintes imposées par le concepteur (figure IIIB-1).

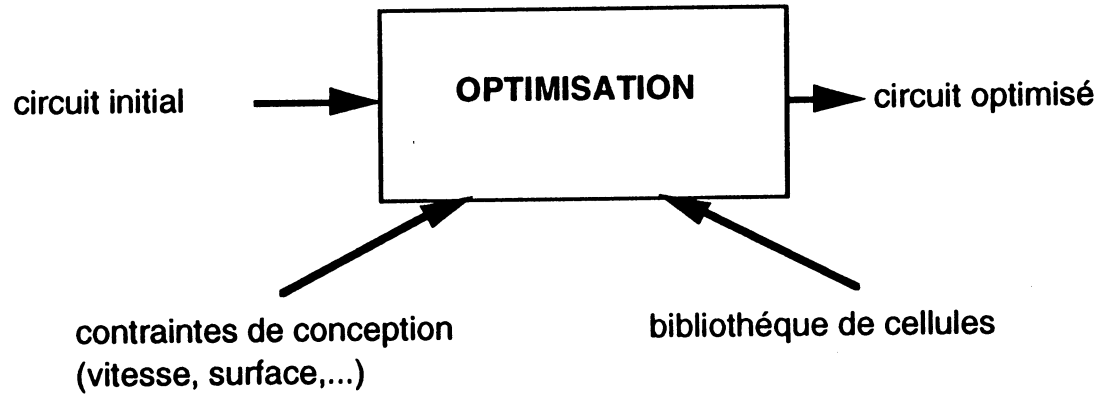


figure IIIB-1 : optimisation de circuits

L'ensemble des outils permettant la mise en oeuvre de cette phase d'optimisation forme un outil à part entière pouvant être totalement dissocié de la chaîne de synthèse. Ainsi, cet outil peut être utilisé séparément par un concepteur ayant défini une première implémentation de son circuit et voulant optimiser une partie ou l'ensemble de celui-ci ; une utilisation interactive <concepteur / outil automatique> peut s'avérer efficace.

Cet outil d'optimisation accepte en entrée un réseau de cellules ou "netlist" représentant le circuit à optimiser et un ensemble de contraintes à respecter. Il fournit en sortie une nouvelle "netlist" correspondant à un nouveau circuit ayant les mêmes caractéristiques fonctionnelles que le

précédent, mais répondant aux spécifications initiales et dont les performances sont améliorées.

Les contraintes intervenant lors de l'optimisation peuvent être de plusieurs ordres :

- contraintes imposées par l'application : souvent elles découlent d'exigences imposées par l'application dans laquelle sera utilisé le circuit en cours de conception. Il pourra s'agir de contraintes temporelles imposant des délais maximaux d'une ou plusieurs sorties, des contraintes de surface...

Souvent les contraintes requises sont incompatibles entre elles, le concepteur peut donc le cas échéant spécifier des compromis.

- contraintes imposées par l'environnement, les conditions dans lesquelles sera utilisé le circuit.
Ici, les contraintes seront énoncées en terme de température, de tension d'alimentation (VDD) ou des caractéristiques de la technologie utilisée.
Ces contraintes engendrent un facteur de corrélation permettant d'ajuster les caractéristiques des cellules de la bibliothèque.

Nous allons surtout nous intéresser au problème de l'optimisation temporelle d'un circuit multicouches. Il représente en fait le souci principal des concepteurs actuels. L'augmentation de surface du circuit combinatoire d'une dizaine de pour-cent ne prête souvent pas trop à conséquence. Par contre, le non respect d'une contrainte de temps pénalise souvent sérieusement l'application désirée.

Les méthodes d'optimisation temporelle d'un circuit multicouches sont soit algorithmique [HAC88], soit basées sur des transformations locales du circuit [JOY86]. Nous avons choisi l'approche à base de règles dont l'intérêt principal est une meilleure prise en compte des caractéristiques de la bibliothèque car à chaque bibliothèque peut être associé un jeu de règles qui lui est propre. De plus, ce jeu de règles peut être enrichi par le concepteur selon ses connaissances ou l'application qu'il désire concevoir.
Ces transformations ou règles sont successivement invoquées par l'outil, et le gain obtenu par application de la transformation est calcule afin de déterminer si la modification doit être retenue ou non.
Les transformations testées peuvent être très différentes et dépendent de l'objectif que l'on désire atteindre. Ce peut être une minimisation en surface [BAR86], [HAC88], [KEU87], une optimisation des performances [BAR86], [KIN89], [HAC88], [HOK87] ou une optimisation offrant un compromis vitesse/surface [SIN88].

De nombreuses techniques d'optimisation ont été décrites dans la littérature mais rares sont celles, à l'exception principalement de celles développées dans SOCRATES [GRE86] ou dans le système de synthèse d'AT&T [KEU88], qui ont proposé des méthodes d'optimisation temporelle efficaces.
Les dernières versions de DAGON [KEU87], MACDAS [SAS86] et MIS [DET87] sont elles surtout orientées vers l'optimisation de surface.

Dans une première partie, les principes de l'optimisation et les règles de transformation de bases seront énoncés.
Dans une seconde partie, une stratégie originale de décomposition de portes logiques de base (AND, NAND, OR et NOR) sera présentée. Cette

méthode permet d'obtenir des décompositions de portes logiques optimales en s'appuyant directement sur les modèles des cellules de la bibliothèque utilisée pour réaliser le circuit. Elle prend en compte les capacités de charge présentes sur les connecteurs des cellules et les temps d' établissements des signaux au niveau de ces connecteurs.

1. Principe de l'optimisation temporelle

Le principe de l'optimisation temporelle est simple. Il consiste, dans un premier temps, à déterminer l'ensemble des chemins critiques par utilisation d'un outil d'analyse temporel. Un chemin critique correspond à un chemin du réseau dont les temps d'établissement des sorties dépassent une contrainte fixée par le concepteur. Ensuite des règles de transformations locales sont appelées successivement sur chacun des chemins critiques. Ces règles d'optimisation locales correspondent à une ou plusieurs cellules interconnectées représentant une expression logique et pouvant être substituée à un ensemble de cellules du chemin critique s'il exprime la même fonction. Une estimation temporelle est alors effectuée afin de valider cette substitution.
A chaque fois qu'une substitution est réalisée, un nouveau calcul temporel est réalise sur tout le circuit afin de déterminer les nouveaux chemins critiques et d'affecter à chaque noeud du circuit ses nouvelles caractéristiques temporelles.
Cette opération est réitérée soit jusqu'à ce que les spécifications soient satisfaites, soit jusqu'à épuisement de toutes les règles susceptibles d'optimiser le circuit.

Calcul temporel :

Deux méthodes sont utilisées pour déterminer les caractéristiques temporelles du circuit, c'est à dire les temps d'établissement des signaux sur les sorties.

Tout d'abord, lors de l'évaluation d'une modification locale une estimation temporelle simplifiée est effectuée afin de ne pas trop pénaliser le temps

de calcul global du processus d'optimisation. Cette estimation utilise les pires cas des paramètres des cellules de la bibliothèque. Pour chaque porte, l'équation linéaire est la suivante :

$$t = D + R*Cld$$

D = le délai interne maximal de la porte ($D = \max(D_{rising}, D_{falling})$)
R = la rampe maximale de la porte ($R = \max(R_{rising}, R_{falling})$)

Cld représente la charge en sortie de la porte. Elle est calculée par la fonction linéaire :

$$Cld = incCap*nPins + defCap$$

incCap : capacité par défaut du connecteur de la porte
defCap : capacité par défaut de la connectique
nPins : nombre de connecteurs sur l'équipotentielle de sortie (ce qui correspond à la sortance de la porte plus 1).

Ensuite, un calcul précis des temps de propagation sur l'ensemble du circuit est effectue afin de vérifier si les spécifications du concepteur sont respectées. Pour cela, un outil appelé "vérificateur temporel" calcule les valeurs temporelles exactes en chaque noeud du circuit pour obtenir les délais d'établissement de chaque sortie primaire. Cet outil utilise les modèles des cellules de la bibliothèque (temps de montée et de descente des portes), ainsi que les capacités de connexion [KIN88] ; il permet ainsi de faire apparaître les chemins critiques.

Règles de base utilisées lors de l'optimisation :

Six règles de base sont successivement invoquées par le programme pour essayer d'optimiser les performances d'un réseau. Ces règles de transformations sont définies de manière procédurale dans le système.
Ces transformations sont : amplification de signaux, élimination des inverseurs de sorties, optimisation booléenne, utilisation des lois de DE MORGAN, élimination des portes complexes, remplacement des portes classiques par leur version rapide et ajout de portes exclusives.

Chacune de ces règles de transformation est détaillée ci-après :

a) Amplification de signaux :

Le temps de traversée d'une porte est fonction à la fois de ses caractéristiques internes et de la charge qu'elle voit en sortie. Le calcul de cette charge a été préalablement défini ; il montre qu'il est fonction de la somme de toutes les capacités d'entrée des portes connectées à la sortie de la cellule.
L'idée consiste à amplifier les sorties fortement chargées par utilisation de portes amplificatrices ou "buffers", de telle sorte que la capacité vue par la porte ne dépende que de la capacité d'entrée du buffer (figure IIIB-2).

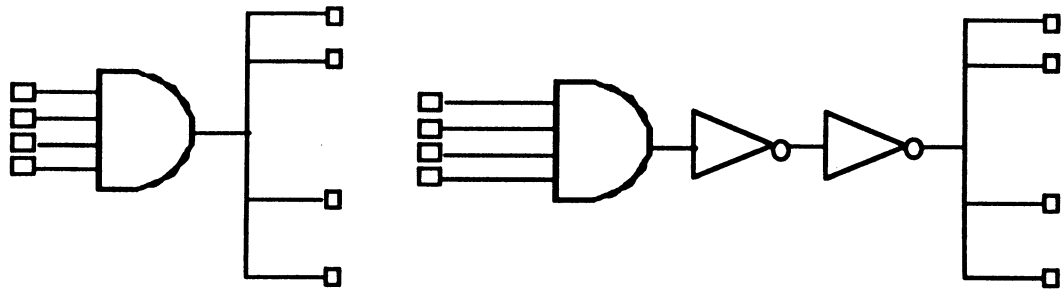


figure IIIB-2 : amplification de signaux

b) Elimination des inverseurs de sortie :

Cette règle remplace les portes négatives (resp. positives) suivies par un inverseur par les portes positives (resp. négatives) correspondantes (figure IIIB-3).

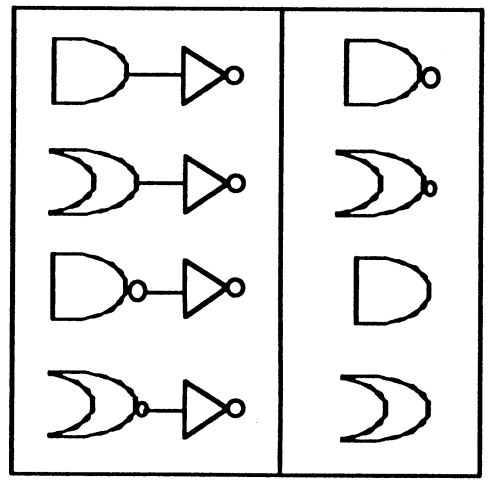


figure IIIB-3 : élimination des inverseurs de sortie

c) Optimisation booléenne :

L'optimisation booléenne consiste à extraire un bloc logique du circuit combinatoire. Il est alors exprimé sous forme de fonction booléenne qui

est optimisée par les diverses étapes de la synthèse logique (minimisation, décomposition et "mapping") pour obtenir un nouveau circuit.

L'exemple suivant illustre d'une transformation possible opérée grâce à cette règle. (figure IIIB-4).

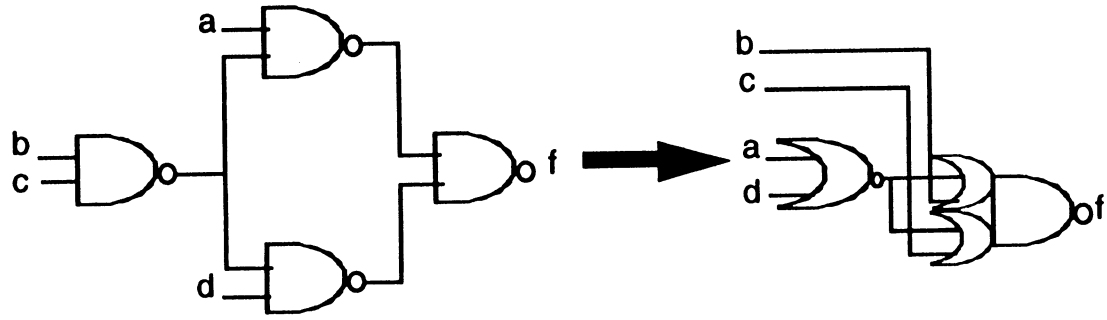


figure IIIB-4 : exemple d'optimisation

Les étapes de cette transformation sont les suivantes.

Après extraction du réseau, son expression polynomiale est calculée :

f = (((bc)'.a)'.((bc)'.d)')' = (bc)'.a + (bc)'.d = ab' + ac' + b'd + c'd

La minimisation de cette expression n'aboutit pas à aucune amélioration, il s'en suit alors une factorisation :

f = a(b' + c') + d(b' + c')=(a + d) . (b' + c')=(a'd' + bc)'=((a'd' + b).(a'd' + c))'

Enfin, le "mapping" technologique donne :

f = AOI22((NOR(a,d),b)(NOR(a,d),c))

d) Règles de DE MORGAN :

En technologie CMOS, les transistors du réseau P sont plus lent que ceux du réseau N. Selon la charge de sortie, il arrive qu'un configuration and(inv(a),inv(b)) soit plus rapide que nor(a,b). Cela correspond à une transformation de DE MORGAN dont les autres sont énoncées ci-après :

nand(a,b) = or(inv(a),inv(b))

nor(a,b) = and(inv(a),inv(b))

and(a,b) = nor(inv(a),inv(b))

or(a,b) = nand(inv(a),inv(b))

e) Elimination des portes complexes :

Les portes complexes permettent de réaliser des fonctions booléennes complexes. Par exemple, la cellule AOI33 représentée ci-dessous exprime la fonction G=[(a+b+c)(d+e+f)]' (figure IIIB-5).

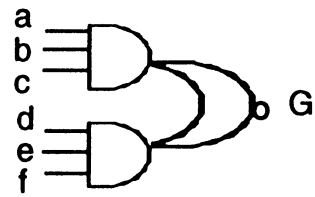


figure IIIB-5 : porte complexe AOI33

C'est une manière compacte d'exprimer une fonction relativement complexe. Mais pour les mêmes raisons que précédemment, ces portes s'avèrent lentes puisque leur étage de sortie est composé d'un grand nombre de transistors. La représentation avec des portes classiques accélère dans presque tous les cas le circuit. Cette transformation est simple, par exemple la porte AOI33 est représentée par nor(and(a,b,c),and(d,e,f)).

f) Utilisation de portes plus rapides :

Dans les nouvelles bibliothèques, chaque porte est concue de deux manières différentes : une version standard qui allie un compromis vitesse/surface et une version rapide ou "double drive" qui est uniquement dimensionnée pour les performances temporelles. En fonction de la position de la porte sur le chemin critique ou non, l'une ou l'autre des versions est utilisée.

g) Ajout de portes exclusives :

Les portes OU-exclusif (XOR et XNOR) permettent également de représenter de manière compacte une fonction logique qui nécessiterait plusieurs cellules de bibliothèque ainsi que leurs inter-connexions. L'implémentation d'une même fonction logique est donc réduite a une seule cellule au lieu de trois cellules avec ses connexions, d'où un gain en nombre de transistors et en surface de routage. Un type de substitution opéré par cette règle est représenté ci-après (figure IIIB-6) :

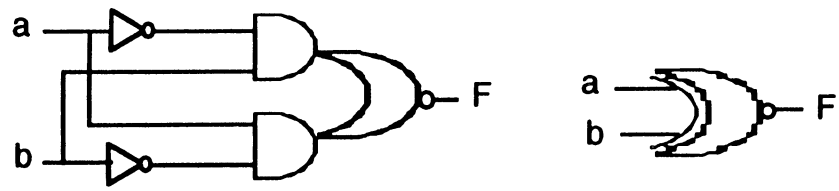


figure IIIB-6 : substitution par des portes exclusives

2. Méthode de décomposition de portes logiques

2.1 Introduction

Dans les paragraphes précédents, nous avons introduit de nombreuses approches qui permettent, à l'aide de différentes techniques de transformation, d'optimiser les performances du circuit.
Nous allons présenter une technique originale permettant de réaliser une optimisation temporelle des circuits [PAU89].

Le principe de base de la décomposition de portes logiques à des fins d'amélioration des performances, consiste à remplacer dans le circuit les portes logiques complexes ayant un grand nombre d'entrées par un ensemble équivalent de portes logiques plus simples dont le nombre d'entrée restera limité (figure IIIB-7).

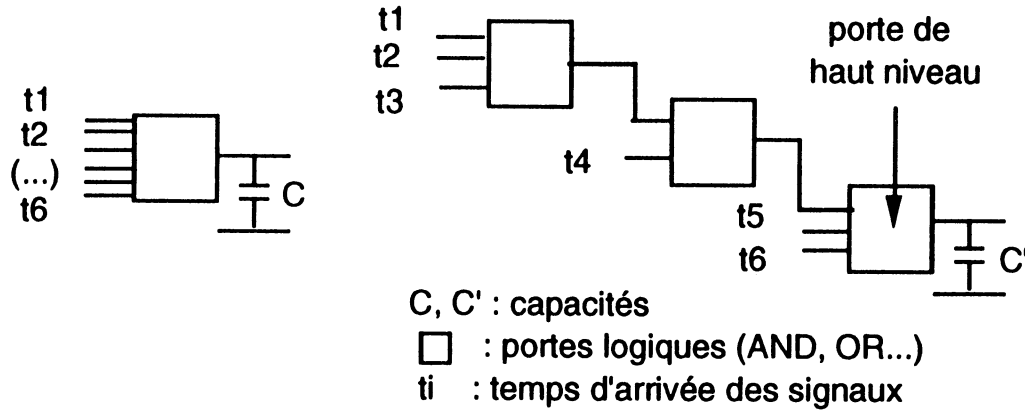


figure IIIB-7 : décomposition de portes logiques

L'intérêt d'une telle approche est double. D'abord, elle permet de réduire le nombre de transistors de la porte logique de haut niveau, c'est à dire

celle qui est le plus proche des sorties primaires. La réduction du nombre de transistors de cette porte induit une diminution de sa capacité de sortie qui est corrélée à l'entrance de la porte (C' < C), d'où une amélioration globale du temps de commutation.
Ensuite, un aspect original de la méthode de décomposition proposée ici, est qu'elle permet de tenir compte lors de la synthèse du circuit des délais d'arrivée associés à chacun des signaux d'entrées du circuit. En effet, dans le cas de signaux d'arrivée non synchrones, il est possible lors de la décomposition de réaliser un circuit dont la structure finale sera telle que les signaux d'entrée arrivant le plus en retard aient moins de logique à traverser.

Par rapport à l'environnement dans lequel a été implémenté cette technique de décomposition, les portes de base de notre bibliothèque qui sont candidates à une décomposition sont les portes AND/NAND et OR/NOR. Les autres types de portes (complexes et disjonction) ne peuvent pas être décomposés.
D'une manière générale, seules les portes AND/NAND seront implémentés sur le chemin critique car elles sont plus rapides que leurs duales OR/NOR. En effet, en CMOS statique les transistors P (en série pour les portes OR/NOR) sont plus lents que les transistors N. C'est d'ailleurs pour cette raison que certains outils de synthèse traitant de l'optimisation temporelle utilisent généralement une représentation interne en portes NAND, AND et inverseurs pour représenter les fonctions logiques [DET87], [HAC88] et [SIN88].
Ainsi, les portes AND et NAND sont généralement mises sur le chemin critique alors que les portes OR, NOR ou complexes (AOI et OAI) se trouvent hors du chemin critique pour minimiser la surface. Ces dernières ne sont donc guère candidates à une optimisation temporelle.

Nous nous sommes donc focalisés sur une méthode permettant de décomposer une porte AND (ou NAND) à entrance trop grande, par un arbre de portes AND (NAND) à entrance plus faible.

2.2 Etat de l'art

Différentes méthodologies basées sur le principe de décomposition élémentaire de portes ont été développées. Nous allons rappeler ici leurs principes et leurs limitations.

Le système SOCRATES [GRE86], exploite un ensemble de règles qui tendent à déplacer les signaux critiques vers les portes de plus haut niveau, portes qui commandent directement les charges du circuit, de telle sorte que les signaux les plus lents aient moins de logique à traverser. Néanmoins, aucune décomposition de portes logiques n'est effectuée et les capacités de charge ne sont pas prises en considération.

HOFMANN et KIM [HOF87] utilisent une méthode de décomposition binaire de portes logiques afin d'en améliorer la sortance par réduction du nombre de transistors de la porte de haut niveau. La décomposition binaire d'une porte consiste en 3 portes ; une de haut niveau à 2 entrées et deux autres attaquant celle-ci à n/2 entrées. Par exemple, le décomposition binaire d'une porte AND à 9 entrées est la suivante :

AND2(AND5, AND4).

Cependant, les capacités de charge ainsi que les temps d'arrivée des signaux ne sont pas pris en compte car seule une amélioration de la sortance de la porte de haut niveau est recherchée.

Le système TECHMAP [HAC88] utilise une heuristique plus sophistiquée qui est aussi basée sur un décomposition binaire de portes (NAND dans ce cas) mais qui prend en compte les temps d'arrivée des signaux. Cette décomposition est de la forme :

NANDn = NAND2 (INV(NANDx), INV(NANDy)) où x + y = n

Après avoir effectue un ordonnancement des entrées en fonction de leur temps d'arrivée, un algorithme récursif détermine par une démarche exhaustive les valeurs des entrances (x et y) des portes NAND de bas niveau permettant une optimisation temporelle. Cette opération est réitérée sur ces portes (NANDx et NANDy) par récursivité. Néanmoins, cette méthode est restreinte à des décompositions binaires et ne considère donc pas le cas général des décompositions n-aires. De plus, les capacités de sorties des portes ne sont pas prises en compte et il n'est pas tenu compte de l'excès de surface induit par la décomposition (grand nombre de portes NAND à deux entrées, d'inverseurs et ajouté au routage induit par leur connexion).

Finalement, d'autres approches proposées par SINGH [SIN88] et KEUTZER [KEU88] sont aussi basées sur des décompositions en portes à deux entrées et ont les mêmes limitations. Cependant, dans le système de SINGH, un compromis vitesse/surface est recherché lors des décompositions. KEUTZER quant à lui, considère la charge de sortie mais seulement lors d'une étape indépendante d'optimisation de phases.

2.3 Modèles linéaires

Durant le processus d'optimisation et lors de l'application des transformations, un point important afin d'obtenir un algorithme de décomposition simple et rapide est d'utiliser des méthodes simples et rapides d'estimation de pire cas des délais des portes. Dans notre approche, cette estimation temporelle est réalisée à l'aide de modèles linéaires simples d'estimation de délais. Ces modèles ont été validés sur quatre bibliothèques de cellules prédiffusées ou précaractérisées de VLSI TECHNOLOGY ([VSa88], [VSb88], [VGa88] et [VGb88]). De plus, nos investigations ont montré que de nombreuses autres bibliothèques de cellules exploitent les mêmes caractéristiques et peuvent ainsi employer les mêmes modèles.

Ces modèles permettent de réaliser facilement une formulation analytique des conditions de réduction maximale des délais, procurant ainsi un algorithme simple de décomposition optimale d'une porte à n entrées en un ensemble de portes de degré inférieur. Ces conditions sont déterminées facilement et rapidement d'où leur intérêt lorsque l'on traite des circuits de grande complexité. Ces modèles fournissent également une estimation rapide du compromis vitesse/surface engendré par la décomposition.

a) Modèles linéaires des portes ET et OU :

Le modèle utilisé est une fonction linéaire du nombre d'entrées de la porte. Il dérive d'une extrapolation linéaire des courbes délais/charge/nombre d'entrées.

Le premier modèle de délai est défini pour les portes AND (un modèle similaire peut être établi pour les portes OR). Il utilise le pire cas entre les temps de montées et de descentes. Il est de la forme :

$$Da\ (n,C_L) = K_d + K_n * n + K_c * C_L$$

n = nombre d'entrées

C_L = capacité de charge

K_d, K_n, K_c = constantes technologiques

Il est supposé que les capacités d'entrée Ci sont identiques pour toutes les entrées des portes AND de la technologie.

Les coefficients de corrélation de ce modèle sont supérieurs à 0.99 pour les portes AND et OR des quatre bibliothèques testées. La simplicité de ce modèle provient du fait que la capacité de charge interne est celle d'un inverseur (formant la porte de haut niveau de la NAND ou NOR). Elle est donc indépendante du nombre d'entrées.

Pour les cellules prédiffusées ou précaractérisées, la surface des cellules est une fonction linaire du nombre d'entrée :

$$S_{AND}\ (n) = K_{Ad} + K_{An} * n$$

b) Modèles linéaires des portes NAND et NOR

Le second modèle est plus complexe puisque les capacités de sorties internes dépendent du nombre d'entrées n de la porte. Le coefficient

modulant la charge dépend aussi de n puisqu'il n'y a pas d'inverseurs connectés à la sortie.
Ces considérations nous ont conduit au modèle suivant :

$$Dn\ (n,C_L) = K_d + K_n * n + K_c * C_L + K_{nc} * n * C_L$$

Le facteur de corrélation pour les bibliothèques testées est de 0.98 dans ce cas.

La surface d'une porte NAND à n entrées est donnée par :

$$S_{NAND}\ (n) = K_{Nd} + K_{Nn} * n$$

Ces modèles seront utilisés par la suite pour définir les conditions d'application des décompositions de logique équilibrées et non équilibrées.

2.4 Décomposition de portes AND

Nous devons considérer deux classes de décompositions. La première est une décomposition non équilibrée utilisée lorsque les temps d'arrivée sur les connecteurs d'entrée des cellules sont différents. La seconde est une décomposition équilibrée qui est utilisée lorsque les temps d'arrivée sont identiques.

a) Décompositions non équilibrées de portes AND :

Un exemple de décomposition non équilibrée est donné en figure IIIB-8. Les temps ti représentent les temps d'arrivée des signaux sur les entrées. Ils sont classes par ordre de temps d'arrivée croissant ($t_{i+1} > t_i$). L'objectif est de définir une équation de gain associée à la décomposition.

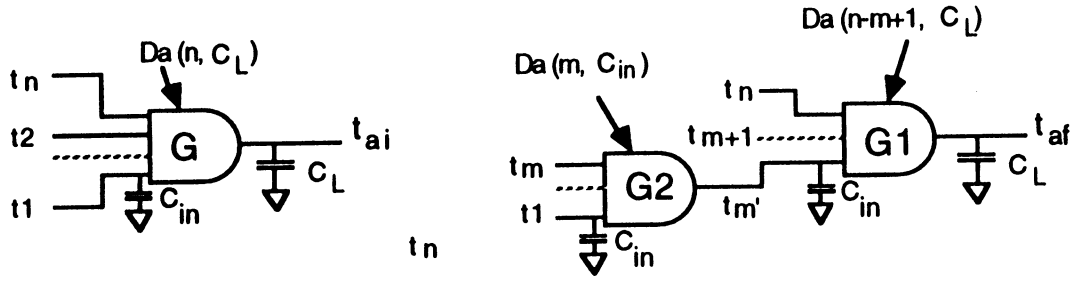


figure IIIB-8 : décomposition non équilibrée de porte AND

En utilisant le modèle précédemment défini, les temps d'arrivée finaux t_{ai} et t_{af} sont donnés par :

$$t_{ai} = t_n + D_a(n, C_L) = t_n + (K_d + K_n \cdot n + K_c \cdot C_L)$$
$$t_{af} = D_a(n\text{-}m+1, C_L) + \max[t_{m'}, t_n], \text{ où : } t_{m'} = t_m + D_a(m, C_{in})$$

$C_{in} = C_i + 2 \cdot C_c + C_w$
C_i = capacité d'entrée d'une porte AND (identiques pour toutes les portes AND).
C_c = capacité d'un connecteur par défaut.
C_w = capacité de connexion par défaut.

Deux cas sont à considérer :

Cas 1 : si $t_{m'} < t_n$ alors le temps d'arrivée t_{af} dépend seulement de t_n et du délai à travers la dernière porte :

$t_{af} = t_n + D_a(n\text{-}m+1, C_L)$, après simplification on obtient :

$$\delta t_{(1)} = t_{ai} - t_{af} = K_n \cdot (m - 1)$$

En d'autres termes le gain temporel dépend uniquement de m dans le cas 1. La ligne horizontale de la figure IIIB-9 ci-dessous représente l'équation

$\delta t_{(1)}$ pour différentes valeurs de m. Dans cette même figure, on définit également les points de transition δ_m, c'est à dire les valeurs de $(t_n - t_m)$ pour lesquelles $t_m' = t_n$. Ils peuvent être calculées par :

$$\delta_m = D_a\,(m, C_{in})$$
$$= (K_d + K_n \cdot m + K_c \cdot C_{in}), \quad \text{(fonction linéaire de m).}$$

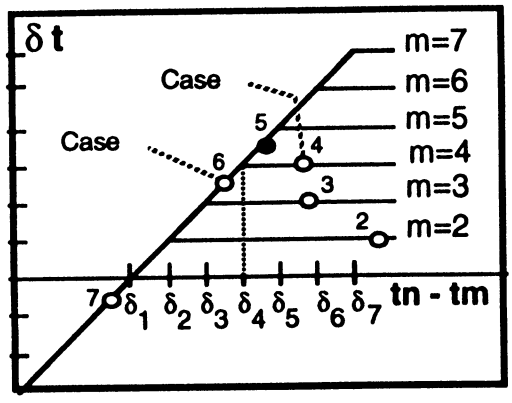
Cas 2 : si $t_m' > t_n$ alors on obtient le gain suivant :

$$\delta t_{(2)} = (t_n - t_m) - (K_d + K_n + K_c \cdot C_{in}) = (t_n - t_m) - D_a\,(1, C_{in})$$
$$= (t_n - t_m) - \delta_1$$

Dans ce cas, le gain est indépendant de *m* et est proportionnel à $(t_n - t_m)$. L'équation $\delta t_{(2)}$ est représentée par la ligne oblique dans la figure IIIB-9. La valeur de δ_1 indique la valeur minimale de $(t_n - t_m)$ requise pour obtenir un gain temporel. Si la valeur maximale de $(t_n - t_m)$ est inférieure à δ_1, alors une décomposition équilibrée peut être envisagée. Ceci est particulièrement intéressant pour les portes NAND qui sont décrites plus loin.

Le gain en surface est donné par :

$$\delta S_{AND} = S_{ANDf} - S_{ANDi} = K_{Ad} + K_{An}$$




```
AND Decomp (G, n):
  For m=2 To n-1 Do:
    If (tn - tm ) > δm-1
    And (tn - tm+1 ) < δm
    Then split (G,n) into (G1,n-m+1)
                      and (G2, m) ;
  End ;
  If output of G2 critical Then
     AND Decomp (G2, m) ;
  AND Decomp (G1, n-m+1) ;
End ;
```

figure IIIB-9. Gain temporel et pseudo-code de décomposition non équilibrée de portes AND

Les cercles de la figure IIIB-9 représentent les points expérimentaux d'une porte AND à 8 entrées. Les six cercles (pris de la droite vers la gauche) correspondent aux valeurs de (t_8-t_m) pour m=2 à 7. Par une simple analyse du graphe, nous obtenons la condition de gain maximal, comme il est décrit dans l'algorithme de la figure IIIB-9.
Pour l'exemple ci-dessus, la condition est remplie pour m=5 puisque :

$$(t_8-t_5) > \delta_4 \quad \text{et} \quad (t_8-t_6) < \delta_5.$$

L'algorithme est récursif car les portes peuvent être décomposées à nouveau. Ceci est illustré dans la figure IIIB-10 utilisant une porte AND à six entrées. Le résultat final (figure IIIB-10c) est atteint après trois décompositions.
Dans la figure IIIB-10d, nous montrons le résultat découlant d'une décomposition en portes à deux entrées utilisant l'algorithme TECHMAP [HAC88] (quatre décompositions sont requises dans ce cas). Il apparait que le délai et la surface sont plus grands que pour la décomposition utilisant des portes de haut niveau à n entrées plutôt que des portes à deux entrées. Le compromis vitesse/surface est donné en figure IIIB-11 pour

les deux méthodes. La première représente les trois itérations de l'algorithme de décomposition en portes à n entrées. La seconde donne les valeurs surface/vitesse pour une décomposition en portes à 2 entrées.

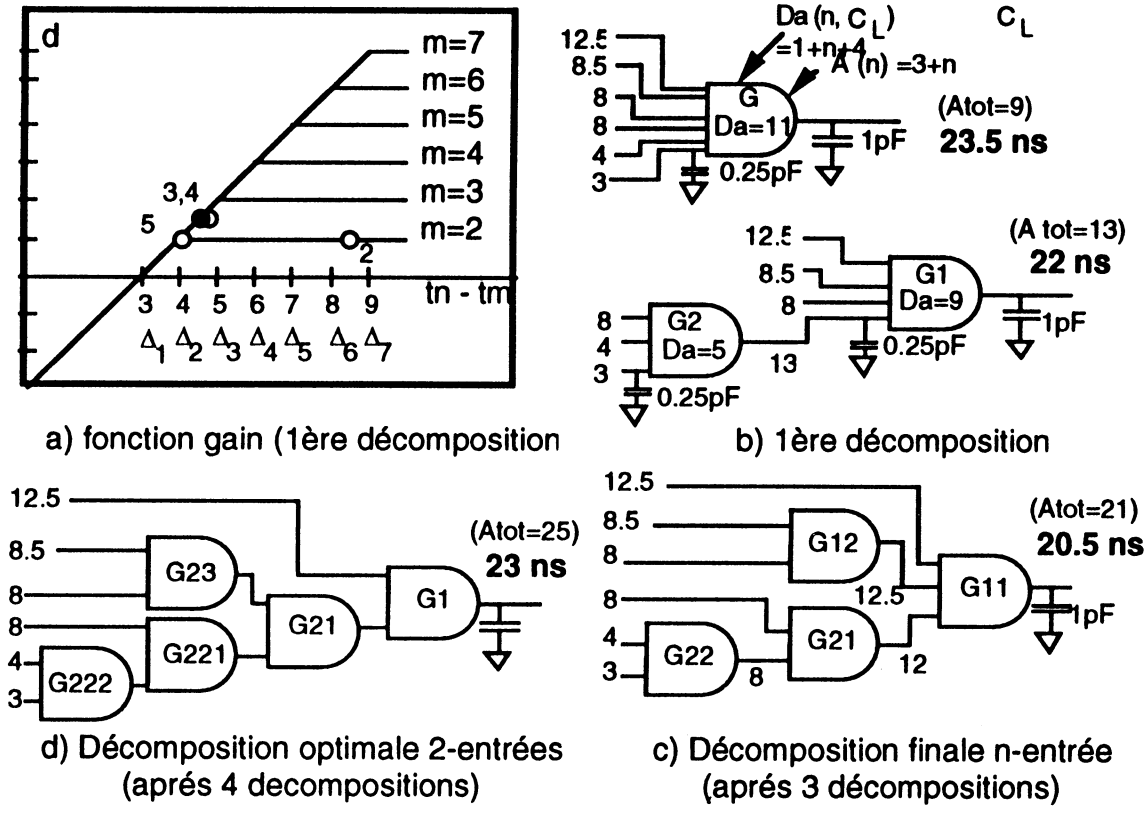


figure IIIB-10 : Exemple de décompositions non équilibrées de portes AND utilisant une stratégie à n-entrées a),b),c) et 2-entrées d)

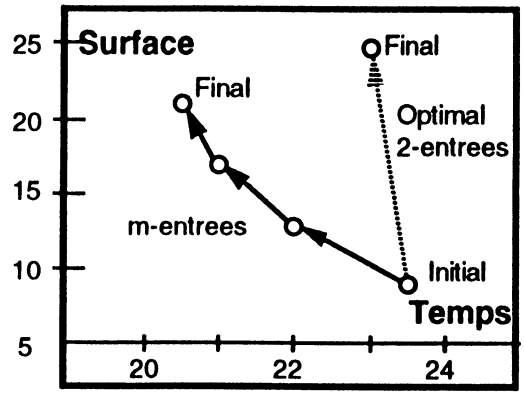


figure IIIB-11 : compromis vitesse / surface

Ce résultat illustre un aspect important des décompositions de portes. Chaque décomposition implique une augmentation du nombre de cellules induisant une augmentation de la surface de routage, d'ou un accroissement de la surface de silicium. Dans notre algorithme, m est maximisé à chaque itération, le nombre de décompositions nécessaires (trois dans notre exemple) est donc minimal. De plus, les portes utilisées ont le plus grand nombre d'entrées possibles permettant ainsi d'optimiser le gain temporel. Ceci n'est pas le cas lors d'une décomposition en portes à deux entrées (quatre décompositions sont nécessaires dans l'exemple).

b) Décompositions équilibrées de portes AND :

La seconde classe de décomposition utilisable avec les portes AND est la décomposition équilibrée.
Après simplifications, l'équation de temps devient :

$$\delta t = t_{ai} - t_{af} = K_n \cdot (n/2 - 2) - K_d - K_c \cdot C_{in}$$

et la condition pour $\delta t > 0$ est :

$$n > 2(K_d + K_c \cdot C_{in}) / K_n + 4$$

Il a été remarqué empiriquement que cette condition est généralement réalisée pour de grandes valeurs de n (approximativement quinze). La décomposition équilibrée des portes AND apparait donc peu intéressante. Nous allons cependant voir dans la suite que cette constatation n'est pas valable dans le cas de décomposition de portes NAND.

2.5 Décompositions de portes NAND

Une approche similaire de celle développée précédemment permet d'obtenir les équations de gains temporels des décompositions de portes NAND.

a) Décompositions non équilibrées de portes NAND :

Une décomposition à m-entrées est donnée dans la figure IIIB-11.

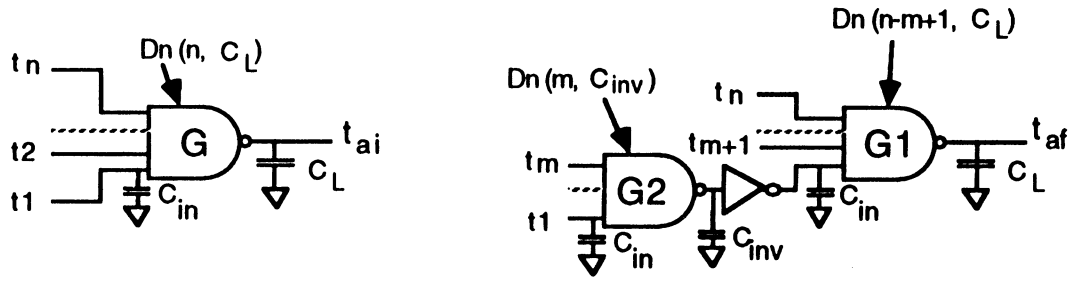


figure IIIB-11 : Décomposition non équilibrée de portes NAND

L'augmentation de surface est donnée par :

$$\delta S_{NAND} = S_{NANDf} - S_{NANDi} = K_{Nd} + K_{Nn} + S_{inv}$$

Dans le cas d'une décomposition de portes NAND deux cas sont à considérer pour le calcul du gain temporel. Le premier est quand $t_{m'} < t_n$:

$t_{af} = t_n + D_n\ (n\text{-}m\text{+}1,\ C_L)$,ce qui donne :

$\delta t_{(1)} = (K_n + K_{nc} \cdot C_L) \cdot (m - 1)$

Le second cas ($t_m' > t_n$), aboutit à l'équation suivante :

$$\delta t_{(2)} = (t_n - t_m) - K_3 - K_4 \cdot C_L + K_{nc} \cdot (C_L - C_{inv}) \cdot m$$

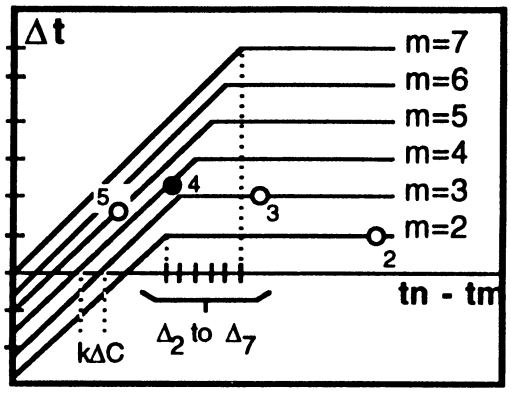
où K_3 and K_4 sont des constantes qui sont fonctions de K_d, K_n, K_c, C_{inv} et D_{inv} (C_{in}). Ces résultats sont illustrés graphiquement dans la figure IIIB-12 pour un ensemble de valeurs de C_L (où $C_L > C_{inv}$). Ces courbes sont similaires à celles obtenues pour la décomposition de portes AND modulo un décalage de la partie linéaire dû à la capacité de charge. Ce décalage est proportionnel à $(C_L - C_{inv})$. C_{inv} étant la plus petite capacité possible, le cas où $C_L < C_{inv}$ n'est pas traité. Les points de coupure sont indépendants de C_L et peuvent être obtenus par l'équation suivante :

$$\delta_m = D_{inv}\ (C_{in}) + D_n\ (m,\ C_{inv})$$

qui ne dépend que de m. Ce qui signifie que l'on peut utiliser le même type d'algorithme que celui des portes AND. La condition de gain maximal est simplement la valeur maximale de m telle que :

$$(t_n - t_m) > \delta_{m-1} - K_{nc} \cdot (C_L - C_{inv})$$

L'algorithme qui en découle est donné par la figure IIIB-12 :




```
NAND Decomp (G, n):
   For m=2 To n-1 Do:
      If (tn - tm ) > Δm-1 – k·ΔC
      And (tn - tm+1 ) < Δm – k·ΔC
      Then split (G,n) into (G1,n-m+1)
                          and (G2, m) ;
   End ;
   If output of G2 critical Then
       NAND Decomp (G2, m) ;
   NAND Decomp (G1, n-m+1) ;
End ;
```

figure IIIB-12 : gain temporel pour une décomposition non équilibrée de portes NAND

La figure IIIB-13 illùstre différentes décompositions dues à plusieurs charges. Comme il l'a été démontré, chaque charge implique une décomposition différente. Les valeurs en gras sont celles obtenues par application de l'algorithme. Les délais sont calculés en utilisant les valeurs de la technologie CMOS 1.5 micron de cellules précaractérisées de VLSI TECHNOLOGY.[VSb88].

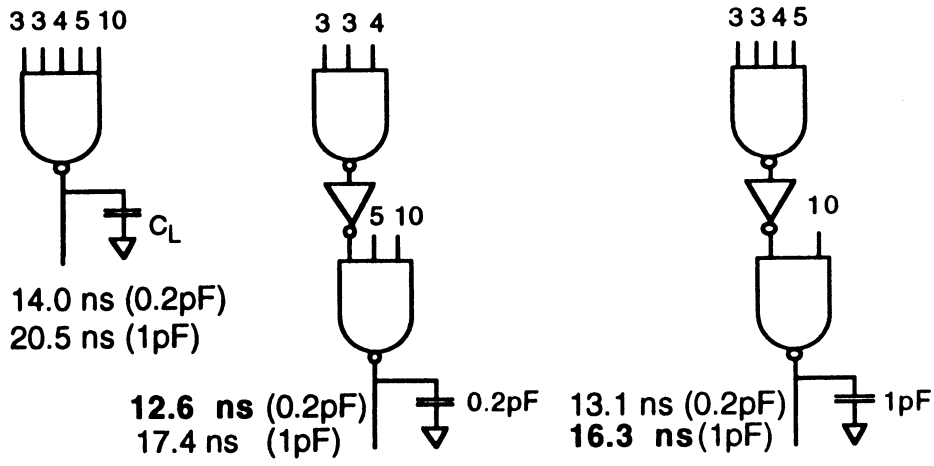


figure IIIB-13 : exemple de décompositions non équilibrées de portes NAND.

b) Décompositions équilibrées de portes NAND :

Une décomposition équilibrée de portes NAND est donnée par la figure IIIB-15 :

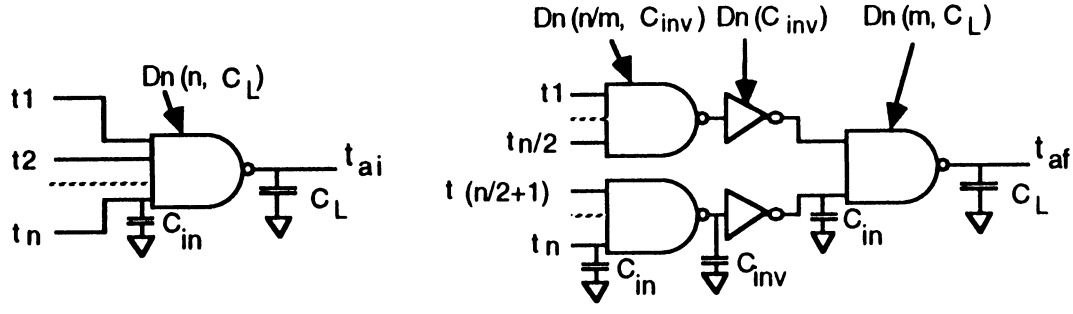


figure IIIB_15 : décomposition équilibrée de portes NAND

L'augmentation de surface est le double de celle obtenue lors d'une décomposition non équilibrée ; l'équation est de la forme :

$$\delta S_{NANDb} = 2 (K_{Nd} + K_{Nn} + A_{inv})$$

En utilisant les équations précédentes, on obtient les temps suivants :

$t_{ai} = t_n + D_n (n, C_L)$

$t_{af} = t_n + D_n (n/2, C_{inv}) + D_{inv} (C_{in}) + D_n (2, C_L)$ où,

C_{inv} = capacité d'entrée de l'inverseur ajoutée de la capacité par défaut du connecteur et de la capacité de connectique.

C_{in} = capacité d'entrée de la porte NAND de haut niveau ajoutée de la capacité du connecteur et de la connectique.

$D_{inv} (C_{in})$ = délai à travers l'inverseur pour une charge C_{in}. Cette valeur est obtenue en utilisant le modèle de la bibliothèque.

Cela donne l'équation de gain temporel suivante :

$$\delta t = -K_1 + K_2 \cdot n + K_{nc} \cdot (n-2) \cdot C_L$$

c'est à dire :

$$\delta t > 0 \quad \text{quand :} \quad C_L > (K_1 - K_2 \cdot n) / (n-2) \cdot K_{nc}$$

K_1 et K_2 sont des constantes qui sont fonction de K_d, K_n, K_c, K_{nc}, C_{inv} et de $D_{inv} (C_{in})$. Ainsi, pour une valeur de n donnée, il possible de définir une valeur minimale de capacité de charge permettant une décomposition favorable (donnant $\delta t > 0$).

Cela nous conduit à définir deux étapes pour l'algorithme :

1. Procéder à une décomposition non équilibrée utilisant l'algorithme de la figure IIIB-12
2. Pour les portes restantes, lorsque la capacité est supérieur à cette capacité minimale, procéder à une décomposition équilibrée.

Pour les systèmes dont le but est de réaliser une décomposition en vue d'un compromis vitesse/surface (par exemple [SIN88]), l'évaluation de $\delta A / \delta t$ grâce aux équations définies précédemment implique systématiquement de réaliser une phase de décomposition non équilibrées avant les décompositions équilibrées.

2.6 Adaptation de l'algorithme à toute bibliothèque

Pour les bibliothèques qui n'obéissent pas aux modèles simplifies de la section 2.3, ou lorsqu'une grande précision est requise, les modèles simplifiés peuvent être substitués de telle sorte que chaque porte est définie par sa propre équation linéaire. Le délai d'une porte à n entrées est alors défini par :

$$D_g\ (n,C_L) = K_d^n + K_c^n \cdot C_L \quad , \quad \text{où :}$$

$K_d^n, K_c^n =$ constantes technologiques spécifiques d'une porte de n entrées.

Les courbes des figures IIIB-9 et IIIB-12 doivent être reconsidérées cas par cas selon chaque valeur de n. Par exemple, il peut être démontré que

la décomposition de portes NAND (NOR) a une équation de gain temporel de la forme :

$$(t_n - t_m) > K_1 + K_2 \cdot C_L \quad \text{où :}$$

$$K_1 = (K_d^{n-m+1} - K_d^{n-m+2}) + (K_d^m + K_c^m \cdot C_{inv}) + D_{inv} (C_{in})$$
$$K_2 = (K_c^{n-m+1} - K_c^{n-m+2})$$

Pour les décompositions non équilibrées de portes AND (OR), la condition est la même si ce n'est que le terme D_{inv} (C_{in}) (délai de l'inverseur) est supprimé et que C_{inv} est remplacé par C_{in}.
De plus, dans la plupart des bibliothèques, le terme K_2 est égal à 0 puisque le "slew rate" est généralement identique pour toutes les portes AND.

En d'autres termes, les mêmes algorithmes que ceux des modèles simplifiés sont utilisables à l'exception des conditions de décomposition qui sont remplacées par celles données ci-dessus. Ainsi, une table de (n-2) valeurs de K_1 et K_2 doit être construite en début d'algorithme en fonction des paramètres de la bibliothèque utilisée. Une bibliothèque dont le nombre maximal d'entrées est de huit, implique le calcul et la mémorisation de 42 valeurs. Dans le cas ou le nombre d'entrées est supérieur à 8 (cela se produit dans les étapes intermédiaires de la décomposition), les modèles linéaires servent à extrapoler les valeurs.

2.7 Résultats expérimentaux et conclusion

Le tableau de la figure IIIB_16 montre les résultats obtenus dans l'environnement de synthèse logique de VLSI TECHNOLOGY. Les exemples utilisés sont les exemples standards de synthèse logique [MCN89]. Afin de

mettre en évidence les effets de l'algorithme implémenté, trois types de résultats sont donnés :

1. Caractéristiques du circuit après l'utilisation des règles de base de l'outil d'optimisation de circuit mais sans l'algorithme de décomposition. Il est à remarquer que la nouvelle "netlist" est de l'ordre de 20 à 30 % plus rapide que celle initialement spécifiée.

2. Le même circuit mais avec l'ajout de la méthode de décomposition 2-entrées (méthode de recherche exhaustive).

3. Résultats finaux incluant l'algorithme de décomposition n-entrées décrit dans cette partie.

La taille des circuits est donnée en terme de portes équivalentes (une porte équivalente équivaut à une porte NAND à 2 entrées en CMOS, soit 4 transistors). Le délai est exprimé en nano secondes.

Nom	sans decomp Surface	Délai	2 - entrées Surface	Délai	n - entrées Surface	Délai
bw	140.0	23.0	140.0	23.0	146.5	21.3
duke2	295.0	32.5	307.0	29.1	305.0	28.9
f51m	111.5	21.8	112.0	21.3	112.0	21.0
misex1	45.0	15.2	46.5	15.2	46.5	15.0
rd53	36.0	17.4	42.0	17.4	43.0	13.8
rd73	81.5	22.4	85.0	18.5	87.5	19.3
sao2	152.0	28.4	157.5	25.5	159.0	24.7
big2	125.0	22.0	125.0	22.0	127.0	21.4
ctrl2	266.0	28.0	268.5	27.8	268.5	27.8
fifowrite	126.0	23.5	126.0	23.5	129.5	20.3
flammand	149.5	25.5	149.5	25.5	152.0	23.9
icdma	104.5	22.4	104.5	22.4	105.5	22.2
planet	645.0	35.6	705.0	31.7	705.5	31.7
uar	395.0	33.8	397.0	31.1	401.0	30.6
TOTAL	2672.0	351.5	2765.5	334.0	2788.5	321.9
RATIOS			+3.5 %	-5.2 %	+4.4 %	-9.2 %
$\delta S / \delta D$			0.67		0.48	

figure IIIB-16 : tableau des résultats

D'après ces résultats, il est clair que les méthodes de décompositions non équilibrées fournissent des optimisations temporelles significatives. De plus, la décomposition en n-entrées aboutit à des résultats deux fois meilleurs que ceux de la décomposition en 2-entrées (-9.2% par rapport à -5.2%), pour une augmentation de surface guère supérieure (+4.4% par rapport à +3.5%). Ceci est mis en évidence par la dernière partie du tableau donnant le rapport $\delta A/\delta t$. Remarquons que dans de nombreux cas, la décomposition en n-entrées a permis d'obtenir des améliorations temporelles alors que la décomposition 2-entrées ne donnait rien.

Conclusion Générale

La synthèse logique est une discipline en perpétuel évolution car elle suit de très près l'évolution technologique et l'état de l'art des moyens informatiques.
La première contribution de cette thèse illustre bien ce fait. En effet, la théorie des automates semblait avoir atteint, il y a une vingtaine d'années, un état stable où les principales questions y compris le problème du codage, avaient été intensément étudiées. Il se trouve que ce problème a dû être reconsidéré sous un aspect tout nouveau. Les solutions proposées précédemment ne pouvant faire face ni à la complexité des problèmes actuels, ni aux critères d'optimisation liés à la cible. Bien sur, des techniques de base comme l'immersion de graphe dans l'hypercube, largement étudiées auparavant ont permis de construire avec rigueur et efficacité de nouvelles méthodes.
On voit donc que les progrès en synthèse logique sont liés à une excellente connaissance du passé. Tout oubli de la théorie passée est une faille grave. Mais une adaptation simple et intelligente est nécessaire. De nouveaux problèmes étant posés, de nouvelles facettes de la synthèse logique sont identiques et viendront compléter l'acquis dans ce domaine. Construire sur le passe et innover pour faire face aux problèmes réels semble être la devise. L'aiguillon de cette recherche est actuellement un passage immédiat de la théorie à l'application, ce qui est peut être nouveau dans ce domaine.

Quant au deuxième apport sur l'optimisation d'une décomposition booléenne, on peut peut-être dire que l'appui sur une théorie existante est moins net. Les problèmes de décomposition booléenne déjà abordés dans le passé, étaient trop éloignés des besoins d'aujourd'hui. Des solutions nouvelles et efficaces doivent être proposées dans un délai très court, et sont encore plus liées à la cible technologique. Il semble que si cette thèse

propose un apport pratique indéniable, la théorie est moins mure sur ce problème et les recherches devront s'intensifier pour dégager des fondements théoriques qui resteront d'un apport durable à la synthèse logique.

Références bibliographiques

[AHO76] A. Aho et S.C. Johnon
Optimal code generation for expression trees
Journal of the ACM, 23(3), pp 488-501, 1976.

[AMA87] R. Amann
Optimal state chains and state codes in finite state machines
IEEE Trans. on CAD, Vol. 8, N°2, Février 1989.

[ARM62] D.B. Armstrong
A programmed algorithm for assigning internal codes to sequential machines.
IEEE Trans. on Elec. Comp., pp 466-472, Août 1962.

[BAR86] K.Bartlett, W.Cohen, A. De Gueus, G. Hachtel
Synthesis and optimization of multilevel logic under timing constraints
IEEE Trans. on CAD of ICs and systems, vol. CAD-5, Octobre 1986, pp 582-596.

[BRA82] R.K. Brayton and C.T. McMullen
The decomposition and factorization of Boolean expressions
ISCAS Proceeding, pp 49-54, Mai 1982

[BRA84] R.K. Brayton et al.
Logic minimization algorithms for VLSI synthesis
Kluwer Academic Publishers, Boston, 1984.

[BRA86] R.K. Brayton et al.
Multiple-Level Logic Optimization System
ICCAD, pp 356-359, Novembre 1986.

[BRA87] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang
Multi-level logic optimization and the rectangular covering problem
ICCAD 87, pp 66-69, Novembre 1987.

[BRO81] D.W. Brown
A state machine synthesizer - SMS
Proc. 18th DAC, pp 301-304, Nashville, Juin 1981.

[CRA85] Michel Crastes de Paulet
Spécification et simulation fonctionnelles de circuits complexes : le système Cadoc
Thèse de Doctorat de l'INPG, Novembre 1985.

[DAN57] G.B. Dantzig
Discrete-variable extremum problems
Operations research, vol. 5, pp 266-277, 1957.

[DAT88] Dataquest Research Newsletter
Logic Design Automation : The Next Generation
Juin 1988.

[DEM83] G. de Micheli
Computer-Aided synthesis of PLA-based finite state machines
ICCAD 83, pp 154-156.

[DEM85] G. de Micheli, A. Sangiovanni-Vincentelli, R.K. Brayton
Optimal state assignment of finite state machines
IEEE Trans. on CAD, Juillet 1985, pp 269-285.

[DEM86] G. de Micheli
Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros
IEEE Trans. on CAD, Vol 5, N°4, Octobre 1986.

[DET87] E.Detjiens, G.Gannot, R. Rudell, A.Sangiovanni-Vincentelli, A.Wang
Technology mapping in MIS
ICCAD, Santa Clara, Novembre 1987, pp 116-119.

[DEV87] S. Devadas et al
Mustang, State assignment of finite state machines for optimal minti-level logic implementations
ICCAD 87, pp 16-19.

[GAJ83] D.D. Gajski, R.H. Kuhn
New VLSI tools
IEEE Trans. on Comp., Décembre 1983.

[GAR79] M.R. Garey et D.S Johnson
Computers and intractability : a guide to the theory of NP-completeness
W.H. Freeman, San Francisco, 1979.

[GRE86] D. Gregory, K. Bartlett, A. de Gueus, G. Hactel
SOCRATES : A system automatically synthesizing and optimizing combinational logic
23ieme DAC, 1986, pp 79-85.

[HAC88] G.D.Hachtel, R.M.Jacoby, C.R.Morrison
Techmap : technology mapping with delay and area optimization
International Workshop on Logic and Architecture Synthesis for Silicon Compilers, Grenoble, Mai 1988.

[HAR66] J. Hartmanis, R.E. Stearns
Algebraic structure, Theory of sequential machines
Prentice Hall, 1966.

[HOF87] M. Hofmann, J.K. Kim
Delay optimization of combinational static CMOS logic
Design Automation Conference, pp 125-132, Juin 1987.

[HOK87] M. Hoffman, J.K. Kim
Delay Optimization of Combinational Static CMOS Logic
Design Automation Conference, pp 125-132, Juin 1987.

[HON74] S.J. Hong, R.G. Cain et D.L. Ostapko
On complementation of Booleans fonctions
IEEE Trans. on Comp., Vol. C21, pp 1072, 1972

[HUF54] D.A. Huffman
The synthesis of sequential switching circuits
Journal of the Franklin Institute, Vol. 257, N° 3, pp 161-190
et N°4 (Avril 1954), pp 275-303.

[JOH79] D. Johannsen
Bristles Blocks - A Silicon Compiler
Design Automation Conference, Juin 1979.

[JOY86] W.H. Joyner et al.
Technology adaptation in logic synthesis
Design Automation Conference, pp 94-100, Juin 1986.

[KAR53] M. Karnaugh
The map method for synthesis of combinational logic circuits"
AIEE Transactions, Part 1, Novembre 1953, pp 593-598.

[KEU87] K. Keutzer
DAGON : Technology binding and local optimization by dag matching
Design Automation Conference, pp 341-347, Juin 1987.

[KEU88] K. Keutzer
Timing optimization in a logic synthesis system
International Workshop on Logic and Architecture Synthesis for Silicon Compilers, Grenoble, Mai 1988.

[KIN88] C. Kingsley
Timing Estimation for Netlist Performance Improvement
International Worshop on Architecture and Logic Synthesis
Grenoble, Mai 1988.

[KIN89] C. Kingsley, F. Poirot, V. Grimblatt
Implementation of VLSI Technology Netlist Optimizer
International Conference on CAD and CG, Pekin, Août 1989.

[KUN68] J. Kuntzmann
Algèbre de Boole
Edition Dunod, Paris, 1968.

[LAW76] E.L. Lawler
Combinatorial optimization : Networks and matroids
Holt, Rinehart and Winston, New-York, 1976.

[LEG88] M.C. Lega
Mapping properties of multi-level logic synthesis operations
ICCD'88, pp 257-261, Octobre 1988.

[LIN89] B. Lin, A.R. Newton
Synthesis of multiple-level logic from symbolic high level description languages
VLSI'89, Munich, Août 1989, pp 187-196.

[LIU63] C.N. Liu
A state variable assignment method for asynchronous sequential switching circuits
ACM, Avril 1963.

[MAI86] Mainsail Language Manual
Xidak Inc., Menlo Park, Californie, Février 1986.

[MCC65] E.J. McCluskey
Introduction to the theory of switching circuit
McGraw-Hill, 1965.

[MCN89] Microelectronics Center of North Carolina
Standard synthesis benchmarks
International Worshop on Logic Synthesis, NC, Mai 1989.

[MEA55] G.H. Mealy
A method for synthesizing sequential circuits
Bell Systems Technical Journal, vol. 34, N° 5, pp 1045-1079
Septembre 1955.

[MOO56] E.F. Moore
Gedanken-Experiments on sequential machines
Automata Studies, Princeton, N.J. Princeton University Press, 1956.

[PAU89] P. Paulin, F. Poirot
Logic decomposition algorithms for the timing optimization of multi-level logic
International Conference on Computer Design, Octobre 1990.

[POI89] F. Poirot, C. Duff, G. Saucier
State assignment using a new embedding method based on an intersecting cube theory
Design Automation Conference, Juin 1989.

[POI90] F. Poirot, C. Duff, G. Saucier
State assignment for controllers for optimal area implementation
European Design Automation Conference, Mars 1990.

[SAK90] K. Sakouti, G. Saucier
A Fast and Effective Technology Mapper on an Autodual library
IFIP Working Conf. on Logic and Architecture Synthesis, Paris 1990.

[SAS86] T. Sasao
MACDAS : Multi-level AND-OR Circuit Synthesis using Two-Variable Function Generators
Design Automation Conference, Juin 1986.

[SAU68] G. Saucier
Encoding of asynchronous sequential networks
IEEE Trans. on E.C., vol. 16, pp 365-369.

[SAU72] G. Saucier
State assignment of asynchronous sequential machines using graph techniques
IEEE Trans. on Comp., Mars 1972.

[SAU79] G. Saucier, D. Pilaud
Conception sure d'automatismes complexes à microprocesseurs
Revue générale d'électricité, Tome 88, n° 12, Décembre 1979.

[SAU90] G. Saucier, P. Abouzeid, F. Poirot
Multilevel synthesis for improvement of routability of circuits
Design Automation Conference, Juin 1990.

[SIC88] P. Sicard
Nouvelles méthodes de synthèse logique
Thèse de Doctorat de l'INPG, Grenoble, Septembre 1988.

[SIN88] K. Jit Sing, A.R. Wang, R.K. Brayton, A. Sangiovanni-Vincentelli
Timing optimization of combinational logic
ICCAD, Santa Clara, Novembre 1988, pp 282-285.

[TIS67] P. Tison
Generalization of consensus theory and application to the minimization of Boolean functions
IEEE Trans. on Elec. Comp., Vol. EC16, pp 446-456, 1967.

[TRI81] S. Trimberger
A structured design methodology and associate software tools
IEEE Trans. on Circuits and Systems, Juillet 1981.

[UNG63] S.H. Unger
A row assignment for delay-free realizations of flow tables without essential hazards
IEEE Trans. on E.C., Vol. 17, pp 146-151.

[VHD87] VHDL : Language Reference Manuel
IEEE Standard 1076-1987.

[VGa88] VGT10 Portable 2 Micron CMOS Gate Array Library
VLSI Technology, 1988.

[VGb88] VGT100 Portable 1.5 Micron CMOS Gate Array Library
VLSI Technology, 1988.

[VSa88] VSC10 Portable 2 Micron CMOS Standard Cell Library
VLSI Technology, 1988.

[VSb88] VSC100 Portable 1.5 Micron CMOS Standard Cell Library
VLSI Technology, 1988.

[VSc90] VSC320 Portable 1 Micron CMOS Standard Cell Library
VLSI Technology, 1990.

[WAL83] P. Wallich
Technology '83 : Automation (Design / Manufactoring)
IEEE Spectrum n° 20, 1983.

[WER82 J. Werner
A Silicon Compiler : Panorama, Wishful, Thinking, or Old Fat
VLSI Design, 1982.

Table des Matières

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 Roland GERBER
- Monsieur Christian LANDRAULT

Monsieur POIROT Franck

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 19 Juin 1990

Pour le Président de l'I.N.P.-G.
et par délégation,
le Vice-Président
P. VENNEREAU